Accepted Manuscript

Trust enforcement through self-adapting cloud workflow orchestration

Hadeel T. El-Kassabi, M. Adel Serhani, Rachida Dssouli, Alramzana N. Navaz

PII: S0167-739X(18)31352-9
DOI: https://doi.org/10.1016/j.future.2019.03.004
Reference: FUTURE 4828

To appear in: *Future Generation Computer Systems*

Received date : 1 June 2018
Revised date : 30 December 2018
Accepted date : 3 March 2019

Please cite this article as: H.T. El-Kassabi, M.A. Serhani, R. Dssouli et al., Trust enforcement through self-adapting cloud workflow orchestration, *Future Generation Computer Systems* (2019), https://doi.org/10.1016/j.future.2019.03.004

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

ACCEPTED MANUSCRIPT

Trust Enforcement Through Self-adapting Cloud Workflow Orchestration

Hadeel T. El-Kassabi[a], M. Adel Serhani[b,*], Rachida Dssouli[a], Alramzana N. Navaz[b]

[a] *Concordia Institute for Information Systems Engineering, Concordia University, Montreal, QC, H4B 1R6, Canada*
[b] *College of Information Technology, UAE University, P.O. Box 15551, Al Ain, UAE*

Abstract

Providing runtime intelligence of a workflow in a highly dynamic cloud execution environment is a challenging task due the continuously changing cloud resources. Guaranteeing a certain level of workflow Quality of Service (QoS) during the execution will require continuous monitoring to detect any performance violation due to resource shortage or even cloud service interruption. Most of orchestration schemes are either configuration, or deployment dependent and they do not cope with dynamically changing environment resources. In this paper, we propose a workflow orchestration, monitoring, and adaptation model that relies on trust evaluation to detect QoS performance degradation and perform an automatic reconfiguration to guarantee QoS of the workflow. The monitoring and adaptation schemes are able to detect and repair different types of real time errors and trigger different adaptation actions including workflow reconfiguration, migration, and resource scaling. We formalize the cloud resource orchestration using state machine that efficiently captures different dynamic properties of the cloud execution environment. In addition, we use validation model checker to validate our model in terms of reachability, liveness, and safety properties. Extensive experimentation is performed using a health monitoring workflow we have developed to handle dataset from Intelligent Monitoring in Intensive Care

*Corresponding author
Email address: `serhanim@uaeu.ac.ae` (M. Adel Serhani)

ACCEPTED MANUSCRIPT

III (MIMICIII) and deployed over Docker swarm cluster. A set of scenarios were carefully chosen to evaluate workflow monitoring and the different adaptation schemes we have implemented. The results prove that our automated workflow orchestration model is self-adapting, self-configuring, react efficiently to changes and adapt accordingly while supporting high level of Workflow QoS.

Keywords: `Cloud, QoS`, Reconfiguration, Self-Adapt System, State machine, Trust assessment, Workflow

---

1. Introduction

Workflow has been proven to be an appropriate model that finds its application in many domains, which features a set of tasks aggregated and executed either in sequence or in parallel to fulfill a particular goal. Workflows executed on a composed cloud services are distinguished by their ability to scale up or down according to the fluctuating nature of job or task requirements. This is achieved through orchestration functionalities, which can result in adding more storage space, auxiliary memory, additional servers, or reinstating corresponding relevant virtual Machines (VMs) in accordance to the sequence events that might take place, such as usage increase, or task failures. These orchestration functionalities allow realtime automated reconfigurations of the appropriate resources. Nevertheless, guaranteeing the Quality of Service (QoS) of the workflow to meet to user requirement level cannot be archived though orchestration only, but also automated monitoring and control of multi-cloud services is necessary.

According to [1], few research initiatives were proposed in the area of designing automated execution and monitoring complex workflow systems. Enabling easy-to-use systems that allow specification of QoS requirements levels and flexible deployments and resource allocation is highly required. This includes building models that describe algorithms and structures to empower these systems. Using state machine-based models to formulate the resource orchestration and autoreconfiguration is recognized for its capability to represent the continuous and dynamic nature of cloud resources. Maintaining the timely state of each

ACCEPTED MANUSCRIPT

entity, such as resources, quality requirements, and tasks performance, allow for easy tracking, efficient monitoring, and automated reconfiguration of the cloud resources and workflow deployment. Existing resource orchestration systems focus on resource configuration, deployment or control. However, they do not provide full automation to support self-configuration and self-healing where failures and performance deficiencies are detected and resolved automatically to maintain the required QoS [1].

Providing runtime intelligence in a sophisticated orchestration system involves high processing capabilities and adding more overhead on the cloud resources to provide analysis of large amounts of real-time monitoring data. Also, some workflows are deployed on multiple clusters and cloud providers. Federated cloud resource orchestration involves connecting multiple interacting cloud services to perform a composed service. Existing orchestration techniques depend on procedural programming using low-level scripting languages and heterogeneous Application Programming Interfaces (APIs), which are highly provider-configuration dependent [2]. This imposes more time and effort burden on the consumer. Hence, various research initiatives have proposed common interfaces and APIs over multiple clouds, such as Apache Deltacloud [3], Apache Libcloud [4], jclouds [5], OpenStack [6]. However, dynamic orchestration using high-level policies specified by administrators instead of consumers is highly compulsory. The currently used service composition techniques, such as the Web Service Business Process Execution Language (BPEL) and Business Process Modeling Notation (BPMN), do not support application resource requirements and constraints, exception handling, and optimized resource scheduling, which are essential for a comprehensive orchestration process [2]. Hence, trust enforcement is highly recommended to support the intelligent orchestration framework that handles the quality requirement of Big Data.

While cloud resource requirements need to be enforced within a dynamic orchestration, a trust evaluation must also be supported. A trust model should consider all the workflow phases and evaluate trust for each composed service, and then aggregate the overall workflow trust scores across multiple cloud

ACCEPTED MANUSCRIPT

providers. The model must carefully deal with all trust components, such as trust propagation, trust aggregation, decomposition, and trust sharing in federated cloud services. The trust score evaluation consists of capturing and monitoring the workflow runtime environment data to provide and maintain required orchestration of QoS levels. Yet, the complexity of orchestrating cloud services for Big Data is emphasized by the growing number of cloud services in terms of quantity, quality, and diversity. Few research initiatives fulfill user requirements in a realtime and context-aware manner, especially with the overwhelming amount of data coming from various sources of high veracity and variety.

Therefore, trust evaluation schemes and models should cope with the nature of intelligent workflow orchestration and composition of cloud services, especially when dealing with scalable and adaptive composition solutions that handle large-scale, highly dynamic, and diverse Big Data services. Supporting trust enforcement on orchestration frameworks creates an additional challenge to assess the contribution of the component services towards the composite services. This is because each service component might have different functionalities, significance, and impact within different compositions. Additionally, any proposed model must consider lightweight monitoring mechanisms with minimal overhead to not affect the overall service performance.

In this paper, we propose a workflow orchestration, monitoring, and adaptation model that relies on trust evaluation to detect QoS performance degradation and perform an automatic reconfiguration to guarantee QoS properties of the workflow. The monitoring and adaptation schemes are able to detect and repair different types of real time errors and trigger different adaptation actions including workflow reconfiguration, tasks migration, and resource scaling. We formalize the cloud resource orchestration using state machine that efficiently captures different dynamic properties of the cloud execution environment and support the monitoring activities of a workflow. We add two crucial components into the basic orchestrator framework: QoS Trust Monitoring and Autoreconfiguration Manager.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

The main differentiation of our framework with respect to other existing frameworks is summarized hereafter:

- We adopt a multidimensional trust evaluation that combines workflow performance-based trust evaluation and cloud resources performance-based trust evaluation. This will lead to the selection of the most appropriate adaptation actions.
- The evaluation of our monitoring and adaptation schemes overhead demonstrated that a minimum overhead both in terms of latency and communication is generated and considered low compared to other frameworks in the literature.
- Automating monitoring and adaptation processes in our framework saves time, shortens the process, and allows efficient control of resources as it continuously retrieves the most updated resource information.

2. Related Work

In this section, we discuss the existing state of the art on service composition and workflow orchestration including: 1) Trust in cloud service composition, 2) QoS and Trust monitoring, and self-healing, 3) dynamic and autonomic workflow orchestration.

2.1. Trust in Cloud Service Composition and Orchestration

Trust evaluation of a single service can be achieved through the propagation of reputation evaluation conducted by users based on historical experience. However, trust evaluation for service composition becomes more sophisticated because of the complexity of evaluating the trust of each component service separately. Despite this complexity, trust evaluation supports intelligence, scalability, and adaptive composition solutions for large-scale, highly dynamic, and orchestration frameworks to guarantee the quality of service requirement. Authors in [7], proposed a contribution-based distribution of reputation approach

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

to propagate the reputation of a composed service to each component service according to the extent to which it contributes to the composed service. The importance or the amount of contribution of each component service towards the composed service is assigned based on its reputation.

Recently, the authors in [8] proposed a trust framework that includes an iterative adjustment heuristic (IAH) model to assess trust in composed services. Service Trust evaluation in federated and interconnected cloud environments is more sophisticated [9]. Customers and different cloud providers need to trust each other to be able to collaborate. Thus, it is essential to evaluate the trustworthiness of cloud and cloud federations [10].

Trust in federated clouds was also addressed in the Sky Computing project [11], which is intended to enable several virtualized sites to increase resource availability. The project studied the trust, VM portability, and connectivity of geographically-spread resources. Bernstein et al. in [12] proposed a blueprint for interconnection of cloud data centers where they addressed issues about virtual machine mobility, storage, network addressing, security in terms of identity and trust, and messaging. However, no trust management was provided in this work.

Few existing cloud federation projects are based on brokering technologies for multi-cloud composed services. Hence, more research needs to be done towards a standardized methodology for handling interoperability and standard interfaces of interconnected clouds [13]. Trustworthiness evaluation models among different cloud providers were proposed and focus on a fully distributed reputation-based trust framework for federated cloud computing entities in cloud federation. In this model, trust values are distributed at each cloud allowing them to make service selection independently [10].

Usually orchestration methodologies provision describing resources of one provider. Other orchestration techniques support cross-provider resources such as Compute Service in JCloud and are used for configuration and management of federated cloud [14].

Trust models are developed to support monitoring, adaptation, and prediction of cloud workflow provision while guaranteeing the required workflow QoS.

ACCEPTED MANUSCRIPT

However, some of the initiatives proposed in the literature which used trust to enhance workflow scheduling, orchestration, and management were not fully utilized to support automatic reconfiguration that guarantees workflow QoS. Our proposed framework supports multidimensional trust evaluation that considers both the performance evaluation of the workflow and the performance evaluation of cloud resources in order to decide about the most appropriate adaptation actions.

2.2. Monitoring Trust in Service Composition and Workflow Orchestration

Monitoring is defined as gathering and analyzing events and performance logs and is necessary for supporting the management of unpredicted and undesired behaviors [1]. It is typically adopted to guarantee the required QoS by the SLAs and maintain stable performance by responding to quality degradation. Existing cloud resource monitoring tools, such as Nagios, CloudFielder, and Splunk are used by DevOps to describe SLAs, recognize glitches, and issue alarms when violations occur [15] [16]. Other Big Data monitoring frameworks like Ganglia [17], Apache Chukwa [18], Sematext [19], and SequenceIQ [20] provision QoS metrics information, such as resource utilization (cluster, CPU, and memory) in addition to application types (disk, network, and CPU-bound) [21]. Alhamazani et al. proposed a multi-cloud application QoS monitoring framework capable of monitoring subapplication distributed components, such as databases and web servers [22]. Other cloud QoS monitoring frameworks were presented in [23] [24] [25].

Most of the monitoring frameworks do not support the Big Data workflow specific QoS requirements, such as time sensitivity or task dependency. They usually monitor the workflow as a black box without involving the details of activities as in Amazon CloudWatch used by Amazon Elastic Map Reduce [26]. Such requirements involve data flow behavior and subactivity process monitoring. Activities in these workflows implicate continuous variations that affect other dependent activities and eventually affect the performance of the overall

ACCEPTED MANUSCRIPT

workflow. Present orchestration frameworks do not comprehensively support intelligent monitoring and automatic reconfiguration to respond to QoS violations. Such violations could occur in the context of a variety of inputs and performance quality characteristics throughout all the activities involved in the Big Data workflows. Additionally, intelligent monitoring should identify and handle the performance violations based on data flow collected logs. The authors in [26] designed a high level orchestration framework incorporating requirement and design specification Big Data workflows management over a cloud environment. However, this work is missing key implementation and validation of Big Data workflow orchestration functionalities and the challenges it involves.

2.3. Dynamic and Automatic Workflow Orchestration

Maintaining the QoS of such complex cloud workflows is very important to end users and applications. However, achieving this requirement necessitates guaranteeing the QoS during workflow execution, which cannot be archived throughout orchestration alone, but also through automated monitoring and control of multi-cloud services and resources. Automating such processes in a very dynamic environment will save time, shorten the processes, allow efficient control of resources and get most updated resource information, analyse monitoring and adaptation records to predict future resource shortage. In the following, we identify and discuss some of the relevant research work in guaranteeing QoS of cloud workflow through automatic orchestration.

Guaranteeing the user required QoS of application execution is the key purpose of cloud resource orchestration. Existing platforms that support Big Data orchestration, such as YARN [27], Mesos [28], and Amazon EMR [29], do not handle failure recovery or automatic scaling to correspond to the application changing requirements, such as the data flow changing volume, velocity or variety [26]. Some initiatives proposed automatic scaling of Big Data processing framework as in [30] for batch processing and in [31] for stream processing. Other orchestration frameworks provide online or interactive dynamic recon-

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

figuration [32] [33]. Web services frequently undergo dynamic changes in the environment such as overloaded resources. Hence, the authors in [34] proposed a multi-dimensional model, named AgFlow, for component services selection according to QoS requirements of price, availability, reliability, and reputation. The model optimizes the composite service QoS required by the user and revises the execution plan to adapt to the changes in the resource performance. Another work was proposed in [35], were an SLA renegotiation mechanism is developed to support and maintain QoS requirements in cloud based system. The SLA violations are predicted based on collected monitoring information of service status such as availability, performance and scalability.

We mean by self-healing as the capability of a workflow to recover its functionality when a problem occurs during execution while guaranteeing the QoS level requirements. Recent research approaches endorse automatic self-optimization workflow orchestration realized by dynamic resource reconfiguration to fulfill Quality of Service (QoS) requirements [1]. An example of an autonomic cloud orchestration engine is CometCloud [36], which supports the integration of local and public cloud services and the distribution and scheduling of these services according to resource status and QoS requirements, including budget, deadline, and workload. Authors in [37] proposed a self-healing Web Service Composition algorithm using a QoS performance-aware prediction technique. Moreover, Schulte et al in [38] proposed fuzzy BPM-aware technique that scales according to VM Key Performance Indicators (KPIs).

Current resource allocation techniques and existing frameworks do not support the dynamic and heterogeneous nature of clouds and resource behaviors. Therefore, the need to provide autonomic cloud computing methodologies that allow better resource allocation based on user QoS requirements as well as failure recovery during runtime is becoming inevitable. Researchers use various key QoS parameters for QoS-aware clouds, such as price, time, and response time. Most optimization techniques rely on the evaluation of time and price while other important QoS attributes (e.g., data privacy) are not considered. Authors in [39] pointed out some QoS parameters used in autonomic cloud com-

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

puting, including scalability, availability, reliability, security, cost, time, energy, SLA violation, and resource utilization. Other research approaches focus on user requirements, such as unit cost per resource, the processing speed of VMs, SLA levels, geolocations, and device capabilities of endusers.

A middleware architecture was proposed by Ferretti et al. in [40] to dynamically reconfigure cloud resources and services according to some QoS requirements specified in the SLA. Monitoring is used to support dynamic management, load balancing, and reconfiguration of resources allocation features. Moreover, a quality aware framework named Q-Cloud is suggested in [41] were resource allocation is performed at runtime. The key requirement is to guarantee QoS among multiple workload applications. The framework used QoS states were to support different levels of application-specific QoS assignments. The authors in [42] proposed adding extra modules to enhance the auto-healing capability of a common cloud service orchestrator. However, they did not provide system state description nor detailed their auto healing algorithms which are both very important features of the proposed solution.

3. Trust Formalization and Evaluation

Using Trust-based quality assessment enables aggregation of multiple and various quality dimensions and attributes into one trust score which facilitates efficient and comprehensive quality assessment. Guaranteeing trust is achieved through enforced monitoring of workflow at different granularity levels including for instance task level, service level and cloud resources level to achieve the targeted QoS.

3.1. Trust Evaluation of Cloud Workflow (Pre-deployment)

In this section, we explain the automatic evaluation of trust through a workflow that will be executed over a composition of cloud services. The selection of cloud services is based on the trust scores automatically evaluated before

ACCEPTED MANUSCRIPT

execution and during execution if reallocation of cloud services or resources is needed. Trust should be based on a set of evaluation criteria with weights assigned to each of these criteria and decided by the user. The first criterion is the reputation of service components, which generally relies on the users experience [7] [43]. This is called objective reputation and is done using monitoring, either by users or third parties [44]. Another form of trust based reputation relies on the opinion of users about the service which is known as subjective reputation. Both objective and subjective reputation can be combined to evaluate the trust and is referred to as hybrid reputation scheme. Trust evaluation based on advertised QoS of service providers and selfexperience can also be used. Each component service participates to the calculation of the overall trust of the composite service based on their contribution towards the composite service. Each QoS attribute participates towards the overall trust evaluation with weights assigned by the user, this is commonly known as user preference based trust. The contribution of each component service should be automatically assigned and calculated. Next section, will detail how QoS attributes are used for workflow trust evaluation.

3.1.1. QoS attributes for workflow Trust evaluation

Various QoS properties have been used in the literature to evaluate the trust. Among these attributes include for instance performance, including network and Cloud services [45], privacy, scalability, and extensibility. Other key metrics suggested in [26] involve the following: 1) delay of event discovery and decision making, 2) throughput, response time and latency of results generation in workflow, 3) distributed file read and write latency, 4) cloud resource utilization and energyefficiency, and 5) quality of network such as stability, routing delays, and bandwidth. In this context, the monitoring system is required to be comprehensive to have a full picture of the problem. In other words, monitoring application parameters measures the highlevel health of the system and will help in detecting the most serious issues. Whereas, monitoring the resource

ACCEPTED MANUSCRIPT

parameters allows finding and resolving the root cause of these issues. These quality parameters are monitored through a collection of cloud resources, such as CPU, memory, file system, and network usage statistics including utilization, saturation, availability, and errors. Also, monitoring is applied to some application-specific quality parameters like throughput, success rate (number of errors), and performance. Existing tools used for monitoring cloud resources like processing, storage, and network include cAdvisor, Heapster, InfluxDB, Google Cloud Monitoring, and many others [46].

3.1.2. Reputation of service components based on their past experience

In our previous work, we evaluated the reputation of a single service, and reputation of composed services can be achieved using multi-attribute optimization techniques to measure and assess the reputation of every single service based on its contribution towards the overall trust of the composed service [47]. The contribution ratio is determined by the user.

3.2. Trust Monitoring for Cloud Workflow Orchestration (Post-deployment)

After deployment, monitoring QoS of the workflow and all the allocated cloud resources will guarantee the satisfaction of customer requirements. Monitoring the CPU utilization, for example, will indicate that the application is performing as expected or experience delays when CPU is overloaded or might crash.

However, the complexity of monitoring Big Data workflows is characterized by the number of different QoS metrics that evaluate different activities and resources of the workflow. Such QoS metrics could be throughput, delay, event detection, response time, read/write latency, CPU utilization, energy efficiency, network delays, and bandwidth. Hence, it is rather challenging to combine all these different metrics into a holistic view across the workflow of different activities, the Big Data framework, and the utilized cloud resources.

ACCEPTED MANUSCRIPT

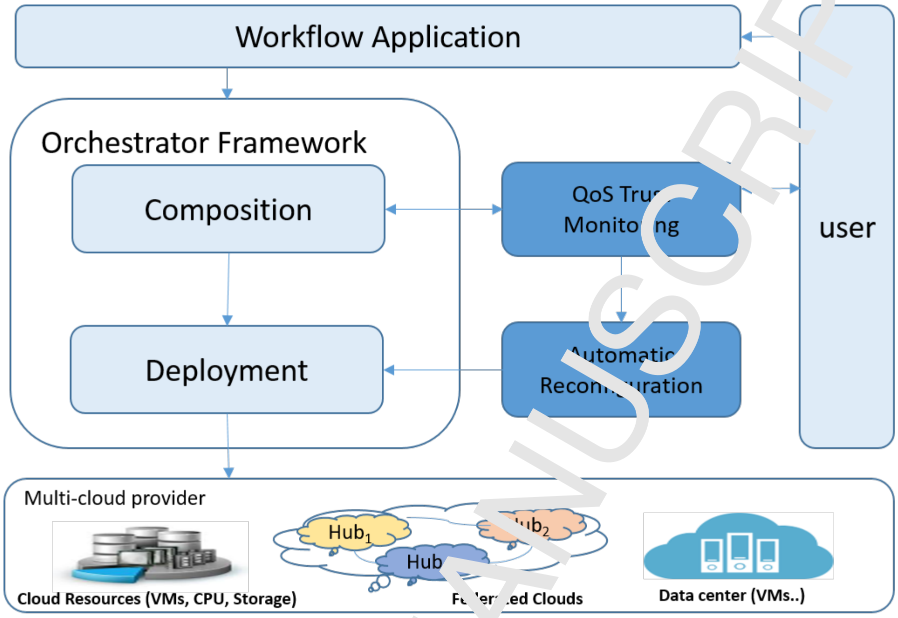

Figure 1: Workflow orchestration framework.

4. Model Architecture

In this section, we describe the architecture we propose to monitor trust and QoS of the workflow orchestration to guarantee self-reconfiguring workflow upon the occurrence of abnormalities. Figure 1 depicts the main architecture components.

4.1. Architecture Components and Key Features

4.1.1. Cloud Workflow Composition

At this stage the tasks composing the workflow are analyzed in terms of tasks specific nature, dependency to other tasks, required processing resource, and data usage. Big Data workflows are composed of various services some of which are dependent on another. In other words, changes in one service affect other dependent services. These services handle workloads with high volume and velocity data and have complex characteristics. Different application domains

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

exhibit different modeling requirements that involve specific domain expertise to specify, understand and manage the entire pipeline of activities, data flow inter-dependencies, and the QoS properties and their levels and ranges. Once the workflow is designed, it is mapped onto an existing orchestration framework for deployment.

4.1.2. Cloud Workflow Deployment

Service level agreement is build and signed by involved cloud providers prior to workflow deployment. Big Data workflow is mapped to orchestration frameworks that include Big Data programming APIs and cloud resources. The selection of suitable deployment configuration is challenging due to the complexity of the workflows and the abundance of selection possibilities. Choosing optimal workflow configuration is one of the open challenges that recently attracted researchers. For example, stream processing requires an optimal combination of various worker instances to minimize the latency of processing activities and to optimize the cloud resources configuration. Such resource configuration includes the location of the data center, node hardware configuration, pricing plan, network latency, and bandwidth availability [26].

4.1.3. Trust-based QoS Monitoring

Workflows monitoring is required to guarantee that the run-time QoS is satisfied and that the deployed cloud resources are optimized. Monitoring basically means collecting performance status logs of all resources and running workflows. The importance of monitoring lies in detecting and handling problems, in addition to empowering flexibility of deployment. For example, monitoring the CPU utilization and data transfer activity will help to determine if containers are overloaded, underloaded, or operating as required [1].

We describe hereafter the main module of our architecture. After deployment, the monitoring module is responsible for monitoring the QoS of the workflow. It is first configured to set the QoS attributes that are required by the user along with their thresholds and acceptable values or range of values. Also, the

ACCEPTED MANUSCRIPT

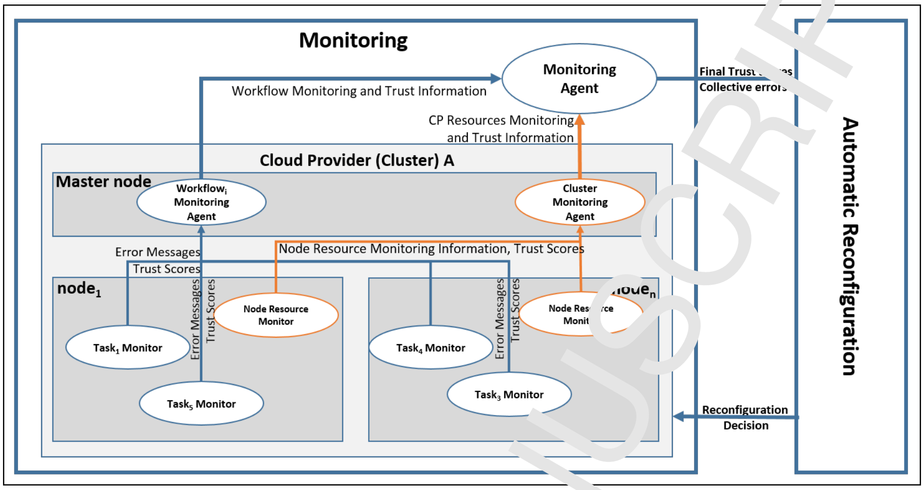

Figure 2: System architecture.

user will assign trust evaluation preference (weight) for each quality metric. Our monitoring system is responsible of monitoring each application including each composed service in the workflow application. Moreover, it is responsible for monitoring each data cluster of the service provider. The monitoring consists of three activities including monitoring the application, monitoring the cloud resources, and the QoS logs analysis. Measurements are taken periodically at different time intervals and the trust score is evaluated as a continuous function on the closed time interval [0, c], if we consider an arbitrary constant c > 0. This has been detailed in section 4.2. Our monitoring system architecture is detailed in Figure 2.

Monitoring the application: a monitoring agent is placed on the master node of each cluster. This agent will continuously check logs generated by the application tasks. The logs contain different measurements collected on executed tasks such as throughput, latency, and errors (I/O error) resulting for example from invalid input or delay due to slow response from other dependencies. However, each task has its specific properties and metrics that should be tracked. Table 1 depicts some key metrics for different application types. Each task in

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

the workflow is instrumented to generate the required measurement saved in the log files.

Table 1: Key metrics for many popular technologies

App Type	Metric Description	Metric Type
HTTP and proxy server.	Number of connections requested, successful and active	Utilization
	Number of requests	Throughput
	Calculated accepts – handled	Error
	Count 4xx and 5xx codes	Error
	Time to process each request (s)	Performance
Data storage Application	Number of read requests	Throughput
	Number of write requests	Throughput
	Number of current connections	Utilization
	Number of available new connections	Utilization
	Data, index, and total extents storage size	Utilization
	Virtual memory usage (MB)	Utilization
	Run time per schema	performance
	Numbers of statements with errors	Error
	Count of connections refused due to server error	Error
	Count of connections refused due to max_connections limit	Error
Processing application (search engine)	Utilization of RAM (JVM heap and the file system cache)	Utilization
	Total number of queries	Throughput
	Total time spent on queries	Performance
	Number of queries currently in progress	Throughput
	Number of queued threads in a thread pool	Saturation
	Number of rejected threads a thread pool	Error

Monitoring the cloud resources: this module is responsible for monitoring the cloud resources orchestration and management. The main metrics to be considered include resource utilization such as CPU usage, node CPU capacity, memory usage, node memory capacity, file system usage, and disk I/O. In addition, the monitoring observes the performance of the container such as container deployments, execution, and performance of required quality attributes.

QoS logs analyzer: part of the monitoring module that is composed of a set of processes distributed among each node. These processes collaborate to diagnose any problems, failures or abnormalities that occur in any application or happen in one of the clusters and evaluate a trust score for each node and

ACCEPTED MANUSCRIPT

task running on each node.

The design of process distribution works as follows: the node worker processes to monitor the node-specific quality metrics, the required metrics are passed through the main monitoring module along with their accepted values and ranges. The diagnose worker processes the watch of the streaming logs, checks the metrics values, and detects any out of range or failure values. The checked metrics values are interpreted, and a trust score and is generated for each task and each node. These trust values are sent to the master node periodically after a specified time interval. Moreover, upon problem detection, a worker process sends a notification message to the master node analyzer process. The later analyses the notification messages coming from all worker processes and identifies the cause of the problem then sends a general notification message to the main monitoring and analyzer agent which resides at the user's side. Sending only the trust scores and the notifications upon failures reduces the communication overhead so that the monitoring activities will not affect the performance of the applications and the host clusters. The main monitoring and analyzer agent is responsible for generating a trust score for each application and cluster and sending the compiled problem notifications to the automatic reconfiguration module.

4.1.4. Cloud Workflow Automatic Reconfiguration and Self-Adaptation

Automatic reconfiguration is the mechanism of taking necessary actions when the monitoring process reports performance degradations. These violations might be within the running workflows, the underlying frameworks or the resources to allow automatic self-reconfiguration and maintain the required level of QoS. For example, if the monitoring process detects a dramatic performance degradation, then the automatic reconfiguration module will trigger operations such as scale up or migrate to preserve the required QoS. Other problems could be produced due to errors or unexpected system behavior that might require restarting the container/VM which requires self-adaptation. The responsibility of the automatic reconfiguration module could be simple or sophisticated recon-

ACCEPTED MANUSCRIPT

figurations depending on the nature and the urgency of the occurred problem.

The complexity of dynamic and automatic reconfiguration of Big Data workflows arises because of its special characteristics are known by its multi-Vs. Hence, the first challenging issue is to model the QoS and estimate the data flow behavior with respect to volume, velocity, and variety and assessing the processing time and workflow I/O. Second, it is challenging to detect the cause of QoS abnormalities in heterogeneous frameworks as it can be originated, for instance, because of resource failure or congestion of network links. Another challenge is to model the runtime QoS changes of the workflow and construct orchestration so that the target QoS is upheld across the different layers of the orchestration framework.

Our automatic reconfiguration module detects the main cause of the problem upon receiving all the error occurrences in all applications and clusters from the primary monitoring module, then issues reconfiguration instructions to the corresponding application or cluster. For example, a delay in task completion and high processing load of the allocated node may trigger an action like moving a node with higher processing power or lower load depending on availability. Another example, when detecting a performance degradation with a storage task, we relocate the task to a node with higher storage capacity. In previous work we have developed a web based application [48] for collecting Big Data workflow QoS preferences from the user and generating a quality specification profile, which is used for task and workflow quality-based trust assessment. It also helps defining preferred threshold values and ranges to be used for quality degradation decision making. For example, a service degradation or failure could be detected when it takes longer than the expected execution time before completion or it generates an unexpected or invalid output. Moreover, we define a service failure rate FR as $FR = totalNumberOfFailures/t$, where t $>$ 0 is a constant time period. Afterwards, the reconfiguration instructions are sent back to the application or cluster to be reflected and deployed. The algorithms of each of the modules are detailed in the following section.

Automatic reconfiguration module: this module evaluates the status of

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

each workflow and generates reconfiguration decisions to improve the performance of each workflow. This module receives and keeps the trust score for each workflow, the trust score for each cloud provider, and the error messages or abnormality notifications. Accordingly, it compares the latest trust score with the previous trust score, and if high, then nothing will be done. However, if low, then reconfiguration decisions should be made. Also upon receiving error messages, reconfiguration decisions are made.

4.2. Automatic Cloud Workflow Trust Evaluation Model

Typically, tasks run independently or are tied together in an ad hoc manner. An orchestration environment, like Kubernetes, link these tasks together in a loosely coupled fashion [46]. The workflow model fits well for our problem requirements however, other models might also be explored. The following detail our monitoring model and Table 2 describes the symbols used.

Table 2: Symbols used.

$\boldsymbol{P}$	number of tasks in the workflow
$\boldsymbol{m}$	number of clusters allocated for a workflow
$\boldsymbol{r}$	number of nodes in a cluster
$\boldsymbol{s}$	number of containers allocated for a task
$\boldsymbol{j}$	number of QoS attributes requested by the user
$\boldsymbol{v}$	number of violation at time t

Let ***Monitor*** $(\boldsymbol{WF}, \boldsymbol{Q})$ denotes a Monitor request to the global monitor $\boldsymbol{GM}$ to initiate workflow monitoring based on a given list of QoS attributes. The Monitor request starts the collection of the deployed workflow QoS logs. The workflow is modeled as a directed acyclic graph $\boldsymbol{WF(T, E)}$ where $\boldsymbol{T}$ = $\{tk_1, tk_2, \ldots, tk_p\}$ denote tasks to be monitored along with the deployment configuration which may include one or more clusters. The number of tasks in the workflow is denoted by p. Each task contributes with a different

ACCEPTED MANUSCRIPT

weight to the overall workflow. We denote the level of importance of a task towards a workflow by il. This value is given by the data analyst who constructed the workflow composition as $\boldsymbol{IL} = \{il_1,\ il_2,\ \ldots,\ il_p\}$, where p is the number of tasks in the workflow. $\boldsymbol{E} = \{(tk_i,\ tk_j) \mid tk_i,\ tk_j \in \boldsymbol{T}\}$, is the set of arcs representing a partial constraint relationship between tasks so that, $\forall\ (tk_i,\ tk_j) \in \boldsymbol{WF}\ (i \neq j)$, and tk_j cannot start until tk_i completes. Let $\boldsymbol{Clusters} = \{cl_1,\ cl_2,\ \ldots,\ cl_m\}$, where m is the number of clusters allocated for a workflow.

A ***Container*** is represented as $C\langle\ cn,\ tk_i,\ n_j,\ cl_k\rangle$, where:

- cn is a container id number, $tk_i \in WF$, a node hosting cn, $n_j \in Nodes$, and $cl_k \in Clusters$ is the cluster that owns the node n_j.

- Each task tk is mapped to one or more node(s) in one or more cluster(s) and is represented as a tuple $tk\langle\ tn,\ \{c_1,\ c_2,\ \ldots,\ c_s\},\ st,\ in,\ out\rangle$, tn is the task name/id, and the second parameter is the list of destination containers allocated for that task. We assume that a task will run in one container per node. Multiple containers will be destined to multiple nodes. st is the state of the task (waiting, active, or completed) and in and out are the input and the output data set respectively.

- The ***node*** $n_k\ \langle\ specs,\ lm\rangle$ is a tuple which represents the specification of the node including *cpu*, *memory*, and a local monitor lm which is responsible for calculating the trust score of the task and detect QoS violations.

- A ***Cluster*** $cl_j \in Clusters$ is modeled as a list of nodes $cl_j = \{n_0,\ n_1,\ \ldots,\ n_r\}$, where n_0 is the master node and n_i is a worker node such that $i \in [1,\ r]$.

$\boldsymbol{Q} = \{q_1,\ q_2,\ \ldots,\ q_j\}$ where j is the number of QoS attributes requested by the user and the weights for each attribute are $\boldsymbol{W} = \{w_1,\ w_2,\ \ldots,\ w_j\}$. We also refer to a list of QoS violations as $\boldsymbol{VList}(\Delta t) = \{v_1,\ v_2,\ \ldots,\ v_n\}$, at a time range/window Δt. We model the violation by a tuple $\boldsymbol{V}\ \langle C,\ Vtype,\ value,\ t\rangle$,

ACCEPTED MANUSCRIPT

where here the violation occurred at time t, is associated to a container tuple, the type of violation, and the value of violation (the abnormal value).

The Local Trust Score $\boldsymbol{LTS}$ is a score representing the level of satisfaction of all requested QoS attributes in $\boldsymbol{Q}$ according to the respective weights $\boldsymbol{W}$. The $\boldsymbol{LTS}$ is specific to each task running on a specific node. If the task is replicated on multiple nodes, then the $\boldsymbol{LTS}$ is aggregated as the average of all $\boldsymbol{LTSs}$ for that task among all containers.

In our model we evaluate the quality of a workflow based on multiple criteria or quality attributes and different preferences of each of these criteria. Multi-Attribute Decision-Making (MADM) [49] is considered a simple, clear, systematic, and logical technique, to help decision making by considering a number of selection attributes and their priorities. They can help to choose the best alternative with the set of selection attributes. They are also considered the most common method used in real, decision-guiding multi-attribute utility measurements.

$\boldsymbol{LTS^t_{ijk}}\langle\ tk_i,\ n_j,\ cl_k,\ qp,\ Q,\ W\rangle$, is calculated using a MADM algorithm while $\boldsymbol{Q}$ and $\boldsymbol{W}$ are the required quality performance values collected from worker node $\boldsymbol{n_j}$ in cluster $\boldsymbol{cl_k}$ for task $\boldsymbol{tk_i}$ at time $\boldsymbol{t}$ (where $\boldsymbol{t} > 0$), their weight, and its contribution towards the trust score respectively. The $\boldsymbol{qp_i'}$ are the normalized task performance according to the QoS required value $\boldsymbol{qp_{target}}$. This guarantees that the trust score will be evaluated based on its proximity of the value to the required QoS value specified by the user and SLA which we describe as the target value (i.e., objective value). Alternatively, the target value could be the arithmetic mean of the maximum and minimum values in an accepted quality range $\boldsymbol{qp_{targe}} = (qp_{min} + qp_{max})/2$.

$$\boldsymbol{qp_i'} = \begin{cases} qp_i/qp_{target}, & qp_{target} > qp_i \\ qp_{taregt}/qp_i, & qp_{target} < qp_i \end{cases} \tag{1}$$

The calculation is performed by a local monitor $\boldsymbol{LM_j}$ residing in each node as a continuous function on the closed time interval [0, c]. If we consider an arbitrary constant $\boldsymbol{c} > 0$, then the average local trust score $\boldsymbol{LTS^t_{ijk}}$ is represented by the

ACCEPTED MANUSCRIPT

following formula:

$$\boldsymbol{LTS_{ijk}} = \frac{1}{c} \int_0^c LTS_{ijk}^t \; dt \tag{2}$$

$\boldsymbol{ALTS_{ik}}$ is the aggregated $\boldsymbol{LTS}$ calculated at the master node $\boldsymbol{n_0}$ as the arithmetic mean of all trust scores collected from all worker nodes in cluster $\boldsymbol{cl_k}$ for a task $\boldsymbol{tk_i}$ at time $\boldsymbol{t}$ as $\boldsymbol{ALTS_{ik}}(t) = 1/r \sum_{i=1}^{r} LTS_{ijk}(t)$, where $\boldsymbol{r}$ is the number of worker nodes for one task $\boldsymbol{tk_i}$ deployed in $\boldsymbol{cl_k}$. The $\boldsymbol{ALTS_{ik}}$ is sent from the master node $\boldsymbol{n_0}$ in each cluster $\boldsymbol{cl_k}$ to the global monitor $\boldsymbol{GM}$. The following two scores $\boldsymbol{GTS_i}$ and $\boldsymbol{WFTS}$ are calculated at the $\boldsymbol{GM}$ as follows:

$\boldsymbol{GTS_i}$ the global trust score, is the average of all trust scores for task $\boldsymbol{tk_i}$ across all clusters at time $\boldsymbol{t}$. $\boldsymbol{GTS_i}(t) = \sum_{k=1}^{m} ALTS_{ik}/m$, where $\boldsymbol{m}$ is the number of clusters, and $\boldsymbol{t}$ is the time at which the trust scores were collected. The workflow trust score at time $\boldsymbol{t}$ is the weighted sum of all $\boldsymbol{GTS_i}$ for all composed tasks according to their importance level il_i towards the workflow $\boldsymbol{WF}$.

$\boldsymbol{WFTS(t)} = \sum_{i=1}^{p} GTS_i(t) \times \; il_i$, where $\boldsymbol{p}$ is the number tasks in a workflow.

A ***Report*** is a message that contains: 1) a workflow trust score, 2) list of trust scores of all composed tasks and 3) a list of QoS violations periodically sent from $\boldsymbol{GM}$ to the $\boldsymbol{ReconfigMgr}$. We model the Report as a tuple:

$\boldsymbol{Report} \left\langle WFTS\,(t)\,,\; \{GTS_1\,(t)\,,\; GTS_2\,(t)\; \ldots\; GTS_m\,(t)\}\,, \{v_1,\; v_2,\; \ldots,\; v_n\}\; \right\rangle.$

The ***Handle*** (*Report*) is the process called by the Global Monitor $\boldsymbol{GM}$ to the $\boldsymbol{ReconfigMgr}$ when a QoS violation is detected during runtime or periodically as explained earlier.

The $\boldsymbol{ReconfigMgr}$ processes the ***Report*** and reaches an automatic reconfiguration decision. The decision function $\boldsymbol{D}$ *At time* $= t$, is modeled as follows:

$$D\,(WFTS_t\,, VList_t) = \begin{cases} 1, & \text{if } V \; != null \\ -1, & \text{if } V = null \;\&\&\; WFTS_t < WFTS_{t-1} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

A ***Decision*** ($NewConfigList$ $\{\langle tk_i, c_j,\; configFile\rangle\}$) message is sent about each workflow to the concerned party to change the configuration.

The $\boldsymbol{NewConfigList}$ includes a list of suggested configurations for one or more tasks in the workflow. Each tuple in $\boldsymbol{NewConfigList}$ contains the task $\boldsymbol{tk_i}$, destination container $\boldsymbol{c_j}$, configuration file $\boldsymbol{configFile}$ which is a script containing the new configuration suggested by the $\boldsymbol{ReconfigMgr}$ usually specified in yaml format, which is a simple commonly-used language for application configurations that is compatible with many other languages and frameworks [50]. It is enhanced for data serialization, configuration settings, log files, and messaging, which fits our framework requirements. The destination of this message is the master node of each cluster hosting the container specified in the $\boldsymbol{NewConfigList}$.

4.3. Automatic Cloud Workflow Trust Evaluation Algorithms

In this section, we propose automatic workflow trust evaluation algorithms during the pre-deployment, post-deployment, and self-adaptation in case of QoS requirements violation. The system architecture of our model is shown in Figure 2 as previously detailed in section 4.1.3.

4.3.1. Pre-deployment Workflow Trust Evaluation

The services are composed of an optimal set based on trust scores according to QoS constraints. The trust scores of each service are generated based on historical QoS logs. Then, we compute the QoS aggregation value of each workflow path and select the best path that meets the QoS requirements. We use the MADM algorithm for trust evaluation of each task. Accordingly, the workflow tasks are mapped to a specific resource that responds to its QoS requirement. Mapping the services to the resources can be achieved using similarity matching as an initial deployment. For example, if the task needs storage, we match it to a resource with high capacity storage resource, and if it requires high processing, we match it to a high processing power server.

4.3.2. Post-deployment Trust Monitoring

Trust monitoring consists of measuring trust values that support the two modes of monitoring operations of periodic or continuous monitoring. The continuous

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

operation mode requires running the monitoring process as a daemon that logs the status of the monitored tasks and system. The trust scores are evaluated by our monitor module which is comprised of two submodules: the local monitor (at master node, or worker node) and global monitor. The following describes the key activities supported by both local and global monitor for the sake of monitoring:

<u>At the local monitor:</u>

1. Collect the performance values according to QoS required list for a task
2. Evaluate a trust score for a task
3. Produce the output of a trust score for a task at node $\boldsymbol{i}$

<u>At the local monitor in master node:</u>

1. Collect trust scores from all local monitors in other nodes for a task.
2. Calculate the average trust scores to get ATS for a task at cluster k.
3. Output is the ATS for a task at cluster k

<u>At the global monitor:</u>

1. Collect ATS aggregated trust scores from all clusters for a task
2. Calculate the average trust scores to get GTS for a task among all clusters and calculates the $WFTS$ for all tasks in a WF according to the task importance (weight) towards WF.

Algorithm 1 depicts this trust score calculation algorithm.

4.3.3. Automatic Reconfiguration of Workflow Orchestration

Algorithm 2 depicts the automatic workflow orchestration reconfiguration algorithm. This algorithm analyzes each task violation by checking the root cause of the violation. For example, it checks if a resource limitation is the cause of the violation such as an overloaded node, then a message is triggered to add a new node to the cluster. However, if the cluster cannot be extended, then a migration message is issued, and the task is allocated to a new cluster (see

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

Algorithm 1 Trust score calculation algorithm

Input:
$Tasks$ //List of Tasks,
$QoSList$ //List of QoS attributes,
$weights$ //Weights of each QoS attribute
Output: $LTSList$ //Local Trust Score updated for each Task
procedure EvaluateLocalTrustAtWorkerNode($Tasks$, $QoSList$, $weights$)
  for $t \leftarrow 1, c$ **do**
    $scoresList_t \leftarrow empty$
    for all $tk \in Tasks$ **do**
      $score \leftarrow 0$
      for all $q \in QoSList$ **do**
        $score \leftarrow score \ + \ measuredQVal_q \times weights_q$
      end for
      $scoresList_t[tk] \leftarrow score$
    end for
  end for
  for all $tk \in Tasks$ **do**
    $LTSList[tk] \leftarrow \frac{1}{c}\int_0^c scoresList_t[tk]\, dt$
  end for
  return $LTSList$
end procedure
Output: $ALTSList$ //Aggregated Trust Score (across nodes) for each Task
procedure EvaluateAgregatedLocalTrustAtMasterNode
  for all $nodes \in Cluster$ **do**
    $getLTSList_{node}$
    for all $tk \in LTSList_{node}$ **do**
      $ALTSList[tk] \leftarrow ALTSList[tk] + LTSList_{node}[tk]$
    end for
  end for
  for all $tk \in Tasks$ **do**
    $ALTSList[tk] \leftarrow ALTS[tk]/nNodes$
  end for
  return $ALTSList$
end procedure
Output: $GTSList$ //Global Trust Score (across clusters) for each Task
procedure EvaluateGlobalTrustAtGlobalMonitor
  for all $cluster \in Clusters$ **do**
    $ALTSList_{cluster}$
    for all $tk \in ALTSList_{cluster}$ **do**
      $GTSList[tk] \leftarrow GTSList[tk] + ALTSList_{cluster}[tk]$
    end for
  end for
  for all $tk \in Tasks$ **do**
    $ALTSList[tk] \leftarrow ALTS[tk]/nClusters$
  end for
  return $GTSList$
end procedure

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

Algorithm 2 Automatic reconfiguration of workflow orchestration algorithm

```
Input:
    taskViolations // QoS task violation List,
    sysViolations // QoS system violation List,
    GTSTable // GTS for each task in WF
Output: NewConfig
procedure AutoReconfigAlgorithm(taskViolations, sysViolations, GTSTable )
    for all tk ∈ taskViolations do
        sv ← findNode(sysViolations)
        if (sv ≠ ∅)
          svType ← violationType(sv)
          if(svType = "sysOverload")
            newConfig[tk] ← addNode(getCluster(sv))
          else if(svType = "sysOverloadNoExtend")
            newConfig[tk] ← migrate(tk)
          endif
        else //problem in task
          newConfig[tk] ← scaleUp(tk)
        endif
    end for
    for all tk ∈ GTSTable do
        avgT ← avg(historyTrust[tk])
        if(trust(tk) ≤ avgT)
          newConfig[tk] ← findNewDeployment(tk)
        else //problem in task
          newConfig[tk] ← ∅
          update(historyTrust[tk], trust(tk))
        endif
    end for
    return newConfig
end procedure
```


ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

Table 3). The algorithm also analyzes the new trust scores for all the tasks in the workflow, and if it detects trust score degradation, then it generates a new configuration decision.

We have implemented the two algorithms, Algorithm 1 for trust evaluation, and Algorithm 2 that is responsible for adaptation and reconfiguration actions upon QoS degradation based on trust evaluated by Algorithm 1.

Table 3: Workflow monitoring messages.

Message	Source	Destination	Parameters	Description
getLTSMsg$_t$	*Master node*	*Worker node*	*Q, W, list{taskid}*	The master node sends this message to all worker nodes in the cluster to collect the task trust values according to the required quality attributes and their weights passed in the message parameters.
replyLTSMsg$_t$	*Worker node*	*Master node*	*List {<taskid, LTS>}, List{sysViolations}*	This message contains a list of all task trust scores from each worker node to the master node as a response to getLTSMsg$_t$ message. This message also contains a list of system violations, such as CPU overload.
sendALTSMsg$_t$	*Master node*	*GM*	*List {<taskid, ALTS>}, List {<node, sysViolations>}*	This message contains the list of aggregated trust scores for each task running on this cluster. Also, it contains a list of system violations for each problematic node.
sendFTSMsg	*GM*	*AR*	*WF, list {<taskid, GTS>}, list{sysViolations}, list{taskViolations}*	This message is sent from the GM to AR for each WF and contains the list of tasks composed in the WF along with their GTS. Also, it contains the list of system violations and list of task violations.
autoReconfig	*AR*	*taskid, node, cluster*	*Reconfig File*	This message contains all reconfiguration commands issued by the AR and regarding each task ids in a certain node and certain cluster.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

5. Cloud Workflow Monitoring Model

5.1. Characterizing System Elements and State Description

In this section, we model the parameters characterizing the state of each system component, such as cloud resources including nodes, containers with their specifications, and workflow, its composed tasks, events, and messages. For workflow components, we base our trust evaluation on the composed tasks. Each task trust is evaluated based on multi-dimensional trust specification. We evaluate a workflow quality base on two dimensions of the quality; the data quality and the service quality (i.e. the quality of the process handling this data). We adopt some of the well-reputed data quality dimensions accepted in the literature Quality dimension for data that are Timeliness, Accuracy, Completeness, and Consistency. Moreover, we use some of the common processing quality dimensions discussed in the literature such as Capacity, Performance, Response time, Latency, Throughput, Accuracy, Availability, Robustness, and Scalability. Moreover, we need to model the workflow and its constraints so that the monitoring system actions take into consideration the workflow status including task choreography, dataflow, recovery, and task dependencies. For example, if we have two tasks, *T1* and *T2*. We call *T2* dependent on *T1* when *T2* is invoked after the *T1* response is received or completed.

We also consider the data flow where the task input and output states are tracked. For each task *T1*, we retain information about the parameters, the data type and format of parameters, and the time expiry and validity of parameters. Additionally, recovery actions should be triggered when an error or delay receiving a response occurs such as *T1* terminate, *T1* reconfiguration (assign to the different cluster), or Ignore error.

5.1.1. Tasks

As described above, a task is modeled as a tuple $tk\,\langle\, tn,\ \{c_1,\ c_2,\ \ldots,\ c_s\}\,,\ st,\ in,\ out\rangle$ and task dependency is modeled in $\boldsymbol{E} = \{(tk_i,\ tk_j) \,|\, tk_i,\ tk_j \in \boldsymbol{T}\}$. In this section, we detail the state, input, and output. Figure 3 shows the states of each

ACCEPTED MANUSCRIPT

Figure 3: Task state machine automata.

task and the related transitions. The state *st* of a task can be idle, running, and completed. Idle state is the state of the task before it starts running, a task is in running state when the previous task is completed, and the input is ready. However, a task is completed when the output is ready.

5.1.2. Events

The event is usually a violation that occurs in a node or to a specific task such as CPU overload, disk full, increasing task errors, and task overload. We construct an event as a message sent to the master node with the format:
sendNodeViolationMsg (source: (node, cloud), dest: master, <Type, value, category>, t).
Accordingly, the master node compiles a list of all received messages to be sent to the General Monitor with the format:
sendClusterViolationMsg (source: (cloud), dest: GM, list {<Type, value, category>}, t)

5.1.3. Monitoring Messages Specification

All the messages used in our workflow monitoring system and their details including source, destination, parameters and description are shown in Table 3.

5.2. Cloud Workflow Monitoring and Adaptation State Machine

We used state machines to formalize our monitoring system in order to validate our system to confirm that it does what is required and satisfy its objectives. In addition, representing the system with state machines enables formal verification to confirm if we are building the software right and that it conforms to

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

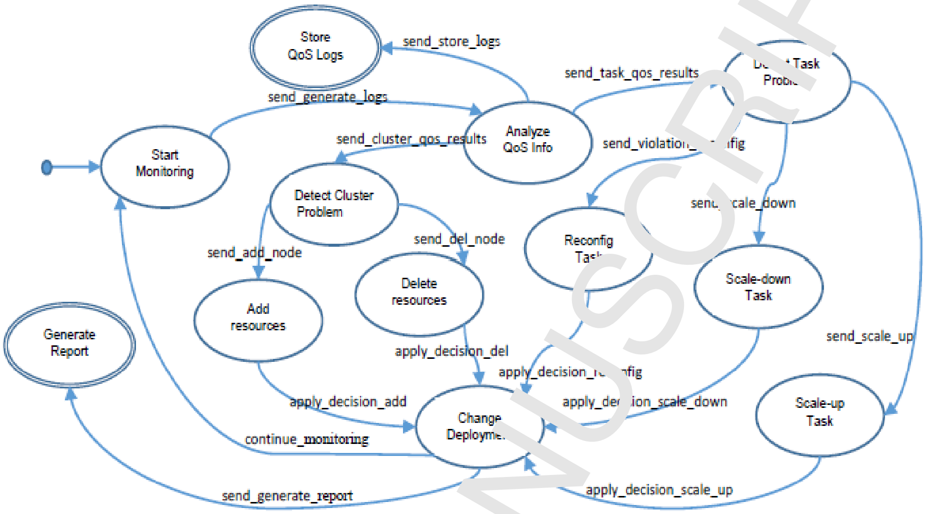

Figure 4: Workflow monitoring and adaptation state machine.

its specifications. We used model checker for formal verification of our monitoring system to prove the correctness of our software with respect to a certain formal specification or property, using formal methods. Figure 4 depicts the state machine automata of our monitoring and reconfiguration framework. The following sections describe in detail this system state machine.

5.2.1. Workflow Monitoring

As mentioned above, monitoring consists of collecting the logs and QoS information regarding all the entities of interest, such as tasks and resources. It is also responsible for updating the trust scores of each task using the collected logs analysis results. Upon violation detection, a violation message is sent to the reconfiguration manager. During monitoring, the states of each entity are updated and kept in the system for further use during the reconfiguration state.

ACCEPTED MANUSCRIPT

5.2.2. Workflow Reconfiguration

Upon a reconfiguration decision, the AR module decides what new configuration is suitable for the situation. The following is the description of the possible changes and the implication of each change regarding the state of the *WF*, task, and resources. The AR module first checks the state of the task, according to the task type (if the task allows scaling during the running state).The task status (e.g. completion time) is estimated based on the type of the task. For example, some tasks' status is estimated based on the percentage of output completion, other types of tasks' status are estimated based on calculation of execution time, which is based on the input size and allocated resources. If the task type is scalable, then scale up or down (by applying the change in the configuration file and deploy) and update the state of the task accordingly.

Scale up: run additional replications of the task on more nodes to handle the heavier traffic input, then update the state with the new number of replicas.

Scale down: when unused replicas are detected, then the replicas are deleted, then update the state with the new number of replicas.

Reconfigure: change the deployment configuration for the task by changing the node or cluster assignment according to considerations such as task type, task state, and task dependency. The task type can be scalable or non-scalable, and the task state can be waiting, running or complete, and the task dependency can be dependent on other tasks or other tasks dependent on this task.

Usually, the type of reconfiguration decision is taken following a QoS violation. For example, a migration decision is only taken depending on the severity level of the violation and the state of the task. If there is an issue within the cluster (e.g., CPU overloaded) and the processing performance is degrading over time, then the decision is to migrate the task to another cluster having the best QoS trust score recently measured. In order to satisfy the self-adaptation feature during reconfiguration, specifically the migration decision, the state of the task plays a significant role. In other words, migration should consider the task and its dependent tasks including all the dependent task list. for simplicity, we

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

do not need to migrate the predecessor tasks. Moreover, all the dependency input data should also migrate.

In case the cluster performance is degraded with a rate higher than a certain threshold, migrating the whole workflow is considered. If the task state is *'waiting,'* then the migration is straightforward, and the task along with its input dataset is migrated to the new destination (e.g., node). However, if the task state is *'completed,'* then migration is performed for the remaining dependent tasks in the workflow along with their input dataset. Nevertheless, when the task state is *'running,'*, many issues should be handled so the workflow required QoS is not affected. On the one hand, if the violation type is causing a service interruption, then we restart the task from the beginning at the new destination by resetting its state to *'waiting.'* On the other hand, deciding whether to move the task immediately or wait until it completes depends on the task completion status. The task completion status can be measured by calculating the percentage of generated output data against the expected output data. If the percentage of completion of a task is higher than a certain threshold, then we wait until the task is *'completed'* and migrate the remaining dependent tasks in the workflow. Otherwise, the task is considered at the beginning stage, and it is reset to *'waiting'* state, then migrated to the new destination.

5.3. Quality Metrics

The following in Table 4 are the common metrics and thresholds used to help in adaptation decision making and reconfiguration actions. Such threshold values are based on the application domain, workflow type, and user requirements. These values are reevaluated for every workflow according to its application domain and nature.

The priority of each of the above metrics varies according to the task QoS requirements. We define two classes of priority, *highPriority* and *lowPriority*. Furthermore, we define two violation alert types, severe and moderate as:

$severeViolationAlert(x) \leftarrow$
$(lowPriority(x) \ \wedge \ EX \ lowPriority(x)) \ \vee \ highPriority(x)$

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

$moderateViolationAlert(x) \leftarrow lowPriority(x) \ \wedge \ \neg \ highViolationAlert(x)$

The reconfiguration decision is issued when a violation alert is received and includes either a high or low violation:

$reconfig(x) \leftarrow highViolationAlert(x) \ \vee \ lowViolationAlert(x)$

Table 4: Quality violations.

Quality Violation	**Threshold**
abnormalCPUUtilization (x)	80%
abnormalHighMemUtilization (x)	80%
abnormalLowMemUtilization (x)	15%
abnormalNetworkAvailability (x)	10%
abnormalDiskAvail(x)	80%

5.4. Validation-based Model Checker

The following describes our monitoring system where an administrator configures and initiates the monitoring process after workflow deployment. Once the system initializes the monitoring process, the QoS logs are generated, and the following actions are sequentially triggered when task abnormality is detected: *Analyze QoS Info, Store QoS Logs, Detect Task problem, Reconfigure Task, Change Deployment, and Generate Report.* Figure 4 detailed in section 5.2, describes the finite state machine of the workflow monitoring and adaptation system where a unique name identifies each state and connected to other states through applicable transactions. The transactions are labeled with names corresponding to the actions.

According to the type of detected problem, the system takes an appropriate action to maintain the required workflow QoS level. In the case of detecting an issue with task execution, such as low task response time is encountered then a scale up state is initiated where more containers are allocated for that task.

ACCEPTED MANUSCRIPT

To formalize our monitoring system, we assume that our system is composed of a set $M = \{1, 2, \ldots, n\}$, of n services interacting together. Each service $i \in M$ is defined by:

1. A set of LS_i finite local states as shown in Figure 4 where *start_monitoring*, *analyze_QoS_info*, *store_QoS_info*, and *detect_task_problem* are some of the system local states.
2. A set of LA_i of finite local actions as shown in Figure 4, for instance, *send_generate_logs*, *send_task_qos_results*, and *send_generate_report* are some of the system local actions.
3. A local protocol $Pr_i : LS_i \rightarrow 2^{LA_i}$ is a function that describes the set of allowable actions at a given local state. For example, the following is one protocol depicted from Figure 4. $Pr_n(analyzeQoSInfo) = \{$*send_cluster_qos_results*, *send_task_qos_results*$\}$.

At a given time, the configuration of all services in the system is characterized as a global state S of n elements represented as $gs = \{e_1, e_2, \ldots, e_n\}$, where each element $e_i \in LS_i$ denotes a local state of the service i. Hence, the set of all global states $GS = \{LS_1 \text{ X } LS_2 \text{ X } \ldots \text{X } LS_n\}$ is the Cartesian product of all the local states of n services. The global transition function is defined as $T : GS \text{ X } LA \rightarrow GS$, where $LA = \{LA_1 \text{ X } LA_2 \text{ X } \ldots \text{X } LA_n\}$. The local transition function is defined as $T_i : LS_i \text{ X } LA_i \rightarrow LS_i$.

Definition (Model) Our model is represented as a *non-deterministic Buchi automaton* as a quintuple $MDL = (G, TR, I, F, v)$ where:

1. $G \subseteq LS_1 \text{ X } LS_2 \text{ X } \ldots \text{X } LS_n$ is a finite set of global states of the system.
2. $TR \subseteq G \text{ X } G$ is a transition relation defined by $(g, g') \in TR$ if there exists a joint action $(a_1, a_2, \ldots, a_n) \in LA$ such that $TR(g, a_1, \ldots, a_n) = g'$. a_i is called a joint action and is defined as a tuple of actions.
3. $I \subseteq G$ is a set of initial global states of the system.
4. $F \subseteq G$ is a set of final global states of the system.

ACCEPTED MANUSCRIPT

5. $V : AP \rightarrow 2^{G}$ is *the* valuation function where AP is a finite set of atomic propositions.

Then MDL, is a Deterministic Buchi Automaton (DBA) if and only if $\forall\ q \in GS$ *and* $a \in i$ it holds that $|TR\ (q,\ a)| = 1$.

Having this formal representation of the system, allows easy implementation using the symbolic model checker, MCMAS [51]. The MCMAS tool is used for automatic verification of the correctness of the system expressed in Computation Tree Logic (CTL) against the reachability, liveness and safety properties [52]. It helps in checking and confirming that our model meets its specification and expectations exhaustively and automatically.

Definition (Syntax). The CTL syntax is represented using the following grammar rules:

$\Phi ::= p \mid \neg\ \Phi \mid \Phi\ \vee\ \Phi \mid EX\ \Phi \mid EG\ \Phi \mid E\ (\Phi\ U\ \Phi)$ where the atomic proposition p ϵ AP; E is the existential quantifier on paths, and X, G, and U are path modal connective standing for "next", "globally", and "until", respectively. The Boolean connectives $\neg$ and $\vee$ are defined and read as "not", and "or" respectively.

Temporal properties:

The correctness of our system model can be checked using CTL by demonstrating the following significant properties:

1. **Reachability property:** given a certain state, is there a computation sequence to reach that state from the initial state? The used reachability properties are defined as:

$$\Phi 1 = EF Detect\_App\_Abnormality \tag{4}$$

$$\Phi 2 = EF Change\_Deployment \tag{5}$$

$$\Phi 3 = EF Save\_QoS\_Logs \tag{6}$$

ACCEPTED MANUSCRIPT

The formulas $\phi 1$, $\phi 2$, and $\phi 3$ check whether or not there exists a path to reach the *Detect_App_Abnormality* state, *Change_Deployment* state, and *Save_QoS_Logs* state respectively.

$$\Phi 4 \;=\; E(\neg Analyze\_QoS \; U \; (Analyze\_QoS \;\wedge\; EF(Collect\_QoS))) \tag{7}$$

The formula $\phi 4$ represents that there exists a path where the *Analyze_QoS* process will not start analyzing QoS data until the QoS data is collected.

2. **Liveness property:** this property reflects that "*something good will eventually happen.*" For example, in all paths globally if the System Analyze QoS detects an abnormality, then there is a path in its future through which the system will deploy the change for automatic reconfiguration thereby enhancing the quality of the orchestration.

$$\Phi 5 \;=\; AG(Detect\_App\_Abnormality \rightarrow EF \; Change\_Deployment) \tag{8}$$

3. **Safety property:** this property ensures that "*something bad never happens.*" An example of a bad situation is when the user enters correctly the required information to configure the system, but the latter never initializes the monitoring cycle.

$$\begin{aligned} \Phi 6 \quad = \quad & AG \; \neg \, (Config\_Monitoring \; (Correct\_Info) \;\wedge \\ & EF \; \neg \, App\_Start\_Monitoring) \end{aligned} \tag{9}$$

6. Experiments and Evaluation

In addition, to the above monitoring system validation using model checker, we describe in this section the experimental evaluation we conducted to assess our workflow monitoring model. Therefore, we first evaluate the system overhead then we evaluate three adaptions schemes we propose to dynamically reconfigure the workflow during its execution to respond to any cloud services performance degradation. We first, describe the environment set-up we configured and the key modules implemented to support monitoring and adaptation. We then depict the workflow we developed for evaluation purposes and the dataset

ACCEPTED MANUSCRIPT

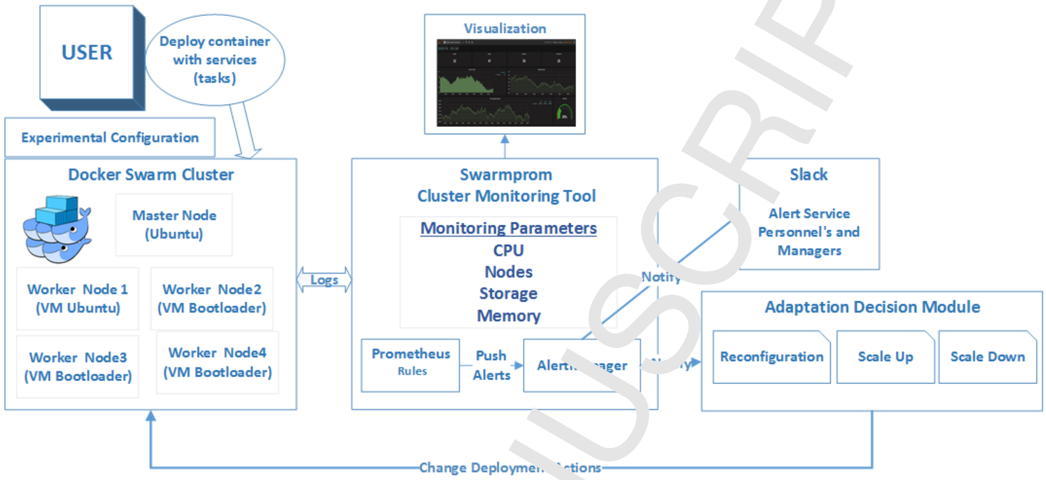

Figure 5: System implementation architecture.

we chose to execute our workflow. A set of scenarios were carefully chosen to evaluate workflow monitoring and the different adaptation schemes we implemented. Finally, we report and discuss the results we have obtained from the experimentations.

6.1. Environment Setup

Figure 5 describes the environment we established to execute, monitor, and dynamically adapt our workflow to respond to different performance degradation situations. In the following, we briefly describe each component of our experimentation configuration:

Docker Swarm Cluster. The Docker swarm cluster consisted of one master node and four worker nodes. We used Oracle Virtual Box driver to create the Docker nodes. These Swarm nodes can run any operating system and be managed on any cloud infrastructure. The workflow shown in Figure 6 is deployed on the Swarm cluster, and a Master node performs the orchestration and cluster management required to maintain the desired state of the swarm. Worker nodes receive and execute tasks dispatched from the manager/master node. To

ACCEPTED MANUSCRIPT

deploy an application to a swarm, a service definition is submitted to a manager node, and the manager node dispatches units of work, called tasks, to the worker nodes [53].

Swarmprom Cluster monitoring tool. This is a monitoring starter toolkit for Docker swarm services [54] equipped with *Prometheus*, *Grafana*, *cAdvisor*, *Node Exporter*, *Alert Manager*, and *Unsee*. These tools serve in providing continuous system performance measurements that are collected and analyzed by our monitoring system. Swarmprom Grafana [55] is configured with two dashboards and Prometheus [56] as the default data source. Monitoring parameters include CPU, memory, storage, and nodes, and Prometheus rules were used to monitor these parameters. Alert manager uses Slack, which is a cloud-based team collaboration tools and services. It brings team's communication together where conversations are organized and made accessible [57]. The Swarmprom Alert Manager can direct alerts through the Slack webhook APIs that is posted to the specific channels and alerts the concerned Managers and Service personnel who are on the move.

Adaptation Decision Module: This implements different reconfiguration decisions and is developed in the Perl language. An agent runs as a background process, which constantly monitors the CPU and memory status of the Docker services. Based on rules, the adaptation decision module inspects the Docker services and performs the necessary automatic reconfiguration of nodes in the cluster, such as scale up or scale down the services.

Visualization Module. This implements a dashboard to visualize in real-time monitoring information, including resource usage of both Swarm nodes and the services running on these nodes. It also integrates some visualization features, such as Zoom in and out, and filtering. Graffana is an open source monitoring dashboard implemented with Docker.

6.2. Workflow and Dataset Description

In this section, we describe the dataset we used in our workflow as well as the workflow implementation and its composing tasks.

ACCEPTED MANUSCRIPT

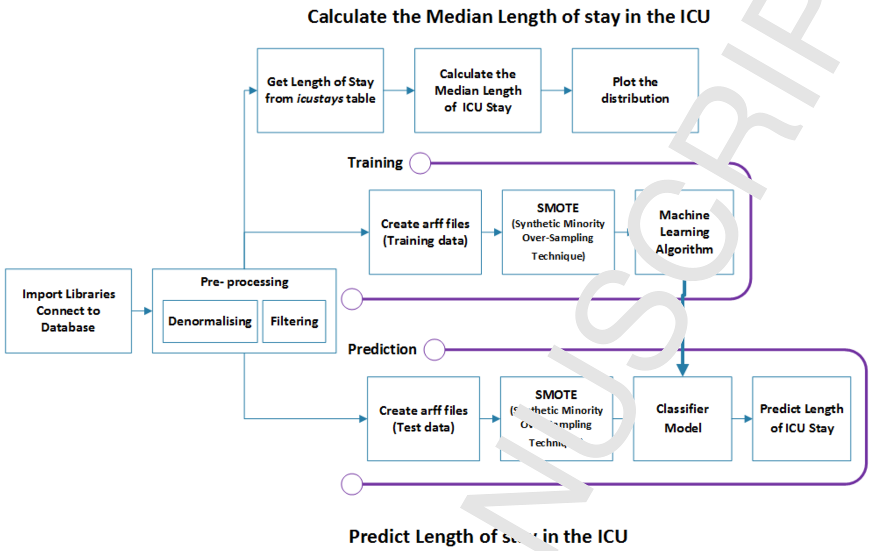

Figure 6: Health monitoring workflow description.

6.2.1. Dataset

The dataset we used to implement our workflow was retrieved from the Multiparameter Intelligent Monitoring in Intensive Care III (MIMICIII) database [58]. The dataset incorporates sixty thousand admissions of patients who stayed in critical care units Medical Center between 2001 and 2012. The database is available via PhysioNet, a web-based data resource that contains various physiological records. The available clinical information includes patient demographics, vital sign measurements, hospital admissions, laboratory tests, medications, fluid intake records and out-of-hospital mortality. We chose this dataset as it conforms with the characteristics of Big Data as it depicts high volume, and velocity, and veracity (diverse). Therefore, it can be considered as a very representative dataset that feeds the different tasks and processes of the workflow.

6.2.2. Workflow Description

Figure 6 describes a health monitoring workflow we developed using the MIMICIII dataset to evaluate different aspects of an automatic reconfiguration workflow

ACCEPTED MANUSCRIPT

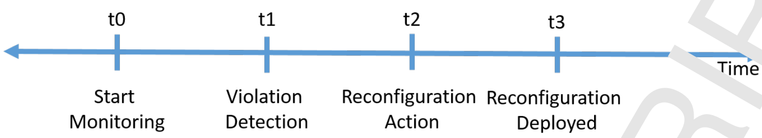

Figure 7: Monitoring time line.

scheme we proposed in this section. The workflow is deployed on the Swarm cluster with PostgreSQL installed and the MIMIC database tables loaded automatically [59] to perform the service tasks as outlined in the workflow. It consists of a set of tasks some of which are sequential and others parallel. The sequential tasks include retrieving data from the MIMIC database and conducting data processing, while the parallel tasks include training and prediction tasks.

6.3. System Overhead Measurement

6.3.1. Latency Overhead

In this section we describe the latency of our framework from data collection to making a decision. For example, in the following scenario described in Figure 7.

T1 is the violation detection time (e.g. cpu utilization overload) T2 is the reconfiguration action start (add node) T3 is the reconfiguration action complete (node is ready) We calculate $Latency = T3 - T1$. Adding a new node is immediate it takes few milliseconds. The mean latency is measured to 4 ms.

6.3.2. Communication Overhead

We estimate the communication overhead by measuring the size of the exchanged messages in the monitoring and the adaptation modules in bytes as follows.

$$Size(getLTSMsg) = 1 + number\ of\ quality\ attributes\ + number\ of\ tasks \tag{10}$$

ACCEPTED MANUSCRIPT

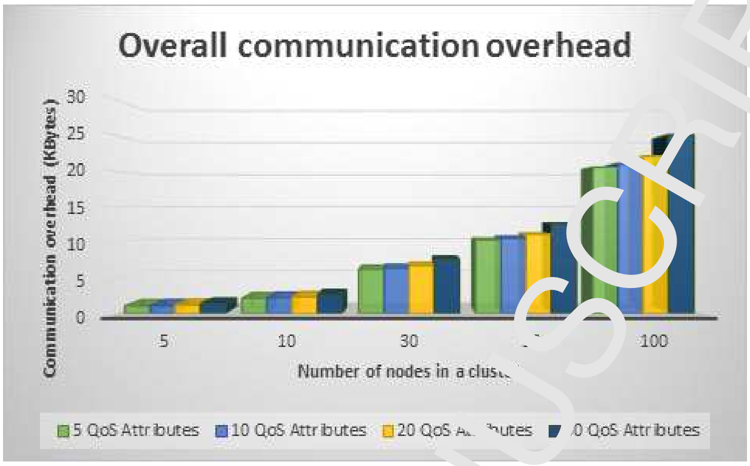

Figure 8: Communication overhead.

$$Size(replyLTSMsg) = (size(LTS) \times number\ of\ tasks) + number\ of\ violations \quad (11)$$

, where size (LTS) is 2 bytes.

$$size(AllRepMsg) = Size(replyLTSMsg) \times number\ of\ nodes \quad (12)$$

Figure 8 depicts that the communication overhead is proportional to both the number of nodes in the cluster and the number of selected quality attributes used for trust measurement. With 100 nodes, 50 quality attributes, and 100 tasks in a workflow the calculated overall communication overhead was nearly negligible (25 KBytes). This proves that our monitoring and reconfiguration framework is lightweight as it does not incur a heavy load on the workflow nor the cloud resources handling it.

From our experiment results we can conclude that our framework is effectively responding to dynamic cloud environment changes when compared to non adaptation scenarios. In our decision making we take into consideration two sources

ACCEPTED MANUSCRIPT

of information: past experience which is used for prediction of resource status and the current monitoring information.

6.4. Cloud Workflow Adaptation Strategies

We use the same workflow with different data sizes and processing complexity. Our baseline for comparison is workflow without adaptation or reconfiguration, measuring throughput, response time, CPU utilization, memory utilization, and execution time.

6.4.1. Scale-up (Client Gain)

In this scenario, we overload some nodes with extra processing tasks to affect the QoS of our workflow under investigation. We check the effect of our proposed framework including the monitoring and the automatic reconfigure modules on the QoS performance of the workflow. First the monitoring module will detect that the currently running tasks have lower performance due to overloading of assigned nodes. Then, it forwards a message to the AR modules which in turn will issue a scale-up command message to the specific task at the assigned cluster (node). Scale-up will add more nodes to process the task, which will result in improving task performance.

6.4.2. Scale-down (Provider Gain)

Scaling down is performed when resources are not utilized in an optimized manner. This is done when the monitoring module detects low utilized nodes' CPU, which requires deletion of under loaded nodes from the cluster. In this scenario, we add an unnecessary number of nodes in the cluster handling the task and check the performance of the cluster before and after the scale-down.

6.4.3. Migration (Client and Provider Gain)

Workflow migration is usually needed if the cluster is overloaded with no extra resources available to be added to the cluster. In this scenario, we overload all the nodes of a cluster until they become slow in processing workflows as required, this will necessitate a migration of the workflow to a new data cluster.

ACCEPTED MANUSCRIPT

We observe the performance of the workflow and the cluster before and after the migration is performed.

6.5. Results and Discussion

In our experiments, we run the aforementioned workflow several times through which we use different dataset sizes and processing resource capacity. We apply our adaptation strategies to the workflow execution and compare the performance against a baseline scenario with no adaptation scheme, such as CPU utilization, memory usage, and trust scores. We run our monitoring system throughout the workflow execution. In our experiments we have collected and inspected data samples from a set of samples, that constitute a representative selection from all data measurements. We took random sample from a population to compute the mean and to find the approximation of mean of a sample. Additionally, we built confidence interval to see how well the mean estimates the underlying population which give the range of values within which there is a specified probability that the value of a parameter lies in it. In our experiments we choose to use 95% confidence interval. Here every point on the graph is an average of 10 measurements taken in 30 seconds duration. For example, for memory usage in scenario 2, most of the taken values within the 95% confidence intervals were overlapping, which verifies that our experimentation was rightly done. We used 10 measurements for each point on the graphs representing all our experiments. Additionally, in all our experiments, every point on the graph is an average of the measurements taken in 30 seconds duration. We considered the following default simulation parameters:

- Node: Each node in our cluster has an Intel CoreTM i7-3770K CPU @ 3.40GHz with Turbo Boast, 32GB of DDR3 RAM, 1TB hard drive, and 64-bit operating system
- Number of Clusters: 1 - 3
- Number of nodes within each cluster: 1 - 6

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

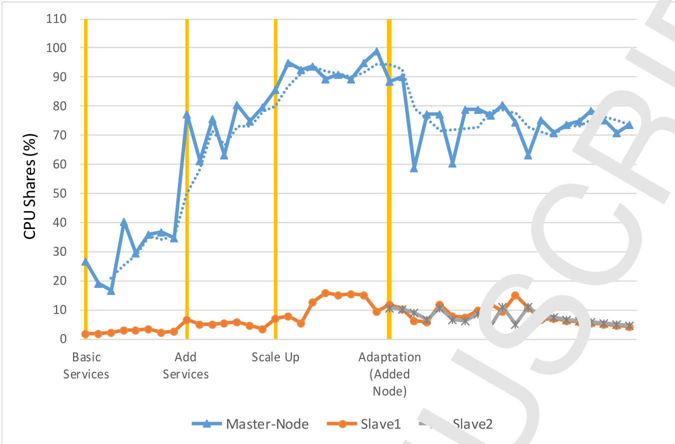

Figure 9: CPU utilization shares.

<u>Scenario 1:</u> In this scenario, we evaluate the CPU utilization of a workflow among the nodes in the cluster. Figure 9 shows that CPU utilization increases as the workflow services are executed. However, the CPU utilization reaches significantly high values when the number of services increases. Thus, our monitoring system detects this issue and alerts the reconfiguration system which decides to add a new node and, accordingly, the load on the existing nodes is relaxed.

<u>Scenario 2:</u> In this scenario, we evaluate the workflow memory usage for one of the nodes in the cluster. After adding a new node to the cluster resulting from an adaptation decision, the overall memory usage is significantly lower when compared to the usage in the case of no adaptation applied despite the increase in the size of the dataset as depicted in Figure 10.

<u>Scenario 3:</u> In this scenario, we monitor the CPU utilization and the memory usage of each task in the workflow. Whenever the CPU and memory performance is degraded, the reconfiguration system suggests adding resources to the cluster such as a new node in order to enhance the overall performance. Figure 11 shows some examples of tasks' memory usage and CPU utilization before and after adding a new node during which the dataset size increase overtime.

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

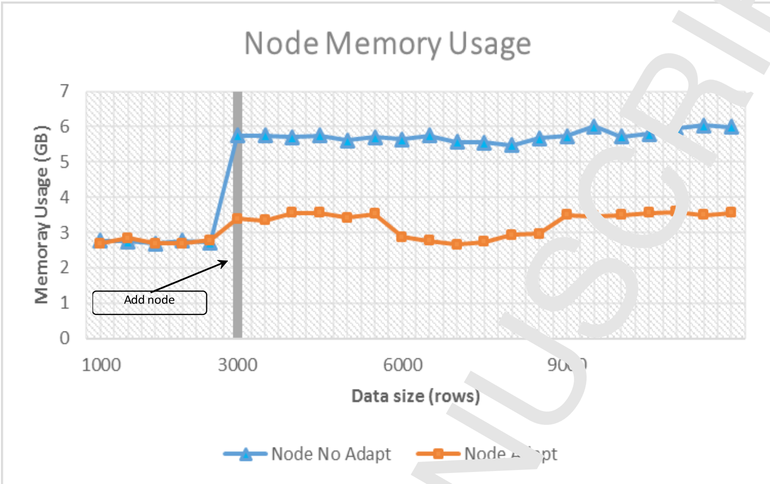

Figure 10: Node memory usage.

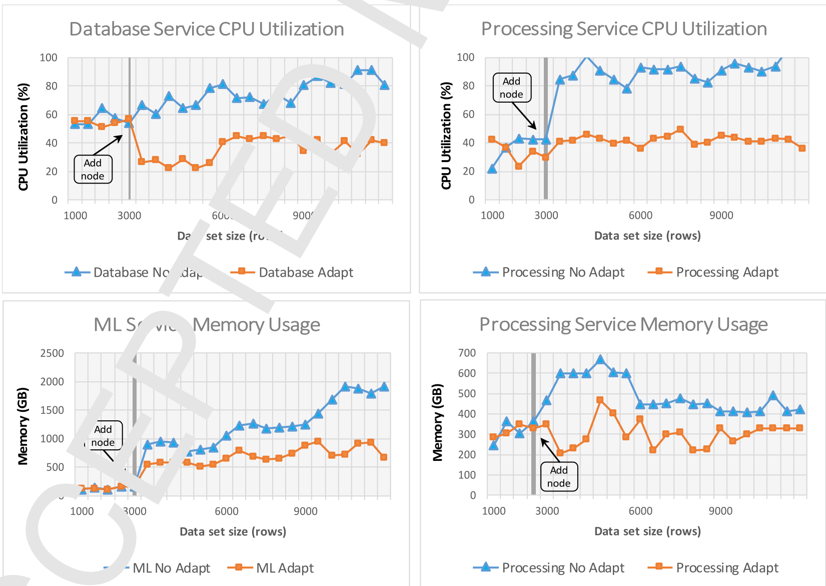

Figure 11: Service CPU utilization and memory usage.

ACCEPTED MANUSCRIPT

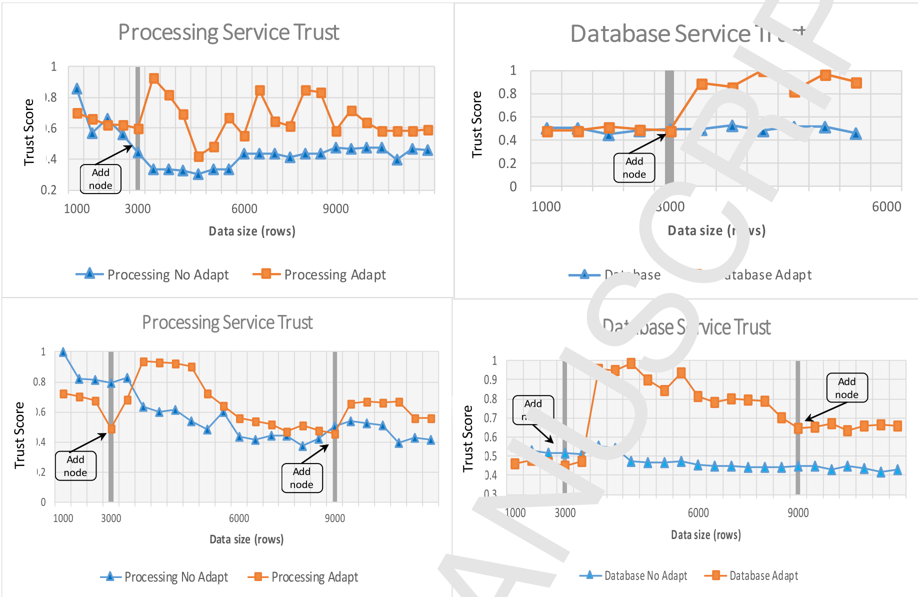

Figure 12: Service trust (1-step and 2-step adaptation).

The figure clearly shows the enhanced performance after adding an extra node.

Scenario 4: In this scenario, we compute different service trust scores for processing and database services. Figure 12 shows examples of service trust scores evaluated over time during which the dataset size is increased. The trust score decreases as the data size increases till a threshold is reached and a new node is added to the cluster. The two upper figures of Figure 12 shows one step adaptation, and the lower two figures depict two-step adaptation. The more the data increases the more nodes are required to process this data, and the trust scores increase after adaptation (i.e., adding extra nodes).

Scenario 5: In this scenario, we use scaled-down adaptation were we delete selected underloaded nodes when the CPU or memory utilization degrades. Figure 13 shows an example of a service resource utilization versus the number of nodes. We start at six nodes, at which we detect a low memory usage and CPU utilization per service. The system decides to delete two nodes which increases the utilization to an accepted level of about 25%. The figure also

ACCEPTED MANUSCRIPT

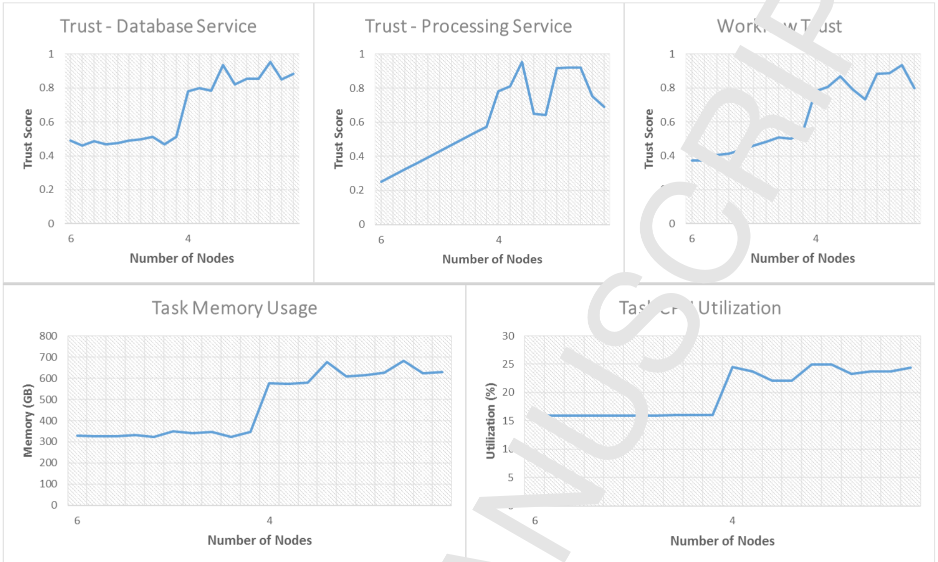

Figure 13: Scale down resources due to low utilization.

shows low Trust scores for some services and the overall workflow when we use an unnecessarily large number of nodes. The trust score increases when the utilization improves after adaptation (i.e., node deletion).

Scenario 6: In this scenario, we reduce the data size to reach low resources utilization. The monitoring system detects the low utilization quality violation and issues a node deletion adaptation decision. Figure 14 shows that after a reduction of data size, memory usage and CPU utilization degrade and eventually the trust score decreases. After deleting the node, the trust increases again as the resource utilization improves.

Scenario 7: In this scenario, we perform a two-stage up-scale by adding a node at each stage. In the first stage, we use smaller dataset sizes, and we incremented it gradually. When the task CPU utilization and memory usage increase above a threshold, a new node is added to the cluster. In the second stage, we further gradually increase the dataset size until the monitored QoS attributes increase beyond the required threshold, and then another node is

ACCEPTED MANUSCRIPT

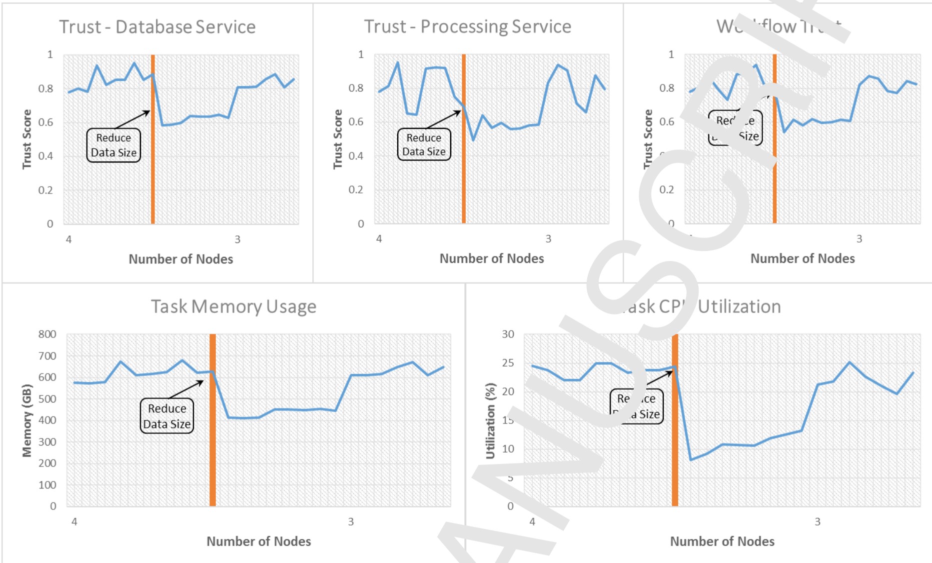

Figure 14: Scale down resources due to data size reduction.

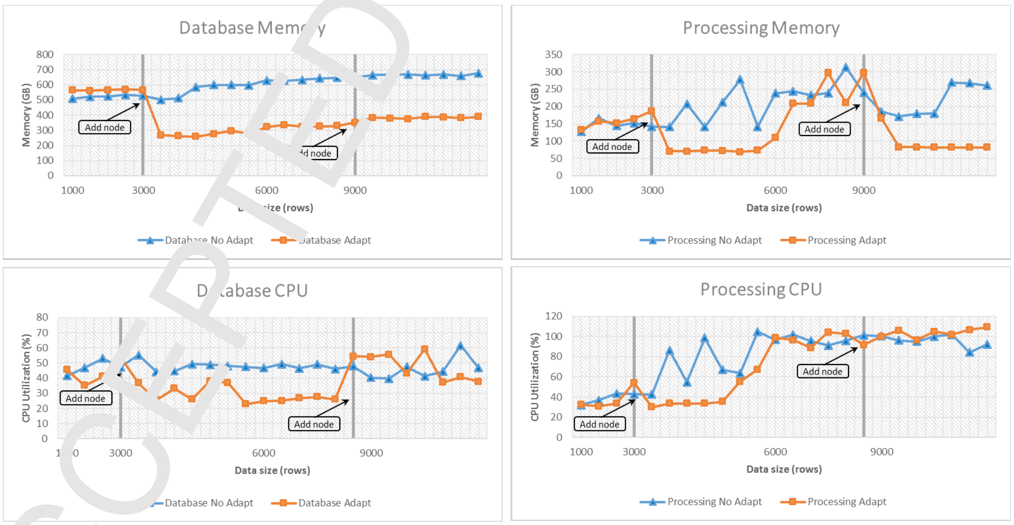

Figure 15: Two-stage resource upscale (node addition).

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

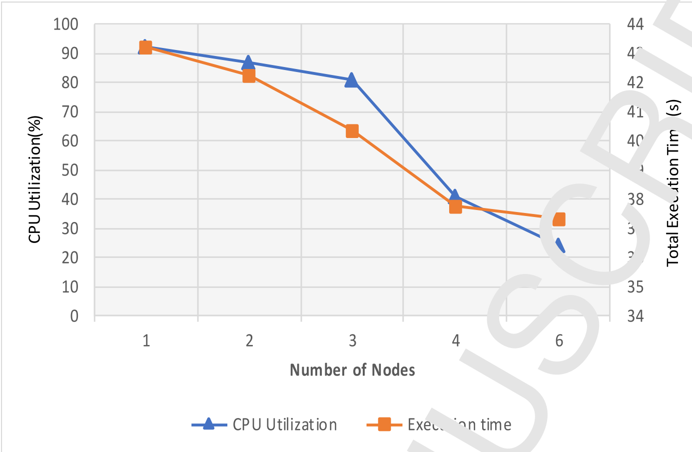

Figure 16: Total execution time.

added. The results show an improvement of the performance after adding a node as shown in Figure 15. For some of the monitored services, the second stage adaptation does not reduce the CPU utilization but maintains a good performance level to compensate for the dataset size increase and prevents the service performance degradation. The figure also shows that our adaptation mechanism displays better QoS performance levels in comparison to the baseline of no adaptation service performance.

Scenario 8: In this scenario, we perform multi-fold adaptation to optimize the total workflow execution time and CPU utilization. We monitor the aforementioned quality attributes and perform multiple node additions and adaptation actions until we reach the required quality level. Figure 16 shows a high CPU utilization level which triggers an adaptation action of adding a new node. However, the second monitoring cycle detected a quality violation and thus more nodes are added until we reach an adequate CPU Utilization. Adding nodes revealed an improvement of the total execution time as shown in Figure 16.

Scenario 9: In this scenario, we evaluate the migration adaptation decision. The currently used cloud cluster has limited resources and shows no possibility of further resource addition. Upon a quality degradation detection, in this case,

ACCEPTED MANUSCRIPT

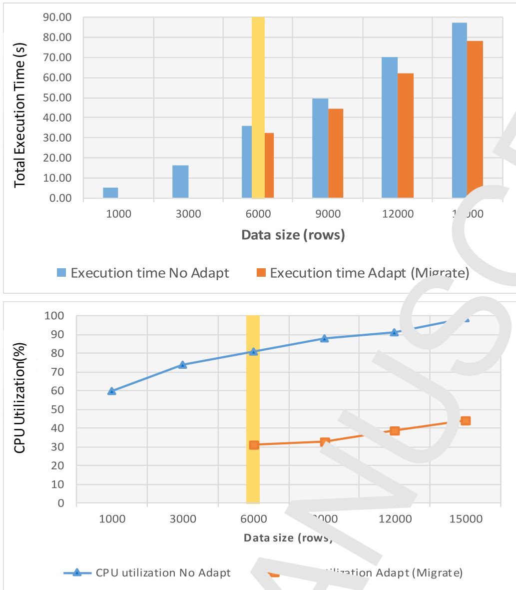

Figure 17: Total execution time and CPU utilization after migration.

CPU utilization, the reconfiguration manager reacts with a decision to migrate the workflow to another selected cluster offering more resources that can fulfill the requirements of the workflow under investigation. For simplicity we decided to migrate the full workflow to another cloud since at a certain data size (6000 rows), the monitoring module detects an unaccepted degradation of performance while there are no more cloud resources to accommodate the increase in data size, the workflow along with its dataset is moved to another cloud. The results show an average of 11.5% improvement of the total workflow execution time and a significant enhancement of CPU utilization after migration for different sizes of the datasets as shown in Figure 17.

6.6. Overall Discussion

In this section, we discuss and evaluate our experimental results, which validated our monitoring and reconfiguration model by adopting the following strategies: 1) overload the system and monitor the workflow and cloud resources, and 2)

ACCEPTED MANUSCRIPT

underload the system and monitor the workflow and cloud resources. After that we test the reaction of the system and its effect on quality. Our objective is to keep the quality performance within the user's required ranges and the accepted trust scores.

Results show that our monitoring system detects the violation triggered when the quality attribute performance goes out the accepted or required range. This is reported to the automatic reconfiguration system which in turn issues the appropriate action to keep the required quality level.

In scenarios 1 through 4, we overload the system, monitored the CPU utilization, memory usage, and trust scores, and detected the quality violation. In all scenarios, the possible reconfiguration actions such as adding new nodes at different stages, confirmed the improvement of the overall performance. In scenarios 5 through 6, we underload the system to detect lower resource utilization; then the reconfiguration manager would deallocate nodes as expected and accordingly improve the resource utilization.

We also tested the workflow migration and its effect on total time execution, and the results showed a significant improvement.

In terms of scalability of cloud resources, our experimenta-tion setup included 6 nodes which we judged sufficient to evaluate our proposed adaptation strategies. However, this setup can scale with more resources and nodes whenever the workflow complexity increases, and its processing and analytics requirements are crucial.

7. Conclusion

Provision of Cloud workflows QoS during execution necessitates monitoring and adaptation. The complexity of this process arises because of the dynamic nature of cloud resources and services, the variety of resources provisioning, and the variation of the workflow contexts and requirements. In this section, we proposed a trust-based model to support monitoring and adaptation of cloud workflows to guarantee a required level of QoS. This model handled the dy-

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

namic nature of cloud resources and services and coped with the complexity of workflow monitoring and adaptation. The proposed model supported workflow self-reconfiguration and self-adaption. Workflow reconfiguration is triggered to respond to performance violation detection after real-time monitoring of cloud resources. To capture different dynamic properties of the workflow and the cloud execution environment, we formalized the cloud resource orchestration using a state machine and we validated it using model checker.

We conducted a series of experiments to evaluate our workflow monitoring, and adaptation using various monitoring and adaptation scenarios executed over a cloud cluster. The workflow is implemented and deployed over a Docker cluster. It fulfills a set of health monitoring processes and datasets where resource shortage is contingent to workflow performance degradation. The results we obtained from these experiments prove that our automated workflow orchestration model is self-adapting, self-configuring and reacts efficiently to various cloud environment changes and adapt accordingly while supporting a high level of workflow QoS.

As future work, we will use the prediction of resource shortage to guarantee QoS prior to violation. This will strengthen our model to benefit from both real monitoring and prediction to proactively react efficiently to performance degradations and resource shortage. We are also currently extending our model while considering more applications to be tested using our framework and provide more performance evaluation scenarios.

References

[1] D. Weerasiri, Configuration and orchestration techniques for federated cloud resources, Ph.D. thesis, The University of New South Wales (2016).

[2] A. L. Lemos, F. Daniel, B. Benatallah, Web service composition: a survey of techniques and tools, ACM Computing Surveys (CSUR) 48 (3) (2016) 33.

[3] deltacloud, http://deltacloud.apache.org, accessed: 2018-05-01.

ACCEPTED MANUSCRIPT

[4] Apache libcloud, `http://libcloud.apache.org`, accessed: 2018-05-01.

[5] jclouds, `http://www.jclouds.org`, accessed: 2018-05-01.

[6] openstack, `http://www.openstack.org`, accessed: 2018-05-01.

[7] B. Yu, M. P. Singh, An evidential model of distributed reputation management, in: Proceedings of the first international joint conference on Autonomous Agents and Multiagent Systems: Part 1, ACM, 2002, pp. 294–301.

[8] X. Li, W. Hu, T. Ding, R. Ruiz, Trust constrained workflow scheduling in cloud computing, in: Systems, Man, and Cybernetics (SMC), 2017 IEEE International Conference on, IEEE, 2017, pp. 164–169.

[9] D. Bernstein, D. Vij, Intercloud security considerations, in: Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, IEEE, 2010, pp. 537–544.

[10] J. Abawajy, Determining service trustworthiness in intercloud computing environments, in: Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on, IEEE, 2009, pp. 784–788.

[11] K. Keahey, M. Tsugawa, A. Matsunaga, J. Fortes, Sky computing, IEEE Internet Computing 13 (5) (2009) 43–51.

[12] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond, M. Morrow, Blueprint for the intercloud–protocols and formats for cloud computing. internet and web applications and services, 2009. iciw'09, in: Fourth International Conference, 2009, pp. 1–3.

[13] A. N. Toosi, R. N. Calheiros, R. Buyya, Interconnected cloud computing environments: Challenges, taxonomy, and survey, ACM Computing Surveys (CSUR) 47 (1) (2014) 7.

ACCEPTED MANUSCRIPT

[14] D. Weerasiri, M. C. Barukh, B. Benatallah, Q. Z. Sheng, R. Ranjan, A taxonomy and survey of cloud resource orchestration techniques, ACM Computing Surveys (CSUR) 50 (2) (2017) 26.

[15] W. Barth, Nagios: System and network monitoring, No Starch Press, 2008.

[16] P. Zadrozny, R. Kodali, Big Data Analytics Using Splunk: Deriving Operational Intelligence from Social Media, Machine Data, Existing Data Warehouses, and Other Real-Time Streaming Sources, Apress, 2013.

[17] Ganglia monitoring system, `http://ganglia.sourceforge.net/`, accessed: 2018-04-01.

[18] Apache chukwa, `http://chukwa.apache.org/`, accessed: 2018-04-01.

[19] Sematext, `https://sematext.com/`, accessed: 2018-04-01.

[20] Sequenceiq, `http://sequenceiq.com/`, accessed: 2018-04-01.

[21] K. Alhamazani, R. Ranjan, K. Mitra, F. Rabhi, P. P. Jayaraman, S. U. Khan, A. Guabtni, V. Bhatnagar, An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the-art, Computing 97 (4) (2015) 357–377.

[22] K. Alhamazani, R. Ranjan, P. P. Jayaraman, K. Mitra, F. Rabhi, D. Georgakopoulos, L. Wang, Cross-layer multi-cloud real-time application qos monitoring and benchmarking as-a-service framework, IEEE Transactions on Cloud Computing.

[23] S. Clayman, A. Galis, C. Chapman, G. Toffetti, L. Rodero-Merino, L. M. Vaquero, K. Nagin, B. Rochwerger, Monitoring service clouds in the future internet, in: Future Internet Assembly, Valencia, Spain, 2010, pp. 115–126.

[24] L. Romano, D. De Mari, Z. Jerzak, C. Fetzer, A novel approach to qos monitoring in the cloud, in: Data Compression, Communications and Processing (CCP), 2011 First International Conference on, IEEE, 2011, pp. 45–51.

ACCEPTED MANUSCRIPT

[25] S. A. De Chaves, R. B. Uriarte, C. B. Westphall, Toward an architecture for monitoring private clouds, IEEE Communications Magazine 49 (12) (2011) 130–137.

[26] R. Ranjan, S. Garg, A. R. Khoskbar, E. Solaiman, P. James, D. Georgakopoulos, Orchestrating bigdata analysis workflows, IEEE Cloud Computing 4 (3) (2017) 20–28.

[27] Yarn, `https://yarnpkg.com/en/`, accessed: 2018-04-01.

[28] Apache mesos, `http://mesos.apache.org/`, accessed: 2018-04-01.

[29] Amazon emr, `https://aws.amazon.com/emr/`, accessed: 2018-04-01.

[30] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, A. Rasin, Hadoopdb: an architectural hybrid of mapreduce and dbms technologies for analytical workloads, Proceedings of the VLDB Endowment 2 (1) (2009) 922–933.

[31] R. Castro Fernandez, M. Migliavacca, E. Kalyvianaki, P. Pietzuch, Integrating scale out and fault tolerance in stream processing using operator state management, in: Proceedings of the 2013 ACM SIGMOD international conference on Management of data, ACM, 2013, pp. 725–736.

[32] A. Castiglione, M. Gribaudo, M. Iacono, F. Palmieri, Exploiting mean field analysis to model performances of big data architectures, Future Generation Computer Systems 37 (2014) 203–211.

[33] D. Bruneo, F. Longo, R. Ghosh, M. Scarpa, A. Puliafito, K. S. Trivedi, Analytical modeling of reactive autonomic management techniques in iaas clouds, in: Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on, IEEE, 2015, pp. 797–804.

[34] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, H. Chang, Qos-aware middleware for web services composition, IEEE Transactions on software engineering 30 (5) (2004) 311–327.

ACCEPTED MANUSCRIPT

[35] A. F. M. Hani, I. V. Paputungan, M. F. Hassan, Renegotiation in service level agreement management for a cloud-based system, ACM Computing Surveys (CSUR) 47 (3) (2015) 51.

[36] H. Kim, M. Parashar, Cometcloud: An autonomic cloud engine, Cloud Computing: Principles and Paradigms (2011) 275–297.

[37] A. Nasridinov, J.-Y. Byun, Y.-H. Park, A qos-aware performance prediction for self-healing web service composition, in: Cloud and Green Computing (CGC), 2012 Second International Conference on, IEEE, 2012, pp. 799–803.

[38] S. Schulte, C. Janiesch, S. Venugopal, I. Weber, P. Hoenisch, Elastic business process management: State of the art and open challenges for bpm in the cloud, Future Generation Computer Systems 46 (2015) 36–50.

[39] S. Singh, I. Chana, Qos-aware autonomic resource management in cloud computing: a systematic review, ACM Computing Surveys (CSUR) 48 (3) (2016) 42.

[40] S. Ferretti, V. Ghini, F. Panzieri, M. Pellegrini, E. Turrini, Qos-aware clouds, in: Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, IEEE, 2010, pp. 321–328.

[41] R. Nathuji, A. Kansal, A. Ghaffarkhah, Q-clouds: managing performance interference effects for qos-aware clouds, in: Proceedings of the 5th European conference on Computer systems, ACM, 2010, pp. 237–250.

[42] X. Li, K. Li, X. Pang, Y. Wang, An orchestration based cloud auto-healing service framework, in: Edge Computing (EDGE), 2017 IEEE International Conference on, IEEE, 2017, pp. 190–193.

[43] S. Nepal, Z. Malik, A. Bouguettaya, Reputation propagation in composite services, in: Web Services, 2009. ICWS 2009. IEEE International Conference on, IEEE, 2009, pp. 295–302.

[44] L. Qu, Y. Wang, M. A. Orgun, L. Liu, H. Liu, A. Bouguettaya, Cccloud: Context-aware and credible cloud service selection based on subjective assessment and objective assessment, IEEE Transactions on Services Computing 8 (3) (2015) 369–383.

[45] J. Huang, G. Liu, Q. Duan, Y. Yan, Qos-aware service composition for converged network-cloud service provisioning, in: Services Computing (SCC), 2014 IEEE International Conference on, IEEE, 2014, pp. 67–74.

[46] Tools for monitoring compute, storage, and network resources, `https://kubernetes.io/docs/tasks/debug-application-cluster/resource-usage-monitoring/`, accessed: 2018-05-01.

[47] H. T. E. Kassabi, M. A. Serhani, R. Dssouli, B. Benatallah, A multi-dimensional trust model for processing big data over competing clouds, IEEE Access 6 (2018) 39989–40007. doi:10.1109/ACCESS.2018.2856623.

[48] M. A. Serhani, H. A. Kassabi, I. Taleb, Towards an Efficient Federated Cloud Service Selection to Support Workflow Big Data Requirements, Advances in Science, Technology and Engineering Systems Journal 3 (5) (2018) 235–247. doi:10.25046/aj030529.

[49] A. Adriyendi, Multi-attribute decision making using simple additive weighting and weighted product in food choice, International Journal of Information Engineering and Electronic Business 6 (2015) 8–14.

[50] Yet another markup language (yaml) 1.0, `http://yaml.org/spec/history/2001-12-10.html`, accessed: 2018-05-01 (2001).

[51] A. Lomuscio, H. Qu, F. Raimondi, Mcmas: A model checker for the verification of multi-agent systems, in: International conference on computer aided verification, Springer, 2009, pp. 682–688.

[52] E. M. Clarke, O. Grumberg, D. Peled, Model checking, MIT press, 1999.

ACCEPTED MANUSCRIPT

[53] Swarm mode key concepts, docker doc, https://docs.docker.com/engine/swarm/key-concepts/, accessed: 2018-05-01 (2017).

[54] S. Prodan, Docker swarm instrumentation with prometheus, https://stefanprodan.com/2017/docker-swarm-instrumentation-with-prometheus/, accessed: 2018-05-01.

[55] Grafana - the open platform for analytics and monitoring, https://grafana.com/, accessed: 2018-05-01 (2017).

[56] Prometheus - monitoring system & time series database, https://prometheus.io/, accessed: 2018-05-01 (2017).

[57] Slack features, https://slack.com/features, accessed: 2018-05-01 (2017).

[58] The mimic-iii clinical database, https://www.physionet.org/physiobank/database/mimic3cdb/, accessed: 2018-05-01 (2017).

[59] Mit-lcp/mimic-code, https://github.com/MIT-LCP/mimic-code/tree/master/buildmimic/docker, accessed: 2018-05-01 (2017).

ACCEPTED MANUSCRIPT

ACCEPTED MANUSCRIPT

Author's Biographies

HADEEL T. ELKASSABI received the Bachelor of Science degree in Computer Science from The American University in Cairo in 1996. She received the M.S degrees in Computer Science from Carleton University, Ottawa in 2003. She is a candidate Ph. D. Student at the Concordia Institute for Information Systems Engineering (CIISE), Concordia University. Her current interest areas are "Big Data", "Cloud Computing", "Trust Modeling" and "Data Quality".

DR. M. ADEL SERHANI is currently an Associate Professor, College of Information Technology, U.A.E University, Al Ain, and U.A.E. He is also an Adjunct faculty in CIISE, Concordia University, Canada. He holds a Ph.D. in Computer Engineering from Concordia University in 2006, and MSc. in Software Engineering from University of Montreal, Canada in 2002. His research interests include: Cloud for data intensive e-health applications, and services; SLA enforcement in Cloud Data centers, and Big data value chain, Cloud federation and monitoring, Non-invasive Smart health monitoring; management of communities of Web services; and Web services applications and security. He has a large experience earned throughout his involvement and management of different R&D projects. He served on several organizing and Technical Program Committees and he was the program Co-chair of the IEEE conference on Innovations in Information Technology (IIT´13), Chair of IEEE Workshop on Web service (IWCMC´13), Chair of IEEE workshop on Web, Mobile, and Cloud Services (IWCMC´12), and Co-chair of International Workshop on Wireless Sensor Networks and their Applications (NDT´12). He has published around 90 refereed publications including conferences, journals, a book, and book chapters.

DR. RACHIDA DSSOULI is a full professor and Director of Concordia Institute for Information Systems Engineering, Faculty of Engineering and Computer Science, Concordia University. Dr. Dssouli received a Master (1978), Diplome d'études Approfondies (1979), Doctorat de 3eme Cycle in Networking (1981) from Université Paul Sabatier, Toulouse, France. She earned her PhD degree in Computer Science (1987) from Université de Montréal, Canada. Her research interests are in Communication Software Engineering a sub discipline of Software Engineering. Her contributions are in Testing based on Formal Methods, Requirements Engineering, Systems Engineering, Telecommunication Service Engineering and Quality of Service. She published more than 200 papers in journals and referred conferences in her area of research. She supervised/ co-supervised more than 50 graduate students among them 20 PhD students. Dr. Dssouli is the founding Director of Concordia Institute for Information and Systems Engineering (CIISE) June 2002. The Institute hosts now more than 550 graduate students and 20 faculty members, 4 master programs, and a PhD program.

ALRAMZANA NUJUM NAWAZ received the IT Engineering Graduate degree from the Cochin University of Science & Technology, India, and the MSc in IT management in 2017 from College of IT, UAE University. She has been working as a Research Assistant in UAE University, College of IT, since 2012. She has more than 15 years' experience in development with the latest technologies including Web services and Mobile development. Her research interests include eHealth, Big data, data mining, and mobile computing.

Author's photos

HADEEL T. ELKASSABI

DR. M. ADEL SERHANI

DR. RACHIDA DSSOULI

ALRAMZANA NUJUM NAVAZ

ACCEPTED MANUSCRIPT