2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing

A Generic QoS Framework for Cloud Workflow Systems

Xiao Liu [*], Yun Yang[1] [**,*] , Dong Yuan [*], Gaofeng Zhang [*], Wenhao Li [*], Dahai Cao [*]

[*] *Faculty of Information and Communication Technologies, Swinburne University of Technology*
Hawthorn, Melbourne, VIC 3122, Australia
[**] *School of Computer Science, Anhui University*
Hefei, Anhui 230039, China
{xliu, yyang, dyuan, gzhang, wli, dcao}@swin.edu.au

Abstract

Due to the dynamic nature of cloud computing, how to achieve satisfactory QoS (Quality of Service) in cloud workflow systems becomes a challenge. Meanwhile, since QoS requirements have many dimensions, a unified system design for different QoS management components is required to reduce the system complexity and software development cost. Therefore, this paper proposes a generic QoS framework for cloud workflow systems. Covering the major stages of a workflow lifecycle, the framework consists of four components, viz. QoS requirement specification, QoS-aware service selection, QoS consistency monitoring and QoS violation handling. While there are many QoS dimensions, this paper illustrates a concrete performance framework as a case study and briefly touches others. We also demonstrate the system implementation and evaluate the effectiveness of the performance framework in our cloud workflow system.

Keywords: *Software Framework, Quality of Service, Workflow System, Cloud Computing*

1. Introduction

Cloud computing is the latest market-oriented computing paradigm [4, 12, 25]. Cloud is characterised by “XaaS”, i.e. everything as a service, typically, SaaS (software as a service), PaaS (platform as a service) and IaaS (infrastructure as a service) [12, 20]. In general, cloud services refer to a variety of services which can deliver computing capability on the service provider’s infrastructure via the Internet. Cloud has many advantages, e.g. the ability to dynamically scale to meet changing system demand; pay-per-use and separation of maintenance duties for cloud services (including software and infrastructure) to reduce capital investment; and the increased scale of resources to facilitate economics of scale, resource sharing and energy saving [4, 10]. More importantly, cloud computing enables new innovative business opportunities for IT (Information Technology) companies such as software development companies and infrastructure service providers (e.g. Google, Microsoft, IBM and Amazon). The world cloud market is estimated to be currently worth more that US$20 billion and is expected to grow rapidly in the upcoming few years [1].

Cloud computing based workflow systems, or cloud workflow systems for short, can be used as platform services (or middleware services) to facilitate the usage of cloud services. With the rapid growth of the cloud market, there will be an increasing number of cloud service providers and large numbers of cloud services. Therefore, cloud platform services which provide programming like environments for the access and composition of cloud services, will play a significant role in the development and running of software applications in the cloud [12]. However, most consumers in the cloud market are not IT professionals (e.g. scientists and businesspersons in non-IT areas), and usually they do not have sufficient knowledge for sophisticated programming. Cloud workflow systems can relieve them from traditional programming with visual modelling tools (and additional help of light-weight scripting language in some cases) [2]. Therefore, cloud workflow systems can significantly enhance the usability of cloud services, which is the key to promote the benefit of the cloud market. Given the advantages of cloud computing, we can envisage that with the rapid growth of the cloud market, the cloud workflow system would become a type of major platform for the design, development and running of software applications in the cloud. Currently, there are a few existing grid workflow systems investigating the migration from grid to cloud such as SwinDeW-C [20], Pegasus in the cloud (http://pegasus.isi.edu/pegasus_cloud.php) and CloudBus (http://www.cloudbus.org/). However, the research on cloud workflow systems is still at an early stage.

At present, given the dynamic nature of cloud computing, a challenging issue is how to deliver cloud workflow applications with satisfactory QoS (quality of service) [7, 8], which is of great importance to both service consumers and service providers. Specifically, low QoS may result in the failures of software execution and investment loss of service consumers; meanwhile, low QoS deteriorates the reputation of service providers in the cloud market and may result in the risk of out-of-business. QoS requirements can have many dimensions. Generally speaking, the major QoS dimensions for cloud software

[1] Corresponding author

services may include performance, reliability, security, availability, accessibility, fidelity, integrity, cost and many others [13, 26].

There are a number of existing studies and projects investigate the support of specific QoS requirements in different software systems. However, due to the large differences between these QoS dimensions in nature, conventional software systems often adopt different sets of software components for different QoS dimensions. Therefore, when a new QoS dimension needs to be supported by the system, a set of new software components will be developed and running independently. However, if there is no unified framework to guide the design and development process, the system complexity as well as the software development cost can be rapidly on the rise. Therefore, start from the design of a cloud workflow system, a generic framework is required to integrate and manage the software components for the support of different QoS dimensions. In this paper, we do not intend to cover the detailed strategies for all QoS dimensions, which is neither possible nor necessary. The focus of this paper is a generic framework which can facilitate the support of different QoS dimensions in cloud workflow systems. Meanwhile, in a cloud workflow system, a workflow instance needs to undergo several stages before its completion, specifically, the modelling stage, the instantiation stage and the execution stage [20]. Evidently, satisfactory QoS cannot be achieved with the sole effort in any single stage, but an effective lifecycle QoS support. To this end, the capabilities of generic software components which are required to realise lifecycle QoS support for cloud workflow applications should be identified in the first place. After that, strategies and algorithms for specific QoS requirements can be designed, implemented and managed by the generic framework as a vehicle for workflow QoS management.

This paper proposes a generic framework for lifecycle QoS support in cloud workflow systems. Covering the whole lifecycle of a workflow instance from modelling, instantiation, to execution, the generic framework consists of four components, viz. QoS requirement specification, QoS-aware service selection, QoS consistency monitoring and QoS violation handling. Based on such a generic framework, the software components for different QoS dimensions can be produced by a unified design and development process. For example, as will be detailed in Section 4, a concrete performance framework for workflow response time is composed of four software components, viz. temporal constraint setting, temporal-aware service selection, temporal consistency monitoring and temporal violation handling.

The remainder of the paper is organised as follows. Section 2 presents the related work and problem analysis. Section 3 proposes a generic framework. Section 4 illustrates a case study on a concrete performance framework for workflow response time, and demonstrates its system implementation and evaluation results. Finally, Section 5 addresses the conclusions and points out the future work.

2. Related Work and Problem Analysis

2.1 Related Work

For the development of distributed or Internet based software systems, the quality of Web services is a fundamental issue [11]. For a single Web service, its quality can often be measured from the following aspects (QoS dimensions): availability, accessibility, integrity, performance, reliability, regulatory and security [13]. Although these QoS dimensions are widely accepted as the major aspects about the quality of Web services, the definition for each QoS dimension can be perceived very differently by different stakeholders [21]. Meanwhile, for specific applications, the software design and development process may also focus on different QoS dimensions. For example, in general, the QoS requirement on performance is on the time-related aspects of Web services. It may contain many sub-level constraints such as response time, execution time, on-time completion rate and throughput (the number of service requests served in a given period of time) [13]. Accordingly, there will be different sets of measurements and strategies to specify and support the requirement on performance.

Unlike traditional workflow systems which mainly invoke and execute software components using their own local software repository, cloud workflow systems utilise software services in the cloud which are accessed through the Internet and executed at the service provider's infrastructure [4, 25]. In the cloud, a cloud workflow is actually a cloud software application composed of many cloud software services. Therefore, the quality of a cloud workflow application is determined by the collective behaviours of all the cloud software services employed by the workflow application. Similar to that of single Web services, the work in [26] concludes that the most common QoS requirements of (distributed/grid based) workflow applications include time (performance), cost, fidelity, reliability and security.

Conventional studies on QoS management mainly focus on QoS requirement specification and QoS-based service selection [11]. Besides workflow systems, many works are produced in the area of Web service composition, which is to some extend, very close to that of building cloud workflow applications with partially ordered cloud software services. The work in [14] presents an approach for build-time aggregation and evaluation of QoS dimensions of a composite service. The aggregation is performed based on abstract patterns according to an extensive collection of workflow control structures. The work in [15] presents two approaches for evaluating the QoS requirement specifications of composite services with the aim to maximise user preferences. QoS-based service selection is usually implemented by QoS-aware service brokers. The work in [24] presents a QoS broker based architecture for efficient Web services selection. While for the general QoS support, QoS-aware software architecture is required from the system design and development perspective. The work in [3] presents a design approach that facilitates software architectures in deciding on suitable software architectures under the consideration of multiple QoS attributes and potentially conflicting stakeholder preferences. There are also some works which target at QoS support for Web services and/or Web service based workflows at different stages of a Web service and/or workflow lifecycle, such as quality prediction [9], quality monitoring [22] and service replanning (dynamic service composition) for system changes [5].

2.2 Problem Analysis

The problem to be addressed in this paper is "*to propose a generic QoS framework which can facilitate a unified process to design and develop software components for lifecycle QoS support in cloud workflow systems*".

In the cloud, the service quality is very uncertain. Many potential problems such as network connection, system overload, failures of software or infrastructures at the service provider side, could all significantly deteriorate the quality of software services. Moreover, besides these technical problems, in the market-oriented cloud environments, service quality will also be affected by economic factors such as the service prices and customer loyalty levels. It is very likely that service providers would offer different levels of service quality such as performance according to different prices. Meanwhile, in order to maximise their benefits, service providers will always try to optimise their resources allocations in a dynamic fashion. In such a case, the quality of a single cloud service would normally not be consistent but subject to changes, sometimes in an imperceptible way.

A cloud workflow instance may consist of many partially ordered cloud services, and probably from a number of different service providers. Therefore, given the uncertainty lies in every cloud service, the quality of a cloud workflow instance becomes a much more complex combinatorial problem. To deliver high quality outputs for cloud workflow systems, or in other words, to ensure the quality of every cloud workflow instance becomes a very challenging issue. It is evident that high QoS cannot be easily achieved without a comprehensively designed framework to support the lifecycle of cloud workflow instances. Meanwhile, given many existing QoS dimensions and also the trend for even more dimensions due to individual consumer requirements in the future cloud market, the framework needs to be generic so that it can facilitate a unified design and development process for software components for different QoS dimensions. Based on such a generic framework, we can envisage that the quality of cloud workflow instances will be assured, and the cost for the development and management of these software components can be reduced.

3. A Generic QoS Framework

In this section, we present a generic QoS framework for lifecycle QoS support in cloud workflow systems. Before the introduction of the framework, we first take a look at the lifecycle of a typical workflow instance. Generally speaking, the lifecycle of a typical workflow instance consists of three major stages, viz. the modelling stage, the instantiation stage and the execution stage [2].

1) At the modelling stage, real world e-business or e-science processes are modelled or redesigned as cloud workflow specifications [10, 17] which may contain the process structures, task definitions for a number of workflow activities, and non-functional QoS requirements such as performance, reliability and security [24]. Based on the cloud workflow specifications, cloud workflow service providers will negotiate with their consumers to settle the service contracts which further determine such as objectives, prices and penalties.

2) At the instantiation stage, based on the service contracts, cloud workflow systems will search for candidate cloud software services which satisfy both functional and non-functional QoS requirements to fulfil the execution of workflow activities. After all the required software services are selected and reserved, cloud workflow instances are ready for execution.

3) At the execution stage, cloud workflow execution engines will coordinate the data and control flows according to the workflow specifications obtained at the modelling stage and employ the candidate software services reserved at the instantiation stage to execute all the workflow activities. Besides, in dynamic system environments, necessary workflow runtime management such as monitoring and exception handling mechanisms will ensure the detection and recovery of functional and QoS violations so that service contracts can be successfully fulfilled.

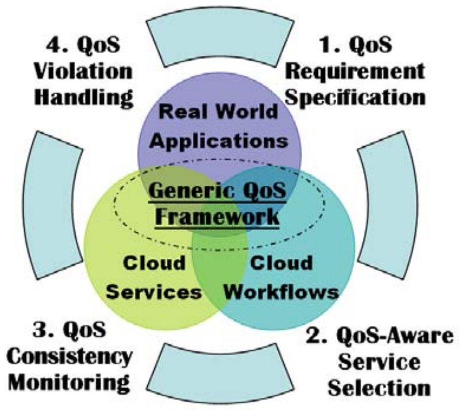
4. QoS Violation Handling
1. QoS Requirement Specification
Real World Applications
Generic QoS Framework
Cloud Services
Cloud Workflows
3. QoS Consistency Monitoring
2. QoS-Aware Service Selection

Figure 1. A Generic QoS Framework

Our generic QoS framework is depicted in Figure 1. Based on the three major stages of a workflow instance lifecycle, the framework consists of four components as shown in the outer cycle of Figure 1,, viz. QoS requirement specification, QoS-aware service selection, QoS consistency monitoring and QoS violation handling, which are implemented in a consecutive order to provide lifecycle QoS support for cloud workflow instances. The three inner cycles stand for the important factors involved in the design and application of cloud workflow systems, viz. real world applications, cloud workflows and cloud services. All the basic requirements for cloud workflows come from real world applications (e.g. business and scientific processes). Real world applications need to be abstracted by workflow service consumers with the support of workflow modelling tools and then create cloud workflow specifications. After these specifications are submitted to the cloud workflow engines, instances of these cloud workflow specifications, or cloud workflow instances for short, are executed with the underlying cloud services such as software and infrastructure services, which are normally with very dynamic service quality.

1) ***QoS requirement specification***. The first component of our generic framework is QoS requirement specification at the modelling stage. The specification of QoS requirements is a very important part of the whole workflow specification which may consist of process structures, task definitions, functional and non-functional (QoS) requirements. In general, QoS requirements can be specified in the form of either quantitative or qualitative QoS constraints.

An entire workflow instance is made of many individual workflow activities. Accordingly, there are both workflow-level QoS constraints (coarse-grained QoS constraints) and activity-level constraints (fine-grained QoS constraints). In practice, cloud workflow service consumers usually only prefer to assign a few coarse-grained QoS constraints, e.g. one deadline and several milestones for the requirement on workflow execution time. However, for service selection and monitoring purposes, fine-grained QoS constraints for individual workflow activities are necessary. Therefore, specific approaches are required to propagate a number of fine-grained constraints based on several consumer specified coarse-grained ones. Note that most coarse-grained constraints and fine-grained constraints may not be in a simple linear accumulation relationship (except cost perhaps), i.e. the sum of fine-grained constraints for a group of workflow activities is not necessarily equal to their coarse-grained constraint. For example, the deadline for a workflow segment only applies to the completion time of the last workflow activities in workflow segment, while the sum of the execution time for all the workflow activities will normally be well over the deadline, especially when there are many parallel paths in the workflow process. Therefore, sophisticated setting approaches need to be designed to address the propagation of fine-grained constraints for different QoS requirements, and ensure the consistency between the coarse-grained and fine-grained constraints. Here, the consistency means that if every individual workflow activities can satisfy its fine-grained QoS constraint, then the whole coarse-grained QoS constraint can be satisfied, and vice versa.

Furthermore, since at the workflow modelling stage, the execution states of workflow instances at runtime are uncertain such as which execution path is taken in a choice structure and which execution path takes the longest execution time in a parallel structure. Meanwhile, the quality of the available cloud service is also uncertain. Therefore, some probabilistic and forecasting strategies are also required to facilitate the setting of QoS constraints.

2) ***QoS-aware service selection***. The second component of our generic framework is the QoS-aware service selection at the instantiation stage. Given the task definition and functional requirement for each workflow activity, cloud workflow systems can usually search for and obtain many available software services in the cloud. However, these software services will be further selected based on non-functional requirements, specifically, the fine-grained QoS constraints assigned for each task by the first component. Clearly, only those software services which have higher quality then the QoS constraints can be selected by this service selection component.

Since there may be more than one QoS dimensions, the selected software services should be able to satisfy all of them. But sometimes, if this is not possible, some trade-off could be made but in a best-effort way to meet most of them. Therefore, some ranking functions should be designed to evaluate and rank the available software services. Furthermore, given some QoS requirements such as reliability and availability, the component will probably select multiple software services from different service providers at this stage to ensure there is some redundancy, namely backup services, to handle the discrepancy during runtime workflow execution. Clearly, redundancy means extra cost. Therefore, at this stage, only one primary software service (e.g. the one with highest rank) will be selected and reserved. As for other backup services, the component will only keep their information without actual reservation. However, some service providers such as Amazon provide discounted prices for reserved services (e.g. Amazon EC2 Reserved Instances or Spot Instances, http://aws.amazon.com/ec2/), hence it is also possible to book for some reserved services in advance.

3) ***QoS consistency monitoring***. The third component in our generic framework is QoS consistency monitoring at the execution stage. QoS consistency monitoring starts from the very beginning of cloud workflow execution. Here, QoS consistency means that the real service quality at the execution stage is consistent with the QoS constraints assigned at the modelling stage. Due to the dynamic nature of cloud computing, workflow execution states need to be kept under constant monitoring and QoS verification. Here, QoS verification is to check the workflow execution states against QoS constraints.

The verification for the quality of a single service is very intuitive, i.e. a simple comparison between QoS constraint and the runtime quality measurement. However, the problem becomes non-trivial for monitoring large-scale cloud workflow instances. First, in order to monitor, i.e. conduct QoS verification at anytime of the execution of a large-scale workflow instance, some probabilistic and forecasting strategies are required to estimate the quality of software services for those non-commenced workflow tasks. This is very similar to QoS constraint setting in the first component but with the access to runtime information. Second, the most intuitive way to conduct QoS verification is to check QoS consistency at every workflow task, so that if every individual software service satisfies its QoS constraint, the quality of the entire workflow instance can be achieved. Clearly, this is very inefficient and probably results in a rapid increase on the cost for QoS management. However, if we have the runtime information or knowledge that some of the software services are of very stable quality (such as produced by very reputable service providers, or having very satisfactory records for the latest workflow instances), the QoS verification can usually be skipped safely. Therefore, we can choose to conduct QoS verification only at some selected activity points, which can be named as QoS checkpoints, to save the cost. However, different QoS dimensions will probably require different information and have different rules for selecting QoS checkpoints. The selection results for different QoS dimensions will need to be compared and compromised in order to make efficient and effective selection.

4) ***QoS violation handling***. The last component of our generic framework is QoS violation handling at the execution stage. When the violation of QoS constraint is detected by the monitoring component, some recovery actions should be taken to handle and try to bring the workflow execution state back to consistency. QoS violation handing is very different from the conventional exception handling of software functional failures in traditional software systems. For example, in traditional software systems, when functional failures happen, the system state can be safely rolled back to its last checkpoint and restart the execution. Such a rollback-and-restart process can be repeated until functional failures are solved. However, as for non-functional QoS violations, this general strategy will often be useless. For example, if temporal violations are detected, i.e. there are some execution delays, the rollback-and-restart strategy cannot compensate the delays but may make the situation even worse. Actually, the time delays can only be compensated by the subsequent non-commenced workflow tasks. If we can reduce the execution time of the subsequent workflow tasks by such as recruiting additional resources or workflow rescheduling (by allocating the activities to fast resources or reducing their queuing time). In such a case, the existing time delays can be compensated and the workflow execution state can be brought back to consistency. Meanwhile, for other QoS requirements such as reliability and security, if violations have been detected at the current checkpoint, the handling process is to minimise the loss while take proactive actions to prevent these violations from happening again in the future.

Generally speaking, for QoS violation handing, firstly, we should try to minimise the existing loss, and secondly (or actually more importantly), we should prevent these violations in the subsequent workflow as much as possible.

4. Case Study: A Performance Framework

Based on our generic framework proposed in Section 3, this section presents a concrete performance framework as a case study to illustrate how it can be applied to specific QoS dimensions. The system implementation and some of its experimental results are also demonstrated.

4.1 A Performance Framework

The performance framework focuses on the workflow performance, or more specifically, it focuses on the response time of workflow applications. By following our generic QoS framework, the performance framework consists of four components which can provide a lifecycle support for high performance in cloud workflow systems.

1) ***Temporal constraint setting***. The first component is temporal constraint setting which assigns both coarse-grained and fine-grained temporal constraints in cloud workflow specifications at the modelling stage. The setting of high quality temporal constraints is very important to the successful on-time completion of cloud workflows.

In our performance framework, temporal constraint setting is realised through a three-step process. The first step is a forecasting process where the workflow activity duration intervals are predicted by a time-series forecasting strategy. The second step is a win-win negotiation process between service consumers and service providers to specify the coarse-grained temporal constraints. The third step is a propagation process where fine-grained temporal constraints are set automatically based on the results of the second step.

Due to the page limit, the detailed strategy and algorithms for the temporal constraint setting component can be found in [17] and hence omitted here.

2) ***Temporal-aware service selection***. The second component is temporal-aware service selection which selects and reserves suitable cloud software services for individual workflow tasks. For temporal constraints alone, the service selection will probably only consider the processing power of the software services such as the speed of the CPU units and the size of the memory spaces. However, in the real world, the service selection process often needs to consider other QoS constraints such as reliability and security at the same time. All these QoS constraints serve as critical criteria for the selection of cloud services and resource management in cloud workflow systems [11].

In our framework, temporal-aware service selection is realised basically through the existing functions of the broker service in the cloud workflow system. The broker service is in charge of discovering available software services from the software service catalogues in the cloud. The conventional broker service may select software services only based on functional requirements, and in some case, one or two primary QoS constraints such as performance and cost. In such a case, when the initial set of available services is returned, our QoS-aware service selection component will further select the candidate services based on the combinatorial QoS constraints. Moreover, it may select one software service as the primary candidate and several others as backup candidates. After that, the candidate sets will be returned to the broker service. The broker service will settle the service contracts with external cloud service providers.

More introduction about the temporal-aware service selection component and broker services can be found in [25] and [24] respectively, and hence omitted here.

3) ***Temporal consistency monitoring***. The third component is temporal consistency monitoring. Based on a temporal consistency model, the temporal consistency states of cloud workflows should be under constant monitoring in order to detect potential temporal violations in a timely fashion. However, as mentioned in our generic framework, the accumulated cost for temporal verification can be very huge in large-scale cloud workflow instances. Therefore, cost-effective strategies need to be designed to detect potential temporal violations in an efficient fashion.

In our performance framework, the function of temporal consistency state monitoring is realised through a two-step process. The first step is temporal checkpoint selection. Given the probability based temporal consistency model, our minimum probability time redundancy based checkpoint selection strategy can choose the minimal set of activity points (i.e. necessary and sufficient checkpoints) for temporal verification. Here, necessity means that only those activity points where real temporal inconsistency states take place are selected and sufficiency means that there are no any omitted activity points. The second process is temporal verification which checks the current temporal consistency states at selected checkpoints with our probability based temporal consistency model. In our performance framework, only two types of temporal consistency states (viz. recoverable and non-recoverable) are defined [16]. Accordingly, only one type of temporal checkpoint and one type of temporal verification are required to determine the current temporal consistency state.

Due to the page limit, the detailed strategy and algorithms for the temporal consistency monitoring can be found in [19] and hence omitted here.

4) ***Temporal violation handling***. The last component is temporal violation handling which deals with recovery of temporal violations. Based on the results of the previous component for monitoring temporal consistency, a necessary and sufficient checkpoint is selected which means a potential temporal violation is detected. When a temporal violation is detected, temporal violation handling strategies should be executed. In our performance framework, we mainly focus on those statistically recoverable temporal violations [18] which can be recovered by light-weight temporal violation handling strategies. For such a purpose, representative metaheuristics based workflow rescheduling strategies are investigated, adapted and implemented under a novel general two-stage local workflow rescheduling strategy to handle temporal violations. Since our temporal violation handling strategy is fully automatic and only utilises existing system resources without recruiting additional ones, the cost of temporal violation handling can be significantly reduced compared to heavy-weight temporal violation handling strategies such as recruiting additional resources and modifying workflow structures [23].

In our performance framework, we have defined three levels of recoverable temporal violations, viz. level I, level, II and level III violations, and designed their corresponding handling strategies, viz. TDA (time deficit allocation), ACOWR (ant colony optimisation based two-stage local workflow rescheduling strategy) and TDA+ACOWR (the combined strategy of TDA and ACOWR). ACOWR is the major strategy which attempts to compensate the time deficits with the reduced workflow execution time through optimising the workflow scheduling plan. Here, "two-stage" means a two-stage searching process designed in our strategy to strike a balance between time deficit compensation and the completion time of other activities while "local" means the rescheduling of "local" workflow segments with "local" resources. Our temporal violation handling strategy only utilises existing resources which are currently deployed in the system instead of recruiting additional resources. Meanwhile, unlike global rescheduling which modifies the global task-resource list for the entire workflow instance, our strategy only focuses on the local workflow segment and optimises the integrated Task-Resource list.

Due to the page limit, the detailed strategy and algorithms for the temporal violation handling component can be found in [18] and hence omitted here.

In the subsequent two sub-sections, we first present the system implementation of our performance framework in our SwinDeW-C cloud workflow system. Afterwards, some experimental results to evaluate its effectiveness are demonstrated. More details about the system environment and simulation experiments can be found online[2].

4.2 System Implementation

SwinDeW-C (**Swin**burne **De**centralised **W**orkflow for **C**loud) [20] is a prototype cloud workflow system running on SwinCloud, a cloud computing test bed. SwinCloud comprises of many distributed computing nodes. Our performance framework is implemented as workflow QoS management tools in SwinDeW-C. Specifically, the temporal constraint setting component and temporal-aware service selection component are mainly interacted with the workflow modelling tool and the broker service. The temporal constraint setting component generates the temporal constraints, which are part of the workflow specification and the basic input data for temporal-aware service selection. The temporal-aware service selection component returns the candidate (best and backup) software services satisfying the temporal constraints through the broker service which is composed of many tool agents. The tool agents are in charge of the data communication between the workflow system and the software services which are either available in the system local repository or delivered by external service providers in the cloud market.

After a workflow specification is submitted to the workflow enactment service, an instance of the workflow

[2]http://www.ict.swin.edu.au/personal/xliu/doc/QoSFramework.rar

execution engine is created to coordinate the data and control flow, and execute the workflow instance by invoking software services which are managed by the tool agents. During runtime, the workflow execution state is constantly monitored by the temporal consistency monitoring component. The workflow execution state can be displayed by a watch list which contains runtime information such as time submitted, time finished, percentage of completion, service status and many other real-time and associated statistics. When the temporal violations are detected, alert messages are sent to the temporal violation handling component which further analyses the workflow execution state and the violated temporal constraints to determine the levels of temporal violations and execute the corresponding violation handling strategies through the interactions with the workflow execution engine and tool agents.

4.3 Experimental Results

The effectiveness of the performance framework is tested with various sizes of cloud workflows and under different environment settings. Due to the page limit, we only demonstrate the global violation rate (the violation of the final deadline) for the workflow instances. The detailed evaluation results for those four components can be found online (see footnote 2).

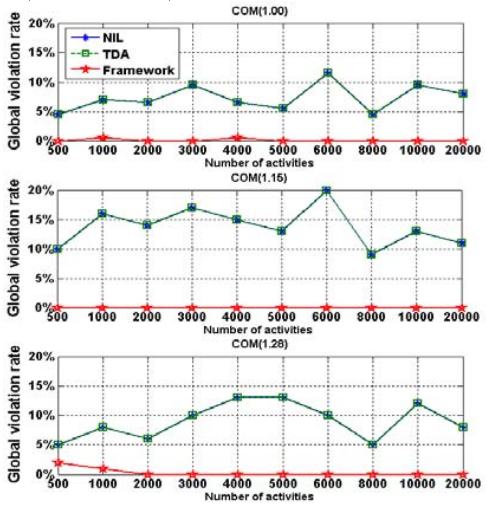

Figure 2. Global Temporal Violation Rate

In our experiment, the size of cloud workflow instances ranges from 500 to 20,000 where the activity durations are generated by normal distribution models with the same settings as for workflow rescheduling. For every group of 20 to 50 activities, an upper bound temporal constraint is assigned. The strategy for setting temporal constraint is adopted from the work in [17] where a normal percentile is used to specify temporal constraints and denotes the expected probability for on-time completion. Here, we conduct three rounds of independent experiments where the temporal constraints are set with three representative normal percentiles of 1.00, 1.15 and 1.28 which denotes the probability of 84.1%, 87.5% and 90.0% for on-time completion (from non-smooth to smooth situations) without any handling strategies on temporal violations (denoted as COM(1.00), COM(1.15) and COM(1.28)). For the comparison purpose, we record global violation rates under natural situations, i.e. without any handling strategies (denoted as NIL), and compared with that of TDA strategy [6] and our three-level handling strategy as introduced in Section 4.1 (denoted as Framework where the three violation handling strategies are implemented automatically according to the levels of temporal violations [18]). In TDA+ACOWR, the maximum iteration times for ACOWR are set as 2. Each round of experiment is executed for 100 times to get the average violation rates.

Figure 2 shows the results on global violation rates. As analysed in [19], since TDA can only delay the temporal violations without actual compensating the time deficits, it may handle local violations but has no effect on global violations. Therefore, the global violation rates for NIL and TDA are overlapping. The global violation rates for NIL and TDA behave very unstably but increase roughly with the number of activities while decreasing with the value of normal percentiles. The average global violation rates for NIL and TDA in each round are 14.6%, 13.8% and 9.0% respectively. With our performance framework, the global violation rate is kept close to zero since most local temporal violations are handled automatically along workflow executions. The average global violation rates of our performance framework in each round are 0.2%, 0.0% and 0.3% respectively, i.e. an overall average of 0.167%. Accordingly, the reductions in the global violation rates are 14.4%, 13.8% and 8.7% respectively, with an average of 12.3%, which is very significant. Therefore, with the implementation of our concrete performance framework, the global violation rates of workflow instances can be significantly reduced, which effectively verifies its satisfactory effectiveness in the support of temporal QoS in cloud workflow systems.

4.4 Discussions

The above case study shows the effectiveness of the performance framework as a concrete example to verify the design of our generic QoS framework. As for many other QoS dimensions, concrete frameworks can be built in the same manner when specific design for each component is available. Since the research for cloud workflow systems is still at an early stage, other concrete frameworks are yet to be built with the future research outcomes. Currently, besides performance, we are making progress on reliability and security for data storage services. Based on our current implementation of the performance framework, the software components for other QoS dimensions are being designed and developed under a unified process. Meanwhile, by sharing the same data repository for real-time and historic data, the support and optimisation for multiple QoS constraints can be incorporated efficiently when specific components are incorporated like add-ins to the performance framework. Therefore, we can claim that our generic QoS framework can ensure the delivery of high QoS over the cloud workflow lifecycle, and reduce the complexity of system design and development.

5. Conclusions and Future work

Due to the dynamic nature of cloud computing, the quality of cloud software services has aroused great concerns of both cloud service providers and service consumers. Especially, since there lacks a generic framework to provide general guidelines for the design and development of software components, how to effectively facilitate lifecycle QoS (Quality of Service) support for cloud workflow applications becomes a challenge. In this paper, to address such an issue, we have proposed a generic framework which consists of four major components, viz. QoS requirement specification, QoS-aware service selection, QoS consistency monitoring and QoS violation handling. As a specific realisation of our generic framework, a concrete performance framework, together with its strategies, system implementation and experimental results, has been demonstrated in detail.

In the future, based on the on-going research, concrete strategies for the software components of many other QoS dimensions will be designed and developed following our generic QoS framework. Furthermore, through the framework, the data communication and knowledge sharing between the components for different QoS dimensions will be further enhanced for solving complex problems such as multi-QoS based service selection, monitoring and violation handling.

Acknowledgements

The research work reported in this paper is partly supported by Australian Research Council under Linkage Project LP0990393 and Discovery Project DP110101340, as well as State Administration of Foreign Experts Affairs of China Project P201100001.

REFERENCES

[1] Australian Academy oF Techonolgy Science and Engineering, *CLOUD COMPUTING: Opportunities and Challenges for Australia*. available at: http://www.atse.org.au/component/remository/ATSE-Reports/Information-Technology/CLOUD-COMPUTING-Opportunities-and-Challenges-for-Australia-2010/, accessed on 1st July 2011.

[2] W. M. P. v. d. Aalst and K. M. V. Hee, *Workflow Management: Models, Methods, and Systems*: The MIT Press, Cambridge, 2002.

[3] T. Al-Naeem, I. Gorton, M. A. Babar, F. Rabhi, and B. Benatallah, "A Quality-Driven Systematic Approach for Architecting Distributed Software Applications," *Proc. 27th International Conference on Software Engineering*, pp. 244-253, 2005.

[4] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," *Future Generation Computer Systems,* vol. 25, pp. 599-616, 2009.

[5] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani, "QoS-Aware Replanning of Composite Web Services," *Proc. 2005 IEEE International Conference on Web Services*, pp. 121-129, 2005.

[6] J. Chen and Y. Yang, "Multiple States based Temporal Consistency for Dynamic Verification of Fixed-time Constraints in Grid Workflow Systems," *Concurrency and Computation: Practice and Experience, Wiley,* vol. 19, pp. 965-982, 2007.

[7] J. Chen and Y. Yang, "A Taxonomy of Grid Workflow Verification and Validation," *Concurrency and Computation: Practice and Experience,* vol. 20, pp. 347-360, 2008.

[8] W. N. Chen and J. Zhang, "An Ant Colony Optimization Approach to a Grid Workflow Scheduling Problem With Various QoS Requirements," *IEEE Trans. on Systems, Man, and Cybernetics, Part C: Applications and Reviews,* vol. 39, pp. 29-43, 2009.

[9] L. Cheung, R. Roshandel, N. Medvidovic, and L. Golubchik, "Early Prediction of Software Component Reliability," *Proc. 30th International Conference on Software Engineering*, pp. 111-120, 2008.

[10] European Commission, *The Future of Cloud Computing, Opportunities for European Cloud Computing Beyond 2010*, available at: http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf, accessed on 1st July 2011.

[11] T. Erl, *SOA: Principles of Service Design*: Prentice Hall, 2008.

[12] I. Foster, Z. Yong, I. Raicu, and S. Lu, "Cloud Computing and Grid Computing 360-Degree Compared," *Proc. 2008 Grid Computing Environments Workshop,* pp. 1-10, 2008.

[13] IBM, *Understanding Quality of Service for Web Services*. available at: http://www.ibm.com/developerworks/library/ws-quality.html, accessed on 1st July 2011.

[14] M. C. Jaeger, G. Rojec-Goldmann, and G. Muhl, "QoS Aggregation for Web Service Composition Using Workflow Patterns," *Proc. 8th IEEE International Conference on Enterprise Distributed Object Computing,* pp. 149-159, 2004.

[15] Z. Liangzhao, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware Middleware for Web Services Composition," *IEEE Trans. on Software Engineering,* vol. 30, pp. 311-327, 2004.

[16] X. Liu, J. Chen, Z. Wu, Z. Ni, D. Yuan, and Y. Yang, "Handling Recoverable Temporal Violations in Scientific Workflow Systems: A Workflow Rescheduling Based Strategy," *Proc. 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid10)*, pp. 534-537, 2010.

[17] X. Liu, J. Chen, and Y. Yang, "A Probabilistic Strategy for Setting Temporal Constraints in Scientific Workflows," *Proc. 6th International Conference on Business Process Management*, pp. 180-195, 2008.

[18] X. Liu, Z. Ni, Z. Wu, D. Yuan, J. Chen, and Y. Yang, "A Novel General Framework for Automatic and Cost-Effective Handling of Recoverable Temporal Violations in Scientific Workflow Systems," *Journal of Systems and Software*, vol. 84, no. 3, pp. 354-376, 2011.

[19] X. Liu, Y. Yang, Y. Jiang, and J. Chen, "Preventing Temporal Violations in Scientific Workflows: Where and How," *IEEE Trans. on Software Engineering,* in press, http://doi.ieeecomputersociety.org/10.1109/TSE.2010.99. 2010.

[20] X. Liu, D. Yuan, G. Zhang, J. Chen, and Y. Yang, "SwinDeW-C: A Peer-to-Peer Based Cloud Workflow System," in *Handbook of Cloud Computing*, B. Furht and A. Escalante, Eds., Springer, pp. 309-332, 2010.

[21] D. A. Menasce, "Composing Web Services: a QoS view," *IEEE Internet Computing,* vol. 8, pp. 80-90, 2004.

[22] F. Raimondi, J. Skene, and W. Emmerich, "Efficient Online Monitoring of Web-Service SLAs," *Proc. 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pp. 170-180, 2008.

[23] N. Russell, W. M. P. van der Aalst, and A. H. M. ter Hofstede, "Workflow Exception Patterns," *Proc. 18th International Conference on Advanced Information Systems Engineering,* pp. 288-302, 2006.

[24] M. A. Serhani, R. Dssouli, A. Hafid, and H. Sahraoui, "A QoS Broker based Architecture for Efficient Web Services Selection," *Proc. 2005 IEEE International Conference on Web Services,* pp. 113-120, 2005.

[25] Z. Wu, X. Liu, Z. Ni, D. Yuan, and Y. Yang, "A Market-Oriented Hierarchical Scheduling Strategy in Cloud Workflow Systems," *Journal of Supercomputing, published online, http://www.springerlink.com/content/t7t8823l8g88l827/fulltext.pdf,* 2011.

[26] J. Yu and R. Buyya, "A Taxonomy of Workflow Management Systems for Grid Computing," *Journal of Grid Computing,* vol. 3, no 3-4, pp. 171-200, 2005.