J Supercomput
DOI 10.1007/s11227-016-1928-z

CrossMark

ATSDS: adaptive two-stage deadline-constrained workflow scheduling considering run-time circumstances in cloud computing environments

Reihaneh Khorsand[1] · **Faramarz Safi-Esfahani**[2] · **Naser Nematbakhsh**[2] · **Mehran Mohsenzade**[1]

Abstract A significant aspect of cloud computing is scheduling of a large number of real-time concurrent workflow instances. Most of the existing scheduling algorithms are designed for a single complex workflow instance. This study examined instance-intensive workflows bounded by SLA constraints, including user-defined deadlines.The scheduling method for these workflows with dynamic workloads should be able to handle changing conditions and maximize the utilization rate of the cloud resources. The study proposes an adaptive two-stage deadline-constrained scheduling (ATSDS) strategy that considers run-time circumstances of workflows in the cloud environment. The stages are workflow fragmentation and resource allocation.In the first stage, the workflows according to cloud run-time circumstances (number of Virtual Machines (VMs) and average available bandwidth) are dynamically fragmented. In the second stage, using the workflow deadline and the capacity of the VMs, the workflow fragments created are allocated to the VMs to be executed. The simulation results show improvements in terms of workflow completion time, number of messages exchanged, percentage of workflows that meet the deadline and VM usage cost compared to other approaches.

Keywords Deadline-constrained workflow · Instance-intensive workflow scheduling · Workflow fragmentation · Cloud computing · Self-adaptive systems · Fuzzy systems

✉ Reihaneh Khorsand
reihaneh_khm@yahoo.com

[1] Department of Computer Engineering, Science and Research branch, Islamic Azad University, Tehran, Iran

[2] Faculty of Computer Engineering, Najafabad branch, Islamic Azad University, Najafabad, Iran

Published online: 02 December 2016

1 Introduction

In a workflow management coalition, the term workflow denotes automation of a business process in which specific documents, information or tasks are transferred from a given participant to another to be processed according to a set of procedural rules [1]. Research in e-government and e-business applications have led to the introduction of a new type of workflow called instance-intensive workflow [2]. Unlike computation-intensive workflows in e-science, instance-intensive workflows are defined based on large numbers of concurrent and simple workflow instances. Samples of instance-intensive workflows are bank check processing, loan allocation processing, insurance claims processing, online train ticketing, and many e-government and e-business scenarios [3–5]. The present study uses a loan-application workflow to illustrate the features of instance-intensive workflows. Special circumstances can increase loan requests and online loan allocation processing systems must be able to handle very large demands for online loan allocation at the same time. Generally, a single loan allocation process is relatively simple and can be modeled as a workflow in a few relatively simple steps. When faced with a huge number of instance-intensive workflows, traditional workflow technology struggles to use existing resources to provide satisfactory services. The emergence of cloud computing provides a solution to this by allowing these workflows to access shared data and run computations on cloud resources effectively. In this situation, cloud workflow scheduling becomes a key issue.

The scheduling problem is to decide which workflow task will be run by which computing resource at what point in time [6]. Tan [7] used a distributed scheduling approach to fragment the workflow into several fragments and transmit them to their performers in related locations. Several studies have addressed the problem of scheduling in distributed systems which create workflow fragments dynamically [7,8]. To increase the proficiency of the scheduling approach, workflow fragmentation methods must adapt themselves to the run-time environment automatically. The dynamics of a cloud run-time environment are usually independent metrics and are the number of physical machines and virtual machines, average available bandwidth among VMs, workloads and deadline of each workflow [9–11]. Few studies have presented adaptive approaches thus far and none consider cloud run-time environment [2,12,13]. Their primary problems are the failure to create adaptable fragments and efficient allocation of workflow fragments to VMs. These eventually result in a high number of messages exchanged, long completion time and high number of SLA violations at run-time.

An adaptive two-stage deadline-constrained scheduling (ATSDS) strategy is proposed in the current study that considers run-time circumstances of deadline-constrained workflows in the cloud environment. The hypothesis is that feedback such as the number of VMs and bandwidth available from the run-time environment will decrease the number of messages exchanged, completion time, VM usage cost and number of SLA violations. The goal is to provide an adaptable and scalable scheduler for the cloud run-time environment. In the first stage of operation (workflow fragmentation), the workflow is dynamically fragmented by considering cloud run-time circumstances (number of VMs and average available bandwidth) to determine the adaptability of the fragments to run-time circumstances. In the second stage (resource

allocation), the workflow deadline and capacity of the VMs are considered when the new fragments are allocated to the VMs to be executed.

Section 2 provides an overview of related works. Section 3 presents the proposed ATSDS approach that considers run-time circumstances in a cloud computing environment. Section 4 provides an performance evaluation of the ATSDS strategy. Conclusions remarks and future works are discussed in Sect. 5.

2 Related work

Research related to the topic can be grouped into three general categories. The first is a type of workflow model topology. The second is workflow fragmentation methods in a distributed environment and adaptability of fragmentation. The third is prior workflow scheduling methods.

2.1 Workflow model topology

A workflow model (also called a workflow specification) is a process that formally specifies two different but complementary factors: (a) the steps involved in workflow execution, such as components, tasks, activities or services; (b) the dependencies between steps which specify the order in which the steps can be executed. The topology of a workflow model can be block based or graph based [14]. In a block-based topology, control flow is specified as a programming language, such as using block structures like "if" or "while" in C programming language. This kind of modeling is similar to that of programming languages in basic computers and leads to a simple conversion to the executable code. Additionally, flow control of workflows in graph-based programming languages is determined by explicit control connections between activities. Kopp [15] and Nguyen [16] showed that all graph-based models can be converted into block-based approaches and vice versa. The present study exploited the block-based method owing to its simplicity and resemblance to some programming languages.

2.2 Workflow fragmentation

A fragmentation method creates workflow fragments that group together some of the workflow model elements (activities, control flows, data flows, etc.) [7]. Workflow fragmentation forms the basic foundation to support workflow distributed executions, increasing scalability, reuse and outsourcing of workflow fragments [17]. Fragmentation can be done at two times under two different strategies. The first strategy is to apply compiler techniques to detect parallel and sequential activities of a workflow for fragmentation. In this approach, a workflow is pre-fragmented in a central server and then the fragments are created statically and sent to distributed servers to be executed. A workflow can be analyzed and improved by a designer or a compiler. This is the simplest method, but is not flexible because static fragments are produced at compilation time. The second strategy is run-time based and dynamically fragments the workflow at run-time. In fact, the fragments do not exist prior to this and are produced dynamically. This provides some degree of variability, but is often costly.

Different methods have been presented for workflow fragmentation, using a variety of techniques with different forms of execution. Distributed approaches are divided into two broad categories: fully- and partially fragmented workflow. A fully fragmented workflow is fully partitioned to the finest level of granularity in the activity level and the fragments are then executed by one or several servers. For example, this method was applied to partition workflows in [5,18–21]. The criteria for partial workflow fragmentation methods differ according to the workflow designer's perspective and produce coarser granularity levels [22–26]. In comparison with the fully fragmented workflow methods, partial workflow fragmentation reduces the number of fragments created, inter-fragment interaction and eventually decreases bandwidth usage. In partial workflow fragmentation, static approaches are based on the opinions of the system analyst and designer at design time [5,18,19].

Some dynamic execution methods create workflow fragments dynamically [7,8]. Tan and Fan [7] introduced a theoretical dynamic workflow fragmentation method using the Petri net. In this method, a business workflow instance could be deployed to a specific server executing a number of immediate tasks while partitioning the remaining parts and sending them to another server. This approach uses a stepwise workflow run-time environment to improve system flexibility. Their approach lacks sufficient fragmentation criteria because it is designed to transfer fragments by judging only their preconditions. Cheng and Zeng [27] introduced a static quality of service (QoS)-based method to partition a centralized workflow model into fragments. Their method uses "minimum execution cost within a deadline" (MCD) and "minimum execution time with a budget" (MTB). They presented a definition for Markov decision process-based fragment scheduling with an MTB objective in a real-time system. The advantage of this method is that it is easy to perform, but it is not flexible because fragmentation is static and compiler based. The current study dynamically fragments each business workflow according to cloud run-time circumstances using a modified version of the hierarchical fragmentation method discussed in Sect. 3.1.

2.2.1 Adaptability of fragmentation

Adaptability aims to adjust various parameters in response to changes at run-time. The main categories for workflow adaptability are on-the-fly, locate/relocate and hybrid methods [4]. In the on-the-fly approach, workflow fragments could be made dynamically at run-time or taken from a repository containing pre-compiled fragments. It can also later reconfigure them to reduce the system burden [7,10,28]. The locate/relocate approach uses workflow fragmentation and searches for the most suitable run-time agent/web service to execute a fragment. The fragments are then relocated to reduce system threshold violation based on run-time circumstances [5]. The hybrid approach combines the first two approaches to obtain more efficient run-time adaptability. To the best of our knowledge, no research has introduced and investigated workflow fragmentation based on the hybrid approach.

2.3 Summary of workflow scheduling methods

Workflow scheduling allows creation of one or more executable workflow instances by applying suitable resources. Schedulers assign the available resources to the workflow tasks/fragments [29]. Several studies [30–38] have discussed the different aspects of workflow/job scheduling and provide a historical perspective and relevant context for the present study. They explain how hardware and software can work in concert on scalable multi-processor systems using illustrative examples and applications for job scheduling. The current study addresses scheduling in the more modern context of the cloud.

Wu et al. [9] introduced a market-oriented hierarchical scheduling strategy for cloud-based scientific workflows which utilizes two basic scheduling stages. In the service level scheduling stage, service-based fragmentation is used to assign tasks-to-service in which workflow tasks are mapped to cloud services. In the task-level scheduling stage, task-to-VM assignment can be done locally in cloud data centers. The tasks of a workflow application are mapped to cloud services in global cloud markets based on their QoS requirements for run-time. Because this method produces many small groups of task-to-VM assignments, to reduce the high overhead, fragmentation methods that are capable of running large-sized workflows in a cloud environment must be provided in future research.

Zhu et al. [39] suggested a two-stage method known as a high-throughput workflow scheduling algorithm within a user-defined deadline that schedules a scientific workflow on an on-demand resource provisioning system such as the cloud. In the first stage, modules are classified into fragments by means of the polynomial-time longest path (LP) algorithm, denoted as “find critical path” (FCP). Next, the prioritized module mapping algorithm “P modules mapping” (PMM) is run to allocate the fragment to the network graph until convergence of the “end-to-end delay” (EED) is reached. In their study, fragmentation is run-time based and the authors intend to implement fragmentation and scheduling in local cloud test beds to support different scientific workflows.

Zhang et al. [10] proposed an iterative ordinal optimization (IOO) method for scientific workflow scheduling on cloud computing platforms. The authors demonstrated that the IOO method can effectively generate a suboptimal schedule for most NP-hard problems. Their approach does not require a post-design phase such as a run-time environment. Because this method produces many small groups of task-to-VM assignments, fragmentation methods that are capable of running large-sized workflows in a cloud environment must be provided in future research to overcome the high overhead.

Liu et al. [40] suggested a compromised-time-cost (CTC) scheduling algorithm which considers the characteristics of cloud computing to accommodate instance-intensive workflows. The CTC algorithm calculates the sub-deadlines for tasks of the last instance and the sub-deadlines for tasks of other instances based on the last instance by assuming that the schedule follows the stream-pipe mode. It then calculates the estimated execution time and cost of each service. It then allocates each task to the service using round-robin scheduling to provide an execution time that does not exceed the sub-deadline and has the lowest total cost.

Table 1 A summary of prior workflow scheduling methods

Literature	Time of fragmentation		Adaptability	Workflow fragmentation method	Scheduling method	Cloud based
	Compiler based	Run-time based				
[7]	–	√	On-the-fly	Based on workflow circumstances	Fragments can migrate to execution servers by mobile agents. Scheduling method is unspecified	–
[2]	–	–	–	No fragmentation	A multiple QoS constrained scheduling strategy of multi-workflows	√
[5]	√	–	Locate/ relocate	Activity driven method (Fully distributed)	It uses an agent-based BPEL centralized engine, namely NINOS. It searches for the most suitable run-time agent/web service to execute an activity	–
[28]	–	√	On-the-fly	Based on run-time circumstances	It uses WADE/JADE performer agents. Scheduling method is unspecified	–
[27]	√	–	–	Based on QoS (MCD, MTB)	Markov Decision Process-based fragment scheduling and use mobile agents in migrating workflow systems	–
[9]	–	√	–	Based on service and task	Two basic scheduling stages; namely, service level and task level in cloud-based execution VMs	√
[39]	–	√	–	Control-path- driven	A two-step scheduling algorithm to achieve end-to-end delay improvement and low overhead in cloud infrastructure	√
[10]	√	–	On-the-fly	Based on task	An iterative ordinal optimization (IOO) method for scientific workflow scheduling	√
[40]	√	–	–	Based on task	An instance-intensive cost-constraint cloud workflow scheduling algorithm named **CTC** which takes cost and time as the main concerns with user	√
[41]	√	–	–	Based on task	It classifies workflow tasks based on user's QoS preference, and combines with staggered sub-deadlines allocation criteria, proposes a **QDA** scheduling algorithm	√
ATSDS	–	√	Hybrid	HPD and based on run-time circumstances	An adaptive two-stage deadline-constrained scheduling strategy that considers the cloud run-time environment circumstances for instance-intensive workflows	√

Li et al. [41] proposed a QoS-based deadline allocation (QDA) algorithm for instance-intensive workflow task scheduling in a cloud environment. QDA references the main sub-deadline allocation criteria of the CTC algorithm, uses the QoS utility function as a service resource selection condition and takes user preferences (i.e., time, cost) into account. The present study compares the proposed ATSDS algorithm with the CTC and QDA in Sect. 4.3.

A summary of prior workflow scheduling methods is shown in Table 1. Prior research for workflow scheduling is constrained with significant limitations. Some [2,14] have no criteria for fragmentation and result in low performance. Existing workflow fragmentation solutions [7,20,21,28] are also insufficient. Because these methods improve distributed execution and scalability, they focus only on fragmentation while neglecting the scheduling component. In addition, some suffer from a lack of fragmentation adaptability or are not able to move fragments to the cloud [20,27,42]. The ATSDS algorithm was developed to address these problems in a cloud computing environment. To improve workflow scalability and achieve user-defined workflow deadlines, both fragmentation and scheduling aspects have been combined (Sect. 3).

3 Proposed approach

The ATSDS framework considers cloud run-time circumstances and its components. Figure 1 shows the position of the proposed workflow scheduler between two basic cloud computing layers. The proposed strategy is accomplished in two phases. The logical fragmentation of workflow in the software-as-a-service (SaaS) layer considers run-time circumstances delivered from the infrastructure-as-a-service (IaaS) layer and physical allocation of fragments to the VMs considers the capacity of the VM in the IaaS layer.

Figure 2 is an overall view of the proposed scheduling strategy and the scheduler that applies the strategy. From the user perspective, the workflows and the values associated with them (workflow relative deadline (RD) and absolute deadline (AD)) are submitted to a Fragmenter component. RD is the relative deadline by which the workflow instances should be completed; missing a RD is acceptable and the remaining workflow fragments will be sent to the critical list. AD is the last completion time by which the workflow instance must be completed; missing an AD is not acceptable and the remaining workflow fragments will be sent to a remove list.

A monitoring component collects the run-time circumstances (number of VMs and average available bandwidth) and forwards the information to the Fragmenter. The Fragmenter dynamically and adaptively partitions/re-partitions each workflow into fragments while considering cloud run-time circumstances. It uses a hierarchical fragmentation method called the hierarchical process decentralization (HPD) [28] that fragments the workflow at a suitable level in a workflow hierarchy. The level of fragmentation is determined at run-time considering number of available VMs and average available bandwidth among VMs. The ATSDS can apply workflow fragments in an adaptive fashion. The fragments for each user workflow are added to a waiting list to be dispatched. The full details of the operation are described in Sect. 3.2.

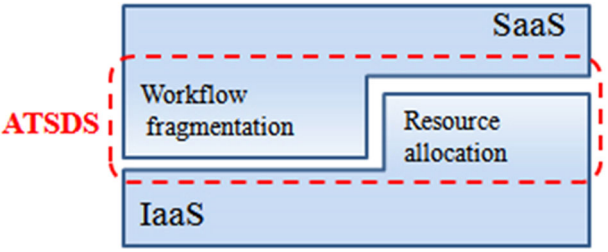

Fig. 1 Position of the proposed scheduling strategy

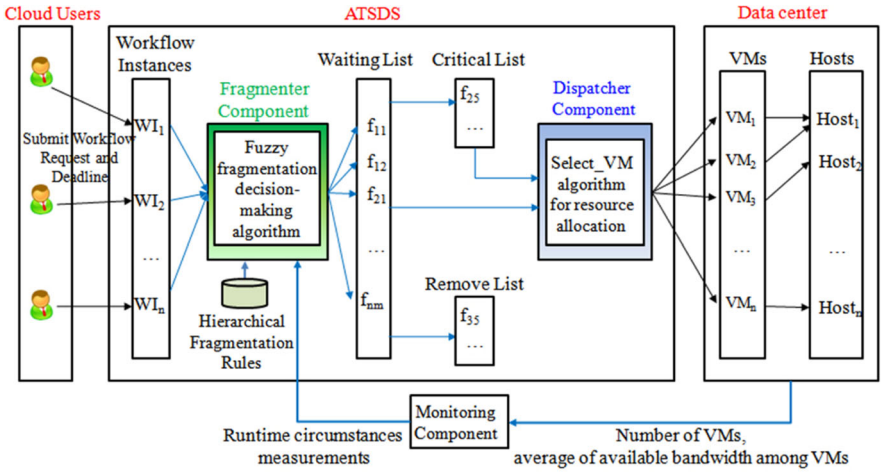

Fig. 2 Adaptive two-stage deadline-constrained scheduling (ATSDS) framework

In the IaaS layer, the second stage of the scheduling scenario contains the waiting, critical and removes lists. The algorithm that processes the waiting list sends the workflow fragments to the dispatcher considering the workflow RD. Under certain circumstances, the waiting list algorithm may send some fragments to the critical or remove lists. The dispatcher allocates the workflow fragments received to the VMs using an appropriate resource allocation method (Sect. 3.3). The scheduler continuously assesses both the waiting and critical lists as long as workflow fragments are being scheduled.

3.1 Reference workflow and hierarchical fragmentation method

Figure 3 shows the loan-application workflow used as a reference model. The loan workflow seeks information from two non-resident web services which send credit statements for the loan requester. When both web services confirm that a requester's credit is good, the loan is approved.

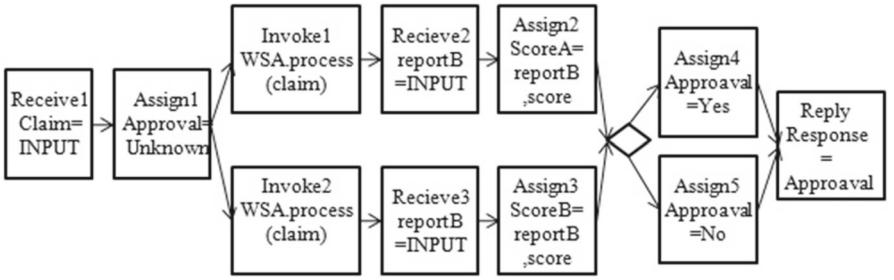

Fig. 3 BPEL view of loan-application workflow [28]

The HPD fragmentation method [28] fragments a business process based on the existing hierarchy of activities. Fragmentation for an HPD could be initiated from any level in a workflow tree using a breadth-first search/traverse algorithm. In the HPD, simple activities are denoted as leaves, while structured activities that keep the most relevant activities together in a distinct fragment are located in the middle.

Figure 4 shows the HPD of the loan application. It is accomplished in four levels with an asterisk denoting the fragments. The whole process tree for one fragment is wrapped by fragmentation at level 0 (HPD0), as for the centralized approach. The subsequent levels of fragmentation are HPD1, HPD2, and HPD3; however, HPD3 is the equivalent of the fully distributed method. It is worth mentioning that at run-time, all fragmentation levels exchanges inputs and outputs between the dependent fragments in a workflow instance as messages. Also, the independent fragments could be executed by different VMs in parallel.

3.2 First stage of scheduling scenario: workflow fragmentation

As described earlier, the fragmentation problem shapes the workflow fragments using a fragmentation method. In distributed systems, this is the duty of the Fragmenter. The Fragmenter determines which method of fragmentation is commensurate with run-time circumstances (left side of Fig. 2). In the present study, the input parameters of the Fragmenter are workflow specification, number of VMs, and average available bandwidth of the VMs. The workflow fragments are the outputs of this component. Two adaptability aspects have been introduced: fragment-to-VM proportionality that is the ratio of workflow fragments to the number of VMs dedicated to the scheduling system and fragment-to-bandwidth adaptability that is fragmentation commensurate with the currently available bandwidth. The Fragmenter is first established on the fragment proportionality aspect and makes decisions on the fragmentation level that are commensurate with the current number of VMs. It then applies available-bandwidth adaptability to offer a suitable fragmentation set that is commensurate with the percentage of available bandwidth. The bandwidth parameter is affected by the environment and may change at any time. Fuzzy logic was used to generate precise solutions from certain or approximate information to model the bandwidth behavior between VMs (Sects. 3.2.1, 3.2.2).

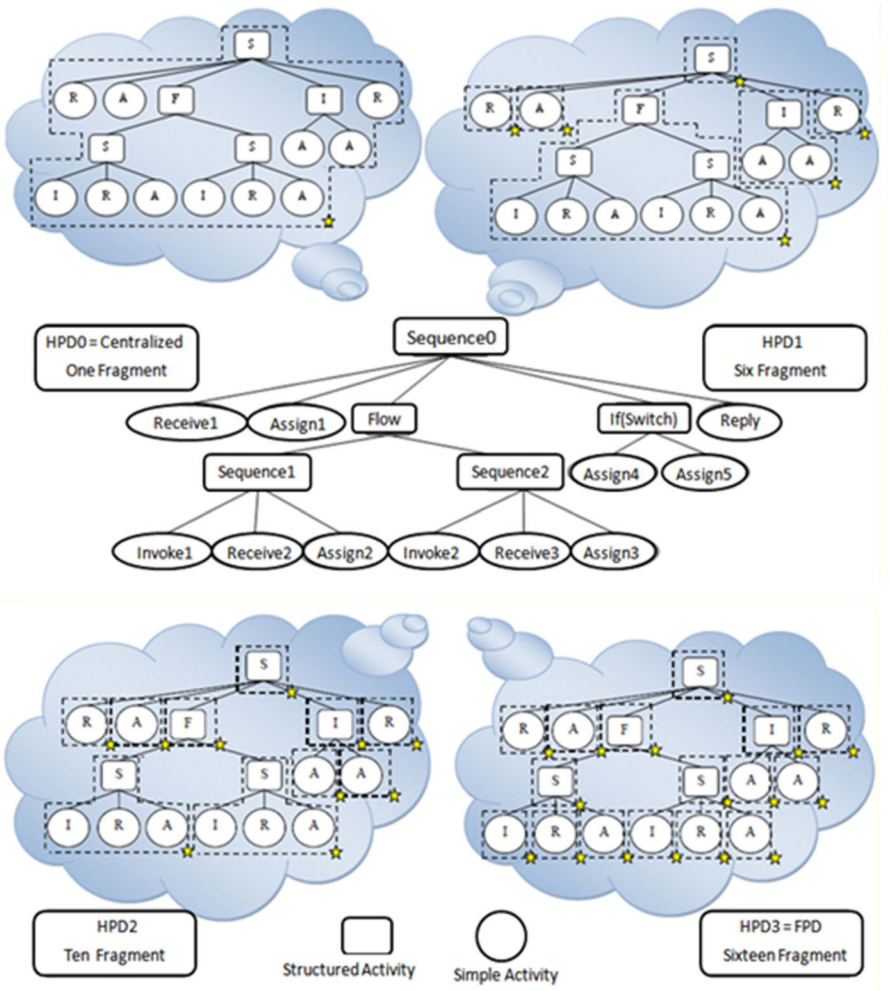

Fig. 4 HPD fragmentation of loan-application business workflow [28]

3.2.1 Fuzzy fragmentation decision-making algorithm

Adaptability is achieved using a modified version of a hierarchical workflow fragmentation method (also called HPD [28]) on a reference loan workflow (Sect. 3.1). Algorithm 1 presents the fuzzy fragmentation decision-making pseudo-code along with its input and output parameters (lines 1–2). The workflow model is first fragmented by the method *fragmentSetArray* for the HPD approach (line 4). The fragment set contains the HPD fragments. The first element is analogous to the centralized fragment set and includes only one member. The last index is also equal to the fully distributed fragment set.

Algorithm 1 Fuzzy fragmentation decision-making

```
1.  Input: wm (Workflow Model), bw (Bandwidth), novm (Number of Virtual Machines)
2.  Output: granularity Level
3.  Begin
4.  fsaHPD = fragmentSetArray (wm, "HPD")
5.  fp = findFragmentProportionality (fsaHPD, novm)
6.  fpl = findFragmentProportionalityLevel (fp)
7.  granularityLevel = findFuzzyGranularity (fsaHPD, fpl, bw)
9.  End
```

The HPD fragmentation of the loan application provides four fragment sets which include 1 (centralized), 6, 10 and 16 fragments (fully distributed approach). After obtaining fragment set *fsaHPD*, fragment proportionality is determined for HPD fragment sets using *findFragmentProportionality*, which determines the number of fragments in which the fragmentation level is closest to the number of VMs (line 5). The number of the fragmentation level is returned by *findFragmentProportionalityLevel* (line 6). Once all the fragments, the available bandwidth and the fragment proportionality level have been obtained, a suitable fragmentation level is returned by *findFuzzyGranularity* (line 7). Algorithm 2 shows how fragment proportionality is calculated. It determines which number of fragments is closest to the number of available VMs (lines 4–10). *FindFragmentProportionalityLevel* has the same functionality as Algorithm 2. The only difference is that the algorithm returns the number of the levels in the process tree which produce a suitable number of fragments. This number is closer to the number of available VMs than the other levels. For example, in an empty communication medium with 7 VMs, the fragmenter produced 6 fragments equal to the HPD1 approach of the loan application.

Algorithm 2 findFragmentProportionality

```
1.  Input: fsa (Fragment Set Array),
    novm (Number of VM)
2.  Output: Fragment Proportionality
3.  begin
4.  distance = novm
5.  fp = 0
6.  For (0≤i< fsa. Length)
7.  if (|numberOfFragments (fsa[i]) –novm|) < distance)
8.  distance = |numberOfFragments (fsa[i]) –novm|
9.  fp = numberOfFragments (fsa[i])
10. Return distance
11. End
```

3.2.2 Fuzzy granularity level calculation

Fuzzy logic is a many-valued logic used for computing in place of the usual (one or zero) Boolean logic [43]. With fuzzy logic, different types of rules can be implemented. Before these rules are applied, all input signals must be converted into linguistic variables. This step is called fuzzification. Different membership functions (or triangular functions) are used to do this. After the transformation into linguistic variables, the inference rules can be applied, but defuzzification is then needed to generate a sharp output value. The current study uses fuzzy logic to model bandwidth behavior between VMs.

To calculate suitable granularity in workflow fragmentation, *findFuzzyGranularity* is used as shown in Algorithm 3. This method receives the fragment set array (*fsa*), fragment proportionality level (*fpl*) and available bandwidth (*bw*) as input and returns the granularity level as the output (lines 1–2). First, the bandwidth is segmented dynamically and then using fuzzy variable *bandwidth*, the *bw* is fuzzified to *fbw* using a singleton function (lines 4–5). For each segmented bandwidth S_i, a new rule such as $S_i \rightarrow Singleton\ (fsa[i].fragmentNo())$ is created by *makeFuzzyRules* (line 6). The rules created are executed using a rule engine supporting Sugeno rules (line 7). The calculated fuzzy granularity level *fGranularity* is later defuzzified using a weighted average function and a crisp value is calculated by applying a defuzzification method (line 8).

The crisp value *fFragments* determines the number of fragments required and then a suitable process-tree level number is returned by *findFragmentProportionality* which is the final result of *findFuzzyGranularity* (lines 9–10). For example, in an empty communication medium with 7 VMs, the Fragmenter produces 6 fragments equal to the HPD1 approach of the loan application, but reduces the fragmentation level in the high-traffic network to HPD0 having lower average bandwidth and creates coarser fragments to achieve adaptive behavior. At the end of this stage, the related fragments of different workflows are sent to the waiting list as outputs of the Fragmenter.

Algorithm 3 Find Fuzzy granularity

```
1.  Input: fsa (Fragment Set Array),
        fpl (Fragment Proportionality level),
          bw (available bandwidth)
2.  Output: granularity level
3.  begin
4.  FuzzyVariable    bandwidth = dynamicBandwidthSegmentation (fsa, fpl+1)
5.  fbw = fuzzify (bandwidth, Singleton (bw))
6.  FuzzyRuleSet = makeFuzzyRules (bandwidth, fpl, fbw)
7.  fGranularity = sugenoRuleExecution (FuzzyRulesSet)
8.  fFragments = defuzzify (fGranularity)
9.  granularity Level = findFragmentProportionality (fsa, fFragments)
10.   Return granularity Level
11. End
```

3.3 Second stage of scheduling scenario: resource allocation

3.3.1 Waiting list processing algorithm

A common problem exists in previous deadline-constrained workflow scheduling algorithms [44–47]. Most use existing performance estimation techniques [46] such as analytical modeling [48] and empirical [49] and historical data [50] to determine the time required to run fragments on the available VMs. Because these methods are based on estimation, they are not sufficiently accurate in operating environments. The current study executes the finish time of a current workflow fragment inside a workflow instance as shown in Fig. 5. Note that this figure is only for an illustration and does not reflect the result of a real execution.

The algorithm that processes the waiting list decides whether or not a workflow instance deadline can be met at any point in the run-time according to the finish time. As shown in algorithm 4, a traverse operation can be performed for every fragment on the waiting list. If the finish time of the current workflow fragment is larger than the absolute deadline, it means the workflow execution could not be completed by the deadline and is analogous to a deadline violation; thus, the fragment will be sent to a remove list (lines 2–5). At the end of each execution round, the status of the fragment is changed to failed on the remove list and the related violation variable is set to true. After declaring the status as output, the fragment is removed from the waiting list (lines 6–8).

As the traverse operation continues, the waiting list processing algorithm examines the current time of the fragment. If it is between the fragment relative deadline and absolute deadline, it should be considered as a critical fragment and sent to the critical list. If there is a fragment in the critical list, it is executed at higher priority than the non-critical fragments (lines 9–11). If none of the previous cases has occurred, i.e.,the finish time of the fragment is not larger than the absolute deadline and the fragment is not critical, then the fragment will be sent to the dispatcher component to be executed normally (lines 12–13).

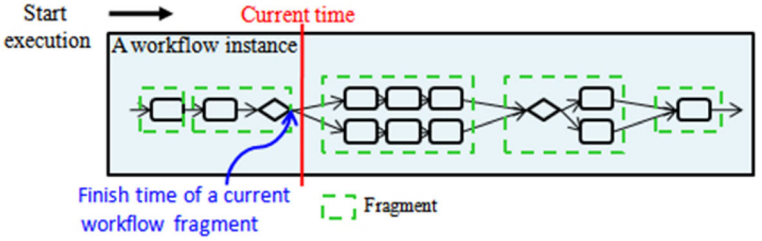

Fig. 5 Finish time of a current workflow fragment for a workflow instance

Algorithm 4 Waiting list processing algorithm

```
1   begin
2   While (WaitingList.size != 0){
3   if (fragment.get FinishTime () > fragment.get Absolute Deadline ())
4           Fragment.set_Violation of deadline(true)
5           Remove List.add (fragment)
6   for(fragment itm: RemoveList){
7         if   (fragmentStatus==FAILED)
8           Waiting List. Removes (fragment)
9   if(fragment.get FinishTime () < getAbsoluteDeadline () && fragment.get FinishTime () >
    getRelativeDeadline ()){
10          Critical List.put (fragment)
11          Submit fragment to the Dispatcher
12  else
13        Submit next fragment to the Dispatcher
14  End
```

3.3.2 Resource allocation

QoS parameters of cloud services vary, but the current study mainly considered execution time T, execution cost C, memory M and storage capacity $S$to describe the QoS of the VMs in what can be described as a 4-tuple $Qs = \{T, C, M, S\}$. $T$and $C$are negative attributes concerning minimization of QoS and $M$and S are positive attributes concerning maximization of QoS. These four attributes were used to calculate the QoS utility function value of the VMs. Positive attributes can be multiplied by -1 to convert them to negative attributes. QoS utility function U(VM) is used to map the QoS attribute vector Qs= {q1(VM), q2(VM), …, qr(VM)} of each candidate VM to a real value. Formula (1) shows how to evaluate $U(VM)$ for each VM:

$$\mathrm{U}(\mathrm{VM}) = \sum_{k=1}^{r} \frac{Q_{j,k}^{\max} - q_k(VM)}{Q_{j,k}^{\max} - Q_{j,k}^{\min}} \times W_k \tag{1}$$

where r is the number of VM QoS attributes, q_k (VM) is the kth QoS attribute of each VM, W_k is user preference, $Q_{j,k}^{\max}$ and $Q_{j,k}^{\mathrm{Min}}$ are maximum and minimum values, respectively, of the kth QoS attribute of all candidate VMs. In accordance with the type of user preference, the workflow system can use the corresponding W_k vector to evaluate the VM QoS utility function using Formula (1). For example, the details of Formula (1) for VM1 are shown in Formula (2) as:

$$\mathrm{U}_{\mathrm{VM1}} = \frac{(\mathrm{MaxMips}_{\mathrm{VM}} - \mathrm{Mips}_{\mathrm{VM1}})}{\mathrm{MaxMips}_{\mathrm{VM}} - \mathrm{MinMips}_{\mathrm{VM}}} * W_{k\,\mathrm{Mips}} + \frac{(\mathrm{MaxCost}_{\mathrm{VM}} - \mathrm{Cost}_{\mathrm{VM1}})}{\mathrm{MaxCost}_{\mathrm{VM}} - \mathrm{MinCost}_{\mathrm{VM}}} * W_{k\mathrm{Cost}} + \frac{(\mathrm{MaxM}_{\mathrm{VM}} - M_{\mathrm{VM1}})}{\mathrm{MaxM}_{\mathrm{VM}} - \mathrm{MinM}_{\mathrm{VM}}} * W_{kM}$$

$$+\frac{(\mathrm{MaxS_{VM}} - S_{\mathrm{VM1}})}{\mathrm{MaxS_{VM}} - \mathrm{MinS_{VM}}} * W_{kS} \tag{2}$$

The VMs usage cost can be calculated using Formula (3) as:

$$\mathrm{Cost} = P_{\mathrm{cost}} \times \frac{\text{Length of workflow fragments}}{\mathrm{Mips}} \tag{3}$$

where P_{cost} is the cost per CPU unit incurred using the VM.

In summary, each workflow application is partitioned into fragments by the Fragmenter (Sect. 3.2). The Fragmenter then submits fragments to the waiting list. The waiting list processing algorithm submits the next fragment to the dispatcher (Sect. 3.3.1). The dispatcher then distributes the workflow fragments to the VMs by sorting their QoS utility function values in ascending order.

4 Evaluations

The loan workflow is used as a business workflow to examine the effectiveness of the proposed ATSDS strategy. The experimental setup and quantitative metrics evaluated are presented and the ATSDS strategy is studied during four experiments.

4.1 Experimental setup

The cloud simulations and experiments were carried out using the CloudSim 3.0.3 toolkit [51] with ten hosts inspired by the reference [10]. The configuration of the experiment environment from the toolkit is shown in Fig. 6. The loan workflow and the HPD fragmentation levels are written in Java. A cloud user sends requests at specific rates of 100, 500, and 1000 per minute. On producing a cloud user request, a bandwidth number (0–100) is exponentially generated that simulates the percentage of network bandwidth available between the VMs. Fragmentation is done by the Fragmenter separately for each workflow instance by considering the HPD fragments and feedback data from the run-time environment. Different workflow fragmentation levels could be used on-the-fly at run-time. After the workflow fragments are produced, they are deployed on available VMs using the ATSDS resource allocation method. In the simulation, each host is equipped with an Intel Xeon processor and 2048 MB of memory. The physical host runs with Linux OS. Different VMs are installed on each physical host. Each VM is equipped with one core processor with 512 MB of memory. MIPS (million instructions per second), the processing capability of each VM, is different for each VM. The value of MIPS for VMs ranges from 250 to 10,000 with a linear rising function ($y = (\text{MIPS of current VM}) \times i/5 + 1$).

4.2 Metrics for evaluating proposed algorithms

The metrics used to evaluate the proposed scheduling strategy are the completion time of the workflow, bandwidth usage based on the number of messages exchanged, deadline violation and VM usage cost. The completion time of the makespan is usually

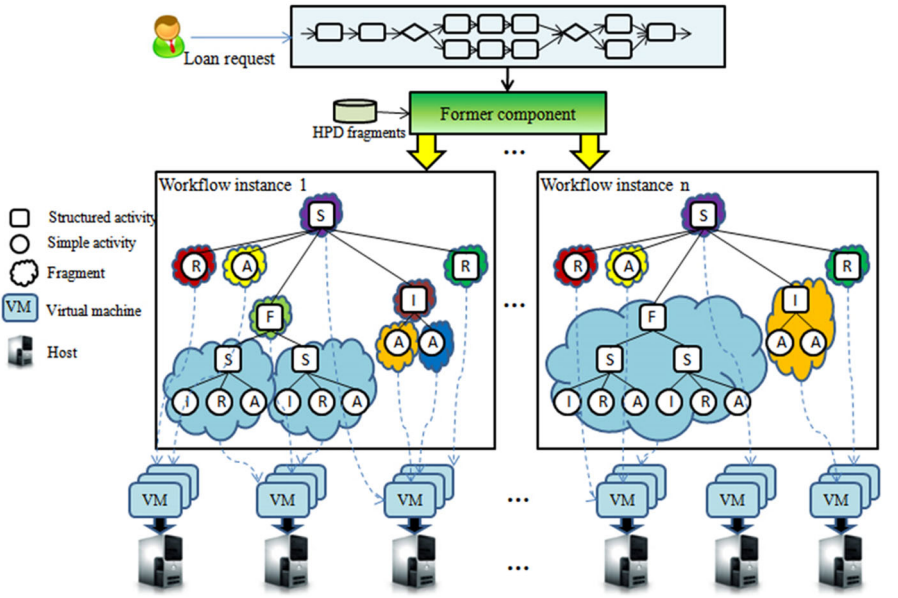

Fig. 6 Experiment environment configuration

the total time from the start of the workflow until all tasks are completed and the outputs are completely produced [12,52]. Because ATSDS applies available-bandwidth adaptability that offers a suitable fragmentation set commensurate with the percentage of bandwidth available, it is important to determine the average available bandwidth for the VMs during workflow execution. The available bandwidth is determined by counting the number of messages exchanged having the same size. The number of messages passed will increase bandwidth usage [28]. The deadline violation metric is defined to set deadlines for workflows. Real-time workflow is highly affected by the deadline, which means it must be executed within a time constraint. Even a slight delay may cause a deadline violation. The VM usage cost can be calculated using Formula (3) as described in Sect. 3.3.2.

4.3 Evaluation experiments

Table 2 shows that experiments 1–3 consist of two parts (a and b) that evaluate the performance of the fragmentation algorithm. Moreover, the overall ATSDS strategy is evaluated in experiment 4.

4.3.1 Experiment 1 (evaluating ATSDS available-bandwidth adaptability against Fully distributed fragmentation along with HPD fragmentation, exponential bandwidth simulator, variable request arrivals, variable VM count)

The proposed ATSDS strategy was equipped with an exponential bandwidth generator and was tested in the presence of 1, 7, 14, and 21 VMs. A cloud user sent requests

Table 2 Experiments configuration

Experiment	Level of workflow fragmentation in the first stage of scheduling scenario	Resource allocation method in the second stage of scheduling scenario	VM count	Number of parallel workflow request arrivals per minute	Size of message	RD & AD	Evaluated metric
1(a)	Variable based on run-time circumstances	Constant (ATSDS)	Variable (1, 7, 14, 21)	Variable (100, 500, 1000)	Constant (15)	–	Reaction of ATSDSto exponential bandwidth
1(b)	Variable based on run-time circumstances for ATSDS and Fully distributed	Constant (ATSDS)	Variable (1, 7, 14, 21)	Variable (100, 500, 1000)	Constant (15)	–	Number of exchanged messages
2	Variable based on VM's count	Constant (ATSDS)	Variable (1, 7, 14, 21)	Constant (100)	Variable (0, 100, 200, 300)	–	Completion time
3(a)	Variable based on run-time ctrcumstances for ATSDS and Fully distributed	Constant (ATSDS)	Constant (63)	Variable (100, 500, 1000)	Constant (300)		Completion time
3(b)	Variable based on run- time circumstances for ATSDS and Fully distributed	Constant (ATSDS)	Variable (7, 21, 63, 189)	Constant (500)	Constant (300)		Completion time
4(a)	Variable based on run-time ctrcumstances for ATSDS and Fully distributed for other approaches	Variable (ATSDS, CTC, QDA)	Constant (21)	Variable (100, 200, 400, 600, 800, 1000)	Constant (15)	10, 13	Number of deadline violations, and the VM usage cost
4(b)	Variable based on run-time ctrcumstances for ATSDS and Fully disttibuted for other approaches	Variable (ATSDS, CTC, QDA)	Constant (21)	Variable (100, 200, 400, 600, 800, 1000)	Constant (15)	15, 18	Number of deadline violations
4(c)	variable based on run-time ctrcumstances for ATSDS and Fully distributed for other approaches	Variable (ATSDS, CTC, QDA)	Variable (1, 7, 14, 21)	Constant (500)	Constant (15)	15, 18	Number of deadline violations and the VM usage cost

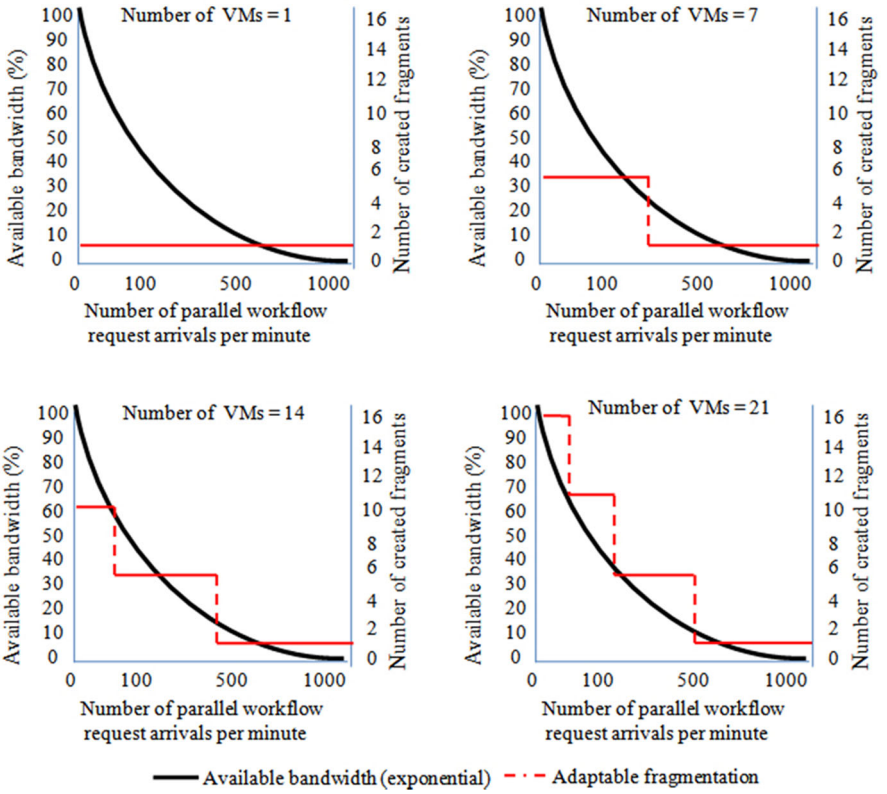

Fig. 7 (Experiment 1(a)): reaction of ATSDS algorithm to exponential bandwidth, HPD fragmentation

to servers at request rates of 100, 500, and 1000 requests per minute. Figure 7 plots the trend of bandwidth change from the left axis and the number of fragments created from the right axis of the graphs. During the experiment, bandwidth was simulated with an exponential bandwidth generator and the number of messages exchanged was counted with applying the fuzzy fragmentation decision-making algorithm. Figure 8 shows that use of the fuzzy fragmentation decision-making algorithm with different numbers of VMs resulted in a decrease in bandwidth usage.

For simplicity, each experiment was denoted according to the following convention: method name +_+ number of VMs. In this experiment, the Fragmenter considered fragment proportionality and consistently adapted the fragmentation of workflows to the available bandwidth. Because the bandwidth followed the exponential distribution pattern, the Fragmenter directed workflow fragmentation from more decentralized fragments to their more centralized equivalents. Despite the increase in the request rate, the number of messages exchanged improved in ATSDS about 93.33, 30 and 46.66% over the fully distributed approach. ATSDS reduced the scalability in a high-traffic network and created coarser fragments, but produced softer fragments in an empty communication medium.

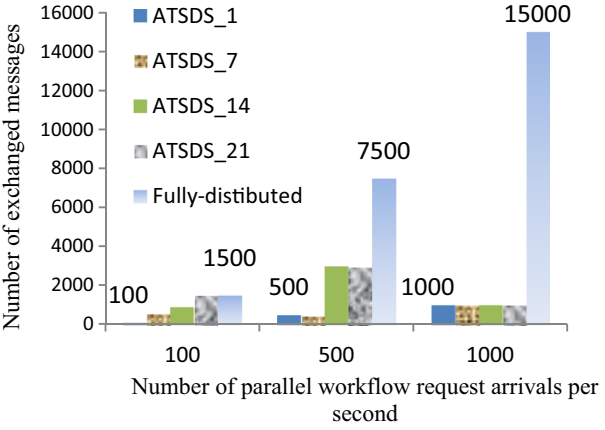

Fig. 8 (Experiment 1 (b)): number of exchanged message comparison between different levels of fragmentation in ATSDS

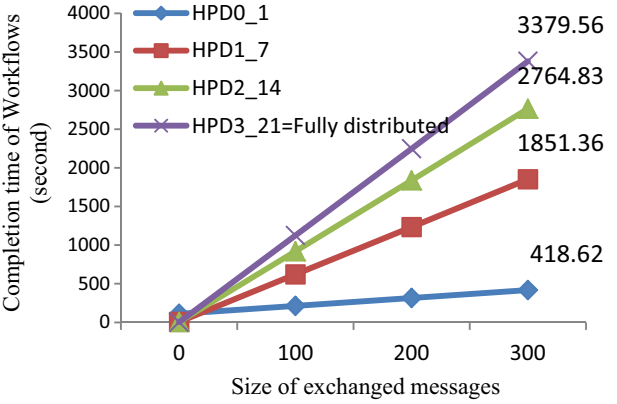

Fig. 9 (Experiment 2): completion time parameter of two workflow scheduling methods plotted against constant workflow request arrivals and the variable-message size

4.3.2 Experiment 2 (evaluating the effect of message size on completion time, constant resource allocation method, constant request rate, variable message size)

Completion time can be affected by the request rate and the size of the messages exchanged between workflow activities. Messages 0, 100, 200 and 300 kb in size were applied. To avoid the effect of request rate, the experiment was executed at a constant request rate of 100 requests per minute. Figure 9 depicts the decrease in completion time of the fully distributed fragments by other loan fragments. At the 100 request rate, the average decrease in completion time by HPD0, HPD1 and HIPD2 was 87.61, 45.21 and 18.18%, respectively. The number of fragments and size of the messages exchanged among workflow activities were effective in this experiment. The fully distributed approach improved when the sizes of the messages increased.

4.3.3 *Experiment 3 (evaluating the completion time parameter of loan workflow instances with different level of fragmentation, constant resource allocation method)*

The completion-time parameter of loan workflow instances was evaluated for the different fragmentation algorithms using the ATSDS and fully distributed execution approaches. In the fully distributed approach, the authors statically partitioned a business workflow to activity level fragments as described in Sect. 2.2 [5].

Figure 10 shows the completion-time parameter of loan workflow instances for the different fragmentation algorithms. The ATSDS approach was equipped with a exponential bandwidth simulator. Figure 10a shows the effect of an increase in the number of workflow instances on the workflow completion-time parameter. A client sent requests to servers at request rates of 100, 500, and 1000 requests per minute. The results show that ATSDS decreased the completion time 16.33% compared to the fully distributed approach as the number of workflow instances varied. Note that ATSDS used the maximum fragmentation level (HPD3) in an empty communication medium (when the amount of request arrivals increased from 1 to 100; Fig. 10a). It produced softer fragments similar to the fully distributed approach, but reduced the fragmentation level in a high-traffic network with a lower average bandwidth and created coarser fragments to achieve adaptive behavior.

To test the scalability and the adaptability of ATSDS, the number of VMs were varied from 7 to 21, 63, and 189 as suggested by Zhang [10]. Figure 10b shows the effects of the number of VMs on workflow completion time. When the VM number increased from 7 to 63, the completion time for both approaches decreased. When it increased from 63 to 189, neither approach changed. It can be concluded that: (1) the workflow completion time will decrease when the VMs increase and will reach a point where further addition of VMs fail to decrease the completion time; (2) a VM number could be configured for the best execution; (3) an increase in the number of VMs up to 63 decreased completion time 43.98% for ATSDS over the fully distributed approach. A method for automatically determining how many VMs is optimal for the best execution must be a subject of future research.

4.3.4 *Experiment 4 (evaluating the deadline violation and the VM usage cost parameters of the loan workflow instances, variable request arrivals, constant level of workflow fragmentation, variable scheduling method)*

Experiment 4 evaluated the performance of the algorithms used in the first and second stages of ATSDS. The proposed strategy was compared with the CTC [40] and QDA [41] scheduling algorithms (in Sect. 2.3). The CTC algorithm only considers cost and the QDA algorithm considers both time and cost. ATSDS takes various aspects of QoS into account. Figure 11a, b shows the effect of an increase in the number of workflow instances up to 1000 per minute for the loan application for deadline violation and VM usage cost, respectively, with different relative and absolute deadline times. In Fig. 11a, the deadline violation and the VM usage cost ATSDS for QDA were 38.67 and 43.75% and for CTC were 16.32 and 0% with an increase in the number of workflow instances. As the number of workflow instances increased, the number of VMs was

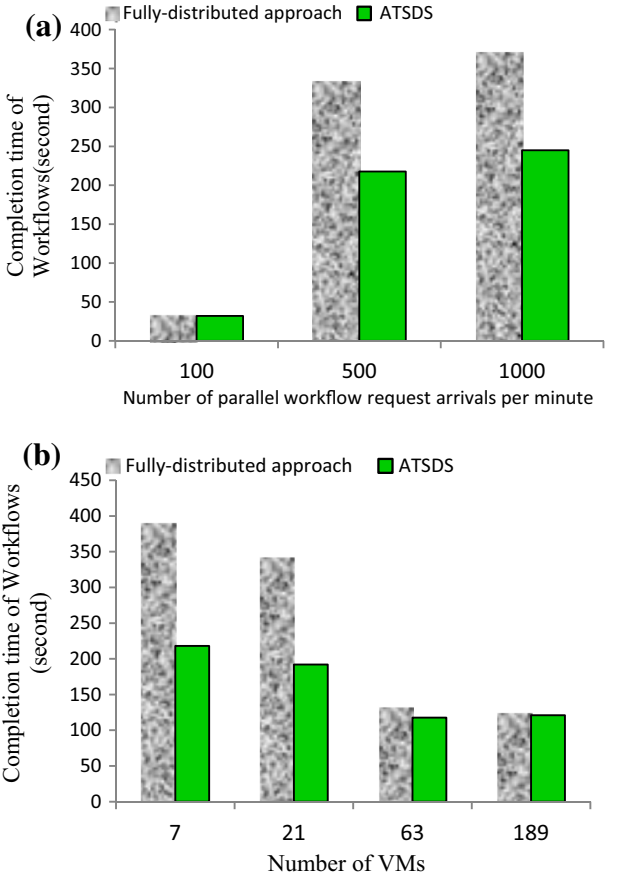

Fig. 10 Completion time parameter of two workflow scheduling methods plotted against number of parallel workflow request arrivals and the number of VMs. **a** (Experiment 3 (a)) Effect of workflow instance number for 63 VMs. **b** (Experiment 3 (b)) Effect of VM number for 500 workflow instance

fixed, meaning deadline violations increased and the performance of all policies in meeting the deadline dropped off.

The VM usage cost of CTC was lower than for QDA and the VM usage cost for ATSDS was almost identical to the CTC for users who prefer the lowest cost. Figure 11b shows that at different relative and absolute deadline times, the deadline violation for ATSDS was 34.25% better than QDA and 50.74% better than CTC with an increase in the number of workflow instances. CTC used the round-robin allocation algorithm, which does not consider the capacity of the VMs and loads the fragments in a circular form. This is why it showed a less favorable value compared to ATSDS in all cases. Because resource allocation in ATSDS operates according to execution time, execution cost, memory, and storage capacity of VMs, it has a greater effect on decreasing the percentage of deadline violations for ATSDS policies compared to QDA and CTC policies for a high number of workflow instances.

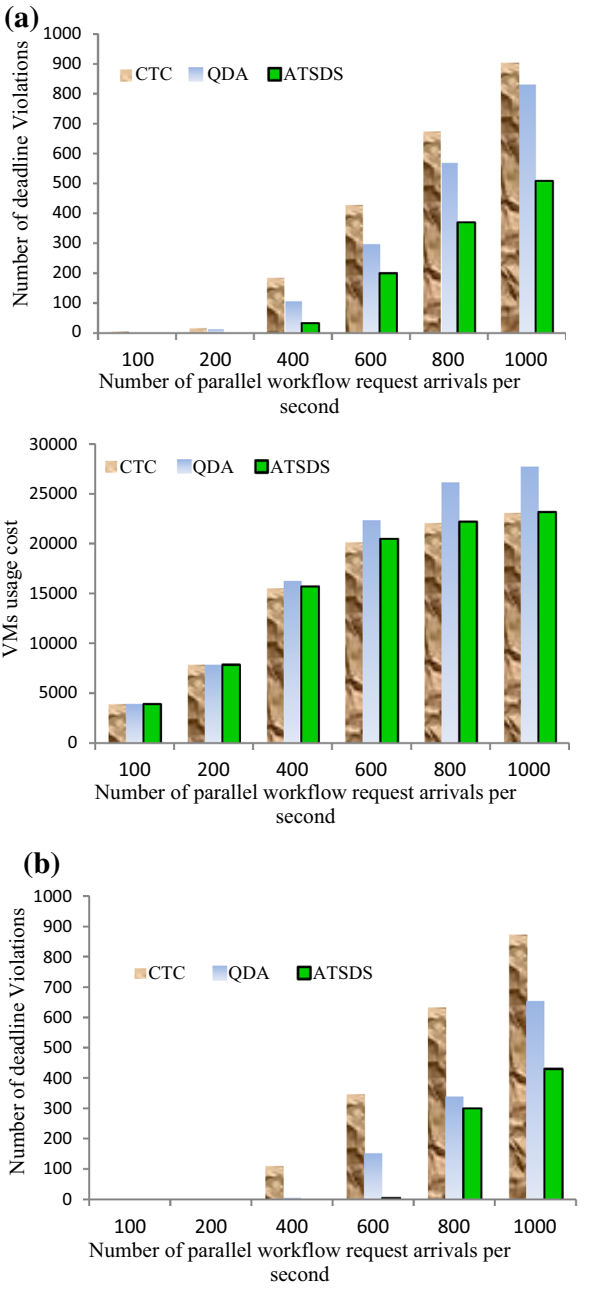

Fig. 11 Comparison diagram of number of deadline violations and the VM usage cost for 1000 workflow instances among the QDA versus CTC versus proposed ATSDS algorithms, VM count = 21. **a** (Experiment 4 (a)-part 1): evaluating the deadline violation parameter, relative deadline time = 10, absolute deadline = 13. **b** (Experiment 4 (a)-part 2): evaluating the VM usage cost parameter, relative deadline time = 10, absolute deadline = 13. **c** (Experiment 4 (b)): evaluating the deadline violation parameter, relative deadline time = 15, absolute deadline = 18

To verify the effect of VM number, the simulation compared deadline violations and VMs usage cost for constant workflow request arrivals. The results are shown in Fig. 12. As seen, the deadline violations and VM usage cost for ATSDS increased 0

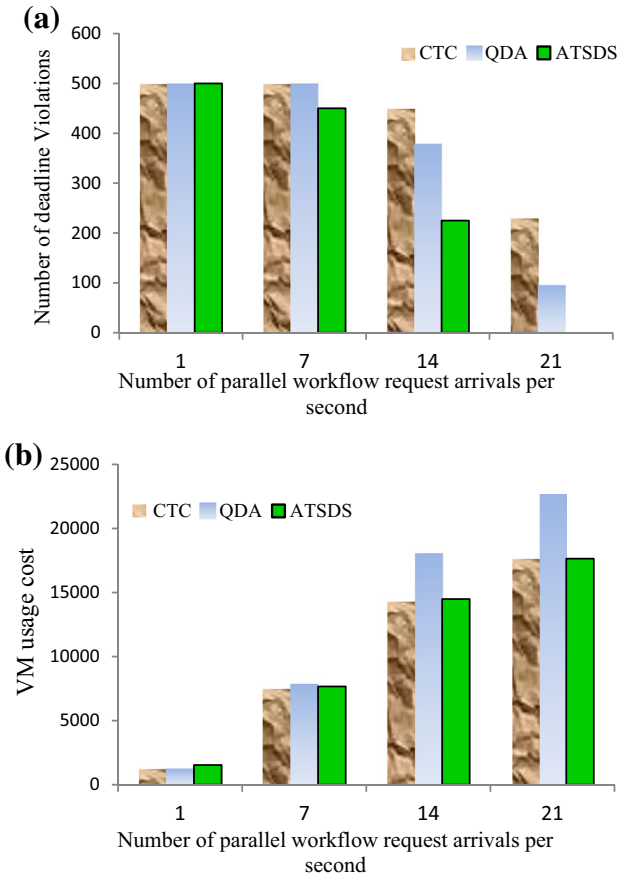

Fig. 12 (Experiment 4 (c)): Comparison diagram of number of deadline violations and the VM usage cost for 1,7,14,21 VMs among the QDA versus CTC versus proposed ATSDS algorithms, constant workflow request arrivals (500). **a** (Experiment 4 (c)-part 1): evaluating the deadline violation parameter, relative deadline time = 15, absolute deadline = 18. **b** (Experiment 4 (c)-part 2): evaluating the VM usage cost parameter, relative deadline time = 15, absolute deadline = 18

and 100% over QDA and 22.08 and ~0% over CTC, respectively, with an increase in the number of VMs. Figure 12a-1 shows that when the VM number increased from 1 to 21, deadline violations for the three approaches decrease. ATSDS achieved the lowest percentage of deadline violations in all cases. For the VM usage cost, Fig. 12a-2 shows that CTC achieved the lowest VM usage cost in all cases. The CTC algorithm did not consider execution time, memory and storage capacity of the VMs; thus the corresponding VMs usage cost was the lowest. ATSDS specified the execution time, memory, and storage capacity of the VMs according to user preference and the simulation results met expectations.

5 Conclusion and future work

Instance-intensive workflows and their execution play an increasingly important role in e-government and e-business applications. The proposed adaptive two-stage scheduling strategy considers run-time circumstances for instance-intensive workflows on the cloud. In the first stage, the Fragmenter is established on the proportional fragment and offers a suitable fragmentation set that is commensurate with the current number of VMs and the percentage of bandwidth available between them. In the resource allocation stage, according to the workflow deadline and the capacity of VMs, the created fragments are allocated to them to be executed. The evaluation results showed that ATSDS more efficiently scheduled business workflows and that the proposed fragmentation algorithm performed better than approaches that do not use adaptive behavior, such as fully distributed execution. Scalability and adaptability were the final targets. The results of simulations indicated that the proposed scheduling algorithm decreased the number of messages exchanged under an increased workload, the workflow completion time, the percentage of workflows that met the deadline and the VM usage cost with respect to CTC and QDA scheduling approaches. Future research will consider other fragmentation algorithms and other adaptability aspects to improve the fragmentation stage for instance-intensive workflows. The current experiments utilized the loan-application business workflow; thus, testing on other real-world application workflows that should be performed in future research. Implementation of auto-scaling mechanisms for the underlying VMs in response to performance requirements is an additional area of interest. The number of essential computing resources at peak times is greater than the average need and, when demand for workflow execution is low, computing resources largely sit idle and waste a large amount of energy. Implementation of elasticity is crucial for delivering resources and reducing the cost of operation. Workflow state management can improve the proposed workflow scheduling system. In other words, if a workflow requires re-fragmentation, then it must be able to retrieve its last state in a new fragment after being re-fragmented.

References

1. Workflow management coalition (1999) Workflow management coalition terminology & glossary. http://www.wfmc.org/
2. Xu R, Liu X, Xie Y, Wang F, Zhang Ch (2014) Logistics scheduling based on cloud business workflows. In: IEEE 18th Int. Conf. Comput. Support. Coop. Work Des
3. Liu X, Ni Z, Yuan D, Jiang Y, Wu Z, Chen J (2011) A novel statistical time-series pattern based interval forecasting strategy for activity durations in workflow systems. J. Syst. Softw. 84:354–376
4. Safi-Esfahani F, Azmi Murad M, Nasir Sulaiman M (2011) Run-time adaptable business process decentralization. Third Int. Conf. Information, Process. Knowl. Manag., no. c, pp 76–82
5. Li G, Muthusamy V (2010) A distributed service oriented architecture for business process execution. ACM Trans. Web, vol. V
6. Sudha M, Monica M (2012) Dynamic adaptive workflow scheduling for instance intensive cloud applications. J Expert Syst (JES), vol. 1, No. 1, Copyr. World Sci. Publ., vol. 1, no. 1, pp. 31–36
7. Tan W (2007) Dynamic workflow model fragmentation for distributed execution. Comput Ind 58(5):381–391
8. Atluri V, Chun S, Mukkamala R (2007) A decentralized execution model for inter-organizational workflows. Distrib Parallel Databases, pp. 55–83

9. Wu Z, Liu X, Z Ni (2013) A market-oriented hierarchical scheduling strategy in cloud workflow systems. J Supercomput, Springer Sci. Media, LLC 63(1):256–293
10. Zhang F, Cao J, Hwang K, Li K, U. Khan S (2015) Adaptive workflow scheduling on cloud computing platforms with iterative ordinal optimization. IEEE Trans Cloud Comput, vol 7161
11. Hoenisch P, Schulte S, Dustdar Sh (2013) Workflow scheduling and resource allocation for cloud-based execution of elastic processes. IEEE 6th Int. Conf. Serv. Comput, Appl, 2013
12. Banerjee S, Adhikari M, Kar S, Biswas U (2015) Development and analysis of a new cloudlet allocation strategy for QoS improvement in cloud. Arab J Sci Eng 40:409–1425
13. Mohialdeen IA (2013) Comparative study of scheduling al-grotihms in cloud computing. J Comput Sci 9(2):252–263
14. Mendling J, Lassen KB (2006) On the transformation of control flow between block-oriented and graph-oriented process modeling languages. Inderscience Enterp. Ltd
15. Kopp O, Martin D, Wutke D, Leymann F (2008) On the choice between graph-based and block-structured business process modeling languages. Model. betrieblicher Informations Syst. (MobIS 2008), Saarbrücken
16. Nguyen BT, Nguyen DH, Nguyen TT (2014) Translation from BPMN to BPEL, current techniques and limitations. Copyr. 2014 ACM 978-1-4503-2930-9/14/12, pp 21–30
17. Mancioppi M, Danylevych O, Karastoyanova D (2011) Toward classification criteria for Process fragmentation techniques. 7th Int. Work. Bus. Process Des., pp 1–12
18. Guo L, Robertson D, Burger Y (2005) A novel approach for enacting the distributed business workflows using BPEL4WS on the multi-agent platform. In: Proc. IEEE Int. Conf. Ebus. Eng. Washington, DC, pp 657–664
19. Viroli M, Denti E, Ricci A (2007) Engineering a BPEL orchestration engine as a multi-agent system. Sci Comput Program 66(3):226–245
20. Baresi L, Maurino A (2006) Towards Distributed BPEL Orchestrations. Electron. Commun. EASST, vol. 3
21. Baeyens T (2013) BPM in the cloud. Springer, Berlin, LNCS 8094, pp 10–16
22. Khalaf R, Kopp O (2008) Maintaining data dependencies across BPEL process fragments. Int J Coop Inf Syst 17:259–282
23. Khalaf R (2006) E Role-based decomposition of business processes using BPEL. IEEE Int. Conf, Web Serv
24. Fdhila W, Yildiz U (2009) A flexible approach for automatic process decentralization using dependency tables University of California. IEEE 7th Int. Conf. Web Serv., ICWS
25. Sadiq W, Sadiq S (2006) Model driven distribution of collaborative business processes. IEEE Int. Conf. Serv. Comput., pp 1–4
26. Zhai Y, Su H (2007) A data flow optimization based approach for BPEL processes partition. IEEE Int. Conf. E-bus. Eng., pp 410–413
27. Cheng J (2012) An agent-oriented approach to process partition and planning in migrating workflow systems. Eng Appl Artif Intell 25(4):837–845
28. Safi-Esfahani F, Azmi Murad M, Nasir Sulaiman M (2011) Adaptable decentralized service oriented architecture. J Syst Softw 84(10):1591–1617
29. Teng F, Yang H, Li T, Yang Y (2013) Scheduling real-time workflow on mapreduce-based Cloud. 978-1-4799-0048-0/13/$31.00 2013 IEEE, pp 117–122
30. Arabnia HR (1990) A parallel algorithm for the arbitrary rotation of digitized images using process-and-data-decomposition approach. J Parallel Distrib Comput, pp 188–192
31. Arabnia HR (1993) Downloaded from Iran library: (http://www.libdl.ir) | Sponsored by Tehran Business School (http://www.tbs.ir). In: Proc. 7th Annu. Int. High Perform. Comput. Conf. (1993) High Perform. Comput. New Horizons Supercomput. Symp. Calgary, Alberta, Canada, June, pp 349–357
32. Arabnia HR (1989) A transputer network for fast operations on digitised images. Int J Eurograph. Assoc. (Comput. Graph. Forum) 8(1):3–12
33. Arabnia HR, Oliver MA (1987) Arbitrary rotation of raster images with SIMD machine architectures. Int. J. Eurographics Assoc. (Computer Graph. Forum), 6(1):3–12
34. Arabnia H. R (1995) A distributed stereocorrelation algorithm. In: Proc. Comput. Commun. Networks (ICCCN'95), IEEE, pp 479-482
35. Arabnia HR (1986) Operations on raster images with SIMD machine architectures. Int. J. Eurograph. Assoc. (Comput. Graph. Forum) 5(3):179–188

36. Arabnia HR (1996) Parallel stereocorrelation on a reconfigurable multi-ring network. J Supercomput, Springer Publ. vol. 10, No. 3, pp 243–270, vol 269, pp 243–269
37. Bhandarkar S, Arabnia HR (1995) The Hough transform on a reconfigurable multi-ring network. Parallel Distrib Comput 24(1):107–114
38. Wani MA, Arabnia HR (2003) Parallel polygon approximation algorithm targeted at reconfigurable multi-ring hardware. J Supercomput 25(1):43–63
39. Zhu M, Cao F (2014) High-throughput scientific workflow scheduling under deadline constraint in clouds. J Commun 9(4):312–321
40. Liu K, Jin H, Chen J, Liu X, Yuan D (2010) A compromised-time-cost scheduling algorithm in SwinDeW-C for instance-intensive cost-constrained workflows on cloud computing platform. Int J High Perform Comput Appl 24(2010):1–16
41. Li H, Ge S, Zhang L (2014) A QoS-based scheduling algorithm for instance-intensive workflows in cloud environment. Control Decis. Conf. (2014 CCDC), 26th Chinese, pp 4094–4099
42. Topcuoglu H, Hariri S, Wu M (2002) Performance-effective and low-complexity. IEEE Trans Parallel Distrib Syst 13(3):260–274
43. Vega D (2010) Towards fuzzy granularity control in parallel/ distributed computing. Int J Child Comput Interact, pp 43–55
44. Moens H, Handekyn K (2013) Cost-aware scheduling of deadline-constrained task workflows in public cloud environments. 978-3-901882-50-0c 2013 IFIP, pp 68–75
45. Ghafarian T (2013) Deadline-constrained workflow scheduling in volunteer computing systems. Springer Int. Publ. Switz
46. Mao M, Humphrey M (2011) Auto-scaling to minimize cost and meet application deadlines in cloud workflows. SC11, Novemb. 12–18, 2011, Seattle, Washington, USA Copyr. 2011 ACM 978-1-4503-0771
47. Ramakrishnan L, Chase JS, Gannon D, Nurmi D, Wolski R (2011) Deadline-sensitive workflow orchestration without explicit resource control. J Parallel Distrib Comput 71(3):343–353
48. Harper jS, Wilcox DV (2000) A toolset for the performance prediction of parallel and distributed systems. Int J High Perform Comput Appl 14(3):228–251
49. Cooper K, Dasgupta A, Kennedy K, Koelbel C, Mandal A, Marin G, Mazina M, Berman F, Casanova H, Chien A, Dail H, Liu X, Olugbile A, Sievert O, Xia H, Johnsson L, Liu B, Patel M, Reed D, Deng W, Mendes C (2004) New grid scheduling and rescheduling methods in the GrADS project. NSF Next Gener. Softw. Work. Int. Parallel Distrib. Process. Symp. St. Fe, IEEE CS Press. Los Alamitos, CA, USA
50. Jang S, Wu X, Taylor V, Texas A, Station C, Mehta G, Vahi K, Deelman E, Way A, Del Rey M, (2004) Using performance prediction to allocate grid resources. Tech. Rep. 2004-25, GriPhyN Proj. USA, pp 1–11
51. Mostinckx S, Van Cutsem T, Timbermont S, Boix E.G, Tanter E, De Meuter W (2011) CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Exp. Publ. Wiley Online Libr., vol 39, no. 7, pp 661–699
52. Li G, Muthusamy V, Jacobsen H (2010) A distributed service-oriented architecture for business process execution. ACM Trans. Web, 4(1)