A Novel Deadline Assignment Strategy for a Large Batch of Parallel Tasks with Soft Deadlines in the Cloud

Xiao Liu[1], Dingxian Wang[1], Dong Yuan[2],Yun Yang[3,2]
[1]Software Engineering Institute, East China Normal University, Shanghai, China
[2]Faculty of Information and Communication Technology, Swinburne University of Technology, Melbourne, Australia
[3]School of Computer Science and Technology, Anhui University, Hefei, China
xliu@sei.ecnu.edu.cn, dingxianwang@gmail.com, {dyuan, yyang}@swin.edu.au

***Abstract*—Deadline assignment is to assign each subtask composing a distributed task with a local deadline such that the global deadline can be met. Today's real-time systems often need to handle hundreds or even thousands of concurrent customer (or service) requests. Therefore, deadline assignment is becoming an increasingly challenging issue with a large number of parallel and distributed subtasks. However, most conventional strategies are designed to deal with a single independent task rather than a batch of many parallel tasks in a shared resource environment such as cloud computing. To address such an issue, in this paper, instead of assigning local deadline for each subtask, we propose a novel strategy which can efficiently assign local throughput constraints for a batch of parallel tasks at any time point along the system timeline. The basis of this strategy is a novel throughput consistency model which can measure the probability of on-time completion at any given time point. The experimental results demonstrate that our strategy can achieve significant time reduction in deadline assignment and achieve the most "consistency" between global and local deadlines compared with other representative strategies.**

Keywords-Deadline Assignment, Throughput, Parallel Tasks, Cloud Computing

I. INTRODUCTION

Deadline assignment is a classic problem in distributed systems where each subtask composing a distributed task must receive a local deadline so that the global deadline can be met [3, 9, 20, 21]. The global deadline usually serves as an important QoS (Quality of Service) constraint specified in the Service Level Agreement (SLA) between the customer and the service provider. In general, a global deadline can be classified into two types, viz. hard deadline and soft deadline. A hard deadline means that if missed, it is a failure. A soft deadline means that if missed, the usefulness of a result will be deteriorated and thus degrades the service quality [10]. In the real world, hard deadlines are often applied in safety-critical areas such as railway traffic control, automotive electronics, and aerospace electronic equipment [8]. In contrast, soft deadlines have a much broader application in the fields where the service quality for handling massive concurrent customer requests is the first priority such as e-Government, e-Business and e-Science [13]. For example, a government taxation office needs to process thousands of tax declarations every day for individual and enterprise customers. Failures of completing these tasks in time will result in significant deterioration of customer satisfaction and even huge financial losses, e.g. the taxation office would have to pay a large amount of money to compensate the interest loss of tax payers for days of the delay over its official deadline. Therefore, on-time completion is critical for delivering services with satisfactory quality [6].

In order to meet the deadlines, a set of QoS management strategies are required. For example, a typical temporal QoS management framework proposed in [14] consists of three basic steps, viz. temporal constraint setting [12], temporal consistency monitoring [17], and temporal violation handling [18]. Specifically, the setting of temporal constraints is the first step in the framework which can be further divided into two sub-steps, viz. the setting of global deadlines, and the setting of local deadlines (i.e. deadline assignment). The global deadline can be either adopted by default (according to the system design) or specified through a negotiation on the price that the customer willing to pay and the level of service quality that the service provider would like to offer [13]. As for local deadlines, they are usually assigned by the system with a chosen deadline assignment strategy to ensure the on-time completion of subtasks. A local deadline plays an important role in the decision-making for resource provision and task scheduling [7, 21]. It also serves as the key objective for the monitoring of subtask execution which is very essential yet challenging in a distributed environment. Therefore, as proved in many studies, the setting of proper local deadlines has significant contributions to the successful on-time completion of distributed real-time tasks [13, 19].

In this paper, we focus on soft real-time systems in the Cloud where tens, hundreds or even thousands of tasks are running in parallel at the same time. Meanwhile, since workflow systems are typical soft real-time distributed systems, many studies in the area of workflow or process management on temporal constraint settings are also investigated [2, 14, 15, 21]. Therefore, in this paper, the terms "task" and "subtask" are interchangeable with the terms "workflow" and "activity" respectively.

Most of the current studies focus on the response time of a single task rather than the throughput of a large batch of parallel tasks. Compared with response time which measures how long a user submitted request takes to complete, throughput defines how many requests have been completed by the system in a basic time unit. Therefore, in general, for running a large batch of parallel tasks, throughput is a better performance indicator because it measures the overall system

performance [11, 16]. For tasks running in the Cloud, given a set of soft deadlines, cloud service providers must be able to provide and maintain a certain level of system throughput so as to achieve satisfactory QoS, e.g. a specific confidence (probability) for on-time completion such as 90%. Such a "certain level of system throughput" is specified in the form of throughput constraints, which explicitly define the bottom-line throughput that a system should provide to achieve on-time completion. However, the setting of throughput constraints for a large batch of parallel tasks is a challenging issue. Specifically, there are three fundamental questions: Q1) *how to measure the system throughput?* Q2) *how to estimate the probability of on-time completion?* Q3) *how to assign local throughput constraints?* Details will be discussed in Section II.B.

In this paper, to answer and address these above mentioned questions, first, we introduce the novel definitions for throughput constraints and candidate constraint points which are specifically designed for a large batch of tasks with soft deadlines; second, we propose a novel throughput consistency model which can measure the probability of on-time completion at any given constraint point. Third, based on a queueing model which can predict activity durations in the Cloud, we propose a novel deadline assignment strategy which can efficiently assign local throughput constraints to any constraint points along the system timeline. Finally, a number of simulated workflows are employed to evaluate our strategy. The results have shown that our strategy can achieve significant time reduction in deadline assignment and achieve the most "consistency" between global and local deadlines (i.e. being most consistent between the on-time completion rates of global and local deadlines) compared with other representative generic strategies.

The remainder of this paper is organized as follows. Section II introduces the related work and presents the problem analysis. Section III introduces a set of new definitions and proposes the novel throughput constraint setting strategy. Section IV demonstrates the evaluation of our strategy through simulation experiments. Finally, Section V addresses the conclusion and points out the future work.

II. Related Work and Problem Analysis

A. Related Work

Deadline assignment is a well-known problem in many real-time distributed systems [20, 21]. Most strategies are designed using basic statistics and simple computation to guarantee a small time overhead which is very important in real-time systems. Kao and Garcia-Molina in [9] examined four classical strategies, viz. Ultimate Deadline (UD), Effective Deadline (ED), Equal Slack (EQS) and Equal Flexibility (EQF) for subtask deadline assignment in a distributed soft real-time environment. Specifically, UD sets the local deadline for a subtask the same as the global deadline. This method is not practical for real-time systems since it gives little control over the subtasks. For the other three strategies, the idea is to fairly distribute the time slack (i.e. the expected time redundancy between the mean response time and the global deadline) to the subtasks. However, different researchers may have different understanding about "fairness", and thus leads to different approaches for slack distribution. For example, ED of a subtask is equal to the deadline of its global task minus the total expected time of its following subtasks. EQS divides the total remaining slack equally among the remaining subtasks. EQF divides the total remaining slack among the subtasks in proportion to their execution times and thus subtasks can share equal flexibility.

Besides the above four generic methods, there are also some other strategies which are designed to optimize specific objectives. The work in [10] proposed a convex optimization framework which can effectively address the deadline assignment problem while maximizing the aggregate quality of service including time and cost. In [19], Marinca, Minet and George propose Fair Laxity Distribution (FLD) and Unfair Laxity Distribution (ULD) to maximize the number of acceptable flows. These two methods are based on the flow minimum sojourn time that should be guaranteed on each visited node. The difference between the two methods is the laxity distribution. FLD applies a fair laxity distribution between the visited nodes, whereas ULD tends to make use of the proportion of the workload to allocate the laxity between visited nodes. FLD and ULD are also compared with two other deadline assignment strategies including fair assignment (FA) and assignment proportional to workload (PTW). FA sets the deadline of a subtask equal to the end-to-end deadline divided by the number of subtasks, and PTW sets the deadline of each subtask according to the proportion of its workload. The studies shows that different scheduling algorithms should be used with different deadline assignment strategies to achieve the maximum number of acceptable flows.

While all the above work adopts global deadline by default, the work in [12] proposes a probabilistic strategy for temporal constraint setting which considers both the setting of global deadlines and local deadlines. With a probability-based temporal consistency model, a negotiation process between the user and the service provider is designed to support the setting of global deadlines and then automatically derive the local deadlines for each workflow activity. The global deadlines and the local deadlines have the same confidence for on-time completion such as 90%. The work in [13] also considers the updates of deadlines at runtime when the workflow execution is either ahead of or behind the schedule. However, they can only work on the response time of single tasks rather than the throughput of parallel tasks.

Although many deadline assignment strategies have been proposed, to the best of our knowledge, there is so far no strategy dedicated to the setting of throughput constraints for a large batch of parallel tasks in a cloud computing environment.

B. Problem Analysis

To deal with a large batch of parallel tasks with soft deadlines in the Cloud, we need to address the following three fundamental questions:

Q1) *How to measure the system throughput?* The generic definition of system throughput is how many tasks have been

completed by the system in a basic observation time unit [4] . Here, a task must be finished completely to be accounted in the throughput measurement. Normally, the basic observation time unit is set to be small so as to support effective monitoring and control. However, in many distributed real-time systems, the task response time can be much longer than the basic observation time unit. Therefore, we need to also consider partial completion in the throughput measurement. Unfortunately, there is so far no definition on the system throughput which can define the partial completion for a large batch of parallel tasks with soft deadlines.

Q2) *How to estimate the probability of on-time completion?* The probability (namely confidence) of on-time completion against specific deadlines is very important for both deadline assignment and runtime monitoring. The accuracy of the probability estimation relies significantly on the effective response time estimation of the subtasks or workflow activities. Conventional response time estimation basically employs random distribution models. However, these random distribution models are mainly based on static statistics rather than real-time system performance. Hence, their effectiveness will be deteriorated in the dynamic system environments such as cloud computing where the resources are elastically scaled according to the real-time system demand. Therefore, new estimation methods are required. Furthermore, to estimate the probability of on-time completion, a probability based temporal consistency model is required [13]. However, current temporal consistency models are only for response time constraints rather than throughput constraints. Therefore, a novel temporal consistency model needs to be defined.

Q3) *How to assign local throughput constraints?* A set of throughput constraints is required to ensure the desired service quality, i.e. the probability of on-time completion. For the running of massive parallel and distributed workflows, a throughput constraint setting strategy is required to automatically assign a set of throughput constraints along the system timeline wherever necessary. However, there is so far no deadline assignment strategy dedicated to the setting of throughput constraints.

Here, we propose two basic measurements which can be used to evaluate the performance of the deadline assignment strategies, viz. efficiency and consistency. Specifically, efficiency means that: 1) the time overhead for the deadline assignment strategy should be trivial; 2) the number of required constraints for runtime monitoring should be as small as possible. For most strategies, the total time overhead can grow very fast when dealing with a large batch of parallel tasks. Therefore, we need a strategy which can maintain its computation overhead. Meanwhile, the number of required constraints decides how many times a deadline assignment strategy needs to be repeated. It also determines the number of monitoring objectives at runtime, and thus decides the monitoring cost [13]. Therefore, we need a strategy which can monitor the system with a small number of constraints. As for consistency, it means that given a global deadline which can guarantee a specific level of service quality such as 90% on-time completion rate for the global task, the local deadlines can also guarantee a consistent service quality of 90% on-time completion rate for the local subtasks. In a cloud computing environment, the users are charged according to the promised service quality. If the provisioned service quality is lower than the promised, the service provider will have to pay the penalty. However, if the provisioned service quality is higher than the promised, namely the resource are over-provisioned, the potential extra cost will be covered by the service providers themselves. Clearly, neither higher nor lower service quality than the promised is desirable. Therefore, we need a strategy which can ensure a high consistency between the global and local deadlines. Both efficiency and consistency will be evaluated and compared in Section IV.

III. A NOVEL DEADLINE ASSIGNMENT STRATEGY

As discussed in Section II.B, current work mainly focuses on the response time of single distributed task. In this paper, we use system throughput for monitoring a batch of parallel tasks. It should be noted that in our strategy, the size of the "batch" is not a fixed value but rather determined by the system at runtime. The batch can start at any arbitrary time point as long as those parallel tasks are having the same deadline. This situation ensures that a common global deadline exists so that a deadline assignment strategy can be applied similarly to each individual task. In this section, we will answer the three problems analyzed in Section II.B in the following three subsections respectively.

A. Workflow Throughput and Throughput Constraints

Generally speaking, workflow throughput, namely the throughput of the workflow system, is the number of workflows that have been completed in a basic time unit [1, 4]. Meanwhile, workflow throughput can also be measured by the number of workflow activities that have been completed in a basic time unit [11]. For the former definition, the objects being observed are the workflows, i.e. the monitoring system will only be notified when an entire workflow has been completed. This is obvious not effective for monitoring the running of a large batch of parallel workflows. It will be too late for violation handing strategies to take place when the workflows have already been finished [17]. As for the later definition, the objects being observed are the workflow activities, i.e. the monitoring system will be notified whenever a workflow activity has been completed. This is much more fine-grained but can result in more monitoring cost. In addition, the later definition does not differentiate the durations of workflow activities. For example, if one workflow activity running for 2 minutes and another one running for 20 seconds are completed at the same time, their contributions to the system throughput are treated the same, e.g. both accounted for one activity completion. However, it is evident that their actual contributions for meeting the soft deadlines are very different.

Here we define some basic annotations: a_i is a workflow activity (equivalent to a subtask) with its mean, minimum, and maximum durations (i.e. response time) denoted as $M(a_i)$, $d(a_i)$ and $D(a_i)$ respectively; the activity duration

weight of a_i is denoted as w_i which represents the influence of the process structure such as sequence, parallelism, iteration and choice to the completion time of the entire workflow [12]; WF_i is a workflow (equivalent to a task) with its mean, minimum, and maximum completion time denoted as $M(WF_i)$, $d(WF_i)$ and $D(WF_i)$ respectively; the basic time unit for monitoring (i.e. the interval for two consecutive monitoring) is denoted as bt. To model these time attributes, we need some basic statistics. Among them, the two most popular statistical values are the expected value μ_i and the standard deviation σ_i for the activity duration of a_i.

Definition 1 (Workflow Throughput). Given a batch of m parallel workflows $Batch\{WF_1, WF_{2,...}WF_m\}$ which starts at system time S_0, the completion of a workflow activity a_i contributes to the completion of the entire batch of workflows with a value of $w_i M(a_i)/T$ where $T = \sum_{i=1}^{m} M(WF_i)$. Here, assume at the current observation time point S_t, the set of new completed activities from the last nearest observation time point S_{t-1} (i.e. $S_t - S_{t-1} = bt$) is denoted as $a\{\}|_{S_{t-1}}^{S_t}$, then the current system throughput is defined as $TH|_{S_{t-1}}^{S_t} = W \times M(a\{\}|_{S_{t-1}}^{S_t})/T$.

Given this new definition for workflow throughput, we can clearly measure how much activities completed during a basic time unit contributes to the completion of the entire batch. To ensure on-time completion, a global deadline together with a set of local milestones is assigned to facilitate the monitoring of workflow execution. In this paper, we will assign throughput constraints instead of conventional response time constraints. Given Definition 1, throughput constraints are the expected accumulated workflow throughputs that should be achieved at a specific system time point. Here, the formal definition for workflow throughput constraints is presented as follows.

Definition 2 (Workflow Throughput Constraint). Given the same batch of workflows as defined in Definition 1, the throughput constraint assigned at system time point S_t is denoted as $THCons|_{S_0}^{S_t}$ which means that the expected accumulated system throughput $\sum_{i=1}^{t} TH|_{S_0}^{S_i}$ from S_0 to S_t should be no less than the value of the assigned throughput constraint. The actual value of $THCons|_{S_0}^{S_t}$ will be decided by the constraint setting strategy.

Throughput constraints can be assigned at any system time point and as many as required by the system. In general, the more dynamic the system performance, the larger the number of local throughput constraints are required. Meanwhile, different from conventional constraint setting where the response time constraints are assigned to workflow activity points, throughput constraint setting are assigned to the system time points [16]. In practice, since there is normally a basic time unit for system monitoring, i.e. bt, local throughput constraints should be assigned accordingly. In this paper, we name the candidate time points for constraint setting as candidate constraint point. The formal definition for candidate constraint point is presented as follows.

Definition 3 (Candidate Constraint Point). Given the same batch of workflows in Definition 1, a system time point S_i along the workflow execution timeline is a candidate constraint point if $S_i - S_0 = n \times bt$ ($n = 1,2,3...$).

To address the problems such as how many constraint points should have and where the constraint points should be, it requires a strategy named temporal checkpoint selection [17]. However, since the focus of this paper is throughput constraint setting, we will leave the throughput checkpoint selection strategy as our future work. In this paper, to simplify our discussion, we will just use an intuitive strategy where a fixed time interval is defined so that the constraint points are equally distributed along the system timeline. More details will be illustrated in Section IV.

B. A Novel Build-Time Throughput Consistency Model

The estimation of on-time completion requires a temporal consistency model. According to the workflow lifecycle, a build-time temporal consistency model can help to check whether the targeted global deadline can be satisfied or not given the expected durations of workflow activities [13]. Therefore, it can help the user and the service provider to negotiate a proper global deadline. Meanwhile, during workflow execution, a runtime temporal consistency model can help to verify at any time point whether the deadline can be met or not, i.e. temporal consistency or inconsistency. In this paper, as we investigate the problem of deadline assignment which occurs at workflow build time, we focus on the build-time throughput consistency model. Here, to simplify the discussion of our work, we assume all the activity durations follow the normal distribution model $N(\mu_i, \sigma_i^2)$ where μ_i is the expected value and σ_i is the standard deviation. However, activity durations follow other distribution models such as uniform and exponential can also be transformed and have limited effects on the model [12]. Based on such an assumption, as discussed in [13], the completion time of the workflow can also be estimated with the weighted joint normal distribution of individual activity durations. A typical feature of normal distribution is the "3σ rule" rule which depicts that for any sample coming from normal distribution model, it has a probability of 99.73% to fall into the range of $[\mu_i - 3\sigma_i, \mu_i + 3\sigma_i]$. Therefore, it is practical to eliminate the outliers and assume $D(a_i) = M(a_i) + 3\sigma_i$ and $d(a_i) = M(a_i) - 3\sigma_i$.

Definition 4 (Throughput Consistency Model). Given the same batch of workflows in Definition 1, and its final deadline denoted as $F(WF_m)$, it is said to be of:

1) Absolute Consistency (**AC**), if $\frac{W \times D(a\{\}|_{S_0}^{F(WF_m)})}{m \times (F(WF_m) - S_0)} \le 100\%$;

2) Absolute Inconsistency (**AI**), if $\frac{W \times d(a\{\}|_{S_0}^{F(WF_m)})}{m \times (F(WF_m) - S_0)} \geq 100\%$;

3) $\alpha\%$ Consistency ($\alpha\%$ **C**), if

$$\lambda_\alpha = 3 \times \frac{m \times (F(WF_m) - S_0) - W \times M(a\{\}|_{S_0}^{F(WF_m)})}{(W \times D(a\{\}|_{S_0}^{F(WF_m)}) - W \times M(a\{\}|_{S_0}^{F(WF_m)})}$$ where λ_α ($-3 \leq \lambda_\alpha \leq 3$) is defined as the $\alpha\%$ ($0 < \alpha < 100$) confidence percentile with the cumulative standard normal distribution function of $F(\lambda_\alpha) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\lambda_\alpha} \ell^{-x^2/2} \bullet dx = \alpha\%$. Here, the factor of 3 is to make sure that λ_α follows the standard normal distribution model.

In general, AC denotes the state that even when every activity is running with its maximum duration, the final deadline can still be met. In contrast, AI denotes that even when every activity is running with its minimum duration, the final deadline still cannot be met. Clearly, both AC and AI are two extreme situations while the rest can be represented by $\alpha\%$ C which is a better measurement for describing the current service quality. For example, many commercial cloud service providers such as Amazon Web Service uses percentage values like 99%, 99.9% and 99.99% for service quality on reliability and availability. Similarly, a confidence value such as 90% of on-time completion is more practical for service providers to specify different levels of service quality.

C. Throughput Constraint Setting Strategy

As can be seen in Definition 4, the expected value and the standard deviation are the two major statistics used in the throughput consistency model. Actually, in most studies, the mean, minimum, and maximum durations are estimated using μ_i and σ_i with representative distribution models such as normal, exponential and uniform [14]. These statistics are usually generated from large sample size and these distribution models are normally used in the scenarios where the system performance is relatively static and the activity durations are independent to each other. In this paper, we investigate the cloud computing environment where the underlying services are provisioned elastically according to the number of parallel workflows in the batch. However, conventional distribution models cannot adapt to such kind of changes, and thus results in inaccurate predictions. Therefore, unlike most of the current studies, we will adopt a latest work on the performance analysis of cloud computing centers where a queueing model can be used to estimate the activity durations [4]. A queueing model is much more powerful and capable of easily adapting to the changes of input tasks and number of services. Due to the space limit of this paper, we have to omit the detailed discussion for the rationale of the model design and present the queueing model used in our paper which is the same as the one proposed in [5] directly as follows.

Definition 5 (A M/G/m/m+r Queueing Model). In a specific batch of workflows, for n workflow activities of the same type, there are m dedicated services where n is normally much larger than m. The queueing model that we adopted is M/G/m/m+r which indicates that the inter-arrival time of requests is exponentially distributed, while task service times are independent and identically distributed random variables that follow a general distribution with mean value of μ_i for a_i. It contains m services and the service order is FCFS. The capacity of the system is m+r which means that the buffer size for incoming request is equal to r, i.e. n - m in this case.

Based on such a model and given the basic statistics, together with the number of parallel workflow activities and the number of services for the current batch, we can efficiently obtain more accurate mean durations which reflect system changes. Please refer to [5] for the formulas of calculating the mean durations, and be noted that in this paper an activity duration is the execution time plus the waiting time. There are many tools available to facilitate the calculation such as popular QtsPlus [4]. Now, we present the novel throughput constraint setting strategy.

Definition 6 (A Novel Throughput Constraint Setting Strategy). Given the same batch of workflows in Definition 1, and its final deadline denoted as $F(WF_m)$, at a candidate constrain point S_i where $S_i - S_0 = n \times bt$ ($n = 1,2,3 \ldots \frac{F(WF_m)}{bt} - 1$), the throughput constraint assigned at S_i by our novel throughput constraint setting strategy is $Cons(S_i) = Cons|_{S_0}^{S_i} = \frac{\sum_{i=1}^{m} M(WF_i)}{m * F(WF_m)} \times W \times M(a\{\}|_{S_0}^{S_i})/T$ which denotes the expected percentage of completion at system time point S_i. Since $T = \sum_{i=1}^{m} M(WF_i)$ as in Definition 1, we can have $Cons(S_i) = \frac{W \times M(a\{\}|_{S_0}^{S_i})}{m * F(WF_m)}$.

It can be easily seen that given the novel definition of workflow throughput, our constraint setting strategy is to assign the expected percentage of completion to the current constraint point. As there is practically no limit on the position of a constraint point when the basic time unit for monitoring bt is small enough, our strategy can efficiently assign throughput constraints as many as required along the system timeline.

IV. Evaluation

In this section, we evaluate our strategy (denoted as TCS for Throughput Constraint Setting) and compared with 4 other representative generic deadline assignment strategies [9] including Ultimate Deadline (UD), Effective Deadline (ED), Equal Slack (EQS) and Equal Flexibility (EQF), according to the two basic performance measurement discussed in Section II.B, viz. efficiency and consistency. We do not compare

with other strategies such as Fair Laxity Distribution and Unfair Laxity Distribution in this paper because they are designed for different optimization objectives.

The computation overhead for the generic strategies on each local deadline is small as they only include simple computation, but they have to be repeated for a large number of times [9]. As for our strategy, the computation overhead with the queueing model is relatively higher but still very small, and it only needs to be repeated for a few times since we use throughput constraints. Therefore, the total computation overhead for each strategy can be regarded as small and similar. Here, to simplify our experiments, we directly compare the number of required constraints for each strategy for the measurement of efficiency. As for the measurement of consistency, we need to specify a global constraint which can guarantee a specific global on-time completion rate (i.e. the service quality), and then compare the local on-time completion rate of each strategy, denoted as LR, with the global on-time completion rate, denoted as GR. Specifically, the consistency rate is defined as follows:

Consistency Rate: $1-|GR-LR|/GR$ **Formula (1)**

In this paper, since our focus is on deadline assignment, we do not discuss about the setting of global constraints. Therefore, we borrow the work in [13] which presents a probability based strategy to statistically guarantee the same on-time completion rate for both global and local constraints. Specifically in this paper, we set the confidence rate as 90% which is the same as in the previous works [12, 13, 17].

A. *Experimental Settings*

In our experiments, we simulate the running of a batch of parallel workflows in the Cloud. The basic experimental settings are shown in Table I.

TABLE I. EXPERIMENTAL SETTINGS

Round #	Number of Parallel Workflows	Number of Cloud Services
1	20	10
2	50	10
3	100	20
4	500	100
5	1000	200
Workflow Settings		
Workflow	A simple workflow with 10 sequential activities	
Service Provision	One service dedicates to one type of activities and the maximum queue length is 50	
Activity Durations	Step 1: randomly generate the mean execution time for each activity using normal distribution models Step 2: 1) For our strategy, the mean durations are generated using the queueing model 2) For other strategies, the mean durations are the sum of the mean execution time and the expected waiting time in the queue of each cloud service	
Constraint Points	1) For our strategy, we select 10 equally distributed time points along the system timeline for a fair comparison 2) For other strategies, every activity is a constraint point	

Like in other literatures [9], to simplify our discussion and focus on the performance of the deadline assignment strategies, we only use a simple workflow with 10 sequential activities. However, our strategy as well as other strategies discussed in this paper can be easily applied to general workflows with different structures and more activities [13]. The number of parallel workflows increases from 20, 50, 100, 500 to 1,000, which covers a typical range of concurrent user requests in a real-time business system. As for the number of cloud services, it is provisioned according to the number of parallel workflows to simulate the dynamic resource provisioning in the cloud. The maximum queue length is set as 50 which is a reasonable size for the execution of most business tasks. Therefore, except for the first round where the number of cloud services is set as 10 (since there is at least one service for one type of activity), the number of parallel workflows and the number of cloud services are set with a ratio of 5:1 given each workflow contains 10 activities. As for the activity durations, the mean execution time is first randomly generated from (30, 300) time units to cover a large searching space using normal distribution models. Afterwards, the queueing model proposed in [5] is adopted to simulate the actual system performance. The mean durations of our strategy are generated by the queueing model while others are calculated as the sum of the mean execution time and the expected waiting time in the queue of each cloud service as in [9]. For a fair comparison, our strategy selects 10 equally distributed constraint points along the system timeline, specifically 10% of the global deadline, 20% of the global deadline, and so on so forth. As for other strategies, every activity is a constraint point since the local deadlines are assigned to each subtask. Each experiment is executed for 1,000 times to get the average values.

B. *Experimental Results*

The number of total constraints and the reduction rates are depicted in Table II.

TABLE II. REDUCTION RATES OF TOTAL CONSTRAINTS

Round #	Number of Parallel Workflows	Number of Total Constraints for TCS	Number of Total Constraints for all other strategies	Reduction Rate
1	20	10	20*10=200	95.0%
2	50	10	50*10=500	98.0%
3	100	10	100*10=1000	99.0%
4	500	10	500*10=5000	99.8%
5	1000	10	1000*10=10000	99.9%

Since all the other strategies assign each workflow activity with a local constraint, the number of total constraints is the same as the number of total workflow activities in our experiments. In contrast, our strategy only assigns one throughput constraint at each constraint point regardless the number of parallel workflows. Therefore, our strategy only needs 10 constraints for each batch of parallel workflows. The reduction rates for each round of experiments are 98.0%, 99.0%, 99.5% 99.8% and 99.9 respectively. This is a significant improvement over the efficiency for monitoring a large batch of parallel workflows.

Next, we will present the results on the comparison of the consistency between global and local on-time completion rates.

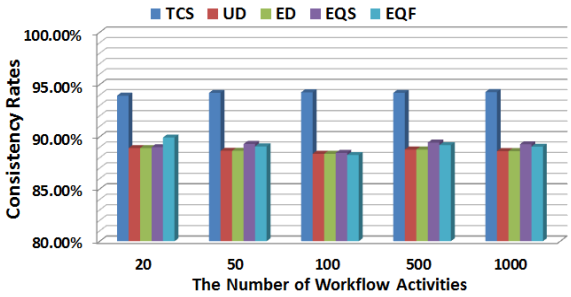

Figure 1. Average Consistency Rates

Figure 1 shows the average consistency rates for the 5 rounds of experiments. Since our global deadline is set with 90% of on-time completion rate using the method in [13], the consistency rate here measures how much the local on-time completion rate approximates to 90%. As can be seen from Figure 1, our strategy TCS has the largest consistency rates in all the 5 rounds of experiments and with an average rate of 94.17%. As for the other strategies, EQF is the second best with an average rate of 89.1%, but very close to the third best EQS with an average rate of 89.0%. UD and ED are almost the same with an average rate of 88.67%. We further take a look at two specific rounds of experiments which are the two extremes, viz. the smallest with 20 parallel workflows and the largest with 1,000 parallel workflows. The results are shown in Figure 2 and Figure 3 respectively. It is surprisingly to see that in Figure 2, EQF is better than TCS in 4 out of the total 9 constraint points. However, it is far less stable as TCS. We reckon this is a result of its adoption of random distribution models. The average consistency rate for TCS is 93.4%, which is still much better than EQF with an average of 89.9%. The other three strategies are all with an average around 88.9%. When the number of parallel workflows increases to 1,000, our strategy shows a significant advantage over the others at every constraint point. The average consistency rate for TCS is 94.3% and the maximum value achieved at the ninth constraint point is 99.1%. This is a clear evidence to demonstrate that TCS is better dealing with a large batch of parallel workflows than others.

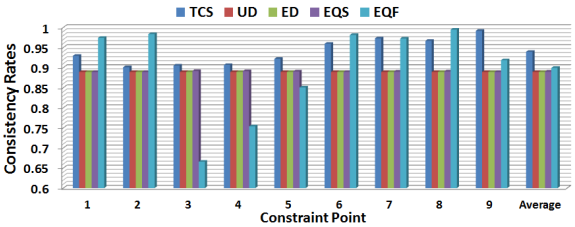

Figure 2. Consistency Rates (20 Parallel Workflows)

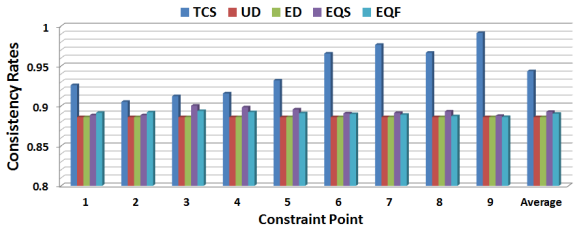

Figure 3. Consistency Rates (1,000 Parallel Workflows)

We also take an in-depth analysis of these generic strategies and their low consistency rates are the results of three major problems: 1) for UD, the local deadlines are often excessively large. Therefore, the local on-time completion rates can be very high but with no effects on the global ones; 2) for ED, only the execution time is considered in the distribution of the time slacks. However, for a large batch of parallel workflows, the major component of the activity duration is the waiting time which is normally much larger than the execution time; 3) for EQS and EQF, they indeed consider the waiting time of the activity itself but without the waiting time of other activities. In addition, the waiting time is calculated with simple estimation [9], and thus it is far less accurate than ours with the queueing model.

To summarize, the experimental results successfully demonstrate that our strategy can effectively address the deadline assignment problem for a large batch of parallel workflows from tens to thousands, and even more can be expected. Our strategy beats all the other generic deadline assignment strategies in both "efficiency" and "consistency", and the advantage is becoming more evident when the number of parallel workflows increases. In addition, the results show that our strategy can maintain a stable performance with different number of parallel workflows and different number of cloud services. This is very important in a cloud computing environment where the resources are dynamically provisioned according to the changing system demand.

V. Conclusions and Future Work

Deadline assignment is a classic problem in distributed systems, but it is becoming increasingly challenging in the soft real-time systems where large batches of tasks are running in parallel in a shared resource environment such as cloud computing. It is intuitive that throughput is a better performance measurement than response time for the monitoring of large batch of parallel tasks. However, to the best of our knowledge, there is so far no existing work dedicated to the setting of throughput constraints to achieve targeted service quality, i.e. specific on-time completion rate for a large batch of parallel tasks running in the Cloud. To address such an issue, we have proposed a novel throughput consistency model which can measure the probability of on-time completion for a large batch of workflows, and a novel throughput constraint setting strategy which can assign a local throughput constraint at any given time point along the system timeline. The experimental results demonstrated that our strategy can

achieve significant advantages over other generic strategies in both efficiency and consistency.

Since this paper focused on the evaluation of the novel definition for throughput constraints and the novel throughput constraint setting strategy, the experiments were designed to be simple and fair for comparison purposes. Therefore, there is still large space for us to improve the experiments such as introducing real-world business processes, and more complicated resource provisioning and task scheduling policies. In the future, we will also investigate the checkpoint selection strategy to determine the best number of constraint points and achieve the targeted on-time completion rates.

ACKNOWLEDGMENT

The research work reported in this paper is partly supported by National Natural Science Foundation of China (NSFC) under No. 61300042 and No. 61021004, Australian Research Council under LP0990393 and LP130100324, the Fundamental Research Funds for the Central Universities, and Shanghai Knowledge Service Platform Project No. ZF1213. Yun Yang is the corresponding author.

REFERENCES

[1] W. M. P. van der Aalst and K. M. V. Hee, Workflow Management: Models, Methods, and Systems: The MIT Press, Cambridge, 2002.

[2] J. Eder, E. Panagos, and M. Rabinovich, "Time Constraints in Workflow Systems", Proc. 11th International Conference on Advanced Information Systems Engineering (CAiSE99), pp. 286-300, Heidelberg, Germany, 1999.

[3] M. Garcia-Valls, R. Fern'ndez-Castro, I. Estevez-Ayres, P. Basanta-Val, and I. Rodriguez-Lopez, "A Bounded-Time Service Composition Algorithm for Distributed Real-time Systems", Proc. 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), pp. 1413-1420, 2012.

[4] D. Gross, J. Shortle, J. Thompson, and C. Harris, Fundamentals of Queueing Theory (Fourth Edition) John Wiley & Sons, 2008.

[5] K. Hamzeh, "Performance Analysis of Cloud Computing Centers Using M/G/m/m+r Queuing Systems," Ieee Transactions on Parallel and Distributed Systems, vol. 23, no. 5, pp. 936-943, 2012.

[6] K. Hwang, J. Dongarra, and G. Fox, Distributed and Cloud Computing: From Parallel Processing to the Internet of Things: Morgan Kaufmann, 2012.

[7] B. Javadi, P. Thulasiraman, and R. Buyya, "Cloud Resource Provisioning to Extend the Capacity of Local Resources in the Presence of Failures", Proc. 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), pp. 311-319, 2012.

[8] J. Jonsson, "A Robust Adaptive Metric for Deadline Assignment in Heterogeneous Distributed Real-Time Systems", Proc. 13th International Parallel Processing and 10th Symposium on Parallel and Distributed Processing (IPPS/SPDP), pp. 678-687, 1999.

[9] B. Kao and H. Garcia-molina, "Deadline Assignment in a Distributed Soft Real-Time System " IEEE Transactions on Parallel and Distributed Systems, vol. 8, no. 12, pp. 428--437, 1993.

[10] J. Lee, I. Shin, and A. Easwaran, "Convex Optimization Framework for Intermediate Deadline Assignment in Soft and Hard Real-Time Distributed Systems," Journal of Systems and Software, vol. 85, no. 10, pp. 2331-2339, 2012.

[11] K. Liu, J. J. Chen, Y. Yang, and H. Jin, "A Throughput Maximization Strategy for Scheduling Transaction-Intensive Workflows on SwinDeW-G," Concurrency and Computation: Practice and Experience, vol. 20, no. 15, pp. 1807-1820, 2008.

[12] 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 (BPM2008), vol. 5204, pp. 180-195, Milan, Italy, 2008.

[13] X. Liu, Z. Ni, J. Chen, and Y. Yang, "A Probabilistic Strategy for Temporal Constraint Management in Scientific Workflow Systems," Concurrency and Computation: Practice and Experience, vol. 23, no. 16, pp. 1893-1919, 2011.

[14] X. Liu, J. Chen, and Y. Yang, Temporal QoS Management in Scientific Cloud Workflow Systems: Elsevier, 2012.

[15] X. Liu, D. Yuan, G. Zhang, W. Li, D. Cao, Q. He, J. Chen, and Y. Yang, The Design of Cloud Workflow Systems: Springer, 2012.

[16] X. Liu, Y. Yang, D. Cao, and D. Yuan, "Selecting Checkpoints along the Time Line: A Novel Temporal Checkpoint Selection Strategy for Monitoring a Batch of Parallel Business Processes," Proc. 35th International Conference on Software Engineering (NIER Track), San Francisco, pp. 1281-1284, 2013.

[17] X. Liu, Y. Yang, Y. Jiang, and J. Chen, "Preventing Temporal Violations in Scientific Workflows: Where and How," IEEE Transactions on Software Engineering, vol. 37, no. 6, pp. 805-825, 2011.

[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. 492-509, 2011.

[19] D. Marincaa, P. Minet, and L. Georgeb, "Analysis of Deadline Assignment Methods in Distributed Real-Time Systems," Computer Communications 27 (2004), vol. 27, no. 2004, pp. 1412–1423, 2004.

[20] B. Ravindran, "Engineering Dynamic Real-Time Distributed Systems: Architecture, System Description Language and Middleware," IEEE Transactions on Software Engineering, vol. 28, no. 1, pp. 30-57, 2002.

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