Scheduling Time-Sensitive Multi-Tier Services with Probabilistic Performance Guarantee

Shuo Liu*, Soamar Homsi*, Ming Fan*, Shaolei Ren†, Gang Quan*, Shangping Ren‡
*Department of Electrical and Computer Engineering, Florida International University, Miami, FL, 33174
†School of Computing and Information Sciences, Florida International University, Miami, FL, 33199
‡Department of Computer Science, Illinois Institute of Technology, Chicago, IL, 60616
Emails: {sliu005, shoms001, mfan001, gang.quan}@fiu.edu, sren@cs.fiu.edu, ren@iit.edu

***Abstract*—Web applications grow tremendously in both scale and scope, the application patterns turn to be more and more sophisticated. It is important but challenging for service providers to lower the operational costs without degrading user experiences, especially in the case where a service provider's profit is closely related to the user experience (e.g. response time.) In this paper, we study the problem of efficiently scheduling multi-tier time sensitive applications on distributed computing platforms with respect to the user's Quality of Service (QoS) requirements. The efficiency refers to the QoS satisfaction with low average response times. The service provider must ensures that service requests be served successfully before end-to-end deadlines with certain probabilities. To solve this problem, we propose an approach to judiciously assign a deadline for each service tier. An application request is dropped if any one of its services misses its deadline. Our simulation results demonstrate that our approach can statistically guarantee the required QoS more efficiently than the other widely applied methods (e.g. acceptance control, first-come-first-serve, deterministic sub deadline assignment, etc.) irrespective of whether the resources are shared or not by multiple different applications.**

***Keywords*-multi-tier services; time-sensitive applications; sub deadline assignment; resource sharing; QoS statistical guarantee;**

I. Introduction

Web applications are drastically changing people's lives. From online banking to online gaming, from online storage to online computing, people's lives have been greatly facilitated. Meanwhile, here come the tough design problems for service providers.

As web applications grow tremendously in both scale and scope, the application patterns become more and more sophisticated. An application request coming from the Internet usually needs to go through multiple service tiers hosted in different machines at different locations. Different from single-tier applications, multi-tier applications have more intricate inter-tier interactions. A change of a request's execution in a tier may disturb its (or even other requests') final QoS guarantee.

In addition, there have been an increasing number of time sensitive applications, such as on-line gaming (e.g. Uncharted 3: Drake's Deception [1]) or other streaming multimedia (e.g. Adobe Media Server [2]) deployed on the web (e.g. Rackspace Cloud Media Hosting [3] and AWS [4].) Because of these time sensitive applications, stringent timing constraints are added to the processing infrastructure for desired user experience. Besides the timing constraints, in order to fully satisfy a client's QoS requirements a service provider must ensure that *adequate* requests can be served successfully in time.

In this paper, we study the problem of how to efficiently manage a set of multi-tier, time sensitive application requests with probabilistic guarantee of their end-to-end deadlines. Specifically, the QoS in this paper refers to a constraint described by the probability of dropping an application's request with respect to its end-to-end deadline. While satisfying the QoS, we intend to reduce the average response time for an application. As response time has become an important performance metric [5], and since the non-increasing time utility functions (TUFs) are widely employed [6] to analyze the relationship between a request's response time and its processing profit, scheduling approaches that have shorter response times are more desirable.

Different from the traditional dynamic resource provisioning approaches [7][8], we propose a stochastic approach that can judiciously prune the application requests on a given distributed platform to address this challenging problem for the situations with and without resource sharing. Previously, many similar studies employed acceptance control or random deletion to guarantee the end-to-end QoS satisfactions [9][10]. Instead of targeting on the potential mischievous requests, these methods randomly selected the requests to reject or remove. We propose a method that identifies the potential failure requests and terminates these requests. However, how to find the potential failure requests and when to remove them from the system are not trivial problems, especially in the situations where computing resources are shared among different applications.

In our approach, a sub deadline is associated with each service of a time sensitive application. A request is dropped if any of its services miss the associated sub deadline. The rationale of our approach is that when an application request is more likely to miss its end-to-end deadline, and thus is of no use to the system, it is better off to remove the request as early as possible. Dropping a request helps saving the precious computing resources and energy for requests that are more likely to be successfully fulfilled, which would most likely be wasted otherwise. However, requests dropping degrades the QoS of the system. Making the appropriate tradeoff is the

key to this problem. To this end, we transform the deadline assignment problem to a queueing problem with reneges [11] and develop an algorithm to determine sub deadlines for the services analytically in the situations with and without resource sharing. Our experimental results demonstrate that our approach is able to statistically guarantee the QoS requirements with higher efficiencies (i.e. achieving required completion ratios with shorter average response times.)

II. Related work

Extensive research has been conducted on the web application resource management problem. A significant part of the research has focused on cloud. Performance goals were achieved by dynamic resource provisioning in virtualized environments [7][8][12]. These approaches dynamically modulated the availability of the underlying computing facilities (e.g. increase/decrease the number of powered-on physical servers) according to the workload predictions or in response to the workload changes. The major difference between these approaches and what we present in this paper is that they assumed that computing facilities for cloud applications can be dynamically adjusted. However, we seek to invent an approach that is more general instead of only being limited for cloud applications. Without considering the elasticity provided by cloud, we intend to manage the executions of web application requests under a pre-defined (provisioned), fixed computing facilities.

There are also many other techniques proposed to manage the executions of application requests on a fixed computing infrastructure. As applications usually consist of a series of dependent services, such applications are usually modeled as direct acyclic graphs (DAG). Most of the work focused on the performance optimization in terms of reducing the average response time under constraints such as the processing budgets or costs. For example, Kamthe et al. [13], proposed a scheduling approach that accurately estimated the earliest start time of each service and then the requests execution sequence were optimized based on those earliest start times and thus the DAG makespan. Similarly, Tang et al. [14], developed a stochastic heterogeneous earliest finish time (SHEFT) scheduling approach to reduce the makespan of a DAG. Our approach is different from these approaches in that we are more focused on the statistical guarantee of the end-to-end deadline satisfactions instead of simply reducing the overall latency of an application.

To control the requests execution subject to their end-to-end deadlines or other timeliness requirements, acceptance control and queue length control via random removal are commonly used [9][15][10][16], especially when a system is under overloading situations. For example, Wang et al. [15] employed acceptance control to optimize a service provider's revenue with the least operational costs for multi-tier services in a virtualized environment. Liu et al. [10] designed an adaptive acceptance control method to optimize the performance for web applications by adjusting the queue lengths. Acceptance control and queue length control via random removal help service providers alleviate the system workload and thus can guarantee timeliness requirements for the accepted requests. However, using acceptance control and queue length control with random removal to achieve probabilistic guarantee can be a challenging problem.

Besides acceptance control and queue length control via random removal, sub deadline assignment is another popular approach for end-to-end deadline guarantee. For example, Hong et al. [17] introduced a technique to assign a sub deadline for each service in a DAG, and the end-to-end deadline can be guaranteed if all services can meet their sub deadlines. Yu et al. [18][19][20] proposed to assign a sub deadline to the service at each tier by dividing the overall end-to-end deadline proportionally to a request's minimum processing time at that tier. Similar idea was employed by Mao et al. in [21], with sub deadlines calculated according to the request's best or worst execution time in a service. There are two major problems with these approaches. First, the deadlines are assigned under the assumption of a priori knowledge of deterministic timing characteristics of the requests. Second, these approaches are highly heuristic and ensuring a probabilistic guarantee of the end-to-end deadlines remains a problem. In our research, we adopt a statistical approach for sub deadline assignment and formally prove that our approach can guarantee the end-to-end deadline in a probabilistic manner.

III. Preliminary

In this section we introduce our system model and formulate our problem formally.

A. Service model

We assume that our application architecture consists of m servers, i.e. $\mathcal{E} = \{E_1, E_2, ..., E_m\}$, each of which provides one dedicated service. The service time of each server, i.e. E_i, is independent from each other and follows an exponential distribution with the average processing time of $\frac{1}{\mu_i}$.

(a) A multi-tier services application.

(b) A queue-based view of an application.

Fig. 1. Application model.

There is a total of k applications $\mathcal{S} = \{S_1, S_2, ..., S_k\}$ in the system. We assume that the requests arrival pattern for each application follows the Poisson distribution, and each application is modeled with a 4-tuple, i.e. $S_i = \{\lambda_i, D_i, R_i, L_i\}$, where

- λ_i is the arrival rate of requests for S_i;
- D_i is the end-to-end deadline of S_i;
- R_i is the end-to-end deadline satisfaction ratio (aka. the completion ratio) for S_i;

- L_i contains an ordered list of servers that S_i needs to go through, i.e. $L_i = \{E_{i1}, E_{i2}, ..., E_{in}\}$, where $E_{ij} \in \mathcal{E}$.

The multi-tier services in an application and their queuing models are illustrated in Figure 1. Note that R_i and D_i together quantify the QoS requirements of S_i, and L_i represents the dependent services that S_i needs to go through. An instance of S_i, triggered by a corresponding service request, is considered successfully completed only if it goes through all the designated servers and finishes its execution before the end-to-end deadline. Otherwise, it is deemed as a failure. For different applications S_i and S_j, $L_i \cap L_j$ may or may not be $\emptyset$, indicating that different applications may or may not share the same servers.

B. Problem definition

We assume that, for each application $S_i = \{\lambda_i, D_i, R_i, L_i\}$, a sub deadline (e.g. d_{ij}) is assigned to each server (e.g. $E_{ij} \in L_i$) that S_i needs to go through. A request is dropped if any of its services miss the deadline. Removing a request from the queue helps save system resources for other requests that are more likely to be completed in time. On the other hand, such an action may lead to violating the QoS requirements as identified by the deadline miss ratios. The problem is then how to judiciously determine the sub deadline for each server. These deadlines are local sub deadlines and the interval between any two adjacent sub deadlines is used to indicate how long a request is allowed for being processed in that server.

With the system model and assumptions introduced above, we formulate our research problem as follows:

Problem 1: Given an application platform $\mathcal{E}$ and an application set $\mathcal{S}$, determine the sub deadline d_{ij} on each server (e.g. $E_{ij} \in L_i$) for application S_i such that no less than R_i percent of S_i requests can successfully meet their end-to-end deadline D_i.

IV. Sub deadline assignment

In this section, we present our approach of solving the sub deadline assignment problem. We first discuss our approach for a simple case in which an application has only two services. We then extend our approach to the case for an application with multiple services.

A. An application with two servers

We first consider the simple case of an application with only two servers. Consider the application $S = \{\lambda, D, R, L(E_1, E_2)\}$ shown in Figure 2. Initially, an application request arrives at E_1 following a Poisson process with a rate of λ. If a sub deadline $d_1 \leq D$ is assigned to E_1, not all requests can pass through E_1 due to the deadline violations. We assume that the entire system reaches the stable status and the renege probability is quite small in each server, thus, according to Burke's Theorem [22], the request arrivals for E_2 is assumed to be a Poisson process as well. Assuming that the requests are deleted after the first server at a probability of p_1, then, the arrival rate for the second server E_2 becomes

$$\lambda_2 = (1 - p_1)\lambda_1. \tag{1}$$

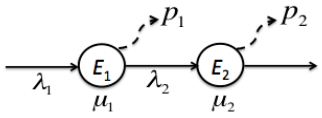


Fig. 2. A two-tier application with sub deadline assignment.

Similarly, when a sub deadline is assigned to E_2 (i.e. D in this case,) some requests are dropped at a probability of p_2 due to deadline violations. Therefore, the probability for a request to survive both E_1 and E_2 is $1 - (p_1 + (1 - p_1)p_2)$.

Our goal is to determine d_1 such that

$$1 - (p_1 + (1 - p_1)p_2) \geq R. \tag{2}$$

The key challenge, however, is to formulate the relations among parameters including $\lambda_1, \lambda_2, p_1, p_2, D$, and R. To this end, we employ the techniques developed for M/M/1 queue with renege [11] to solve our problem. M/M/1 queue with renege is a queueing model that depicts the impatient customers who leave the queue if not fully served within a giving time frame on a server. Specifically, let the arrival rate of requests on a server be λ and the processing rate be μ, and let a customer's maximum waiting length be τ (i.e. the interval between two adjacent sub deadlines in our scenario.) Then the probability that customers renege from the server can be formulated as

$$p = \frac{(1 - \rho)e^{\alpha(\rho - 1)}}{1 - \rho e^{\alpha(\rho - 1)}} \tag{3}$$

where $\rho = \lambda/\mu$ and $\alpha = \mu \cdot \tau$. Similarly, to calculate p_1 in our case, we have

$$p_1 = \frac{(1 - \frac{\lambda_1}{\mu_1})e^{d_1\mu_1(\frac{\lambda_1}{\mu_1} - 1)}}{1 - \frac{\lambda_1}{\mu_1}e^{d_1\mu_1(\frac{\lambda_1}{\mu_1} - 1)}} \tag{4}$$

Based on Equation (3), we calculate p_2 by plugging $\lambda = \lambda_2$ and τ_2 into Equation (3). However, to formulate the renege probability of E_2, we have to take the execution status on E_1 into consideration. Since the renege probability on each server will stay small (because of the completion ratio constraint,) we assume that the request arrivals of E_2 still follows a Poisson distribution. Then, another M/M/1 queue can be modeled for E_2. Therefore, let $d_2 = D$, we formulate p_2 as

$$p_2 = \frac{(1 - \frac{\lambda_2}{\mu_2})e^{(D - d_1)\mu_2(\frac{\lambda_2}{\mu_2} - 1)}}{1 - \frac{\lambda_2}{\mu_2}e^{(D - d_1)\mu_2(\frac{\lambda_2}{\mu_2} - 1)}}. \tag{5}$$

The feasible solution of d_1 satisfies Equations (2), (4), and (5). To identify the solution for d_1, we can use sophisticated numerical algorithms when necessary. For ease of presentation, Algorithm 1 employs a simple search method to find the solution of d_1. Note that Algorithm 1 does not always produce a feasible solution of d_1. When λ_1 is extremely large or when μ_1 or μ_2 is too small, there is no possible sub deadline assignment that is able to statistically guarantee the QoS requirements. On the other hand, if Algorithm 1 does produce

Algorithm 1 Sub deadline assignment for a single application with two servers.

1: **TwoTier**$(\lambda, D, R, E_1, E_2)$
2: **Input**: $\lambda, D, R, \mu_1, \mu_2$.
3: **Output**: The sub deadline for E_1, i.e. d_1.
4: $d_1^* = 0$;
5: **while** $d_1^* \leq D$ **do**
6: Calculate p_1 and p_2 based on Equations (4) and (5) using d_1^*;
7: **if** $p_1 + (1-p_1)p_2 \leq 1 - R$ **then**
8: $d_1 = d_1^*$;
9: return d_1;
10: **end if**
11: $d_1^* = d_1^* + \delta$, where δ is the minimum time interval;
12: **end while**
13: Output: ("No feasible solution for the given constraints!");
14: exit();

a solution, we can ensure the probability to successfully fulfill the requests without violating the end-to-end deadlines. This conclusion is summarized in the following theorem.

Theorem 1: Given a single application with two servers E_1 and E_2, i.e. $S = \{\lambda, D, R, E_1, E_2\}$, let the processing rates of the two servers be μ_1 and μ_2, respectively. Assume that the relative error when modeling the servers E_1 and E_2 using M/M/1 independent queues is small enough. Then the sub deadline derived from Algorithm 1 can guarantee the deadline meet ratio no smaller than R.

Proof: Algorithm 1 ensures that the proportion of the total dropped requests is no more than R (line 7). Also, since d_1 is found in such a way that Equations (2), (4) and (5) are all satisfied. According to [11], this ensures that all remaining requests can be processed before d_1 and D when going through the first and second server, respectively. ∎

In addition, we have made a number of interesting observations that are listed in the following theorem.

Theorem 2: For an application S and one of its servers, i.e. E_i, with processing rate of μ_i, let the corresponding request arrival rate be λ_i, the deadline be d_i, and the renege probability at E_i be p_i. Then,

- The larger the λ_i, the larger the p_i is;
- The larger the μ_i, the smaller the p_i is;
- The larger the d_i, the smaller the p_i is.

Proof: From Equation (3), we have

$$\begin{aligned}\frac{\partial p}{\partial \rho} = &\{[(1-\rho)e^{\alpha(\rho-1)}]'[1-\rho e^{\alpha(\rho-1)}]- \\ &[(1-\rho)e^{\alpha(\rho-1)}][1-\rho e^{\alpha(\rho-1)}]'\}/[1-\rho e^{\alpha(\rho-1)}]^2\end{aligned} \tag{6}$$

Now let

$$\begin{aligned}F(\rho) &= [(1-\rho)e^{\alpha(\rho-1)}]'[1-\rho e^{\alpha(\rho-1)}] - \\ &\quad [(1-\rho)e^{\alpha(\rho-1)}][1-\rho e^{\alpha(\rho-1)}]'\end{aligned}$$

Simplify $F(\rho)$ we have

$$\begin{aligned}F(\rho) &= -e^{\alpha(\rho-1)} + [e^{\alpha(\rho-1)}]^2 + (1-\rho)\alpha e^{\alpha(\rho-1)} \\ &= e^{\alpha(\rho-1)}[-1 + e^{\alpha(\rho-1)} + (1-\rho)\alpha]\end{aligned}$$

Let $x = (1-\rho)\alpha$, we have

$$F(\rho) = e^{-x}(-1 + e^{-x} + x) \tag{7}$$

According to [23], as long as $x > -1$, $-1 + e^{-x} + x > 0$. Since $x = (1-\rho)\alpha$, and $\rho < 1$, then x is always larger than -1 and Equation (7) is positive. Therefore, from Equation (6) we have

$$\frac{\partial p}{\partial \rho} > 0. \tag{8}$$

To prove the first two bullet points in the theorem, we only need to note that

$$\frac{\partial p}{\partial \lambda_i} = \frac{\partial p}{\partial \rho} \times \frac{\partial \rho}{\partial \lambda_i} > 0. \tag{9}$$

and

$$\frac{\partial p}{\partial \mu_i} = \frac{\partial p}{\partial \rho} \times \frac{\partial \rho}{\partial \mu_i} < 0. \tag{10}$$

To prove the third bullet point in this theorem, we have

$$\begin{aligned}\frac{\partial p}{\partial d} =&\{[(1-\rho)e^{\mu\cdot d\cdot(\rho-1)}]'[1-\rho e^{\mu\cdot d\cdot(\rho-1)}] \\ &[(1-\rho)e^{\mu\cdot d\cdot(\rho-1)}][1-\rho e^{\mu\cdot d\cdot(\rho-1)}]'\} \\ &/[1-\rho e^{\mu\cdot d\cdot(\rho-1)}]^2 \\ =&\{[(1-\rho)\mu(\rho-1)e^{\mu\cdot d\cdot(\rho-1)}][1-\rho e^{\mu\cdot d\cdot(\rho-1)}] \\ &-\rho\mu(1-\rho)^2[e^{\mu\cdot d\cdot(\rho-1)}]\}/[1-\rho e^{\mu\cdot d\cdot(\rho-1)}]^2 \\ =&-\mu(1-\rho)^2 e^{\mu\cdot d\cdot(\rho-1)}/[1-\rho e^{\mu\cdot d\cdot(\rho-1)}]^2 \\ <&0\end{aligned} \tag{11}$$

∎

B. An application with multiple servers

We now extend our approach to the case of single application with multiple servers. Similarly, we assume that the system has reached the stable state and each server can be modeled as a M/M/1 queue.

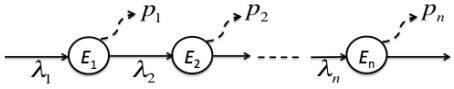


Fig. 3. Application model with more than two servers.

Take the three-tier application shown in Figure 3 as an example. After the first server, assume that there will be p_1 percent of all requests deleted due to the sub deadline violations. The arrival rate for the second server is thus $\lambda_2 = (1-p_1)\lambda_1$. After the second server, another p_2 percent of the requests among the λ_2 are discarded. Then, only $\lambda_3 = (1 - p_1 - (1-p_1)p_2)\lambda_1$ are left for the third server. Equivalently, the arrival rate of the requests for tier $n+1$ from tier n can be formulated as

$$\lambda_{n+1} = (1-p_n) \cdot \lambda_n. \tag{12}$$

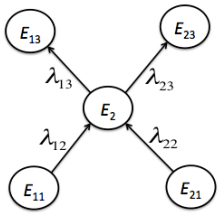


Fig. 4. Applications with shared tiers.

The total ratio of requests that have been removed through n ($n \geq 2$) servers can be formulated as

$$\lambda_1 \cdot \{p_1 + \sum_{t_i=2}^{n} (\Pi_{t_j=1}^{t_i-1}(1-p_{t_j}) \cdot p_{t_i})\} \quad (13)$$

Provided that all the remaining requests in the system are able to meet their end-to-end deadlines, in order to satisfy the probabilistic guarantee of R, we only need to require that

$$\lambda_1 \cdot \{p_1 + \sum_{t_i=2}^{n} (\Pi_{t_j=1}^{t_i-1}(1-p_{t_j}) \cdot p_{t_i})\} \leq (1-R) \cdot \lambda_1 \quad (14)$$

For an application consists of n servers, based on Equation (3) we have

$$p_n = \frac{(1-\frac{\lambda_n}{\mu_n}) e^{(d_n - d_{n-1})\mu_n(\frac{\lambda_n}{\mu_n}-1)}}{1-\frac{\lambda_n}{\mu_n} e^{(d_n-d_{n-1})\mu_n(\frac{\lambda_n}{\mu_n}-1)}}. \quad (15)$$

Note that in the equation above, p_n is the request removal probability on E_n, and $(d_n - d_{n-1})$ is the allowed processing time of a request on server E_n. We can then formulate the equations recursively for the servers $E_1, E_2, ..., E_n$. The $d_1, d_2, ..., d_n$ that satisfy Equation (14) are the sub deadlines for each server.

C. Multiple applications with shared servers

Due to issues such as software licenses and resource/cost constraints, different applications usually need to go to the same server for services. We now discuss how to assign sub deadlines for multiple applications in a shared environment.

In this scenario, some tiers that belong to different applications have to share the same function unit (hosted on the same server). The sub deadline calculation is more intricate here. As shown in Figure 4, two applications share the same second service tier. Each application has its own QoS required pair $\{D_1, R_1\}$, and $\{D_2, R_2\}$. Algorithm 1 cannot be simply applied to calculate the sub deadlines for the applications. We employ statistical multiplexing to describe the sharing situation. However, it is worth pointing out that our method is not limited to the statistical multiplexing. Different application requests on the same server share the same processing queue. Since we assume that the arrival rates of both applications on the shared server follow Poisson distribution, the combined workload arrival is still a Poisson process. The processing time of each request on the shared server is exponentially distributed. Therefore, the shared server can be treated as a M/M/1 queue as well. The response time relationship between the two applications and the combined workload can be formulated as following [24]:

$$\frac{1}{U-(\lambda_{12}+\lambda_{22})} = \frac{1}{\mu_{12}^* - \lambda_{12}} = \frac{1}{\mu_{22}^* - \lambda_{22}} \quad (16)$$

where U is the shared server's processing rate. λ_{12} and λ_{22} are the arrival rates of the two applications at the second tier. μ_{12}^* and μ_{22}^* are the corresponding simulated processing rates of the two applications under the situation in which applications are processed independently with no resource sharing. The formulations of μ_{12}^* and μ_{22}^* can be derived through simple transformations of Equation (16).

$$\begin{aligned} \mu_{12}^* &= U - \lambda_{22} \\ \mu_{22}^* &= U - \lambda_{12} \end{aligned} \quad (17)$$

With the derived μ_{12}^* and μ_{22}^* in the shared environment, we can apply Equation (3) to obtain the renege probabilities for each of the applications. Due to the fact that they share the same service tier, the sub deadline calculations for the two applications are interdependent. We apply an iterative approach to derive the sub deadlines for each of the applications.

We first assume that no renege happens for application S_2. Then, application S_1 can be treated as a serialized application with a middle tier shared with application S_2. With Equation (17), we use the method introduced in the previous section (Algorithm 1) to calculate the sub deadlines for S_1. Once S_1's sub deadlines are available (e.g. d_{11}), replace the λ_{12} in Equation (17) with the derived arrival rate of application S_1 (e.g. λ_{12}^*) in the shared tier using the calculated S_1's sub deadlines (e.g. d_{11}). Then, S_2's sub deadlines are calculated following the same method. These two sub deadline calculation steps for S_1 and S_2 compose an iteration. After each iteration, sub deadline calculations for S_1 and S_2 are conducted again based on the results derived from the previous iteration. The whole procedure stops when the maximum sub deadline difference between two adjacent iterations of both applications are smaller than a pre-defined threshold T (or reach a maximum iteration number.) The details of our approach for multiple applications with shared service tiers is summarized in Algorithm 2.

V. Simulation Study

In this section, we use simulations with synthetic parameters to investigate the performance of our approach.

We compared our high efficient sub deadline approach ("HESD") with several widely used methods. Acceptance control ("ac"), random deletion ("rd") and First-Come-First-Server ("fifo") have no sub deadline constraints. Then, a local sub deadline calculation method "det" (similar to [17] and [18]) deterministically assigns sub deadlines to servers. Finally, a global sub deadline assignment method "pdf" based on request processing time distribution is employed for comparisons. The summary of the methods is listed as follows:

Algorithm 2 Sub deadline assignment for multiple applications with shared service tiers.

K number of N-tier applications running on M number of servers
Input: $\lambda_i, D_i, R_i, E_1, E_2, \ldots, E_m$. $\forall i \in K$, $\forall m \in M$.
Output: The sub deadline for applications S_1, S_2, ..., S_N.
for $(i = 1; i \leq K; i++)$ **do**
  for $(j = 1; j \leq N; j++)$ **do**
    $PreSD_{ij} = 0$;
  end for
end for
Initialize $Stop$ to 0.
Initialize $Loop\_num$ to 0.
while (!Stop) **do**
  for $(i = 1; i \leq K; i++)$ **do**
    $Loop\_num = Loop\_num + 1$;
    Assume no renege for applications S_{i+1}, S_{i+2}, $\ldots S_N$;
    Application $S_1, \ldots, S_{i-1}$ have sub deadlines calculated in the previous loops;
    Apply Algorithm 1 for application S_i to get SD_{ij};
  end for
  if $(max[|PreSD_{ij} - SD_{ij}|] < T, \forall i \in K, \forall j \in N)$ && $(Loop\_num <= Max\_loop\_num)$ **then**
    Set Stop to 1;
  end if
  $PreSD_{ij} = SD_{ij}$;
end while

- **ac**: In the acceptance control method, based on the available computing capacity, a maximum acceptable request arrival rate is provided in order to guarantee that R percent of the accepted requests can be successfully completed before their end-to-end deadlines. If a service request is accepted, it will be processed until it is fully fulfilled.
- **rd**: In the random deletion method, all the request arrivals are accepted into the service chain. However, a bottleneck service tier will be identified. Based on the capacity of the bottleneck tier and the QoS required pair {D, R}, a correspondent amount of requests will be removed randomly from the bottleneck tier.
- **fifo**: The first come first serve method processes each request till completion according to their arrival order.
- **det**: In the deterministic local sub deadline assignment method, each server is assigned a local deadline that is proportional to the server's average response time.
- **pdf**: This is a method that applies the request processing time probability density function (PDF) on each server to find out how likely that the server's follow-up processing can meet the request's end-to-end deadline [25]. We extend their method to a global sub deadline assignment method. The extended version calculates the sub deadlines backward from the exiting point of the application. Thereby, the sub deadline calculated for each server is actually the latest time instance to statistically guarantee that the follow-up servers can finish the request's processing before the request's pre-defined end-to-end deadline.

In order to show the applicability of our method, we implement FIFO and processor sharing (PS) scheduling policies in our simulation with the sub deadline constraints derived by our method.

A. Single application without shared resources

We tested our method based on a three-tier application. Each tier was assumed to be hosted in an isolated processing unit. No resource or service sharing existed among the tiers. The parameters of the three-tier application were randomly generated. The processing rates of the servers were set to be 120, 115, and 110 requests/second, respectively. The execution times of the requests on the different servers were randomly generated according to each server's processing rate, and the execution times followed exponential distributions. The impacts of different arrival rates and processing rates on the QoS satisfactions were studied. We ran 200000 requests of each application for 5 rounds to average the randomness in each method.

1) Arrival rate effects: We first analyzed the effects of different request arrival rates on the QoS satisfaction.

We set the end-to-end deadline to be $0.4s$ with a completion ratio requirement of 90%, and gradually changed the arrival rate from 99 to 107 request/second with 2 request/second intervals.

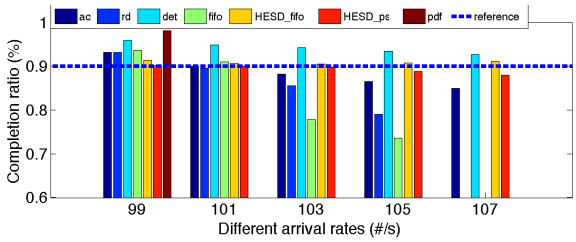


Fig. 5. Completion ratio comparison.

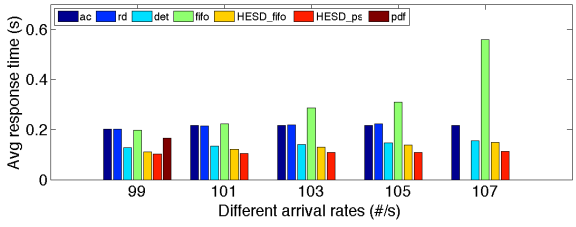


Fig. 6. Average response time comparison.

Figure 5 shows the completion ratios achieved by the methods at different arrival rates. Intuitively, the higher the arrival rate, the worse the completion ratio for most of the methods since the queues are getting more congested with the increasing number of requests. Every request has to experience

longer delay in each server in order to be fully served. The results shown in Figure 6 clearly supports our inference.

Our proposed method "HESD_fifo" had the lowest average response times while guaranteeing the required completion ratio. Since it is able to statistically guarantee the QoS required pair, its completion ratio kept stable around the required value through out the entire test cases. "HESD_ps" had similar results as "HESD_fifo" when the arrival rate was low. As the arrival rate got higher, its performance degraded slightly. It is interesting to observe that even though "pdf" satisfied the QoS required pair in the first testing case, its derived sub deadlines had less effect as those calculated via our proposed method. The number of completed requests was much larger than the one required to meet the QoS and resulted in significantly higher average response times. Additionally, the "pdf" method is not flexible to the parameter changes. When the arrival rate grew higher, "pdf" struggled and failed to provide sub deadlines for the application. The performance of "det" was surprisingly good in this test. However, it is not able to statistically guarantee the QoS required pair as the resources become stringent. On the contrary, our method is capable of indicating the potential infeasibility when Algorithm 1 could not provide corresponding sub deadlines. "ac" and "rd" satisfied the QoS requirements while the arrival rates stayed low. Nevertheless, as the arrival rate got higher (e.g. 107 requests/second in our case,) in order to guarantee that 90% of the accepted requests could finish before the end-to-end deadline, only around 85% original requests could be accepted for "ac" and 0% for "rd".

2) processing rate effects: We further studied the processing rate effects on different methods.

The end-to-end deadline was set to be $0.5s$ with a completion ratio requirement of 90%. Arrival rate was 90 requests/second. The processing rates were set to be 120, 115, 110 requests/second initially, and were gradually degraded from 100% to 92% of the full capacities, with 2% degrading intervals.

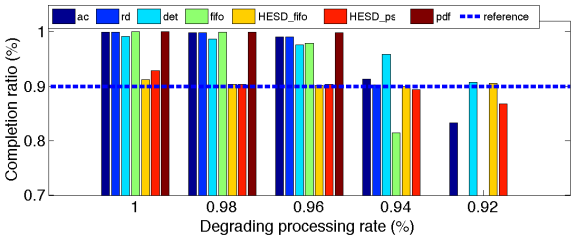


Fig. 7. Completion ratio comparison.

Figures 7 and 8 depict the completion ratios and average response times, respectively. With the full processing capacities, all methods were able to satisfy the QoS requirements. As the processing rate degraded, completion ratios achieved by all the methods except "HESD" dropped. The changes in processing rate did not have significant impacts on the completion ratio in our proposed method ("HESD_ps" degraded slightly as the processing capacity dropped.) However, we can easily tell

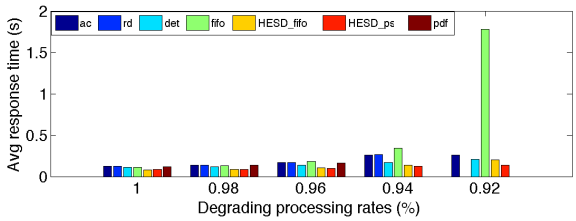


Fig. 8. Average response time comparison.

from the figure that the lower the processing rate, the larger the average response times for all the methods. "pdf" failed to provide sub deadlines for the last two settings.

B. Multiple applications with shared resources

We finally studied the performance of our proposed method in a shared environment based on the architecture shown in Figure 9. Since "ac" and "rd" are similar, "det" has no statistical guarantee, and "fifo" is always the worst on the average response time, we did not include them into the comparisons.

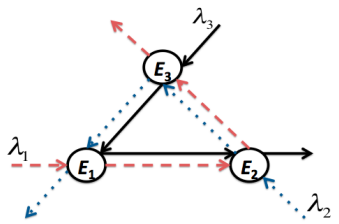
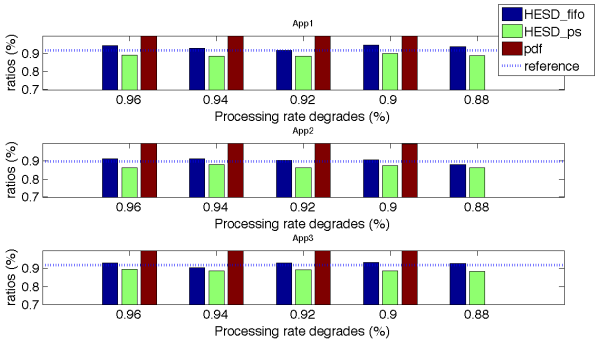


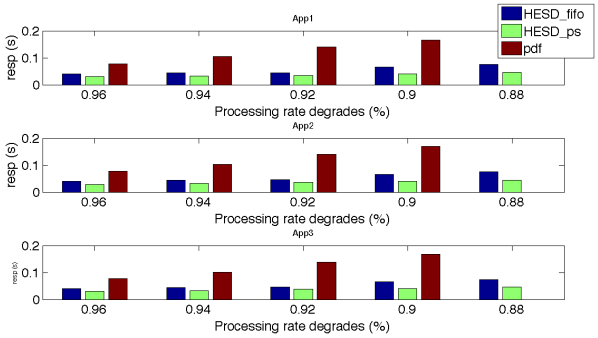
Fig. 9. Applications with shared servers.

As shown in Figure 9, three different applications share three services. The requests of the first application λ_1 enter the system from node E_1. After going through nodes E_2 and E_3, the successfully processed requests leave the system from E_3. Similar to the first application, the requests of the second (third) application enter the system from node E_2 (E_3), and leave the system from nodes E_1 (E_2). We set the processing rate of the three nodes at 350, 360, 370 requests/second, and gradually degraded the processing capacities of the three nodes simultaneously from 96% to 88% with a degrading interval of 2%. The three applications had the same arrival rate at 100 requests/second. The QoS required pairs are $\{92\%, 0.5s\}$, $\{90\%, 0.4s\}$, and $\{92\%, 0.5s\}$, respectively. 10000 number of requests of each application were tested, and the comparisons of the completion ratios and average response times are illustrated in Figure 10.

From 10(a) we can tell that the "pdf" method had similar performance as in the single application situations. The sub deadlines derived were relatively relaxed compared to those calculated by "HESD". Not many requests were removed in "pdf". When the processing capacities degraded to 88% in our case, "pdf" failed to assign sub deadlines to the applications subject to the desired QoS required pairs. On the contrary, the completion ratios of "HESD_fifo" kept stable around the



(a) Completion ratio comparison.



(b) Average response time comparison.

Fig. 10. Comparison for multiple applications with shared resources.

required levels at different parameter settings and its average response times were obviously smaller than "pdf" (Figure 10(b).) "HESD_ps" had a stable completion ratio as well but failed to reach the required amount (around 4% lower) even though it had the lowest average response times. The average response times reflected a raising trend as the processing capacities degraded in all the methods.

VI. Conclusion

In this paper, we propose a sub deadline assignment approach for applications with and without resource sharing. We model the multi-tier time-sensitive applications as $M/M/1$ queues with reneges. It is able to statistically guarantee the QoS requirements with high efficiencies by judiciously discarding mischievous failure requests in early stages. Precious computing resources can be used more effectively and efficiently by the promising requests. Our proposed method has the potential to increase the net profit for a service provider.

VII. Acknowledgement

This work is supported in part by NSF under projects CNS-1423137 and CNS-1018108.

References

[1] Aws case study: Naughty dog. http://aws.amazon.com/solutions/case-studies/naughty-dog/.
[2] Adobe media server. http://www.adobe.com/products/amazon-web-services.html.
[3] Rackspace cloud media hosting. http://www.rackspace.com/cloud/media/.
[4] Amazon web services. http://aws.amazon.com.
[5] Applications performance equals response time, not resource utilization. http://www.virtualizationpractice.com/applications-performance-equals-response-time-not-resource-utilization-9916/.
[6] Young Choon Lee, Chen Wang, Albert Y. Zomaya, and Bing Bing Zhou. Profit-driven service request scheduling in clouds. In *CCGRID*, pages 15–24. IEEE, 2010.
[7] Shengquan Wang, Waqaas Munawar, Jun Liu, Jian-Jia Chen, and Xue Liu. Power-saving design for server farms with response time percentile guarantees. In Marco Di Natale, editor, *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 273–284. IEEE, 2012.
[8] Palden Lama and Xiaobo Zhou. Efficient server provisioning with end-to-end delay guarantee on multi-tier clusters. In *IWQoS*, pages 1–9. IEEE, 2009.
[9] Kleopatra Konstanteli, Tommaso Cucinotta, Konstantinos Psychas, and Theodora A. Varvarigou. Admission control for elastic cloud services. In Rong Chang, editor, *IEEE CLOUD*, pages 41–48. IEEE, 2012.
[10] Xue Liu, J. Heo, Lui Sha, and Xiaoyun Zhu. Adaptive control of multi-tiered web applications using queueing predictor. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pages 106–114, 2006.
[11] D. Y. Barrer. Queuing with impatient customers and ordered service. *Operations Research*, 5(5):pp. 650–656, 1957.
[12] Minghong Lin, Adam Wierman, Lachlan L. H. Andrew, and Eno Thereska. Dynamic right-sizing for power-proportional data centers. In *INFOCOM*, pages 1098–1106. IEEE, 2011.
[13] Ankur Kamthe and Soo-Young Lee. A stochastic approach to estimating earliest start times of nodes for scheduling dags on heterogeneous distributed computing systems. *Cluster Computing*, 14(4):377–395, 2011.
[14] Xiaoyong Tang, Kenli Li, Guiping Liao, Kui Fang, and Fan Wu. A stochastic scheduling algorithm for precedence constrained tasks on grid. *Future Gener. Comput. Syst.*, 27(8):1083–1091, October 2011.
[15] Xiaoying Wang, DongJun Lan, X. Fang, Meng Ye, and Ying Chen. A resource management framework for multi-tier service delivery in autonomic virtualized environments. In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, pages 310–316, 2008.
[16] S. Muppala and Xiaobo Zhou. Cosac: Coordinated session-based admission control for multi-tier internet applications. In *Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th Internatonal Conference on*, pages 1–6, 2009.
[17] Shengyan Hong, T. Chantem, and X.S. Hu. Meeting end-to-end deadlines through distributed local deadline assignments. In *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd*, pages 183 –192, 29 2011-dec. 2 2011.
[18] Jia Yu and Rajkumar Buyya. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Sci. Program.*, 14(3,4):217–230, December 2006.
[19] Jia Yu, R. Buyya, and Chen Khong Tham. Cost-based scheduling of scientific workflow applications on utility grids. In *e-Science and Grid Computing, 2005. First International Conference on*, pages 8 pp. –147, july 2005.
[20] Jia Yu and R. Buyya. A budget constrained scheduling of workflow applications on utility grids using genetic algorithms. In *Workflows in Support of Large-Scale Science, 2006. WORKS '06. Workshop on*, pages 1 –10, june 2006.
[21] Ming Mao and Marty Humphrey. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 49:1–49:12, New York, NY, USA, 2011. ACM.
[22] T.G. Robertazzi. *Computer Networks and Systems: Queueing Theory and Performance Evaluation*. Telecommunication networks and computer systems. Springer, 2000.
[23] V. Ramaswami Aiyar. On the exponential inequalities and the exponential function. *The Mathematical Gazette*, 4(61):pp. 8–12, 1907.
[24] Moshe Zukerman. Introduction to queueing theory and stochastic teletraffic models. *arXiv preprint arXiv:1307.2968*, 2013.
[25] M. Zivkovic, J.W. Bosman, J. L. Van den Berg, R.D. Van der Mei, H.B. Meeuwissen, and R. Nunez-Queija. Dynamic profit optimization of composite web services with slas. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1–6, Dec 2011.