Resos: Dynamic Replication in Service-oriented Systems

Mathias Björkqvist  
IBM Research Zurich Lab  
Rüschlikon, Switzerland

Lydia Y. Chen  
IBM Research Zurich Lab  
Rüschlikon, Switzerland

Walter Binder  
University of Lugano  
Lugano, Switzerland

Abstract—Service-oriented systems, consisting of multiple services and service composition execution engines (CEEs), are commonly deployed to deliver requested web services. As the workloads of applications fluctuate over multiple time scales, it is economical to autonomically and dynamically adjust system capacity, i.e., the number of replicas for various services and CEE. In this paper, we propose a novel replica provisioning policy, Resos, which adjusts the number of CEEs and services replicas periodically based on the predicted workloads and the target effective utilization values. In particular, Resos models the workflow balance and dependency between CEE and service replicas by estimating the blocking probability at CEE replicas. Moreover, we also derive the analytical bounds of CEE effective replicas by estimating the blocking probability at CEE replicas. We evaluate Resos on a simulated service-oriented system, which hosts CEE and service replicas on multi-threaded servers. The workload evaluated is derived from utilization traces collected from production systems. Through simulation, we demonstrate that Resos can affectively reduce the replica consumption while maintaining target performance metrics, i.e., utilization.

I. INTRODUCTION

Service-oriented systems are commonly composed of distributed web services [1], [11]. Applications’ requests, consisting of multiple invocations of web services, show a strong time varying behavior, e.g., time of day and day of the week effects [2], [5], [14], [15]. Such systems process requests by invoking corresponding service compositions, which are often represented as business processes or as workflows of services and which are typically deployed upon startup of the system [1]. To maintain the target service level agreement (SLA) and continuous availability of services, multiple replicas of resources need to be deployed. This includes service nodes that execute the requests, and composition execution engines (CEEs), which are dedicated engines that invoke the corresponding services.

Providers of service-oriented systems aim at delivering satisfactory performance in a cost effective manner. On one hand, the operational cost is proportional to the number of CEEs and service replicas deployed. On the other hand, system performance, e.g., response time and resource utilization, hinges on the provisioning of replicas in processing time-varying requests and the balancing of the load across replicas. Related studies [14], [20] show that striking a good balance between conflicting objectives, i.e., operational cost and performance, is not an easy task, especially in multi-tier systems. Statically providing a maximum number of replicas to CEEs and services may guarantee the performance at a high operational cost, whereas and unbalanced loads and under-provisioned CEE and service replicas could lead to a significant performance degradation. Dynamically and accurately adjusting service capacities, [5], [9] i.e., the number of active CEE and service replicas, depending on clients’ workloads, has been shown to be effective in solving the dilemma of balancing between performance targets and operational cost.

Various service replication strategies [6], [13], [21], [22] have been developed for fault tolerance service systems; however, only the replication of atomic services has been considered and the optimal number of replicas has not been addressed. To optimize SLA and operational cost, the optimal provisioning of service replicas has mainly been shown in the context of simple single-tier web hosting systems [9], i.e., clients send requests directly to services. For multi-tier web hosting systems, most existing studies [14], [20] design replication policies independently for each tier. In reality, the provisioning of CEE replicas depends on the performance of the second-tier service layer, due to the blocking of I/O which is a result of the processing of consecutive service invocations within a composition. The performance of the service replicas depends on the invocations dispatched by CEE replicas and the corresponding load balancing among service replicas. It is very challenging to dynamically provide resources in systems with multiple tiers, i.e., CEE and service tiers, which encounter time-varying and -correlated workloads.

In this paper, we consider a service-oriented system consisting of multiple, distributed replicas of CEEs and different services. To minimize the operational cost, we develop a novel replication manager, Resos, which dynamically and periodically adjusts the number of active CEE and service replicas. In each control window, Resos aims to maintain the target utilization for CEEs and service replicas by monitoring workload and performance statistics. In particular, we estimate the nominal and effective utilization of CEEs, which explicitly factors in inter-dependency among CEEs and services using the derived non-blocking probability at CEEs. To study the

1In this paper, the term “composition” refers to any composition of Web services, and the term “composition execution engine (CEE)” refers to middleware for executing compositions, such as BPEL engines [3].
impact of load balancing among service replicas on Resos, we consider load-aware and -oblivious balancing schemes in selecting service replicas at the CEEs. Our simulation results show that Resos achieves cost effective provisioning of replicas, whose effective utilization is well maintained at the target value and which deliver satisfactory end-to-end response time.

The contributions of this paper are threefold; first, our model and analysis of service-oriented systems capture the key system features: time-varying workloads, parallelism of replicas, i.e., thread pools, and the dependency between CEEs and service replicas. Second, we develop a novel replication manager, Resos, which considers the aforementioned features and dynamically controls replicas of CEEs and services based on the monitored workload and performance metrics. Third, we provide bounding analysis on effective and nominal utilization for CEEs, which essentially need to be kept less utilized than the service replicas by a factor of the non-blocking probability of the CEE threads.

This paper is organized as follows: The system architecture is explained in Section II. The service replication manager, Resos, and service selection policy are described in Section III and Section IV respectively. Section V contains the experimental results. Related studies are summarized in Section VI. Section VII concludes this paper.

II. SYSTEM OVERVIEW

In this paper we consider a service-oriented system as depicted in Figure 1. Service compositions are deployed in the CEE and exposed through service interfaces to clients, who belong to various applications. When a service composition is invoked by a client, the CEE creates an instance of the composition and executes it. During the execution, atomic services are invoked. We assume a service provider that hosts both the service compositions (in a CEE) and the atomic services that are invoked when compositions are executed. We assume that client do not directly invoke the atomic services; clients only invoke the exposed service compositions. In our model, the CEE and the atomic services can be replicated.

A. Client Requests and Composite Services

For each client service composition deployed in the CEE, we assume that the client requests generated from different applications may have disparate service compositions and workload characteristics (i.e., time-varying arrival rates). In this paper, we consider only sequential service compositions where atomic services are invoked in a given order. Here we do not model different workflow patterns [16] such as parallel split. For example, consider a system with two atomic services, denoted by $S_1$ and $S_2$. Two possible service compositions are $\{S_1, S_2\}$ and $\{S_2, S_1\}$. For the first composition, $S_1$ is first invoked and then $S_2$, whereas $S_2$ is invoked twice consecutively in the second composition.

In particular, we assume that client requests are generated from different applications. A composition request from application $a$ is defined by the service invocation set, $\omega_a$, consisting of the sequence of service invocations. For example, $\Omega_a = \{S_i, S_j\}$ represents that a request from application $a$ has two sequential service $i$ invocations.

B. Atomic Services

The system hosts $i$ types of atomic services, subscripted by $i \in \{1 ... I\}$. There may be multiple replicas, $n_i$, for each service type. All service types are considered stateless, i.e., for each invocation of an atomic service by the CEE, a different replica may be bound. Each replica has a queue for incoming requests (i.e., service invocations by the CEE) and maintains a thread-pool of fixed size, $t_i$, for processing these requests. Concurrent service invocations can be processed in parallel as long as there are threads available. That is, sequential code sections in service replicas are not modeled.

We model each service replica as a queueing system with one queue and multiple servers, each of which represent a single core/thread. An active replica processes service invocation requests sent by a CEE in FCFS manner, and the service $i$ execution time per thread is random variable with mean $d_i$. The response time of a service invocation is the sum of the queuing time and the execution time. We denote the average response time of service $i$ by $W_i$.

C. CEEs

There are $n_c$ CEE replicas, each of which is a distributed queueing system. A CEE replica queues incoming client requests that are then processed in FCFS order. We let the average queuing time at a CEE replica be $Q_c$. The CEE has a thread-pool of fixed size, $t_c$, to execute client requests in a parallel fashion. Each CEE thread processes one requests after the other, executing the corresponding service composition. For an invocation of service $i$, each thread selects a replica of service $i$, according to one of the load balancing schemes described in Section IV. The average CEE execution time to process a service invocation is assumed $d_c$. The invocations of atomic services are handled with blocking I/O: a single thread is used for the entire execution of an instance of a service composition, and this thread will block while waiting for the results of invoked atomic services. Since each instance of a service composition is executed sequentially by a single thread, we model only sequential compositions (see above).

D. Replication Manager: Resos

The replication manager, referred as to Resos, determines the number of CEE replicas, $n_c(t)$, and the number of service

![Fig. 1. Architecture overview](image-url)
replicas for each service type, \( n_i(t) \), in discrete time windows of fixed length. We assume that each replica is deployed on a separate machine (i.e., resource contention between replicas on the same node need not be considered in this simplified model). In total, the provider has \( N \) machines to host all replicas. For all the windows, \( n_i(t) + \sum n_s(t) \leq N \). The replication manager keeps at least one replica for each service type, i.e., \( n_i(t) \geq 1 \), and for CEE, i.e., \( n_c(t) \geq 1 \).

CEE and service replicas can be activated or deactivated in slotted windows by Resos. When a replica is deactivated, it receives no more requests (client requests in the case of a CEE replica, service invocations from a CEE in the case of a service replica), but it still needs to complete the processing of all pending requests in its queue. We assume it takes negligible warmup time for a replica before starting to process the incoming requests.

E. Average End-to-end Request Response Time, \( R_a \)

In summary, a client from application \( a \) requests a service composition, \( \Omega_a \), consisting of multiple service invocations. The average end-to-end response time of an application request, \( R_a \), is the summation of (1) the queue time at a CEE replica, \( Q_c \), (2) the CEE execution times \((\Omega_a)_s c \), and (3) the response time of all service invocations composed in \( \Omega_a \). Thus, one can obtain

\[
R_a = Q_c + |\Omega_a| d_c + \sum_{i \in \Omega_a} W_{i},
\]

where \(|\Omega_a|\) denotes the cardinality of \( \Omega_a \), i.e., the number of invocations in a service composition of application \( a \). Herein, we assume the network time is negligible compared to the processing time and queuing time at CEEs and service replicas.

III. RESOS: OPTIMIZING PERFORMANCE OF CEE AND SERVICE REPLICAS

System utilization has commonly been used as a performance metric for designing resource provisioning policies [5], [17]. Typically, the target utilization is purposely kept below the maximum capacity, e.g. 80%, for handling variations in the workloads [12]. Certain load balancing algorithms can be very effective in reducing variance of workloads and greatly enhance application response times [4], given the same levels of resource provisioning and system utilization, especially when the system is moderately loaded. Combining both observations, we propose a hierarchical solution to attain a cost-performance effective service system: (1) coarse-grained CEE and service replica provisioning by Resos and (2) fine-grained load balancing algorithms among available service replicas.

A. Monitoring and Predicting Workloads

To design a replica provisioning policy, the very first step is to monitor and further predict the workloads [5], [12], [14]. As there are two distinct tiers in our system, CEEs and services, their workloads characteristics need to be monitored separately. At the CEE tier, we focus on request rates of each application, whereas at the service tier we collect statistics of total invocations rates of each service.

We let \( \lambda_a \) be the request rate of application \( a \). The total request rate received by CEEs is the summation of all applications, i.e., \( \lambda = \sum \lambda_a \). We denote the invocation rate of requests received for service \( i \) replicas by \( \lambda_i \). As an application request consists of various and multiple service invocations, the total request rate is less than the total service innovation rates, i.e., \( \sum \lambda_i \lambda_a \ll \sum \lambda_a \). Note that \( \lambda \) and \( \lambda_a \) fluctuate in multiple time scales, and so does \( \lambda_i \).

Resos monitors the request rates of all applications and invocation rates of all services for all control windows. At the beginning of the control window, Resos obtains the estimate of \( \lambda_i(t) \), and \( \lambda_a(t) \), using historical statistics. In particular, Resos uses a simple last value prediction, i.e., using the arrival rate of the previous control window,

\[
\hat{\lambda}_i(t) = \lambda_i(t-1) \quad \hat{\lambda}_a(t) = \lambda_a(t-1).
\]

B. Controlling Replicas

Due to the blocking I/O in CEE threads, we consider two types of utilization for CEE: nominal and effective. The former computes the fraction of time CEE threads are busy processing compositions, whereas the latter computes the fraction of time CEE threads are busy or blocked waiting for service invocation requests to return. For service replicas, the effective and nominal utilization are equivalent. Resos aims to maintain the effective utilization of active CEE and services replicas at the target values, \( U^* \). In the following, we first derive the effective utilization and then obtain the replica control policy for services and CEEs, respectively.
1) Service Replicas: The utilization of active service $i$ replicas, $U_a$ is defined as the invocation rate, $\lambda_a$ divided by the aggregate capacity provisioned, i.e., $n_a t_c / d_{c_i}$, according to the utilization law [8]. At every control window, Resos provides sufficient number of replicas, $n_a(t)$, such that the effective utilization is less than the target value,

$$U_a(t) = \frac{\lambda_a(t) t_c}{n_a(t) t_c} \leq U^*, \forall i, t. \tag{3}$$

After substituting $\tilde{\lambda}_a(t)$ and following algebraic manipulation, Resos controls $n_a(t)$ by following

$$n_a(t) = \lfloor \frac{\lambda_a(t) t_c}{U^* t_c} \rfloor, \forall i, t. \tag{4}$$

2) CEE Replicas: The effective utilization of CEEs considers the blocking I/O in dealing with sequential service invocations. We let $P(t)$ be the non-blocking probability of sequential invocations within a composition. The effective capacity of all CEE replicas at window $t$ is the product of the aggregate CEE capacity and non-blocking probability, $n_c(t) t_c / d_c P(t)$. The workload sent to CEEs from application $a$ is the request rate multiplied by the number of invocations in an composition, $\lambda_a[\Omega]$. Therefore, the aggregate workload of CEEs at window $t$ is:

$$\lambda_c = \sum \lambda_a[\Omega_a].$$

Similar to Eq. 3, one can then write the effective utilization of CEEs at window $t$ as

$$U^{eff}_c(t) = \frac{\lambda_c(t) t_c}{n_c(t) t_c P(t)} \leq U^*, \tag{5}$$

where $U_c$ denotes the nominal utilization. One can see that $U_c$ is higher than $U^{eff}_c$ by a factor of blocking probability, $P$.

We derive $P(t)$ as the weighted average of the non-blocking probability from applications, because the blocking depends on the composition defined in application. Let $P_a(t)$ be the non-blocking probability of application $a$, then write

$$P(t) = \sum_a \frac{\lambda_a(t) t_c}{\lambda(t)} P_a(t),$$

where $\frac{\lambda_a}{\lambda}$ is the percentage of application $a$ requests out of total application requests.

The non-blocking probability of application $a$ requests can be derived from the fraction of CEE processing time over the the summation of CEE processing and blocking time. For a composition request, CEE processing time is the processing time per invocation multiplied by the number of invocations, $d_c[\Omega_a]$. The blocking time is essentially the summation of service response times of all invocations, $\sum_{i \in \Omega} W_i(t)$. As such, we can express $P_a(t)$ as a function of $d_c$ and $W_i(t)$:

$$P_a(t) = \frac{|\Omega_a| d_c}{\sum_{i \in \Omega} W_i(t) + |\Omega_a| d_c}. \tag{6}$$

Note that $W_i(t)$ here is not stationary as the provisioning of service replicas changes across different time windows. To obtain $P_a(t)$, we propose to substitute $W_i(t)$ by an estimate based on last window statistics,

$$W_i(t) = W_i(t-1), \forall i, t. \tag{7}$$

Using the estimated total request rate, the application request rate, and the response time of services, one can obtain

$$P(t) = \sum_a \frac{\lambda_a(t) t_c}{\lambda(t)} P_a(t)$$

$$= \sum_a \frac{\lambda_a(t) t_c}{\lambda(t)} \frac{|\Omega_a| d_c}{\sum_{i \in \Omega} W_i(t) + |\Omega_a| d_c}. \tag{8}$$

Combining Eq. 6, and 8 and some algebraic manipulations, Resos controls the number of CEE replicas by following

$$n_c(t) = \lfloor \frac{\lambda_c(t) t_c}{P(t) t_c U^*} \rfloor \leq \lfloor \frac{\lambda_c(t) t_c}{P(t) t_c U^*} \rfloor, \forall t. \tag{9}$$

In summary, Resos monitors statistics regarding to application request rates, service invocation rates, the utilization of CEE and service replicas, and the response time of service invocations. Using the collected and estimated statistics, Resos activates and deactivates replicas at the beginning of each window. Note that the statistics monitored in Resos can be easily collected on production systems.

C. Bounding Analysis on CEE Performance

One can see that the effective utilization of CEE is higher than the nominal utilization, which is commonly measured by the utility tools. Following our model and analysis in the previous subsection, we derive the upper bound of nominal utilization as a function of the target utilization values. Consequently, one can use such an upper bound as a simple rule of thumb to evaluate the CEE performance of service systems.

Theorem 3.1: The upper bound of nominal utilization of CEE replicas is

$$U_c \leq (U^*)^2.$$ 

The upper bound of $U^*$ is achieved when the non-blocking probability is equivalent as the target utilization, $P = U^*$.

Proof: We start the proof by first deriving a looser upper bound of nominal utilization. Then, using the optimal value of non-blocking probability, we can reach a tighter bound, which only depends on the target utilization.

From Eq. 6, one can write $U_c \frac{\lambda}{\lambda} \leq U^* \leq 1$. First, as $P \leq 1$, we know $U_c \leq U^*$. Secondly, as $U_c \frac{\lambda}{\lambda} \leq 1$, we know $U_c \leq P$. Combining both observations, one can get a loose upper bound of $U_c$, by taking the minimum of $P$ and $U^*$. $U_c \leq \min\{U^*, P\}$.

When $U^* \geq P$, $U_c \leq P$; whereas when $U^* \leq P$, $U_c \leq U^*$.

Consequently, the upper bound of $U_c$ increases in $P$ and stays constant at $U^*$, after $P$ reaches $U^*$. In other words, when $P =
$U^*$, the upper bound $U_c$ is maximized. Taking $P = U^*$ into Eq. 6, one can get $U_c \frac{1}{c} \leq U^*$, and then $U_c \leq (U^*)^2$.

Theorem 3.1 points out that to achieve the nominal utilization upper bound, the non-blocking probability should be at least as high as the target utilization. However, the non-blocking probability at CEEs is bounded by the relative difference between the CEE processing time and response time of service invocation. The maximum achievable non-blocking probability is when there is no queuing time at the service replicas. Comparing such a non-blocking probability with the target utilization, one can gauge how tight the nominal utilization is bounded by $(U^*)^2$. When the maximum achievable non-blocking probability is lower than $U^*$, the nominal utilization is lower than $(U^*)^2$, whereas when the maximal achievable non-blocking is greater than $U^*$, the nominal utilization might reach $(U^*)^2$.

IV. SERVICE SELECTION: D-RR AND D-SQ

To evenly balance the loads on the distributed replicas, we adopt two service selection algorithms. Shortest Queue (D-JSQ) [7]. For each service invocation in a request, a CEE thread selects service replicas using only statistics collected at the local CEE replica. That is, threads of a replica have the service replica statistics from their local requests, but not the aggregate statistics from all CEE replicas. In the following, we describe two selection policies:

1) Distributed Round Robin Selection (D-RR):

Each CEE replica maintains a round-robin list of active service replicas. At the beginning of each control window, the list is updated by adding (removing) the newly activated (deactivated) service replicas. Upon service replica selection, the CEE thread requests the next service replica from the round-robin list and sends the invocation request accordingly. D-RR is completely load oblivious and the resulting loads on service replicas may not be optimally balanced.

2) Distributed Shortest Queue Selection (D-SQ):

A CEE replica keeps statistics of outstanding service invocation requests sent by its threads and the corresponding queueing information at active service replicas. Upon service replica selection, the CEE thread selects the replica with the lowest number of queued invocations based on the locally maintained statistics. The implementation overhead is limited compared to the conventional shortest queue selection, which collects queueing statistics from all CEE replicas. D-SQ is partially load aware, practical, and has good potential for reducing response time [4] and balancing loads on replicas.

As D-SQ is expected to achieve lower response times of service invocation than D-RR, one can expect that the resulting non-blocking probability is higher for D-SQ, according to Eq. 8.

V. EVALUATION

In this section, we evaluate Resos in combination with two load balancing schemes using trace-driven simulation. We first describe the simulated environment: the workload generator and the system scenarios. Our evaluation results, based on the average of ten simulation runs, show that Resos can effectively reduce the consumption of CEE and services replicas, while maintaining the target utilization and minimizing the response time of service invocations.

A. Simulator and System Configuration

We built an event-driven simulator of service-oriented systems in Java, as shown in Figure 1 and 2. Requests are generated from applications. A CEE replica has $t_s = 32$ threads to process service compositions and invocation in parallel. The execution per CEE thread is assumed exponentially distributed with average $d_t = 0.5$ second. A service replica is configured to have $t_r = 4$ threads, independent of service types. Resos collects workload statistics at every control window and activates/deactivates replicas of CEE and services at the beginning of a window. The length of the control windows is chosen according to workload characteristics and prediction schemes.

1) Simulation Workload: The arrival patterns of requests from different applications are not commonly available to the public, due to the business confidentiality. The most widely used traces are world cup web site workloads that date back to 1998 [2], [12], or are derived from the TPC-W benchmark [14], which was last updated in 2001. In contrast to conventional approaches, we seek an alternative to generate the workload – converting the CPU utilization traces of an existing production system into the workload input of a discrete simulator [5], [10], [17]. According to the basic utilization law [8], the utilization multiplied by a normalized constant is essentially the throughput, which in turns reflects the request rate, especially when the load is below 100% utilization.

We collect utilization traces from four servers engaging in web services at financial, airline and media industries, during 10 am-12pm on October 20, 2011. The trace from one server is considered as one application. The utilization values are the average computed over 15 minutes. To obtain the request rate per second, we multiply the utilization values with the processing power of the server, i.e., the number of cores. We illustrate the rationale by an example. Let the utilization value be 35% for a 16 core server. This implies that, on average, 5.6 (0.35 · 16) cores are busy. We further assume that a core is occupied by a single request and such a value corresponds to the request arrival rate for a small granularity, i.e., second. As such, we obtain the request rates for four applications, shown in Figure 3. One can clearly see that the workloads are time-varying.

Due to the limitation of the coarse granularity in collecting utilization we are unable to collect the higher moment statistics and further fit the empirical distribution of utilization. Consequently, we assume that the arrivals of requests follow Poisson processes for each 15 minutes and that their means fluctuate according to Figure 3. Once requests are generated, they are then immediately forwarded to available CEE replicas in a random fashion.
2) Simulated System Scenarios: In particular, we consider the following two specific system scenarios and their compositions:

- **System scenario I:**
  The system provides a single type of service, namely $S_0$. Requests are generated from two applications, i.e., $a = \{1, 2\}$, whose requests rates correspond to app1 and app2 in Fig. 3. Their service compositions are $\Omega_1 = \{S_0, S_0\}$, $\Omega_2 = \{S_0, S_0, S_0\}$. The execution time of a replica thread at $S_0$ is assumed exponentially distributed with mean $d^{\text{INV}} = 1$. The maximum number of available CEE and service replicas are $n_c = 9$, $n_{s_0} = 33$. The length of each control window is 100 seconds.

- **System scenario II:**
  The system provides three services, namely $S_0, S_1$ and $S_2$. Clients’ composition requests are generated from four applications, whose requests rates correspond to app1, app2, app3 and app4 in Fig. 3. Their service compositions are $\Omega_1 = \{S_0, S_1, S_0\}$, $\Omega_2 = \{S_0, S_2\}$, $\Omega_3 = \{S_0, S_1, S_2\}$, and $\Omega_4 = \{S_2, S_0, S_1\}$. The execution times of a replica thread at $S_0, S_1$ and $S_2$ are assumed exponentially distributed with means $d^{\text{INV}}_0 = 1$, $d^{\text{INV}}_1 = 1.5$, and $d^{\text{INV}}_2 = 2.5$ seconds, respectively. The maximal number of available CEEs, $S_0$, $S_1$ and $S_2$ replicas are $n_c = 18$, $n_{s_0} = 19$, $n_{s_1} = 26$, and $n_{s_2} = 37$, respectively. The length of each control window is 150 seconds.

![Fig. 3. Request rates of applications, $\lambda_a$.](image)

For both scenarios, we set the target utilization of the active CEE and service replicas to be 85% and 80%, respectively. Such values are chosen by empirical experiences [12]. Note that Resos aims to maintain the CEE effective utilization, which includes the blocking time, at the target value. For each simulation run, we collect the performance metrics, averaged over all control windows, i.e., replica savings, nominal and effective utilization of CEE and service replicas ($U_{\text{eff}}$, $U_c$, $U_s$), queueing time at CEEs ($Q_c$), response time of service invocations ($W_c$), and end-to-end request response time ($R_c$). In particular, the replica savings are computed as one minus the number of total active replicas divided by the maximal number of available CEE and service replicas. Using this metric, one can estimate the cost savings, given the target performance. For both system scenarios, we compute the average of the aforementioned metrics over ten simulation runs and present them in Tables II and III. Moreover, for the purpose of comparison, we additionally simulate a static replication policy which keeps the number of active service replicas at the maximum for all control windows, independent of workloads. The lowest end-to-end response time can be achieved via maximum replica provisioning.

B. System Scenario I

We apply Resos on system scenario I, with two different workload load prediction schemes and two service selection schemes, D-SQ and D-RR, and summarize the performance metrics in Table II. To verify the accuracy of workload prediction in Resos, we use Resos with actual application request and invocation rates and the default last value predictions. One can observe the performance degradation is 15%–25% when using last value prediction in Resos, with any given service selection.

When comparing D-SQ and D-RR under "actual" prediction, D-RR achieves similar replica savings and effective utilization as D-SQ; however, D-RR has roughly 20% higher invocation and end-to-end response time. The utilization of service replicas is slightly under the target value of 80%, whereas the effective utilization of CEEs is roughly 15% lower than the target value, due to the large number of threads in a CEE replica. As expected, D-SQ can achieve a lower response time via better load balancing, and thus a lower non-blocking probability at CEE threads that is reflected by the relative difference between $U_{\text{eff}}$ and $U_c$. Moreover, due to a low non-blocking probability, the CEE nominal utilization is way lower than the its upper bound, according to Theorem 3.1 One can expect D-RR to have even worse performance when the workload prediction is inaccurate, i.e., over- and under-estimating. Consequently, we provide a higher spare capacity for CEE and set a slightly lower utilization target when applying Resos with last value prediction and D-RR, i.e., 80% and 75% respectively for both scenario I and II.

When applying Resos with the last value prediction specified in Eq. 2, the replica savings are around 50%, and D-RR has slightly lower replica savings due to a lower target utilization. The average queuing time at CEEs is significantly higher than in the "actual" case, and consequently the end-to-end response time of the applications is higher than in the "actual" case by 15%–25%. Even with higher provisioning of CEE and service replicas, Resos with last value prediction and D-RR selection still has the worst queueing time at the CEE and consequently the worst application end-to-end response times. As pointed out earlier, the service selection can fine tune the performance, but the provisioning of the replicas are the first order parameters to control.

Overall, Resos with last value prediction can achieve (1) significant replica savings; (2) CEE and service utilization that is slightly under than the target values; and (3) very low end-to-end request response times that are only slightly higher than the response times under static maximum provisioning.


C. System Scenario II

We summarize the performance metrics of applying Resos with "actual" and "last value" prediction in Table III. Following the observation and rationale in scenario I, we set the target utilization of CEEs and services to 80% and 75%, respectively, when applying Resos with last value prediction and D-RR.

One can make the following general observations, which are similar to the ones made in scenario I: The replica savings achieved by Resos is quite significant, compared to providing the maximum number of replicas in all windows. Resos maintains the CEE and service utilization just slightly below the target values. In particular, when applying Resos with "actual" prediction, the end-to-end response time, $R_1$, $R_2$, $R_3$, and $R_4$, is roughly 10% higher than with static maximum replica provisioning. It strongly supports the accuracy of Resos in predicting performance metrics, especially in a more complex system. The difference between "actual" and "last value" prediction is more visible in CEE queueing time ($Q_c$) and thus degrades the end-to-end response time roughly by 10 – 20%.

We plot the run time results of applying Resos with last value prediction and D-SQ in Figure 4. The number of CEE and service replicas are highly correlated, because the number of CEE replicas determines the invocation rates received by service replicas. As such, the CEE utilization oscillates in a greater range than the service utilization. Queueing time at the CEEs is fairly low, except for two spikes around 70 and 80 minutes. The invocation response times for all services are even more stable, except for a spike around 80 minute. Overall, Resos is able to provide sufficient numbers of CEE and service replicas, keep them well utilized, and maintain stable response times, given the load fluctuation over the time.

VI. RELATED WORK

The related work regarding replicated web services is mainly discussed in two contexts: fault tolerant services, and resource provisioning.

In order to provide highly dependable service-oriented systems, various service replication framework and strategies are developed in different system scenarios. Many studies [6], [13], [21], [22] consider the replication of atomic services and do not address the optimal number of replicas. Salas et. al. [13] developed a replication framework, WS-Replication, which enables the deployment in a set of sites. In particular, WS-Replication respects web service autonomy and exclusively

![Fig. 4. Run time control and performance of Resos with D-JQ, on scenario II.](image-url)

<table>
<thead>
<tr>
<th>Workload</th>
<th>Performance Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Prediction</td>
<td>Service Selection</td>
</tr>
<tr>
<td>Actual</td>
<td>D-SQ</td>
</tr>
<tr>
<td>Actual</td>
<td>D-RR</td>
</tr>
<tr>
<td>Last value</td>
<td>D-SQ</td>
</tr>
<tr>
<td>Last value</td>
<td>D-RR</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Maximum Static Provisioning</th>
</tr>
</thead>
<tbody>
<tr>
<td>case</td>
</tr>
<tr>
<td>-----------------------------</td>
</tr>
<tr>
<td>none</td>
</tr>
<tr>
<td>none</td>
</tr>
</tbody>
</table>
TABLE III

<table>
<thead>
<tr>
<th>Workload</th>
<th>Performance Statistics</th>
<th>Resos</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Prediction Selection</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Savings [%]</td>
<td>$U_C^{S}$ [%]</td>
</tr>
<tr>
<td>Actual</td>
<td>D-SQ</td>
<td>10.39</td>
</tr>
<tr>
<td>Actual</td>
<td>D-RR</td>
<td>10.39</td>
</tr>
<tr>
<td>Last value</td>
<td>D-SQ</td>
<td>10.39</td>
</tr>
</tbody>
</table>

VII. CONCLUSION

In this paper, we study a service-oriented system hosting multiple replicas of CEEs and multiple services. Our system model captures the dynamic workloads, and the interdependency between CEE and service replicas equipped with multiple threads. To reduce operational cost, as well as minimize the end-to-end response time of applications, we develop Resos, a dynamic replication manager. Resos periodically adjusts the provisioning of replicas such that both CEEs’ and services’ effective utilization is kept at target values. Resos explicitly factors in the dependency between CEEs and services, using the derivation of non-blocking probability at CEEs. Furthermore, we provide theoretical bounding analysis on CEEs and derived optimal/maximal nominal utilization. Our trace driven simulation results show that Resos with simple last value workload prediction can achieve a great replica saving and keep CEE and service replicas well utilized, while maintaining low response times, especially when the loads on service replicas are well balanced. For future work, we plan to set up a hardware testbed and explore the architecture features.

optimizing system resources and performances. Resos can be orthogonally combined with existing replication frameworks and fault tolerant replication strategies.

To design a scalable and cost-effective service-oriented system, dynamic resource provisioning is very critical, especially when encountering time-varying workloads. A number of studies [5, 9] focus on a single tier web server system, whereas multi-tier web server systems are well addressed in [14, 15, 20]. Resource provisioning strategies in multi-tier systems often consider each tier independently from other tiers. Petrucci et al. [12] implement a dynamic service provisioning policy to optimize power consumption on a heterogeneous cluster. While most provisioning studies monitor the request rate, Singh et al. [14] monitors not only the request rate but also the mix of applications. Resos incorporates the dependency among CEEs and services into the model by monitoring application mixes and providing the sufficient number of replicas to achieve the resource utilization targets.

REFERENCES