Policy Learning for Adaptive Allocation of Cloud Resources to Multi-tier Web Applications

Waheed Iqbal*, Matthew N. Dailey
Computer Science and Information Management
Asian Institute of Technology, Thailand
{waheed.iqbal, mdailey}@ait.asia

David Carrera
Technical University of Catalonia (UPC) and
Barcelona Supercomputing Center (BSC), Spain
dcarrera@ac.upc.edu

1. INTRODUCTION

From the user's point of view, *response time* is the most important quality attribute of a Web service, and Web service providers aim to limit response time at minimal cost. Dynamic resource scaling in cloud computing could potentially enable specific response time guarantees, but current service-level agreements (SLAs) offered by cloud infrastructure providers do not address response time.

Hosted multi-tier Web applications, in which each tier is deployed to a separate set of fixed virtual machines, can only service a limited number of requests concurrently before some bottleneck occurs. Once a bottleneck occurs, the application will saturate, response time will grow dramatically, and the application will start failing to service requests.

For simple Web applications, it is possible to achieve low operational costs by first minimizing resource allocation then dynamically scaling allocated resources as needed to handle increased loads [1]. However, for multi-tier Web applications, it is more difficult to automatically identify the exact location of a bottleneck and scale the appropriate resource tier accordingly. This is because multi-tier applications are complex and bottleneck patterns may be dependent on the specific pattern of workload at any given time. It is possible to identify bottlenecks for a specific application by monitoring and profiling the low-level hardware resource utilization in each tier [2]. However, these fine-grained techniques have to deal with extra monitoring overhead, virtualization complexity, and end-user security concerns involved in installing monitoring agents on rented virtual machines. Furthermore, low-level hardware profiling without monitoring response time metrics could not identify performance issues created by software misconfiguration. One example is whether the number of connections from each server in the Web server tier to the database tier is appropriate.

To address these issues, in this paper, we present a method for learning appropriate application- and workload-specific resource provisioning policies in real time. We develop a formal model for and techniques to identify the parameters of workload patterns for multi-tier Web applications using access logs and unsupervised machine learning. The method automatically identifies groups of URIs with similar resource utilization characteristics from historical access log data. We also design and empirically evaluate a method for satisfying a SLA that provides a maximum response time guarantee that works by reactively identifying bottlenecks for specific workload patterns and then learning resource allocation policies, all based on coarse-grained access log monitoring in real time. The policy learner initially uses a trial and error process to identify an appropriate bottleneck resolution policy in the context of a specific workload pattern then exploits that policy to reduce violations of the SLA while minimizing resource utilization. The approach does not require pre-deployment profiling or any insights about the application. We evaluate our proposed system on a EUCALYPTUS-based private cloud and the RUBiS benchmark Web application. The experimental results show that the proposed method can enable cloud providers or Web service owners to offer response time guarantees for multi-tier Web applications using adaptive resource allocation without any prior knowledge of the application's resource utilization or workload patterns.

In the rest of the paper we provide an overview of our proposed workload pattern model, policy learning, and initial experimental results.

2. WORKLOAD PATTERN MODEL

Our workload pattern model consists of two components:

URI space partitioning: a partitioning of the application's URI space into requests with similar resource utilization characteristics. We collect a sufficient number of application logs then preprocess them to extract the URI path, document size, and service time for each request. We use a Gaussian mixture model clustering algorithm to group the requests into clusters based on document size and service time. We then construct a map from each URI path to the corresponding cluster ID.

Workload pattern: our workload pattern model is probabilistic. In each independent trial of the random experiment, we wait for the arrival of a single HTTP request from a remote client and observe the cluster $c \in \{c_1, \ldots, c_k\}$ that the request URI path falls into. Let C be the random variable describing which of the k clusters a request falls into. A probability distribution $P(C)$ over the clusters defines a *workload distribution* for the Web application as $P(C) = (P(c_1), P(c_2), \ldots, P(c_k))$. Given the mapping from URI paths to cluster IDs, identifying the workload pattern for a specific interval of time is a simple matter of observing the frequency of arrival of requests for URI paths in each cluster.

*Ph.D student at Asian Institute of Technology, Thailand.

3. POLICY LEARNING

We currently use a simplified greedy version of Q-learning to build, through online observation, an estimate of the value $Q(s_t, a)$ of each action in each state. The learning agent begins with no knowledge and monitors, over each interval of time, for SLA violations. Whenever the agent detects a violation, when the amount of information is sufficient, it performs the optimal action according to the current estimated Q value function. When knowledge of the value function is insufficient, the agent attempts all possible actions using a simple exhaustive exploration algorithm. Algorithm 1 gives pseudocode for our policy learning and decision making algorithm.

The **system state** $s_t = (U, P(C), \lambda, p)$ at time t contains the configuration of the Web application's tiers U, the current workload pattern $P(C)$, the current arrival rate λ, and the current 95^{th} percentile of the response time p. For an n-tier application, we define a configuration by the vector $U = (u_1, \cdots, u_n)$, where element u_i indicates the number of machines allocated to tier i. The **action** a the agent can select at any point in time is a particular scale-up strategy. For our benchmark two-tier Web application, the possible scale-up strategies are to scale up the Web tier (a^w), to scale up the database tier (a^d), to scale up both tiers (a^b), or do nothing ($a^\emptyset$). The set of possible actions the agent can perform is thus $A = (a^w, a^d, a^b, a^\emptyset)$. The **environment** that agent interacts with is a stochastic function $E(s_t, a)$ mapping current state s_t and action (scaling strategy) a to a new state. We also allow the agent to instantaneously retract a previously executed action in the current environment with a function $E'(s_t, a)$. The **reward function** r encourages the learning agent when it is successful and discourages it when it is unsuccessful at maintaining the SLA with minimal allocation of resources. We use the immediate reward function

$$r(s_t) = \frac{1}{p + \alpha \sum_{i=1}^{n} u_i}. \quad (1)$$

α specifies the relative weight of the response time and resource minimization objectives.

4. EXPERIMENTAL EVALUATION

To evaluate the proposed method, we performed an experimental evaluation in which we first learned a URI partitioning model for the RUBiS benchmark Web application and then performed two experiments using different workload patterns. Both experiments executed in three phases in turn named **Exploration** (we initialize an empty policy and let the system learn in real time using the proposed policy learning algorithm), **Exploitation** (the agent simply uses the already-learned dynamic resource provisioning policy to automatically resolve bottlenecks), and a **Baseline** method similar to that of Urgaonkar et. al [3], who scale up every replicable tier every time a bottleneck occurs. We use the same pattern of workload growth in all experiments. Table 1 shows the total allocated CPU hours and the percentage of requests violating the SLA during the Exploitation and Baseline phases of each experiment. In both experiments, the system allocated fewer total CPU hours under the proposed policy learning approach at the cost of a slightly higher but still acceptable percentage of requests violating the SLA.

Input: Environment functions E and E', state space S, initial state s_0, actions A
Output: Estimated state-action value function $Q : S \times A \mapsto \mathbb{R}$.

For all $s \in S$, $a \in A$, $Q(s, a) \leftarrow 0$
$t \leftarrow 1$
$s_1 \leftarrow E(s_0, a^\emptyset)$
while *true* **do**
  Extract p (95^{th} percentile service time) from s_t
  if $p > \tau$ *(SLA violation detected)* **then**
    if *for any* a, $Q(s_t, a) = 0$ **then**
      for *each* $a \in A \backslash a^\emptyset$ **do**
        $t \leftarrow t + 1$
        $s_t \leftarrow E(s_{t-1}, a)$
        $Q(s_{t-1}, a) \leftarrow r(s_t)$
        $s_t \leftarrow E'(s_t, a)$
      end
    end
    $a_t \leftarrow \text{argmax}_a\, Q(s_t, a)$
  else
    $a_t \leftarrow a^\emptyset$
  end
  $t \leftarrow t + 1$
  $s_t \leftarrow E(s_{t-1}, a_t)$
end

Algorithm 1: Policy learning algorithm.

Table 1: Experimental results summary.

		Total allocated CPU hours	% of requests violating SLA
Exp 1	Exploitation	17.7	1.03
	Baseline	19.8	0.35
Exp 2	Exploitation	24.7	1.75
	Baseline	27.0	0.233

The proposed policy learning approach could help cloud providers to offer a multi-tier Web application hosting service with an SLA providing specific response time guarantees while minimizing resource utilization. We are currently improving the exploration phase of the policy learning algorithm and planning a direct comparison of the proposed method with the Amazon EC2 Auto Scaling service.

5. REFERENCES

[1] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Gener. Comput. Syst.*, 27:871–879, June 2011.

[2] J. Rao and C.-Z. Xu. Online capacity identification of multitier websites using hardware performance counters. *IEEE Transactions on Parallel and Distributed Systems*, 99, 2010.

[3] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood. Agile dynamic provisioning of multi-tier internet applications. *ACM Transactions on Autonomous and Adaptive Systems*, 3:1–39, 2008.