Efficient Resource Allocation and Power Saving in Multi-Tiered Systems

Andrew Caniff[1], Lei Lu[2], Ningfang Mi[3], Ludmila Cherkasova[4], Evgenia Smirni[5]
[1,2,5]College of William and Mary, Williamsburg, VA
[3]Northeastern University, Boston, MA
[4]Hewlett-Packard Laboratories, Palo Alto, CA
[1,2,5]{awc, llu, esmirni} @cs.wm.edu, [3]ningfang@ece.neu.edu, [4]lucy.cherkasova@hp.com

ABSTRACT

In this paper, we present Fastrack, a parameter-free algorithm for dynamic resource provisioning that uses simple statistics to promptly distill information about changes in workload burstiness. This information, coupled with the application's end-to-end response times and system bottleneck characteristics, guide resource allocation that shows to be very effective under a broad variety of burstiness profiles and bottleneck scenarios.

Categories and Subject Descriptors

C.4 [**PERFORMANCE OF SYSTEMS**]: Reliability, availability, and serviceability; H.3.4 [**INFORMATION STORAGE AND RETRIEVAL**]: Systems and Software—*Performance evaluation (efficiency and effectiveness)*

General Terms

Algorithms, Performance

Keywords

Resource Allocation, Burstiness, Multi-tiered Systems

1. INTRODUCTION

Resource allocation in a multi-tiered system is more challenging than in a single-tiered one. In a multi-tiered system, the bottleneck tier regulates the request flow and dominates performance. Alleviating the bottleneck tier by assigning more processing power is straightforward but should be done with caution as the bottleneck may simply shift to another tier [5]. Traditional provisioning triggers resource reallocation when certain thresholds are violated [5]. The effectiveness of such techniques depend on astute selection of their parameters. What makes resource allocation even more challenging in a multi-tiered system is the phenomenon of bottleneck switch that further exacerbates the difficulty of the problem [3, 4]. Resource allocation that requires saving power without compromising performance becomes a conundrum for system designers. In this work, we present a parameter-free algorithm called Fastrack that *quickly* tracks achievable performance and workload burstiness to self-adjust the allocation of available resources with the aim of optimizing performance while using *minimal* resources.

Fastrack uses online measurements to determine whether the system experiences a *true peak* or simply variability in user arrivals and quickly determines the start of a burst, signaling the need to assign more computing resources. Correspondingly, it also detects the end of a burst, i.e., rapid returns to normal traffic intensity, signaling the need to reduce computing resources without any performance penalty.

WWW 2010, April 26–30, 2010, Raleigh, North Carolina, USA.
ACM 978-1-60558-799-8/10/04.

2. ALLOCATION ALGORITHM: FASTRACK

Our focus is on effective resource allocation in multi-tiered systems, and we assume an architecture such as the one used by the TPC-W benchmark, a standard benchmark that is routinely used for capacity planning of e-commerce systems and that consists of a front server (hosting a web server and an application server), and a back-end database. Client requests may cycle between the front and back-end (database) servers before they are returned to the client. The TPC-W benchmark implements a fixed number of emulated browsers (EBs) that send requests to the system.

Burstiness in request arrivals results in the phenomenon of persistent "bottleneck switch" where performance measures are counter-intuitive, e.g., user SLOs are grossly violated while performance measures such as device utilizations are moderate [3]. In [3], the authors proposed to incorporate the *index of dispersion I* [1] into new capacity planning models of multi-tier enterprise systems. We use I to infer information about the patterns of upcoming workloads, i.e., we use statistical information for the bursts to strengthen the accuracy of workload prediction. We show that I can provide a simple yet powerful way for prompt identification of the *start* and the *end* of a bursty period.

During a workload surge, the algorithm uses a "pro-active" approach to quickly identify the surge, and tames its effects by summoning new resources before performance starts to suffer. On the other hand, after the algorithm detects a quiet period, it releases resources with a slower pace such that jobs that are accumulated during a burst are flushed and the operation of the system reverts to normal. Timely identification of the start and end of bursts is in the core of Fastrack and together with the system's SLOs guides when is best to expand or to contract the number of resources to the application.

In addition to detecting bursty conditions, it is also important to keep track of changes in the target performance measures and continuously compare them with those of systems SLOs, which are usually in the form of percentiles of user response times (RTs). In the absence of a burst Fastrack employs *reactive* mode and looks at the performance of the current request batch to adjust provisioning. Fastrack monitor deviations of the user performance measures from the target SLOs and quickly adjust resources aiming at minimizing these deviations.

3. PERFORMANCE EVALUATION

We evaluate the effectiveness of Fastrack by simulating the workload flows in a typical TPC-W 2-tier implementation (i.e., a front server and a database server). For our experiments, we extend the basic model to include a pool of 8 front servers that can be brought online/offline during the experiment. We focus on the effectiveness of the algorithm under a wide variety of workloads to investigate Fastrack's effectiveness under different

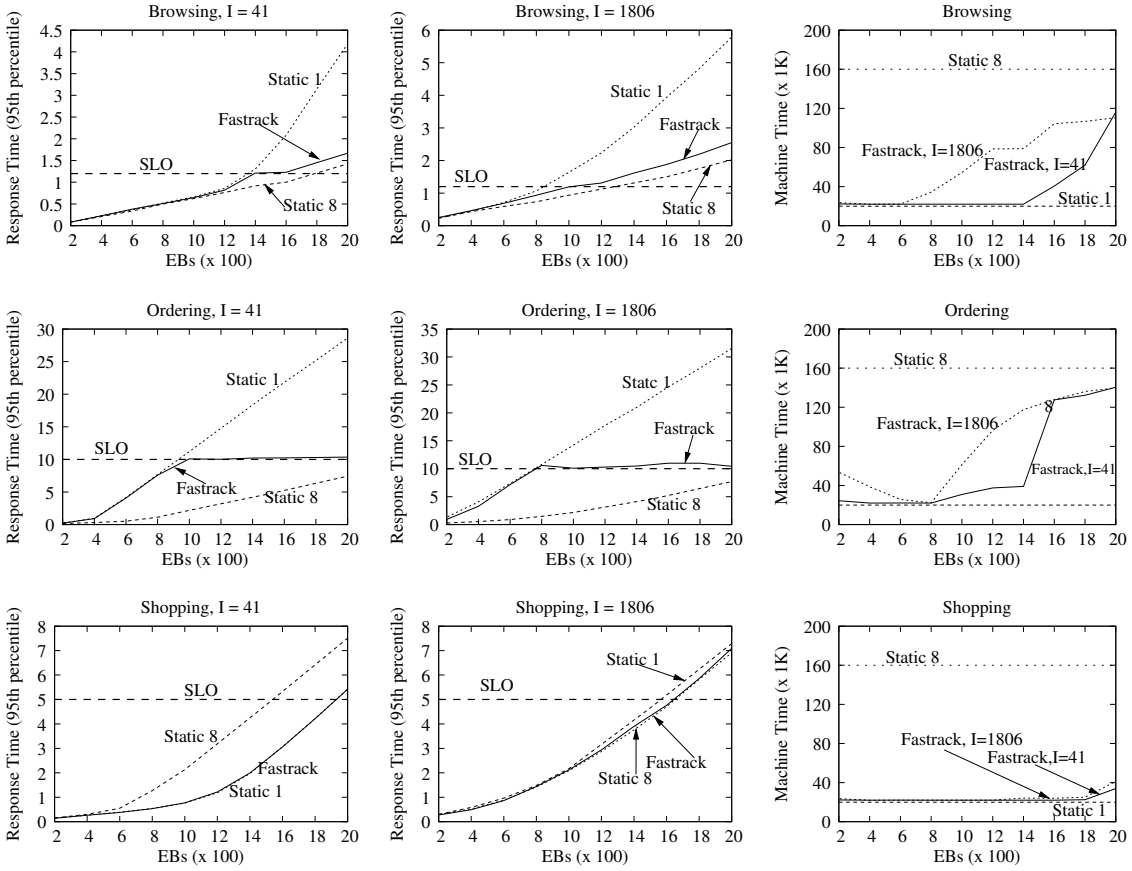

Figure 1: 95th percentiles of response times and raw machine times under three transaction mixes and two burstiness profiles. Each row shows performance numbers for a different transaction mix. The left column is response time percentiles for the low burstiness case, the middle column is the high burstiness case, and the right column summarizes the power usage of each configuration.

burstiness levels in workload traffic, different loads, different workloads demands (i.e., different bottlenecks), and also under the case of bottleneck switch. Our purpose is to show the robustness of the algorithm under all of the above conditions.

TPC-W defines three transaction mixes: *browsing*, *shopping*, and *ordering* mixes. The think times of emulated browsers are modeled by using two different MAPs [2], each with a different burstiness profile. Consistent with the TPC-W specification, both MAPs average user think time is equal to 7 seconds but their SCV is equal to 20 (i.e., inter-arrival times are very variable). Furthermore, the two MAPs have different burstiness profiles: one results in the index of dispersion $I = 41$, i.e., very low burstiness, and the other one with $I = 1,806$ which constitutes significant burstiness.

Figure 1 depicts the performance results for the various experiments. The figure, organized as a three by three grid, presents performance numbers in the form of 95th percentiles of user response times for the three TPC-W mixes when the arrival workload exhibits slight burstiness (leftmost column of graphs), high burstiness (middle column of graphs), and raw machine times units for the front tier (rightmost column). All results are presented for various populations (emulated browsers) in the system such that we show performance in low loads (low populations) and high loads (high populations). The first row of graphs corresponds to the browsing mix, the second one to the ordering mix, and the last row to the shopping mix. Each plot in Figure 1 shows the results for Fastrack, the two boundary cases with a static number of front servers equal to 1 and 8, and the target SLOs. We set a different SLO for each mix. This is a user defined input to the algorithm, and is dependent on the performance level needed for the application. We consider the three mixes to be representative of three different application types, each with a different SLO. The last column of graphs shows the machine times as a function of population for the static 1 and 8 cases (the two parallel flat lines that correspond to the two boundary static cases) as well as the machine times for Fastrack. The *power usage* is measured in raw machine seconds, i.e., the sum of times that front servers are in operation. Naturally, the closer the Fastrack machine times are to the lower flat line, the lower the power consumption.

Our algorithm Fastrack is able to stay near the response time SLO (or close to the response time with 8 servers, when SLO can not be met) for all the mixes, and is able to save significant power while doing so. Our results uniformly show that Fastrack is a robust, parameter free algorithm that can opeate seemlessly in a variety of settings.

4. REFERENCES

[1] R. Gusella. Characterizing the variability of arrival processes with indexes of dispersion. IEEE JSAC, 19(2):203–211, 1991.

[2] G. Casale, E. Zhang, E. Smirni. KPC Toolbox: Simple Fitting Using Markovian Arrival Processes. In the 5th Intl. Conf. on Quantitative Evaluation of Systems, 2008.

[3] N. Mi, G. Casale, L. Cherkasova, E. Smirni. Burstiness in multi-tier applications: Symptoms, causes, and new models. In ACM/IFIP/USENIX Intl Middleware'2008.

[4] N. Mi, G. Casale, L. Cherkasova, E. Smirni. Injecting realistic burstiness to a traditional client-server benchmark. Proc. of the 6th Intl. Conference on Autonomic Computing (ICAC), 2009.

[5] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, T. Wood. Agile Dynamic Provisioning of Multi-tier Internet Applications. Transactions on Adaptive and Autonomous Systems (TAAS), March, 2008.