# A Black-Box Approach for Web Application SLA

Jeremy Philippe
INRIA
Grenoble, France
Jeremy.Philippe@inria.fr

Noël De Palma
INPG
Grenoble, France
Noel.Depalma@inria.fr

Sara Bouchenak
University of Grenoble I
Grenoble, France
Sara.Bouchenak@inria.fr

Fabienne Boyer
University of Grenoble I
Grenoble, France
Fabienne.Boyer@inria.fr

Daniel Hagimont
INRIA
Grenoble, France
Daniel.Hagimont@inria.fr

## ABSTRACT

Web servers nowadays have to cope with unprecedented amounts of workload, due to increasing popularity and complexity; in particular, dynamically generated content becomes the standard, hence the term Web application. Providing enough resources to sustain these workloads is a grand challenge, thus the application's runtime environment is often trusted to third-party hosting providers. To optimize resource utilization, hosting providers tend to share server machines between several applications. In such context, it is desirable to make sure that hosted applications are guaranteed a predictable level of performance, in other words implement service-level agreements (SLAs).

In this paper, we present an approach to Web application SLA, based on profiling and geared at black-box server components, which allows us to express SLAs using application-level metrics, such as request rates. We conducted experiments in the context of a two-tiers server architecture shared by two e-commerce applications, and were able to enforce performance guarantees expressed as a number of concurrent user sessions.

## Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

## General Terms

Performance, Measument, Management, Algorithms, Experimentation

## Keywords

Web Applications, Quality of Service, Performance Isolation, Request Scheduling

## 1. INTRODUCTION

When several Web applications share server machines, one particular challenge is to enforce performance isolation, i.e. one application should not be able to "steal" resources from other colocated applications. Performance isolation leads to the specification of service-level agreements (SLAs) which quantify performance guarantees.

Currently, research propositions for SLA implementation are limited in two important ways. First, they require special-purpose runtime environments able to perform fine-grained resource usage accounting, which severely limits applicability to production environments. A second limitation is the lack of expressiveness of the metrics used to quantify performance; indeed resource-level metrics are often used, e.g. CPU or network bandwidth. Unlike static Web servers, simply specifying resource-level bandwidth guarantees is not enough to quantify the performance of Web applications. Rather, application-level metrics should be used, e.g. request rates or response times. In this paper, we present an approach that strives to overcome these limitations.

## 2. APPROACH

Here is an example of a SLA based on application-level metrics, i.e. metrics that explicitely express application performance:

(Q) Response time for 95% of requests is under 500 ms.

(W) Number of concurrent client sessions is under 256.

It can be seen that we decompose the SLA in two parts: the (Q) part specifies the QoS received by individual client requests, while the (W) part specifies the maximum workload intensity for which (Q) can effectively be guaranteed. To enforce an SLA, one must provision each application with enough resource bandwidth to ensure that it can sustain (W) while providing (Q) to client requests. To achieve that result, it is necessary to:

(i) determine how much resource bandwidth should be available for the application to reach the specified performance level.

(ii) ensure that each application effectively gets that bandwidth, even when competing for resources.

(i) requires the profiling of the application, in order to map application-level performance metrics to resource-level metrics. (ii) requires performance isolation, that is appropriate resource scheduling so that each application gets a predictable share of resource bandwidth.

To perform profiling, we dedicate server machines to applications and inject realistic workload while resource utilization is monitored using standard system-wide tools. The advantage of that approach is that it does not require fine-grained instrumentation, as is the case of online profiling. Thus it can be used with black-box server systems.

To ensure performance isolation when applications share server machines, we use a form of logical partitioning based on a frontend request scheduler. Because profiling determines the cost of application requests, the scheduler does not need to track each application's resource usage, which would again require fine-grained scheduling, unavailable on black-box systems. Instead a simple weighted round-robin scheduling discipline can be used. That discipline allows us to enforce performance isolation and lets unused processing capacity to be redistributed to other applications that would need it.

## 3. EVALUATION

To evaluate the validity of our approach we conducted experiments involving two realistic Web applications, RUBiS (an eBay clone) and TPC-W (an Amazon clone). RUBiS and TPC-W were deployed on common hardware, composed of a frontend Web tier and a backend database tier. We considered the following SLAs:

| | Workload |
|---|---|
| RUBiS | 512 concurrent sessions |
| TPC-W | 64 concurrent sessions |

Because these two applications are well-known benchmarks, commonly used to quantify the relative performance of different runtime environments, realistic client emulators readily exist to inject workload, thus facilatating the profiling task. We observed an expected linear correlation between resource utilization and workload intensity for all resources considered (CPU, network, disk). Only CPU utilization of the database tier was considered in the profiles since it was an order of magnitude greater than utilization of other resources.

With these profiles, we were able to map application-level metrics to resource-level metrics:

| | RUBiS | TPC-W | Total |
|---|---|---|---|
| Number of clients | 512 | 64 | |
| Request rate | 70.1 req/s | 74.9 req/s | |
| CPU utilization | 21.5% | 18.9% | 40.4% |

These profiles show that the server machines should provide more than enough processing bandwidth to sustain the peak workload specified in the SLAs. However, if one application receives workload that exceeds the SLA specification, it must not impact performance of the other application (if well-behaved). To evaluate performance isolation, we injected a TPC-W workload with increasing intensity, thus simulating an overload, while RUBiS received the peak workload specified in the SLA (512 concurrent sessions). We observed the aggregate processing rate delivered to RUBiS clients to verify that the performance guarantees were met. In figure 1, we show comparative results for 5 request scheduling disciplines:

(1) `direct` No frontend scheduler is present (clients access the server directly).

(2) `fifo` First in, first out discipline.

(3) `prio` Priority-based discipline (RUBiS clients are strictly prioritized).

(4) `wrr-0.5` Weighted round-robin discipline (RUBiS has weight 0.5).

(5) `wrr-0.25` Weighted round-robin discipline (RUBiS has weight 0.25).

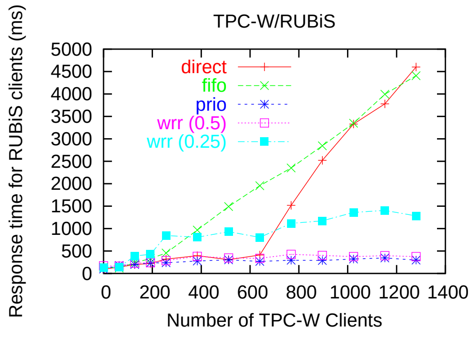

**Figure 1: Service isolation**

These graphs show that (i) when not using a class-aware discipline (1 and 2), overloaded applications (TPC-W) can easily "steal" resources from well-behaved applications (RUBiS); (ii) if properly configured (based on profiling information), a performance isolation discipline (4) essentially works as a strict-priority discipline (3), only in favor of well-behaved applications; and (iii) if not configured properly, a performance isolation discipline (5) still provides performance guarantees, only these guarantees do not fit the requested SLA.

## 4. CONCLUSIONS

In this paper, we have proposed a solution for SLA implementation in the context of Web applications. Contribution of our approach is twofold; first, it can be applied to black-box middleware components; and second, it allows the use of application-level metrics in SLA specifications. This is done with two steps, an offline profiling step to predict the cost of requests and an online scheduling step to enforce performance isolation. We conducted experiments in the context of J2EE Web applications, and validated our approach using two well-known J2EE benchmarks that reproduce real-life applications.