Future Generation Computer Systems 29 (2013) 913–925

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Towards transparent and distributed workload management for large scale web servers

Shengzhi Zhang [a,*], Wenjie Wang [b], Haishan Wu [c], Athanasios V. Vasilakos [d], Peng Liu [a]

[a] The Pennsylvania State University, University Park, PA, USA
[b] Shanghai Synacast Media Tech. (PPLive) Inc., China
[c] IBM Research - China, Beijing, China
[d] Computer Science Department, Kuwait University, Kuwait

ARTICLE INFO

Article history:
Received 25 June 2012
Received in revised form
1 October 2012
Accepted 6 October 2012
Available online 30 October 2012

Keywords:
Resource management
Decentralized control
Service level agreement

ABSTRACT

The rapid expansion of cloud offerings poses fundamental tasks for workload management in a large scale server farm. In order to achieve satisfactory Quality of Service (QoS) and reduce operation cost, we present a fully distributed workload management system in a large scale server environment, e.g., cloud. Different from existing centralized control approaches, the workload management logic hierarchically spreads on each back-end server and front-end proxy. The control solution is designed to offer both overload protection and resource efficiency for the back-end servers, while achieving service differentiation based on Service Level Agreement (SLA). The proposed system can directly work with legacy software stack, because the implementation requires no changes to the target operating system, application servers, or web applications. Our evaluation shows that it achieves both overload protection and service classification under dynamic heavy workload. Furthermore, it also demonstrates negligible management overhead, satisfactory fault-tolerance and fast convergence.

1. Introduction

The typical architecture of server farms usually consists of multiple tiers of servers, for example, HTTP servers in the front, proxy tier in the middle, then application and database servers at the back-end. The HTTP server tier filters out invalid or malicious requests and forwards legitimate ones to the proxy tier, which in turn routes these requests to the corresponding application servers for processing. As the server farm scales to an infrastructure consisting thousands of network components located in many separated data centers, efficient workload management becomes a challenging task to achieve satisfactory QoS and reduce operation cost. Furthermore, fault tolerance against management component failure deserves serious attention, since even the slightest outage may cause immeasurable financial loss and impact customers' trust. Hence, an efficient and robust management solution is highly desired for large scale server farms, especially with the rapid expansion of cloud offerings these days.

Workload management for multi-tier server farms has been widely studied in the literature, e.g., admission control [1–4], QoS management [5–9], overload control [10,11], and novel workflow scheduling algorithms [12–14]. Although these approaches work well for centralized deployment, most of them are limited in managing large scale server farms due to scalability, complexity and fault tolerance issues. The centralized solutions, usually designed for an infrastructure with a small amount of proxies and application server suffers from scalability problems. A common practice to improve the scalability of the centralized solutions is to group proxies and application servers into small clusters, and apply the solution within each cluster instead. However, such a scheme increases management overheads, e.g., additional effort to retain load balance and traffic control among clusters, as well as administrators' effort to set up and manage each cluster. Actually, such a scheme also hinders the resource relocation and dynamic provisioning which are critical in a cloud environment. Furthermore, the centralized control approach also suffers from the well-known single point of failure problem. The controlling component going down would cause the whole cluster to malfunction, which is quite difficult to identify and resolve in a large scale server environment, thus hurting QoS. Usually, a backup solution and smooth transition to the backup solution should be carefully designed to preserve service assurance and minimize the QoS penalty, which makes the management issue further complicated.

In this paper, we propose a decentralized workload management system to achieve load balancing, overload protection and

* Corresponding author. Tel.: +1 814 206 4609.
E-mail addresses: suz116@psu.edu (S. Zhang), wenjiewang@pplive.com (W. Wang), haishwu@gmail.com (H. Wu), vasilako@ath.forthnet.gr (A.V. Vasilakos), pliu@ist.psu.edu (P. Liu).

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.10.004

QoS differentiation in large scale server farms. The control functionality is split and spread among all the servers and proxies, which perform resource management and traffic control automatically and independently. Specifically, each back-end server runs a sub-controller, which audits the local resource consumption, e.g., CPU capacity, memory usage, bandwidth availability, etc. Then, the sub-controller dispatches the available resources as *quotas* to the proxy tier based on traffic path regulations. The proxy tier strictly adheres to the resource dispatching decision of each sub-controller, to avoid overloading the back-end servers. Meanwhile, each proxy maintains multi-level priority queues for the incoming workloads with different QoS requirements. Based on SLA, the available resources are further dispatched to different queues to achieve service differentiation.

The proposed control framework is totally compatible with a multi-tier large server environment. Our solution does not depend on accurate modeling of workloads, deep instrumentation of the controlled system, or any change to the source code of commodity web applications, server software and the target operating systems. Thus, it is completely transparent to both the application servers and web applications, and involves little deployment effort even in large scale server farms. Driving the framework with some benchmarks and workload generators, we demonstrate that in the cloud environment, it is capable of managing thousands of servers for fundamental and realistic goals, e.g., overload protection, QoS differentiation and resilience to system component failure.

The rest of this paper is organized as follows. We introduce our workload management scenario and overview the proposed system in Section 2. The details of our control components are presented in Sections 3 and 4. In Section 5, we evaluate the framework under various workload, and evaluate the performance overhead and fault tolerance. Finally, we discuss related work in Section 6 and conclude in Section 7.

2. Management scenario and design overview

In this section, we start with the description of our management scenario and corresponding challenges, followed by the principles of our design. Then we present the overview of our decentralized approach with control functionality spreading among sub-controllers and proxies.

2.1. Management scenarios

Our workload management focuses on the network traffic bypassing proxy-tier to back-end application servers in the multi-tier web server architecture. Such an approach can also be applied to regulate the traffic flow between application servers and database servers. Our targets are to prevent overloading the critical resources of application servers, and to retain the QoS differentiation. Although dynamic resource provisioning, e.g., [15–18], can also work with the temporary workload boost, continuous peak workloads will run out all available resources. Hence, appropriate workload admission control must be performed in this scenario.

Fig. 1 shows a typical example of our workload management scenario, where the HTTP server tier is not shown for conciseness. Proxy-tier forwards incoming requests to back-end application servers based on request forwarding path regulation and resource availability of the servers. The problem of maximizing the resource usage without overloading can be solved by applying network flow theory, if global knowledge can be obtained. However, in large scale server farms with millions of requests flooding into a pool of proxies and available resources spreading on thousands of back-end servers, such knowledge is infeasible to be collected immediately by control components. Instead, each proxy always has to make traffic admission decisions based on its local view, which may hurt QoS and cause serious load imbalance.

From another perspective, virtual machine (VM) sharing is a common practice for efficient usage of cloud resources. We recognize three levels of resource sharing in our management scenarios, *application instance*[1] *sharing, virtual machine sharing*, and *physical server node sharing*, sorted from the application level to the bottom machine level. Specially, different proxies can route service requests to the same application instance for processing (i.e., application instance sharing); multiple application servers (e.g., Java Virtual Machine) could run in one virtual machine (i.e., virtual machine sharing); while multiple virtual machines can run on a physical server (i.e., physical server node sharing). The sharing of an upper level resource also indicates the sharing of a lower level resource. Actually, there is even resource competition within the same resource sharing level. For instance, requests to the same application may have different QoS requirements, usually stated in the SLA.

Due to the complexity of the resource sharing in large scale serve farms, the resource that one application instance can consume highly depends on other instance co-existing on the same physical server. Hence, it is challenging to dynamically make localized decisions on workload forwarding to achieve efficient resources sharing and desired QoS. Furthermore, the workload characteristic of applications may change significantly in a short period of time especially in cloud, e.g., flash crowd. In response to such changes, application instance relocation, allocation and revocation will be performed quickly and frequently.

The above workload dynamics and application placement management will gradually tangle the resource sharing among different levels, which further increases the complexity of the resource management problem. Such mesh-like request forwarding paths also impose the challenge of dynamically and precisely estimating the bottleneck resource consumption on application servers. Accurate modeling of back-end servers requires deep instrumentation of the corresponding servers, especially on the application server level. Even with deep instrumentation, the model may still need a long time to learn the resource capacity and the characteristics of workloads.

In this paper, QoS requirements are defined to be the combination of preferential fairness and average response time assurance. Such a combination implies that resources cannot be totally granted to the service requests with higher priority in a system overload scenario. This reflects the management principle of QoS assurance in practice, particularly for some enterprise private cloud. Instead of dedicating an abundant amount of resources for higher priority workloads and starving the lower priority ones, a certain degree of degradation can be tolerated in exchange for continuous service of all workloads. The service provider will have the flexibility to configure the degree of tolerance by means of business importance for workloads with different priorities.

2.2. Design principles

Our workload management scenario and the corresponding challenges require us to design a solution with the following properties.

QoS assurances. Based on SLA, priority should always be given to a workload with higher importance. However, upon overload, a pre-defined level of degradation can be tolerated, that is, a certain

[1] We refer to the presence of an application on an application server as an application instance.