The Journal of Systems and Software 115 (2016) 111–129

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

A QoS-aware self-correcting observation based load balancer

Veerabhadra Rao Chandakanna [a,*], Valli Kumari Vatsavayi [b]

[a] CSE Department, MVGR College of Engineering Affiliated to JNTUK, Kakinada, Vizianagaram 535005, Andhra Pradesh, India
[b] Department of Computer Science & Systems Engineering, AU College of Engineering, Andhra University, Visakhapatnam 530003, Andhra Pradesh, India

ARTICLE INFO

Article history:
Received 18 October 2015
Revised 9 January 2016
Accepted 29 January 2016
Available online 6 February 2016

Keywords:
Dynamic load balancing
Fault tolerance
Quality of Service

ABSTRACT

Any service offered by a *load balanced cluster* is deployed on every member of the cluster. The Sliding window based Self-Learning and Adaptive Load Balancer (SSAL) is an observation based load balancer that optimizes throughput. It gives single point entry to access any service hosted on the cluster. This paper proposes a QoS-aware and Self-correcting observation based Load Balancer (QSLB) that extends the SSAL to (i) prevent the single point of failure of the load balancer, (ii) manage the cluster capacity, (iii) support the QoS monitoring, and (iv) estimate the cluster capacity needed to meet the QoS benchmarks. Redundant QSLBs collaborate to estimate the capabilities of the individual cluster members, share the available cluster capacity, and monitor the QoS parameters. Two models to estimate the cluster capacity needed to meet the QoS benchmarks are proposed. Experiments were conducted to test the QSLB's features. The experimental results confirmed that (i) the overhead to support these QSLB features is minimal, (ii) the QSLBs retained their share of the cluster capacity even in dynamic environments, and (iii) using the recommended cluster capacity improved the QoS met percentage. The proposed model improves fault tolerance, assists in cluster capacity management, and monitors the QoS.

1. Introduction

In our day to day, we use distributed applications such as Gmail, Twitter, Facebook, WhatsApp, etc., without noticing that several components running on a network of computers are collaborating (by exchanging messages) to fulfill our requests. Such applications have huge computational demand and/or deal with huge amount of data. They are usually run on datacenters, or simply on a set of tightly connected computers working together and that can be logically treated as a single system. This computer cluster needs a load balancing hardware or software to distribute the requests efficiently. A *load balanced cluster* is an abstraction for a set of identical servers, that host same set of services. A request received by a load balancer can be served by any cluster member. The services can be easily scaled by provisioning new cluster members and adding them to the cluster. Although a cluster can be formed by non-identical cluster members (that host different services), the load balancing and scaling implementations become more complex, and it is not usually preferred to create a load balanced cluster. All the cluster members must host the same set of services (including their version numbers) to produce consistent results. A simple *load balanced cluster* with a load balancer, two clusters (Cluster1 and Cluster2), and J2EE instances as cluster members are shown in Fig. 1. Chandakanna and Vatsavayi (2014) proposed a framework (SACF) that keeps all the cluster members of a load balanced cluster in sync all the time. A *virtual address* is created to represent any member of a given load balanced cluster. The end users use the virtual address when sending their requests. Any request sent to the virtual address can be served by any member of the load balanced cluster. A load balancer uses a load balancing algorithm to maximize the throughput, or minimize the response time or overloading of any specific server.

Failover capability is provided to a service by replicating the service on multiple servers, and a load balancer that knows the set of servers that replicated the session state of a given request. The static load balancers assign the incoming requests to the servers based on pre-known capability ratios of the servers. Load Balancers based on the static load balancing algorithms such as *min–min, min–max* (Gopinath and Vasudevan, 2015) do not use the current state of the cluster members to distribute the load. Some of the dynamic load balancers periodically collect various attributes like CPU speed, memory available, CPU busy time/idle time, queue length, etc. to estimate the capabilities of the servers. The incoming requests are distributed based on the assessed servers' capabilities. Such systems typically use a monitoring entity at each cluster member to collect the performance attributes, and an analyzer

* Corresponding author. Tel.: +91 9492826715.
E-mail addresses: cvrao1972@gmail.com (V.R. Chandakanna), vallikumari@gmail.com (V.K. Vatsavayi).

http://dx.doi.org/10.1016/j.jss.2016.01.042
0164-1212/

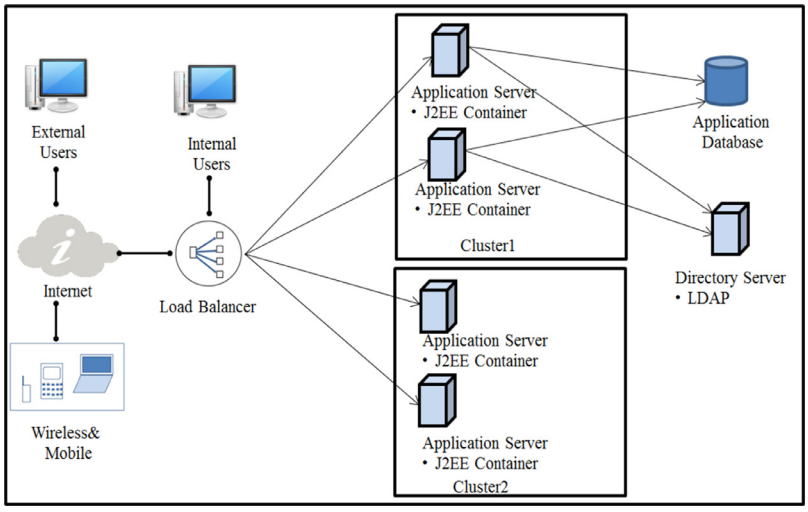

Fig. 1. J2EE clustered environment.

that analyzes the collected attributes and estimates the servers' capabilities. Many dynamic load balancing algorithms based on the *Bio-inspired, Game Theory-Based, Genetic Algorithms, Reliability Focused Load Balancers,* and *Observation Based models* are proposed in the literature.

We define (i) a *stable environment* where the server capabilities do not change overtime and the processing times of different requests are similar and (ii) an *unstable environment* where the server capabilities can change overtime and the processing times of different requests can widely vary. Static load balancing algorithms are well suited for the stable environments. Dynamic load balancing algorithms are well suited for the unstable environments. Most of the dynamic load balancing algorithms need to manage monitoring software on every server and collect feedback to know the state of the servers periodically to adjust the load distribution model. Observing the performance of the cluster members is another way to estimate the cluster members' capabilities. This form of dynamic load balancing is simple, as it neither requires managing the monitoring entity at each cluster member nor the data collected from each cluster member to be sent to the analyzer. The load balancers based on the Observation Based models do not need explicit feedback from the cluster members to know their current capabilities. They estimate cluster member capabilities based on the observations made by the load balancer such as the fast responding server, server with the least number of connections, etc. Commercial products like Citrix NetScaler (Citrix.com, 2015) and Big-IP (F5.com, 2015) support dynamic load balancing algorithms (citrix.com, 2015; Devcentral.f5.com, 2015) based on the fast response and the least number of connections observed by the load balancer. *The Observation Based load balancers* estimate the capabilities of a cluster member by using the observed response processing times of the requests sent to the server and/or the number of active connections currently made to the server. The *quickest response time based load balancer* tracks the cluster member that is currently producing the fastest response. An incoming request is assigned to the fastest cluster member known at that point. The *minimum connections based load balancer* keeps track of the number of active connections to each cluster member. An incoming request is assigned to the cluster member with the least number of connections. For each cluster member, the *lowest response time and minimum connections (LRTM) model* maintains two separate ranking orders, one based on the *number of active connections* and the other based on the *average response time*. An incoming request is assigned to the cluster member with the minimal total rank value. Chandakanna and Vatsavayi (2015) proposed a Sliding window based Self-learning and Adaptive Load balancer (SSAL), an observation based load balancer that can produce optimal throughput in both stable and unstable environments. The SSAL logically divides time into fixed size feedback intervals, and the observed performance of the cluster members in every feedback interval is used to make a correction to the load distribution model. It assigns enough requests to keep all the servers busy in every feedback interval and makes a correction after every feedback interval. The proposed architecture of the Sliding window based Self-Learning and Adaptive Load Balancer (SSAL) is shown in Fig. 2.

The SSAL is composed of a *Request Handler*, a *Self-Learning Scheduler* (SLS), a set of *Self-Learning Dispatchers* (SLDs), an *Input Queue*, and a set of *Output Queues* (one per server). The Request Handler receives the incoming requests and adds them to the Input Queue. The Self-Learning Scheduler (SLS) logically divides the time into fixed size intervals (feedback intervals). In each interval, the SLS assigns batch size(BS) number of requests to all the cluster members in proportion to the estimated capabilities of the servers. After every feedback interval, the SLS makes an incremental correction to the assessed servers' capabilities using the observed cluster members' performance in the previous interval. Each SLD process the next request from its Output Queue by sending it to its associated server and recording the performance of its server. All the SLDs periodically report their performance observations to the SLS.

1.1. Problem analysis

The end users do not have explicit knowledge about the servers processing their requests. They send their requests to the SSAL (load balancer) and the SSAL forwards the request to one of the servers. When the SSAL is down, none of the user requests can be processed, even though the servers are ready to process the requests. This creates a single point of failure for the system. To prevent the single point of failure, redundant load balancers (SSALs) are used in practice. The issues created by running multiple SSALs in parallel, additional features to enhance the SSAL are described in the following sections.

1.1.1. Estimate the capabilities of the servers

The performance of the servers observed by each SSAL needs to be periodically consolidated to estimate the overall capabilities of the servers. A correction model that uses the information