A Load Balancing Algorithm in Multi-tenancy Environment

Tao Zhao, Hailong Sun, Yu Tang, Xudong Liu
School of Computer Science and Engineering
Beihang University
{zhaotao,sunhl,tangyu,liuxd}@act.buaa.edu.cn

ABSTRACT

Multi-tenancy brings new challenges to load balancing, since it incurs resource competition and different QoS requirements of hosted applications. Therefore, servers with multiple deployed applications need a proper request scheduling policy to guarantee their quality of service, e.g., response time. However, when under heavy loads, mean response time of some applications may become too high to be acceptable due to the mutual intervention among tenants. In this work, we propose a new load balancing algorithm, "Server Throughput Restriction(STR)", based on M/G/s/s+r queueing model, in order to guarantee each application's mean response time.

Categories and Subject Descriptors

H.3.4 [**Information Systems**]: Systems and Software—*distributed systems*

General Terms

Algorithms, Design

Keywords

Cloud Computing, Multi-tenancy, Load Balancing

1. INTRODUCTION

Load balancing in cloud computing differs from that in traditional environments like cluster or grid computing due to the sharing of resources with multi-tenancy. Nowadays, PaaS providers such as Heroku and CloudBees isolate code from different applications on the same OS instance so as to improve resource utilization, which brings new challenge to load balancing as the server is shared by several applications. This way, some of the applications' response time may not be guaranteed due to the competition for shared resources. Therefore, in this work, we mainly focus on QoS-aware load balancing issues for multi-tenancy application environments in PaaS layer, and specifically we aim at ensuring that each application's mean response time should not be higher than its corresponding threshold in spite of resource competing among applications.

In this work, a new load balancing algorithm named STR is proposed for multi-tenancy environment with CPU-intensive applications. We assume that applications' request time distribution on an idle server is known a priori, then we use M/G/s/s+r queueing model to analyze the mean response time of each application and obtain the throughput restriction equation of each server based on the concept of traffic intensity.

2. THROUGHPUT RESTRICTION CALCULATION

By applying the M/G/s/s+r model, the mean response time of the application is positively correlated with the arrival rate of requests. In other words, once we restrict the maximum throughput that a server could handle, the mean response time of the deployed applications can be controlled as well. The following gives the mean response time of CPU-intensive application in closed form. Detailed definition of symbols is shown in Table 1.

$$\begin{aligned} W &= \mu + \frac{\mu R_G}{s\left(\frac{s!}{(\lambda\mu)^s}\sum_{j=0}^{s-1}\frac{(\lambda\mu)^j}{j!}\left(1-\frac{\lambda\mu}{s}\right)^2+1-\frac{\lambda\mu}{s}\right)} \\ &= \mu + \Delta(\lambda) \end{aligned} \tag{1}$$

2.1 Single-application Environment

For a server that has one deployed application, the maximum throughput of the application λ_{max} can be obtained by solving the following optimization equation:

$$\max_{\lambda} W \leq T_R \tag{2}$$

where T_R is the threshold of the application's mean response time. When the actual arrival rate is less than λ_{max}, all the requests would be dispatched to the hosting server and the mean response time is guaranteed.

2.2 Multi-tenancy Environment

In the context of multi-tenancy, a server has several deployed applications and the maximum throughput is needed so that each application's mean response time can be guaranteed. It is difficult to solve directly since applications do no have the same threshold. To begin with, consider the single-application situation that the maximum throughput λ_{max} can be easily obtained by the unique threshold. Next,

Table 1: The definition of symbols

Symbol	Definition
s	Number of CPUs
r	Request queue length
λ	Poisson arrival rate of requests
μ	Mean CPU time of the application
ρ	Traffic intensity
W	Mean response time of requests
R_G	The ratio of the mean waiting time in the M/G/s and M/M/s queues

λ_{max} can be transformed to the maximum traffic intensity by its definition $\rho = \lambda\mu/s$, which is the maximum intensity that each of the server's CPU can reach (although $\rho < 1$, it is not the CPU utilization). Thus each application's mean response time is also positively correlated with ρ. For the applications deployed on the same server, each of their corresponding traffic intensity in single-application environment can be calculated, then we sort them in ascending order because each value of ρ specifies a level of throughput restriction.

Let i be the index of the applications on a server, ρ_i be the calculated maximum traffic intensity in single-application environment, and λ_i be the arrival rate of requests. All applications can be seen as an integrated one with the arrival rate of $\sum \lambda_i$ and the mean CPU time of $\sum \lambda_i\mu_i / \sum \lambda_i$. We first choose ρ_1 be the server's traffic intensity, then the following equation can be obtained:

$$\sum \lambda_i \cdot \frac{\sum \lambda_i\mu_i}{\sum \lambda_i} = \sum \lambda_i\mu_i \leq s\rho_1, i = 1, 2, ..., n \qquad (3)$$

The right side of Equation 3 specifies the server's total throughput restriction, while the other side is the linear combination of each application's request arrival rate with weights that represent the mean CPU time. When the applications' loads are small or staggered, the linear combination of the arrival rates would satisfy the inequality. Therefore, they can be deployed on the same server with guaranteed mean response time, thus improves resource utilization and flexibility. We call Equation 3 the server's throughput restriction equation.

When the server's actual throughput reaches the first level of throughput restriction, additional requests can be served if and only if the request number of the application with traffic intensity ρ_1 is zero, thus we can raise the throughput restriction level by the following equation:

$$\sum \lambda_i\mu_i \leq s(\rho_2 - \rho_1), i = 2, 3, ..., n \qquad (4)$$

Generally, when the server's actual throughput reaches level $k (k = 0, 1, ..., n-1, \rho_0 = 0)$, additional requests can be served if and only if the request number of the applications with traffic intensity $\rho_1, ..., \rho_k$ is zero, then:

$$\sum \lambda_i\mu_i \leq s(\rho_{k+1} - \rho_k), i = k+1, k+2, ..., n \qquad (5)$$

3. LOAD BALANCING ALGORITHM

This section proposes our Server Throughput Restriction based load balancing algorithm, or STR in short, which is a time slot based strategy with available throughput reset to the throughput restriction at the beginning of each slot. Therefore, the throughput restriction in STR is equivalent to the limitation of number of requests with different weights in a time slot.

For each incoming request, STR will first extract the application context that the request belongs to from its HTTP head, which contains the request weight and the server list where the application is deployed, then dispatch the request to the server with maximum available throughput and simultaneously meets throughput restriction. In addition, each server's throughput restriction level can be raised up by the methods mentioned in Section 2.2. If no server is available, the request is rejected by the load balancer as its acceptance will lead to performance deterioration of some applications. The dispatching strategy is similar to the Worst-Fit algorithm in online bin-packing problem. Best-Fit strategy is not suitable here as servers are packed one by one, thus may cause unevenly distributed loads as applications may have different repetitions. The detailed description of STR is shown in Algorithm 1.

Algorithm 1 Load balancing with STR

```
 1: for each $request$ do
 2:     $context \leftarrow$ extractApplicationContext($request$)
 3:     $server\_list \leftarrow context$.server_list
 4:     $weight \leftarrow context$.weight
 5:     $server \leftarrow$ maxAvailableThroughput($server\_list$)
 6:     if $weight > server$.available_throughput then
 7:         NOT_ADMITT($request$)
 8:     else
 9:         update($server$.available_throughput,$weight$)
10:         send($request$,$server$)
11:     end if
12: end for
```

4. CONCLUSIONS

In multi-tenancy environment, applications are deployed on the same server to improve resource utilization. When under heavy loads, some applications' response time can exceed their threshold due to the mutual intervention among tenants. In this work, we propose a new load balancing algorithm named STR, which dispatches requests according to the throughput restriction based on the M/G/s/s+r queueing model, thus guarantees each application's mean response time demands.

5. ACKNOWLEDGMENTS

This work was supported by China 863 program (No.2012 AA011203), National Natural Science Foundation of China (No.61103031, No.61370057), A Foundation for the Author of National Excellent Doctoral Dissertation of PR China, Beijing Nova Program.

6. REFERENCES

[1] J. M. Ewing and D. A. MenascÂt'e. Business-oriented autonomic load balancing for multitiered web sites. In *IEEE International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, London, UK, September 2009.

[2] S. Walraven, T. Monheim, E. Truyen, and W. Joosen. Towards performance isolation in multi-tenant saas applications. In *Proceedings of the 7th Workshop on Middleware for Next Generation Internet Computing*, Montreal, Quebec, Canada, December 2012.