Bottleneck Detection and Solution Recommendation for Cloud-Based Multi-Tier Application

Jinhui Yao[1] and Gueyoung Jung[2]

[1] Palo Alto Research Center (PARC), USA
jinhui.yao@xerox.com
[2] AT&T Research Labs, USA
gjung@research.att.com

Abstract. Cloud computing has gained extremely rapid adoption in the recent years. In the complex computing environment of the cloud, automatically detecting application bottleneck points of multi-tier applications is practically a challenging problem. This is because multiple potential bottlenecks can co-exist in the system and affect each other while a management system reallocates resources. In this paper, we tackle this problem by developing a comprehensive capability profiling of such multi-tier applications. Based on the capability profiling, we develop techniques to identify the potential resource bottlenecks and recommend the additional required resources.

Keywords: Cloud, Bottleneck Detection, Multi-tier Application.

1 Introduction

Cloud computing has gained extremely rapid adoption in the recent years. Enterprises have started to deploy their complex multi-tier web applications into these clouds for cost-efficiency. Here, the cloud-based multi-tier application consists of multiple software components (i.e., tiers) that are connected over inter- and/or intra-communication networks in data centers. Detecting application bottleneck points of multi-tier applications is practically a challenging problem, and yet it is a fundamental issue for system management. Hence, it is desirable to have a mechanism to monitor the application performance changes (e.g., application throughput changes) and then, to correlate system resource usages of all components into the application performance saturation for system diagnosis.

However, automatically pinpointing and correlating bottlenecked resources are not trivial. One of important factors we should focus on is that multiple potential bottlenecks can co-exist typically by oscillating back and forward between distributed resources in the multi-tier applications[6, 9], and they affect each other while a management system performs resource reallocations to resolve the immediate bottlenecks observed individually. Therefore, certain potential and critical bottlenecks may not be timely noticed until other bottlenecks are completely resolved. In this paper, we tackle this problem by developing a comprehensive capability profiling of such multi-tier

X. Franch et al. (Eds.): ICSOC 2014, LNCS 8831, pp. 470–477, 2014.

applications in the cloud. Based on the capability profiling, we develop techniques to identify the potential resource bottlenecks and recommend the required resources to provide adequate performance without the bottleneck oscillation between current and potential bottleneck resources.

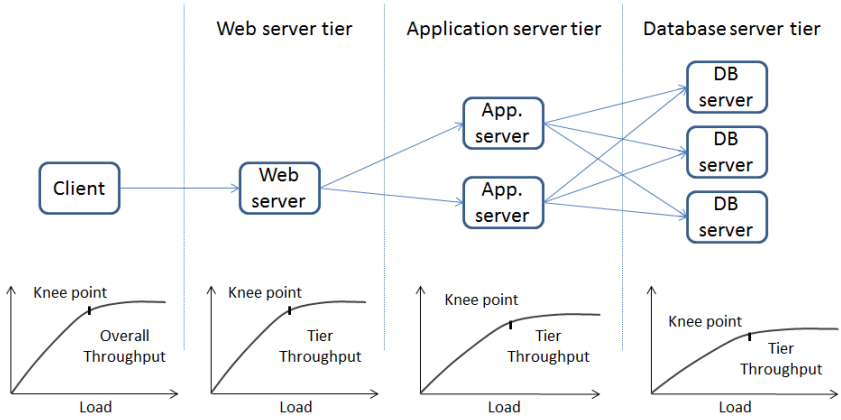

Fig. 1. A 3-tier web transaction application and throughput curves of tiers

2 Bottleneck Detection Using Knee Point Detection

The upper part of Figure 1 illustrates a 3-tier web transaction application that consists of the front-end web server, the middle application servers, and the back-end database servers. Servers in the middle and the back-end tiers handles web transactions in parallel. The lower part of Figure 1 illustrates bottleneck patterns of these tiers. The bottleneck pattern of the application throughput can be described as a knee point of throughput curve, while the workload to the application increases.

As shown in these bottleneck patterns, despite of the initial rapid increase in the application throughput, after the knee point, the throughput cannot increase further because some of system resources of tiers are bottlenecked. Similar bottleneck patterns are shown in all tiers, but at different observation points. This is because a bottleneck in one of the tiers will eventually trigger bottleneck patterns in the other tiers. With these observations, it is important to first capture such knee points, which represent the starting point of bottleneck pattern, of all involved tiers and resource usages of each tier. Then, we can identify the bottleneck causes, in the context of system resources, by analyzing the temporal relations among the bottleneck patterns of all tiers and resource.

2.1 Individual Knee Point Detection

The application throughput can be defined as the number of user requests that successfully get through all tiers. The throughput of the application will keep increasing as the load increases until a certain point, after that point, throughput cannot increase further

more because the system bottleneck occurs. Figure 2 illustrates the bottleneck pattern of the throughput. In the figure, we have plotted normalized throughput of the application against normalized load to the application. To capture the knee point, our system first generates a linear line that connects the first measurement point to the last measurement point and then, computes its length (i.e., z in the figure). Second, at each measurement point according to the measurement window size, we compute the length of the orthogonal line drawn from the linear line to the measurement point (i.e., the height h_k in the figure, where k is each measurement point). To compute the height of each measurement point, it generates two lines and computes their lengths (i.e., x_k and y_k in the figure). First line is drawn from the first measurement point to the current measurement point, and the second line is from the current measurement point to the last measurement point. Then, using cosine rule and sine rule, the height is computed as following,

$$h_k = x_k \sin(\cos^{-1}((x_k^2 + z^2 - y_k^2)/2x_k z))$$

Finally, the knee point is the measurement point that has the highest height from the linear line among all measurement points. And this knee point indicates the capability of this application (i.e., potential bottleneck starting point of the application or the tier being considered).

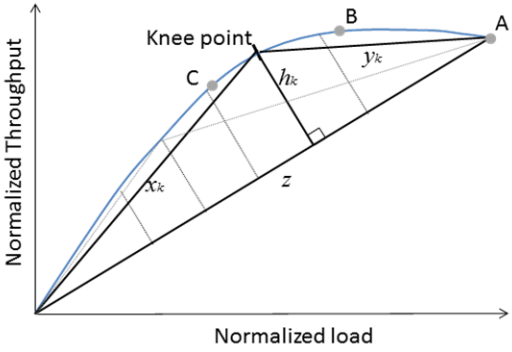

Fig. 2. Bottleneck pattern

2.2 Performance Profiling for Identifying the System bottlenecks

In our approach, we capture the change rate of resource usage (i.e., slope) of each resource type and the change rate of workload throughput until the capability is reached (i.e., knee point), while load increases. The resource usage change rate before the knee point can approximately indicate the degree of contribution of each resource type to the throughput change and the performance capability. These change rates are directly used to build our performance model of the multi-tier application, which is used to infer which resources are the current and potential bottlenecks.

Figure 3 shows the change rates of resource usages and three representative resource types, while load increases over the time. In this illustration, the change rate of CPU is higher than memory usage and disk I/O. It can indicate that CPU contributes more to the workload throughput than memory and disk I/O, and CPU can be bottlenecked first on its knee point. Note that the knee points of three resource types occur at different measurement points.

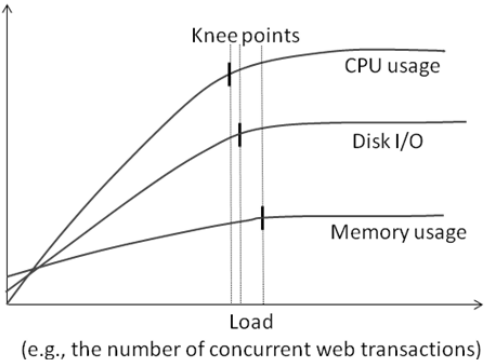

Fig. 3. The change rates before knee points

As shown in Figure 3, throughput increases until the performance capability is reached at knee point. The performance capability is determined by some resource types that consume the most of their available capacities (i.e., bottlenecked).Following this intuition, one straightforward method to find the bottlenecked resource is to sort the knee points to find out which one occurs first. However, a more challenging question will be to find the capability gap (i.e., resource shortage), between the currently bottlenecked resources and the required performance. The capability gap will indicate the additional resource needed for such potential bottleneck resources. And then, it will indicate when the bottleneck is transferred to the other potential bottleneck resources after the management system resolves the current bottleneck by allocating the additional amount of resources. Our system tackles this challenge by defining a quantitative performance model for each individual resource type to identify its correlation to the performance capability of application. Specifically, for each resource type *j*, a quantitative performance model can be defined as,

$$T_j = f(U_j | (C_j = c, \exists j \in R) \wedge \ (C_{r'} = \infty, \ \forall r' \in R, r' \neq j)) \quad (1)$$

, where T_j is the application throughput to be achieved with the normalized resource usage rate, U_j, over given resource capacity (i.e., $C_j = c$) of a resource type *j*. *R* is a set of all resource types for the application, and *r'* is a resource type in *R*, where *r'* is different resource type from *j*. We consider *r'* has unlimited capacities so that we can compute the correlation of only *j* to T_j.

While the throughput of the system is determined by the usage of different resources, the resource usage itself is driven by the amount of the load that the system is undertaking. The correlation between the load and the resource usage can be defined as a linear function (note that the correlation between the load and the resource usage can be defined as a non-linear function, however, we have focused on the resource usage before the knee point in our performance modeling, and observed the linear function):

$$U_j = \alpha_j L + \gamma_j \quad (2)$$

, where *L* is the amount of load, α_j is the change rate of resource usage (e.g., a slope in a linear function), and γ_j is an initial resource consumption in the system. We can

obtain α_j and γ_j by calibrating the function to fit into actual curve observed. In this fitting, we use the change rate of resource usage before knee point.

According to Equation 1, the throughput T_j equals to a function of the resource usage U_j, given that all other resource types have unlimited capacities (i.e., $C_{r'} = \infty$). Therefore, this implies that T_j reaches its maximum when the resource being considered is completely utilized (e.g.,$U_j = 1$, when it is normalized). Thus, from Equation 1, we can derive the maximum load L_j^{max}, which this resource can undertake at its knee point, as follows:

$$L_j^{max} = \frac{1-\gamma_j}{\alpha_j} \tag{3}$$

We can compute the maximum loads of all different types of the resources with the same way, to produce a set of maximum loads as $\{L_1^{max}, L_2^{max}, L_3^{max} \ldots, L_n^{max}\}$, where n is the number of resource types in the system. Once we have obtained the set of all maximum loads, finding the bottleneck resource, intuitively enough, is to find the resource that has the lowest maximum load, since the resource having the lowest maximum load has the earliest knee point$r^{Bottleneck} = \text{argmin}_j L_j^{max}$.

3 Estimating Resource Shortages for Potential Bottlenecks

Using the performance model (Equations 1-3), we can identify the bottleneck in the current configuration of the multi-tier application. However, other resource types may potentially become the next immediate bottleneck after an additional amount of the bottlenecked resource is allocated. Since multiple potential bottlenecks may co-exist in the system configuration as a form of bottleneck oscillation [6, 9], it is necessary to evaluate the total shortage of all resource types of interest in order to consistently achieve the target throughput. These resource shortages indicate the gap between the amount of resource needed and the amount that is currently utilized.

Based on Equation 1 and 2 in the previous section, we can follow the same intuition to define the correlation between the load L and the throughput T of each component, before reaching the knee point of the throughput curve, as a linear function.

$$T = \beta\ L \tag{4}$$

, where L is the amount of load andβ is the change rate of throughput. Similarly, we can obtain β by calibrating the function to fit into an actual curve. As mentioned earlier, the correlation between the load and the application throughput can be defined as a non-linear function. However, we have focused on the throughput before the knee point in our performance modeling, and observed the linear function is a good approximation while calibrating the function. By substituting Equation 4 into Equation 2 in the context of L, we have

$$U_j = \frac{\alpha_j}{\beta} T + \gamma_j \tag{5}$$

In Equation 6, if we define the target throughput as T^*, and the required performance capability value as U_j^* of a resource type j (i.e., the usage rate U_j required to achieve T^*), we can replace T and U_j with T^* and U_j^*, respectively, in the equation. Here, (α_j / β) indicates the normalized increase rate of the resource usage to increase a unit of throughput. Thus, the equation indicates how much resource capability is required to meet T^*. Note that if U_j^* is more than 1, it indicates that more resource capability is required to meet T^* than currently available in the configuration, and in this case the normalized resource shortage is thus $U_j^{\Delta} = U_j^* - 1$. With this equation, the required capability of component x, defined as $U^{x,*}$, for the workload and its throughput goal is a set of such required performance capability values:

$$U^{x,*} = \{U_1^*, U_2^*, U_3^*, \ldots, U_n^*\} \quad (6)$$

, where n is the number of resource types in the component being considered. Then, the capability shortage for all corresponding resource types in Equation 6 is represented as $U^{x,\Delta} = \{U_1^{\Delta}, U_2^{\Delta}, U_3^{\Delta}, \ldots, U_n^{\Delta}\}$. Similarly, the same method can be applied to all other components in the multi-tier application.

4 Preliminary Evaluation

To evaluate our approach, we have used an online auction web transaction workload, called RUBiS (http://rubis.ow2.org) that is deployed as a 3-tier web application including Apache web server, Tomcat servlet server, and back-end MySQL database server. The workload provided by RUBiS package consists of 26 different transaction types such as "Home," "Search Category". Some of transactions need database read or write transactions, while some of them only need HTML documents. We have created a database intensive workload by increasing the rate of database read/write to making the MySQL server tier being bottlenecked.

Figure 4 shows 3 throughput curves of Apache web server, Tomcat server, and MySQL database server. The figure points out 3 knee points (the red circles in the figure) that have been computed by the technique described in Section 2.1. The earliest knee point has been observed in the database tier as shown in the figure, and it correctly indicates the database tier is bottlenecked for the database intensive workload. As mentioned above, we intentionally set up the workload to make the database server bottlenecked. Note that throughputs of 3 servers are different since, by workload setup, some user requests are controlled not to go through all tiers. As shown in Figure 5, obviously, CPU is the bottlenecked resource type in the current configuration. This can be identified by computing the earliest knee point, similar with the way of identifying bottlenecked tier above. Note that the knee points of disk I/O and network I/O are located at the last measurement points. This is because there are no obvious knee points of these resource types, so the last measurement point is used.

Alternatively, we can identify the bottlenecked resource type by computing the maximum load that each resource type can handle as described in Section 2.2. The result is the set {925.5, 2762.5, 15840.4, 79204.4}, which represents $\{L_{CPU}^{max}, L_{Mem}^{max}, L_{NW}^{max}, L_{disk}^{max}\}$ as the maximum load of CPU, memory, network, and disk, respectively. It also shows

that CPU is the bottlenecked resource type because L_{CPU}^{max} has the lowest maximum load. When we see the maximum throughput of the database tier in Figure 4, it shows the similar amount of load at the knee point. Therefore, it indicates that our performance model is accurate enough to compute the resource shortage. Note that we have also measured their source usages in Web and App tiers, and observed significant under-utilizations of all resources so that they have very high maximum loads.

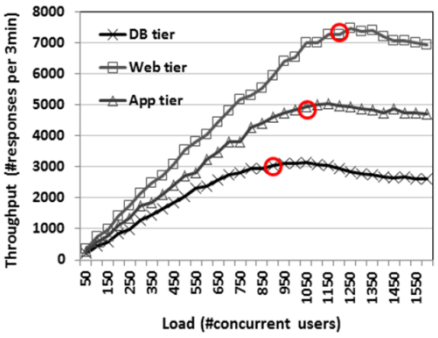

Fig. 4. Knee points of 3-tier application

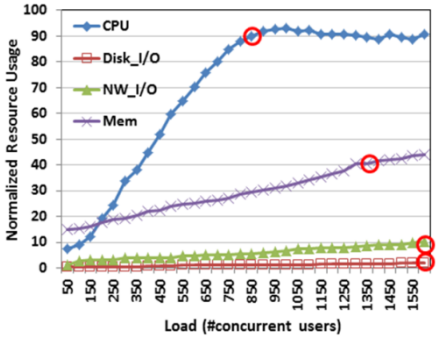

Fig. 5. Resource usages in the DB tier

5 Related Work

Cloud has gathered pace, as most enterprises are moving toward the agile hosting of multi-tier applications in public clouds, many researchers have focused on three different research directions: 1) updating application architecture to move from legacy systems to clouds [3], 2) evaluating different clouds' functional and non-functional attributes for allowing cloud users to correctly make a decision on which cloud to host applications [2, 4, 7, 8], and 3) efficiently orchestrating virtual appliances in a cloud, which may also include negotiations with cloud users. While some highly related previous work has principally focused on estimating rudimentary cloud capabilities using benchmarks [8] and automated performance testing [9],our approach focuses on the precise characterization of application capabilities in a cloud infrastructure.

Analytical models like [1, 5]have been proposed for bottleneck detection and performance prediction of multi-tier systems. They predict system performance based on burst workloads and then, determines how much resource to be allocated for each tier of the application for the target system response time. And there are numerous efforts that have addressed the challenges of managing cloud application performance. For example, [10, 11, 12] are based on very detailed understanding of the system resource utilization characteristics. Performance management solutions like AzureWatch (http://www.paraleap.com/azurewatch) continuously monitor the utilization of the various resource types and send a notification once they are saturated.

6 Conclusion and Future Work

In this paper, we presented an approach to identify the bottleneck resource and to provide the view of potentially co-existing bottlenecks in the cloud-based multi-tier applications. We developed a comprehensive modeling to profile the resource capabilities of the target system. Based on this profiling, we correlated the performance degradations to the resource bottlenecks. The preliminary evaluation results show that our approach is feasible to be used for bottleneck detection and resource shortage estimation for cloud recommender system.

References

1. Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P.: Dynamic provisioning of multi-tier internet applications. In: Int. Conf. on Autonomic Computing, pp. 217–228 (2005)
2. Jayasinghe, D., Malkowski, S., Wang, Q., et al.: Variations in performance and scalability when migrating n-tier applications to different clouds. In: Int. Conf. on Cloud Computing, pp. 73–80 (2011)
3. Chauhan, M.A., Babar, A.M.: Migrating service-oriented system to cloud computing: An experience report. In: Int. Conf. on Cloud Computing, pp. 404–411 (2011)
4. Cunha, M., Mendonca, N., Sampaio, A.: A declarative environment for automatic performance evaluation in IaaS clouds. In: Int. Conf. on Cloud Computing, pp. 285–292 (2013)
5. Casale, N.M., Cherkasova, G.L., Smirni, E.: Burstiness in multi-tier applications: symptoms, causes, and new models. In: Int. Conf. on Middleware, pp. 265–286 (2008)
6. Wang, Q., Kanemasa, Y., et al.: Detecting Transient Bottlenecks in n-Tier Applications through Fine-Grained Analysis. In: Int. Conf. on Distributed Computing Systems, pp. 31–40 (2013)
7. Calheiros, R., Ranjan, R., Beloglazov, A., DeRose, A.C., Buyya, R.: CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience 41, 23–50 (2011)
8. Yao, J., Chen, S., Wang, C., Levy, D., Zic, J.: Accountability as a service for the cloud. In: IEEE Int. Conf. on Services Computing (SCC), pp. 81–88 (2010)
9. Malkowski, S., Hedwig, M., Pu, C.: Experimental evaluation of n-tier systems: Observation and analysis of multi-bottlenecks. In: Int. Sym. on Workload Characterization, pp. 118–127 (2009)
10. Abdelzaher, T.F., Lu, C.: Modeling and performance control of internet servers. In: Int. Conf. on Decision and Control, pp. 2234–2239 (2000)
11. Diao, Y., Gandhi, N., Hellerstein, J.L., Parekh, S., Tilbury, D.M.: Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache web server. In: Network Operation and Management Symposium, pp. 219–234 (2002)
12. Diao, Y., Hu, X., Tantawi, A.N., Wu, H.: An adaptive feedback controller for sip server memory overload protection. In: Int. Conf. on Autonomic Computing, pp. 23–32 (2009)