An Improving Fault Detection Mechanism in Service-Oriented Applications based on Queuing Theory

Yang Shuo
School of Software,Shanghai Jiaotong University
Email:daivdsure@gmail.com

Hao-peng Chen
School of Software,Shanghai Jiaotong University
Email:chen-hp@sjtu.edu.cn

Abstract

SOA has become more and more popular, but fault tolerance is not fully supported in most existing SOA frameworks and solutions provided by various major software companies.SOA implementations with large number of users, services, or traffic, maintaining the necessary performance levels of applications integrated using an ESB presents a substantial challenge, both to the architects who design the infrastructure as well as to IT professionals who are responsible for administration. In this paper, we improve the performance model for analyzing and detecting faults based on the queuing theory[6]. The performance of services of SOA applications is measuring in two categories (individual services and composite services).We improve the model of the individuals services and add the composite services performance measuring.

1. Introduction

Serviced-oriented Architecture (SOA) is a novel methodology for systems development to provide services to either end-user applications or to other services distributed in a network. Services are the natural evolution of object-oriented and component-oriented programming models, and web services are becoming the prominent paradigm for electronic business and interoperable applications across heterogeneous systems.

One of the important characteristics of SOA-based applications that are different from traditional software is dynamic discovery and composition of services [1]. This means that instead of spending a lot of time on designing and coding, we build these applications based on SOA through service discovery and composing. In SOA, a service encapsulates reusable business functionalities with platform-independent interface contracts. A well constructed, standards-based SOA can empower a business environment with a flexible infrastructure and processing environment. And the word “dynamic” just means SOA-based applications should have the ability to discover and composite services into themselves in runtime.

It is possible to build non-SOA applications using Web Services, so it is wrong to assume that Web Services imply SOA. However, in a complex SOA application, the communication between services can be simply point-to-point, and also multi-point to multi-point in a complex SOA [2]. Another important development in the SOA environment is an Enterprise Service Bus (ESB). ESB is a standards-based integration platform that combines messaging, web services, data transformation, and intelligent routing in a highly distributed environment. The objective of an ESB is to route messages between resources in a reliable manner; that is, it guarantees message delivery. Message routing may be done synchronously or asynchronously between source and target systems. Messages may be also be transformed from a source format into a target format as they pass through the bus. Other facilities offered by an ESB include load balancing and failover. ESB middleware is available for both the Microsoft. NET and Java J2EE environments. This middleware may support a variety of SOA broker technologies such as Web services, Java JCA, Microsoft DCOM and CORBA. The advantages that an ESB brings to the SOA environment are security, reliability, scalability and the ability to interconnect older SOA broker technologies with Web services [3].

Dynamic discovery and composition requires four capabilities: (a) to identify existing services that fail to satisfy the functional or performance requirements for SOA-based applications; (b) to generate queries to locate alternative services that could replace existing ones; (c) to efficiently execute these queries at runtime; (d) to dynamically replace existing services during application execution [4]. So it is necessary for the applications to detect faults and locate the alternative services to replace existing one that fail to satisfy the functional or performance requirements at runtime.

As the four capabilities above, the performance of SOA-based applications is definitely important to dynamic discovery and composition. It can be seen that the SOA performance problem falls into two broad categories: ensuring sufficient performance of individual services as well as of the composite services [5]. Individual services provide service interfaces that encapsulate existing systems, ensuring their performance necessitates managing the

1. This paper is supported by the National High-Tech Research Development Program of China (863 program) under Grant No. 2007AA01Z139

978-0-7695-3499-2/08 $25.00 © 2008 IEEE
DOI 10.1109/SOSE.2008.46

performance of the components, applications, and systems that lie beneath the services abstraction. Well-established capacity planning methods, techniques and tools can be leveraged to manage the performance of individual services, such as logging-based instrumentation [7], or simulating the load on service interfaces by load testing in a similar way in simulating traditional web application performance [8].

Based on the research in [6], among the four capabilities required by dynamic discovery and composition, fault detection, the capability to identify the failed existing services, takes precedence over the other three, because successful fault detection is the guarantee for them to be employed properly. As we have mentioned, the purpose of fault detection is to identify two kinds of services: the services that fail to satisfy functional requirements and the services fail to satisfy performance requirements. We focus on the latter kind of services, because the failure of the former kind of services may be caused by semantic errors which have to be manually identified and corrected. Meanwhile, the failure of the former kind of services can be automatically detected via appropriate mechanisms. We put forward a fault detection mechanism, which is based on the queuing theory, to detect the services that fail to satisfy performance requirements. We also give a reference service model and one reference architecture of fault-tolerance control center of ESB based on our fault detection mechanism. But the queueing theory model used in [6] is not suitable and comprehensive enough, as it doesn't including the situation of the composite services. Dealing with the performance issues of services invoked in SOA applications always fall into the second category and it is far more complex than that of atomic services.

In this paper, we put forward an improved fault detection mechanism, which is based on the queuing theory, especially queuing networks, to detect the services that fail to satisfy performance requirements.

2. Rationale of this mechanism

The purpose of our mechanism is to detect the services which have not enough capability to satisfy the performance requirements specified by application assemblers. Performance, to put it simply, is how quickly the system can respond to a given logical operation from a given individual user. Response time is a measure of the amount of time the system consumes while processing a user request, which is made up of three parts: latency, which is the amount of time spent processing overhead just to get to the point of carrying out a service; wait time, which is the time spent waiting for the service, or, while the service is executing, the time spent waiting for resources; and service time, which is the time needed to process the request when no waiting is involved [9].

As we analysis in [6], we can find out that the service requests processing has the following six features :

1. The interarrival times between any two successive service invocation requests are independent of each other and have a common distribution.
2. The clients would receive responses if requests are processed by service in time, or receive exceptions due to the timeout of waiting. They even could abort the service invocation requests as their wills.
3. The service times needed for every request are not only dependent on the status of services, but also identically distributed. Furthermore, they are independent of interarrival times.
4. The requests can be served in many possible orders, such as first come first served, last come first served, shortest processing time first, random order, round robin, and so on. However, the first come first served is still the predominant order.
5. There may be a single service instance or a group of service instances processing the requests. Thus, there are several possible kinds of service capacity.
6. Since the cache or buffer of the service hosting environment is finite, the number of waiting requests is limited. It means if the waiting room of a service hosting environment is fully occupied, when extra requests arrive at this service, they would be lost.

The above six features have shown that the model of service requests processing is a typical queuing model, so we can resort to queuing theory to establish our fault detection mechanism.

We add a Service QoS Descriptor in control enter of Enterprise Service Bus (ESB), which contains the several variables(containing the original mean response time , updating mean response time) and acceptable nonnegative error e and acceptable number of successive failed periods n of each service in system. When the application works, we continuously calculate the mean number of requests waiting for serving, the mean waiting time of requests, and the mean sojourn time of requests in a period which contains certain amount of unit time, and compare these real-time variables with the expected variables stored in Service QoS Descriptor. There must be difference between these two set of variables, especially we focus on the difference between the expected sojourn time and the real-time mean sojourn time. If the difference exceeds e for n times, we consider the corresponding service

deviates its steady state, and as a result, it cannot satisfy the performance requirement any longer.

3. The algorithm this mechanism employs

There are several items to discriminate different queues, including: the distribution of the arrival times between service invocation requests, the distribution of service time, and the number of service instances. [10]

In SOA-based applications, the number of service invocation requests and the service time has Poisson or exponential distribution, so the queues have an important property: memoryless property. By convention, we use M to respectively indicate the number of service invocation requests and the service time.

For the services, there are several instances to serve the clients. For example, if the service is implemented as a servlet or EJB in J2EE, there is an instance pool in application server to manage its multiple instances. These instances are parallel instances, which mean each instance serves only one client at any time. By convention, we use S to indicate the number of service instances.

For any application server, its capacity is limited. So the services hosting in application servers have the upper limitation of the number of clients they could serve. This upper limitation is the sum of the number of service instances and the number of waiting requests the queue could hold. By convention, we use k to indicate the service capacity.

We respectively analysis the performance of performance of individual services and the composite services.

First, we use the queue model M/G/S/K to analysis the individual services. We can use its features to analyze services and detect the performance faults. We use the following symbols to indicate the basic conceptions in queuing theory:

1. λ: indicates arrival rate, which denotes the rate at which requests arrive at the service.
2. μ: indicates service completion rate, which denotes the rate at which responses depart from the service.
3. ρ: indicates the occupation rate or server utilization, which denotes the fraction of time the server is working.
4. p_j : indicates the probability that there are j requests in a queue when the queue reaches its steady state. In particular, P0 denotes there is no request in the queue, which means the requests could be served immediately after they arrive in the queue and need not to wait.
5. L: indicates the expected length of queue, which equals to the mean number of requests in service and the mean number of requests in the queue.
6. L_q : indicates the expected length of waiting queue.
7. W: indicates the expected sojourn time.
8. W_q: indicates the expected waiting time.

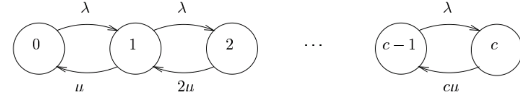

Figure1: Flow diagram for M/M/c/c model

1. We set the unit time as 1 second, so in every second, we record the number of requests arriving in the service λ_i , the number of requests departing from each instance of the service $\mu_{i,j}$. For every period, such as every 100 seconds, we calculate the λ and μ by the following formulae:

$$\lambda = \frac{1}{n}\sum_{i=1}^{n}\lambda_i \quad (1)$$

$$\mu = \frac{1}{Sn}\sum_{i=1}^{n}\sum_{j=1}^{S}\mu_{i,j} \quad (2)$$

2. Calculate ρ by the following formula:

$$\rho = \frac{\lambda}{S\mu} \quad (3)$$

3. Calculate p_n by the following formula :

$$p_n = \frac{\frac{\left(\frac{\lambda}{\mu}\right)^n}{n!}}{\sum_{n=0}^{c}\frac{\left(\frac{\lambda}{\mu}\right)^n}{n!}} = \frac{\frac{\rho^n}{n!}}{\sum_{n=0}^{c}\frac{\rho^n}{n!}} \quad (4)$$

4. Hence, the so-called blocking probability B(c, ρ)

$$B(c,\rho) = p_k = \frac{\frac{\rho^k}{k!}}{\sum_{n=0}^{k}\frac{\rho^n}{n!}} \quad (5)$$

5 Calculate L by the following formula:

$$L = \sum_{j=0}^{k} j p_j \quad (6)$$

6 Calculate L_q by the following formula:

$$L_q = \sum_{j=0}^{k-s} j p_{s+j} \quad (7)$$

7 Calculate effective arrival rate by the following formula:

$$\lambda_e = \lambda(1 - P_k) \quad (8)$$

8 Calculate W by the following formula:

$$W = \frac{1}{\lambda_e} L \quad (9)$$

9 Calculate W_q by the following formula:

$$W_q = \frac{1}{\lambda_e} L_q \quad (10)$$

10 Compare W or W_q with W’ or W_q’ stored in service QoS Descriptor (it depends on which one is more important for users) to check

$$W - W' \le e \quad (11)$$

Or

$$W_q - W_q' \leq e \ (12)$$

11. If the inequation in step 10 is true, than clear the counter C, which counts the number of successive failed periods. Then repeat to step 1.

12 If the inequation in step 10 is false, increase C by 1, and compare C with n, if C is not greater than n, repeat to step 1; otherwise, we consider that the service can not satisfy our performance requirement. As a result, we successfully detect a fault by our criteria.

When all instances are busy, the incoming requests would be lost. So for some critical system in which the loss of requests is forbidden, B(c, ρ) is also an assistant parameter to determine whether the service works well.

So far we have only looked at the individual services, the second situation is focus on the composite services. The performance model here is solved using the mean value analysis algorithm for multi- class closed system [11]. Now we assume R job classes and K = (K_1, K_2,…K_R)

The computation of performance measures is as follows

1. $\pi_i(k)$ is the marginal probability that there are exactly $S_i = k$ jobs at node I is given by:

$$\pi_i(k) = \frac{F_i(k)}{G(K)} G_N^{(i)}(K-k) \quad (13)$$

Where

$$F_i(k) = \begin{cases} k_i! \frac{1}{\beta_i(k_i)} \left(\frac{1}{\mu_i}\right)^{k_i} \prod_{r=1}^{R} \frac{1}{k_{ir}!} e_{ir}^{k_{ir}} & , \text{Type} - 1 \\ k_i! \prod_{r=1}^{R} \frac{1}{k_{ir}!} \left(\frac{e_{ir}}{\mu_{ir}}\right)^{k_{ir}}, & \text{Type} - 2,4 \\ \prod_{r=1}^{R} \frac{1}{k_{ir}!} \left(\frac{e_{ir}}{\mu_{ir}}\right)^{k_{ir}}, & \text{Type} - 5 \end{cases} \quad (14)$$

, $G_N^{(i)}$ can be interpreted as the normalization constant of the network with k jobs and node i removed from the network.

2. Calculate the throughput of node i in the load-dependent or load- independent case is given by the formula:

$$\lambda(K) = \frac{G(K-1)}{G(K)} \ (15) \text{ and } \lambda_i(K) = e_i \frac{G(K-1)}{G(K)} \ (17)$$

3. Calculate the workload of the node i ,

$$\rho_i = \frac{\lambda_i}{m_i \mu_i} = \frac{e_i}{m_i \mu_i} \frac{G(K-1)}{G(K)} \ (18)$$

4 Calculate the mean number of jobs

$$\overline{K_i} = \sum_{k=1}^{K} \left(\frac{e_i}{\mu_i}\right)^k \frac{G(K-k)}{e_i G(K-1)} \ (19)$$

5 Calculate the mean response time if jobs at node I can be determined with the help of Little's theorem

$$\overline{W_i} = \frac{\overline{K_i}}{\lambda_i} = \sum_{k=1}^{K} \left(\frac{e_i}{\mu_i}\right)^k \frac{G(K-k)}{e_i G(K-1)} \ (20)$$

6 Compare W_i with W_i' stored in service QoS Descriptor (it depends on which one is more important for users) to check

$$W_{qi} - W_i' \leq e \ (21)$$

7 If the inequation in step 10 is true, than clear the counter C, which counts the number of successive failed periods. Then repeat to step 1.

8 If the inequation in step 10 is false, increase C by 1, and compare C with n, if C is not greater than n, repeat to step 1; otherwise, we consider that the service can not satisfy our performance requirement. As a result, we successfully detect a fault by our criteria.

4. Case Study

In the first case, supposed we have an SOA –based application, and we invoke web services independently, we suppose this service is of M/G/6/6

values	0	1	2	3	4	5	6
Numbers of λ_i	17	25	25	16	11	5	1
Numbers of $\mu_{i,j}$	180	164	137	38	8	2	0

Table 1: The numbers of various values of λ_i and $\mu_{i,j}$

We calculate λ

$$\lambda = \frac{1}{n} \sum_{i=1}^{n} \lambda_i = 1.98$$

Then , calculate μ :

$$\mu = \frac{1}{Sn} \sum_{i=1}^{n} \sum_{j=1}^{S} \mu_{i,j} = 0.99$$

Then occupation rate ρ is :

$$\rho = \frac{\lambda}{S\mu} = 1/3$$

$$B(c, \rho) = p_6 = 0.011$$

Then λ_e is

$$\lambda_e = \lambda(1 - P_k) = 1.76$$

Then L is

$$L = \sum_{j=0}^{k} j p_j = 0.231$$

Since S and k are all 6, so $L_q = 0$. The W is

$$W = \frac{1}{\lambda_e} L = 0.13$$

Now, we can compare W with W' stored in service QoS Descriptor, and determine whether the service is failed to satisfy the performance requirements.

In the second case, supposed we composite services in the SOA applications on ESB, we used the above performance model to analyze the following closed queuing network with N =3 nodes and K = 3 jobs. The first node has ml = 2 and the second node has m2 = 3 identical service stations. For the third node we have m3 = 1. The service time at each node is exponentially distributed with respective rates:

$\mu_1 = 0.8\text{sec}^{-1}$, $\mu_2 = 0.6\text{sec}^{-1}$, $\mu_3 = 0.4\text{sec}^{-1}$

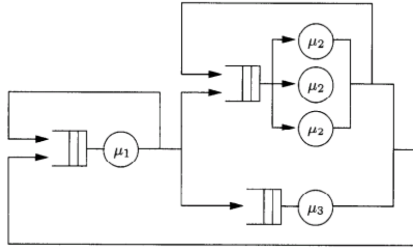

Figure2:A closed queuing networks

Thus The marginal probabilities for the single server node 3 can be computed :

$$\pi_3(0) = \left(\frac{e_3}{\mu_3}\right)^0 \frac{1}{G(3)}\left(G(3) - \frac{e_3}{\mu_3}G(2)\right) = 0.528$$

$$\pi_3(0) = \left(\frac{e_3}{\mu_3}\right)^1 \frac{1}{G(3)}\left(G(2) - \frac{e_3}{\mu_3}G(1)\right) = 0.312$$

$$\pi_3(0) = \left(\frac{e_3}{\mu_3}\right)^2 \frac{1}{G(3)}\left(G(1) - \frac{e_3}{\mu_3}G(0)\right) = 0.132$$

$$\pi_3(0) = \left(\frac{e_3}{\mu_3}\right)^3 \frac{1}{G(3)}\left(G(0) - \frac{e_3}{\mu_3} * 0\right) = 0.028$$

And also, we compute the marginal probabilities for the node 1 and 2

$$\pi_1(0) = \frac{F_1(0)}{G(3)} G_N^1(3) = 0.211,$$

$$\pi_1(0) = \frac{F_1(1)}{G(3)} G_N^1(2) = 0.398$$

$$\pi_1(0) = \frac{F_1(2)}{G(3)} G_N^1(1) = 0.282$$

$$\pi_1(0) = \frac{F_1(3)}{G(3)} G_N^1(0) = 0.109$$

and

$\pi_2(0) = 0.295$, $\pi_2(1) = 0.412$

$\pi_2(2) = 0.242$ $\pi_2(3) = 0.051$

Then throughput can be computed

$$\lambda_1 = e_1 \frac{G(2)}{G(3)} = 0.945, \lambda_2 = e_2 \frac{G(2)}{G(3)} = 0.630$$

$$\lambda_3 = e_3 \frac{G(2)}{G(3)} = 0.189$$

The utilization are given by the performance model above,

$$\rho_1 = \frac{\lambda_1}{m_1\mu_1} = 0.590, \quad \rho_2 = \frac{\lambda_2}{m_2\mu_2} = 0.350$$

$$\rho_3 = \frac{\lambda_3}{\mu_3} = 0.473$$

The mean number of jobs at the multiserver nodes is

$$\overline{K_1} = \pi_1(1) + 2\pi_1(2) + 3\pi_1(3) = 1.290$$

$$\overline{K_2} = \pi_2(1) + 2\pi_2(2) + 3\pi_2(3) = 1.050$$

And for the single server node3 ,

$$\overline{K_3} = \left(\frac{e_3}{\mu_3}\right)\frac{G(2)}{G(3)} + \left(\frac{e_3}{\mu_3}\right)^2 \frac{G(1)}{G(3)} + \left(\frac{e_3}{\mu_3}\right)^3 \frac{G(0)}{G(3)} = 0.660$$

For the mean response time,

$$W_1 = \frac{\overline{K_1}}{\lambda_1} = 1.366,$$

$$W_2 = \frac{\overline{K_2}}{\lambda_2} = 1.667,$$

$$W_3 = \frac{\overline{K_2}}{\lambda_2} = 3.498,$$

Finally, we compute the mean response time of the three nodes in the queuing networks, we separately compare the W_i and W_i'

5. Conclusions

In this paper, we put forward a fault detection mechanism, which is based on the queuing theory, to detect the services that fail to satisfy performance requirements. We improve the mechanism of performance measuring, and we can prove the correctness of this mechanism, we lack of experiment data yet, because it is difficult to establish a simulation application to validate our mechanism. As we have mentioned in [6], the existing frameworks or platforms have no capability of fault tolerance, some of them even have no open APIs provided for us to extend them. With the emergence of open source SOA platforms, we can choose a proper one to extend it to have the capability of fault detection, and establish a practical SOA-based application to validate the correctness of this mechanism in the future.

6 References

[1] W.T. Tsai, Chun Fan, Yinong Chen, R. Paul, and Jen-Yao Chung, "Architecture classification for SOA-based applications", Proc. of the Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC 2006), April, 2006, pp. 8-15

[2] Yan Liu, Ian Gorton, Liming Zhu, "Performance Prediction of Service-Oriented Applications based on an Enterprise Service Bus", 31st Annual International Computer Software and Applications Conference (COMPSAC 2007)
[3]Colin White,"What Do SOA and ESB mean in Business Intelligence", available at http://www.b-eye-network.com/view/3018
[4]G. Spanoudakis, A. Zisman, and A. Kozlenkov, "A service discovery framework for service centric systems", Proc. of 2005 IEEE International Conference on Services Computing (SCC 2005), July 2005, pp. 251 – 259
[5] Davis, D. and Parashar, "M. Latency Performance of SOAP Implementations". In Proceedings of the IEEE Cluster Computing and the GRID 2002 (CCGRID'02), Berlin, Germany, IEEE, 2002
[6] HAO-PENG CHEN, CHENG ZHANG," A Fault Detection Mechanism for Service-Oriented Architecture Based on Queueing Theory", 7th IEEE International Conference on Computer and Information Technology (CIT 2007)
[7] Carolyn McGregor, Josef Schiefer, "A Framework for
Analyzing and Measuring Business Performance with Web Services," cec, p. 405, 2003 IEEE International Conference on E-Commerce Technology (CEC'03), 2003.
[8 Almeida, V. A. and Menascé, D. A. 2002. Capacity Planning: An Essential Tool for Managing Web Services. IT Professional 4, 4 (Jul. 2002), 33-38. DOI= http://dx.doi.org/10.1109/MITP.2002.1046642]
[9] Ted Neward, Effective Enterprise Java, Addison Wesley Professional, Boston, August 26, 2004
[10] AndreasWillig, "A Short Introduction to Queueing Theory", July 21, 1999, available at: www.tkn.tu-berlin.de/curricula/ws0203/ue-kn/qt.pdf
[11] Menascé, D. A and Almeida, V. A., Capacity Planning for Web Services: metrics, models and methods, Prentice Hall, 2001, ISBN 0-13-065903-7.