Proceedings of the Sixth International Conference on Machine Learning and Cybernetics, Hong Kong, 19-22 August 2007

A FAULT DETECTION MECHANISM FOR FAULT-TOLERANT SOA-BASED APPLICATIONS

HAO-PENG CHEN, ZHI-YONG WANG

School of Software, Shanghai Jiao Tong University, Shanghai 200240, China
E-MAIL:chen-hp@sjtu.edu.cn, kindywater@163.com

Abstract:

Fault tolerance is an important capability for SOA-based applications, since it ensures the dynamic composition of services and improves the dependability of SOA-based applications. Fault detection is the first step of fault detection, so this paper focuses on fault detection, and puts forward a fault detection mechanism, which is based on the theories of artificial neural network and probability change point analysis rather than static service description, to detect the services that fail to satisfy performance requirements at runtime. This paper also gives reference model of fault-tolerance control center of Enterprise Services Bus.

Keywords:

Soa; Fault detection; Artificial neural network; Probability change point analysis; Performance requirement

1. Introduction

Service-Oriented Architecture (SOA) has been the most attractive approach of software system development for constructing complex distributed applications. By this approach, the SOA-based applications are built mainly through service discovery and composing but not coding. Application developers identify desired services in the service repository which can realize different parts of the functionality of an application and compose the available services into the target application through an appropriate application composition process.

One of the important characteristics of SOA-based applications that are different from traditional software is dynamic discovery and composition [1]. It means that SOA-based applications can discover and compose services into themselves at runtime, which is necessary for the applications to locate the alternative services and replace existing ones that fail to satisfy the functional or performance requirements during their execution. Dynamic discovery and composition is the prerequisite for fault-tolerant SOA-based applications.

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 [2]. Among these capabilities, fault detection, the capability to identify the failed existing services, is the precondition of other three capabilities, because successful fault detection is the guarantee for them to be employed properly. We can conclude, therefore, fault detection is the most important capability for fault-tolerant SOA-based applications.

In recent a couple of years, many companies and organizations have provided their own SOA frameworks or solutions. For example, IBM provided a service-based foundation for enterprise-level business integration, WebSphere Integration Reference Architecture (WIRA), which provides a comprehensive set of services to support business integration for a large range of complex enterprise applications [3]. OASIS provided a reference model for service oriented architecture, which is based on unifying concepts of SOA and may be used by architects developing specific service oriented architectures or in training and explaining SOA [4]. Microsoft provided its own SOA building block: BizTalk server, which along with WCF could play a pivotal role in organizations that have existing investments which need to be enabled to form a service-oriented architecture [5]. TIBCO, the world leading software company in business integration and process management, provided an Event-Driven SOA framework [6]. Oracle also offered Oracle SOA Suite, which is a comprehensive, hot-pluggable software suite for the building, deployment, and management of a Service-Oriented Architecture [7]. BEA's AquaLogic is also a product family for supporting SOA [8].

These frameworks or solutions, however, have not emphasized the capability of fault tolerance. Actually, in [3], WebSphere Architecture even was considered as a

1-4244-0973-X/07/$25.00 ©2007 IEEE

service-oriented architecture without fault tolerance. This fact shows fault tolerance is an important issue for IT industry, but it has not completely resolved yet.

As we 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 focused 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 mechanism.

This paper puts forward a fault detection mechanism, which is based on the theories of artificial neural network and probability change point analysis, to detect the services that fail to satisfy performance requirements. This paper also gives reference architecture of fault-tolerance control center of ESB (Enterprise Services Bus).

2. Rationale of this mechanism

The purpose of our mechanism is to detect the services which have not satisfied performance requirements any longer. 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 things: 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, once 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].

Since response time is an important index of service performance, we can use it to determine whether the specified service satisfies its performance requirement or not. A fixed value of response time, however, is not suitable for SOA-based application, because the number of service invocation requests at any time is a variable but not a constant, we even cannot determine the maximum of this number when we develop the services. It means if we specify a fixed value or a fixed range of response time to indicate the acceptable performance requirement of service, this value or range hardly reflect the real time context.

Instead of specifying a fixed range of response time, we should dynamically predict the current response time at runtime according to the real time context. We use certain number of the past mean response times for given period of time to accomplish this prediction, for instance, we can dynamically predict the current mean response time for 1 second according to 15 past mean response times for given period of time, which respectively represent mean response time for 1 second of 15 seconds before the time under predicting. It is obviously that it is non-linear relationship between the past mean response time and current mean response time, so we establish an artificial neural network to compute the function between them, because artificial neural network is capable to compute any logic or arithmetical function [10].

After dynamically predicting the current mean response time, we need to compare this predicted value with the current real-time mean response time to check whether the error between them is greater the acceptable error that we configured in advance. It is straightforward that if the error between them is not greater than the acceptable error, it means the service satisfies its performance requirement. But when the error between them is greater than the acceptable error, the situation is much more complex. We cannot simply say the service fail to satisfy its performance requirement, because the error beyond may be caused by two reasons: the number of service invocation requests increased sharply, and the service itself has been corrupted. For the former reason, it does not indicate that the service has failed to satisfy its performance requirement, so we need to discriminate the two kinds of reasons.

We can use probability change point analysis, which is also a mathematical tool for analyzing non-linear system, to determine whether the error beyond is caused by the sharply increased number of service invocation requests. We take a sample space of the numbers of service invocation requests to periodically check whether the sharp increase happened. If the probability is stabilized at a certain value, it means there is no sharp increase happened. In this case, if the error beyond takes place, the reason for it is the service has been corrupted, and we say we detect a fault. If the probability is changed at an unknown time t, we say t is a probability change point, it means the error beyond should be due to the sharp increase of the number of service invocation requests. In this case, we consider that we need find more service instances to balance the load.

Since the services are highly dynamic, and their states are maybe different at any time. In order to improve the accuracy of prediction, the artificial neural network should study continuously. So in our mechanism, the prediction error would always be sent back to the artificial neural network as a feedback, and we would always use the newest learned function.

3. Prediction of current mean response time

As we described, we use artificial neural network to prediction the current mean response time. Actually, artificial neural network apply the function approximation theorem to compute the target function, so the result function is an approximate function but not the real function. Our aim is to reduce the error between the approximate function and real function. In this section, we describe the structure of this artificial neural network, the predicting algorithm we used.

3.1. The structure of artificial neural network this mechanism used

The typical artificial neural network models include back propagation neural network， radical basis neural network, Hopfield neural network, and so on [11]. They all compute the approximate function based on the overall function approximation theorem. However, each of these individual models all has some defects. For example, back propagation neural network is apt to fall into the local minimum error point, although we can use some improved algorithm, such as gradient descent momentum algorithm and conjugated gradient algorithm, to improve the predict result. Another example is radical basis neural network, its principle is mapping the input space into a new space which has enough dimensions, and searching in the new space for the surface which can best fit the training data. If the values of input neurons are all close to each other, the mapping result would be difficult for searching.

Since the each single model all has some defects, we use a composite model which combines the back propagation model and radical basis model. The structure of this model is shown in Figure 1.

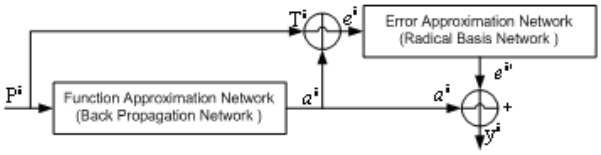

Figure 1. The structure of the artificial neural network which this fault detection mechanism used

In above structure, Function Approximation Network is a back propagation network, which can accomplish the first time approximation. Meanwhile, Error Approximation Network is a radical basis network, which can use the error generated in the first time approximation to approximate again. The outputs of the two networks compose the final output. This composite model can reduce the error generated by back propagation network and radical basis network, but it needs a longer training process.

3.2. The training and predicting algorithm

The algorithm applied into the composite artificial neural network includes two parts: the first part is the training algorithm for training the artificial neural network, the second part is predicting algorithm for predict the current mean response time.

Before we describe the training algorithm, we explain the signs in Figure 1 at first. P^i is an input sample; T^i is a learning sample; a^i represent the output generated by Function Approximation Network according to input P^i; e^i represent the error between learning sample T and the predicting result a^i; $e^i{}'$ is the output of Error Approximation Network; y^i is the final output of this composite artificial neural network.

The training algorithm has five steps, as the followings:

Choose sample space S, which includes enough samples for training.

For each sample P^i in S, use it as the input of Function Approximation Network and train this network with some back propagation algorithm, such as gradient descent momentum algorithm, to get an output a^i.

For each output a^i, calculate e^i according to formula (1).

$$e^0 = T - a^0 \tag{1}$$

For each e^i, use it as the input of Error Approximation Network and train this network with radical basis algorithm to get an output $e^i{}'$.

After training with the sample space S, we can use the predicting algorithm to dynamically predict the current mean response time. As we mentioned, when we use this network for prediction, the inputs of network are past mean response times for given period of time. We need to preprocess the inputs according to formula (2) to limit all the input values into the range from 0 to 1. In formula (2), $a_{\max}$ is the maximum of all input values, $a_{\min}$ is the minimum of all input values, a_i is an input value, and a_i' is the processed value of a_i.

$$a_i' = \frac{a_{\max} - a_i}{a_{\max} - a_{\min}} \tag{2}$$

After preprocessing the input values, we use them as input of composite network to calculate the final output y^i according to formula (3).

$$y^{i} = a^{i} + e^{i} \quad (3)$$

After getting y^{i}, we can compare it with current real-time mean response time to check whether the error between them is greater the acceptable error that we configured in advance. If the error is beyond the acceptable error, we need to use the change point analysis to determine whether the error beyond is caused by corruption of service.

4. Probability change point analysis of service invocation requests

We use probability change point analysis to prognosticate the sharp increase of the number of service invocation requests. Similarly to the artificial neural network we designed, the learning samples are past numbers of service invocation requests for given period of time. The sample space for change point analysis is usually much greater than the one for predicting the current mean response time. If we write M samples in a given unit period of time, and we take all the samples in N units of time, we will get a sample space R, which has $N \times M$ independent samples $r_{11}, r_{12}, \ldots, r_{N \times M}$, and $r_{n \times m}$ represents the number of service invocation requests at m time of n unit.

The algorithm for find change points is described as the followings:

Observe the sample space for M times, and observe a single column of the sample matrix per time. In the s^{th} single observation, we compare r_{1s}, r_{2s}, …, r_{Ns} with threshold t, and use o_s to represent the probability of the observed event occurrence. We use x_s to represent whether there is a sharp increase of the number of service invocation requests at the time of M in a single observation. If o_s is greater then C, then x_s is set to 1, otherwise, x_s is set to 0. Then, after M times observation, we will get another independent sample space x_1, x_2, …, x_M, and x_t represents the times of event occurred (0 or 1) in the single t^{th} time observation.

Calculate p_1 and p_2 according to the formula (4) and (5).

$$p(x_t = 1) = 1 - p(x_t = 0) = p_1, t = 1, 2, ..., m-1 \quad (4)$$

$$p(x_t = 1) = 1 - p(x_t = 0) = p_2, t = m, ..., N \quad (5)$$

Calculate u_k and v_k according to formula (6) and (7) to respectively represent the accumulated times the observed event occurred and did not occur in former k observations.

$$u_k = x_1 + x_2 + ... + x_k \quad (6)$$

$$u_k + v_k = k \quad (7)$$

It is obvious that u_k / k is the frequency of sharp increase occurrence in former k observations, and u_M / M is the frequency of observed event occurrence in all the M observations.

Calculate the statistical value T_k according to formula (8):

$$T_k = k(u_k / k - u_m / M), k = 1, 2, ..., M \quad (8)$$

Calculate the mathematical expectation of T_k according to formula (9):

$$E(T_k) \begin{cases} kM^{-1}(M - m + 1)(p_1 - p_2) & 1 \le k \le m-1 \\ M^{-1}(M - k)(m-1)(p_1 - p_2) & m \le k \le M \end{cases} \quad (9)$$

We can see that if $p_1 \neq p_2$, $|E(T_k)|$ increases with the $k = 1, 2, ..., m$, and reaches the maximum at $k = m-1$, and then decreases with $k = m, ..., M$, and at $k = m, ...M$, it reaches the minimum 0. if $p_1 = p_2$, $|E(T_k)|$ is always equal to 0.

Calculate $|T_1|$, $|T_2|$, … $|T_M|$, and calculate $\tilde{m}$ according to formula (10):

$$\left|T_{\tilde{m}}\right| = \max(|T_1|, |T_2|, ..., |T_M|) \quad (10)$$

Then $m^* = \tilde{m} + 1$ is the change point.

If there are several $\tilde{m}$ all make $\left|T_{\tilde{m}}\right|$ to the maximum, then, it is shown that there are multiple change points, and each of them represents a sharp increase in the number of service invocation requests.

If the error between predicted mean response time and real-time mean response time is greater acceptable error at the time which is a change point, it means we need find more service instances to balance the load. If the error beyond took place at the time which is not a change point, it means the service has been corrupted, and we successfully detected this fault.

5. Reference architecture of fault-tolerance control center of ESB

In [3], the authors abstracted the architecture of fault-tolerance control center of ESB. We extend this architecture by adding three components into it to support our fault detection mechanism. The reference architecture is shown in Figure 2.

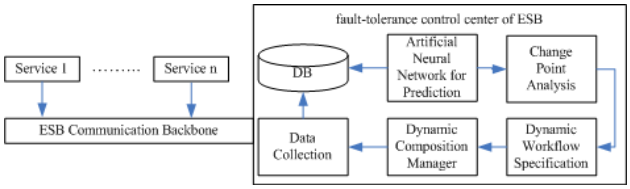

Figure 2. Reference architecture of fault-tolerance control center of ESB

In this reference architecture, the Data Collection Service, Dynamic Composition Manager, and Dynamic Workflow Specification are put forward by the authors of [3]. Data Collection Service will keep on monitoring the behaviors of the participating services and collect data at runtime [3]. We add a Database to store the collected data, because the sample spaces for mean response time prediction and change point analysis are too much to store in memory. The Artificial Neural Network for Prediction queries the Database to get appropriate samples to predict the current mean response time. The predicting result is sent to Change Point Analysis Module to check whether there exists a change point of the number of service invocation requests. The checking result is send to Dynamic Workflow Specification to determine whether the workflow specification should be modified or not [3]. If the workflow specification is modified, the Dynamic Composition Manager will re-composite the workflow at runtime [3].

This architecture is a coarse-grain one, when applying to specific SOA-based application, it needs to be refined and customized.

6. Conclusion

In this paper, we put forward a fault detection mechanism, which is based on the theories of artificial neural network and probability change point analysis, to detect the services that fail to satisfy performance requirements. Although artificial neural network and probability change point analysis are also mature theories and we can prove the correctness of this mechanism, we lack of experiment data yet. We will establish a practical SOA-based application to validate the correctness of this mechanism in future.

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] 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

[3] S. Simmons, "Introducing the WebSphere Integration Reference Architecture: A Service-based Foundation for Enterprise-Level Business Integration", IBM WebSphere Developer Technical Journal, Aug. 17, 2005, available at: http://www-128.ibm.com/developerworks/websphere/techjournal/0508_simmons/0508_simmons.html.

[4] C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F Brown, Rebekah Metz, "Oasis Reference Model for Service Oriented Architecture", Aug. 2, 2006, available at: http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf

[5] Daniel Rubio, "BizTalk Server: Microsoft's SOA building block", Jan. 24, 2006, available at: http://searchwebservices.techtarget.com/tip/0,289483,sid26_gci1161311,00.html

[6] TIBCO General Purpose White Papers, "Event-Driven SOA: A Better Way to SOA", available at: http://www.tibco.com/resources/solutions/soa/event_driven_soa_wp.pdf

[7] Oracle SOA Suite Datasheet, "Oracle SOA Suite", October 2006, available at: http://www.oracle.com/technologies/soa/oracle-soa-suite-datasheet.pdf

[8] Bruce Silver, "The 2006 BPMS Report: BEA AquaLogic BPM Suite v5.5", October 2006, available at: http:// www.bpminstitute.org/research/single-research/article/bea-aqualogic-bpm-suite-v5-5.html

[9] Ted Neward, *Effective Enterprise Java*, Addison Wesley Professional, Boston, August 26, 2004

[10] Martin T. Hagan, Howard B. Demuth, Mark H. Beale. *Neural Network Design*, PWS Publishing Company, 1996.

[11] Tom M. Mitchell, *Machine learning*. McGraw-Hill Companies, Inc., 1997.

[12] Xiang Jing-tian, Shi Jiu-en, *Statistics Methods for Data Process of Nonlinear System*, Science Press, Beijing, China, 2000.