A PROACTIVE ADAPTATION MODEL BASE ON RESOURCE PREDICTION

Hua Wang and Jian Yu

School of Information and Electronic Engineering, Zhejiang University of Science and Technology, China

Email: {wanghua96@126.com}

Abstract

The adaptation of the software architecture caused by the change of the software survival environment needs to be realized through the reconfiguration of the connector of the software architecture. To tackle the limitation of passive connectors, a proactive adaptation model based on resource prediction is proposed by employing Kalman filter, which realizes the modification of component connection and the dynamic configuration of the architecture, and enhances the adaptability of the target system. Three proactive adaptation strategies are proposed, and the basis of the adaptation policy is quantified by the influence factors value. After the method is applied to an actual online health assessment assistant system, the adaptability and service quality of the target system are improved.

Key Words: Connector, Proactive adaptation, Resource prediction, Kalman filter.

1. Introduction

In the software architecture (for the convenience of description, the software architecture is abbreviated as architecture or SA), the interactive relationship between components is explicitly expressed as a connector, and the connector is regarded as an equally important position as a component, which is a very important contribution in SA research [1, 2]. This is mainly because the connector can express the complex semantic relationship between components, and it also helps to encapsulate the variability, which has an important impact on the adaptability of the architecture. The system structure adaptation caused by the change of the software survival environment needs to be realized through the adaptation of the connector.

Treating the connector as an independent architecture entity helps to separate component connection requirements (coordination mechanism) and functional requirements (computing functions), isolate the mutual influence of component changes and software survival environment, and facilitate the modification of component

connections and the architecture. The dynamic configuration is an effective way to improve the adaptability of the system structure.

There have been past attempts at using the current survival environment resource information to generate SA configuration information through connectors [3-5]. The use of resources [6-7] is passive, leading to partial utility value is optimal while ignoring the diminishing utility caused by time changes. In particular, software architects have almost no assistance in reasoning about questions such as: How should we stage the adaptation to achieve business goals in the presence of the variability and uncertainty of the software survival environment? Aiming at the shortcomings of insufficient applicability of the two prediction methods proposed in [8], a lightweight resource prediction method based on Kalman filter is proposed. Kalman filter [9] is described by a series of recursive mathematical formulas. Its main advantage is that it consumes little calculation and storage. Also, it provides an efficient and computable method to estimate the state of the process and minimize the estimated mean square error.

A small number of states ensures scalability in large-scale computing environments. Furthermore, unlike other prediction techniques based on machine learning [10], the Kalman filter does not require training. To this end, a predictive connectors model based on resource prediction is proposed by employing Kalman filter. Experiments with an actual operating online health assessment system show that this method provides a predictive adaptive configuration mechanism for applications, and improves the predictability and service quality of applications.

2. Resource prediction framework

2.1 Environmental awareness and resource capture

In order to build a bridge between the software survival environment and SA, and at the same time as the input end of the resource prediction model, *Sensor* and *Measure* components are used to build environment awareness and resource capture infrastructure as shown in Fig. 1.

Fig. 1 Environment sensor and resource capturing

The bottom layer is a set of sensors, deployed in the software surviving environment, and the *Sensor Reporting Bus* publishes perception information of different dimensions obtained from the environment. The different dimensions here mainly reflect the different entity types of survival environment, for instance, the computing resource types (CPU, memory, and network conditions), and the probability feature (initial probability and observation probability, etc.). The environmental information represented by these different entity types needs to be obtained through different *Sensors*. The *Sensor Reporting Bus* collects the low-level measurement information, and then encapsulates it into higher-level model information after being interpreted by *Measure*. These model information is distributed to *Measure Consumers* through the *Measure Bus*. On the one hand, the model information can be directly used as the input of the *Resource Predictor* to construct or reconstruct the connector based on it, and on the other hand, it can also be used as the input for generating proactive adaptive policies.

Sensor is implemented based on Java RMI (Remote Method Invocation) [11]. The implementation classes and interfaces of this layer are shown in Fig. 2.

Fig. 2 Class implement of *Sensors*

The *Sensor* is designed to facilitate the execution of monitoring programs, such as the network command *netstat*. This mechanism enables on-time monitoring, which help to reduce the amount of data collected. In order to support the publish/subscribe communication mechanism, JMS (Java Message Service) [12] is employed to provide communication services within the network range between components, such as *SensorReportBus* and *SensorSubscriber*.

2.2 Kalman Filter Mathematical Model for Resource Prediction

Before diving into how proactive adaptation is done let's introduce the corresponding foundational theories of Kalman filter. The resource prediction model is based on the literature [13]. Let $W_t \sim N(0, Q_w)$, $G_t \sim N(0, Q_g)$ and $H_t \sim N(0, Q_h)$, where N represents the noise covariance matrix of process excitation that satisfies the *Normal Distribution*, which is the white noise of the process signal.

$$\text{If } o_t = \begin{bmatrix} G_t \\ H_t \end{bmatrix}, x_t = \begin{bmatrix} U_t \\ A_t \end{bmatrix}$$

there is:

$$\hat{x}_t = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \hat{x}_t + o_t$$

Here x_t is a state vector.

Through the initial value $\hat{x}_1 = P(x_1 | y_0)$ and the covariance matrix:

$$C_1 = E((x_1 - \bar{x}_1)(x_1 - \bar{x}_1)^T) = E \begin{pmatrix} (U_1 - \bar{U}_1)(U_1 - \bar{U}_1) & (U_1 - \bar{U}_1)(A_1 - \bar{A}_1) \\ (U_1 - \bar{U}_1)(A_1 - \bar{A}_1) & (A_1 - \bar{A}_1)(A_1 - \bar{A}_1) \end{pmatrix}$$

we get the estimated value

$$\hat{x}_{t+1} = \begin{bmatrix} \bar{U}_{t+1} \\ \bar{A}_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \bar{U}_t \\ \bar{A}_t \end{bmatrix} + \Delta t^{-1} (y_t - U_t) \begin{bmatrix} \theta_{U_t} \\ \theta_{A_t} \end{bmatrix}$$

Solve this equation and get:

$$\bar{U}_{t+1} = \bar{U}_t + \bar{A}_t + \Delta t^{-1} (y_t - U_t) \theta_{U_t}$$

$$\bar{A}_{t+1} = \bar{A}_t + \Delta t^{-1} (y_t - A_t) \theta_{A_t}$$

Here,

$$\Delta t = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + [Q_w] = [Y_{11}] + [Q_w]$$

$$Y_t = E((x_t - \bar{x}_t)(x_t - \bar{x}_t)^T)$$

User requests for services may present periodic characteristics. For example, a news website experiences a peak visit period from 8 to 9 in the morning from Monday to Friday. Kalman filter can also realize periodic information prediction in time series. For this reason, the periodic part S_t is introduced in the observation equation:

$$y_t = U_t + A_t + S_t + w_t$$

In particular, when there is a random deviation in strict periodicity, we can add a G_t :

$$S_{t+1} = -S_t - \dots - S_{t-d+2} + G_t \quad t = 1, 2, \dots$$

By introducing the state vector

$$x_t = [S_t \ S_{t-1} \ \dots \ S_{t-d+2}]^T$$

the sequence s is given by the observation equation

$$S_t = [1 \ 0 \ 0 \ 0 \ \dots \ 0]^T x_t$$

Here, $x_t = cx_t + g_t$, $x_t = [U_t \ A_t \ S_t \ S_{t-1} \ \dots \ S_{t-d+2}]^T$, and the matrix c :

$$c = \begin{bmatrix} 1 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & -1 & -1 & \dots & -1 & -1 \\ 0 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 & 0 \end{bmatrix}$$

and

$$\mathbf{g}_t = [g_t \ h_t \ Z_t \ 0 \cdots 0]^T$$

This leads to the observation equation

$$y_t = [1 \ 0 \ 1 \ 0 \cdots 0]^T x_t + w_t$$

In some scenarios, m -step forward prediction needs to be made to obtain observations:

$$\hat{x}_t = p_t(x_{t+1}) = E_t p_{t-1}(x_t) + \theta_t \Delta t^{-1} (y_t - p_{t-1}(x_t))$$

and get:

$$\hat{x}_{t+m} = p_t(x_{t+m}) = E_{t+m-1} p_t(x_{t+m-1}) = (E_{t+m-1} E_{t+m-2} \cdots E_{t+1}) p_t(x_{t+1}) \quad t=2,3,\cdots$$

$$\hat{y}_{t+m} = p_t(y_{t+m}) = F_{t+m} p_t(x_{t+m}) \quad t=1,2,\cdots$$

Here, in the above three equations, let $|x_t| = d_x, |y_t| = d_y$, $p_t(x)$ represents the best prediction of x , $\{F_t\}$ is a $d_x \times d_y$ matrix, and $\{E_t\}$ is a $d_y \times d_y$ matrix.

The resource predictor is responsible for periodically obtaining environmental information from the *Sensor* and *Measure*, and calculating the availability of future resources in the time series through the Kalman filter.

3. Proactive adaptation strategy

3.1 Impact factor value

In order to express the user's satisfaction degree with the operating state of the system in the current survival environment, a function is designed, which we term impact factor value (*IFV*) function. *IFV* is quantified by synchronization penalty, component health level, QoS (Quality of Service) preference and service provider preference as follows.

(1) Synchronization penalty function

The reconfiguration of connectors will affect the customer's perception of the service. For example, online video services may dynamically adjust the image coding algorithm according to the predicted bandwidth situation. The customer needs to be notified to maintain the continuity of the service during the adaptation, which will cause the customer to be distracted leading to a decline in

service quality. This can be measured by synchronization cost.

$$IFV_{sp}(c) \propto P_{c \rightarrow c'}^{dgr}$$

Here, $dgr \in [0..1]$ represents the user's satisfaction degree when the component c is replaced with the component c' , and the larger the value $IFV_{sp}(c)$, the greater the synchronization cost.

(2) Component health level function

The function is expressed by the inherent properties of components, such as execution time, failure probability, and abnormal probability, which are all important indicators to measure component quality. The function is expressed as follows:

$$IFV_{ht}(c) \propto \prod T_c^{wt} \prod F_c^{wf} \prod E_c^{we}$$

where, wt , wf , and we respectively represent the weights of execution time, failure probability, and abnormal probability in the component health level. T, F, E represent the health level values of the above three attributes, and their value ranges are all $[0..1]$. Both F and E are probability density functions, which are set by the component provider.

(3) QoS preference function

The function contains the various QoS requirements of the user for the request, as well as their weight, i.e. preferences of users. For example, the QoS of online video services includes average bit rate, average response time, video quality and service price, as shown in Table 1.

QoS can be measured by an impact factor function, which is related to each QoS, expressed by the formula as follows.

$$IFV_{qp}(c) \propto \prod_{d \in \dim(s)} U_d^{weight}$$

here, for each QoS attribute d of the component c , it represents the user's preference for c on d . For a certain d , U_d^{weight} is equal to zero means that even though other QoS

attributes have a high degree of preference, c is meaningless to users.

Table 1. Video QoSs and preferences of users

Quality attribute	Description	Component Service	Weight
qBR	Average bit rate	VideoService.bitrate()	50%
qRT	Average response time	LaptopClient.avgRespTime()	30%
qVD	Video quality	VideoService.resolution()	18%
qC	service price	VideoService.cost()	2%

(4) Service provider preference function

Customers have different degrees of preference for the services of service providers, which determines users' choice of providers. For instance, users may strongly require Safari, but will also accept Firefox or Opera, and will be happy with other browsers such as Edge, but may have a lower rating, whose IFV can be expressed as follows.

$$IFV_{pp}(s) \square L_{s \rightarrow cs}^w$$

Here, w is the user's preference for selecting service s from the component set cs , and its value range is $[0..1]$.

The total IFV can be expressed by the calculation of the previous functions as follows.

$$IFV_{total}(s) \square \prod_{c \rightarrow c'} \frac{T_c^{wt} \square F_c^{wf} \square E_c^{we} \square U_d^{weight} \square L_{s \rightarrow cs}^w}{P_{c \rightarrow c'}^{dgr}}$$

3.2 Proactive adaptation strategy

The proactive adaptation strategy will help the connector have proactive reconfiguration capabilities. For example, if the network bandwidth is predicted to fall below a certain threshold in the next three minutes, the system will limit requests from low-priority users and those remaining users will get a satisfactory QoS while low-priority users will be served within an acceptable

delay. The following three types of proactive adaptations are common in practice.

(1) Choose a better configuration

The predicted value of the QoS attributes of the survival environment can be calculated to obtain the total $IFV_{total}(s)$ of component service s . The connector selects the strategy with the largest total IFV as the basis for proactive reconstruction.

(2) Reduce synchronization cost

When the survival environment changes, the connector will be reconfigured. If the change of the survival environment immediately triggers the reconfiguration of the connector, such continuous adaptation will sharply increase the IFV of the synchronization cost. Through the proactive resource prediction model, a specific forward prediction time can be set for resource prediction. The QoS attribute changes of the continuous survival environments can be considered globally, and proactive adaptation can be given, thereby reducing the accumulated synchronization cost.

(3) Avoid unnecessary adaptations

Assuming that the execution time of the adaptive strategy is Δt , the time when the measurement value of a certain QoS attribute in the survival environment is predicted to exceed the preset threshold $\Delta t'$. If $\Delta t > \Delta t'$, it means that the adaptation is unnecessary, otherwise it will increase the value of the synchronization cost. Δt can be obtained by the attribute value of execution time of the adaptive strategy, and the resource prediction model can calculate $\Delta t'$.

4. Experiment

To demonstrate the usefulness of our approach, we show how it can be applied to an actual adaptation process. During the operation of online Health assessment assistant System (*HaaS*, in www.joyhar.com), due to market promotion, it experienced a peak period of visits. In order to retain customers, *HaaS* needs to improve the proactive

and adaptive capabilities of the system. The adaptive process scenario is described as follows:

Step1. The *Survival Environment Sensor* reports that there is enough bandwidth available;

Step2. The customer requests to provide a graphic report, and the graphic report service is added through *HealthReport.replace (GRAPHIC)*;

Step3. As the graphical report requires more bandwidth, the bandwidth usage rate is predicted to increase;

Step4. The component *HealthReport* suspends the graphic report service, because the excessive bandwidth load prevented *HaaS* from accepting more user requests, which led to the decline of customer QoS;

Step5. The phenomenon of increasing bandwidth has disappeared, so the bandwidth has returned to a usable state to provide services for more users.

HaaS provides image services such as *PACS* and X-ray photography that provide very valuable materials when predicting certain diseases. When super VIP customers request this type of service, more bandwidth is required. According to the three different adaptive adaptation

strategies in Section 3.2, the experimental results are analyzed as follows.

4.1 Choose a better configuration

After the execution of **Step2**, the user's visits increased, resulting in an increase in response time and a decrease in available bandwidth. Between *addConnectionPool* (increase database connection pool) and *HealthReport.replace(TEXT)* (the graphic report replaced with the text one), *HaaS* chose the latter in order to keep client requests in response. After a short time, the bandwidth has increased to an available level and *HaaS* once again chooses *HealthReport.replace (GRAPHIC)* to maintain customer satisfaction. If customer visits has been maintained at a high level, *HaaS* will also reduce the response time of the service through *addConnectionPool*. It can be seen that the adaptation of *replace(TEXT)* in *HaaS* is temporary because it results in a decrease in the total *IFV*. After employing resource prediction, *HaaS* chose the *addConnectionPool* strategy to increase the total *IFV* as shown in Fig.3 below. When the number of user requests continues to increase, the adaptive method of resource prediction improves performance more and more.

Fig. 3 Performance comparison of choosing optimal configuration

4.2 Reduce synchronization costs

In **Step 3**, the system response time increases, and *HaaS* selects *addConnectionPool* to reduce the system response time. After that, user requests has exceeded the system

throughput provided by *HaaS*. At this time, *addConnectionPool* is called again to increase the system throughput. Two consecutive adaptation make $IFV_{sp}(c) < 0.8$ (the preset value) and reduce the QoS. Through *m*-step forward prediction (*m* is 5 time windows), *HaaS* selects

one-time call `addconnectionpool` (number = 2), which successfully reduces the synchronization cost. The experimental results show that the *IFV* of synchronization cost is reduced by 12%.

4.3 Avoid unnecessary adaptation

In the **Step2**, it is predicted that customer visits will increase and the CPU load will also increase. *HaaS* will increase the system throughput by employing *addServer* (increasing the application server Tomcat) and load

balancing (Tomcat+Apache). It is estimated that the execution time of *addServer* is 3 seconds, and it is predicted that the CPU load will be reduced after 2.3 seconds, so that more customer requests can be received. In this scenario, *HaaS* chose to abandon the *addServer* service. Fig. 4 shows the performance comparison of these two different adaptations.

Fig. 4 Performance comparison of avoid unnecessary adaptation

In Fig.4, the left shows the data of the CPU utilization in two different scenarios. Although the resource prediction method does not adopt an adaptation policy when the CPU load increases, it shows better performance when the load drops subsequently. In terms of response time (shown on the right), there are similarities: the resource prediction method has a longer response time when the CPU load is high, but it is better than the method of selecting the *addServer* service after the CPU load drops.

We also tested periodic adaptation scenarios, as shown in Fig.5 as follows. After implementing *HaaS* in a sanitarium in Hangzhou that is a city in China, it was discovered that the second month of each quarter was the peak period of medical examination services, and *HaaS* had withstood the test of a surge in visits. By implementing periodic adaptation strategies using *addServer* and *addConnectionPool* in advance, *HaaS* has improved QoS.

Fig. 5 Periodical resource prediction

5. Conclusions

The variability and uncertainty of the software survival environment puts forward new requirements for the adaptability of the architecture. The adaptation of the traditional connector is passive and the conditions of the survival environment are changed to trigger the self-adaptability by policies being set in advance. In order to overcome this partial and passive adaptation method, a proactive adaptation model based on resource prediction is proposed. The policy is quantified through the *IFV* as the basis for selecting the policy. An online health assessment assistant system is adapted to make it proactive self-adaptation. The experimental results show that the performance of the target system is improved when resource prediction is used, and the proactive model has application feasibility.

References

- [1] Mary Shaw, Garlan D. Software Architecture: Perspectives on an Emerging Discipline[M]. New York: Prentice Hall, 1996.
- [2] Wermelinger, Michel, Fiadeiro, et al. Connectors for Mobile Programs[J]. *IEEE Transactions on Software Engineering*, 1998, 24(5):331-341.
- [3] Selic B . Specifying Dynamic Software Systems Architectures[J]. *Software and Systems Modeling*, 2021(20):595-605.
- [4] Kautz O, Rumpe B, Wortmann A. Automated semantics-preserving parallel decomposition of finite component and connector architectures[J]. *Automated Software Engineering*, 2020(27):119-151.
- [5] Rouland Q, Hamid B, Jaskolka J. Formal specification and verification of reusable communication models for distributed systems architecture[J]. *Future Generation Computer Systems*, 2020(108):178-197.
- [6] Cameron D, Konstantinov A, Ould-Saada F, et al. The Advanced Resource Connector for Distributed LHC Computing[J]. *Proceedings of Science*, 2010(46):1-5.
- [7] Chang K C, Chu K C, Wang H C, et al. Agent-based middleware framework using distributed CPS for improving resource utilization in smart city[J]. *Future Generation Computer Systems*, 2020(108):445-453.
- [8] Cheng S W, Poladian V V, Garlan D, et al. Improving Architecture-Based Self-Adaptation through Resource Prediction[J]. *Software Engineering for Self-Adaptive Systems*. Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2009(5525):71-88.
- [9] Kalman R. E. A New Approach to Linear Filtering and Prediction Problems[J]. *Transaction of the ASME-Journal of Basic Engineering*, 1960: 35-45
- [10] Alonso J, Torres J, Berral J L, et al. Adaptive on-line software aging prediction based on machine learning[C]. *IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*. IEEE, 2010:507-516.
- [11] Trail: RMI (The Java Tutorials), Retrieved from: <https://docs.oracle.com/javase/tutorial/rmi/index.html>.
- [12] Wetherbee J, Nardone M, Rathod C, et al. EJB Message-Driven Beans[M]. 2018.
- [13] Musolesi M., Hailes S., Mascolo C. Prediction of Context Information using Kalman Filter Theory[M]. UCL Research Note RN/04/22, University College London Department of Computer Science, UCL, UK 2004.