The Embodiment of Autonomic Computing in the Middleware for Distributed System with Bayesian Networks

Bo-Yoon Choi, Kyung-Joong Kim, and Sung-Bae Cho

Dept. of Computer Science, Yonsei University
134 Shinchon-dong, Sudaemoon-Ku, Seoul 120-749, South Korea
bychoi@sclab.yonsei.ac.kr, kjkim@cs.yonsei.ac.kr,
sbcho@cs.yonsei.ac.kr

Abstract. In this paper, we propose an intelligent middleware framework based on the concept of the autonomic computing. Because decisions for optimizing a system are highly dependent on the status of the system in the near future, predictions of the status can boost up the performance of system. Bayesian networks that are frequently used in the domain of user modeling are adopted as a diagnosis engine. Since there are many different kinds of behavioral patterns, it is impossible to model all of them into a single model. A Bayesian network is learned for each typical behavioral pattern. Experimental results show that the proposed method offers a promising way of intelligent middlewares.

1 Introduction

Component-based applications are independent of component's location and flexible in the change of its structure. When a user makes use of a process or takes a service from a server using internet at distributed environment, a middleware organizes a component set of applications for a user on real time to satisfy user's request. Meanwhile, it can optimize components structure of an application for improvement of the performance according to a user's environment. Such a process is called as autonomic computing [1], which is a skill for the development of an adaptive application reflecting immediately to changing system and updating its function by component insertion or deletion. It continuously optimizes itself to provide a service requested, given the state of a user's environment.

This paper proposes an intelligent middleware using Bayesian networks for autonomic computing, which is more flexible. Because Bayesian network is based on the probability theory, it can naturally deal with such uncertainties without any significant effort. Also, it can provide two directional inferences (forward and backward). Intelligent middleware consists of data collection, performance analysis and component configuration or reconfiguration. When a user operates a distributed application at the circumstance of distributed systems, the middleware collects information about the system and the component set. After processing collected data, they are inputted to the diagnosis engine and used as evidences of Bayesian networks in the diagnosis engine for the performance analysis and the system diagnosis. The inferred result from the learned Bayesian network can be used to make the decision of component swapping.

D.-S. Huang, K. Li, and G.W. Irwin (Eds.): ICIC 2006, LNCS 4113, pp. 1047 – 1052, 2006.

2 Diagnosis Engine Using Bayesian Networks

System resources and the user's behavior are considered as relevant information sources to infer future resource levels. System resources include the speed of the network, free memory, CPU usage, and the type of device. User behavior can be modeled from the history of application execution, whether the system is on or off, usage time, and the switching of tasks. Eric Horvitz *et al*. attempted to model the user's behavior patterns through activities on the desktop with the help of a camera and acoustic sensors [5]. Because our platform assumes the reconfiguration of middleware, the information sources for inference are at a relatively lower level than those of application reconfiguration.

```
1: /* n : Number of variables */
2: /* A[i,j] : Score gain when an edge from the jth node to the ith node is connected */
3: /* Score(B) : Score of Bayesian network structure B */
4: /* Score(B, j→i) : Return a score when B has an edge (j→i) */
5: /* Find_Max(A) : Return an edge (j→i) that has the maximum A[i,j] */
6: /* Min : minus infinity */
7: /* Ancestors(x_i) : A set of nodes that have a path from the node to x_i */
8: /* Decendants(x_i) : A set of nodes that have a path from x_i to the node */
9: /* Stop(); If all (i,j), A[i,j] ≤ 0 or A[i,j]=Min then True */
10: FOR i=1 to n { Pa(x_i) = φ ; }
11: FOR i=1 to n FOR j=1 to n
12:   IF i ≠ j THEN A[i,j]=Score(B, j→i) – Score(B);
13: WHILE(TRUE) {
14:   (i, j)=Find_Max(A);
15:   IF A[i,j]>0 THEN Pa(x_i) = Pa(x_i) ∪ {x_j} ;
16:   A[i,j]=Min;
17:   FOR a=1 to n FOR b=1 to n
18:     IF x_a ∈ Ancestors(x_j) ∪ {x_j} && x_b ∈ Decendants(x_i) ∪ {x_i}
19:     THEN A[a,b]=Min;
20:   FOR k=1 to n {IF A [i,k]>Min THEN A[i,k]=Score(B, k→i) – Score(B); }
21:   IF Stop()==True THEN break;
22: }
```

Fig. 1. Overview of the greedy search for Bayesian network

We use $< B, \theta_B >$ to denote a Bayesian network with a structure B and probability parameters θ_B. $P < B, \theta_B >$ denotes the joint probability distribution of all the variables of this network. A possible structure of a Bayesian network can be drawn as a directed acyclic graph. $B = (V, E)$, where the set of nodes $V = \{x_1, x_2, ..., x_n\}$ represents the domain variables and E, a set of arcs, represents the direct dependency among the variables. For each variable $x_i \in V$, the conditional probability distribution is $P(x_i \mid Pa(x_i))$, where $Pa(x_i)$ represents the parent set of the variable x_i.

$$P < B, \theta_B >= P(x_1, x_2, ..., x_n) = \prod_{i=1}^{n} P(x_i \mid Pa(x_i))$$

Bayesian probabilistic inference is one of the most popular models for inference and representation of environments with insufficient information [6][7]. Usually the structure is designed by an expert. To develop appropriate Bayesian networks for this specific problem given a little prior knowledge, learning is essential. Using a scoring function that returns a numerical value for the appropriateness of the given data in the Bayesian networks, search algorithms such as greedy and genetic algorithms attempt to maximize the score.

From an empty network with no edge, the greedy algorithm repeats the procedure of adding an edge that maximizes a score gain on the current structure and fixes the new structure as a current one until the structure converges. Though the algorithm can get stuck in the local minimum, it can perform well if the number of variables is relatively small. If the number of variables is large, a global search algorithm such as a genetic algorithm is a more appropriate choice. In this domain, we assume that relevant variables are selected by the expert and the learning procedure is conducted by the greedy algorithm. Figure 1 shows the pseudo code of the greedy search.

The intelligent middleware adopts the Bayesian network as a diagnosis engine for the performance analysis of distributed applications. A structure of Bayesian network is important to improve the efficiency of the diagnosis engine. It is difficult to develop highly efficient diagnosis engine using only one structure of Bayesian network which is suitable for various users. Therefore, the intelligent middleware creates Bayesian networks of the diagnosis engine dynamically according to the pattern of users' computer usage. For example, when a user executes a distributed application, the intelligent middleware collects data in advance about system environment and executed process list. Such data are used for learning Bayesian networks. The Bayesian network is maintained if the diagnosis engine produces good performance. Otherwise, when the engine does not work or the pattern of a computer usage is changed, Bayesian networks will be reconstructed again.

3 Experimental Results

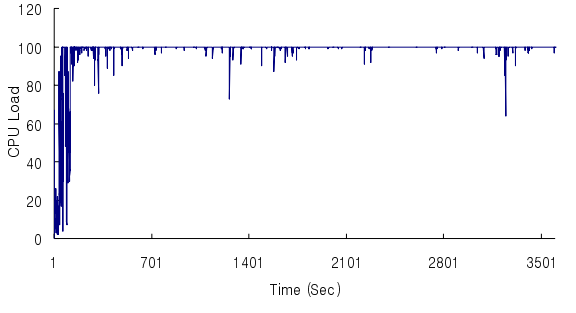
Data were collected from a student and a programmer for two hours. After installing a monitoring program, they used their computers in the same way as usual. The monitoring program collected the usage patterns of specific programs and system resource information. The student played continuously an online game with the player of another computer and the programmer worked for programming and accessed steadily a database for inquiring data. In addition, they performed several kinds of programs and also their systems executed many basic processes for system management. Table 1 shows the details of the process executed by a user and a system.

A half of the data which present the proper change of a system state is used for training and the remaining is used for test. The purpose of the Bayesian network was to infer the future CPU load given the current CPU load and the user's current job list. From the preliminary study, the performance of a distributed application showed unexpected quality degradation when the CPU load was kept 100% for more than 5 seconds, but when the CPU load was kept 100% for less than 5 seconds, it did not affect the performance of the application. Sometimes the CPU load became the state of 100% for less than 5 seconds. In this case, most of the cases was the time when a

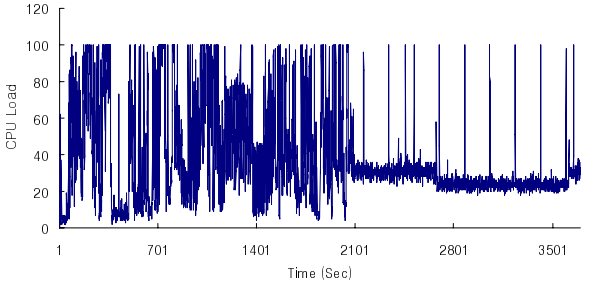
user executed new process. It did not make a system problem or degrade the performance of the system. Otherwise, when the CPU load was kept 100% for more than 5 seconds, we could predict a system problem led to a system overload. In this paper, the Bayesian network infers the probability of the event that CPU is kept 100% for more than 5 seconds.

Table 1. List of process executed by a user and a system

	List of processes
User 1	SystemIdleProcess, System, smss, csrss, winlogon, services, lsass, ati2evxx0, svchost0, svchost1, svchost2, svchost3, svchost4, spoolsv ,ati2evxx1, explorer, V3monnt, V3monsvc, myLinker, DrVirus, ctfmon, msmsgs, trayapp, alg, LCDPlyer, wscntfy, CDSLicenseMng, conime, taskmgr, CpuUsage, ProcMon, TalesWeaver, InphaseNXD, IEXPLORE, skcbgm, npkagt, regsvr32, npkcsvc, npdownv
User 2	SystemIdleProcess, System, smss, csrss, winlogon, services, lsass, svchost, spoolsv, explorer, MsgPlus, monsysnt, v3p3at, daemon, ctfmon, ProtHelp, ClientSM, conime, ahnsdsv, ahnsd, AszTray, monsvcnt, v3impro, msnmsgr, putty, TOAD, notepad, editplus, CpuUsage, ProcMon, EMEDITOR, IEXPLORE, v3syson, rundll32, npcopyv, npdownv, NPMON, logon, sucer, supdate, autoup

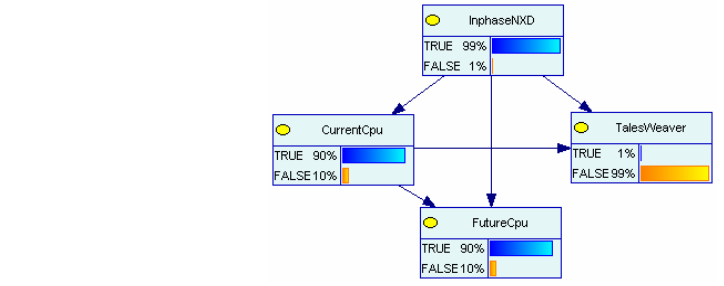


(a) user 1

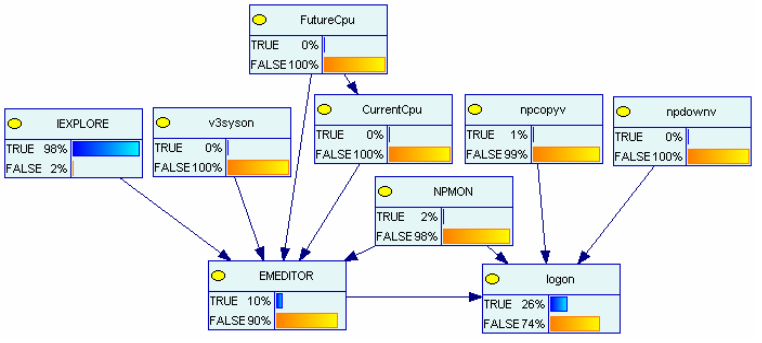


(b) user 2

Fig. 2. CPU load change in the training data (one hour)



(a) user 1



(b) user 2

Fig. 3. The structure of Bayesian networks

Table 2. Prediction accuracy of the Bayesian networks on the test data

	CPU load is 100%		CPU load is not 100%		Accuracy	
	User 1	User 2	User 1	User 2	User 1	User 2
CPU load is 100%	1275	17	83	3	93.88%	85.00%
CPU load is not 100%	81	5	2135	3572	96.34%	99.86%
Total	1356	22	2218	3575	95.41%	99.77%

Figure 2 shows the CPU load changes in the data of user 1 and user 2. Figure 3 shows the structure of the learned Bayesian networks of user 1 and user 2. The learned networks contain relatively small nodes because some nodes have no edge. The structures depend on the computer usage pattern of each user, which led to the different structures of Bayesian networks for user 1 and user 2. The Bayesian network of user 1 has four nodes. For user 2, the Bayesian network has nine nodes. Table 2 shows the prediction accuracy of the learned models. The Bayesian network for user 1 shows 95.41% prediction accuracy and the model for user 2 shows 99.77% prediction accuracy.

4 Conclusions and Future Works

In this paper, we have proposed a middleware framework for distributed applications. Because distributed environments are highly dynamic and contain uncertainties due to noise and network delays, it is crucial to develop robust diagnosis engines. Bayesian networks can model probabilistic dependencies among random variables provide a flexible two-directional inference mechanism (forward and backward inferences). Experimental results show that the proposed method is promising enough to estimate future states given evidence for selecting proper services. To deal with various situations, it is necessary to use more than one Bayesian network. In this context, the management of several Bayesian networks including construction, aggregation, and replacement has to be developed as an independent system module. It is also necessary to adapt previous learned models given the recently collected data.

Acknowledgement

This research was supported by Brain Science and Engineering Research Program sponsored by Korean Ministry of Commerce, Industry and Energy.

References

1. Jephart, J.O., Chess, D.M.: The Vision of Autonomic Computing. IEEE Computer, vol. 36, no. 1, January (2003) 41-50
2. Kon, F. et al.: Design, Implementation, and Performance of an Automatic Configuration Service for Distributed Component Systems. Software-Practice and Experience, vol. 34, (2004) 1-39
3. Diaconescu, A., Murphy, J.: A Framework for Automatic Performance Monitoring, Analysis and Optimization of Component based Software Systems. Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems, (2004) 29-34
4. Heckerman, D., Geiger, D., Chickering, D.: Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. Machine Learning. vol. 20, (1995) 197-243
5. Horvitz, E., Kadie, C. M., Paek, T., Hovel, D.: Models of Attention in Computing and Communications: From Principles to Applications, Communications of the ACM, vol. 46, no. 3, (2003) 52-59
6. Charniak, E.: Bayesian Networks without Tears. AI Magazine, vol. 12, no. 4, (1991) 50-63
7. Stephenson, T.: An Introduction to Bayesian Network Theory and Usage. IDIAP-RR00-03, (2000)