SA BASED SOFTWARE DEPLOYMENT RELIABILITY ESTIMATION CONSIDERING COMPONENT DEPENDENCE[1]

Su Xihong Liu Hongwei Wu Zhibo Yang Xiaozong Zuo Decheng
(*School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China*)

Abstract Reliability is one of the most critical properties of software system. System deployment architecture is the allocation of system software components on host nodes. Software Architecture (SA) based software deployment models help to analyze reliability of different deployments. Though many approaches for architecture-based reliability estimation exist, little work has incorporated the influence of system deployment and hardware resources into reliability estimation. There are many factors influencing system deployment. By translating the multi-dimension factors into degree matrix of component dependence, we provide the definition of component dependence and propose a method of calculating system reliability of deployments. Additionally, the parameters that influence the optimal deployment may change during system execution. The existing software deployment architecture may be ill-suited for the given environment, and the system needs to be redeployed to improve reliability. An approximate algorithm, A*_D, to increase system reliability is presented. When the number of components and host nodes is relative large, experimental results show that this algorithm can obtain better deployment than stochastic and greedy algorithms.

Key words Software Architecture (SA); Software deployment; System reliability; Component; Redeploy

CLC index TP31

DOI 10.1007/s11767-011-0561-5

I. Introduction

In the past few decades, it is very evident that the size and complexity of software systems are growing, especially in embedded, mobile, and distributed software systems. Possible reasons include: increase in the speed and capacity of hardware, the emergence of wireless ad hoc networks, proliferation of sensors, and handheld computing devices, and so on. A number of researches have shown that a promising approach to resolve the challenges is to employ the principles of software architectures[1]. Software Architecture (SA) is a collection of models that capture software system principle design decisions in the form of components (location of system computation and data management), connectors (location of component interaction), and configurations (specific arrangements of components and connectors intended to solve specific problems)[2].

With the separation of software deployment from software development process, it has been an independent stage. Software deployment is referred to as a collection of activities, which is to make software available for use until uninstalling it from devices[3]. These activities include delivery, installation, configuration, activation, updating, reconfiguration, and uninstallation of the software[4]. The research of software deployment began in recent years and the SA based software deployment has an even shorter history[5]. The allocation of the system software components on host nodes is called as system deployment architecture. A few researchers have begun to study availability of system deployment architecture in resource constrained environment[6–8]. However, little work has been done on reliability estimation of system deployment.

Several recent approaches have begun to quantify software reliability at the level of architectural

[1] Manuscript received date: September 20, 2010; revised date: January 18, 2011.
Supported by the High Technology Research and Development Program of China (No. 2008AA01A201), National High Technology Research, Development Plan of China (No. 2006AA01A103), the High Technology Research and Development Program of China (No. 2009AA01A404).
Communication author: Su Xihong, born in 1981, female, Ph.D.. Harbin Institute of Technology, Harbin 150001, China.
Email: suxihong07@126.com.

models, or at least in terms of high-level system structure[9–18]. Architecture-based software reliability estimation is calculated on the basis of the information of component reliabilities and application architecture[19]. These models have some assumptions, for example, the behaviors of components satisfy Markov property[17], the reliabilities of the individual components are known[12–14] and so on. However, these reliability estimation approaches are typically limited to certain classes or exclusively concentrates on software reliability, neglecting the influence of hardware resources and system deployment. Additionally, these models mainly predict the reliability of a software system early in its development, for example, architectural design.

In this paper, system reliability of software deployment stage is investigated. The parameters influencing reliability may change during system execution and the system may need to be redeployed to improve reliability.

Marija[8] uses Avala algorithm to redeploy the system for improving availability. Avala algorithm adopts greedy algorithm to select the "best" host node or "best" component every time. We propose A*_D algorithm to redeploy system. The experimental results show that it is effective for improving system reliability.

This paper is organized as follows. Section II explains the related concepts. Section III states the problem definition. Section IV gives a brief description of three algorithms: exact algorithm, stochastic algorithm and greedy algorithm, and presents a A*_D algorithm. Experimental results and conclusions are given in Sections V and VI.

II. Related Concepts

1. Component dependence

A graph $G=(C,E)$ is undirected graph, C represents a set of software components. $C=\{C_1, C_2,\cdots,C_n\}$, $|C|=n$. E represents a set of interactions among software components.

Definition 1 Direct component dependence

Given an undirected graph $G=(C,E)$, if $\exists C_i$, $C_j\in C$, $(C_i,C_j)\in E$, then component C_i epends on C_j. The relationship of component C_i and C_j is direct component dependence.

Definition 2 Indirect component dependence

Given an undirected graph $G=(C,E)$, if $\exists C_i$, $C_j\in C$, $(C_i,C_j)\notin E$, $\exists m_1, m_2, \cdots, m_k$, $k\le|C|-2$, $(C_i,C_{m_1})\in E$, $(C_{m_1},C_{m_2})\in E,\ldots,(C_{m_k},C_j)\in E$, then the relationship of component C_i and C_j is indirect component dependence.

Direct component dependence and indirect component dependence are all regarded as component dependence. Given an undirected graph $G=(C,E)$, $\forall C_i, C_j, C_k\in C$, in order to describe whether component C_j is more important than component C_k for component C_i, we introduce the concept of component dependence degree.

Definition 3 Component dependence degree

Given an undirected graph $G=(C,E)$, $\forall C_i$, $C_j\in C$, $\mathrm{cd}_{i,j}$ describes the degree of component C_i depends on component C_j. $\mathrm{cd}_{i,j}$ is a real number in [0,1]. $\forall C_i$, we assume that $\mathrm{cd}_{i,j}=0$.

Finding good deployment architecture is determined by many factors. We translate multi-dimensional factor influence on deployment into degree matrix of component dependence among components. For different systems, how to make use of these factors to generate the degree matrix of component dependence should be different. For example, component dependence degree may be a function of these factors. The function may be linear function, exponential function, logarithm function, and so on. At last, we normalize component dependence degree to a real number in [0,1].

Matrix CD describes the component dependence degree between two arbitrary components for one system run-time. $C=(C_1, C_2, \cdots, C_n)$ is a finite set of software components.

$$\mathbf{CD}=\begin{array}{c} \\ C_1 \\ \vdots \\ C_n \end{array}\begin{array}{c} \begin{array}{ccc} C_1 & \cdots & C_n \end{array} \\ \begin{bmatrix} \mathrm{cd}_{1,1} & \cdots & \mathrm{cd}_{1,n} \\ \vdots & \cdots & \vdots \\ \mathrm{cd}_{n,1} & \cdots & \mathrm{cd}_{n,n} \end{bmatrix} \end{array}$$

In general, the component dependence degree of direct component dependence is larger than the one of indirect. Additionally, it is easy to make sense $\mathrm{cd}_{i,j}\neq\mathrm{cd}_{j,i}$, $i\neq j$.

2. Sytem reliability

Reliability is regarded as the probability for components or systems to function successfully against the expected environment for a given time.

Reliability information of subsystems or components is the basis of system reliability estimation[20]. For reliability estimation of software deployment stage, consider that the main sources of system reliability degradation are component failure and host node failure. If a host node fails, the components deployed on the host node would fail. Therefore, the system reliability can be calculated as follows.

$$R_{\text{system}}=1-p_{\text{system}}=1-\bigcup_{k=1}^{M}\text{ph}_k-\bigcup_{h=1}^{M}\text{pcsh}_h \tag{1}$$

$$\text{pcsh}_i=\left(1-\prod_{j=1}^{m}\left(1-\text{pc}_j\right)\left(1-\text{ph}_i\right)\right)\left(1-\text{cdh}_i\right) \tag{2}$$

$$\bigcup_{i=1}^{M}\text{pcsh}_i=\sum_{i=1}^{M}\text{pcsh}_i-\sum_{1\le i<j\le M}\left(\text{pcsh}_i\times\text{pcsh}_j\right)+\cdots+(-1)^{M-1}\prod_{i=1}^{M}\text{pcsh}_i \tag{3}$$

where

R_{system} : system reliability;

P_{system} : failure probability of system;

M : number of host nodes;

m : number of components deployed on H_i;

pcsh_i : failure probability of components deployed on host node H_i;

pc_j : failure probability of component C_j;

ph_i : failure probability of host node H_i;

cdh_i : sum of dependence degree among components deployed on host node H_i.

III. Problem Statement

In the face of host node failure and component failure, how to deploy software components on host nodes greatly influences system reliability. There exist many parameters influencing system deployment architecture, such as frequency of interaction among components, software architectural styles, possible restrictions on component location and so on. During system execution, the values of these parameters may change, and degree matrix of component dependence also may change. For this reason, the currently executing deployment may be ill-suited for the given environment. That is, the system needs to be redeployed for reliability improvement. However, finding a system deployment architecture that will maximize its reliability is an exponential complex problem, in general, the complexity is $O(m^n)$ (n is the number of software components and m is the number of host nodes). Therefore, we need to devise approximate algorithm to deploy software components on host nodes to improve system reliability.

In order to explain the approximate algorithms clearly, we present data structure description of deployment architecture and introduce the meanings of variables. Data structure of deployment architecture is described in Fig. 1. Flag value may be 1, 0 or –1.

```
DeployArchitecture {
    int flag;
    double systemReliability;
    int delComWhichHost;
    int[ ] comDeployedHost [n];
}
```

Fig. 1 Data structure of deployment architecture

$$\text{flag}=\begin{cases}1, \text{it will be redeployed}\\ 0, \text{it has been redeployed}\\ -1, \text{it will not be redeployed}\end{cases}$$

The value of systemReliability describes system reliability of the deployment architecture. delComWhichHost describes the host node from which the selected component is deleted. comDeployedHost describes the allocation of software components on host nodes. For example, if the value of comDeployedHost[2] is 3, that is, component C_2 has been deployed on host node H_3.

IV. Deployment Algorithms

We briefly describe three existed algorithms for system redeployment. These algorithms include exact algorithm, stochastic algorithm, and greedy algorithm. Then, we develop A*_D algorithm according to A* algorithm for system deployment.

1. Exact algorithm

This algorithm calculates system reliability of all possible deployment architectures, and selects the one that has maximum reliability. The exact algorithm finds one optimal deployment at least. In general, the complexity of this algorithm is $O(m^n)$. Even if the number of host nodes and components is very small, this algorithm still would be com-

putationally too expensive.

2. Stochastic algorithm

This algorithm randomly selects a deployment architecture that has flag 1 each time. It will halt when flag values of newly generated deployment architectures all equal –1. This process needs to be repeated a desired times, and the best obtained deployment is selected. The complexity of this algorithm is polynomial, since it randomly selects one deployment to calculate system reliability from m deployments architectures each time, and that takes $O(n^2)$ time.

3. Greedy algorithm

This algorithm always makes the choice that looks best at the moment[21]. After system initialization, this algorithm selects the "best" deployment architecture to redeploy each time. The "best" one needs to satisfy two conditions: (1) its flag value is 1; (2) its system reliability is max. The algorithm will halt when the flag values of newly generated deployment architectures all equal –1. The remained deployment architecture with flag 0 is the optimal one.

The algorithm needs to calculate system reliability for each deployment and chooses the "best" one from m deployment architectures. This process takes $O(mn^2)$ time. This process is repeated n times at most, and the optimal deployment architecture is selected. Therefore, the complexity of this algorithm is $O(mn^3)$.

4. A*_D algorithm

A* algorithm combines the advantages of both the depth-first search and the breadth-first search. On the basis of an evaluated function, it selects the best one from all available nodes to extend. We leverage A* algorithm for system redeployment, A*_D, as shown in Fig. 2.

In order to explain this algorithm, we consider an example system. This system consists of 3 host nodes $\{H_1, H_2, H_3\}$ and 6 components $\{C_1, C_2, C_3, C_4, C_5, C_6\}$, as shown in Fig. 4(a). flag is used to describe the current status of deployment architectures. All the six components may be deployed on host node H_1, H_2 or H_3. On the basis of degree matrix of component dependence, the component that has minimum dependence degree is selected each time. This process is repeated $m-1$ times (m is the number of host nodes). The $m-1$ components will be deployed on other $m-1$ host nodes. In the example system, we suppose that $C_6, C_5, C_4, C_3, C_2, C_1$ is the order of component dependence degree from small to large. Therefore, C_5 and C_6 are selected to initialize the system.

```
A*_D()
{
Step 1  Initialize system deployment, InitializeDeploy
 (componentDependenceMatrix);
Step 2  Select the deployment architecture that
has maximum reliability, DA_M;
Step 3  Choose next component that will be deployed,
ccs=ChooseComponent (delComWhichHost);
Step 4  Deploy the selected component on other
host nodes, Redeploy (DAs,DA_M,ccs);
Step 5  Prune the existed deployment architectures,
Prune (DAs);
Step 6  Sort the remained deployment architectures,
Sort (DAs);
Step 7  If the value of flag  of  some deployment
architecture is 1, go to Step 2;
Step 8  The  remained  deployment architecture
with flag 0 is the optimal one.
}
```

Fig. 2 A*_D algorithm of system deployment

```
Prune(List<DeployArchitecture> DAs)
{
  optimal=new DeployArchitecture();
  for (int i=0; i<DAs.size();i++){
     if (DAs.get(i).flag==-1)
           DAs.remove(i);
     if (DAs.get(i).flag==0)
           if (DAs.get(i).systemReliability>optimal.system-
Reliability)
              optimal=DAs.get(i);
           else DAs.remove(i);
           }
}
```

Fig. 3 Prune method

Fig. 4(b) is the result of system initialization. In order to illustrate, Fig. 4(a) is called the root node,

and its level is 0. Then generates six new deployment architecture as the children of the root node, and this level is 1.We use $DA_{i,j}$ to represent the j deployment architecture of the level i. $RDA_{i,j}$ is system reliability of $DA_{i,j}$. For example, the reliability order of six deployment architectures is: $RDA_{1,1}>RDA_{1,2}>RDA_{1,3}>RDA_{1,4}>RDA_{1,5}>RDA_{1,6}$.

We select $DA_{1,1}$ to redeploy, as shown in Fig. 4(c). The selected component is C_4. Since $DA_{1,1}$ has been redeployed, its flag value becomes 0. $RDA_{2,2}<RDA_{1,1}$, the flag value of $DA_{2,2}$ is –1. $RDA_{2,1}>RDA_{1,1}$, the flag value of $DA_{2,1}$ is 1. On the basis of prune method, prune $DA_{2,2}$ and $DA_{1,1}$. The reliability order of remained deployment architectures becomes $RDA_{2,1}>RDA_{1,2}>RDA_{1,3}>RDA_{1,4}>RDA_{1,5}>RDA_{1,6}$.

In Fig. 4(d), select $DA_{2,1}$ to redeploy, and the selected component is C_3. We obtain $DA_{3,1}$ and $DA_{3,2}$. The flag value of $DA_{2,1}$ becomes 0. On the basis of prune method, prune $DA_{3,1}$ and $DA_{3,2}$. The reliability order becomes $RDA_{1,2}>RDA_{1,3}>RDA_{1,4}>RDA_{1,5}>RDA_{1,6}$.

In Fig. 4(e), $DA_{1,2}$ that has flag 1 is selected to system redeploy. The selected component is still C_4. The newly generated deployment architectures are $DA_{2,3}$ and $DA_{2,4}$. The flag value of $DA_{2,3}$ is 1. The flag value of $DA_{2,4}$ is –1 and $DA_{2,4}$ will be pruned. The reliability order becomes $RDA_{2,3}>RDA_{1,3}>RDA_{1,4}>RDA_{1,5}>RDA_{1,6}$.

In Fig. 4(f), select $DA_{2,3}$ to redeploy. The flag value of $DA_{2,3}$ becomes 0. On the basis of prune method, $RDA_{2,3}>RDA_{2,1}$, prune $DA_{2,1}$. $DA_{3,3}$ and $DA_{3,4}$ are also pruned. The reliability order becomes $RDA_{1,3}>RDA_{1,4}>RDA_{1,5}>RDA_{1,6}$.

Similarly, we extend other deployment architectures that have flag 1, as shown in Fig. 4(g). We extend $DA_{1,3}$, $DA_{1,4}$, $DA_{1,5}$, $DA_{1,6}$. The flag values of these four deployment architectures all become 0. Additionally, the flag values of newly generated deployment architectures all are –1. On the basis of prune method, $DA_{1,3}$, $DA_{1,4}$, $DA_{1,5}$, $DA_{1,6}$, $DA_{2,5}$, $DA_{2,6}$, $DA_{2,7}$, $DA_{2,8}$, $DA_{2,9}$, $DA_{2,10}$, $DA_{2,11}$, $DA_{2,12}$ are pruned. Therefore, $DA_{2,3}$ that has flag 0 is the optimal deployment architecture.

This algorithm makes use of breadth-first algorithm search to select $2m$ (m is the number of host nodes) deployment architectures that have higher reliability. For one selected deployment architecture, this algorithm calculates system reliability of newly generated $m-1$ deployment architectures, and uses depth-first search to select better one. This process takes $O(mn^2)$ (n is the number of components) time. It is repeated n times at most, and takes $O(mn^3)$ time. Therefore, the complexity of this algorithm is $O(m^2n^3)$.

V. Experimental Results

In order to evaluate the four algorithms, we run these algorithms on four generated systems that have different numbers of host nodes and components. The first case contains 3 host nodes and 8 components. The second case contains 4 host nodes and 11 components. The third case contains 10 host nodes and 100 components. The fourth case contains 15 host nodes and 200 components.

For each case, the system inputs are randomly generated. These inputs include degree matrix **CD** of component dependence among components, failure probabilities of n components and failure probabilities of m host nodes. Failure probabilities of host nodes are real numbers in [0.99,1]. Failure probabilities of components are real numbers in [0.9,1].The simulated experiments perform on Intel Core2 Quad Q9400 Processor@2.66GHz, 3GB DDR2, and run JVM 1.5 on Windows XP.

As illustrated in Fig. 5 and Fig. 6, the different algorithms exhibit different performance both in terms of achieved reliability and execution time. For the first and the second case, the exact algorithm finds the optimal deployment architecture and needs to be computationally too expensive (when the number of host nodes and components is very small). Therefore, it is not applicable to systems that have many host nodes or components. However, for four different cases, A*_D algorithm is slower than the stochastic algorithm (by the factor of m^2n) and greedy algorithm (by the factor of m), but it can achieve higher system reliability.

VI. Conclusions

SA based software deployment model provides a high-level architectural description of software and hardware models during system deployment stage. It can be used to analyze quality attributes of different deployments. We provide a new method of

calculating architectural-level reliability incorporating the influence of system deployment and hardware resources. When reliability decreases, system needs to be redeployed for reliability improvement. Because finding the deployment architecture that maximize reliability is an exponentially complexity problem, we need to devise approximate algorithms for system deployment. On the basis of A* algorithm, we proposes A*_D algorithm for system deployment. In order to explain the algorithm, we give the execution process of an example system that consists of 3 host nodes and 6 components. From the case with 15 host nodes and 200 components, we can conclude that A*_D algorithm can deal with large scale deployment and has good scalability. Experimental results show that A*_D algorithm can give better deployment than stochastic algorithm and greedy algorithm.

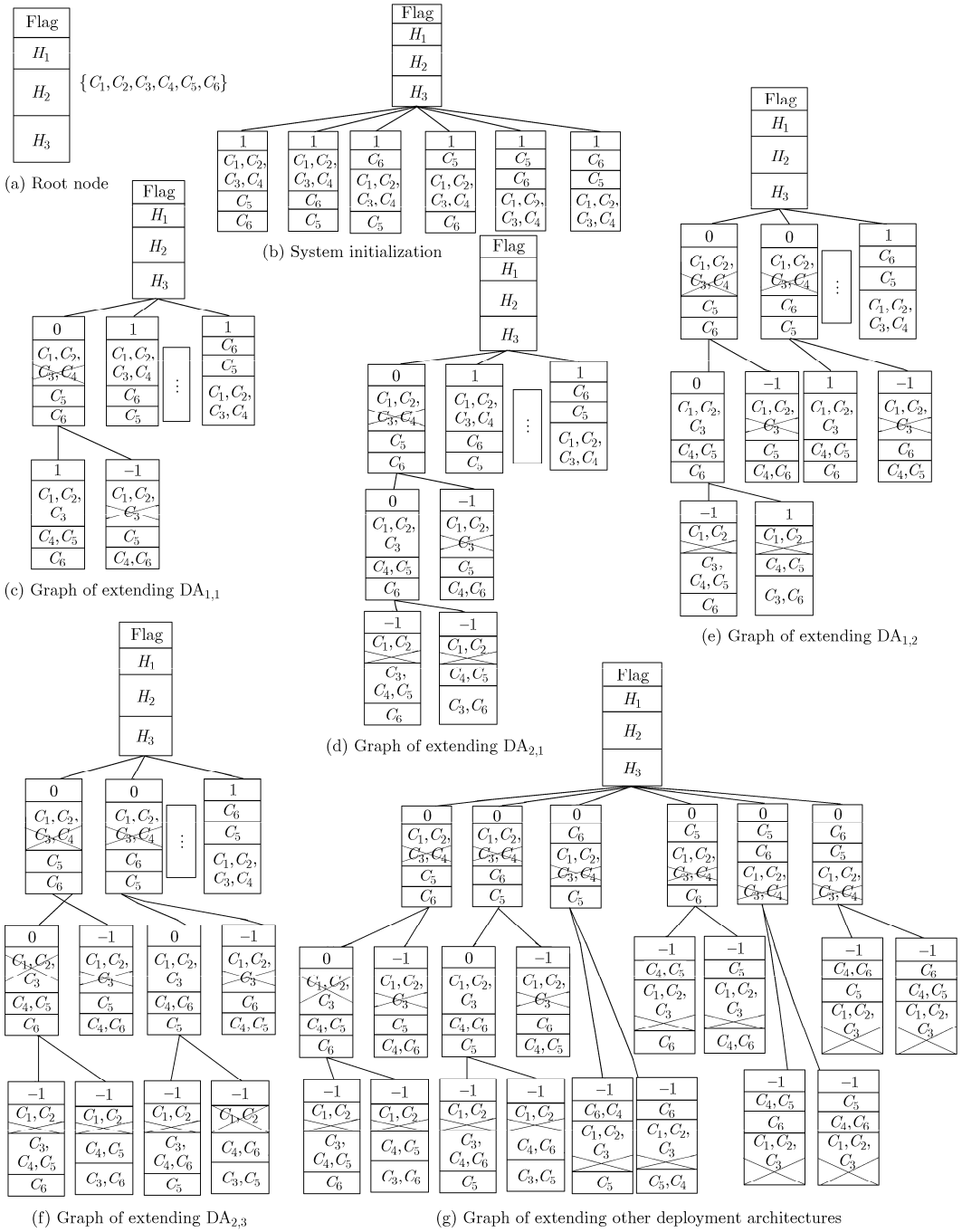

Fig. 4 Execution process of the example system by A*_D algorithm

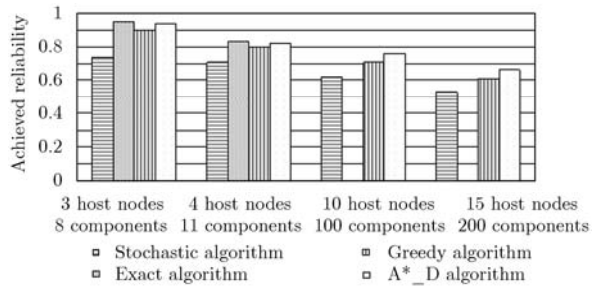

Fig. 5 Achieved system reliability of four algorithms

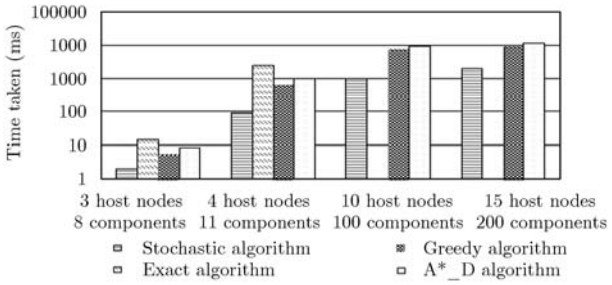

Fig. 6 Taken time of four algorithms

References

[1] C. Seo, *et al.*. Exploring the role of software architecture in dynamic and fault tolerant pervasive systems. First International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments (SEPCASE'2007), Los Angeles, CA, May 20–26, 2007, 9–15.

[2] N. Medvidovic and S. Malek. Software deployment architecture and quality-of-service in pervasive environments. In Proceedings of the 2007 International Workshop on Engineering of Software Services for Pervasive Environments (ESSPE'2007), Dubrovnik, Croatia, Sep. 4, 2007, 47–51.

[3] A. heydarnoori, F. Mavaddat and F. Arbab. Towards an automated deployment planner for composition of web services as software components. *Electronic Notes in Theoretical Computer Science*, **160**(2006)8, 239–253.

[4] D. Ayed, C. Taconet, G. Bernard, and Y. Berbers. An adaptation methodology for the deployment of mobile component-based applications. 2006 ACS/ IEEE International Conference on Pervasive Services, Lyon, June 26–29, 2006, 193–202.

[5] Mei Hong and Shen Jun-Rong. Progress of research on software architecture. *Journal of Software*, **17**(2006)6, 1257–1275 (in Chinese).
梅宏, 申峻嵘. 软件体系结构研究进展. 软件学报, **17**(2006,)6, 1257–1275.

[6] M. Mikic-Rakic and N. Medvidovic. Architectural-level support for software component deployment in resource constrained environments. In Proceeding of the IFIP/ACM Working Conference on Component Deployment, Berlin, Germany, June 20–21, 2002, 31–50.

[7] S. Malek, M. Mikic-Rakic, and N. Medvidovic. A style-aware architectural middleware for resource-constrained, distributed system. *IEEE Transactions on Software Engineering*, **31**(2005)3, 256–272.

[8] M. Mikic-Rakic, S. Malek, and N. Medvidovic. Improving availability in large, distributed component-based systems via redeployment. *Lecture Notes in Computer Science*, 3798(2005), 83–89.

[9] V. Cortellesa and V. Grassi. A modeling approach to analyze the impact of error propagation on reliability of component-based systems. *Lecture Notes in Computer Science*, **4608**(2007), 140–156.

[10] S. Gokhale and K. Trivedi. Reliability prediction and sensitivity analysis based on software architecture. 13th International Symposium on Software Reliability Engineering, Annapolis, MD, USA, Nov. 12–15, 2002, 64–75.

[11] K. Goseve-Popstojanova, *et al.*. Architectural level risk analysis using UML. *IEEE Transactions on Software Engineering*, 29(2003)10, 946–960.

[12] K. Goseva-Popstojanova, *et al.* Architecture-based approaches to software reliability prediction. *Journal of Computer & Mathematics with Applications*, **46** (2003)7, 1023–1036.

[13] R. Reusser, *et al.* Reliability prediction for component-based software architectures. *Journal of Systems and Software*, **66**(2003)3, 241–252.

[14] W. Wang, Y. Wu, and M. Chen. An architecture-based software reliability model. 1999 Pacific Rim International Symposium on Dependable Computing, Hong Kong, China, Dec. 16–17, 1999, 143–150.

[15] S. M. Yacoub, *et al.*. Scenario-based reliability analysis of component-based software. In 10th International Symposium on Software Reliability Engineering, Boca Raton, FL, USA, Nov 1–4, 1999, 22–31.

[16] S. Gokhale, M. R. Lyu, and K. S. Trivedi. Reliability simulation of component-based software systems. The 9th International Symposium on Software Reliability Engineering, Paderborn Germany, Nov. 4–7, 1998, 192–201.

[17] K. Seigrist. Reliability of systems with markov transfer of control. *IEEE Transaction Software Engineering*, **14**(1998)7, 1049–1053.

[18] S. Yacoub, B. Cukic, and H. Ammar. A scenario-

based analysis for component-based software. *IEEE Transactions on Reliability*, **53**(2004)4, 465–480.

[19] S. S. Gokhale. Architecture-based software reliability analysis: overview and limitations. *IEEE Transactions on Dependable and Secure Computing*, **4**(2007)1, 32–40.

[20] T. Jin. Hierarchical variance decomposition of system reliability estimates with duplicated components. *IEEE Transactions on Reliability*, **57**(2008)4, 564–573.

[21] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms. Beijing, Higher Education Press, 2001, 370–378.