考虑组件复制的SA软件部署可靠性研究

苏喜红，刘宏伟，吴智博，杨孝宗，左德承

（哈尔滨工业大学 计算机科学与技术学院，150001 哈尔滨，suxihong07@gmail.com）

摘　要：复制软件组件能提高服务的可靠性和系统可靠性，然而，复制额外的软件组件需要消耗系统可用系统资源. 为了充分利用系统可用资源，得到更高的可靠性优化值，设计了启发式的贪婪复制算法，该算法根据单位带宽的可靠性优化值增量*OB*和单位内存的可靠性优化值增量*OM*，利用贪婪思想选择出将被复制的两个软件组件集合，其中具有更高可靠性优化值的集合是该算法确定的将被复制的软件组件集合. 实验结果表明：当给定有限的系统可用资源时，与贪婪复制算法相比，该算法能得到更高的可靠性优化值和更高的服务可靠性.

关键词：组件复制；可靠性；软件部署；软件体系结构

中图分类号：TP311.5　　**文献标志码**：A　　**文章编号**：0367－6234(2012)03－0054－05

Reliability of SA based on software deployment considering component replication

SU Xi-hong, LIU Hong-wei, WU Zhi-bo, YANG Xiao-zong, ZUO De-cheng

(School of Computer Science and Technology, Harbin Institute of Technology, 150001 Harbin, China, xuxihong07@gmail.com)

Abstract: To make use of available system resources and obtain higher reliability optimum, a heuristic greedy replication algorithm is designed. This algorithm is based on the increased reliability optimum of each unit's bandwidth *OB* and memory *OM*. It uses the greedy idea to select two sets of software components that may be replicated. The software components of the set with higher reliability optimum are selected to replicate by the algorithm. Given the limited available system resource, comparing with greedy replication algorithm, this algorithm can obtain higher reliability optimum and higher service reliabilities. Therefore, when the available system resource is constrained, the heuristic greedy replication algorithm is a good method of selecting software components to replicate.

Key words: component replication; reliability; software deployment; software architecture

随着分布式软件与网络的快速发展，软件部署成为整个软件生命过程中的一个独立阶段[1]. 基于SA(Software Architecture)的软件部署是指软件组件在主机节点上的一种分配. 已有的研究[2－5]表明部署对系统的可靠性、可用性、延迟、容错等有重要影响，尤其是在具有生命期长、高度异构和不可预测的普适计算环境更为明显[6].

收稿日期：2011－03－11.

基金项目：国家高技术研究发展计划重大基金资助项目(2008AA01A201).

作者简介：苏喜红(1981—)，女，博士研究生；
刘宏伟(1971—)，男，教授，博士生导师；
吴智博(1954—)，男，教授，博士生导师；
杨孝宗(1939—)，男，教授，博士生导师；
左德承(1972—)，男，教授，博士生导师.

Kutzner等[7－9]对大规模网络节点的状态分析可知，在普适、移动和分布式的环境下，经常会有节点或者组件面临失效. 然而，如果软件组件有副本，系统仍可正常工作. Popescu[10]利用贪婪算法选择将被复制的软件组件. 然而，贪婪算法经常找到局部优化的将被复制的软件组件. 本文在分析可靠性优化函数*OF*和贪婪算法的基础上，设计了启发式的贪婪复制算法.

1　可靠性优化函数

复制软件组件能提高系统可靠性，但是需要消耗额外的系统资源. 在选择将被复制的软件组件时，决策者需要考虑服务的重要性、软件组件消

耗的资源、软件组件的可靠性、计算机硬件和网络等等. 在选择好将被复制的软件组件时,决策者也决定了将这些软件组件部署到主机节点上. Malek[11]将软件组件复制的选择转换为优化问题,优化问题的目标函数为

$$OF = \sum_{C_i=1}^{|C|}\sum_{s_j=1}^{|S|}\eta \times R(s_j,C_i) \times Cri(s_j)$$

式中:$|C|$ 为系统中软件组件的数目;$|S|$ 为系统中服务的数目;$R(s_j,C_i)$ 为软件组件 C_i 和它的组件副本提供的服务 s_j 的可靠性;$Cri(s_j)$ 为服务 s_j 的重要性.

$$\eta = \begin{cases}1, & \text{如果 } cs_{i,j} = 1;\\ 0, & \text{如果 } cs_{i,j} = 0.\end{cases}$$

每个软件组件副本为原始软件组件的一个附加实例,根据软件组件副本的数目来计算提供服务的可靠性, 得到

$$R(s_j,C_i) = 1 - \prod_{p}^{|\text{ReplicaSet}(C_i)|}\text{FailProbability}(s_j,\text{Replica}(R_p,C_i)).$$

式中:$|\text{ReplicaSet}(C_i)|$ 为软件组件 C_i 和它的组件副本总数目;$\text{Replica}(R_p,C_i)$ 为软件组件 C_i 或者它的副本组件提供服务 s_k.

$$\text{FailProbability}(s_k,C_i) = ph_j + (1 - ph_j) \times pc_i.$$

式中: 软件组件 C_i 部署在主机节点 H_j 上,ph_j 为主机节点 H_j 的失效率;pc_i 为软件组件 C_i 的失效率. 这里不考虑网络物理链接对系统可靠性的影响,认为网络物理链接是完全可靠的.

此外,系统所需的峰值带宽 BMS 为

$$BMS = \sum_{1\leqslant j\leqslant m,1\leqslant i\leqslant m,j\neq i} BM_{ij}.$$

式中 BM_{ij} 为主机节点 H_i 向主机节点 H_j 单向通信所需的带宽.

2　启发式的贪婪复制算法

2.1　启发式的贪婪复制算法描述

由于贪婪算法[12]每次在满足系统可用资源的情况下,选择使目标函数 OF 最大化的软件组件. 然而,对于某些软件组件,复制这些软件组件虽然能把 OF 提高最大,但消耗的可用内存和网络带宽也相当多. 基于这个原因,本文设计了启发式的贪婪复制算法,如图 1 所示. 该算法估计出复制某个软件组件增加的优化值、需要的带宽和内存,求出增加的优化值和需要的内存的优化比值 OM、增加的优化值和需要的带宽的优化比值 OB,将这两个优化比值作为选择软件组件进行复制的主要依据. 根据 OM 选择出一组满足约束条件的软件组件集合,根据 OB 选择出一组满足约束条件的软件集合,其中,优化值更高的软件组件集合就是该算法确定的将被复制的软件组件集合, 其中, $H(C_i) = H_j$ 为软件组件 C_i 部署到主机节点 H_j 上.

```
输入: n 个软件组件 C = {C1,C2,…,Cn},
      m 个主机节点 H = {H1,H2,…,Hm};
输出:将被复制的软件组件;
{flagband = 0, flagmemory = 0;
for 任意组件 Ci
    for each host node Hk ∈ H
  {设 Ci 副本部署到 Hk(Hk ≠ H(Ci));
计算增加的优化值 IOV (Ci,Hk);
计算增加的带宽 IB (Ci,Hk);
计算增加的内存 IM (Ci,Hk);
OB(Ci,Hk) = IOV (Ci,Hk)/IB (Ci,Hk);
OM(Ci,Hk) = IOV (Ci,Hk)/IM (Ci,Hk);}
/* 根据 OB 确定将被复制的软件组件 /
   while(flagband = 0)
   {选择当前最大值 OB(Ci,Hk);
If ((Ci 需要的内存 < = 系统可用内存)
&&(Ci 需要的带宽 < = 系统可用带宽))
    SelectCi, and mark Hk;
else flagband = 1;}
/* 根据 OM 确定将被复制的软件组件/
   while(flagmemory = 0)
{选择当前最大值 OM(Cj,Hp);
If ((Cj 需要的内存 < = 系统可用内存)
&&(Cj 需要的带宽 < = 系统可用带宽))
    SelectCj, and mark Hp;
else flagmemory = 1;}
输出有更高优化值的软件组件集合;}
```

图 1　启发式的贪婪复制算法的伪代码

2.2　启发式贪婪复制算法的正确性和时间复杂度分析

2.2.1　启发式的贪婪复制算法的正确性分析

启发式的贪婪复制算法是正确的,如果它对每一个输入都最终停止,而且产生正确的输出.

1)对于每一个输入算法都能最终停止.

设系统有 n 个软件组件 $C = \{C_1,C_2,\cdots,C_n\}$ 和 m 个主机节点 $H = \{H_1,H_2,\cdots,H_m\}$. 对于每个软件组件,假定将其复制其他 $m - 1$ 主机节点上后,计算增加的内存、增加的带宽和增加的优化值,这需要常数时间 $O(m)$. n 个软件组件的初始化过程最多执行 $O(mn)$ 步.

根据 OB 和 OM 来搜索将被复制的软件组件集合,如果选择的软件组件需要内存超过系统当前可用内存,或者需要的带宽超过系统当前可用带宽,则这两个循环将结束. 在软件组件复制前,设系统可用内存为有限的数 M_s,系统可用带宽为有限的数 B_s,在 n 个软件组件中,假定需要内存最小的为 $C_a \in C$,C_a 需要的内存为 M_a,那么最多经过$\lceil M_s/M_a \rceil$步,循环将由于系统可用内存分配完

而停止；同理，设需要带宽最小的软件组件为 $C_b \in C$，C_b 需要的带宽为 B_b，那么最多经过 $\lceil B_s/B_b \rceil$ 步，循环将由于系统可用带宽分配完而停止. 也就是最多经过 $\min\{\lceil M_s/M_a \rceil, \lceil B_s/B_b \rceil\}$ 步，两个循环将停止.

因此，对于每一个输入，最多经过 $O(mn) + \min\{\lceil M_s/M_a \rceil, \lceil B_s/B_b \rceil\}$ 步，该算法将停止.

2) 对于每一个输入算法都能产生正确的结果

这里不正确的结果包括两种情况：选择的软件组件复制集合不满足系统可用内存的限制和选择的软件组件复制集合不满足系统可用带宽的限制.

每次选择将被复制的软件组件时，首先判断该软件组件是否满足系统可用内存要求和带宽要求. 根据 OB 来选择软件组件时，若软件组件需要的内存超过了系统可用内存，或者软件组件需要的带宽超过了系统可用带宽，则循环下次执行时判断不满足执行条件，循环将退出. 对于根据 OM 来选择软件组件时，如果软件组件需要的内存超过了系统可用内存，或者软件组件需要的带宽超过了系统可用带宽，则循环下次执行时，判断不满足执行条件，循环将退出. 也就是如果遇到不满足内存和带宽约束条件的软件组件，算法将停止，也就选择出的将被复制的软件组件都满足约束条件. 因此，对于每一个输入，算法都能产生出正确的结果.

综上所述，启发式的贪婪复制算法是正确的.

2.2.2 启发式的贪婪复制算法时间复杂度分析

该算法初始化过程需时间 $O(mn)(m <= n)$，根据 OB 选择出一个最大值，选择出一个软件组件及其应该部署到的主机节点，需时间 $O(n^2)$. 若复制 $kn(k > 0, k$ 为常数) 个软件组件，那么花费的时间为 $O(kn^3)$. 根据 OM 选择出一个最大值，选择出一个软件组件及其应该部署到的主机节点，需时间 $O(n^2)$. 如果复制 ln ($l > 0$, l 为常数) 个软件组件，那么需时间为 $O(ln^3)$. 其中，$O(mn + (k + l)n^3) = O(n^3)$. 因此，启发式的贪婪复制算法的时间复杂度为 $O(n^3)$.

3 实验结果及分析

3.1 实验输入

系统包括 12 个软件组件和 4 个主机节点. 实验输入包括软件组件的大小、软件组件间的交互频率、软件组件间交互事件的大小、软件组件的失效率、主机节点的失效率和服务的重要性，并且根据文献[10]给出输入值的区间范围.

其中：$Size_k$ 为软件组件 C_k 的大小，为[1, 5]间的任意实数. $Size_1 = 3.53$，$Size_2 = 4.73$，$Size_3 = 4.87$，$Size_4 = 4.48$，$Size_5 = 3.19$，$Size_6 = 2.57$，$Size_7 = 4.86$，$Size_8 = 3.62$，$Size_9 = 3.18$，$Size_{10} = 3.67$，$Size_{11} = 4.93$，$Size_{12} = 2.48$；$fcc_{i,j}$ 为软件组件 C_i 和 C_j 间的交互频率，为[0,7]间的任意整数. $fcc_{3,4} = 2$，$fcc_{4,3} = 3$，$fcc_{8,9} = 4$，$fcc_{9,8} = 3$，$fcc_{6,8} = 3$，$fcc_{8,6} = 2$，$fcc_{1,10} = 5$，$fcc_{10,1} = 5$，$fcc_{1,3} = 6$，$fcc_{3,1} = 5$，$fcc_{2,5} = 1$，$fcc_{5,2} = 2$，$fcc_{6,7} = 2$，$fcc_{7,6} = 1$，$fcc_{5,6} = 7$，$fcc_{6,5} = 5$，$fcc_{4,11} = 2$，$fcc_{11,4} = 3$，$fcc_{1,12} = 1$，$fcc_{1,12} = 1$. 其他的软件组件间的交互频率为 0；$dcc_{i,j}$ 为软件组件 C_i 和 C_j 间的交互事件大小，为[1,8]间的任意实数. $dcc_{3,4} = 7.55$，$dcc_{4,3} = 6.20$，$dcc_{8,9} = 1.70$，$dcc_{9,8} = 1.20$，$dcc_{6,8} = 2.10$，$dcc_{8,6} = 2.20$，$dcc_{1,10} = 2.37$，$dcc_{10,1} = 1.80$，$dcc_{1,3} = 2.02$，$dcc_{3,1} = 1.60$，$dcc_{2,5} = 4.85$，$dcc_{5,2} = 3.28$，$dcc_{6,7} = 5.10$，$dcc_{7,6} = 5.30$，$dcc_{5,6} = 1.16$，$dcc_{6,5} = 1.20$，$dcc_{4,11} = 5.39$，$dcc_{11,4} = 6.00$，$dcc_{1,12} = 4.00$，$dcc_{1,12} = 3.00$. 其他的软件组件间的交互的事件大小为 0；pc_i 为软件组件 C_i 的失效率，为 [0,0.3] 间的任意实数 $pc_1 = 0.035\ 0$，$pc_2 = 0.230\ 7$，$pc_3 = 0.167\ 9$，$pc_4 = 0.154\ 2$，$pc_5 = 0.039\ 6$，$pc_6 = 0.071\ 4$，$pc_7 = 0.163\ 1$，$pc_8 = 0.294\ 4$，$pc_9 = 0.093\ 6$，$pc_{10} = 0.166\ 6$，$pc_{11} = 0.195\ 4$，$pc_{12} = 0.162\ 4$；ph_j 为主机节点 H_j 的失效率，为[0,0.1]间的任意实数. $ph_1 = 0.007\ 8$，$ph_2 = 0.046\ 2$，$ph_3 = 0.082\ 4$，$ph_4 = 0.059\ 4$；$Cri(s_i)$ 为服务 s_i 的重要性，为[1,3]间的任意整数. $Cri(s_1) = 2$，$Cri(s_2) = 3$，$Cri(s_3) = 2$，$Cri(s_4) = 2$；$cs_{i,j}$ 为软件组件 C_i 某种程度上提供服务 s_j，$cs_{i,j}$ 为 0 或者为 1. $cs_{3,1} = 1$，$cs_{11,1} = 1$，$cs_{4,1} = 1$，$cs_{2,2} = 1$，$cs_{5,2} = 1$，$cs_{6,2} = 1$，$cs_{7,2} = 1$，$cs_{8,3} = 1$，$cs_{9,3} = 1$，$cs_{1,4} = 1$，$cs_{10,4} = 1$，$cs_{12,4} = 1$.

3.2 资源消耗和得到的优化值的比较

本实验分析不同复制策略的资源消耗和得到的可靠性优化值. 这里采用文献[10]给出 12 个软件组件和 4 个主机节点的一种部署架构关系，如图 2 所示. 软件组件 C_3、C_9 和 C_{12} 部署到主机节点 H_1，软件组件 C_8 和 C_{10} 部署到主机节点 H_2，软件组件 C_4 和 C_6 部署到主机节点 H_3，软件组件 C_1、C_2、C_5、C_7 和 C_{11} 部署到主机节点 H_4.

当系统中可用资源非常有限，只能复制一部分软件组件. 为了充分利用系统可用资源和得到更高的优化值，贪婪算法每次选择使目标优化值提高最大的软件组件来复制. 然而，这个策略通常得到近似最优的软件组件集合，本文利用启发式的贪婪复制算法选择出将被复制的软件组件. 贪婪算法和启发式的贪婪复制算法选择的软件组件所消耗的系

统资源和得到优化值的比较如表 1 所示.

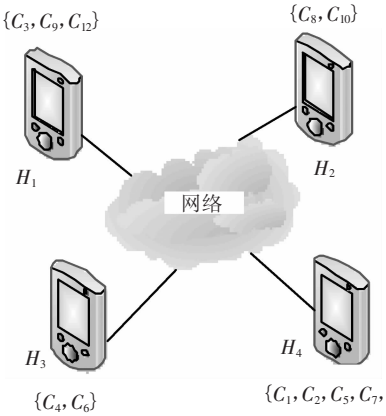

图 2　软件组件在主机节点上的部署图

表 1　贪婪算法和启发式的贪婪复制算法的结果比较

算法	所需的带宽	所需的内存	*OF*
基本情况	161. 17	186. 16	22. 969 4
贪婪算法(130%)	195. 08	238. 29	24. 407 3
启发式的贪婪复制算法(130%)	204. 13	241. 21	24. 665 3
贪婪算法(140%)	219. 63	258. 70	24. 929 3
启发式的贪婪复制算法(140%)	193. 70	257. 99	25. 135 3

在表 1 中,第 1 种情况为当系统可用资源增加 30%,第 2 种情况为系统可用资源增加了 40%. 当系统可用资源增加 30% 时,贪婪算法选择复制的软件组件为 C_2、C_7 和 C_{12},软件组件 C_2 和 C_7 的组件副本部署到主机节点 H_1 上,软件组件 C_{12} 的组件副本部署到主机节点 H_3;启发式的贪婪复制算法选择复制的软件组件为 C_2、C_8、C_{10} 和 C_{12},其中软件组件 C_2、C_8 和 C_{10} 的组件副本部署到主机节点 H_1 上,软件组件 C_{12} 的组件副本部署到主机节点 H_4. 当系统可用资源增加 40% 时,贪婪算法选择将被复制的软件组件为 C_2、C_7、C_{10} 和 C_8,这些组件副本均部署到主机节点 H_1 上;启发式的贪婪复制算法选择复制的软件组件为 C_2、C_7、C_8、C_{12} 和 C_6,其中软件组件 C_2、C_7、C_8 和 C_6 的组件副本均部署到主机节点 H_1 上,C_{12} 的组件副本部署到主机节点 H_3 上.

从表 1 可以看出,当系统可用资源增加 30% 时,启发式的贪婪复制算法实际消耗的带宽和内存均高于贪婪算法,但启发式的贪婪复制算法在充分利用系统可用资源的情况下,能得到更高的优化值. 当系统可用资源增加 40% 时,启发式的贪婪复制算法实际消耗的带宽和内存都低于贪婪算法,并且启发式的贪婪复制算法得到的优化值高于贪婪算法. 因此,在系统可用资源有限的情况下,与贪婪算法相比,启发式的贪婪复制算法能充分利用系统可用资源,更有效地选择将被复制的软件组件,得到更高的可靠性优化值.

3.3　复制算法对可靠性影响的比较

对于资源受限情况,为了使实验简明,分析系统可用资源增加 30% 和系统可用资源增加 40% 的这两种情况. 当系统可用资源增加 30% 时,采用启发式的贪婪复制算法和贪婪算法选择需要被复制的软件组件,软件组件提供的服务可靠性如图 3 所示. 图 3 中,G30 为用贪婪算法得到软件组件提供的服务可靠性;A30 为用启发式的贪婪复制算法得到的软件组件提供的服务可靠性.

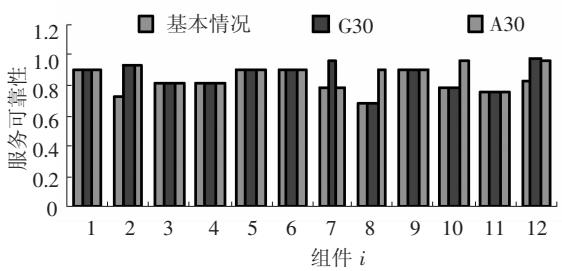

图 3　系统资源增加 30% 时,组件提供服务的可靠性变化

从图 3 可以看出,当系统可用资源增加 30%,贪婪算法选择复制的软件组件为 C_2、C_7 和 C_{12},启发式的贪婪复制算法选择复制的软件组件为 C_2、C_8、C_{10} 和 C_{12}. 这两个复制算法都选择提供服务重要性高的软件组件进行复制. 贪婪算法复制软件组件后,消除了复制前系统中的薄弱点 C_2,但软件组件 C_8 提供的服务可靠性仍然很低. 而且,软件组件复制后,C_8 提供的服务可靠性与其他服务可靠性的差距加大. 启发式的贪婪复制算法复制软件组件后,C_2 和 C_8 提供的服务可靠性明显提高,消除复制前系统中的薄弱点 C_2 和 C_8.

当系统可用资源增加 40% 时,采用启发式的贪婪复制算法和贪婪算法选择需要被复制的软件组件,软件组件提供的服务可靠性如图 4 所示. 图 4中,G40 为采用贪婪算法复制软件组件后,软件组件提供的服务可靠性;A40 为采用启发式的贪婪复制算法选择复制的软件组件后,软件组件提供的服务可靠性.

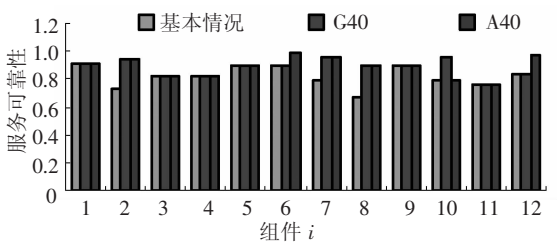

图 4　系统资源增加 40% 时,组件提供服务的可靠性变化

从图 4 可以看出,当系统可用资源增加 40%

时,贪婪算法选择复制的软件组件为 C_2、C_7、C_8 和 C_{10},启发式的贪婪复制算法选择复制的软件组件为 C_2、C_7、C_8、C_6 和 C_{12}. 当系统可用资源受限时,两种算法都选择软件组件可靠性低和其提供服务重要性高的 C_2 和 C_8 来复制,消除了复制前系统中的薄弱点,使系统更稳定地提供服务. 与贪婪算法相比,启发式的贪婪复制算法能充分利用有限的可用系统资源,更有效地选择出将被复制的软件组件,提高更多服务的可靠性.

当给定有限的系统可用资源时,为了更清楚地看出系统中服务可靠性的整体变化,给出了服务可靠性的均值和标准差的比较,如表2所示.

表2 服务可靠性的均值和标准差的比较

指标	平均值	标准差
基本情况	0.818 8	0.076 5
G30	0.862 6	0.089 5
A30	0.880 7	0.067 4
G40	0.884 3	0.062 9
A40	0.889 2	0.074 1

从表2可以看出,当系统可用资源增加30%和40%时,启发式的贪婪复制算法得到的服务可靠性的平均值都高于贪婪算法,与复制前的服务可靠性相比,整个服务的可靠性有了明显的提高. 当系统可用资源增加30%时,启发式的贪婪复制算法得到的服务可靠性的标准差低于复制前,也就是系统中各个服务的可靠性差距变小. 贪婪复制算法得到的服务可靠性的标准差高于复制前,使系统中各个服务的可靠性差距变大. 另外,启发式的贪婪复制算法得到的服务标准差低于贪婪算法,表明系统中各个服务的可靠性差别更小,不存在提高整体可靠性的薄弱点. 当系统可用资源增加40%时,启发式的贪婪复制算法得到的服务标准差高于贪婪算法,再结合图4可以看出,复制软件组件 C_6 和 C_{12},其服务可靠性提高幅度较大,使得服务的可靠性差距变大,而这是一种有益的提高.

4 结 论

1) 在分析可靠性优化目标函数 OF 和贪婪算法的基础上,设计了启发式的贪婪复制算法,该算法根据 OB 和 OM 来选择将被复制的软件组件;

2) 当增加相同的系统可用资源时,与贪婪算法相比,该算法能充分利用系统可用资源,得到更高的可靠性优化值和服务可靠性.

参考文献:

[1] 梅宏, 申峻嵘. 软件体系结构研究进展[J]. 软件学报. 2006, 17(6): 1257 – 1275.

[2] BASTARRICA M C, SHVARTSMAN A A, DEMURJIAN S A. A binary integer programming model for optimal object distribution[C]//International Conference on Principle of Distributed Systems. France: CiteSeerx Press, 1998: 211 – 226.

[3] HUNTG, SCOTT M L. The coign automation distributed partitioning system[C]//Proceedings of the Third Symposium on Operating System Design and Implementation. Berkeley, CA: USENIX Association, 1999:187 – 200.

[4] MALEK S. A user-centric Approach for Improving a Distributed Software System's Deployment Architecture [D]. USA: Dissertation of University of Southern California, 2007: 1 – 186.

[5] MIKIC-RAKIC M, MALEK S, MEDVIDOVIC N. Improving availability in large, distributed, component-based systems via redeployment[J]. Lecture Notes in Computer Science, 2005, 3798/2005: 83 – 98.

[6] MALEK S, SEO C, RAVULA S, *et al.* Reconceptualizing a family of heterogeneous embedded systems via explicit architectural support[C]//Proceedings of the 29th International Conference on Software Engineering (ICSE'07). Washington, DC: IEEE Computer Society, 2007: 591 – 601.

[7] KUTZNER K, FUHRMANN T. Measuring large overlay networks-the overnet example[J]. Informatik Aktuell, 2005, Teil IV: 193 – 204.

[8] PLISSONNEAU L, COSTEUX J L, BROWN P. Analysis of peer-to-peer traffic on ADSL[J]. Lecture Notes in Computer Science, 2005, 3431/2005: 69 – 82.

[9] EIDENBENZ S, WIDMAYER P. An approximation algorithm for minimum convex cover with logarithmic performance guarantee[J]. SIAM Journal on Computing, 2003, 32(3): 654 – 670.

[10] POPESCU D. Framework for replica selection in fault-tolerant distributed systems[R/OL]. Technical Report USC-CSSE-2007-702, 2007. http://csse.usc.edu/csse/TECHRPTS/2007/2007_main_exp.html.

[11] MALEK S, MEDVIDOVIC N, SEO Chiyoung, *et al.* A user centric approach for improving a distributed software system's deployment[R/OL]. USC-CSE-2006-602, 2006. http://csse.usc.edu/csse/TECHRPTS/2006/2006_main_exp.html.

[12] GARLAN D, SHAW M. An introduction to software architecture[J]. Advances in Software Engineering and Knowledge Engineering, 1993, 2(1):1 – 39.

(编辑 张 红)