

基于划分算法的 SaaS 寻址中断软件生成策略

周相兵^{1,2,3*}, 杨兴江^{1,2}, 马洪江¹

(1. 阿坝师范高等专科学校 计算机科学系, 成都 611741; 2. 电子科技大学 计算机科学与工程学院, 成都 610054;

3. 成都理工大学 信息科学与技术学院 成都 610059)

(* 通信作者电子邮箱 3dsmaxmaya@163.com)

摘要:针对软件即服务(SaaS)软件生成时所面临的 Web 服务和表述性状态转移(REST)接口难识别的问题,提出一种基于划分算法的 SaaS 软件生成方法。该方法采用划分算法将云计算系统中的各 SaaS 的功能进行划分,将各功能定义成一个节点,并定义各节点属性和计算各节点的相似度,以实现划分分类,从而提高功能搜索效率。在此基础上,根据需求变化实现功能寻址中断完成新的 SaaS 软件生成。最后通过在 Amazon 下的一个 SaaS 销售软件生成为例进行分析表明,该方法有效且可行。

关键词:软件即服务;划分算法;寻址中断;节点相似;Web 服务;云计算

中图分类号: TP311.5 **文献标志码:** A

Strategy of SaaS addressing and interrupt for software generation based on partitioning algorithm

ZHOU Xiang-bing^{1,2,3*}, YANG Xing-jiang^{1,2}, MA Hong-jiang¹

(1. Department of Computer Science, Aba Teachers College, Chengdu Sichuan 611741, China;

2. School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu Sichuan 610054, China;

3. School of Information Science and Technology, Chengdu University of Technology, Chengdu Sichuan 610059, China)

Abstract: There are some SaaS problems for Web service and REST (Representational State Transfer) interfaces recognition in the software generation process. Therefore, an approach was proposed based on partitioning algorithm, which employed partitioning algorithm to implement function partition of SaaS and define difference nodes for difference functions. At the same time, the similarity between nodes was defined to accomplish partition, which improved the efficiency of SaaS functions. Secondly, according to the changing requirements, addressing and interrupt approach was presented to realize software generation of SaaS. Finally, an SaaS online sale software in Amazon cloud computing was analyzed, which approves that the approach is feasible and available.

Key words: Software as a Service (SaaS); addressing and interrupt; nodes similarity; Web service; cloud computing

0 引言

云计算目的是实现网络环境可伸缩性和可靠性,是一种新型、基于 Internet 的分布式计算方法,是对并行计算、分布计算和网格计算的进一步扩展,以构建下一代计算机网络^[1],而软件即服务(Software as a Service, SaaS)是以云计算为基础的一种快速、共享、灵活、重用、租约的 IT (Information Technology) 应用程序生成方法。在面向用户上, SaaS 具有无硬件、无维护、无升级等多种优点,因此出现了很多相关的 SaaS 平台产品,如 Google Apps Engine、Salesforce.com、Amazon EC2 (Amazon Elastic Compute Cloud)、Microsoft Office Live、NetSuite Suiteflex 等和相关应用方法^[2], 这些 SaaS 产品在很大程度上降低了软件开发的成本预算风险,延长了软件生命周期,提高了软件的可复用性。但它们的实质还是以 Internet 为基础的,通过在物理层上部署多种虚拟机实现 SaaS 租约和组装^[3],并通过 Web 服务接口和表述性状态转移 (Representational State Transfer, REST) 来实现按需应用程序生成^[4]。其中一个最关键的问题之一就是怎样在云计算系

统中通过 Web 服务和 REST 接口将不同 SaaS 实体的功能进行重组生成按需的软件。因此,首先将 SaaS 的功能定义为节点,采用划分算法进行划分,并通过计算相似度进行分类。划分算法最早是 EDA (Electronic Design Automation) 工具中的基础算法,也是实现超大规模集成电路层次化设计的重要方法^[5],近年已经被应用到了复杂网络和网络社区划分^[6]、负载均衡划分^[7]等。本文则采用划分算法实现 SaaS 划分,为在需求下实现 SaaS 中的各功能寻址中断完成新的 SaaS 软件生成。寻址中断在文献[4]进行了描述和定义,在本文中则进一步优化寻址中断方法,并引入划分算法,以提高 SaaS 软件生成效率。

1 SaaS 划分方法

SaaS 划分的主要思想是将云计算系统中(主要包括 IaaS (Infrastructure as a Service) 和 PaaS (Platform as a Service)) 的 SaaS 软件实体的各功能形式化为图的节点,并计算各节点的相似度,这里相似度是指两个节点在同一云计算系统的概率,与节点间连接的传递权值正相关^[8],即按相似度高的节点进

收稿日期: 2011-07-21; 修回日期: 2011-09-16。

基金项目: 四川省科技厅计划项目(2010JY0J41); 湖南省自然科学基金资助项目(10JJ6099)。

作者简介: 周相兵(1980-),男,四川仪陇人,副教授,硕士研究生,CCF 会员,主要研究方向:服务与云计算、开源软件、旅游信息化; 杨兴江(1971-),男,重庆人,副教授,博士研究生,CCF 会员,主要研究方向:计算机软件、计算机安全; 马洪江(1968-),男,四川邛崃人,教授,主要研究方向:计算机网络、服务计算。

行划分。然后采用划分算法把图分类的相似度的节点生成树,即通过相似度把在图中生成按需求的划分树。

定义 1 将 SaaS 层次定义为 $saas = \langle saas_1(f_{11}, f_{12}, \dots, f_{1n}), saas_2(f_{21}, f_{22}, \dots, f_{2n}), \dots, saas_n(f_{n1}, f_{n2}, \dots, f_{nn}) \rangle$, $f_{nn} = \langle ws_{n1}, ws_{n2}, \dots, ws_{nn} \rangle$, $ws = \langle wss, wsp, qos, flow, wsd, uddi \rangle$ 。其中 $saas$ 中表示 SaaS 的功能组成, f 表示各功能由 Web 服务组成(包括服务接口), ws 中分别表示请求服务、响应服务、服务质量、服务流程、WSDL 和 UDDI。

定义 2 f_{nn} 中的服务接口衔接定义一个接口自动机 $SFA = \langle A, L, S, T \rangle$, 其中: $A \subseteq M \times O$, 表示 SFA 中可能出现的动作集, 即 M 代表一个有穷的方法集合, O 代表一个有穷的结果集合; L 表示 Web 服务接口在 SaaS 中的位置集合, $1 \in L$ 是返回位置; S, T 表示 Web 服务迁移集合, $S \cap T = \emptyset$, $S \cup T \neq \emptyset$, $A \subseteq (S \times T) \cup (T(S) \rightarrow L)$ 。

定义 3 将 SaaS 需求定义为 $saasr = \langle r_1, r_1 \rightarrow r_2, r_1 \rightarrow r_2 \rightarrow r_3, \dots, r_1 \rightarrow r_2 \rightarrow r_3 \rightarrow \dots \rightarrow r_n \rangle$, 且满足关系 $\delta: saasr \rightarrow saas$, δ 指定义 2 中的 SFA , 它是为寻址中断指定 SaaS 中的服务功能。

定义 4 将 SaaS 定义为 $G = \langle V, E, Sim(V_i, V_j) \rangle$ 是一个有权有向图, V 是顶点集, E 是边集, $Sim(V_i, V_j)$ 是权集, 顶点数 $n = |V(G)|$, 边数 $m = |E(G)|$; 并且对于 $\forall e_k \in E$, $\exists w_k \in Sim(V_i, V_j) (1 \leq k \leq m)$, 则 $e_k = (v_i, v_j) \in E$ 表明节点 v_i 与 v_j 间的依赖关系, w_k 为该边所对应 2 个节点的数据通信量(即相似度), 则根据文献[8]得节点 i 与 j 间的相似度矩阵 A_{ij} 为:

$$A_{ij} = \sum_{u=1}^m \left(\frac{1}{\prod_{l=1, l \neq i, j=1}^n d_{ij} D_l} \right)$$

其中: d_{ij} 为节点 i, j 间的距离, 节点连接数为 D_i 。同时, 在计算节点 i 与 j 间的距离 d_{ij} 时, 为了避免直接取得无穷大的情况, 采用相似度除以矩阵中该行元素值的和并进行归一化处理, 从而得到 A_{ij}' 。

定义 5 设 $P = \{P_1, P_2, \dots, P_n\}$ 是 $G = \langle V, E, Sim(V_i, V_j) \rangle$ 的一个划分, P_i 为 G 中部分节点的集合, 称为模块, 并记 $X(p_i)$ 是属于 P_i 的所有划分节点集合。 P_i 的最小划分用 $X(-1, -1, \dots, -1)$ 表示, 最大划分用 $X(-n, 0, \dots, 0)$ 表示^[9]; $P(v_i)$ 为节点 v_i 所在的模块。且该划分满足以下条件:

- 1) $\bigcap_{i=1}^n X(P_i) = \emptyset$; 2) $\bigcup_{i=1}^n X(P_i) = V$; 3) $\forall P_i \in P \wedge P_i \neq \emptyset$;
- 4) $\forall e_k = (v_i, v_j) \in E, X(v_i) \rightarrow X(v_j)$ 。

定义 6^[10] SaaS 划分由三部分组成: $SaaS_p = \langle root(i, x), Jion_block(i, j, x), meet_hash(x_1, x_2) \rangle$, 其中, $root(i, x)$ 表示将 G 划分成树的根, $Jion_block(i, j, x)$ 表示 P_i 和 P_j 两个模块连接, $meet_hash(x_1, x_2)$ 表示采用 Hash 表来平衡不同块的连接的时间。

这时可以得到如下的 SaaS 划分算法, 该算法由三个模块组成: 其中 Module1 是用来获得划分, Module2 用来实现所划分模块的连接, Module3 用来判断各模块间的满足性。

算法 1 SaaS 划分算法。

Input: SaaS 需求;

Output: SaaS 划分结果。

//算法 1 中的 Module1 的描述

Module1: $root(i, x)$;

//获得划分

1) $\delta(SFA): saasr \rightarrow saas$;

//SaaS 需求通过接口自动机启动 SaaS 软件实体

2) if ($X(p_i) \neq \emptyset$)

3) for(int $a = 0; a++;$ $a < i$)

4) for(int $b = 0; b++;$ $b < n$)

5) $Sim(a, b) \rightarrow A_{ij}$; //计算各功能节点的相似度

6) $\max[Sim(a, b)] \rightarrow root(i, x)$;

//在每一组中的选最大的相似度作为 SaaS 划分树的根

7) $Sim(a, b): saasr \rightarrow Sort(X(p_i))$;

//按照 SaaS 需求对节点相似度分类

8) $\{X(saas_1), X(saas_2), \dots, X(saas_n)\}$;

//获得 SaaS 的划分结果

9) else

10) return 1);

//算法 1 中的 Module2 的描述

Module2: $Jion_block(i, j, x)$;

//划分模块连接

1) $\delta(SFA): saasr \rightarrow saas$;

2) if ($\bigcap_{i=1}^N X(saas_i) = \emptyset \ \&\& \ \bigcup_{i=1}^N X(saas_i) = saas$)

3) for(int $a = 0; a++;$ $a < i$) // i 表示 SaaS 划分模块数

4) $root(i, X(saas_i)) \rightarrow r_i$;

5) $root(j, X(saas_j)) \rightarrow r_j$;

6) if ($r_i \neq r_j$)

7) $X(saas_i) \rightarrow s_i$; //将 SaaS 划分临时存储到 s_i 中

8) $X(saas_j) \rightarrow s_j$;

9) if ($(s_i < s_j) \ \&\& \ ? \ Sim(a, b)$)

//? 表示功能节点(Web 服务)相似度值相同或相近

10) $r_j \rightarrow X(saas_i)$;

11) $s_i + s_j \rightarrow X(saas_i)$;

12) else

13) $r_i \rightarrow X(saas_j)$;

14) $s_i + s_j \rightarrow X(saas_i)$;

15) return ($X(s_i + s_j)$);

16) else

17) return 1);

//算法 1 中的 Module3 的描述

Module3: $meet_hash(x_1, x_2)$;

//模块连接满足性判断

1) $\delta(SFA): saasr \rightarrow saas$;

2) $size(X(saas_i)) \rightarrow n$; //size 测量指划分模块的大小

3) if ($x(saas_i) \rightarrow x(saas_j)$)

4) for(int $a = 0; a < n; a++$)

5) $root(i, X(saas_i)) \rightarrow r_i$;

6) $root(j, X(saas_j)) \rightarrow r_j$;

7) if ($HT[r_i, r_j]$)

//建立 r_i, r_j 的哈希表

8) $HT[r_i, r_j] \rightarrow r$;

9) $X(r) - 1 \rightarrow X(r)$;

10) $r \rightarrow X(i)$;

11) $r \rightarrow X(j)$;

12) else

13) $i \rightarrow HT[r_i, r_j]$;

14) $-1 \rightarrow X(i)$;

15) $j \rightarrow HT[r_i, r_j]$;

16) $-1 \rightarrow X(j)$;

17) return ($X(saas_i + saas_j)$); //返回模块连接的满足性判断

18) else

19) return 1);

在 Module1 中, 其复杂度 $O(\text{Module1}) = n * i(n$ 表示参与 SaaS 软件生成的功能总数, i 表示 SaaS 模块划分数), 在 SaaS 需求下, 通过计算各 SaaS 中的功能节点的相似度, 并选择相似度最大的节点作为根, 这样避免了根的多样性, 简化了 SaaS 划分树形成的过程; 特别地, 通过相似度来分类 SaaS, 使得 SaaS 划分更清晰、更能在 $\delta(SFA): saasr \rightarrow saas$ 下实现划分模块的连接。

在 Module2 中, 其复杂度 $O(\text{Module2}) = i \lg n(n$ 指参与 SaaS 软件生成的模块总数, i 为 SaaS 模块划分数), 实现了各个 SaaS 划分模块连接, 而连接是在 $\delta(SFA): saasr \rightarrow saas$ 下

通过各 SaaS 中的功能进行的,这为第2章寻址中断完成样的 SaaS 软件生成创造了条件。

在 Module3 中,其复杂度 $O(\text{Module3}) = n * m * t$ (n 表示模块总数, m 表示哈希序列数, t 表示希树叶子节点的最大容量);而采用哈希表的目的是解决任意两个 SaaS 模块连接在 $i \text{ lb } n$ 下的满足性,即构建平衡树和其他数据结构的一致性。

综上,算法1的复杂度可以表示为 $O(\text{算法1}) = n * i + i \text{ lb } n + n * m * t$ 。现通过下面引理可以得到 SaaS 划分定理。

引理^[10] 如果算法1通过多次 $\delta(SFA): saasr \rightarrow saas$ 所得到的最小划分树,则这个划分结果的深度从不超过 $\text{lb } n$ 。

定理1 对于任意的 SaaS 划分 $X(saas_i)$ ($i = 1, 2, \dots, N$),当 $\delta(SFA): saasr \rightarrow saas$ 时,至少存在一棵深度从不超过 $\text{lb } n$ 的划分树。

2 优化 SaaS 寻址中断的 SaaS 软件生成方法

在有效的云计算系统中,通过在 IaaS 和 PaaS 的支持下,

表1 SaaS 寻址方法描述

符号	名称	描述
A_1	直接寻址	if $\forall f_{nn}, \exists f_{nn} \rightarrow saas, ws \rightarrow f_{nn}$, then $saasr: SaaS \rightarrow saas$
A_2	间接寻址	if $\forall f_{nn}, \exists f_{nn} \rightarrow saas, ws_i \rightarrow ws_j \rightarrow f_{nn}$, then $saasr: SaaS \rightarrow saas$
A_3	f_{nn} 内寻址	if $\forall \{ws_1, ws_2, \dots, ws_n\}, \exists ws_i \rightarrow f_{nn} \rightarrow saas$, then $saasr: SaaS \rightarrow saas$
A_4	SaaS 内寻址	if $\forall \{f_{1n}, f_{2n}, \dots, f_{nn}\}, \exists f_{nj}(ws_i) \rightarrow saas_i, f_{nj}(ws_j) \rightarrow saas_j, saas_i \rightarrow saas_j$, then $saasr: SaaS \rightarrow saas$
A_5	f_{nn} 间寻址	if $\forall \{ws_1, ws_2, \dots, ws_n\}, \{f_{1n}, f_{2n}, \dots, f_{nn}\}, \exists ws_i \rightarrow f_{nn}$, then $saasr: SaaS \rightarrow saas$
A_6	SaaS 间寻址	if $\forall \{ws_1, ws_2, \dots, ws_n\}, \{f_{1n}, f_{2n}, \dots, f_{nn}\}, \exists \{saas_1, saas_2, \dots, saas_n\}, saas_1(f_{1n}(ws_1)) \rightarrow saas_2(f_{2n}(ws_2)) \rightarrow \dots \rightarrow saas_n(f_{nn}(ws_n))$, then $saasr: SaaS \rightarrow saas$
A_7	按需直接寻址	if $\forall \{saas_1, saas_2, \dots, saas_n\}, \exists saas_i \rightarrow ws_i \rightarrow f_{nn} \rightarrow saas$, then $saasr: SaaS \rightarrow saas$
A_8	按需间接寻址	if $\forall \{saas_1, saas_2, \dots, saas_n\}, \{f_{1n}, f_{2n}, \dots, f_{nn}\}, \exists saas_i \rightarrow f_{in} \{ws_1, ws_2, \dots, ws_n\} \rightarrow saas_i \{f_{1n}, f_{2n}, \dots, f_{nn}\}$, then $saasr: SaaS \rightarrow saas$

算法2 SaaS 软件生成的寻址算法。

Input: SaaS 需求;

Output: 满足需求的 SaaS 寻址结果。

```

1)  $\delta(SFA): saasr \rightarrow saas$ ;
2) call Module1; //调用算法1中的 Module1
3) if ( $(\{X(p_i)\} \neq \emptyset) \&\& ws$ )
4)   call Module2; //调用算法1中的 Module2
5)   for (int  $i = 0; i < 8; i++$ ) //8表示8种寻址方法
6)     for (int  $j = 0; j < N; j++$ ) //SaaS划分个数
7)       if ( $A_i(0,1) = 1$ ) //表示寻址成功
8)          $A[i][j] = 1$ ;
           //将寻址成功的标志存储在数组  $A[i][j]$  中
9)   Call Module3;
10)  else
11)    return 4);
12)  else
13)    return 2);

```

算法2的复杂为 $O(\text{算法2}) = n * i + i \text{ lb } n + n * m * t + 8N$,此算法首先调用算法1中的 Module1 对有效的云计算系统中的 SaaS 进行划分,判断 SaaS 划分的结果;其次调用 Module2 实现模块间连接,并在连接的过程中根据 SaaS 需求实现 SaaS 划分寻址,将寻址成功用标记1存在数组 $A[i][j]$ 中;然后调用 Module3 进行 SaaS 划分间连接的满足性判定。

2.2 SaaS 中断

定义8 SaaS 软件生成中断描述为 $SaaSI = \langle IQ, IN[m][n], IO, IR, IV, Queue \rangle$,其中:

1) $IQ = \{IQ1, IQ2, \dots, IQ10\}$ 表示中断请求。

当 SaaS 划分不能满足需求时,发出中断请求 $IQ1$;

当 SaaS is busy 时,发出中断请求 $IQ2$;

按需重新生成 SaaS 软件已经成为当前软件工程重要的研究领域^[11],也是实现软件高耦合松散、强复用、按需自动生成软件最有效的方法之一。芬兰赫尔辛基工业大学的 Peilin Guo 在一个研究报告中指出:在一定时间段内遗留系统与 SaaS 将同时存在,并明确指出 SaaS 最终将占据整个软件市场^[12]。针对如何获取一种有效的方法将云计算系统中的 SaaS 通过需求重新生成一个新的 SaaS 软件,本文在第1章中采用划分方法将 SaaS 进行了划分,这样可以更进一步明确 SaaS 的状态和应用范围。下面将在文献[4]的基础上进一步优化寻址中断方法,即根据 SaaS 的划分结果进行寻址中断 Web 服务,并以划分模块为基础寻址中断生成新的 SaaS 软件。

2.1 SaaS 寻址

定义7 SaaS 软件生成寻址类型定义为 $SaaSA = \langle A_1(0, 1), A_2(0, 1), \dots, A_8(0, 1) \rangle$,并且存在关系: $saasr: SaaSA \rightarrow saas$,其中 $A_i(0, 1)$ ($i = 1, 2, \dots, 8$) 表示不同的寻址方法,(0, 1) 表示寻址不成功与成功。其描述如表1所示。

当 SaaS 划分模块连接不成功时,发出中断请求 $IQ3$;

当 SaaS 划分模块连接不存在满足性时,发出中断请求 $IQ4$;

当 SaaS 寻址不成功时,发出中断请求 $IQ5$;

当 SaaS 中的功能实体 f 未被激活时,发出中断请求 $IQ6$;

当 f 中的 Web 服务不可用时,发出中断请求 $IQ7$;

当 f 与 SaaS 划分不匹配时,发出中断请求 $IQ8$;

当 Web 服务与 f 不匹配时,发出中断请求 $IQ9$;

当 SaaS 支撑系统 PaaS 不可用时,发出中断请求 $IQ10$ 。

2) $IN[m][n]$ 表示中断号数组,记录各中断请求的信息,其中 $m = 10$,代表10种不同类型的中断请求, n 为不同中断请求的数目,即 $IN[10][n]$ 。

3) IO 表示中断判优。设置一个中断请求判优触发器 $T_{INE}(0, 1)$,当 $T_{INE} = 0$ 时,判优停止;为 $T_{INE} = 1$ 时,判优运行。

算法3 中断判优。

Input: IQ ;

INIT $IN[10][n]$;

1) Call 算法2;

2) if ($(T_{INE} == 1) \&\& IN[10][n] \neq \emptyset$)

3) for (int $i = 0; i < 10; i++$)

4) for (int $j = 0; j < n; j++$)

5) $IQ \rightarrow IN[i][j]$;

6) else

7) $T_{INE} = 0$;

4) IR 表示中断响应。当 $T_{INE} = 1$ 转向 $T_{INE} = 0$ 时,表示 IR 已经完成了某中断请求的响应。将断点保存在数组 $BP[10][N]$ 中,再次获取 Web 服务接口,并重新寻址实现服务组合或 SaaS 划分连接。

算法 4 中断响应。

```

Input IR;
INIT IO;
1) Call 算法 3;
2) while ( $T_{INE}$ )
3)   for(int  $i = 0; i < 10; i++$ )
4)     for(int  $j = 0; j < N; j++$ )
5)       if ( $T_{INE} == 1$ )
6)          $IR \rightarrow BP[10][N]$ ;
7)       else
8)         return IO;          //返回中断判优
9)        $T_{INE} = 0$ ;

```

5) IV 表示中断向量。由 $A_i(0,1)$ 、 IQ 、 IR 组成向量中断,用数组 $IV[R][S]$ 来记录中断向量,并形成中断向量表(用 0,1 表示,1 表示中断请求与中断响应作用,0 表示中断请求与中断响应不作用)。

算法 5 中断向量表。

```

Input saasr;
INIT  $IV[M][N]$ ;
1) Call 算法 4;
2) if ( $saasr; SaaS \rightarrow SaaS$ )
3)   for(int  $i = 0; i < N; i++$ )
4)     for(int  $j = 0; j < N; j++$ )
5)       ( $\delta(SFA); saasr \rightarrow saas$ )  $\rightarrow IV[R][S]$ ;
           //表示在  $\delta(SFA); saasr \rightarrow saas$  作用下,
           //中断请求与中断响应作用后的中断向量结果
6)   if ( $IQ \rightarrow IR$ )
7)      $IV[i][j] = 1$ ;
8)   else
9)      $IV[i][j] = 0$ ;
10) else
11)   return 2);

```

6) $Queue$ 表示中断处理队列。中断队列用来实现中断处理过程消息分派、信息传送等操作。

算法 6 SaaS 中断算法。

```

Input: SaaS 需求;
Output: SaaS 生成结果
INIT: Queue;          //初始化中断处理队列
DEFINE IF(0,1);
           //定义中断允许标记,0 表示不允许,1 表示允许
1) Call 算法 5;
2) if ( $IN[10][n] \neq ()$ )
3)   for(int  $i = 0; i < N; i++$ )
4)     for(int  $j = 0; j < N; j++$ )
5)       ( $Pre(IN[i-1][j-1]), Suc(IN[i][j])$ )  $\rightarrow$ 
           Queue(IQ);
           //Pre, Suc 分别表示队列的前继,后驱,判断中断请求
6)        $T_{INE} = 1$ ;
7)        $Y = Record(T_{INE})$ ;          //记录判优数
8)        $IF = 1$ ;
9)       while ( $T_{INE} \&\& IF$ )
10)        ( $Pre(BP[i-1][j-1]), Suc(BP[i][j])$ )  $\rightarrow$ 
            Queue(IR);
11)        if ( $BP[i][j] \neq \emptyset$ )
12)          ( $Pre(IV[i-1][j-1]), Suc(IV[i][j])$ )  $\rightarrow$ 
            Queue(IV);
13)           $T_{INE} = 0$ ;
14)           $IF = 0$ ;
15)        else
16)          return 10);
17)        ( $Queue(IQ), Queue(IR)$ )  $\rightarrow Queue(IV)$ ;

```

```

18) Output(SaaS software);
19) else
20) return 2);

```

算法 6 是 SaaS 生成过程中所涉及到的中断算法,其中断发生主要由 SaaS 划分和 SaaS 中断引发。算法 6 的复杂度为 $O(\text{算法 6}) = n * i + \text{ilb } n + n * m * t + 8N + 10N + 10n + 2Y * N^2$ (N 指 SaaS 划分个数),即算法 6 就是整个面向划分和寻址中断过程的 SaaS 软件生成的复杂度。在算法 6 中,首先调用算法 5 完成 SaaS 划分、寻址和中断;接着对中断过程中所涉及到的量进行处理,并将处理结果放到相应的队列中,以便在有效的云计算系统中实现按需的 SaaS 软件生成;最后输出 SaaS 软件生成结果^[14]。

3 案例分析

本文的案例分析通过 Amazon 提供的 IaaS、PaaS 和 SaaS 构造一个在线销售系统的 SaaS 生成为例,并将本文的算法 1~6 编写一个程序包,其编写方法参考文献[4,13]来完成。该系统主要由制造商和分销商两大业务体构成,即分销商从制造商处购买商品并出售给客户。在 Amazon 云环境中提供了可供使用的 Web 服务接口,而在使用 Amazon 云前需要创建 Access Key ID 实现身份验证或读取队列相关项的密钥,若要创建、操作队列相关项时,还需要使用 Secret Access Key,在注册 Amazon 时就会得到这些密钥。这些操作可以在 Amazon 所提供的网页上获得,不过需求通过信用卡支付至少 1 美元的费用。通常情况下,分销商要判定库存量是否能够满足客户订购请求,若库存量不能够满足,则分销商需要向制造商发出一个购买订单;制造商收到订单后返回一个订单汇总给分销商,如图 1 所示^[4]。其中制造商、分销商、订单、库存等就是一个软件实体,即 SaaS;而涉及的 Amazon SQS(Amazon Simple Queue Service)、Amazon EC2 等就是云计算所支持平台,即 PaaS;其他的就是 IaaS。

实现 SaaS 软件生成流程^[4]如下:

明确软件自动生成的需求,若某时有一客户 Customer 需要在基于 Amazon 的销售系统上购买一本书,即产生的 SaaS、服务模块和基本服务和涉及的软件服务有:

$SaaS = (\{\text{验证}\}, \{\text{搜索}\}, \{\text{制造商}\}, \{\text{分销商}\}, \{\text{库存}\}, \text{操作}\{\text{创建通道/队列服务}, \text{发送/接收消息服务}, \text{购买订单服务}, \text{查看服务}, \text{订单汇总服务}, \text{商品类型服务}, \text{XML 处理服务}, \text{返回服务}, \text{配置服务}\})$ 。并且假设这些 SaaS 都是公布于 Internet 中的可用的软件实体,开发人员(Developer)可以任意操作。

$F = (\text{验证模块}, \text{搜索模块}, \text{销售模块}, \text{支持模块})$;

$ws = (\text{登录服务}, \text{搜索服务}, \text{显示服务}, \text{订单服务}, \text{付费方式服务}, \text{付费服务})$;

$PaaS = (\text{Amazon SQS}, \text{Amazon EC2}, \text{Amazon S3}, \text{Amazon SDB})$ (注:Amazon S3 指 Amazon Simple Storage Service)。

下面主要从浏览订单、查询经销商、查询制造商、浏览库存(如图 2~5)角度与文献[4]进行比较。

从图 2~5 易知,在某一时刻内,采用 SaaS 划分策略的寻址中断软件生成方法与文献[4]相比有明显的优点。这是因为采用划分 SaaS 后, SaaS 在 Internet 中状态更明确,更易于 SaaS 新的软件生成;而且销售商、制造商与库存量更易达到平衡(如图 3、4),这是由于基于 Amazon 的在线销售系统生成效率提高了;同时,库存量随时间变化也趋于稳定化(如图 5)。

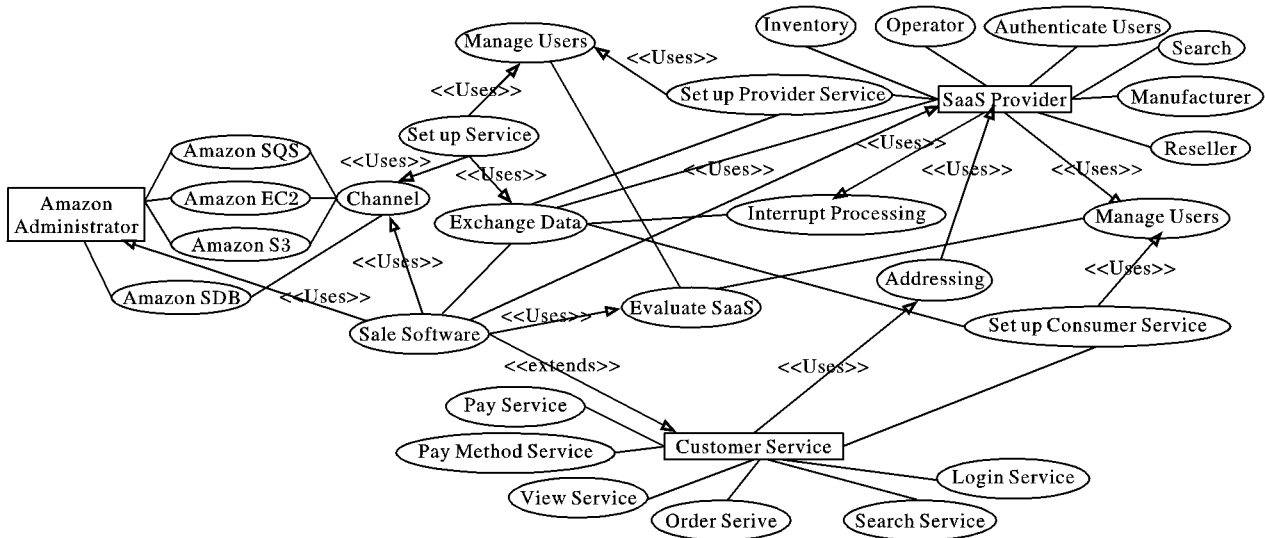


图1 基于 Amazon 云销售系统业务流程

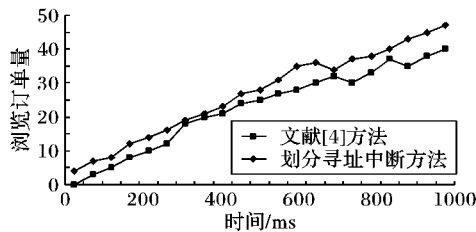


图2 浏览订单量与时间关系的变化曲线

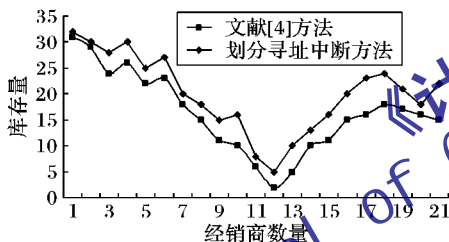


图3 经销商数量与库存量关系的变化曲线

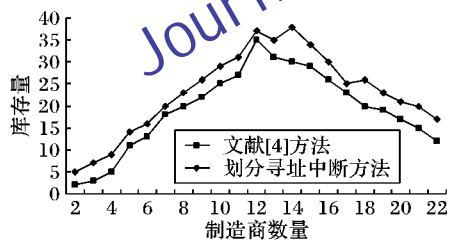


图4 制造商数量与库存量关系的变化曲线

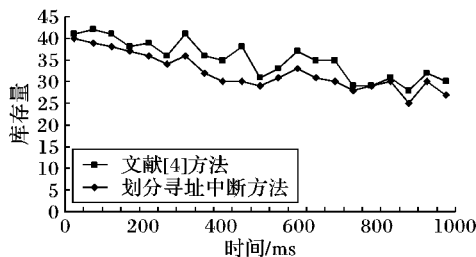


图5 库存量与时间关系的变化曲线

4 结语

本文对 SaaS 定义进行了层次化的形式定义,这样可以更准确地描述 SaaS 在有效的云计算系统中的作用;并引入划分原理,通过相似度将 SaaS 进行划分分类,这样可以更方便按需 SaaS 寻址中断生成新的 SaaS 软件系统。最后,通过在

Amazon 云系统中的在线销售系统为例分析了本文方法的可行性和有效性。在后续工作中,将继续研究在云计算环境下, SaaS 软件生成原理和应用方法,并逐步建立旅游云,以便最终将研究成果应用在旅游云中。

参考文献:

- [1] RINGS T, GRABOWSKI J, SCHULZ S, *et al.* Grid and cloud computing: Opportunities for integration with the next generation network [J]. *Grid Computing*, 2009, 7(3): 375 - 393.
- [2] 孔兰菊,李庆忠,史玉良,等. 面向 SaaS 应用基于键值对模式的多租户索引研究[J]. *计算机学报*, 2010, 33(12): 2239 - 2247.
- [3] di COSTANZO A, de ASSUNÇÃO M D, BUYIA R. Harnessing cloud technologies for a virtualized distributed computing infrastructure[J]. *IEEE Internet Computing*, 2009, 13(5): 24 - 33.
- [4] 周相兵,余莹,马洪江,等. 一种基于云计算的 SaaS 组装方法研究[J]. *小型微型计算机系统*, 2010, 31(10): 1942 - 1953.
- [5] 蒿杰,彭思龙. 多级划分算法的后处理与评价方法[J]. *小型微型计算机系统*, 2010, 31(1): 160 - 163.
- [6] PALLA G, DERENYI I, FARKAS I, *et al.* Uncovering the overlapping community structure of complex networks in nature and society [J]. *Nature*, 2005, 435(7043): 814 - 818.
- [7] 张颖星,姚益平. 乐观策略下并行离散事件仿真动态负载均衡优化算法[J]. *计算机学报*, 2010, 33(5): 813 - 821.
- [8] 杨柳,曹玖新,刘波,等. 基于无偏 Q 值反馈的社区划分算法[J]. *东南大学学报: 自然科学版*, 2011, 41(1): 31 - 36.
- [9] WEISS M A. *Data structures and algorithm analysis* [M]. 2nd ed. Redwood City, California: Benjamin/Cummings Publishing Company, 1994.
- [10] FREESE R. Partition algorithms [EB/OL]. [2011 - 05 - 16]. <http://www.math.hawaii.edu/~ralph/Notes/Partitions/partitions.pdf>.
- [11] SHIM J, HAN J, KIM J, *et al.* Patterns for configuration requirements of Software-as-a-Service [C]// SAC 2011: Proceedings of the 2011 ACM Symposium on Applied Computing. New York: ACM, 2011: 155 - 161.
- [12] GUO P. A survey of software as a service delivery paradigm [EB/OL]. [2011 - 06 - 09]. http://cse.ttk.fi/en/publications/B/5/papers/guo_final.pdf.
- [13] STEWART B J. Leveraging amazon Web services for enterprise application integration [EB/OL]. [2011 - 05 - 19]. http://www.ibm.com/developerworks/xml/library/x-amazonwsms/?S_TACT=105AGX52&S_CMP=content.
- [14] 陈小兵,武泽旭. 支持多类终端与服务定制的 SaaS 软件服务架构[J]. *计算机应用*, 2010, 30(10): 2754 - 2757, 2762.