

一种基于 QoS 的服务构件组合方法

廖 渊^{1),2),3)} 唐 磊^{1),2)} 李明树^{1),3)}

¹⁾(中国科学院软件研究所互联网软件技术实验室 北京 100080)

²⁾(中国科学院研究生院 北京 100039)

³⁾(中国科学院软件研究所计算机科学国家重点实验室 北京 100080)

摘 要 在面向服务的架构中,如何利用已有的服务构件组装成新的服务成为当前此领域的一个研究热点.该文以构件化嵌入式操作系统 Liquid 为背景,给出了一种基于 QoS 的服务构件组合方法,在满足组合服务的功能需求同时,满足其 QoS 需求.文章详细给出了所涉及的服务模型、QoS 模型以及构件选择基本算法.为使此服务构件组合方法适应于动态变化的系统环境,该文进一步对构件选择基本算法进行优化,给出了构件选择的启发和协商算法.最后通过实验,比较和分析三种算法的性能.

关键词 服务;构件;服务质量;组合

中图法分类号 TP311

A Method of QoS-Aware Service Components Composition

LIAO Yuan^{1),2),3)} TANG Lei^{1),2)} LI Ming-Shu^{1),3)}

¹⁾(Laboratory for Internet Software Technologies, Institute of Software, Chinese Academy of Sciences, Beijing 100080)

²⁾(Graduate School of the Chinese Academy of Sciences, Beijing 100080)

³⁾(Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100080)

Abstract How to composite a service from its components is a very active area of research on service-oriented architecture. This paper proposes a method of QoS-aware service components composition, which has applied into Liquid—A component-based embedded OS. In this method, not only functional requirements but also QoS constraints of composite service can be satisfied. In this approach, service is models as composite components that have QoS characteristics. And based on the service model and QoS model, a basic algorithm of component selection is given. In order to make this service composition method suit dynamic system, heuristic and negotiation approach are used to improve the basic selection algorithm performance. At last, for the purpose of verification and comparison of those three different algorithms, the results of an experiment are presented.

Keywords service; component; quality of service; composition

1 引 言

随着网络技术的发展,嵌入式设备联网的趋势

日渐明显. 嵌入式操作系统的运行环境也从孤立的单机环境发展为复杂的网络分布式环境,应用服务形式也出现了向以构件为基本服务单位发展的趋势,促使我们需要研究嵌入式系统的系统构件技术

收稿日期:2004-11-30;修改稿收到日期:2005-01-24. 本课题得到国家“八六三”高技术研究发展计划软件重大专项(2002AA1Z2302, 2004AA1Z2050)和国家自然科学基金项目(60273026)资助. 廖 渊,男,1975 年生,博士研究生,研究方向为实时系统、操作系统、构件软件工程、SOA. E-mail: liaoy@itechs.iscas.ac.cn. 唐 磊,男,1980 年生,博士研究生,研究方向为实时系统、软件工程. 李明树,男,1966 年生,博士,研究员,博士生导师,主要研究方向为软件需求工程、实时系统.

和服务构件技术,适应普适计算(Pervasive Computing)^[1]的需求. Liquid^[2,3]正是在此背景下研究开发的一种基于构件、面向服务的嵌入式操作系统. 如图 1 所示,在 Liquid 中,系统和应用服务是以功能构件作为提供服务的基本单位的,而 Liquid 内核不仅能

够对各种功能构件提供运行支持环境,同时为满足普适计算适应性需求,对环境变化具有应激能力,此层能够支持服务构件的动态组合,根据系统资源环境的变化以及构件的功能和 QoS 属性动态组合已有功能构件形成服务满足应用需求.

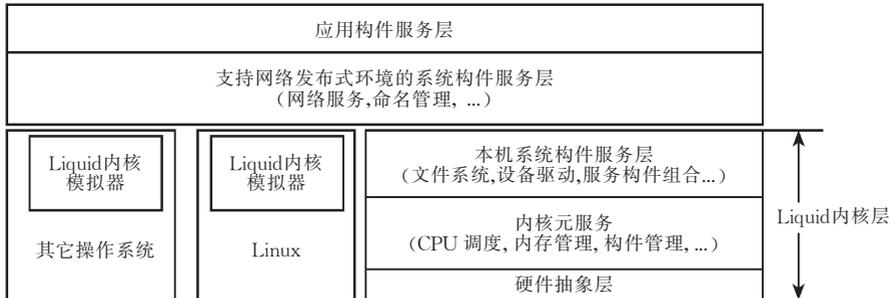


图 1 Liquid 操作系统层次结构图

虽然当前研究领域提出了各种关于服务组合方法^[4~7],但是基于 QoS 的服务组合方法较少,且这些方法不适合应用于 Liquid 操作系统中,主要原因有:(1)多数服务组合方法面向分布式企业计算环境,一些假设条件并不适合于嵌入式应用中,如 Zeng^[4]所定义的 QoS 模型完全针对商业应用;(2)多数已有的研究工作并没有考虑普适计算的动态变化环境;(3)目前多数基于 QoS 的服务组合方法为线性组合方法,只支持特定的服务类型,这样极大限制了这些方法的实际可用性.

针对这些问题,本文给出一种基于 QoS 的构件服务组合方法,首先给出此方法所针对的服务模型,通过分析服务构件的 QoS 特征进而定义其 QoS 模型;接着对所定义的服务模型和 QoS 模型的服务组合条件进行了分析,进而给出了 3 种用于服务组合的构件选择算法:基本算法、启发式算法、协商算法.其中,构件选择基本算法能够满足服务组合的基本需求,而启发式算法和协商算法则针对构件选择基本算法给出了不同的优化方法.通过实验,比较和分析了本文所提 3 种组合算法的性能;最后总结全文.

2 相关工作

本节简述相关工作.在操作系统和中间件领域,有许多关于 QoS 管理的研究^[8],特别是为了满足普适计算环境中应用对于 QoS 的需求,2KQ^[9],Gaia OS^[10,11],TAO^[12],QuO^[13],QoSME^[14],Agilos^[15]等系统给出了动态 QoS 管理方法.这些工作主要集中于 QoS 规范描述方法、系统和应用层的 QoS 映射、对动态资源的 QoS 适应方法以及系统对应用的 QoS

保证方法.这些工作很少涉及服务构件组合.

当前,对服务组合的研究非常活跃^[16],产生了许多服务组合方法,如基于服务语义本体的 Web 服务组合方法 DAML-S^[17].然而,在这些工作中,基于 QoS 的服务组合方法较少,其中 Gu 等^[18,19]和 Zeng^[4]分别针对 P2P 和 Web 服务应用各自给出了基于 QoS 的服务组合方法.由于所针对的应用不同,这些方法并不能应用于 Liquid 操作系统中,但是其中的一些想法被本文借鉴.如,Zeng 针对 Web Services 商业应用定义了一个 5 维 QoS 模型,基于此模型设计了本地和全局两种服务选择算法.其中基于整数规划方法的全局服务选择算法思想被本文借鉴.Gu 针对 P2P 应用特点,定义了基于 QoS 的服务系统模型,并且基于此系统模型,给出了适合于 P2P 环境下的服务组合方法,本文 QoS 模型借鉴了此方法关于服务的 QoS 属性描述部分.

3 基于 QoS 的服务构件组合

3.1 服务模型

在 Liquid 操作系统中,服务由单个原子构件或者由一组相关构件所提供,针对一组构件的服务请求,本文称为复合服务请求.应用的服务请求被 Liquid 截取,如果是针对单个原子构件的服务请求,直接映射到相应构件实例中;如果是复合服务请求,则 Liquid 通过服务构件组合方法,将服务请求映射到相关构件实例中,如图 2 所示,一个复合服务请求包括两个部分:(1)服务所涉及构件的功能配置图(Function Graph,FG).本文限定 FG 为一个有向无环图,它描述了所涉及构件的功能需求及其构件

间的相互功能依赖关系. 在 FG 中, 两个节点相连表示两者之间具有控制流或数据流依赖关系, 构件的功能配置图采用 XML 语言描述; (2) 服务的 QoS 需求. 其中, q_i^{req} 表示一个 QoS 需求参数的约束, 如延迟、视频输出帧率等的取值约束:

$$QoS_{\text{req}} = [q_1^{\text{req}}, q_2^{\text{req}}, \dots, q_m^{\text{req}}], \quad i = 1, 2, \dots, m \quad (1)$$

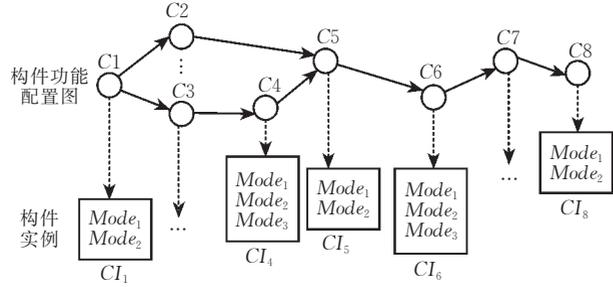


图 2 复合服务请求及构件映射

构件实例为每个构件部署到 Liquid 中的物理运行单元, 运行时, 构件实例表现为操作系统中的一个独立线程. 由于嵌入式应用往往有实时等 QoS 需求, 因此在 Liquid 中, 构件有 QoS 特征 (QoS-Aware), 而在运行时, 构件实例 CI 的 QoS 特征表现为

$$QoS_{CI} = (QoS_{\text{in}}, Res, QoS_{\text{out}}) \quad (2)$$

$$QoS_{\text{in}} = [q_1^{\text{in}}, q_2^{\text{in}}, \dots, q_h^{\text{in}}], \quad i = 1, 2, \dots, h;$$

$$Res = [q_1^{\text{res}}, q_2^{\text{res}}, \dots, q_k^{\text{res}}], \quad i = 1, 2, \dots, k;$$

$$QoS_{\text{out}} = [q_1^{\text{out}}, q_2^{\text{out}}, \dots, q_n^{\text{out}}], \quad i = 1, 2, \dots, n.$$

其中, QoS_{in} 和 QoS_{out} 分别表示构件实例所提供和需求的 QoS 属性列表, 这些 QoS 属性表现为应用构件的属性, 如视频服务构件的视频帧率; Res 则为构件实例运行时所占用的系统资源列表, 这些 QoS 属性由系统资源管理服务所提供, 比如 CPU 带宽占用率、网络带宽等等. QoS_{in} , Res , QoS_{out} 三者之间的关系是: QoS_{out} 中 QoS 属性值依赖于相关 Res 和 QoS_{in} 中的 QoS 属性值, 即构件实例为输出 QoS_{out} , 它需要相邻构件实例为其提供一定的 QoS_{in} 并且需要系统资源提供 Res 中的资源.

在 Liquid 中, 构件功能配置图中的节点被映射为具体构件实例, 这些构件实例的依赖关系等同于构件功能配置图所定义的依赖关系, 如图 2 中的 CI_5 和 CI_6 间的关系. 如果相邻构件实例 C_i 和 C_{i+1} 间的 QoS 属性形成依赖关系, 则 C_i 的某一输出 QoS 属性 $QoS_{\text{out}}.q_g^{\text{out}}$ 为 C_{i+1} 的一个所需 QoS 属性 $QoS_{\text{in}}.q_h^{\text{in}}$:

$$C_{i+1}.QoS_{\text{in}}.q_h^{\text{in}} = C_i.QoS_{\text{out}}.q_g^{\text{out}} \quad (3)$$

通过等式(3)的 QoS 依赖传递关系, 易知构件功能配置图中的尾节点构件实例的 QoS_{out} 元素用来满足等式(1)所定义的服务 QoS 需求, 如图 1 的

CI_8 . 有向图源节点构件实例的 QoS 特征中无 QoS_{in} 元素定义, 如图 1 的 CI_1 .

图 2 中的每个构件实例具有不同的模式 (Mode) 用于适应动态变化的系统资源环境, 每种模式实现相同的构件功能接口, 但是由于构件实现方法不同, 从而表现出不同的 QoS 特征, 具体而言, 构件运行在不同模式时, 其 QoS_{in} , Res , QoS_{out} 中的 QoS 属性值约束条件不同, 这些约束条件用于限制 Liquid 系统资源管理元服务的 QoS 以及相邻构件的输出 QoS 的取值范围, 从而保证构件实例的 QoS_{out} 在一定限制范围内. 例如为达到实时播放效果, 要求视频播放构件的输出帧率等于 30fps (frame/s), 它需要 CPU 调度元服务提供约 33% 的 CPU 资源带宽. 选择具体构件实例模式由系统资源状态和构件实例依赖关系决定.

通过以上分析, 在 Liquid 操作系统中, 对于复合服务请求, 服务组合方法需要根据其构件功能配置图 FG 将其映射到相关构件实例中, 并根据 FG 和系统资源状态, 选择构件实例的具体运行模式. 从而使得等式(1)的 QoS_{req} 需求能够满足, 并且每个构件实例的 QoS_{in} , Res 的约束条件能够满足.

3.2 QoS 模型

QoS 模型给出了服务构件的 QoS 特征, 由构件 QoS 规格及其组成部分描述. QoS 规格给出了服务构件实例的 QoS 属性值约束定义. 为便于描述, 下面先给出一个 QoS 属性的约束定义.

定义 1. QoS 属性约束. 一个 QoS 属性约束 $QoSConstraint = (q, op, Value)$ 给出 QoS 属性 q 的取值约束, $op \in \{=, \subseteq\}$ 为约束操作, $Value$ 称为 QoS 属性 q 的约束值. 当 q 约束为单值时, $q = Value$; 当 q 约束范围为集合时, $q \subseteq Value$.

构件 QoS 规格的作用在于描述等式(2) QoS_{in} , Res , QoS_{out} 中每个 QoS 属性值的约束定义, 给出构件对每个 QoS 属性的期望. 同时为使构件实例具有适应性, 通过定义 QoS 模式使得每种 QoS 属性具有多种可选择约束定义. 对于一个 QoS 属性, 通过 QoS 属性约束定义规定每个 QoS 的取值范围. 如图 3 所示, 对应于 Liquid 构件模型, QoS 规格定义如下.

定义 2. QoS 规格. 形式上, 一个 QoS 规格是一个 3 元组 $QoSProfile = (Component, ModeList, TransitionList)$, 其中.

(1) $Component$ 为 QoS 规格所描述的构件类型.

(2) $ModeList$ 为 QoS 模式列表: $ModeList = \{\langle Mode_i \rangle | i = 1, 2, \dots, n\}$, 其中, $Mode_i$ 为 QoS 模式,

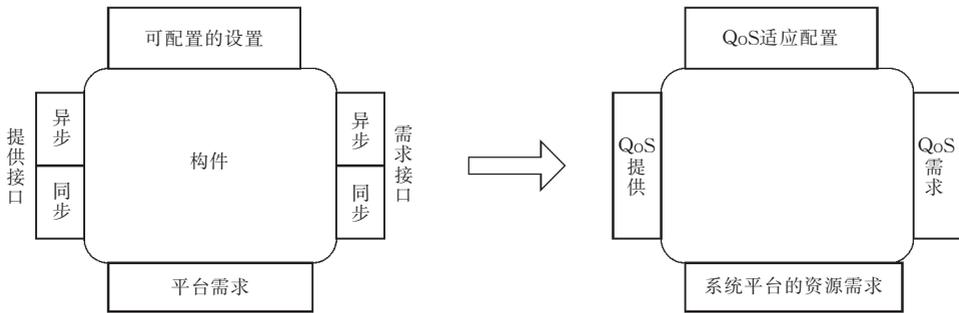


图 3 服务构件的 QoS 特征

描述了等式(2)中每个 QoS 属性值的约束定义; n 为 QoS 规格存在的模式数。

(3) $TransitionList$ 为 QoS 模式转换列表: $TransitionList = \{ Transition_i \mid i = 1, 2, \dots, m \}$, $Transition_i = \langle Mode_i, Mode_j, function_k \rangle$ 指定当构件实例 QoS 运行模式从 $Mode_i$ 变化到 $Mode_j$ 时, 构件框架调用的回调函数 (callback) $function_k$; m 为 QoS 规格存在的转换数。

以上构件规格的 $TransitionList$ 描述了图 3 构件的 QoS 适应配置, 而 $ModeList$ 中的 $Mode_i$ 描述了构件 QoS_{in} , Res , QoS_{out} 中每个 QoS 属性约束定义, QoS 模式定义如下。

定义 3. QoS 模式. 形式上, 一个 QoS 模式是一个三元组 $Mode = (ProvideList, UseList, ResList)$ 。

(1) $ProvideList = \{ Provide_i \mid i = 1, 2, \dots, n \}$ 为构件提供的 QoS 属性约束定义, $Provide_i$ 给出了构件实例 QoS_{out} 中某一 QoS 属性值的约束定义。

(2) $UseList = \{ Use_i \mid i = 1, 2, \dots, m \}$ 为构件需求的 QoS 属性约束定义, Use_i 定义了构件实例 QoS_{in} 中 QoS 属性值的约束。

(3) $ResList = \{ Res_i \mid i = 1, 2, \dots, h \}$, Res_i 则描述构件实例 Res 中 QoS 属性值的约束。

对于一个 QoS 属性, 通过 QoS 属性约束定义方式, 每个 QoS 模式定义了其不同的取值范围. 同一 QoS 规格中, 不同 QoS 模式所定义的 QoS 属性相同, 不同在于其 QoS 属性约束定义, 即取值范围不同。

QoS 模式中 Res_i 的 QoS 属性约束直接或者间接转换为资源占用量, 例如, CPU、网络占用带宽. 明确构件实例所需资源占用量, 不同于 $Provide_i$ 和 Use_i , Res_i 中的 QoS 属性约束的约束操作只能取 '=', Res_i 中的 $Value$ 代表此模式下构件实例所占用的资源量。

3.3 服务构件组合算法

3.3.1 构件选择基本算法

如图 4 所示, 在 Liquid 中, 构件服务组合问题可以描述为: ① 针对复合服务请求的功能配置图, 映射到相关构件实例; ② 根据服务的 QoS 请求 QoS_{req} 和系统资源状态, 选择构件实例的特定 QoS 模式, 使得 QoS_{req} 得到满足, 并且在相邻构件实例间等式 (3) 能够成立以及所有构件实例所需资源得到满足。

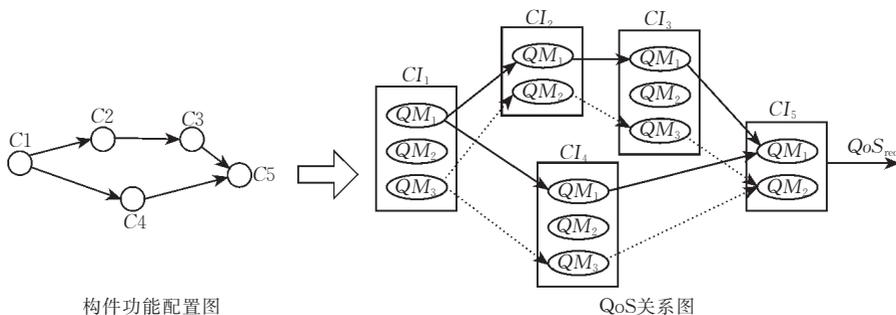


图 4 服务构件组合

对于步骤①, Liquid 根据构件功能配置图, 可以从构件注册表查找到相关已部署的构件实例. 而在步骤②中, 对于两个相邻构件实例 CI_i, CI_{i+1} 而言, 如果等式(3)需要成立, 同一 QoS 属性 q 需要分别满足两个构件实例所选 QoS 模式相应提供和需求的 QoS 属性约束定义: $CI_i.Mode_j.Provide_k$ 和

$CI_{i+1}.Mode_n.Use_m$ 。

定义 4. 满足关系. 构件实例 CI_i 所需 QoS_{in} 中的 QoS 属性被相邻构件提供的 QoS 属性所满足, 则 $\forall q, q \in CI_i.QoS_{in}, \exists CI_j, CI_j$ 为 CI_i 的相邻构件, 且 $q \in CI_j.QoS_{out}$, 如果 CI_i 和 CI_j 所选 QoS 模式分别为 $Mode_n$ 和 $Mode_m$, 在两模式中 q 定义的约束值

为 $Value_i$ 和 $Value_j$, 则

$$\begin{cases} \text{若 } Value_i \text{ 为单值, 则 } Value_i = Value_j \\ \text{若 } Value_i \text{ 为集合值, 则 } Value_i \cap Value_j \neq \emptyset \end{cases} \quad (4)$$

因此对于构件服务组合步骤②首先需要为构件实例选择 QoS 模式使得相邻构件实例间形成满足关系, 对于这一步, Liquid 根据构件功能配置图 FG 和组成构件 QoS 规格构造服务的 QoS 关系图 QG (QoS-relationship Graph). 如图 4 所示, 在 QG 中, 每个构件实例的 QoS 模式为其节点, 依照定义 4 的满足关系连接.

如果在 QG 中存在一个连通图, 此图所有尾节点和源节点分别为 FG 尾节点和源节点中的 QoS 模式, 则称此图中的 QoS 模式集合为服务的一种 QoS 配置 (QoS Configuration, QC) 选择, 如图 4, $\{CI_1.QM_1, CI_2.QM_1, CI_3.QM_1, CI_4.QM_1, CI_5.QM_1\}$ 和 $\{CI_1.QM_5, CI_2.QM_2, CI_3.QM_3, CI_4.QM_3, CI_5.QM_2\}$ 为此例的 QoS 配置选择. 本文采用反向深度优先遍历 FG 的方法, 在遍历过程中, 选择出构造服务的 QoS 配置集合, 算法如下:

算法 1. 选择服务的 QoS 配置.

输入: 构件功能配置图: FG

输出: 服务的可选 QoS 配置集合: QCSet

Begin

1. 将所有 FG 边反向, 设 FG 边反向后变为 TFG;
2. 查找出 TFG 的源节点集合, 设为 SOURCE 全局变量;
3. 查找出 TFG 的尾节点集合, 设为 SINK 全局变量;
4. for 每个节点 $u \in V[TFG]$ do $visited[u] \leftarrow \text{false}$;
//置该节点未被访问标志
5. QCSet $\leftarrow \emptyset$;
6. for 每个节点 $u \in SOURCE$ do
7. begin
8. QC_{tmp} $\leftarrow \emptyset$;
9. if $visited[u] = \text{false}$ then
 Construct_QC(u, QC_{tmp});
10. QCSet $\leftarrow QCSet \cup QC_{tmp}$;
11. end

End

Procedure Construct_QC(u, QC)

Begin

1. $visited[u] \leftarrow \text{ture}$;
2. for 每个 QoS 模式 $QM_i \in QoSProfile[u]$ do
3. Begin
4. $all_QoS_Satisfied \leftarrow \text{true}; QC_Item \leftarrow \emptyset$;
5. for 每个 QoS 属性 $q_i \in QM_i$ 需求合约
 and $all_QoS_Satisfied = \text{true}$ do
6. Begin
7. $q_Satisfied \leftarrow \text{false}$;

8. for 每个顶点 $v \in Adj[u]$ do
9. for 每个 QoS 模式 $QM_j \in QoSProfile[v]$ do
10. if q_i 被 QM_j 满足 then $QC_Item \leftarrow QC_Item \cup \{QM_j\}$,
 $q_Satisfied \leftarrow \text{true}$;
11. if $q_Satisfied = \text{false}$ then $all_QoS_Satisfied \leftarrow \text{false}$;
12. End
13. if $all_QoS_Satisfied = \text{true}$ then
14. if $u \in SOURCE$ then $QC \leftarrow QC \cup QC_Item$;
15. else for $Item \in QC$ do if $QM_i \in Item$ then
 $Item \leftarrow Item \cup QC_Item$;
16. else if $u \notin SOURCE$ then
17. for $Item \in QC$ do if $QM_i \in Item$ then $QC \leftarrow QC_Item$;
18. End
19. if $u \in SINK$ then return; else for 每个顶点
 $v \in Adj[u]$ do Construct_QC(v, QC);

End

由于深度遍历图的算法时间复杂度为 $O(V^2)$, V 为 FG 的节点数, 而对每一个节点构造满足关系, 算法运算复杂度为 $O(M^2)$, M 为服务构件所含 QoS 模式最大数, 因此, 以上算法的时间复杂度为 $O(V^2 \times M^2)$. 在选择出服务的所有 QoS 配置之后, Liquid 选择出满足服务的 QoS 请求 QoS_{req} 的 QoS 配置, 即 QoS 关系图尾节点模式所提供的 QoS 属性约束和 QoS_{req} 形成满足关系. 这些被选择出的 QoS 配置形成服务的可选 QoS 配置集合. 然后, Liquid 计算其中每种 QoS 配置所需要的资源占用量, 为其组成构件 QoS 模式 ResList 中所有资源之和:

$$Res_{QC} = \left[\sum_{i=1}^N r_1, \sum_{i=1}^N r_2, \dots, \sum_{i=1}^N r_k \right] \quad (5)$$

式(5)中, N 为服务的组成构件数, k 为所涉及的所有资源种类. 例如图 4, 假设此服务只涉及一种资源, 且 $\{CI_1.QM_1, CI_2.QM_1, CI_3.QM_1, CI_4.QM_1, CI_5.QM_1\}$ 为可选 QoS 配置, 则它的资源总量为 $Res_{CI_1.QM_1} + Res_{CI_2.QM_1} + Res_{CI_3.QM_1} + Res_{CI_4.QM_1} + Res_{CI_5.QM_1}$. 在 Liquid 中每种资源都有约束条件, 定义如下.

定义 5. 资源约束. 一种资源约束是一个 4 元组 $Resource = (Name, Capability, Available, t)$, 其中 $Name$ 为资源名称, 代表一种资源类型; $Capability$ 描述了系统所有这种资源的资源总量, $Capability \in \mathbb{R}$; 而 $Available$ 给出了系统运行在 t 时刻, 这种资源的有效值, $Available \in \mathbb{R}$.

每种资源由相应 Liquid 内核资源管理元服务进行管理, 如 CPU 调度服务等. 可选 QoS 配置只有当式(5)中 $\sum_{i=1}^N r_j \leq Available_j$ ($1 \leq j \leq k$) 时, 此 QoS 配置称为可行配置 $QC_{Available}$. 当 Liquid 接纳一个复合服务请求时, 将存在以下两种情况:

- (1) 由算法 1 得到的服务可行 QoS 配置存在多

个可行配置. 此时需要从中挑选一个做为服务的实际 QoS 配置. 在构件选择基本算法中, 挑选方法是计算每种可行 QoS 配置的权重, 计算如下

$$Weight_{QC} = \sum_{i=1}^k (W_i \times \sum_{j=1}^N r_i) \quad (6)$$

其中 N 为服务的组成构件数, k 为所涉及的所有资源种类, $W_i \in [0, 1]$, $\sum_{i=1}^k W_i = 1$, W_i 是 Liquid 为 r_i 资源设定的权重固定值, 用来代表一种资源被占用的频繁度, W_i 越高, 代表此资源越不容易被申请.

容易看出等式(6)计算得到的 $Weight_{QC}$ 是 QoS 配置所占用的平均资源值, 为减少服务所占用资源量, Liquid 将选择 $Weight_{QC}$ 最小的一个 QoS 配置为服务的实际运行 QoS 配置, 并根据此 QoS 配置中的 QoS 模式设置相应构件实例的模式.

(2)由算法 1 得到的可选 QoS 配置不存在可行配置. 此时, 在构件选择基本算法中, Liquid 将拒绝复合服务请求.

3.3.2 构件选择优化算法

以上构件选择基本算法存在两个方面的缺陷: 首先, 在构件选择基本算法中, 等式(6)的 W_i 是 Liquid 为 r_i 资源设定的, 代表其被占用频繁度的固定值, 不能够反映资源动态占用情况, 对此进行优化成构件选择启发算法, 采用启发式方法动态设定 W_j 为

$$W_j = \left(\frac{\sum_{i=1}^N r_j}{Avaivable_j} \right)_{\text{average}} / \sum_{j=1}^K \left(\frac{\sum_{i=1}^N r_j}{Avaivable_j} \right)_{\text{average}} \quad (7)$$

其中, $\left(\frac{\sum_{i=1}^N r_j}{Avaivable_j} \right)_{\text{average}}$ 代表系统一段运行时间内,

第 j 种资源被占用率的平均值, 而通过等式(7)的计算, W_j 能够动态反映此种资源被占用频繁度.

其次, 在构件选择基本算法中, 当算法 1 得到的可选 QoS 配置不存在可行配置, 此时, Liquid 将拒绝复合服务请求. 然而, 此时可以通过和 Liquid 系统中已有服务 $\{Service_j | j=1, 2, \dots, n\}$ 协商资源的方法, 尝试是否能够接收复合服务请求, 构件选择协商算法正是在构件选择启发算法上对此进行优化.

假设集合 $QCSet_{\text{Available}} = \{\langle QC_1, Res_{QC_1}, Weight_{QC_1} \rangle, \dots, \langle QC_n, Res_{QC_n}, Weight_{QC_n} \rangle\}$ 为每个服务可行 QoS 配置说明集合. 协商资源的目的是: 在包括新申请服务 $Service_{\text{new}}$ 在内的系统所有服务 $\{Service_j | j=1, 2, \dots, n+1, n$ 为系统中已有服务 $\}$ 在 $QCSet_{\text{Available}}$ 中选择出一个 QoS 配置 QC_{ij} , 使得 $Res_{QC_{ij}}$ 能够被满足, 且

$\sum_{j=1}^{n+1} Weight_{QC_{ij}}$ 值最小. 为此服务间关于资源协商可以采用整数规划方法(Integer Programming)^[20]: 规划的

限制条件是 $\sum_{i=1}^{n+1} (Res_{QC_i} \cdot \sum_{j=1}^N r_j) \leq Capability_j, (1 \leq j \leq k,$

k 为资源种类数), 目标函数则是 $\min(\sum_{j=1}^{n+1} Weight_{QC_{ij}})$.

如果资源协商成功则 Liquid 接受新的复合服务请求, 否则拒绝. 整数规划为 NP 难问题, n 个服务协商资源的计算时间复杂度为 $O(m^n)$, m 为 n 个服务所含最大可行配置数. 考虑到时间复杂度, 因此, 采用资源协商方法判断是否接受新服务的准入, 只适合于 n 较小的情况.

4 实验数据分析

我们定义 QoS 成功率(QoS Success Rate, QSR)为衡量构件服务组合算法性能的标准, 从第 3 节分析可知, 一个复合服务组合请求被申请成功, 需要满足: (1) 系统存在其构件功能配置图中所需构件实例; (2) 服务的 QoS 请求被满足; (3) 所涉及的构件实例的资源需求能够满足, 且相邻构件间形成 QoS 满足关系. 假设在所有 $Total_{\text{request}}$ 个服务申请请求中, $Success_{\text{request}}$ 个服务申请成功, 则 QoS 成功率定义为 $QSR = \frac{Success_{\text{request}}}{Total_{\text{request}}}$.

我们设计了模拟实验方案, 统计比较采用不同构件选择算法的 QSR, 如图 5(a)所示, 模拟实验环境由三台具有 Liquid 在 PC 上开发的实验版本的 PC 联网组成 $\{P_1, P_2, P_3\}$, PC 主频为 1.4GHz. 由 3.3 节可知, 影响 QSR 的因素为资源约束和服务的资源需求. 假设 $random(n, m)$ 产生 n 到 m 间的随机数, 实验中, 我们每隔 $random(3, 9)$ 秒反复随机地从 P_1 产生一个复合服务请求. 被模拟的三种复合服务的构件功能配置图及服务构件部署如图 5(b)所示, 且经过算法 1 计算得到的可行 QoS 配置资源需求如表 1 所示, 目前, Liquid 只实现 CPU 的资源预留管理, 因此在此, r_1, r_2, r_3 分别代表 P_1, P_2, P_3 的 CPU 资源.

实验每隔 3 秒统计系统的 QSR. 设定每个申请成功服务的持续时间为 $random(60, 180)$ s, 当服务完成后, 其占用资源将被相应 Liquid 资源管理元服务回收. 实验中 $Capability_{r_1}, Capability_{r_2}$ 和 $Capability_{r_3}$ 大小设定为 100%. 在构件选择基本算法中, r_1, r_2 和 r_3 三种资源的权重分别设定为 $W_1 = 0.5, W_2 = 0.3, W_3 = 0.2$.

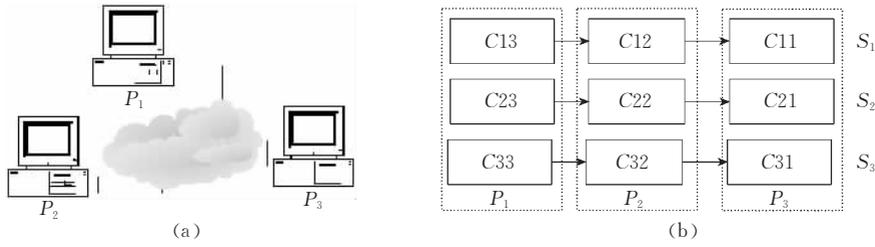


图 5 模拟实验环境

表 1 复合服务可行 QoS 配置的资源需求

服务	可行 QoS 配置	$\sum r_1(\%)$	$\sum r_2(\%)$	$\sum r_3(\%)$
S ₁	QC ₁	20	10	9
	QC ₂	9	27	10
	QC ₃	8	12	32
S ₂	QC ₁	8	25	15
	QC ₂	16	8	20
S ₃	QC ₁	10	11	19
	QC ₂	17	10	10
	QC ₃	5	16	28

从以上资源约束参数和表 1 可行 QoS 配置的资源需求易知,同时在系统中的服务数目不会超过 10,因此,我们采用穷举方法计算协商算法中的整数规划最优解.图 6 给出了实验结果,即 QSR 随时间的变化情况.可见,经过启发式调节资源的权重值,启发式算法的 QSR 要高于构件基本选择算法.而进一步通过资源协商方法,QSR 能够再次提高.模拟研究的结果表明,通过启发式调节资源权重和通过

服务资源协商方法,QSR 能够提高.

在实验过程中,我们记录每个被 Liquid 接收的复合服务请求所选可行 QoS 配置.表 2 给出了在每种构件选择算法中,复合服务所选可行 QoS 配置所占比例.实验结果显示了三种算法对动态资源变化的适应性,从中可知,相对构件选择基本算法,启发和协商算法具有一定的适应性.虽然协商算法在 QSR 上优于其它两种方法.

表 2 复合服务所选可行 QoS 配置比例

服务	可行 QoS 配置	基本算法(%)	启发算法(%)	协商算法(%)
S ₁	QC ₁	0	65.19	40.88
	QC ₂	0	21.50	25.44
	QC ₃	100	13.31	33.68
S ₂	QC ₁	0	36.52	41.34
	QC ₂	100	63.48	58.66
S ₃	QC ₁	100	15.82	30.20
	QC ₂	0	70.44	54.10
	QC ₃	0	13.74	15.70

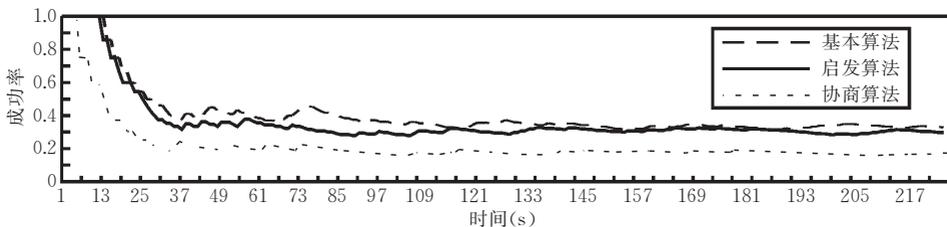


图 6 不同构件选择算法的 QSR 比较

5 结束语

本文针对 Liquid 操作系统中服务构件的特点,给出了一种基于 QoS 的服务构件组合方法,其主要特征有:(1)定义了能够适应于动态环境的服务模型和 QoS 模型.本文所提服务和 QoS 模型具有一定通用性,此服务构件组合方法可以运用到其它面向服务的架构中;(2)提出了三种基于 QoS 的构件选择算法,并通过实验数据分析,构件选择启发和协商算法能够改进构件选择基本算法对动态资源状态的适应性,但是,由于协商算法运用了整数规划方法,具有较高的计算时间复杂度,因此限制其应用范围.

怎样在 Liquid 中动态结合这三种算法,在提高服务构件组合性能的同时,降低时间复杂度是我们未来研究的方向之一.

参 考 文 献

- 1 Satyanarayanan M. . Pervasive computing: Vision and challenges. IEEE Personal Communications, 2001, 8(4): 10~17
- 2 Ma Bo, Zhang Yi, Shi Xing-Guo. Liquid meta-services: A component-based operating system layer for pervasive computing. In: Proceedings of the 1st International Conference on Embedded Software and System(ICESS2004), Zhejiang, China, 2004, 317~322
- 3 Liao Yuan, Li Ming-Shu. A QoS-aware component-based middleware for pervasive computing. In: Proceedings of the 1st In-

- ternational Conference on Embedded Software and System(IC-ESS2004), Zhejiang, China, 2004, 158~167
- 4 Zeng Liang-Zhao, Benatallah Boualem, Ngu Anne H. H., Dumas M., Kalagnanam J., Chang H.. QoS-aware middleware for Web services composition. *IEEE Transactions on Software Engineering*, 2004, 30(5): 311~327
 - 5 Raman B., Katz R. H.. Load balancing and stability issues in algorithms for service composition. In: *Proceedings of IEEE INFOCOM 2003*, San Francisco, CA, 2003, 1477~1487
 - 6 Liu C., Yang L., Foster I., Angulo D.. Design and evaluation of a resource selection framework for grid applications. In: *Proceedings of the Symposium on High Performance Distributed Computing(HPDC-11)*, Edinburgh, Scotland, 2002, 63
 - 7 Xu Dong-Yan. An integrated and QoS-aware framework for multimedia service management[Ph. D. dissertation]. Department of Computer Science, University of Illinois at Urbana-Champaign, 2001
 - 8 Aurrecochea C., Campbell A. T., Hauw L.. A survey of QoS architectures. *Multimedia Systems*, 1998, 6(3): 138~151
 - 9 Nahrstedt K., Xu D., Wichadakul D., Li B.. QoS-aware middleware for ubiquitous and heterogeneous environments. *IEEE Communication Magazine*, 2001, 39(11): 2~10
 - 10 Román M., Hess C. K., Cerqueira R., Ranganathan A., Campbell R. H., Nahrstedt K.. Gaia: A middleware infrastructure to enable active spaces. *IEEE Pervasive Computing*, 2002, 1(4): 74~83
 - 11 Nahrstedt K., Wichadakul D., Xu D.. Distributed QoS compilation and runtime instantiation. In: *Proceedings of the 8th IEEE/IFIP International Workshop on Quality of Service*, Pittsburgh, USA, 2000, 198~207
 - 12 Schmidt D., Levine D., Cleeland C.. Architectures and patterns for high-performance, real-time corba object request brokers. In: *Marvin Zelkowitz ed.. Advances In Computers*, Academic Press, 1999, 48: 1~118
 - 13 Zinky J., Bakken D., Schantz R.. Architecture support for quality of service for corba objects. *Theory and Practice of Object Systems*, 1997, 3(1): 1~20
 - 14 Patricia Gomes Soares Florissi. QoSME: QoS management environment[Ph. D. dissertation]. Columbia University, 1996
 - 15 Li B., Nahrstedt K.. A control-based middleware framework for quality of service adaptations. *IEEE Journal of Selected Areas in Communications, Special Issue on Service Enabling Platforms*, 1999, 17(9): 1632~1650
 - 16 Benatallah B., Casati F.. *Distributed and Parallel Database, Special Issue on Web Services*. Kluwer Academic Publishers, 2002
 - 17 Ankolekar A., Burstein M., Hobbs J. R., Lassila O., McDermott D., Martin D., McIlraith S. A., Narayanan S., Palolucci M., Payne T., Sycara K.. DAML-S: Web service description for the semantic Web. In: *Proceedings of the 1st International Semantic Web Conference(ISWC 02)*, Sardinia, Italy, 2002, 348~363
 - 18 Gu X., Nahrstedt K.. A scalable QoS-aware service aggregation model for peer-to-peer computing grids. In: *Proceedings of the 11th IEEE International Symposium, High Performance Distributed Computing (HPDC)*, 2002, 73~82
 - 19 Xu D., Nahrstedt K.. Finding service paths in a media service proxy network. In: *Proceedings of SPIE/ACM Multimedia Computing and Networking Conference(MMCN)*, California, USA, 2002, 171~185
 - 20 Karloff H.. *Linear Programming*. Cambridge, MA, USA: Birkhauser Boston Inc., 1991



LIAO Yuan, born in 1975, Ph. D. candidate. His research interests include real-time systems, component-based software engineering and service-oriented architecture.

TANG Lei, born in 1980, Ph. D. candidate. Her research interests include real-time systems and software process.

LI Ming-Shu, born in 1966, Ph. D., professor. His research interests include software requirement engineering and real-time systems.

Background

This work is supported by programs as follows:

The National High Technology Research and Development Program (863 Program) under grant No. 2002AA1Z2302 and 2004AA1Z2050, which aimed to design and implement a new component-based embedded operating system for pervasive computing-Liquid. It provides customers with flexible and transparent component-based services, pervasive computing support, QoS-aware mechanism and system development environment that includes system adjustment and testing tools. To this day, six papers of the project have been released by the research group. Simultaneously, the prototype of Liquid kernel and its various services have been developing. The result of this paper is used to compose those service components based on their non-functional properties, specially, QoS.

The National Natural Science Foundation of China under grant No. 60273026, that is focused on user-driven requirements acquisition and cooperation mechanism in distributed environments based on Internet/Web architecture. The main objective of the project is to study on user-driven requirements acquisition framework and multi-level user cooperation mechanism. In order to present result of the research, the group has been developing a supporting tool prototype and demonstrates its use with the development of software quality management system. How to design the tool to realize user-driven non-functional requirements (e. g. software quality) acquisition in distributed environments is a challenge. The method of QoS-aware service components composition presented in this paper is used in the tool prototype.