第 29 卷 第 7 期
2006 年 7 月
计 算 机 学 报
CHINESE JOURNAL OF COMPUTERS
Vol. 29 No. 7
July 2006

基于用户视角的组合 Web 服务响应时间优化

王亚沙 赵俊峰 谢 冰
(北京大学信息科学技术学院软件工程研究所 北京 100871)

摘 要 针对响应时间,通过选择、调用合适的成员 Web 服务,对组合 Web 服务的 QoS 进行优化. 该优化方法基于用户的视角,将组合 Web 服务运行时用户一次等待周期中需要执行的所有任务作为一个整体进行调度;同时根据组合 Web 服务流程的不同环节中用户对等待时间的不同敏感度,设置调度参数,降低关键环节超时的风险. 实验表明,此方法可以有效地优化组合 Web 服务的响应时间,提高用户的满意度和 QoS.

关键词 Web 服务;服务质量;响应时间;组合 Web 服务;服务选择
中图法分类号 TP311

Response Time Optimization of Composite Web Services from the Perspective of Users

WANG Ya-Sha ZHAO Jun-Feng XIE Bing
(*Institute of Software Engineering, School of Electronics Engineering and Computer Science, Peking University, Beijing* 100871)

Abstract Web services technology enables integration of heterogeneous software by encapsulating them into Web services, and building new applications through this way. In addition to the necessary functions, the composite Web services should provide the quality of service (QoS) needed by the users. Response time is one of the important attributes of QoS. The response time of composite Web services is hard to compute because of the loop, conditional selection and concurrent structures in its execution processes. Aiming at composite Web services response time, an optimization method by selecting and invoking appropriate component Web services is proposed in this paper. From user's perspectives, this optimization method considers all the tasks' response time to be executed in one user-waiting-period as a whole when selecting Web services; and according to the different user-sensitivity-levels in the execution process of a composite service, this optimization method adopts different Web services selection schemes to reduce the risk of overtime for critical points in the process. Experiment results show this method effectively optimizes the response time of composite Web services, improves users' satisfaction and services' QoS.

Keywords Web services; quality of service; response time; composite Web services; service selection

1 引 言

作为一种新型的计算范型,Web 服务技术已经成为推动软件工程格局演化的强大力量. Internet 上异构的应用被封装成 Web 服务后,可以很容易地通过 UDDI, WSDL, SOAP 等协议实现交互、协作和集成,快速搭建出具有丰富功能的新应用. 在实际

收稿日期:2006-02-17;修改稿收到日期:2006-04-21. 本课题得到国家"八六三"高技术研究发展计划项目基金(2005AA113030)、国家自然科学基金(60473059)资助. 王亚沙,男,1975 年生,博士,讲师,主要研究方向为软件复用、Web 服务技术、软件过程管理等. 赵俊峰,女,1974 年生,博士,讲师,CCF 高级会员,主要研究方向为软件复用技术、构件技术、大型分布式构件库系统等. E-mail: zhaojf@sei.pku.edu.cn. 谢 冰,男,1970 年生,博士,副教授,CCF 高级会员,主要研究方向为软件工程、形式化方法、分布式系统.

应用中，我们要求通过组合多个 Web 服务形成的组合服务提供用户所需的功能，同时也具备用户可以接受的服务质量(QoS). 但是，因为组合 Web 服务通常需要通过网络调用多个由不同供应者提供、分布在网络的不同区域、运行在异构软硬件平台上的成员 Web 服务，所以对其 QoS 的预测和管理都较为复杂. 随着 Web 服务应用的逐渐深入，针对组合 Web 服务 QoS 的研究已经成为当前业界和学术界共同关注的热点.

Web 服务的 QoS 属性包括多个方面，研究者普遍关注的主要包括价格、响应时间、成功执行率、可获得性等[1,2]. 本文并不打算对 QoS 属性进行综合、全面地探讨，而是将注意力聚焦在响应时间这一重要属性上，研究组合 Web 服务响应时间的优化问题. 这主要出于两个方面的考虑：首先，响应时间是设计和管理业务流程时需要考虑的关键问题和普遍问题[3,4]；另外，由于组合 Web 服务的执行流程中可能包含循环、并发、条件选择等结构，使得其响应时间的计算较之价格、成功执行率、可获得性等属性更为复杂，是 QoS 研究中的难点. 本文的组合 Web 服务响应时间优化方法经过稍微修改同样可以应用到其它 QoS 属性中.

本文强调从用户的视角对组合 Web 服务的响应时间进行优化. 根据 ITU-T 对 QoS 的定义[5]，QoS 是一个综合指标，用于衡量使用一个服务的满意程度，描述关于一个服务的某些性能特点. 从这个定义可以看到，QoS 的实质是用户的满意程度. 专门就响应时间而言，要使组合 Web 服务的 QoS 满足用户的要求，就是要使用户的等待时间尽量控制在用户可以接受的范围之内. 而用户在使用一个组合 Web 服务时，其等待时间通常取决于多个成员 Web 服务的响应时间(包括网络传输时间和 Web 服务的执行时间). 本文对组合服务业务流程中导致用户等待并显著影响用户满意度的流程片断进行了标识. 这些流程片断被称为时间临界区(Time Critical Section，TCS，见第 3 节). 因为用户感觉到的等待时间是整个 TCS 的执行时间，所以本文将 TCS 中调用的所有 Web 服务作为一个整体进行调度，使 TCS 的整体执行时间满足用户的要求. 另外，本文还为各 TCS 赋予了一个被称作时间敏感度级别的属性，用于标识用户在组合服务业务流程的不同环节对响应时间的敏感程度，即级别越高意味着当服务的响应时间超过用户允许的范围时用户对服务的满意程度随时间下降越快，QoS 下降也就越快. 本文通过服务运行时对成员 Web 服务的选择，使敏感度较高的关键环节的等待时间以更大的概率满足用户的需求.

本文提出的“基于用户视角的组合 Web 服务响应时间优化方法”分为服务运行前和运行时两个步骤实施：

(1)设计时，从用户的视角出发，标识组合 Web 服务业务流程中的 TCS，设定 TCS 的最大允许时间、时间敏感度级别等属性，并在组合服务规划时，为了简化运行时对 Web 服务的选择，根据相关的规则对 TCS 进行必要的分解.

(2)在运行时，由一个公共的服务代理(service broker)，根据 Web 服务的实时响应时间以及 TCS 的属性，为组合 Web 服务业务流程中的各个任务选择并绑定合适的成员 Web 服务，以优化 TCS 的执行时间，从而提高组合 Web 服务的 QoS. 本文中的服务代理已经在北京大学青鸟构件库管理系统 JB-CLMS[6]中实现，关于该服务代理如何获得各备选 Web 服务实时响应时间的方法见文献[7]，本文不作讨论.

本文第 2 节给出本文中组合 Web 服务的规约方法；第 3 节介绍 TCS 的概念及其在组合服务中的应用；第 4 节描述 TCS 的分解方法；第 5 节介绍组合运行时对成员 Web 服务的选择；第 6 节介绍实验并分析实验结果；第 7 节介绍其他研究者的相关工作并讨论其与本文工作的区别；第 8 节总结全文.

2 组合 Web 服务的规约

本文将组合 Web 服务规约为一组由控制流和数据流联系在一起的抽象任务. 抽象任务简称为任务，是对具有特定功能的 Web 服务的抽象，可以用一个语义上的、无二义性的描述来表达. 实现某任务功能的具体 Web 服务称为该任务的实例服务. 组合服务运行时，任务的响应时间是其绑定的实例服务的响应时间. 某任务的所有实例服务构成该任务的实例服务集. 同一个实例服务集中的各实例服务功能相同，但 QoS 可能不同. 本文对组合服务响应时间的优化是通过在运行时为各任务选择并调用具有合适响应时间的实例服务实现的.

本文采用一种与文献[8]类似的状态图表达组合 Web 服务的业务流程，与文献[8]不同的是，我们的图形中加入了一个菱形符号，连接流程中不同的条件分支，使得条件分支结构的表达更加明确. 除了

这个新加入的菱形符号，图形中还包括状态和迁移两种元素. 其中迁移用来表达任务之间的数据流和控制流，并且可以在其上标注事件、条件和操作等信息. 状态包括原子状态和复合状态两种. 原子状态上标注了其代表的任务的标识，进入一个原子状态，表示其标注的任务与一个具体的 Web 服务绑定，并且该 Web 服务被调用. 复合状态又包括 OR 状态和 AND 状态两种. 一个 OR 状态包括一个单一的区域，而 AND 状态包括多个用虚线分割的区域，各区域并发执行. 由此可见，OR 状态可以对多个状态进行分组，用于模块化的用途；而 AND 状态则用于表达流程中的并发性.

作为示例，图 1 演示了某网上书店的组合 Web 服务的流程，为了便于讨论，图中对实际流程进行了简化. 在这个例子中，首先查询符合条件的书籍列表，接着将书籍列表按照用户的喜好排序并输出，此时用户既可以选择感兴趣的书籍加入购物车（一个临时存储区，存放用户本次购书事务中希望购买的书籍信息），也可以重新发起另一次查询. 用户将所有感兴趣的书籍全部放入购物车后启动结算过程. 结算过程中，有两条线索并发执行. 其中一条线索首先确定用户选购书籍的书源，然后根据书源所在地和用户要求的送货地址，计算配送时间；另一条线索则根据用户的信用等级和书籍的销售情况分别计算每本书籍的折扣和整个订单的总金额. 当并发的两条线索全部执行完后，将生成一份包含本次购买书籍清单、书籍实际购买价格、订单总金额、送货时间等信息的订单明细，并提交给用户确认. 此时用户可以选择修改订单（通过增加或删除购物车中的书籍，并重新结算实现）、取消订单或者确认订单. 订单确认后将存档，然后组合服务将向书籍配送点发送配送通知，从而结束一次购书的流程.

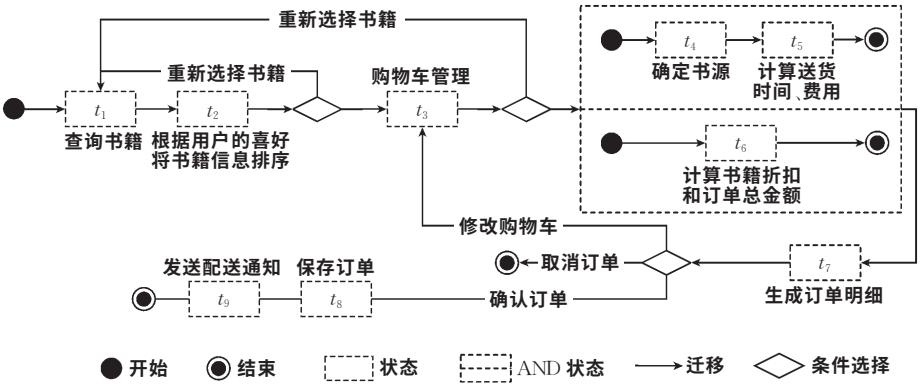


图 1　网上书店组合服务的状态图

3　组合 Web 服务的流程中时间临界区的标识

时间临界区（Time Critical Section，TCS）是组合服务流程中的一个片断，这个片断需满足以下两个条件：(1)该流程片断执行时用户将经历一个完整的等待周期. 即从进入时间临界区起，用户开始等待组合服务的某个输出或反馈. 时间临界区执行完毕后，用户获得输出或反馈，结束等待. (2)当该流程片断的执行时间超过一个门限值后，组合服务的 QoS 将明显下降，或者说这个流程片断对执行时间较为敏感.

时间临界区建立起了服务的外部视角（用户看待服务的视角）与内部细节（服务执行的内部细节）之间的对应关系. 从用户的角度来看，他们并不关心服务的等待时间是由哪个具体的任务或者 Web 服务导致的，他们关心的是从向组合服务提供输入参数后，需要多长时间才能得到反馈；而从服务的设计和管理者的角度来看，他们有必要了解导致用户某一次等待的是业务流程中的哪些任务，这些任务应该调用哪些实例服务才能将用户的等待时间控制在可以接受的范围以内. 通过 TCS 可以标识构成用户某一次等待的流程片断，并将其作为一个整体进行全局的调度，更加有针对性地提高组合服务的 QoS.

另外，只有对执行时间敏感的流程片断才被标识为 TCS. TCS 对时间的敏感程度通过最大允许时间和时间敏感度级别这两个属性来刻画. 最大允许时间是一个门限时间，记作 $t(tcs)$，在组合服务的某次运行中，如果 TCS 的执行时间超过了这一门限，QoS 就会明显下降. 时间敏感度级别是一个大于零的整数，记作 $l(tcs)$，用于刻画 TCS 对时间的敏感程度，数值越大则表示 TCS 的时间敏感度越高. 当

TCS 的执行时间超过最大允许时间时，组合服务的 QoS 随时间下降的速度也就越快. 最大允许时间和时间敏感度级别可以由领域专家根据其组合服务所属领域的领域知识在组合服务设计阶段设置.

图 2 给出了在网上书店组合服务中 TCS 的示例. 在这个例子中，我们共识别了 3 个 TCS：tcs_1 包括了从用户输入书籍查询条件到得到查询结果的这段时间内组合服务执行的流程片断；tcs_2 则是从用户选中预购书籍到这些书籍被成功加入购物车的过程中组合服务执行的流程片断；tcs_3 则是从用户开始结算到结算完成并输出订单明细组合服务执行的流程片断. 这 3 个 TCS 都分别对应一次完整的用户等待周期. 需要说明的是，虽然延长 t_8 或 t_9 的响应时间会推迟书籍配送员（网上书店的另一类用户）收到书籍配送通知的时间，但是我们并没有把涉及 t_8 和 t_9 的流程片断标识为第 4 个 TCS. 这源于两个方面的原因：首先，因为配送员与组合服务的交互是异步的，也就是说，配送员并不会一直坐在电脑旁等待某个特定配送通知的出现；其次，在网上书店的业务中，配送通知到达这一事件本身对时间并不敏感，配送通知送达的时间提早或者推迟一个小时，都不会对业务产生明显的影响. 因此，虽然在 t_9 之后存在用户与服务的交互，但是包括 t_8 和 t_9 的流程片断对时间不敏感，故此不将其标识为 TCS.

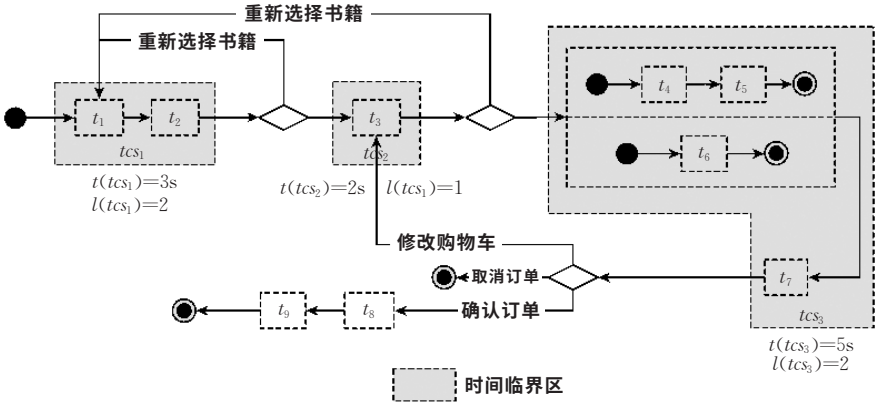


图 2 网上书店组合服务中的时间临界区示例

另外，我们还分别设置 tcs_1，tcs_2 和 tcs_3 的最大允许时间为 3s，2s 和 5s，设置其时间敏感度级别为 2，1 和 2. 这里我们为 tcs_1 和 tcs_3 设置了较 tcs_2 更高的时间敏感度级别. 下面以 tcs_1 和 tcs_2 为例来讨论不同时间敏感度级别的含义. 按照网上书店的流程，只有用户选中的书籍全部成功加入到购物车以后，才可以进入到结算的环节. 如果用户选择完预购的书籍后，希望马上结算，而此时 tcs_2 还没有执行完，那么用户将被迫等待，如果等待时间超过最大允许时间，用户的满意度将下降，组合服务的 QoS 也随之降低. 如此看来，tcs_2 对时间敏感. 但是，进一步分析后我们发现，用户并非每次执行完 tcs_2 都会选择结算，很多情况下用户会重新输入书籍查询条件，继续搜索下一本书籍，这时即使 tcs_2 的执行时间延长用户也不会觉察到，对服务的满意度自然也不会下降. 而 tcs_1 则不同，用户每次发起对书籍的查询操作都会等待 tcs_1 执行完成后输出的查询结果，所以 tcs_1 执行时间延长，总是会影响到用户对服务的满意度. 另外，我们的网上书店会在用户选择结算时，首先列出用户本次购买的书籍清单，所以用户在等待 tcs_2 完成的这段时间内，可以选择先核查订单中的书籍；作为对比，用户在等待查询到的书籍列表时，得不到其它任何反馈，除了等待别无选择，从而更容易对等待发出抱怨. 由此可见，当执行时间超过最大允许时间时，tcs_1 将比 tcs_2 导致更快的 QoS 下降速度，也就是说 tcs_1 比 tcs_2 对时间更加敏感，所以应该为 tcs_1 设置更高的时间敏感度级别.

4 TCS 的分解

TCS 的执行时间是由其中各任务的响应时间造成的. 但是因为 TCS 的流程中可能包含循环、并发、条件选择等结构，所以 TCS 的执行时间计算较为复杂. 为了在组合服务运行时简化对实例服务的选择和调度，在服务运行前的规划阶段会首先将 TCS 分解为一组更小的流程片断——子 TCS（sub-TCS），分解后的 sub-TCS 须满足下面两个条件：

（1）这组子 TCS 是对 TCS 的划分（互不重合，并且覆盖整个 TCS）；

（2）各 sub-TCS 的执行时间是其中包含的所有任务响应时间之和.

TCS 的分解包括 3 个步骤.

(1)流程“解循环”操作.

对于 TCS 流程中的循环结构，首先利用文献[9]中的方法进行“解循环”操作.“解循环”的方法十分简单，即通过统计组合服务执行的历史数据，计算循环的平均执行次数 n，在后续的计算中，循环体内各任务的响应时间等于其执行一次响应时间的 n 倍. 在下面的讨论中，认为 TCS 的流程图都是无环的.

(2)标定 TCS 中的 sub-TCS.

在这一步骤中，TCS 将被划分为一组 sub-TCS，每个 sub-TCS 由顺序执行的任务组成，从而使 sub-TCS 的执行时间等于其中各任务响应时间之和. 该步骤的算法如图 3 所示，此算法通过递归地调用函数 TCSComposition 完成 sub-TCS 的标定. 图 4 是一个包含选择和并发流程的 TCS 的 sub-TCS 标定结果，其中任务上方的数字是该任务的平均响应时间(平均响应时间的含义见下文).

```
TCS ← The TCS in which sub-TCS will be identified
SubTCSIdentification(TCS, FALSE);
function SubTCSIdentification (in ProcSec, Out
                                SubTCSIdentified )
//ProcSec 需要标识 sub-TCS 的流程片断
//SubTCSIdentified 是 boolean 型变量，如果 SubTCSIdenti-
//cation 在本次调用中标记了新的 sub-TCS，其值为 TRUE，
//否则为 FALSE
{ BEGINNODE ← begin node of the process in ProSect;
  ENDNODE ← end node of the process in ProSect;
  NODE ← BEGINNODE;
  SubTCSIdentified ← FALSE;
  while (NODE != ENDNODE ){
     NODE ← next node of the process in ProSect;
     switch NODE is type of do{
        case conditional node:
           for(each conditional branch){
              SubTCSIdentified = TRUE;
              NewProcSec ← this conditional branch;
              SubTCSIdentification(NewProcSec, FLAG);
              if( !FLAG) mark NewProcSec as a sub-TCS;}
        case fork node:
           for(each concurrent thread){
              SubTCSIdentified = TRUE;
              NewProcSec ← this thread;
              SubTCSIdentification(NewProcSec, FLAG);
              if( !FLAG) mark NewProcSec as a sub-TCS;}
     }//switch end
  }//while end
}//function end
```

图 3　sub-TCS 标定算法

(3)为各 sub-TCS 分配最大允许时间和时间敏感度级别.

sub-TCS 时间敏感度级别可以简单地设置为其所属 TCS 的时间敏感度级别，而最大允许时间的设定相对复杂一些，包括 4 个步骤.

(1)向服务代理查询 TCS 中所有任务的平均响应时间. 任务 t_i 的平均响应时间记作 $\bar{r}(t_i)$，其计算公式如下：

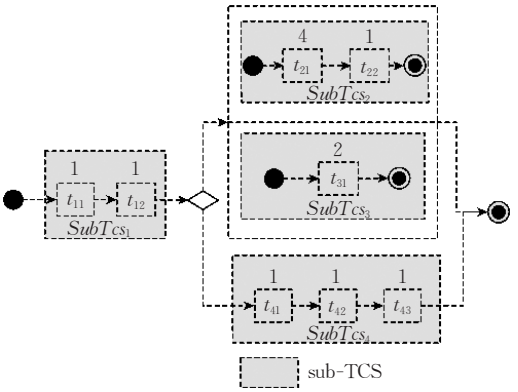


图 4　sub-TCS 标定的实例

$$\bar{r}(t_i)=\frac{\sum_{j=1}^{n}\bar{r}(s_{ij})}{n} \tag{1}$$

其中，s_{ij} 是 t_i 的实例服务，$\bar{r}(s_{ij})$ 是 s_{ij} 的平均响应时间(即一段时间内 s_{ij} 响应时间的平均值)，n 是 t_i 所有实例服务的个数.

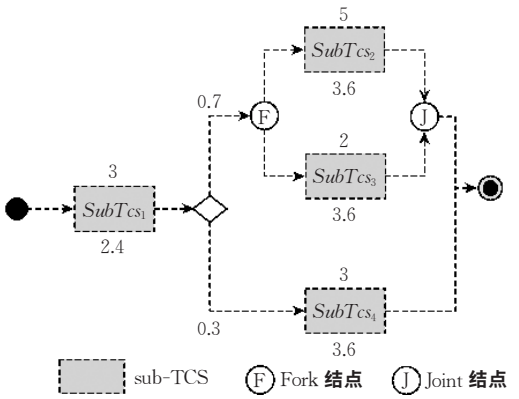
(2)计算各 sub-TCS 的平均执行时间. sub-TCS $SubTCS_k$ 的平均执行时间记作 $\bar{e}(SubTcs_k)$，其计算公式如下：

$$\bar{e}(SubTcs_k)=\sum_{l=1}^{m}\bar{r}(t_{kl}) \tag{2}$$

其中，$t_{k1}, t_{k2}, \cdots, t_{km}$ 是 $SubTcs_k$ 中的所有任务.

(3)根据 TCS 的流程生成 TCS 的 sub-TCS 图. sub-TCS 图是一种扩展的网络图[10]，其中 sub-TCS 被标识为结点，sub-TCS 的平均执行时间标识为结点的权重，sub-TCS 之间的状态迁移标识为结点之间的依赖关系，并且在图中加入了条件选择结点，用于连接流程中不同的条件分支，并为不同的条件分支分配了其执行的概率(各条件分支的执行概率可以通过统计组合服务在一段时间内的运行情况得到，也可以由专家根据领域知识直接分配)；另外还加入了 fork 和 joint 结点，用于表达流程中的并发关系. 图 5 演示了图 4 中 TCS 的 sub-TCS 图，连线上的数字是条件分支的执行概率(连线上没有标明数字意味着该路径的执行概率为 100%)，结点上方标识的是该结点的权值.

(4)将 TCS 的 sub-TCS 图作为输入，计算各 sub-TCS 的最大允许时间. 结点 $SubTcs_i$ 的最大允许时间记作：$p(SubTcs_i)$. sub-TCS 最大允许时间的计算算法如图 6 所示. 该算法通过调用两个主要的函数实现. 其中，函数 *ComputeCriticalPathLen*



说明：各结点上方的数字是 sub-TCS 的平均响应时间，整个 CTS 的最大允许时间为 6s，结点下方的数字是 sub-TCS 的最大允许时间.

图 5 sub-TCS 最大允许时间设定的实例

用于计算 sub-TCS 图中关键路径的长度. 所谓路径长度是指某一路径上所经历的所有结点的权值之和，而关键路径长度是指从开始结点到结束结点的所有路径中，长度最大路径的路径长度. 在计算关键路径长度时，对于由一个条件选择结点连接的多个条件分支，按照各分支执行的概率计算所有分支路径长度的加权平均值，并用此平均值作为所有条件分支统一的路径长度；对于并发执行的多条线索，选择这些线索的最大路径长度作为所有线索的路径长度，参与关键路径长度的计算. 例如，图 5 中 $SubTcs_2$ 与 $SubTcs_3$ 分属两条并发执行的线索，且 $SubTcs_2$ 所在路径的路径长度较大，所以选择其路径长度 5 作为所有并发线索的路径长度；$SubTcs_2$ 与 $SubTcs_3$ 所在的并发线索又同时与 $SubTcs_4$ 分属于两条不同的条件分支，计算条件分支路径长度的加权平均值为 4.4(5×0.7+3×0.3)，并将其作为所有条件分支的路径长度；最后加上 $SubTcs_1$ 的权值 3，图 5 中关键路径长度为 7.4. 函数 *ComputeMaxPermittedTime* 调用 *ComputeCriticalPathLen* 计算各结点的权值

```
STCS ← sub-TCS state diagram of the TCS;
MPT ← maximum permitted time of TCS;
ComputeMaxPermittedTime(STCS, MPT);
function ComputeMaxPermittedTime(in ProcSec, in MPT)
//ProcSec: sub-TCS 状态图中的片断;
//MPT: ProcSec 的最大允许时间
{BEGINNODE ← begin node of the process in ProcSec;
  ENDNODE ← end node of the process in ProcSec;
  NODE ← BEGINNODE;
  ComputeCriticalPathLen(ProcSec, CPT);
  while (NODE != ENDNODE ){
    NODE ← next node of the process in ProcSec;
    switch NODE is type of do {
      case sub-TCS node:
        //NODE.MPT: sub-TCS 结点的最大允许时间
        //NODE.ET: sub-TCS 结点的平均执行时间
        NODE.MPT ← (NODE.ET/CPT) * MPT;
      case conditional node:
        //CBET: 临时变量,计算条件分支响应时间的加权平均值
        CBET ← 0;
        for(each conditional branch){
          NewProcSec ← this conditional branch;
          //NewProcSec.P: 该条件分支的执行概率
          ComputeCriticalPathLen(NewProcSec,
                                 NewProcSec.P, CBT);
          CBET ←CBET+CBT; }
        MaxCB ← (CBET/CPT) * MPT;
        for(each conditional branch){
          NewProcSec ← this conditional branch;
          ComputeMaxPermittedTime(NewProcSec, MaxCB);}
      case fork node:
        for(each concurrent thread) {
          NewProcSec← this concurrent thread;
          //TEDT: 数组,TEDT[i]存储并发线索 i 的平均执行时间
          ComputeCriticalPathLen(NewProcSec,P,TEDT[i]);}
        //Max 返回数组 TEDT 中各元素中的最大值
        MaxTHD ← (Max(TEDT)/CPT) * MPT
        for(each conditional branch){
          NewProcSec ?this concurrent thread;
          ComputeMaxPermittedTime(NewProcSec,MaxTHD); }
    } //switch end
  } //while end
} //function end
function ComputeCriticalPathLen(in ProcSec, in P,
                                Out CPT)
//ProcSec: sub-TCS 图中的片断
//P: ProcSec 执行的概率
//CPT: ProcSec 中关键路径的响应时间
{BEGINNODE ← begin node of the process in Proc-
 Sec;
  ENDNODE ← end node of the process in ProcSec;
  NODE ← BEGINNODE;
  while (NODE != ENDNODE ){
    NODE ← next node of the process in ProcSec;
    switch NODE is type of do {
    case sub-TCS node:
    //NODE.ET: 结点 NODE 的平均执行时间
      CPT ←CPT+P * NODE.ET;
    case conditional node:
      for(each conditional branch){
        NewProcSec ← this conditional branch;
        //NewProcSec.P: 该条件分支的执行概率
        //CBT:临时变量,用于存储该条件分支的
        //平均执行时间
        ComputeCriticalPathLen(NewProcSec,
                               NewProcSec.P,CBT);
        CPT ← CPT+CBT * P;
      }//for end
    case fork node:
      for(each concurrent thread){
        NewProcSec ← this concurrent thread;
        //TEDT: 临时数组,TEDT[i]并发线索 i 的
        //平均执行时间
        ComputeCriticalPathLen(NewProcSec,P,
                               TEDT[i]);
      } //for end
      //Max 返回数组 TEDT 中各元素的最大值
      CPT ←CPT+Max(TEDT)
    } //switch end
  } //while end
} //function end
```

图 6 sub-TCS 最大允许时间分配算法

占关键路径长度的比例，并用此比例跟整个 TCS 最大允许时间相乘，得到该结点（即 sub-TCS）分配最大允许时间. 计算原则是，关键路径上的结点（或多个结点组成的流程片断）根据其权值占关键路径长度的比例，从整个 TCS 的最大允许时间中分得相应比例的时间；并发执行的多条线索，按照路径长度最大的线索为所有线索分配最大允许时间；由同一个选择分支结点连接的多条条件分支，统一按照平均路径长度分配最大允许时间. 例如，设整个 TCS 的最大允许时间为 6s，图 5 中各结点下方的数字是按照图 6 中的算法为结点分配的最大允许时间.

5　运行时 Web 服务的选择和调度

完成对 TCS 的分解后，服务代理将利用各 sub-TCS 的最大允许时间和时间敏感度级别为各任务选择具有合适响应时间的实例服务，以优化组合服务的响应时间. 在这一过程中，组合服务在每个任务执行之前，都会先访问服务代理，由服务代理为任务选择合适的实例服务. 运行时服务代理对 Web 服务的选择和调度通过以下步骤实现：

（1）服务执行引擎向服务代理汇报 sub-TCS 信息.

组合服务执行过程中，当流程进入某 sub-TCS 时（即开始执行某 sub-TCS 中的第一个任务之前），负责解析并执行组合服务的服务执行引擎将首先访问服务代理，并将该 sub-TCS 的信息报告给服务代理. 汇报的信息包括 sub-TCS 的组成（即该 sub-TCS 由哪几个顺序执行的任务组成）、sub-TCS 的时间敏感度级别、sub-TCS 的最大允许时间.

（2）服务代理根据时间敏感度级别对 sub-TCS 的最大允许时间进行预留.

时间敏感度级别越高意味着该 sub-TCS 超时所导致的 QoS 下降越大. 组合服务在实际执行过程中，由于 Web 服务的响应时间随时都可能变化，所以无法通过事先的调度确保 sub-TCS 没有超时的风险. 为了降低 sub-TCS 超时的可能性，服务代理根据时间敏感度级别对 sub-TCS 的最大允许时间进行预留，即在为任务选择实例服务时，留有一定比例的余量，使调度时实际使用的最大允许时间（称作实际最大允许时间，记作 $AMPT$）小于服务执行代理汇报的最大允许时间（称作汇报最大允许时间，记作 $RMPT$），从而降低 sub-TCS 超时的风险. 服务代理根据一组事先设定的预留比例常量 r_1，r_2，…，r_n（$0<r_1<r_2<\cdots<r_n<1$），计算各级别的预留值. 对于时间敏感度级别为 i 的 sub-TCS，其实际最大允许时间的计算公式为

$$AMPT=(1-r_i)\times RMPT \tag{3}$$

（3）服务代理对 sub-TCS 中各任务的响应时间进行动态调度.

本步骤又可以进一步被分为 3 个子步骤：

①为任务计算目标调度时间（OST）. 当 sub-TCS $SubTcs_k$ 的第 i 个任务 t_{ki} 请求服务代理选择实例服务时，服务代理计算该任务的目标调度时间，计算公式为

$$OST(t_{ki})=\frac{\bar{r}(t_{ki})}{\bar{r}(SubTcs_k)-(T-T_0)}\times[AMPT-(T-T_0)] \tag{4}$$

其中，T_0 是 sub-TCS 开始的时间（即服务执行引擎向服务代理汇报 sub-TCS 信息的时间），T 是当前时间. 因为同一个 sub-TCS 中的各任务是顺序执行的，所以当第 i 个任务的服务请求到达时，前面的 $i-1$ 个任务都已经执行完成了，这些任务的响应时间也已经发生了（即 $T-T_0$）. 上面的公式利用这一信息更新新任务的目标调度时间，从而实现对任务响应时间的动态调整. 也就是说，如果前面 $i-1$ 个任务的实际响应时间超过了预先计划的值，可以通过为后面的任务选择更快的实例服务将时间赶回来；反之，如果前面 $i-1$ 个任务的响应时间比计划的短，那么可以为后面的任务选择响应速度慢一些但其它 QoS 属性更优的实例服务.

②为任务挑选实例服务. 选择的规则：如果存在至少一个响应时间小于目标调度时间的实例服务，则选择这些实例服务中响应时间最大的一个；否则选择响应时间最小的实例服务.

③将选中的实例服务通告给组合服务执行引擎，组合服务执行引擎与其绑定，并调用这些实例服务.

6　实　验

为了验证本文提出的组合服务响应时间优化方法，我们编写了一个简单的组合 Web 服务，该服务仅包括一个如图 4 所示的 TCS. 实验中采用北京大学的青鸟构件库管理系统作为 UDDI 注册中心和服务代理[7]. 整个实验是在 2GHz Pentium 4 处理器和 256MB 内存的 PC 机器上进行的. 在服务代理中

设置时间敏感度级别为 1，2，3 级时的预留比例常量分别为 0，0.1，0.2. 为图 4 中的每个任务各注册了 5 个实例服务. 各实例服务的平均响应时间设置如表 1 所示.

表 1 各任务实例服务的平均响应时间

任务	实例服务响应时间(s)				
	1	2	3	4	5
t_{11}	0.8	0.9	1.0	1.1	1.2
t_{12}	1.6	1.8	2.0	2.2	2.4
t_{21}	2.4	3.2	4.0	4.8	5.6
t_{22}	0.8	0.9	1.0	1.1	1.2
t_{31}	1.4	1.7	2.0	2.3	2.6
t_{41}	0.7	0.9	1.0	1.1	1.3
t_{42}	0.3	0.5	1.0	1.5	1.7
t_{43}	0.2	0.4	1.0	1.2	2.2

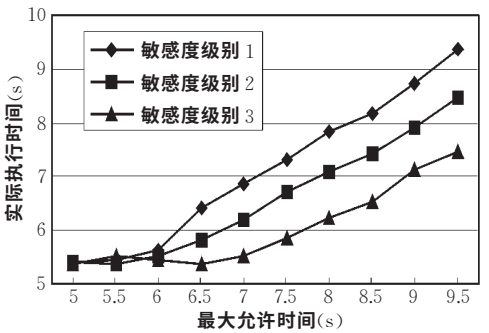
实验在 A，B 两种条件下进行.

条件 A. 所有实例服务的响应时间在实验过程中保持不变.

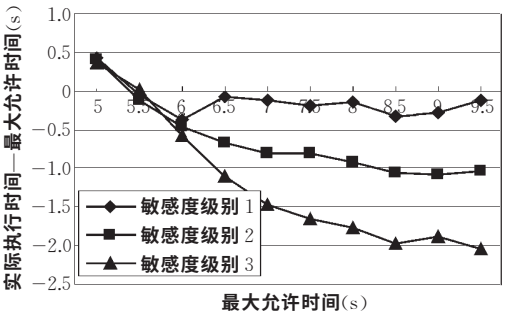
条件 B. 各实例服务的响应时间按 50% 的概率，每隔 0.5s 上下浮动一次，浮动幅度为平均响应时间的 20%～40%；同时各实例服务每隔 1s，以 30% 的概率失效一次，失效时间为 2s.

在两种条件下，调整 TCS 的最大允许时间和时间敏感度级别，每种情况独立实验 10 次，取其平均值作为实验结果. 实验数据如图 7 所示.

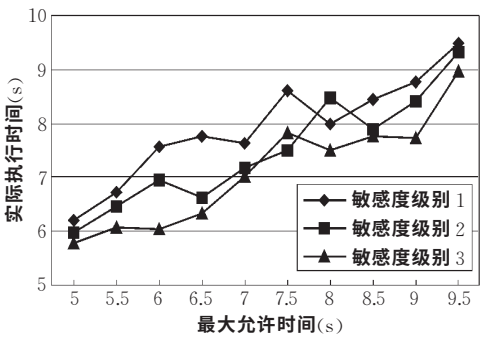
从图 7(a)和图 7(b)中可以看到在条件 A 下，当最大响应时间小于 6s 时，TCS 的实际响应时间基本稳定，不随最大允许时间和时间敏感度级别而变化. 这是因为即使为所有的任务选择最快的实例服务，TCS 的执行时间至少也需要 5.6s，当最大响应时间小于或接近这一极限时间时，算法为关键路径上的任务选择了最快的实例服务. 当最大响应时间大于 6s 后，TCS 的实际执行时间稳定增长并始终小于其最大允许时间. 当时间敏感度级别较高时，由于调度时采用了时间预留策略，使得 TCS 的实际执行时间偏小，与最大允许时间的差距较大(在图 7(b)中这一点显示得尤为明显). 这说明当所有 Web 服务的响应时间始终保持稳定时(条件 A)，时间预留策略并无必要. 但是当 Web 服务的响应时间变化较为频繁时(条件 B)，情况就不同了. 从图 7(c)和图 7(d)看，在条件 B，TCS 的实际执行时间出现了明显的随机波动，特别是从图 7(d)可见，当最大允许时间设置较小(小于 7s)，而且时间敏感度较低时，TCS 的实际响应时间超出了其最大允许时间. 但是相同情况下，时间敏感度级别较高时，因为时间预留和对任务响应时间的动态调整，使得 TCS 的实际响应时间超出最大允许响应时间的概率大大降低.



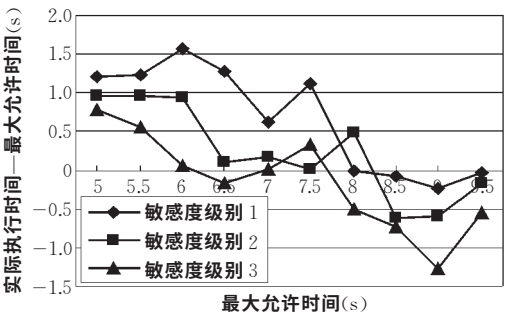
(a) 条件 A 下实际响应时间随最大允许时间变化的曲线



(b) 条件 A 下实际响应时间与最大允许时间之间的差值随最大允许时间变化的曲线



(c) 条件 B 下实际响应时间随最大允许时间变化的曲线



(d) 条件 B 下实际响应时间与最大允许时间之间的差值随最大允许时间变化的曲线

图 7 TCS 在不同最大允许时间和敏感度级别下的实际响应时间

7　相关研究工作及其与本文方法的比较

文献[11,12]研究了组合 Web 服务的 QoS 管理和优化问题，他们综合考虑服务的执行时间和价格，为组合服务中的任务选择具有恰当 QoS 的 Web 服务. 文献[13]根据用户对各 QoS 属性的不同侧重，为其赋予不同的权值，并提出了一套归一化的方法，将不同 QoS 属性的取值按照其权值统一为一个整体的取值，辅助用户为组合服务中的任务选择具体的 Web 服务. 上述方法在为组合服务选择成员 Web 服务时，是针对各任务单独进行的，没有考虑各任务之间的关系. 文献[8,9,14,15]与之不同，他们对组合服务中的所有任务进行了全局的考虑，采用线性规划、遗传算法等方法对组合服务的整体 QoS 进行了全局的优化，并且在服务运行中，当发生影响组合服务 QoS 的事件时(例如：某预选的 Web 服务的 QoS 发生显著的变化，出现了具有更优 QoS 的新 Web 服务，或者组合服务确定了本次运行中将执行的条件分支等)，重新执行优化算法，对 Web 服务的选择方案进行调整. 这些方法虽然对多个任务进行了全局的优化，但是没有考虑不同组合服务业务逻辑和流程上的差异，将全局优化的范围固定设置为组合服务中所有任务. 文献[16]给出了一种在网格环境中进行服务选择和组合的方法，该方法采用服务提供者投标的方式，优化组合服务的响应时间，并在整个网格内部均衡不同结点的负载，但是此文也没有从用户的视角，对组合服务流程中不同环节对时间的敏感程度进行分析. 与之不同，本文从用户的角度出发，将组合服务中的任务划分为若干个 TCS，并在 TCS(或 sub-TCS)范围内对多个任务进行全局优化，以提高用户对服务的满意度. 另外，在服务运行时多次执行优化算法，将带来很大的计算开销和网络传输开销，特别是当优化算法由一个集中的计算单元(比如：服务代理)执行时，将显著影响整个体系结构的扩展性；同时，运行时执行优化算法还将会延长组合服务的响应时间，降低 QoS. 本文的方法中，大部分的计算量都在组合服务运行前完成，并且在组合服务运行中通过实时计算 TCS(或 sub-TCS)的剩余最大允许时间，使得 Web 服务的选择可以根据 Web 服务实际响应时间与预期响应时间的差距动态调整，从而大大降低了方法的开销，同时提供了较好的动态优化效果. 本文还提出了时间敏感度级别的概念，实验显示其对于降低组合服务中关键环节超时的风险具有明显的效果.

8　结　论

Web 服务的 QoS 管理是学术界和工业界共同关注的热点问题. Web 服务的 QoS 包括诸如：响应时间、价格、成功执行率、可获得性等多个属性. 其中，响应时间是被所有研究者共同关注的重要属性，组合 Web 服务流程中的循环、并发、条件选择等结构导致响应时间的计算较之其它 QoS 属性更加复杂. 本文针对响应时间这一 QoS 属性展开了研究工作，提出了一种基于用户视角的组合 Web 服务响应时间优化方法. 该方法首先从用户的视角出发，对组合服务的流程进行分析，将用户一次完整等待过程中组合服务执行的流程片断标识为一个时间临界区(TCS)，并且为每一个 TCS 设置了最大允许时间和时间敏感度级别这两个属性. 最大允许时间描述用户可以接受的该 TCS 的最大响应时间(超过这一时间，用户对服务的满意度将下降)，而时间敏感度级别描述 TCS 一旦超时，用户满意度下降的速度. 在组合服务运行前，还需要将 TCS 分解为一组 sub-TCS，使得 sub-TCS 中的任务全部是顺序执行的，并且为 sub-TCS 分配最大允许时间和时间敏感度级别. 运行时服务代理对组合服务中各任务的实例服务进行调度，并将 TCS(或 sub-TCS)作为一个整体，对其中的任务进行全局的优化，同时根据 TCS(或 sub-TCS)的时间敏感度级别为其预留一定比例的时间，降低组合服务流程中关键环节超时的风险. 由于大部分的计算量都在服务运行前完成，所以本文的方法具有较小的开销；由于运行时根据 Web 服务实际响应时间对 Web 服务实施动态调度，所以本文的方法还具有较好的动态优化效果. 实验表明，本文的方法可以有效地优化组合 Web 服务的响应时间，提高用户的满意度和服务的 QoS. 作者下一步工作是将本文提出的方法扩展到对诸如可靠性、可用性等 QoS 属性中，对多个 QoS 属性进行综合的优化.

参　考　文　献

1　Ran Shu-Ping. A model for Web services discovery with QoS. ACM SIGecom Exchanges, 2003, 4(1): 1～10

2 Zhao Jun-Feng, Wang Ya-Sha, Xie Bing, Yang Fu-Qing. A management framework of component supporting QoS of component. ACTA Electronica Sinica, 2004, 32(12A): 165～168 (in Chinese)
(赵俊峰,王亚沙,谢 冰,杨芙清.一种支持构件服务质量的构件管理框架.电子学报, 2004, 32(12A): 165～168)

3 Eder J., Panagos E., Rabinovich M.. Time constraints in workflow systems. In: Proceedings of the 11th International Conference(CAiSE'99), 1999, 286～300

4 Gillmann M., Weikum G., Wonner W.. Workflow management with service quality guarantees. In: Proceedings of ACM SIGMOD International Conference, Management of Data, Madison, USA, 2002, 228～239

5 ITU-T Recommendation 1.350, General aspects of quality of service and network performance in digital network, including ISDNs. ITU-T, 1993

6 Sun Yan-Chun, Zou Wei, Ma Liang. A component library management system based on Internet. Computer Engineering, 2000, 26(11): 85～86(in Chinese)
(孙艳春,邹 炜,马 亮.一个基于 Internet 的构件库管理系统 JBCLMS.计算机工程, 2000, 26(11): 85～86)

7 Zhao Jun-Feng. Research on feedback management and run-time application supporting techniques for software component library[Ph. D. dissertation]. Beijing: Peking University, 2005 (in Chinese)
(赵俊峰.构件库反馈管理及运行时应用支持技术的研究[博士学位论文].北京:北京大学, 2005)

8 Zeng L. Z., Benatallah B., Dumas M.. QoS-aware middleware for Web services composition. IEEE Transactions on Software Engineering, 2004, 30(5): 311～327

9 Canfora G., Di Penta M., Esposito R., Villani M. L.. Qos-aware replanning of composite Web services. In: Proceedings of the IEEE International Conference on Web Services, Orlando, Florida USA, 2005, 121～129

10 Pinedof M.. Scheduling: Theory, Algorithms, and Systems. 2nd Edition. New York, USA: Prentice Hall, 2001

11 Georgakopoulos D., Schuster H., Cichocki A., Baker D.. Managing process and service fusion in virtual enterprises. Information System, Special Issue on Information System Support for Electronic Commerce, 1999, 24(6): 429～456

12 Casati F., Shan M. C.. Dynamic and adaptive composition of e-services. Information Systems, 2001, 26(3): 43～162

13 Liu Yu-Tu, Anne N. H. H., Zeng Liang-Zhao. QoS computation and policing in dynamic Web service selection. In: Proceedings of the World Wide Web Conference, New York, USA, 2005, 66～73

14 Cardoso J.. Quality of service and semantic composition of workflows[Ph. D. dissertation]. University of Georgia, Athens, Georgia, USA, 2002

15 Ardagna D., Pernici B.. Global and local QoS constraints guarantee in Web service selection. In: Proceedings of the IEEE International Conference on Web Services, Orlando, Florida USA, 2005, 806～807

16 Cheung W. K., Liu J., Tsang K. H., Wong R. K.. Toward autonomous service composition in a grid environment. In: Proceedings of the IEEE International Conference on Web Services, San Diego, California, USA, 2004, 550～557

WANG Ya-Sha, born in 1975, Ph.D., lecturer. His research interests include software reuse, Web services technology, and software process.

ZHAO Jun-Feng, born in 1974, Ph. D, lecturer. Her research interests include software reuse, component technology, and software reuse repository.

XIE Bing, born in 1970, Ph. D., associate professor. His research interests include software engineering, formal method and distributed system.

Background

The software component library is one of the infrastructures of software reuse, which provides the management mechanism of components. It is one of the important research area studied by the software institute of school of electronics engineering and computer science of Peking University. With the development of distributed component technologies, such as Web services, more and more software fulfill the required functions via Web services invocation. In this case, the software component library should manage Web services and their QoS. Zhao Junfeng's dissertation "Research on Feedback Management and Run-time Application Supporting Techniques for Software Component Library" has conducted intensive research on the QoS modeling and metric of Web services, the collecting and management of QoS information based on the component library and run-time application supporting techniques. The work of the paper is conducted under the background and extends the research mentioned above.