一种基于条件Pi演算的组合服务柔性演化模型

刘 涛 曾国荪
(同济大学计算机科学与技术系 上海 200092)
(同济大学嵌入式系统与服务计算教育部重点实验室 上海 200092)

摘 要 组合服务应当具有适应所处环境和商业规则不断变化的能力。现有的服务组合语言和执行机制缺乏应对动态变化所需的可变性和适应性。尽管已有很多扩展,但是组合服务的动态适应正确性还缺乏保障。提出了一种基于条件Pi演算的组合服务柔性演化机制。通过增加归属操作符和条件控制符对经典Pi演算进行扩展,使之与事件-条件-动作模式更好地结合起来,从而提出了一种描述组合服务流程的方法。从分析组合服务各种变化的场景出发,提出了11种基本的变化场景及其不同的柔性演化模式,对每种模式进行了形式化描述和分析。该方法能够保证组合服务的可变性和适应正确性。

关键词 条件Pi演算,服务自适应,柔性演化模式
中图法分类号 TP393 **文献标识码** A

Model for Flexible Evolution of Composite Services Based on Conditional Pi Calculus

LIU Tao ZENG Guo-sun
(Department of Computer Science and Technology, Tongji University, Shanghai 200092, China)
(The Key Laboratory of Embedded System and Service Computing, Ministry of Education, Tongji University, Shanghai 200092, China)

Abstract Composite services ought to be adaptable to changing environments and business rules. Existing service composition languages and execution engines lack variability and adaptability needed to cater for dynamic changes. In spite of many extensions, how to ensure the correctness for the dynamic adaptation of composite services remains a challenge. We proposed a mechanism for flexible evolution of composite services based on conditional Pi calculus. We extended classical Pi calculus through introducing the conditional control operator and the belonging operator, which suits the Event-Condition-Action(ECA) pattern better. A method to describe composite service processes was proposed based on conditional Pi calculus and ECA. Through scenario analysis of various composite service changes, eleven basic dynamic scenarios and corresponding flexible evolution patterns were proposed. Each pattern was formalized and analyzed to ensure the variability and adaptation correctness of composite services.

Keywords Conditional Pi calculus, Service adaptation, Flexible evolution pattern

1 引言

Web服务是拥有标准接口描述的可编程模块,通过标准通信协议提供通用可访问性。Web服务可以用来动态构建复杂应用。服务组合能够将组织内部和组织间的服务联合起来提供更复杂的功能,称为组合Web服务。

组合服务的开发环境和部署环境经常发生动态变化。然而,已有的服务组合语言的可变性和适应性并不好。因此,服务组合自适应问题的研究正吸引越来越多的注意力,主要分为动态适应和静态适应。静态适应要求组合服务在设计时发生改变,如文献[1]提出了设计时可变性的管理方法,文献[2]提出了自上而下变化的自动管理框架。动态适应允许在运行时改变服务实例的行为,如面向方面的编程可以用于客户化组合服务实例[3]。在工作流研究领域,同样存在动态适应问题,如文献[4]提出用继承关系保证工作流实例的动态正确性,文献[5]提出了采用迹等价方法解决相同问题。

Papazoglou教授提出了服务演化的概念来描述服务的不断变化,并且提出采用服务协议来保证其正确性[6]。但是,以上提及的各种方法都缺乏形式化的描述。我们提出采用柔性演化的模式来应对组合服务的不断变化,并且以扩展的Pi演算来作为其形式化描述语言,为组合服务自适应提供一种形式化基础。

2 基于条件Pi演算的组合服务建模

2.1 条件Pi演算

Pi演算[7]已经成为并发理论中最重要的模型之一,它由

本文受863项目(2007AA01Z425,2009AA012201),973专项(2007CB316502),国家自然基金项目(90718015),NSFC-微软亚洲研究院联合资助项目(60970155),教育部博士点基金项目(20090072110035),上海市优秀学科带头人计划项目(10XD1404400),高效能服务器和存储技术国家重点实验室开放基金项目(2009HSSA06)资助。

刘 涛 博士生,主要研究领域为服务计算、自主计算,E-mail:taoliu.tongji@gmail.com;曾国荪 博士,教授,博士生导师,主要研究领域为网格计算、信息安全。

Robin Milner 对 CCS(Calculus of Communicating Systems)进行扩充而得到,常被用来描述进程间的交互行为和并发通信操作。Pi 演算有多种表示形式[8],本文采用一种较为通用的形式,定义如下。

定义 1 Pi 演算的最基本组成部分是名字,不区分变量、数值等,如通信通道、数据值和进程名等都作为名字处理[7]。其基本语法如式(1)所示:

$$\begin{aligned} &P::=M|P|P'|vzP|!\ P \\ &M::=0|\pi.P|M+M' \\ &\pi::=\bar{x}\langle y\rangle|x(z)|\tau|[x=y]\pi \end{aligned} \tag{1}$$

其中:

- P 代表进程;
- $P|P'$ 表示 P 和 P' 并行执行;
- vzP 表示名字 z 被限制在进程 P 内部,或者产生唯一的新名字 z;
- $!\ P$ 表示进程 P 被重复复制无穷次;
- 0 表示空进程,没有动作发生;
- $M+M'$ 表示互斥执行 M 和 M',即要么执行 M,要么执行 M';
- $\bar{x}\langle y\rangle$ 为输出前缀,表示通过连接 x 发送名字 y,然后继续进程 P;
- $x(z)$ 为输入前缀,表示进程 P 从连接 x 接收到一个名字并替换为 z 后,继续进程 P;
- τ 为表示进程 P 的内部动作,外部不可见;
- $[x=y]\pi$ 表示如果 x 匹配 y,执行进程 P。

以上简单回顾了 Pi 演算的基本定义,有关于 Pi 演算的进一步信息可以参考文献[7,8]。

Pi 演算是一种非常有用的形式化语言,但也有其局限性。特别是用 Pi 演算描述多个 Web 服务间的协作和语义时,它不能描述服务间交互行为的约束规则和服务的归属。通过深入分析 Web 服务组合的特点,我们对 Pi 演算进行语义扩充,提出了条件 Pi 演算(Conditional Pi-calculus),简称 CPi 演算[9,10]。下面给出其定义。

定义 2 CPi 演算对 Pi 演算进行了两方面的扩展:

- 对式(1)中的前缀定义 π 进行扩充,引入条件控制符号 $[\theta]$,其中 θ 为一条件表达式,则可重新定义前缀 π 为:

$$\pi::=\bar{x}\langle y\rangle|x(z)|\tau|[\theta]\pi$$

- 对 Pi 演算进行归属扩展,即引入归属符号@,@是一个二元操作符,则 $N_1@N_2$ 表示名字 N_1 归属于名字 N_2。

条件控制符号$[\theta]$的引入可以看作是对式(1)前缀定义中 $[x=y]\pi$ 的广义扩展,前者可以表达比后者更广泛的条件,即二者的关系可以表达为$\{[\theta]\pi\rangle\supset\{[x=y]\pi\rangle$。而归属符号@的引入是为了将其更好地应用于 *Web* 服务,例如 P@S 表示进程归属于服务 S。式(2)中扩展后的前缀 π 可以分别定义如下:

- $\pi.P::=[\theta]\bar{x}\langle y\rangle.P$,表示输出动作受局部条件 θ 控制,条件满足时,通过信道 x 输出消息 y,然后执行进 P 程。
- $\pi.P::=[\theta]x\langle z\rangle.P$ 表示满足条件 θ,则从信道 x 接收消息 z,然后执行进程 P。
- $\pi.P::=[\theta]\tau.P$ 表示满足条件 θ,则执行动作 τ,然后执行进程 P。

CPi 演算对 Pi 演算中进程间交互行为的约束规则和进程的归属进行了适当扩展,并继承了大部分 Pi 演算的语法和语义,有关的具体语法与语义规则可参考文献[9,10]。

2.2 组合服务建模

关于 Web 服务的形式化模型有很多,但都不适合于描述适应性 Web 服务组合。下面用我们提出的 CPi 演算对 Web 服务组合进行形式化建模。

定义 3 假设 Web 服务组合 S 由 n 个原子服务组合而成,用 S_i 表示,其中 $i=\{1,\cdots,n\}$。所有参与组合服务 S 交互执行的各原子服务操作的集合用 C 表示,即 $C=\{p,\cdots,q\}$,称为服务操作。

定义 4 假设组合服务 S 的一个原子服务为 S_i,该原子服务一个操作 p 参与组合服务 S 的执行,则使用归属符号记为 $p@S_i$,表示服务操作 p 归属于原子服务 S_i,@称为服务归属。

有了以上定义的服务操作与服务归属,我们定义组合服务的基调如下。

定义 5 组合服务 S 的服务基调(Composite Service Schema),可以定义为一个五元组:

$$\sum_S=(S,A,E,C,\varphi) \tag{3}$$

如式(3)所示,其中:

- S 为组合服务绑定的原子服务的集合,$S=\{S_1,S_2,\cdots,S_n\}$。
- $A=\bigcup_{i=1}^{n}A_i$ 为 S 中各原子服务所拥有的服务操作的集合,其中 $A_i=\{a_{i1},a_{i2},\cdots,a_{im}\}$ 表示原子服务 S_i 所拥有的服务操作的集合。
- E 表示组合服务 S 所有调用执行过程的集合。E 中的某次调用执行过程,例如 $S_1\prec S_2\prec S_3\prec S_4\prec\cdots$,为一偏序序列。
- C 表示组合服务 S 所有执行过程中原子服务的服务操作偏序序列。例如,$(S_1,a_{11})\prec(S_2,a_{21})\prec(S_3,a_{31})\prec(S_1,a_{21})\prec(S_4,a_{41})\prec\cdots$为一服务操作偏序序列。其中$(S_i,a_{ij})\in A$ 表示执行原子服务 S_i 的服务操作 a_{ij},用服务归属来表示就是 $a_{ij}@S_i$。将服务操作偏序序列中的(S_i,a_{ij})对应为 $a_{ij}@S_i$,再将服务操作偏序序列中的"$\prec$"符号对应为顺序操作符".",则上式用 CPi 演算可表示为$(a_{11}@S_1).(a_{21}@S_2).(a_{31}@S_3).(a_{12}@S_1).(a_{42}@S_4).\cdots$。
- $\varphi:A\to A^*$ 为 A 到 A^* 的一个操作偏序关系映射,使得 $\varphi(S,E)=C$,即 φ 是一个组合服务 S 到每个原子服务 S_i 之间操作偏序序列的映射函数。

我们用 $\sum_S$ 来刻画了一个组合服务的组织结构,进行了上层合成服务对下层组成服务集的抽象。C 表明了组合服务某个执行过程与特定服务的特定操作的关系,而 E 表明了原子服务到组合服务的抽象。操作偏序关系映射的引入,使得组合服务的结构框架和服务调用机制有机地结合起来。

3 组合服务的流程模式

直接使用 CPi 演算来描述组合服务的流程模式存在不便之处,因此本文将 CPi 演算与 ECA 相结合,更清晰地表达组合服务的流程模式和柔性演化。

ECA(Event,Condition,Action),即事件-条件-动作,来源于主动数据库系统[10],在工作流和 Web 服务领域都有广泛应用,也已成为自主计算的主要方法之一[11]。ECA 规则中

的事件(Event)用来指定规则何时适用,如果符合给定条件(Condition)则执行相应动作(Action)。下面用我们提出的CPi演算来描述ECA方法。

定义6 基于CPi演算的ECA基本形式可表示为:

$$x(z).[\theta].\tau.\bar{y}\langle z'\rangle.0 \tag{4}$$

如式(4)所示,其中:

- $x(z)$表示ECA中的事件。
- 条件控制符$[\theta]$表示需要满足的条件。
- $\tau.\bar{y}\langle z'\rangle$表示在事件被触发并且满足约束条件的情况要执行的动作,τ用来表示不可见的动作,$\bar{y}[z']$表示能够触发其他进程。

即进程接收表示为$x(z)$的ECA触发事件,与给定条件$[\theta]$进行比较,执行相关的内部动作τ,最后通过$\bar{y}\langle z'\rangle$来触发另一进程。可以更广义的表示为:

$$\{x_i(z_i)\}_{i=1}^{m}.\{[\theta_j]\}_{j=1}^{n}.\tau.\{\bar{y}\langle z'\rangle\}_{k=1}^{o}.0 \tag{5}$$

如式(5)所示,基于CPi演算的ECA可以接收m个触发,检查n个设定条件,发出o个触发。

应用到描述组合服务的流程模式时,我们将定义3中的每个服务操作看作是一个进程,也就是一个ECA。例如定义5中$a_{ij}@S_i$的ECA形式可表示为$x(z).[\theta].a_{ij}@S_i.\bar{y}\langle z'\rangle.0$。

下面我们用一个例子来说明如何使用基于CPi演算的ECA来对组合服务流程片段进行描述,为了突出重点,我们根据需要省略非必须的部分元素。假设p、q、r分别表示归属相应原子服务的服务操作,分别用大写字母A、B、C表示其对应的进程,然后用一个完整的表达式来分别表示每个流程。

图1中的选择分支片段可以表示为:

$A=x(m).[\theta]p@S_i$

$B=y(n).[\psi]q@S_j$

即 $x(m).[\theta]p@S_i+y(n).[\psi]q@S_j$

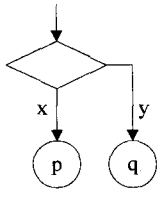


图1 组合服务选择分支片段

由上例可以看出,服务归属和条件控制符的引入很好地解决了组合服务中服务操作交互执行时的归属表达和条件控制问题。

4 组合服务的柔性演化

组合服务应该具备柔性演化的能力,以便随着环境的需求和变化持续地改变自己,并且操作功能和运行性能没有任何损失,总体说来就是要让服务组合过程具有动态的自主能力[12]。组合服务的开发、部署和维护处于一个经常发生动态变化的开放环境中[13],因此本文提出了基于CPi演算的组合服务柔性演化模式来应对这种动态改变。

由于客户需求和商业逻辑都是不断变化的,因此Web服务组合必须是柔性的,才能跟得上服务环境的不断变化。参考文献[14]中给出的工作流模式,我们针对不同场景的组合服务柔性演化分别进行深入研究,提出了以下各小节的组合服务柔性演化模式(Flexible Evolution Pattern of Composite Service,以下简称FEPCS)。

假设组合服务S,其服务基调采用定义5中$\sum_S=(S,A,E,C,\varphi)$的形式,C是中的服务操作偏序序列集合。假设$c\in C$为某一操作序列,用c'表示柔性演化后的操作序列,显然$c'\in C$,即$c\xrightarrow{POSAP}c'$。和上节一样,我们用服务归属的形式来表示每个服务操作,而用大写字母分别表示每个服务操作所对应的独立进程。为了更清晰地说明柔性演化的过程,我们采取图示与CPi演算表达式相结合的方式进行描述。

4.1 服务操作插入演化

设ic为S的一个服务操作,要加入到服务操作序列c中,则有$c\xrightarrow{insert(ic)}c'$。下面分别讨论4种常见的插入服务操作演化模式。

FEPCS1-1:顺序插入演化,即直接在两个顺序执行的服务操作中插入一个操作,如图2所示。

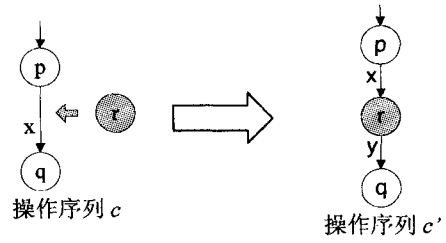


图2 FEPCS1-1:服务操作顺序插入演化

服务操作顺序插入前的序列c表示为:

$A=p@S_i.[\theta]\bar{x}\langle m\rangle.0$

$B=[\psi]x(n).q@S_j$

即 $p@S_i.[\theta]\bar{x}\langle m\rangle.[\psi]x(n).q@S_j$

其中ic为:$C=r@S_k$。

服务操作顺序插入后的序列c'表示为:

$A=p@S_i.[\theta]\bar{x}\langle m\rangle.0$

$C=[\zeta]x(n).r@S_k.[\xi]\bar{y}\langle e\rangle.0$

$B=[\psi]y(f).q@S_j$

即 $p@S_i.[\theta]\bar{x}\langle m\rangle.[\zeta]x(n).r@S_k.[\xi]\bar{y}\langle e\rangle.[\psi]y(f).q@S_j$

随着服务操作的插入,消息的发送接收条件及信道都要做出相应的改变,即体现了变化、可适应动作的模式。

为了简便,在不涉及发送接收消息参数的前提下,以下将省略消息参数,即将$\bar{x}\langle m\rangle$、$x(n)$分别简化为$\bar{x}$、x。

FEPCS1-2:并行插入演化,即插入一个服务操作与原有服务操作并行执行,具体场景如图3所示。

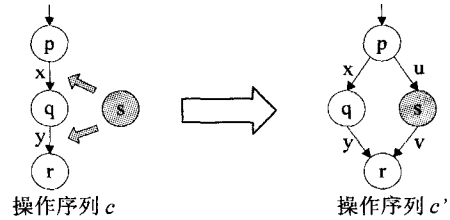


图3 FEPCS1-2:服务操作并行插入演化

服务操作并行插入前的序列c表示为:

$A=p@S_i.[\theta]\bar{x}.0$

$B=[\zeta]x.q@S_j.[\xi]\bar{y}.0$

$C=[\psi]y.r@S_k$

即 $p@S_i.[\theta]\overline{x}.[\zeta]x.q@S_j.[\xi]\overline{y}.[\psi]y.r@S_k$

其中 ic 为：$D=s@S_l$。

服务操作并行插入后的序列 c' 表示为：

$A=p@S_i.([\theta]\overline{x}|[\eta]\overline{u}).0$

$B=[\zeta]x.q@S_j.[\xi]\overline{y}.0$

$D=[\mu]u.s@S_l.[\sigma]\overline{v}.0$

$C=([\psi]y|[\varsigma]v).r@S_k$

即 $p@S_i.([\theta]\overline{x}|[\eta]\overline{u}).(([\zeta]x.q@S_j.[\xi]\overline{y})|([\mu]u.s@S_l.[\sigma]\overline{v})).([\psi]y|[\varsigma]v).r@S_k$

并行插入一个服务操作，意味着两个服务操作将并行执行，而他们的前驱和后继服务操作也由发送、接收单一消息变为并行发送、接收两条消息。

FEPCS1-3：条件插入演化，与顺序插入类似，即在两个顺序执行的服务操作中间插入一个服务操作，但所插入的服务操作是否能够执行取决于其本身的限定条件，如图 4 所示。

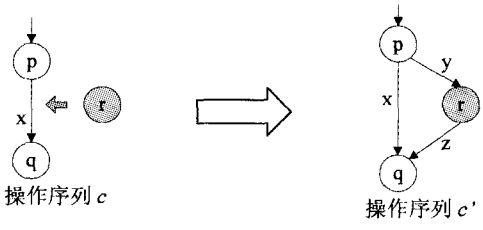


图 4 FEPCS1-3：服务操作条件插入演化

服务操作条件插入前的序列 c 表示为：

$A=p@S_i.[\theta]\overline{x}.0$

$B=[\psi]x.q@S_j$

即 $p@S_i.[\theta]\overline{x}.[\psi]x.q@S_j$

其中 ic 为：$C=[\sigma]r@S_k$。

服务操作条件插入后的序列 c' 表示为：

$A=p@S_i.([\theta]\overline{x}+[\xi]\overline{y}).0$

$C=[\zeta]y.[\sigma]r@S_k.[\eta]\overline{z}.0$

$B=([\psi]x+[\mu]z).q@S_j$

即 $p@S_i.([\theta]\overline{x}+[\xi]\overline{y}.[\zeta]y.[\sigma]r@S_k.[\eta]\overline{z}).([\psi]x+[\mu]z).q@S_j$

条件插入意味着服务操作序列片段的执行有两种互斥的方式。其一是按照原有方式顺序执行，没有任何改变。其二是在原有服务操作间执行一个带条件限定的操作，满足条件时执行插入操作，不满足时执行结果与原来一样。

FEPCS1-4：拷贝插入，与顺序插入类似，不同之处在于所插入的服务操作不是全新的操作，而是将操作序列中原有的一个操作拷贝后插入到所需位置中，如图 5 所示。

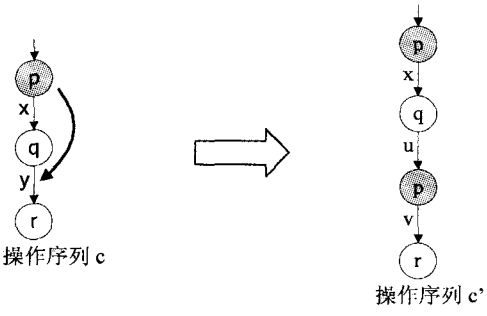


图 5 FEPCS1-4：服务操作拷贝插入演化

服务操作拷贝插入前的序列 c 表示为：

$A=p@S_i.[\theta]\overline{x}.0$

$B=[\psi]x.q@S_j.[\xi]\overline{y}.0$

$C=[\zeta]y.r@S_k$

即 $p@S_i.[\theta]\overline{x}.[\psi]x.q@S_j.[\xi]\overline{y}.[\zeta]y.r@S_k$

其中 ic 为：$D=p@S_i$。

服务操作拷贝插入后的序列 c' 表示为：

$A=p@S_i.[\theta]\overline{x}.0$

$B=[\psi]x.q@S_j.[\alpha]\overline{u}.0$

$D=[\beta]u.p@S_i.[\sigma]\overline{v}.0$

$C=[\varsigma]v.r@S_k$

即 $p@S_i.[\theta]\overline{x}.[\psi]x.q@S_j.[\alpha]\overline{u}.[\beta]u.p@S_i.[\sigma]\overline{v}.[\varsigma]v.r@S_k$

拷贝插入可看作是顺序插入的一种特例，但因为发生这种流程变更的场景很多，所以把它单独列出。

总之，插入服务操作在组合服务动态变化中发生的非常多，随着服务操作的插入，原有执行序列也要做出柔性演化。

4.2 服务操作删除演化

与插入服务操作相对应的自然是删除服务操作，它可以看作是插入服务操作的逆向过程。设 dc 为操作序列 c 中的一个服务操作，现在要将 dc 从操作序列 c 中删除，即 $c \xrightarrow{delete(dc)} c'$。下面是从顺序操作中删除一个服务操作的具体刻画（见图 6）。

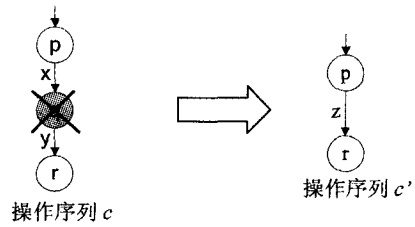


图 6 FEPCS2：服务操作删除演化

服务操作删除前的序列 c 用 CPi 演算表示为：

$A=p@S_i.[\theta]\overline{x}.0$

$B=[\zeta]x.q@S_j.[\xi]\overline{y}.0$

$C=[\psi]y.r@S_k$

即 $p@S_i.[\theta]\overline{x}.[\zeta]x.q@S_j.[\xi]\overline{y}.[\psi]y.r@S_k$

其中 dc 为：$D=q@S_j$。

服务操作删除后的序列 c' 用 CPi 演算表示为：

$A=p@S_i.[\sigma]\overline{z}.0$

$C=[\varsigma]z.r@S_k$

即 $p@S_i.[\sigma]\overline{z}.[\varsigma]z.r@S_k$

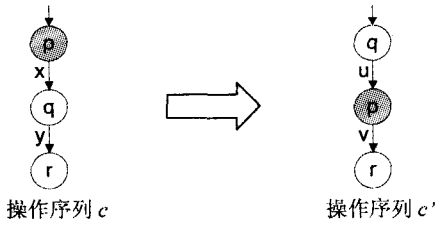
由上可见，删除服务操作比插入服务操作要简便得多，最重要的动作就是更新要删除操作的前驱和后继操作的发送和接收信道，使其能直接通信，绕过要删除的操作。同理，要删除并行、互斥等复杂流程中的一个操作，过程与以上类似，只需更新前驱和后继服务操作的通信约束条件和通信信道。

4.3 服务操作迁移演化

组合服务根据特定的约束来确定服务操作调用的先后次序，而这种事先确定的次序经常需要随环境和需求的变化而改变，因此服务操作的迁移成为可适应服务组合的重要一环。与插入服务操作相似，迁移服务操作需要考虑的情景很多，我们挑选比较有代表性的 3 种详细说明如下。

假设 mc 为某一操作序列 c 中的一个服务操作，现在需要移动 mc 在操作序列 c 中所处的位置，即有 $c \xrightarrow{move(mc)} c'$。

FEPCS3-1：顺序迁移演化，即将一串顺序执行的服务操作中的一个位置移动，迁移完成后服务操作序列仍然顺序执行，如图 7 所示。



操作序列 c　　操作序列 c'

图 7　FEPCS3-1：服务操作顺序迁移演化

服务操作顺序迁移前的序列 c 表示为：

$A=p@S_i.[\theta]\bar{x}.0$

$B=[\zeta]x.q@S_j.[\xi]\bar{y}.0$

$C=[\psi]y.r@S_k$

即　$p@S_i.[\theta]\bar{x}.[\zeta]x.q@S_j.[\xi]\bar{y}.[\psi]y.r@S_k$

其中 mc 为：$D=p@S_i$。

服务操作顺序迁移后的序列 c' 表示为：

$B=q@S_j.[\sigma]\bar{u}.0$

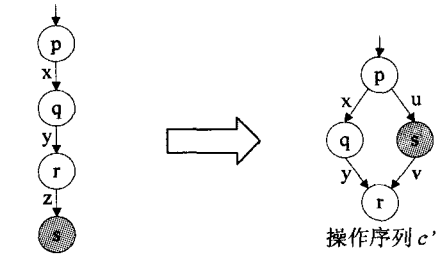
$A=[\varsigma]u.p@S_i.[\eta]\bar{v}.0$

$C=[\mu]v.r@S_k$

即　$q@S_j.[\sigma]\bar{u}.[\varsigma]u.p@S_i.[\eta]\bar{v}.[\mu]v.r@S_k$

由上可见，服务操作顺序迁移的主要改变在于消息发送接收条件的更新及信道的更改。显然，顺序迁移可以由 FEPCS2 的删除和 FEPCS1-1 的顺序插入联合实现，即在服务操作原有位置执行删除，再在新位置执行插入即可。

FEPCS3-2：并行迁移演化，即将原本顺序执行的某一服务操作移动，使它与其他操作并行执行，如图 8 所示。



操作序列 c　　操作序列 c'

图 8　FEPCS3-2：服务操作并行迁移演化

服务操作并行迁移前的序列 c 表示为：

$A=p@S_i.[\theta]\bar{x}.0$

$B=[\zeta]x.q@S_j.[\xi]\bar{y}.0$

$C=[\psi]y.r@S_k.[\alpha]\bar{z}.0$

$D=[\beta]z.s@S_l$

即　$p@S_i.[\theta]\bar{x}.[\zeta]x.q@S_j.[\xi]\bar{y}.[\psi]y.r@S_k.[\alpha]\bar{z}.[\beta]z.s@S_l$

其中 mc 为：$E=s@S_l$。

服务操作并行迁移后的序列 c' 表示为：

$A=p@S_i.([\theta]\bar{x}|[\eta]\bar{u}).0$

$B=[\zeta]x.q@S_j.[\xi]\bar{y}.0$

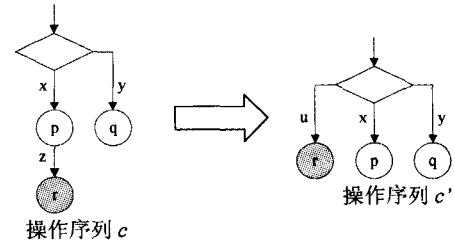
$D=[\mu]u.s@S_l.[\sigma]\bar{v}.0$

$C=([\psi]y|[\varsigma]v).r@S_k)$

即：$p@S_i.([\theta]\bar{x}|[\eta]\bar{u}).(([\zeta]x.q@S_j.[\xi]\bar{y})|([\mu]u.s@S_l.[\sigma]\bar{v})).([\psi]y|[\varsigma]v).r@S_k$

由上可见，并行迁移服务操作可以由 FEPCS2 的删除和 FEPCS1-2 的并行插入联合实现，即在服务操作原有位置将其删除，再在新位置插入已删除的服务操作即可。

FEPCS3-3：条件迁移演化，即将服务操作序列中的某一操作移动到条件选择分支结构中，使其成为可能执行的操作之一，如图 9 所示。



操作序列 c　　操作序列 c'

图 9　FEPCS3-3：服务操作条件迁移演化

服务操作条件迁移前的序列 c 表示为：

$A=[\theta]x.[\psi]p@S_i.[\alpha]\bar{z}.0$

$B=[\eta]y.[\mu]q@S_j$

$C=[\beta]z.r@S_k$

即　$([\theta]x.[\psi]p@S_i.[\alpha]\bar{z}.[\beta]z.r@S_k)+[\eta]y.[\mu]q@S_j$

其中 mc 为：$D=r@S_k$。

服务操作条件迁移后的序列 c' 表示为：$A=[\theta]x.[\psi]p@S_i$

$B=[\eta]y.[\mu]q@S_j$

$C=[\xi]u.[\zeta]r@S_k$

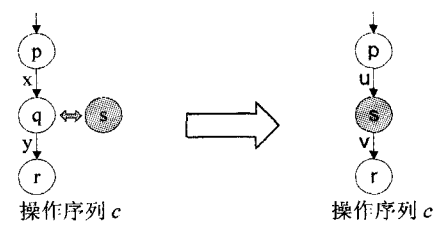
即　$[\xi]u.[\zeta]r@S_k+[\theta]x.[\psi]p@S_i+[\eta]y.[\mu]q@S_j$

由上表可见，条件迁移服务操作可以由 FEPCS2 的删除和 FEPCS1-3 的条件插入联合实现，即在服务操作原有位置将其删除，再在新位置加入带有控制条件的原有服务操作即可。

虽然迁移服务操作可以由删除服务操作与插入服务操作联合实现，但是我们是在抽象层面上考虑可适应服务组合，仍有必要将迁移服务操作单独提出，特别是其中包含的通信信道的更改与条件控制符的更新有其特别之处。

4.4　服务操作替换演化

当一个服务操作不再符合客户需求时，必然要求一个新的服务操作来替换它。设 oc 为操作序列 c 中的一个服务操作，现在要用新的服务操作 nc 从操作序列 c 中替换 oc，即 $c \xrightarrow{replace(oc,nc)} c'$。替换服务操作具体如图 10 所示。



操作序列 c　　操作序列 c'

图 10　FEPCS4：服务操作替换演化

服务操作替换前的序列 c 用 CPi 演算表示为：

$A=p@S_i.[\theta]\overline{x}.0$

$B=[\psi]x.q@S_j.[\xi]\overline{y}.0$

$C=[\zeta]y.r@S_k$

即 $p@S_i.[\theta]\overline{x}.[\psi]x.q@S_j.[\xi]\overline{y}.[\zeta]y.r@S_k$

其中 oc 为：$D=q@S_j$；nc 为：$E=s@S_l$。

服务操作替换后的序列 c' 用 CPi 演算表示为：

$A=p@S_i.[\alpha]\overline{u}.0$

$B=[\beta]u.s@S_l.[\sigma]\overline{v}.0$

$C=[\varsigma]v.r@S_k$

即 $p@S_i.[\alpha]\overline{u}.[\beta]u.s@S_l.[\sigma]\overline{v}.[\varsigma]v.r@S_k$

要用新的服务操作替换不再适用的操作时，需要对原有的通信信道及通信控制条件进行更新。显然，替换服务操作可由 FEPCS2 的删除和 FEPCS1-1 的顺序插入联合实现。

4.5 服务操作交换演化

服务操作序列中的服务操作的先后顺序并不是一成不变的，需要根据客户的需求和服务演化的需要做出不断的调整，交换服务操作就是将序列中的两个服务操作互换位置。设 sc，tc 为操作序列 c 中的两个服务操作，要在操作序列中交换 sc 和 tc，即 $c \xrightarrow{swap(sc,tc)} c'$。交换服务操作具体如图 11 所示。

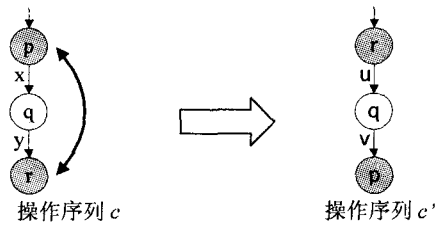


图 11 FEPCS5：服务操作交换演化

服务操作交换前的序列 c 用 CPi 演算表示为：

$A=p@S_i.[\theta]\overline{x}.0$

$B=[\psi]x.q@S_j.[\xi]\overline{y}.0$

$C=[\zeta]y.r@S_k$

即 $p@S_i.[\theta]\overline{x}.[\psi]x.q@S_j.[\xi]\overline{y}.[\zeta]y.r@S_k$

其中 sc 为：$D=p@S_i$；tc 为：$E=r@S_k$。

服务操作交换后的序列 c' 用 CPi 演算表示为：

$A=r@S_k.[\alpha]\overline{u}.0$

$B=[\beta]u.q@S_j.[\sigma]\overline{v}.0$

$C=[\varsigma]v.p@S_i$

即 $r@S_k.[\alpha]\overline{u}.[\beta]u.q@S_j.[\sigma]\overline{v}.[\varsigma]v.p@S_i$

交换服务操作时，原有操作序列中的服务操作不变，只是先后次序流程不同，但是交换前后的通信信道和控制条件都需要重新建立。显然，交换服务操作可以由两个 FEPCS3-1 的顺序迁移联合完成。

4.6 服务操作并行演化

组合服务的流程应能够根据需求不断调整，反映到操作序列层面就是其相互依赖关系发生改变。在组合服务的某段操作序列中可能限制某些操作顺序执行，但随着需求的变化限制不再成立时，这些操作应该并行执行。服务操作并相互就是使操作序列中原来顺序执行的服务操作变为并行执行。设 ac，bc，dc 为操作序列 c 中的 3 个服务操作，要在操作序列使其并行化，即 $c \xrightarrow{parallelize(ac,bc,dc)} c'$，具体如图 12 所示。

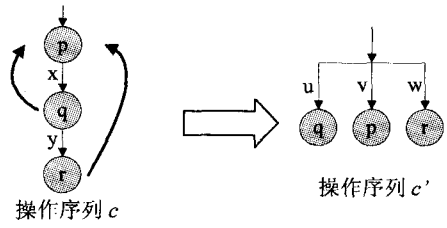


图 12 FEPCS6：服务操作并行演化

服务操作并行化前的序列 c 用 CPi 演算表示为：

$A=p@S_i.[\theta]\overline{x}.0$

$B=[\psi]x.q@S_j.[\xi]\overline{y}.0$

$C=[\zeta]y.r@S_k$

即 $p@S_i.[\theta]\overline{x}.[\psi]x.q@S_j.[\xi]\overline{y}.[\zeta]y.r@S_k$

其中 ac 为：$D=p@S_i$；bc 为：$E=q@S_j$；dc 为：$F=r@S_k$。

服务操作并行化后的序列 c' 用 CPi 演算表示为：

$A=[\alpha]v.p@S_i$

$B=[\beta]u.q@S_j$

$C=[\gamma]w.r@S_k$

即 $[\alpha]v.p@S_i|[\beta]u.q@S_j|[\gamma]w.r@S_k$

服务操作并行化时，原有操作序列中的服务操作不变，只是由顺序执行变为并行执行，服务操作之间不再存在相互依赖。当然，并不是所有服务操作都能够并行化，对于可并行化的条件进行讨论超出了本文的范围。

4.7 增加依赖演化

前面的 FEPCS 主要针对操作序列中的服务操作进行柔性演化，而服务流程的改变也有单纯针对依赖关系而发生的演化。假设某一服务操作因为需求的变化而需要另外接收另一服务操作发出的消息才能执行，这时就形成了增加依赖的情景。设 ac、bc 为操作序列 c 中的两个服务操作，且在 ac、bc 间不存在直接的消息传递，设要增加的通信信道用 nt 表示，则要在 ac、bc 间增加依赖 nt，可表示为 $c \xrightarrow{addDepndency(ac,bc,nt)} c'$，具体如图 13 所示。

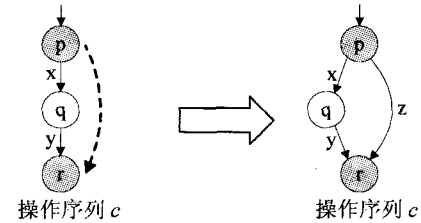


图 13 FEPCS7：增加依赖演化

服务操作增加依赖前的序列 c 表示为：

$A=p@S_i.[\theta]\overline{x}.0$

$B=[\psi]x.q@S_j.[\xi]\overline{y}.0$

$C=[\zeta]y.r@S_k$

即 $p@S_i.[\theta]\overline{x}.[\psi]x.q@S_j.[\xi]\overline{y}.[\zeta]y.r@S_k$

其中 ac 为：$D=p@S_i$；bc 为：$E=r@S_k$；nt 为：z。

服务操作增加依赖后的序列 c' 表示为：

$A=p@S_i.([\theta]\overline{x}|[\alpha]\overline{z}).0$

$B=[\psi]x.q@S_j.[\xi]\overline{y}.0$

$C=([\zeta]y|[\beta]z).r@S_k$

即 $p@S_i.([\theta]\overline{x}|[\alpha]\overline{z}).$

$([\psi]x.q@S_j.[\xi]\overline{y}.[\zeta]y|[\beta]z).r@S_k$

增加依赖时，原有操作序列中的服务操作保持不变，而是相互之间的依赖关系发生了改变。增加依赖模式针对信道的建立而进行，建立依赖关系的两个服务操作需要对消息的发送和接收做出可适应更新。

4.8 去除依赖演化

显然，FEPCS8 与 FEPCS7 中的增加依赖是互逆的，是去除操作序列中两个服务操作间已存在的依赖，同时进行相应的可适应变化。假设某一服务操作因为需求的变化而不再需要接收另一服务操作发出的消息才能执行，就可以将此依赖去除。设 ac、bc 为操作序列 c 中的两个服务操作，且在 ac、bc 间存在通信信道 rt 表示，则要在 ac、bc 间去除依赖 rt，可表示为 $c \xrightarrow{removeDepndency(ac,bc,rt)} c'$，具体如图 14 所示。

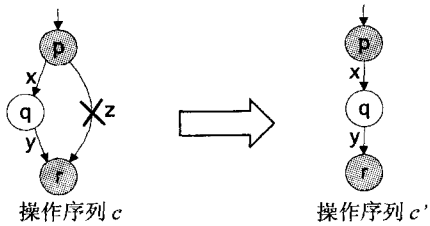


图 14 FEPCS8：去除依赖演化

服务操作去除依赖前的序列 c 表示为：

$A=p@S_i.([\theta]\overline{x}\mid[\alpha]\overline{z}).0$

$B=[\psi]x.q@S_j.[\xi]\overline{y}.0$

$C=([\zeta]y\mid[\beta]z).r@S_k$

即：$p@S_i.([\theta]\overline{x}\mid[\alpha]\overline{z}).$

$([\psi]x.q@S_j.[\xi]\overline{y}.[\zeta]y\mid[\beta]z).r@S_k$

其中 ac 为：$D=p@S_i$；bc 为：$E=r@S_k$；rt 为：z。

服务操作去除依赖后的序列 c' 表示为：

$A=p@S_i.[\theta]\overline{x}.0$

$B=[\psi]x.q@S_j.[\xi]\overline{y}.0$

$C=[\zeta]y.r@S_k$

即 $p@S_i.[\theta]\overline{x}.[\psi]x.q@S_j.[\xi]\overline{y}.[\zeta]y.r@S_k$

去除依赖时，原有操作序列中的服务操作保持不变，而是相互之间的依赖关系发生了改变。去除依赖模式针对信道的消除而进行，消除依赖关系的两个服务操作需要删除相应的消息发送和接收。

4.9 更新条件演化

在组合服务控制流程中存在很多条件转移语句，而这些条件总是在随着业务需求的变化不断进行更新。假设服务操作序列中的某个条件选择分支分别执行服务操作 ac 或者 bc，而它们能够执行的条件分别为 lex 或者 lex' 为真。当需要进行分支条件更新时就是更新条件 lex 和 lex'，可表示为 $c \xrightarrow{updateCondition([lex]ac,[lex']bc)} c'$，具体如图 15 所示。

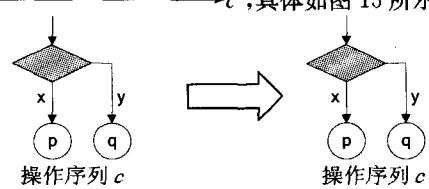


图 15 FEPCS9：更新条件演化

服务操作更新条件前的序列 c 表示为：

$A=[\theta]x.[\alpha]p@S_i$

$B=y(n).[\psi]q@S_j$

即 $[\theta]x.[\alpha]p@S_i+[\psi]y.[\beta]q@S_j$

其中 $[lex]ac$ 为：$C=[\alpha]p@S_i$；$[lex']bc$ 为：$D=[\beta]q@S_j$。

服务操作更新条件后的序列 c' 表示为：

$A=[\theta]x.[\xi]p@S_i$

$B=y(n).[\zeta]q@S_j$

即 $[\theta]x.[\xi]p@S_i+[\psi]y.[\zeta]q@S_j$

服务操作序列进行条件更新时，原有服务操作保持不变，仅是选择分支的转移条件发生了改变。

4.10 嵌入选择分支演化

组合服务的流程总是在不断变更，有时需要根据业务需求为某些服务操作的执行增加判定条件，即根据约束条件来选择那些操作应该执行。假设操作序列 c 中已有服务操作 ac、bc，现在需要为其是否执行增加判定条件，则可表示为 $c \xrightarrow{embedCondition(ac,bc)} c'$，具体如图 16 所示。

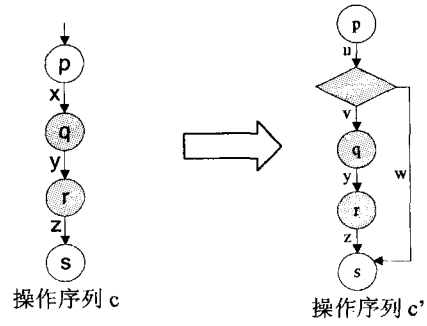


图 16 FEPCS10：嵌入选择分支演化

服务操作嵌入选择分支前的序列 c 表示为：

$A=p@S_i.[\alpha]\overline{x}.0$

$B=[\beta]x.q@S_j.[\xi]\overline{y}.0$

$C=[\zeta]y.r@S_k.[\eta]\overline{z}.0$

$D=[\mu]z.s@S_l$

即 $p@S_i.[\alpha]\overline{x}.[\beta]x.q@S_j.[\xi]\overline{y}.[\zeta]y.r@S_k.[\eta]\overline{z}.$
$[\mu]z.s@S_l$

其中 ac 为：$E=q@S_j$；bc 为：$F=r@S_k$。

服务操作嵌入选择分支后的序列 c' 表示为：

$A=p@S_i[\sigma]\overline{u}.O$

$B=[\varsigma]v.[\theta]q@S_j.[\xi]\overline{y}.0$

$C=[\zeta]y.[\theta]r@S_k.[\eta]\overline{z}.0$

$D=([\mu]z+[\delta]w).s@S_l$

即 $p@S_i.[\sigma]\overline{u}.([\varsigma]v.[\theta]q@S_j.[\xi]\overline{y}.[\zeta]y.[\theta]r@$
$S_k.[\eta]\overline{z}.[\mu]z+[\delta]w).s@S_l$

以上描述了如何将服务操作嵌入到可以选择执行或者不执行的分支中去，用 CPi 演算进行描述就是当服务操作控制条件为真时执行操作，当服务操作控制条件为假时则绕过。如果服务操作控制条件为真，则和改变前的服务操作序列一致。FEPCS1-3 中的服务操作条件插入与以上所述的嵌入选择分支存在明显不同：服务操作条件插入的重点在于插入，插入的操作可以执行也可以不执行；而嵌入选择分支模式中没有新增加的操作，而是原有操作根据条件选择执行或者不执行。需要注意的是，无论嵌入到同一选择分支中去的服务操作是几个，都要受到同一个的操作控制条件的约束，即同一选择分支中的所有操作要么都执行，要么都不执行，例如以上所示的 $[\theta]$。

4.11 嵌入循环演化

组合服务控制流程随业务需求不断变化，有些服务操作本来仅执行一次，但现在需要不断执行直到满足指定条件为止，也就是将某些操作改为循环执行。假设服务操作序列 c 中有操作 ac、bc，现在需要根据需求将其不断执行直到满足特定条件为止，可表示为 $c \xrightarrow{embedLoop(ac,bc)} c'$，具体如图 17 所示。

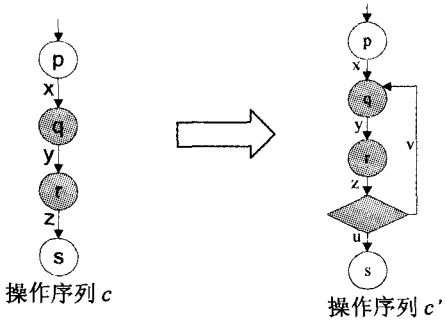


图 17 FEPCS11：嵌入循环演化

服务操作嵌入循环前的序列 c 表示为：

$A=p@S_i.[\alpha]\bar{x}.0$

$B=[\beta]x.q@S_j.[\xi]\bar{y}.0$

$C=[\zeta]y.r@S_k.[\eta]\bar{z}.0$

$D=[\mu]z.s@S_l$

即 $p@S_i.[\alpha]\bar{x}.[\beta]x.q@S_j.[\xi]\bar{y}.[\zeta]y.r@S_k.[\eta]\bar{z}.[\mu]z.s@S_l$

其中 ac 为：$E=q@S_j$；bc 为：$F=r@S_k$。

服务操作嵌入循环后的序列 c' 表示为：

$A=p@S_i.[\alpha]\bar{x}.0$

$B=([\beta]x+![\sigma]v).![\varsigma]q@S_j.![\xi]\bar{y}.0$

$C=!([\zeta]y.[\varsigma]r@S_k.[\eta]\bar{z}).0$

$D=[\theta]u.s@S_l$

即 $p@S_i.[\alpha]\bar{x}.(([\beta]x+![\sigma]v).!([\varsigma]q@S_j.[\xi]\bar{y}.[\zeta]y.[\varsigma]r@S_k.[\eta]\bar{z})).[\theta]u.s@S_l$

要将服务操作嵌入循环中，想要增加循环终止的判定条件，也就是在 FEPCS10 中的嵌入选择分支的基础上给服务操作增加复制操作符，嵌入到循环中的服务操作受到同样的终止判定条件约束，如上例中的 $[\varsigma]$。根据判定条件的不同，所嵌入的服务操作执行次数也就不同，如果仅执行一次则与嵌入循环前的服务操作序列相同。

组合服务流程的变化有很多种，本节我们以所提出的基于 CPi 演算的柔性演化模式解析了主要的变化情景及其相应的适应动作。虽然复杂的 FEPCS 可以由数个简单的 FEPCS 联合实现，但是过程复杂且失去了广泛性，因此在不过分重复的前提下我们提出了 11 种具有代表性的柔性演化模式。

5 相关工作

服务组合自适应的研究正受到越来越多的关注，已有的研究工作可以分为静态自适应和动态自适应。静态自适应方面，Chang 等提出了一种面向服务的分析设计方法用来为潜在的客户群开发自适应的组合服务[1]；Liu 等提出了一种面向服务企业中便于管理自上而下变化的框架[2]。他们的工作主要集中于架构层次的设想和方法论，而没有涉及具体的组合服务如何应对各种变化，本文主要针对组合服务的各种演化进行了形式化描述。面向方面的软件开发技术能够处理静态和动态自适应问题，Erradi 等提出使用方面编织技术提供了组合服务运行时自适应和个性化[3]。他们的工作主要在于技术层面，而本文是将组合服务的实例演化进行形式化。

使用进程演算对组合服务进行分析和验证的研究有很多，其中基于 Pi 演算的研究较为领先，如廖军[15]、邓水光[16]等从不同角度做了出色的研究。但是他们的工作都集中在组合服务的验证，而对组合服务的变化和自适应缺少分析，本文使用扩充的 Pi 演算对组合服务的各种变化进行详细的分析。Puhlmann 等提出了一种使用 Pi 演算来形式化工作流的方法[17]，本文使用的扩充 Pi 演算能够更加清晰地描述组合服务的演化。本文的灵感来源于文献[14]，Weber 等用图形化和自然语言的方式给出了进程敏感信息系统的变化模式，而本文针对组合服务自适应问题，使用形式化方法对各种变化和适应情景进行了详细的分析。

结束语 本文提出了一种基于条件 Pi 演算的组合服务柔性演化模型。我们对经典的 Pi 演算进行了扩展，增加了归属操作符和条件控制符，使之更适合于 Web 服务的建模。正是由于引入了条件控制符，使得 CPi 演算与 ECA 模式能够良好地结合起来，提出了一种描述组合服务流程的方法。从分析组合服务变化的场景出发，提出了 11 种主要的组合服务柔性演化模式，对每种模式进行了分析。未来的工作主要是构建模式匹配原型系统来使得我们提出的柔性演化模型切实地解决组合服务的自适应问题。

参 考 文 献

[1] Chang S H, Kim S D. A Service-Oriented Analysis and Design Approach to Developing Adaptable Services[C]// Proceedings of the 5th International Conference on Services Computing. 2007: 204-211

[2] Liu X, Bouguettaya A. Managing Top-down Changes in Service-Oriented Enterprises[C]// Proceedings of the 5th International Conference on Web Services. 2007: 1072-1079

[3] Erradi A, Maheshwari P, Padmanabhuni S. Towards a Policy-Driven Framework for Adaptive Web Services Composition[C]// Proceedings of the 1st International Conference on Next Generation Web Service Practices. 2005: 33-38

[4] Aalst W M P. van der, Basten T. Inheritance of Workflows: An Approach to Tackling Problems Related to Change[J]. Theoretical Computer Science, 2002, 270(1/2): 125-203

[5] Rinderle S, Reichert M, Dadam P. Flexible Support of Team Processes by Adaptive Workflow Systems[J]. Distributed and Parallel Databases, 2004, 16(1): 91-116

[6] Papazoglou M. The Challenges of Service Evolution[C]// Proceedings of the 20th international conference on Advanced Information Systems Engineering. 2008: 1-15

[7] Milner R. Communicating and Mobile Systems: The Pi Calculus [M]. Cambridge: Cambridge University Press, 1999

[8] Sangiorgi D, Walker D. The pi-calculus: a Theory of Mobile Processes[M]. New York: Cambridge University Press, 2003

[9] Zhou J, Zeng G. A mechanism for grid service composition behavior specification and verification[J]. Future Generation Computer Systems, 2009, 25(3): 378-383

[10] 周静，曾国荪. 基于 CPi-calculus 的网格服务行为研究[J]. 计算

机科学,2007,34(6):13-18
[11] Dayal U, Hsu M, Ladin R. Organizing long-running activities with triggers and transactions[J]. ACM SIGMOD Record, 1990,19(2):214
[12] Huebscher M C, McCann J A. A survey of autonomic computing-degrees, models, and applications[J]. ACM Computing Surveys(CSUR),2008,40(3):7
[13] Papazoglou M P, Traverso P, Dustdar S. Service-oriented computing: State of the art and research challenges[Z]. Computer, 2007:38-45
[14] Ardagna D, Comuzzi M, Mussi E. Paws: A framework for executing adaptive Web-service processes[Z]. IEEE SOFTWARE, 2007:39-46
[15] Weber B, Reichert M, Rinderle-Ma S. Change patterns and change support features-enhancing flexibility in process-aware information systems[J]. Data & knowledge engineering, 2008, 66(3):438-466
[16] 廖军,谭浩,刘锦德. 基于 Pi-演算的 Web 服务组合的描述和验证[J]. 计算机学报,2005,33(4),635-643
[17] 邓水光. Web 服务自动组合与形式化验证的研究[D]. 杭州:浙江大学,2007
[18] Puhlmann F, Weske M. Using the pi-calculus for formalizing workflow patterns[C] // Proceedings of Third International Conference on Business Process Management. 2005:153-168

(上接第 203 页)

3)系统的安全性较高,不会造成带宽瓶颈和单点故障。

缺点主要有:

1)需要在主机上增加代理软件;

2)数据访问的安全性难以控制。

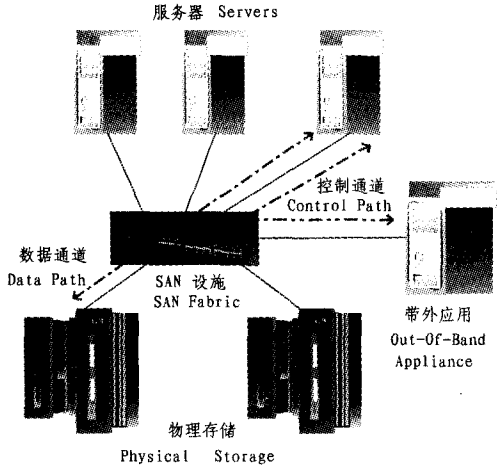


图 4 带外虚拟存储模型

4 虚拟存储技术比较分析

结合以上所介绍的虚拟存储技术原理及优缺点,我们从以下方面对其进行对比,如表 1 所列,并根据对比结果进行分析。

表 1 虚拟存储技术比较

	基于主机端	基于存储设备	对称式	非对称式
通用性	否	否	是	是
单节点上可管理性	不支持	不支持	支持	支持
服务器 OS 独立性	否	是	是	是
文件系统独立性	否	是	是	是
存储子系统独立性	是	否	是	是
复杂性	简单	简单	复杂	复杂
存储池	不支持	支持	支持	支持
性能	较高	较高	较低	较高
SAN 扩展性	低	低	低	高
安全性	低	高	高	高

从以上对比不难看出,从通用性、可扩展性、独立性、复杂性和安全性等方面考虑,对称式虚拟存储和非对称式虚拟存储技术表现均较为良好;而在性能方面,非对称式虚拟存储比对称式虚拟存储性能更良好。

通过以上分析,我们得出以下结论:对于低端用户实现虚拟存储最好的方式是基于主机端的虚拟存储,该种架构虽然扩展性和安全性都不是很好,但其实施简单,能耗及成本都较低,足够满足低端用户对于存储的需求;对于高级用户或者企业实现虚拟存储最好的方式就是通过非对称式架构使用 SAN 应用和代理的结合。这种方法在可扩展性、性能、可靠性等方面具有明显优势,不仅能够满足高端用户海量数据存储的需求,也保持了较低的成本和能耗。

结束语 本文主要介绍了 3 种虚拟存储技术方式的实现及其优缺点,并从通用性、独立性、复杂性、扩展性和安全性等方面对其进行了比较,通过分析得出了对于不同需求用户应选择何种虚拟存储方式的结论。虚拟存储技术在实际应用中也存在一些局限,例如与其结合的应用服务多面向高端用户,不同厂商产品、用户技术标准的不统一为其实施带来更大的困难等,这些问题的解决还有待我们进一步的研究。总而言之,虚拟存储技术在不断的研究和发展中,一定会体现出越来越大的价值。

参 考 文 献

[1] 郑纬民,汤志忠. 计算机系统结构(第二版)[M]. 北京:清华大学出版社,2000:98-144
[2] 张成峰,谢长生,罗益辉,等. 网络存储的统一与虚拟化[J]. 计算机科学,2006,33(6):11-14
[3] 陈其铭,张宇,林荣. 虚拟存储技术及现状分析[J]. Computer Knowledge And Technology,5(2):452-453,473
[4] 柏新才,吴学智,马宁. 虚拟存储技术及应用分析[J]. Ship Electronic Engineering,2008,1:761-961
[5] High Performance Storage Virtualization Architecture[R]. A Store White Paper, http://www.store-age.com
[6] 谭生龙. 存储虚拟化技术的研究[J]. 北京:微计算机应用,2010,31(1):33-38
[7] 康潇文,杨英杰,杜鑫. 基于虚拟存储的数据容灾关键技术研究[J]. 成都:计算机应用研究,2009,26(7):2603-2606
[8] 徐梦,宗坤. 虚拟存储技术的研究和应用[J]. 安徽:计算机与信息技术,2010,1:37-38
[9] Gibson G A, Meter R V. Network-attached storage architecture [J]. Communications of the ACM,2000,43(11):37-45