

一种高效的 Web 服务迁移算法*

胡 坤, 刘绍华, 钟 华, 魏 峻

(中国科学院 软件研究所 软件工程技术中心, 北京 100080)

摘 要: 为了满足 Web 服务动态分布的需求, 并保证 Web 服务迁移的一致性, 提出一种基于 Broker 的 Web 服务迁移算法 WS Broker。其在保证迁移一致性的基础上, 提高了迁移效率。另外, 通过分析客户请求之间的依赖关系, 提前执行了某些阻塞的客户请求, 缩短了服务中断时间。实验结果表明, WS Broker 在保证迁移一致性的基础上有效地提高了 Web 服务迁移的效率。

关键词: Web 服务; 服务迁移算法; 迁移一致性; 依赖关系

中图分类号: TP393 文献标志码: A 文章编号: 1001-3695(2007)04-0064-04

Efficient Web Services Migration Algorithm

HU Kun, LIU Shao-hua, ZHONG Hua, WEI Jun

(Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing 100080, China)

Abstract: To meet the requirement of Web services' dynamic distribution and maintain the consistency of Web services migration was provided. A Broker-based Web services migration algorithm(WS Broker) was proposed. WS Broker guarantees the migration consistency and improves the migration performance. In addition, by analyzing the dependency relationship among the requests interrupted by the migration, WS Broker could process some of them in advance, which greatly reduces the total latency of request processing caused by the migration. The result of the test shows that WS Broker improves the performance of Web services migration efficiently.

Key words: web services; service migration algorithm; migration consistency; dependency relationship

0 引言

Web 服务以 XML、SOAP、WSDL 和 UDDI 为核心, 支持开放、动态的互操作模式; 具有良好的封装性和强大的集成能力; 可极大地降低系统集成的复杂性和开销, 因此获得了产业界的广泛支持和学术界的重视^[1]。尤其是 UDDI 的出现, 使 Web 服务可以被动态地发现和绑定。但是在 UDDI 将代表 Web 服务的 URL 发送给客户端, 客户端利用 URL 与 Web 服务建立连接后, 它们之间的连接却是静态的。这种静态结构限制了 Web 服务在运行时适应其执行环境变化的能力, 无法满足在任意运行时刻动态更换运行环境的需求。为了满足 Web 服务动态分布的需求, 需要实现 Web 服务的迁移。

实现 Web 服务的迁移, 具有以下优点: Web 服务在运行期间从高负载的服务器迁移到低负载的服务器, 可以提高系统的整体性能和吞吐量。将 Web 服务迁移到被访问数据所在服务器, 可以降低网络访问开销。服务器需要定期关机维护。维护期间, 可以将 Web 服务迁移到其他服务器。当源服务器可用时, 再将 Web 服务迁移回来, 可以提高 Web 服务的可用性。

Web 服务可以分为无状态和有状态两类。无状态 Web 服务对于任意客户端在任意时刻发出的请求, 总能返回同一结果, 如报道天气预报的 Web 服务。无状态 Web 服务不具有运

行时状态, 迁移实现相对较简单, 不是本文研究的重点。有状态 Web 服务对于同一客户端发出的请求, 返回的结果可能依赖于该客户端的上一次请求, 如电子商务中提供购物车服务的 Web 服务。因此, 服务器需要为访问有状态 Web 服务的每个客户端维护一个状态信息。与客户端对应的 Web 服务状态信息在处理客户端第一次请求时被创建, 在处理完所有相应的请求后被销毁。有状态 Web 服务迁移就是将与它相关的某些或全部 Web 服务状态信息从一台服务器迁移到另一台服务器。本文重点研究有状态 Web 服务的迁移。下面提到的 Web 服务迁移表示的是有状态 Web 服务的迁移。

目前, 研究人员已经提出了很多 Web 服务迁移算法。但是研究内容主要集中在如何对 Web 服务进行迁移, 很少研究如何保证 Web 服务迁移的一致性。然而, 保证 Web 服务迁移一致性, 对保证 Web 服务的可用性至关重要。尤其在当今电子商务已经成为一些企业的主要收入来源, 在对代表企业关键业务的 Web 服务进行迁移时, 保证 Web 服务迁移前后的一致性, 可以减少由于服务不可用造成的交易失败, 从而为企业赢得更多利润并保持良好形象。保证 Web 服务迁移一致性, 需要满足状态一致性、引用一致性和消息一致性三个约束。

本文提出的 Web 服务迁移算法 WS Broker 可以保证 Web 服务迁移的一致性。在保证一致性的基础上, 本文提出了两种改进迁移效率的策略: 利用基于 Broker 的迁移算法, 减少了

收稿日期: 2006-02-16; 修返日期: 2006-04-23 基金项目: 国家“863”计划资助项目(2005AA112030)

作者简介: 胡坤(1981-), 男, 山东嘉祥人, 硕士研究生, 主要研究方向为网络分布式计算、软件工程技术(hukun@otcaix.iscas.ac.cn); 刘绍华(1976-), 男, 江西丰城人, 助理研究员, 博士, 主要研究方向为网络分布式计算、 workflow 技术、服务协作理论、中间件; 钟华(1971-), 男, 研究员, 博士, 主要研究方向为网络分布式计算、软件工程技术; 魏峻(1970-), 男, 研究员, 博士, 主要研究方向为软件工程、软件理论、网络分布式计算。

迁移过程中 Web 服务中断时间, 降低了迁移后 Web 服务对源服务器的依赖; 通过分析客户请求之间的依赖关系, 减少了客户请求的阻塞时间, 进一步缩短了服务中断时间。实验结果表明, WS Broker 可以有效地提高 Web 服务迁移的效率。在下文中, 将 Web 服务迁移前的服务器称为源服务器, 迁移后的服务器称为目标服务器。

1 相关工作

目前, 关于 Web 服务迁移的研究很多, 但还没有一种有效的方法来保证 Web 服务迁移的一致性。文献[2] 提出的一种端对端的 Web 服务迁移框架 LAMS, 可以帮助移动用户通过 WAP 协议实现 Web 服务迁移; 但是 LAMS 不能迁移 Web 服务的运行时状态, 只能将 Web 服务重新部署到目标服务器, 这显然不能保证迁移一致性。文献[3] 在 .NET 框架之上, 利用面向方面的编程(AOP) 技术实现了专门负责对象和进程迁移的中间件; 这个中间件只可以在迁移时间非常短的情况下保证迁移一致性, 具有很大的局限性。实际上, Web 服务迁移通常需要经过网络传输, 迁移时间是不能被轻易忽略的。文献[4] 利用基于服务定义的逻辑检查点技术, 提高了 Web 服务迁移的效率, 但是忽略了在迁移过程中收到的客户请求, 因此它也不能保证迁移一致性。文献[5] 重点研究如何利用 P2P 技术查找迁移后的 Web 服务, 并不保证迁移一致性。文献[6] 中把 EJB 组件分为无状态和有状态两类。为了保证有状态 EJB 组件的迁移一致性, 在迁移前, 需要让其进入安全状态, 停止处理客户请求。这种方法虽然保证了一致性, 但增加了服务中断时间。文献[7] 提出的 Web Session 迁移算法, 在迁移过程中不允许改变源服务器上的 Web Session 状态, 否则不能保证迁移一致性。文献[8] 提出一种基于操作系统实现的 TCP Session 迁移机制。该算法虽然利用基于日志的回溯方法保证了 TCP Session 的迁移一致性, 但是需要在操作系统级别对迁移进行支持, 所以不具有通用性。Web 服务与 EJB 和 Web Session 相比, 具有很多相似性, 但是也有自己的特点。例如, Web 服务不包含真正业务逻辑的实现, 所以 Web 服务不可照搬它们的迁移方法。

文献[9, 10] 介绍了进程迁移算法。Total Copy 是最早使用的进程迁移算法, 实现相对较简单。该算法虽然保证了进程迁移的一致性, 但是在迁移过程中, 进程挂起时间过长, 容易导致客户请求因超时而得不到处理。为了解决这个问题, Theime 在 V 操作系统中提出了 Pre-Copy 算法。Pre-Copy 算法缩短了进程挂起时间, 但是为了保证迁移一致性, 需要再次迁移不一致的进程状态, 增加了迁移成本。Demand Page Algorithm 算法只迁移进程在目标机器执行所需要的最少状态信息, 当进程需要时, 再迁移其他状态信息。该算法虽然降低了进程迁移的时间, 但是产生了 Residual Dependency^[10] 问题。与进程相比, Web 服务具有迁移粒度小, 不需要迁移整个进程状态的特点。所以 Web 服务也不可照搬进程迁移算法。

2 迁移算法

2.1 迁移模型

Web 服务迁移是把 Web 服务的状态信息从源服务器迁移

到目标服务器。在迁移过程中, 首先在源服务器中提取 Web 服务的状态信息, 然后将状态信息从源服务器传送到目标服务器, 最后在目标服务器创建 Web 服务, 恢复其状态。每个 Web 服务的状态信息包含两部分内容: 服务描述信息, 包括客户端可以访问的 Web 服务操作、Web 服务对业务逻辑实现的引用等信息。这部分信息是由服务提供方指定的, 运行期间不会发生改变。运行时状态, 也称为服务上下文(Service Context)。它是在客户端与服务实现之间传播的对话性质的或者上下文性质的信息, 如某种事务 ID 或会话 ID。

Web 服务迁移可以只把 Web 服务的一部分状态信息迁移到目标服务器, 仍保留源服务器中的 Web 服务让其为其他客户端继续提供服务, 也可以将 Web 服务的所有状态信息迁移到目标服务器, 然后删除源服务器中的 Web 服务。为了实现 Web 服务迁移, Web 服务的状态信息都必须是可以被序列化的。也就是说, 源服务器必须能够把 Web 服务状态信息序列化到字节流中; 目标服务器必须能够从字节流中反序列化出 Web 服务的状态信息。本文只研究 Java 语言环境中的 Web 服务迁移。Java 语言通过 Serializable 接口支持对象序列化, 只要代表 Web 服务状态信息的 Java 对象实现 Serializable 接口, Java 虚拟机就可以对其进行序列化和反序列化。

Web 服务迁移需要 Web 服务运行时环境的支持。Web 服务运行时环境是指在运行时对 Web 服务进行管理的软件, 如 AXIS 1.2^[11]、OnceAS^[12] 等。通过直接修改 Web 服务运行时环境的源代码, 添加对 Web 服务迁移的支持。该方法虽然比较简单, 但是对于不同的 Web 服务运行时环境都要重新修改, 使工作量巨大, 无法在实际中得到广泛应用。本文利用 AOP 编程技术, 可以在不修改 Web 服务运行时环境源代码的基础上, 实现对 Web 服务迁移的支持。如图 1 所示, Migration Server 既负责把 Web 服务迁移到目标服务器, 也负责接收迁移到本服务器的 Web 服务。Migration Server 可以被实现为一种特殊的服务, 专门负责服务迁移。例如, 在 AXIS Engine 中, Migration Server 可以作为一个特殊的 Web 服务; 在 OnceAS 中, 则可以作为被微内核管理的一个系统服务。

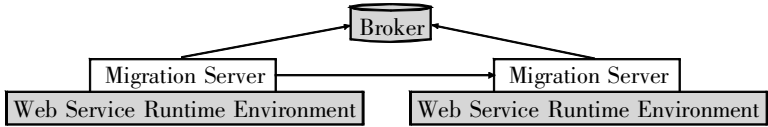


图 1 服务迁移

在迁移过程中, 继续让 Web 服务在源服务器处理客户请求, 提高了 Web 服务的可用性, 却造成 Web 服务迁移前后的状态不一致。为了保证迁移一致性, 在迁移到目标服务器的 Web 服务开始处理客户请求之前, 必须与迁移前的 Web 服务进行状态同步。目前, 进行状态同步的方法大致可以分为两类: 在 Web 服务迁移之后, 继续传递不一致的状态。这种方法增大了 Web 服务迁移的成本。在迁移之后, Web 服务立即开始处理客户请求, 需要时再对状态进行同步。这种方法造成 Residual Dependency^[13] 问题, 也增加了客户请求的处理时间。为了弥补这两类方法的不足, 本文在服务迁移的过程中引入了一个 Broker(图 1)。源服务器不再把不一致状态直接迁移到目标服务器, 而是存放到 Broker 中; 目标服务器只需要与保存在 Broker 中的状态进行同步即可。这种方法降低了迁移成本, 也解除了目标服务器与源服务器的 Residual Dependency。

Broker可以由分布式文件系统实现,也可以由数据库系统实现。

2.2 基于操作依赖关系的性能改进

根据 2.1 节的分析,迁移到目标服务器的 Web 服务在执行客户请求之前需要先进行状态同步。但是根据 Web 服务的特点,每个客户请求可以对应 Web 服务的一个操作,通过分析请求之间的依赖关系,可以让某些阻塞的请求提前执行,不必等待状态同步完成。例如,由类 Account 封装成的 Web 服务提供了 setOperation 和 getOperation 两种操作。setOperation 对类 Account 的某个私有属性进行写操作, getOperation 返回这个私有属性的值。如果在服务迁移过程中,源服务器继续执行了 Web 服务的 getOperation 操作,而 getOperation 只是把结果返回给客户端,并不改变 Web 服务的状态。迁移之后,在目标服务器进行状态同步之前,如果后续请求要求执行 setOperation 操作, setOperation 就不必等待同步完成。

类 Account 描述如下:

```
Class Account{
private:
    String name;
    int account;
    int interest;
public:
    String getName( );
    String setName( String name);
    int getTotalAmount( );
    int inquiryInterest( );
    void save( int money);
}
```

定义 1 Web 服务定义为 $Service = \{ Property, Operation \}$ 。其中, Property 是 Service 包含的属性集合, $Property = \{ p_i \}, i = 1, \dots, N, N$ 为属性的数目; Operation 是 Service 包含的操作集合, $Operation = \{ o_j \}, j = 1, \dots, M, M$ 为操作的数目。

定义 2 在 Property 和 Operation 之间存在两种关系,即 Read 和 Write。其中, $Read = \{ o_j, p_i \mid p_i \in Property, o_j \in Operation \}$, 表示操作对属性具有读操作,即属性出现在操作内部某个表达式的右侧; $Write = \{ o_j, p_i \mid p_i \in Property, o_j \in Operation \}$, 表示操作对属性具有写操作,即属性出现在操作内部某个表达式的左侧。

定义 3 $read_j = \{ p_i \mid o_j, p_i \in Read, 1 \leq i \leq N \}$ 表示与操作 o_j 具有读操作关系的属性集合; $write_j = \{ p_i \mid o_j, p_i \in Write, 1 \leq i \leq N \}$, 表示与 o_j 具有写操作关系的属性集合。

定义 4 如果 $(read_i \cap write_j) \cap (write_i \cap write_j) \neq \emptyset$, 则 o_i 依赖于 o_j , 表示为 $o_i \rightarrow o_j$ 。如果 $(read_i \cap write_j) \cap (write_i \cap write_j) = \emptyset$, 则 o_i 不依赖于 o_j 。需要注意的是这种依赖关系具有自反性,但不具有传递性。

定义 5 在迁移过程中源服务器处理的客户请求对应的操作集合表示为 UnSynOperation, $UnSynOperation \subseteq Operation$ 。在目标服务器同步过程中需要阻塞的客户请求所对应的操作集合表示为 DepOperation, $DepOperation \subseteq Operation$ 。

类 Account 中, 设 $getName \rightarrow setName, getTotalAmount \rightarrow save, inquiryInterest \rightarrow save, save \rightarrow save$ 。如果在迁移过程中, 服务在源服务器继续执行了 setName 和 inquiryInterest 操作, 即 UnSynOperation 为 { setName, inquiryInterest }, 则 DepOperation 集合为 { getName, setName }。显然, 后续请求到达时, 只有 getName

和 setName 需要阻塞, 其他方法可以继续执行。

在把不一致状态保存到 Broker 之前, 源服务器需要把 Web 服务在迁移过程中执行的客户请求所对应的操作集合, 即 UnSynOperation 集合发送给目标服务器。目标服务器根据 UnSynOperation 集合, 生成 DepOperation 集合。为了提高运行效率, 用 Hash 表存放 DepOperation 集合, 可以在 $O(1)$ 时间复杂度内, 判断后续请求是否出现在 DepOperation 集合中。需要注意的是 Web 服务各操作之间的依赖关系, 是由 Web 服务提供方在部署前提供的。

2.3 迁移一致性约束

为了保证 Web 服务迁移一致性, 必须满足下面三个约束:

(1) 状态一致性约束。设 t_1 是 Web 服务在源服务器停止处理客户请求的时刻, t_2 是 Web 服务在目标服务器开始继续处理客户请求的时刻。状态一致性约束表示为 Web 服务在 t_1 时刻的状态信息等于在 t_2 时刻的状态信息。

(2) 引用一致性约束。Reference 表示 Web 服务不可被迁移的引用集合, 如 Web 服务对业务逻辑的引用、在运行时打开的文件句柄或 Socket 链接等。引用一致性约束表示为 Web 服务在迁移后可以重新更新 Reference 集合的引用。

(3) 消息一致性约束。设 Request 表示客户请求, Result1 表示 Web 服务在不迁移的情况下返回的客户结果, Result2 表示 Web 服务在迁移过程中返回的客户结果。消息一致性约束表示为对于同一客户请求, Web 服务在迁移过程中返回的结果 Result2 等于在不迁移的情况下返回的结果 Result1。需要注意的是丢失客户请求是违反了 Web 服务的消息一致性的一种特殊情况。

2.4 算法实现

在 Web 服务迁移之前, Migration Server 首先决定何时进行迁移, 即迁移策略。本文中实现的 Migration Server 可以根据服务器的 CPU 利用率或内存利用率, 主动发起迁移; 也可以按照用户命令对 Web 服务进行迁移。Migration Server 其次选择目标服务器。如果用户已经预先指定好目标服务器的地址, Migration Server 可直接向目标服务器发送迁移请求; 否则, Migration Server 广播迁移要求, 其他 Migration Server 接收到迁移请求后, 如果发现具备迁移的条件, 就返回应答。Migration Server 收到响应后, 就向目标服务器发起迁移。

图 2 是算法的执行步骤:

- (1) 源服务调用用户实现的 PostHandler 接口, 处理文件句柄等不可迁移的资源。
- (2) 源服务器把 Web 服务的描述信息发送到目标服务器。
- (3) 目标服务器开始部署 Web 服务, 创建 Web 服务执行所需要的必要信息。
- (4) 源服务器把 Web 服务运行时状态信息发往目标服务器。
- (5) 目标服务器调用用户实现的 PostHandler 接口, 处理文件句柄等不可迁移的资源。
- (6) 源服务器暂停 Web 服务执行。在暂停 Web 服务执行的时刻, 如果 Web 服务正在执行某个客户请求, 则必须等待 Web 服务处理完该请求, 再暂停其执行。
- (7) 目标服务器恢复 Web 服务的状态, 检查 Web 服务对

业务逻辑实现的引用是否正确。如果引用为本机地址或相对地址, 则更新为绝对地址。

(8) 源服务器将迁移过程中改变的状态信息保存到 Broker 中。

(9) 源服务器将迁移过程中执行过的客户请求对应的操作集合发送给目标服务器。

(10) 目标服务器与保存在 Broker 中的状态信息进行同步。

(11) 目标服务器开始处理客户请求, 按照 2.3 节定义的操作依赖关系, 判断是否需要阻塞将要处理的客户请求。

(12) 源服务器把收到的客户请求转发到目标服务器。

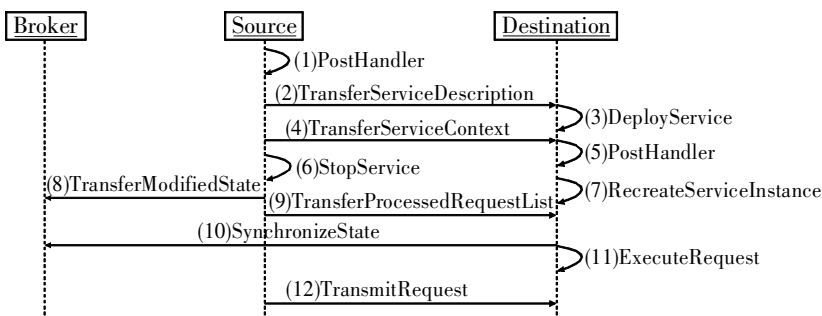


图 2 迁移算法执行的序列

3 算法一致性分析

下面简要分析 WS Broker 算法是如何满足 Web 服务迁移一致性约束的。

性质 1 WS Broker 可以满足 Web 服务迁移的状态一致性约束。WS Broker 算法中, 在源服务器把不一致状态存放到 Broker 的过程中, 通常需要经过网络传输, 所以迁移时间不可被轻易忽略。因此, 在迁移不一致状态之前, WS Broker 算法暂停 Web 服务执行, 即不再让 Web 服务处理客户请求。在暂停 Web 服务执行的时刻, 如果 Web 服务正在执行某个客户请求, 必须等待处理完该请求, 再暂停其执行。Web 服务在目标服务器开始执行前, 必须先与 Broker 内的不一致状态进行同步。在 WS Broker 算法中, 为了迁移效率, 提前执行了某些阻塞的客户请求, 但是通过对客户请求之间依赖关系的分析, 算法仍然可以满足状态的一致性约束。通过以上分析可见, WS Broker 算法可以满足状态一致性约束。

性质 2 WS Broker 可以满足 Web 服务迁移的引用一致性约束。在源服务器开始传送 Web 服务的状态信息前, WS Broker 算法执行用户实现的 PostHandler 接口, 处理文件句柄、Socket 链接等不可被迁移的引用。迁移之后, WS Broker 算法在恢复服务状态前, 首先调用 PostHandler 接口, 重新建立与 Web 服务相关的文件句柄、Socket 链接等; 其次检查并更新 Web 服务对业务逻辑的引用。因此, WS Broker 算法可以满足引用一致性约束。

性质 3 WS Broker 可以满足 Web 服务迁移的消息一致性约束。WS Broker 算法中, 在 Web 服务迁移过程中, 源服务器的 Web 服务继续执行, 所以不会丢失服务迁移过程接收到的客户请求。源服务器暂停 Web 服务执行后, 如果收到客户请求, 由源服务器负责转发给目标服务器, 从而保证了客户请求的不丢失。另外, 根据性质 1, WS Broker 虽然提前执行了某些阻塞的客户请求, 但是仍然可以满足消息的一致性约束。通过以上分析可见, WS Broker 算法可以满足消息一致性约束。

由性质 1 ~3 得, WS Broker 算法可以满足 Web 服务迁移的一致性约束。

4 实现与评价

本文在中国科学院软件所自主研发的 SOAP 处理引擎 SOAPExpress 中, 实现了迁移算法的原型。为了评估迁移算法的性能, 本文也实现了另外两种迁移算法: WS Total Copy 和 WS Pre-Copy。WS Total Copy 算法是指在 Web 服务迁移之前, 源服务器暂停 Web 服务执行; 迁移后, 目标服务器再继续执行 Web 服务。WS Pre-Copy 类似于本文提出的迁移算法, 不同点在于, WS Pre-Copy 在状态同步完成前, 必须阻塞所有的后续客户请求。

实验环境由两台软硬件环境相同的机器组成, CPU 为 Pentium 4 2.8GHz, 内存为 512MB, 网卡 10Mbps, 操作系统是 Windows XP SP2, HTTP 服务器软件为 Tomcat 5.0.1。利用 SOAPTest 模拟 1 000 个并发客户端, 访问其中一台服务器。当被访问服务器的 CPU 利用率超过 50% 时, Migration Server 主动迁移服务到另外一台服务器。如表 1 所示, 实验结果表明, 本文提出的迁移算法 WS Broker 在服务迁移时间和服务中断时间上都要优于 WS Total Copy 和 WS Pre-Copy 算法。

表 1 Web 服务迁移开销

	Migration Time(s)	Service Interrupt Time(s)
WS Broker	1.861	0.412
WS Total Copy	2.032	2.032
WS Pre-Copy	1.85	0.732

5 结束语

保证 Web 服务迁移一致性, 对于提高服务可用性具有重要意义。本文提出的基于 Broker 的 Web 服务迁移算法 WS Broker, 在保证 Web 服务迁移一致性的基础上, 提高了服务迁移效率。在今后的工作中, 将对算法作以下改进: 还需要进一步分析 Web 服务状态的组成部分, 迁移 Web 服务在目标服务器执行所需要的最小状态集合, 进一步减少迁移时间。如何扩展 WSDL 文档, 在部署时得到 Web 服务各操作之间的依赖关系。

参考文献:

[1] 杨芙清. 软件工程技术发展思索[J]. 软件学报, 2005, 16(1): 1-7.

[2] MIFSUD T, STANSKI P. Measuring performance of dynamic web service migration using LAMS: proceedings of 2002 International Conference on Software, Telecommunications and Computer Networks (SoftCom 2002) [C]. New York: IEEE Communication Society, 2002.

[3] TROGER P, POLZE A. Object and process migration in .NET: proceedings of the 8th IEEE International Workshop on Object-Oriented Real-time Dependable Systems (WORDS '03) [C]. Washington: IEEE Computer Society Press, 2003: 139-146.

[4] MEEHEAN J, LIVNY M. A service migration case study: migrating the condor schedd[EB/OL]. [2005]. <http://www.cs.wisc.edu/~jmeehean/Papers/ScheddMigration.pdf>.

[5] HAMMERSCHMIDT B C, LINNEMANN V. Migrating stateful web services using apache axis and P2P: proceedings of the IADIS International Conference on Applied Computing[C]. Carvoeiro: IADIS Press, 2005: 433-441.

(下转第 70 页)

首先, 将本文提出的判决反馈载波干扰矩阵估计方法与文献[7]提出的 Hadamard 载波干扰矩阵估计方法(截取长度取为 $P = N/2$ 和 $P = N/4$, N 为系统子载波个数)及载波干扰未消除的系统进行对比仿真。图 2 显示它们的信干比仿真曲线。从中可看出, 相对于 Hadamard 载波干扰矩阵估计方法, 判决反馈的载波干扰消除效果更好。这主要是 Hadamard 载波干扰估计方法中仅估计部分干扰系数致使载波干扰矩阵估计误差较大所致。

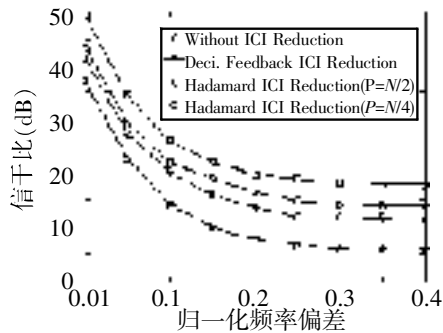


图 2 不同载波干扰消除方法的对比效果

接下来, 将判决反馈的载波干扰消除方法结合文献[2]所建议的比特交织时频编码调制技术, 仿真了这两种载波干扰消除方法联合作用下的系统误码率。取归一化载波频率偏差为 0.1, 子载波分别为 QPSK、16-QAM、64-QAM 及 256-QAM 等调制方式。为便于性能对比, 在给出 AWGN 信道下的无频偏系统性能的同时也对基于比特交织编码调制的载波干扰消除方法、Hadamard 载波干扰消除(截断长度分别为 $P = N/2$ 和 $P = N/4$, N 为系统子载波个数)及本文所提出判决反馈等方法进行了仿真。最终的仿真结果如图 3 所示。从图中可以看出, 联合判决反馈和比特交织编码调制的方法进一步提高了系统性能且获得了较好的载波干扰消除效果, 这有利于抗扰和去扰协同作用的结果。

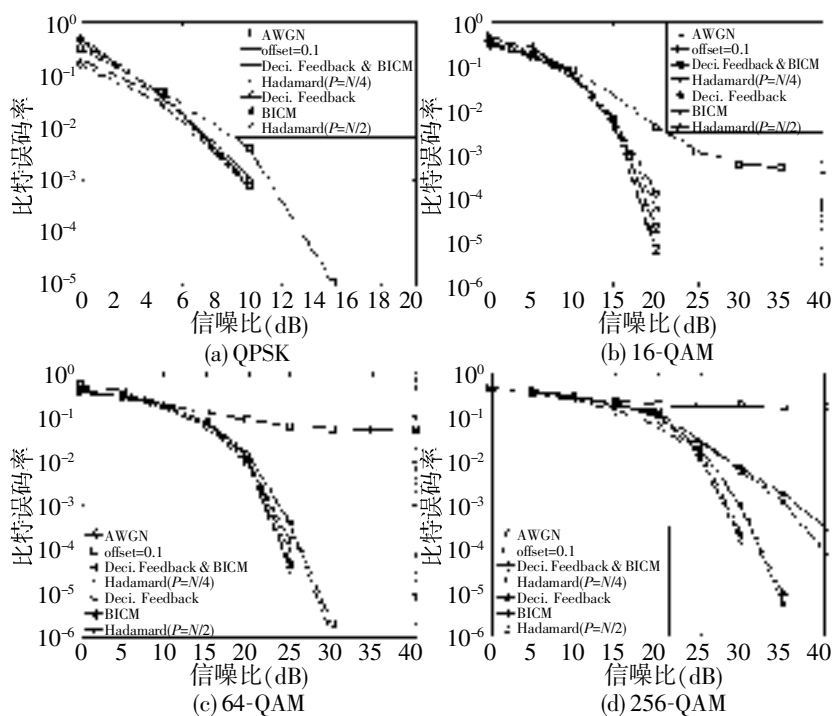


图 3 在 offset=0.1 情况下 Hadamard、判决反馈、BICM、联合 BICM 和判决反馈的系统误码率曲线

4 结束语

本文根据载波干扰矩阵可用傅里叶变换矩阵进行对角化的特点, 提出了基于判决反馈的载波干扰消除方法。该方法仅需一个训练符号估计初始载波干扰矩阵, 再根据符号判决结果及时重新估计载波干扰矩阵, 可较快地跟踪载波干扰的动态变化, 弥补了 Hadamard 载波干扰矩阵估计精度差的不足。同时,

该方法通过与比特交织编码调制方法相结合可进一步提高系统的误码性能。

参考文献:

- [1] OLLET T, VAN B, MOENECLAEEY M. BER sensitivity of OFDM systems to carrier frequency offset and wiener phase noise[J]. **IEEE Transactions on Communications**, 1995, **43**(234): 191-193.
- [2] HUANG Defeng, LETAIEF K B, LU Jianhua. Bit-interleaved time-frequency coded modulation for OFDM systems over time varying channels[J]. **IEEE Transactions on Communications**, 2005, **53**(7): 1191-1199.
- [3] GARCIA P, ORTEGA A, MINGO de J, *et al.* Nonlinear distortion cancellation using LINC transmitters in OFDM systems[J]. **IEEE Transactions on Broadcasting**, 2005, **51**(1): 84-93.
- [4] RYU H G, LI Yingshan, PARK J S. An improved ICI reduction method in OFDM communication system[J]. **IEEE Transactions on Broadcasting**, 2005, **51**(3): 395-400.
- [5] SEKCHIN C, POWERS E J. Cancellation of inter-carrier interference in OFDM systems using a nonlinear adaptive filter: IEEE International Conference on Communications[C]. New Orleans: IEEE, 2000: 1039-1043.
- [6] TOMASIN S, GOROKHOV A, YANG Haibing, *et al.* Iterative interference cancellation and channel estimation for mobile OFDM[J]. **IEEE Transactions on Wireless Communications**, 2005, **4**(1): 238-245.
- [7] WU H C, WU Yiyan. A new ICI matrices estimation scheme using hadamard sequence for OFDM systems[J]. **IEEE Transactions on Broadcasting**, 2005, **51**(3): 305-314.
- [8] SUN Fengwen, JIANG Yimin, BARAS J S. On the convergence of the inverses of toeplitz matrices and its applications[J]. **IEEE Transactions on Information Theory**, 2003, **49**(1): 180-190.
- [9] GRAY R M. Toeplitz and circulant matrices: a review[EB/OL]. (2005-11-02). <http://www-ee.stanford.edu/~gray/CIT006-journal.pdf>.

(上接第 67 页)

- [6] 范国闯. 支持 EJB 动态分布的组件迁移模型与算法[J]. 软件学报, 2004, **15**(3): 404-413.
- [7] MASUTTI O. Distributed web session management[D]. Switzerland: University of Zurich, 2000.
- [8] SULTAN F, BOHRA A, IFTODE L. Service continuations: an operating system mechanism for dynamic migration of internet service sessions: proceedings of the 22nd International Symposium on Reliable Distributed Systems(SRDS '03) [C]. Washington: IEEE Computer Society Press, 2003: 177-186.
- [9] RICHMOND M, HITCHENS M. A new process migration algorithm[J]. **ACM SIGOPS Operating Systems Review**, 1997, **31**(1): 31-42.
- [10] ROUSH E T. The freeze free algorithm for process migration[D]. Urbana-Champaign: University of Illinois, 1995: 1-162.
- [11] Apache AXIS[EB/OL]. <http://ws.apache.org/axis/>.
- [12] 黄涛, 陈宁江. OnceAS/Q: 一个面向 QoS 的 Web 应用服务器[J]. 软件学报, 2004, **15**(12): 1787-1799.
- [13] LAWRENCE R. A survey of process migration mechanisms[J]. **ACM SIGOPS Operating Systems Review**, 1988, **22**(3): 28-40.