清 华 大 学

综 合 论 文 训 练

题目：Web 服务性能评估及其在赛百平台中的应用

系　　别：　自动化系

专　　业：　自动化

姓　　名：　王　震

指导教师：　曹军威 副研究员

辅导教师：　杨吉江 副研究员

2007 年　6 月　15 日

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：学校有权保留学位论文的复印件，允许该论文被查阅和借阅；学校可以公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存该论文。

(涉密的学位论文在解密后应遵守此规定)

签　名：_____导师签名：_____日　期：_____

中文摘要

Web 服务/SOA 技术已经成为未来网络技术发展的一个趋势，是下一代互联网很重要的技术核心。同时为了满足人们对共享网络资源的进一步的需求，美国自然科学基金提出构建赛百平台（Cyberinfrastructure，简称 CI）。Web 服务技术将成为构建 CI 的技术之一。然而 Web 服务技术目前来说还并不成熟，虽然有很多构建 Web 服务的工具，但不同的工具构建的 Web 服务有不同的特性，各有优缺。那么测试这些软件包的性能就成为了目前的当务之急。通过测量性能，了解工具性能方面的优缺点，一是可以为提高工具性能提供参考，二是可以根据实际需求选择合适的工具。

本文的主要工作为三项：提出自己的测试方案，对测试结果进行分析，比较之下得出结论。本文中主要通过支持时间（overhead time），往返时间（round-trip time）和服务完成时间（process time）三个参数来描述 Web 服务的性能，观察不同负载下这三个参数的变化情况，比较之下可以看出 Axis+Java 适合于在小负载情况下构建 Web 服务，而 Gsoap+C 适合于大负载情况。在实际情况中要根据实际需求来决定采用哪种工具。

通过测试可以看到，Web 服务技术就目前来说有很多优点，相当适合于 CI 的构建，但就其表现出来的性能来说距离 CI 的需求还有一定的距离，不论采用哪种工具，在访问量较大的情况下，支持时间（overhead time）的开销不容忽视，也就是说用户等待的时间大部分花在了与服务没有直接联系的消息传递上面。同时往返时间随着负载的增加也增长较快，很难承担大访问量的 Web 服务构建。

本文对最常用的两种构建 Web 服务的工具进行了测试，得到了很有意义的结果。但就从评判 Web 服务工具在 CI 平台下性能表现来说还有两项工作亟待解决：首先是如何用参数的形式测量和评判程序重复利用性，可移植性这两个对于 CI 构建至关重要的特点。其次是如何将被构建服务的实际需求用参数的方式描述出来。这两项工作将是本文后续的研究方向。

关键词：性能评估　赛百平台　Web 服务 服务性能 SOA

ABSTRACT

Web Services/SOA has entered the mainstream of Internet technologies and become the core technique for next generation Internet infrastructure. In order to meet people's requirement of resource sharing across Internet, the US National Science Foundation proposed to build Cyberinfrastructure (CI) in 2003.Web Services will be an essential technique to enable CI, though not well developed yet. While there are many software tools which can be used to implement Web Services, each has its own features. Thus, performance evaluation of these software tools has become an urgent task. By testing the performance and investigating characteristics of various tools, references on how to improve the performance of these tools are provided in this work. It is advised that Web Services developers should choose appropriate tools for different implementations.

Two Web Services implementations, Gsoap+C and Axis+Java, are investigated, evaluation approaches are proposed, and the conclusion was educed after analyzing testing results in this paper. Overhead time, round-trip time and process time are used to describe the performances of Web Services. By observing these three metrics in different workloads, it is concluded that Web Services based on Axis perform much better than those implemented using Gsoap when workload is low, but Gsoap is more suitable for building heavy loaded Web Services. The choice of implementations should be made according to different situations.

Although Web Services technology is suitable to enable CI, from this work, we can observe that current Web Services implementations hardly meet the CI performance requirements. Whichever we choose, overhead time will cost too much to be ignored, which means clients spends too much time on the work that has nothing to do with the service provision itself. It can be concluded from the testing data that RTT increases quickly as the workload increases, which means

these Web Services implementations can not serve large number of requests. The low efficiency and quickly increasing interaction time can not be accepted by CI enabling.

In this paper, a Web Services implementation benchmark is proposed, two most wildly used Web Services tools are tested, and a valuable conclusion is achieved. There are still two challenging issues to address. Firstly, how to characterize code reusability and portability is important for the CI performance investigation. Secondly, quantitative requirement description of practical Web Services is another challenging issue to be addressed further.

Keywords： Performance Evaluation Cyberinfrastructure Web Services Service Performance SOA

主要符号对照表

SOA	面向服务架构（Service-oriented architecture）
CI	赛百平台（Cyberinfrastructure）
WSDL	服务描述语言（Web 服务 Description Language）
UDDI	统一描述、发现和集成（Universal Description, Discovery and Integration）
XML	可扩展标记语言
RRT	请求往返时间（Round Trip Time）
PRT	服务完成时间（Process Time)
OHT	服务支持时间（Overhead Time）

目　　录

第1章 序言

1.1 赛百平台（Cyberinfrastructure）的提出

随着科学技术的发展，人们对于计算能力的要求越来越大。2000 年美国建设的激光干涉引力波天文台（Laser Interferometer Gravitational-wave Observatory，简称 LIGO）[1]投入了科学运行，LIGO 每天产生 1 万亿字节的数据，来自全世界 40 多个科研单位的 500 多名研究人员组成了 LIGO 科学合作组织（LIGO Scientific Collaboration，简称 LSC）[2]，共享分布在美国和欧洲的 10 多台高性能计算机的几千个 CPU 和上千万亿字节的存储能力，联合投入到 LIGO 的数据分析工作中。

这仅仅是进入 21 世纪以后，在物理、天文、生物和医药等研究领域出现的众多联合项目的代表之一。随着科技的发展，科学项目逐渐呈现出跨组织联合攻关，实验设备昂贵，数据量大等特点。计算机和网络逐渐成为科学研究不可或缺的一部分。“绝大部分的现代科学家都在使用和依赖先进科学计算来完成日常研究，并且渴望得到更快的处理能力”[3]。不管是采用典型的两种研究方法：理论/分析和实验/观察中的哪一种，现代的研究都离不开高性能计算能力和宽带网的支持。美国国家科学基金从上世纪 80 年代就开始陆续资助先进计算技术的研究，包括在特定领域推动相关的应用。至今已经建设了数十个面向特定领域的信息、计算和数据的网络环境。典型例子有地震仿真网络（Network for Earthquake Engineering Simulation[4]，简称 NEES）、开放科研网格 （Open Science Grid，简称 OSG）[6]、国家虚拟天文台（US National Virtual Observatory，简称 NVO）[6]和万亿次网格（TeraGrid）[6]等等。各国也随之陆续建立了很多专门用于特定科研领域的网格环境。我国也很快建立了中国国家网格，中国教育科研网格等专项领域网格。

但这些网格并没有实现资源的深层次的共享。理想中的网格应该是将网络上的所有资源在逻辑上看作是一个无差别的资源池，每一个用户无需知道逻辑资源的物理位置等地层信息，根据自己的需要和权限自由的获取自己想要的资源，例如计算能力，存储能力，某些特定仪器的使用等，甚至可以组

建最适合自己的临时小型的网格。相比之下，目前网格主要存在以下一些问题：

1．网格共享的广度不够：各个网格之间的资源是不能共享的。对于这些网络社区、虚拟组织和网格环境，虽然在其内部实现了资源共享，但彼此之间却各自独立，不能互通，也就是说没有实现真正意义上的开放平台。

首先，近年来大型计算和存储的需求急剧增加，如果在各个领域分别投入，那么就会使得投入太大，资源的利用率不高。从投资成本来说这样不是优化的做法。

其次随着跨学科的研究越来越普遍和重要， 越来越多的科学研究不再是个人行为，跨组织、跨学科的联合攻关乃至国际合作成为必须。如合作承担国际热核聚变实验堆计划的 7 个成员是欧盟、中国、韩国、俄罗斯、日本、印度和美国，中国要承担整个项目 100 亿美元中 10％。为了让不同专业不同组织的人协同工作，就必须建立一个跨领域的网络共享平台，让所有使用这个平台的人可以相互交流，自由的使用调配参与学科领域的资源。

2．目前的网格共享程度不够：目前的网格中的资源往往仅仅指计算能力，存储能力，信息资源或者是特定领域中的特定仪器。事实上一切有利用价值而又可以与网络相连的仪器和随之而来的服务都可以看作是资源。无论是与网络连接的传感器，控制器或者天文望远镜，还是通信，存储，搜索服务，都可以看作是网格当中的资源。这就需要在逻辑层上封装这些物理资源，使之在逻辑上有一个标准的接口，提供给用户使用。

基于以上的这些原因，美国国家科学基金在 2003 年公布的专家咨询报告（一般称为 Atkins 报告）中提出来了 Cyberinfrastructure（以下简称 CI）的概念。

那么到底什么是 CI 呢？在 Atkins 报告中，对 CI 的概念有如此解释：“基础架构（Infrastructure）一词从上世纪二十年代开始被用作公路、电网、电话、桥梁以及铁路等类似的公共设施的总称，为工业经济的发展起到了支撑作用。虽然好的基础架构经常被人忽视，其重要性也只有在出现故障时才能体现出来，但基础架构的确是人类创造出的复杂而昂贵的系统。CI 作为一个新名词主要指基于分布计算机、信息和通信技术的基础架构。CI 对于知识经济的重要性可以与基础架构对工业经济的支撑作用相比拟。

对照现实世界中基础架构的重要性，CI 是从基础架构的角度，重新审视数字空间资源的结果。在数字世界中，我们已经有一些类似现实世界中基础架构的成功例子，通过简短的 Email 地址，就可以实现通信；通过简单的域名就可以登陆相应的 Web 主页，以了解相关信息。Email 和 Web 实现了数字世界中的信息共享，但更深层次的资源共享只有通过 CI 才能实现。在数字空间中，我们有计算、数据、网络和软件等资源，还有许多数字化的仪器设备等等。在科研领域，CI 资源的典型例子有高性能计算机、大规模数据存储、大型天文望远镜、粒子加速器、引力波天文台以及环境监测系统等。CI 淡化物理资源的概念，通过构建逻辑资源层（又称 Cyberresources），实现资源的透明访问，而不必关心资源具体的物理地址。CI 的框架如图 1.1 所示。

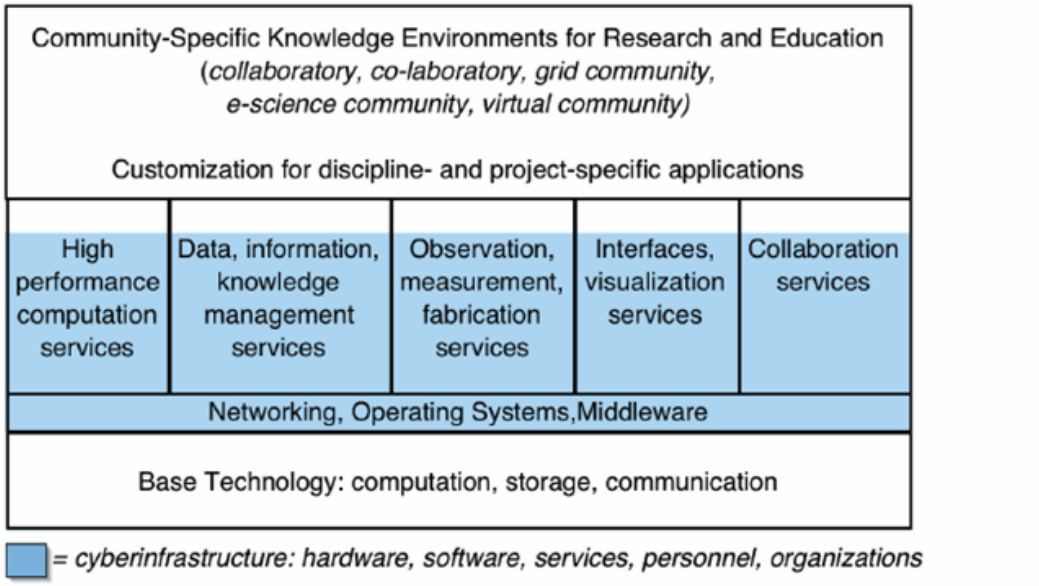

图1.1　Cyberinfrastructure结构图

CI 实际上还要提供一个基于资源共享的应用使能开放环境（又称 Cyberenvironments）。典型的 CI 应用类型包括高性能计算、数据分析与可视化、虚拟组织与服务等。比如，近十年发展起来的网格计算技术，主要用于实现虚拟组织内跨组织的资源共享。如前面提到的 LIGO 数据网格，CI 可以提供一个开放环境，使得应用领域的科学家可以根据自己的需要，实现特定的网格服务，而且可以通过 CI 逻辑资源层，动态地添加资源。将来 LIGO 的引力波科学家可以使用 CI 开放环境直接构建 LIGO 的数据网格，而毋需

通常那样，从底层资源开始，再向上逐步做起。LIGO 数据网格的计算能力不够时，尚可通过 CI 操作，动态获取其他空闲（如 TeraGrid）资源。从这个意义上说，网格是 CI 的具体应用，CI 是网格的构建平台，两者关系如图 1.2 所示。

图1.2　CI与网格的关系图

事实上，CI 是作为一种理念或者概念已经被很多人和国家所重视，它是网格技术的进一步发展。目前对 CI 需求最迫切的还是科学研究领域。形象地说，未来 CI 就像科学家的手臂可以用来快速的构建满足特定领域需要的网格和虚拟组织环境，进而方便的伸向任何可用的科研资源。科学家可以全身心地投入到解决科学问题本身，而不是耗费大量的时间和精力去获取资源。

1.2 Web 服务技术介绍

CI 的概念一经提出，便得到工业界的高度重视，基础架构符合技术发展的客观规律，也很容易让人联想到 Internet 和 Web 的成功，开始都是为了满足科研需要，而最终则导致一场信息技术的革命。CI 同样可以使人产生足够的联想和期待，但不管在理论上设想的又多好，到底是纸上谈兵，终归是要落实到具体实现上的。目前来看，CI 实现的技术难点主要为：

CI 逻辑资源层的实现：CI 资源逻辑层实现的技术挑战主要来源于 CI 环境下资源的多样性。不同资源的交互方式和安全要求都有很大的不同，需要进行软件上的统一封装，隐藏物理上的不同。

CI 应用使能开放环境的实现：CI 应用使能建立在资源共享的基础上，CI 虽然从基础架构的角度出发，在资源与应用使能方面进行了层次化，实现了统一，但是一些贯穿于各个层次之间的问题解决起来尚有挑战性。

在这个需求下，有一种技术可以很好的用来实现 CI：SOA 方法及 Web 服务技术。

SOA（Service Oriented Architecture）[8],即面向服务架构，是目前数个大公司如 IBM、微软、SAP、BEA 力推的未来软件设计方法，它将应用程序的不同功能单元以服务的形式进行封装，通过这些服务之间定义良好的接口和契约联系起来。接口是采用中立的方式进行定义的，它独立于实现服务的硬件平台、操作系统和编程语言。这使得构建在各种这样的系统中的服务可以以一种统一和通用的方式进行交互。它的架构如图 1.3 所示：

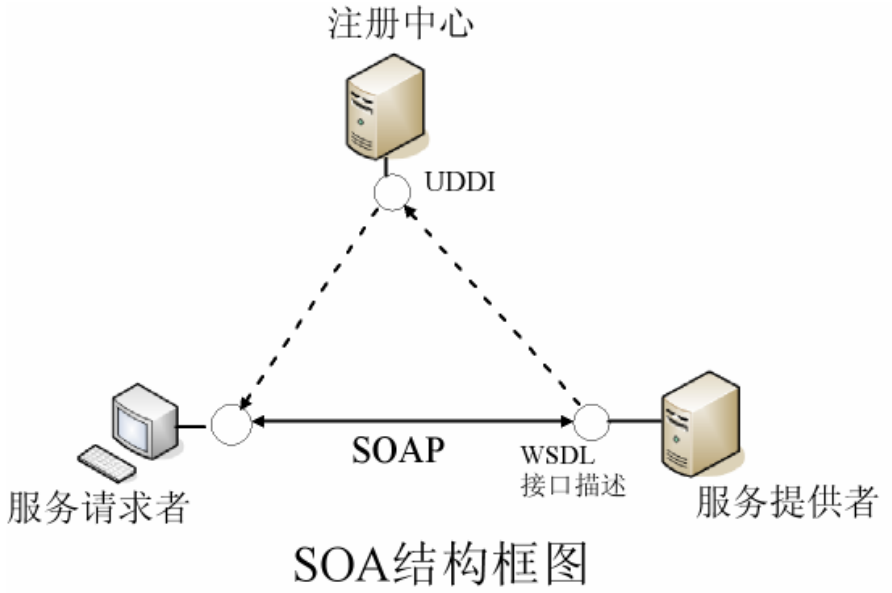

图1.3　SOA结构框图

由此带来的优点有：

平台无关，统一封装：由于是采用中立的协议进行封装和通信的，所以在 SOA 框架下，不同系统不同应用单元都可以自由的相互交流信息，共享资源。这个特点对于 CI 的实现是至关重要的一点，在 CI 的理念中，所有的

能够连接到网络中的资源，例如个人 PC，服务器，能够联入网络的实验设备，甚至是各种传感器，所有存在使用价值的单元，都视作是统一的网络资源，都应是无视平台不同和个体差异而可以共享的。SOA 的平台无关和统一封装恰恰可以解决这个问题。

松散耦合，组件相对独立：系统内的组件单元以服务的形式封装后，仅留有一个接口以供访问，而且是采用中立的协议通信和对接口进行描述，当一个服务组件需要进行修改时，只要保证接口不变，就不需要对系统的其他部分进行任何改动。所以可以称之为松散耦合，组件相对独立。松耦合系统的好处有两点，一点是它的灵活性，可以灵活的将某个组件重复利用到不同的系统当中，另外一点，当组成整个应用系统的每个服务的内部结构和实现逐渐地发生改变时，它能够继续存在。而紧耦合意味着应用系统的不同组件之间的接口与其功能和结构是紧密相连的，因而当需要对部分或整个应用程序进行某种形式的更改时，它们就显得非常脆弱。对于 CI 来说，应用系统可以认为是扩大到了整个网络，服务组件就是网络中所有存在使用价值的服务资源。而 Internet 松散的时变的使用环境，必然要求 CI 的实现方案是松散耦合而且其中组件是相对独立的。SOA 能够很好的解决这一问题。

那么什么是 Web 服务技术呢？简单的来说可是实现 SOA 方法的技术就是 Web 服务 技术。与 CI 关系如图 1.4：

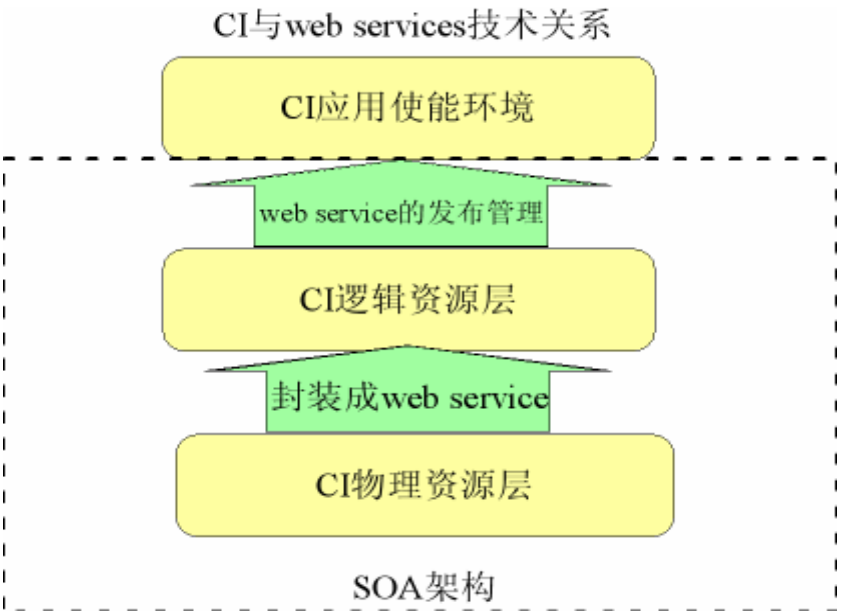

图1.4　CI与Web服务技术关系图

目前主流有两种实现方法，分别是基于 Java 语言实现和基于 C 语言实现 Web 服务。在 Java 语言的实现工具有：Tomcat+Axis；C 的工具有：Gsoap，Axis2C；Python 的工具有：Soappy 等。

不过不管采用哪种语言，Web 服务技术都是完全基于 XML（可扩展标记语言），XSD（XML Schema）等独立于平台、独立于软件供应商的标准，是创建可互操作的、分布式应用程序的新平台。它可以带来以下一些优点：

长项一：跨防火墙的通信。

CI 环境中，网络资源将遍布有网络的各个地方，而且将有成千上万的分布在世界各地用户访问这些网络资源，那么用户和服务器之间的通信将是一个棘手的问题。因为客户端和服务器之间通常会有防火墙或者代理服务器。在这种情况下，使用 DCOM 就不是那么简单，而且由于服务端接口各不相同，针对每个服务都进行客户端开发也是不可能的。传统的做法是，选择用浏览器作为客户端，写下一大堆 Asp 页面，把应用程序的中间层暴露给最终用户。这样做的结果是开发难度大，程序很难维护。

如果中间层组件换成 Web 服务的话，就可以从用户界面直接调用中间层组件，从而省掉建立 Asp 页面的那一步。要调用 Web 服务，可以直接使用 Microsoft Soap Toolkit 或.NET 这样的 SOAP 客户端，也可以使用自己开发的 SOAP 客户端，然后把它和应用程序连接起来。不仅缩短了开发周期，还减少了代码复杂度，并能够增强应用程序的可维护性。同时，应用程序也不再需要在每次调用中间层组件时，都跳转到相应的“结果页”。

从经验来看，在一个用户界面和中间层有较多交互的应用程序中，使用 Web 服务这种结构，可以节省花在用户界面编程上 20%的开发时间。另外，这样一个由 Web 服务组成的中间层，完全可以在应用程序集成或其它场合下重用。最后，通过 Web 服务把应用程序的逻辑和数据“暴露”出来，还可以让其它平台上的客户重用这些应用程序。

长项二：应用程序的系统级集成。

企业级的应用程序开发者都知道，企业里经常都要把用不同语言写成的、在不同平台上运行的各种程序集成起来，而这种集成将花费很大的开发力量。同样在 CI 环境下，如果一个用户想要调用多个网络服务来构建自己的应用，那么同样要面对这个问题。而且现在的实际情况是大部分时候用户是无法得到被调用服务的程序代码，也就不可能做到无缝的将其嵌入到自己

的应用当中去。但采用 Web 服务技术可以很好的解决这个问题，CI 环境中网络提供的服务单元可以用标准的方法把功能和数据“暴露”出来，供其它应用程序使用。

一个很实际的例子就是目前 Google 公司将自己公司的搜索业务以 Web 服务的方式提供给用户，用户可以通过了解公布出来的服务接口来调用它或者将其嵌入到自己的应用当中去。只要接口不变，那么即使 Google 公司的搜索业务进行升级对于用户来说也是没有关系的。

长项三：跨组织信息集成。

用 Web 服务可以集成应用程序，提高代码的重复利用，实现资源的共享。但当这种共享跨越组织的界限时会怎么样呢？这里涉及到很多个问题：如何将自己的服务发布？而又如何知道在网络的哪个地方有自己想要的服务？而这个服务的访问方式或者说接口又是怎么定义的？这些问题在 CI 环境下更加凸现出来：在网络资源都封装为统一的服务的前提下，如何发布，搜寻，管理这些服务，使得这些服务能够有效率的安全的，而且是在可控的条件下被 CI 用户使用？

Web 服务有一整套相关的方法和协议来处理这个问题：统一的数据格式 XML，标准服务接口描述语言 WSDL，用于服务发布注册和搜索的协议 UDDI（Universal Description, Discovery and Integration）。这方面 Web 服务 的一个很好的实际应用就是 B2B（Business-to-Business）。通过 Web 服务，公司可以把关键的商务应用发布在专门的电子商务中心或者自己公司的门户上门，将访问方式和功能“暴露”给指定的供应商和客户。例如，把电子下单系统和电子发票系统“暴露”出来，客户就可以以电子的方式发送订单，供应商则可以以电子的方式发送原料采购发票。用 Web 服务来实现 B2B 集成的最大好处在于可以轻易实现互操作性。只要把商务逻辑“暴露”出来，成为 Web 服务，就可以让任何指定的合作伙伴调用这些商务逻辑，而不管他们的系统在什么平台上运行，使用什么开发语言。这样就大大减少了花在 B2B 集成上的时间和成本，让许多原本无法承受 EDI 的中小企业也能实现 B2B 集成。

Web 服务 虽然很适合于 CI 的实现，但也有一些很重要的问题需要解决。

1.3 问题提出

Web 服务和 SOA 方法早在 2000 年就被人提出，原本开发出来就是为了解决电子商务中程序应用跨组织，跨系统交互问题，得到了 IT 业数个大公司如微软，IBM，SUN，Ariba、Intel 等的倾力支持，其推出的协议和框架如 XML，WSDL，SOAP 也得到了广泛的认可。但经过这么多年发展，直到目前为止，Web 服务技术并没有得到广泛的使用，即是在电子商务方面，Web 服务技术也被人认为是一种屠龙之技，虽然很重要也是未来 Web 的发展方向，但就目前的需求来说并不很实用。所以现在 Web 服务技术更多停留在学术讨论阶段。那么到底是什么阻止了 Web 服务从学校和研究院走向各个 IT 公司，走向实际应用呢？

主要原因就是 Web 服务性能不够理想，这里性能不够主要包括：显著增加了额外开销，处理速度不够快，最大处理请求数不够大等。如果要使用 Web 服务技术实现 CI 的构建，那么 Web 服务的处理性能将显著的影响 CI 的性能。原因有二：

Web 服务技术对资源进行封装，以统一的形式对用户和上层服务提供服务和功能，完成的是 CI 环境下物理资源到逻辑资源的转变，它是 CI 环境中很底层的一层。所以封装方法的性能将影响到整个 CI 环境。而且在 CI 环境下很多资源本身的处理能力并不是很强，例如传感器，一些实验设备等，那么在这种情况下如果仅仅为了封装这个资源，就产生了大量的系统开销，就会得不偿失。

在 CI 环境中如果用户想构建自己的网格或者科学环境，往往需要将多个底层的服务组合起来完成自己的功能。任何两个服务之间的一次通信都要分别进行两次 SOAP 包的解析和封装，那么多个服务之间的通信将产生大量的解析与封装行为，封装和解析 SOAP 包的性能是影响 Web 服务性能的主要因素。所以良好的 Web 服务性能是能否构建有效上层服务和开放使能环境的关键因素之一。

目前 Web 服务 主要通过基于 C/C++和 Java 两种编程语言的工具来实现。完成同样的功能，不同的编程语言和工具包，表现出来的性能也各不相同。Web 服务的应用五花八门，很难将不同应用放在一起比较衡量服务性能。所以测试 Web 服务 的性能，归根结底还是测试 Web 服务工具包的好与坏。本文的主要工作就是依托标准典型的 Web 服务应用，在不同实现方法下，

测试出 Web 服务负载曲线，连接速度等方面的参数，然后将两者进行比较，得出相应结论。

1.4 本文工作

根据前面提出的问题，本文的主要工作为：

第一步：学习目前 Web 服务测试方法和相关领域研究成果，提出自己的测试方案；

第二步：根据测试方案，使用上文提到到两种工具，分别建立 Web 服务；

第三步：进行测试，同时对测试得到的数据进行分析；

第四步：得出结论，对以后工作提出展望；

1.5 文章结构

本文的其余章节将如下安排：

第 2 章对目前 Web 服务测试方法和相关领域的研究工作进行分析总结，指出有待改进和值得本文借鉴之处，制定本文采用的测试方案，说明测试环境及测试前提。

第 3 章在本文的测试方案基础上对测试数据进行分析，从多个角度用多种方法处理数据，得出测试结果。

第 4 章给出最终结论，分析测试方法的缺点。简要总结本文工作，给出研究结论，并叙述未来可能的工作方向。

第2章 Web 服务测试方法

2.1 相关技术

虽然本文的主要内容是以几项 Web 服务技术作为测试对象，不过 Web 服务实现技术的测试必须依托一定的应用才能够完成。以一个统一典型的应用作为实例，采用不同的技术实现这个应用的 Web 服务，然后通过测试这个具体 Web 服务的性能来了解不同 Web 服务实现技术的特点。

目前针对 Web 服务进行性能测试的方法并不多。大部分集中与 Web 服务功能测试[9]，而在实现技术方面的测试也多是关于 SOAP 工具包的测试[10]，我们要测试的是作为 Web 服务实现工具，它是一个系统的工具，SOAP 工具包仅仅是其中很重要的一个组成部分。虽然如此，还是有很多相关的测试方法可以借鉴。

首先 Web 服务架构是分布式的，作为一种分布式架构的中间件，其测量方法有着和一般分布式中间件测量方法的共同之处。分布式应用一般将通过响应时间[11]，吞吐量和可测量性作为描述分布式应用特性的三个最重要的参数．在本文中，针对 Web 服务自身的特性和 CI 环境下的需求，很重要的就是需要测量 overhead 时间和服务的往返时间 RTT 在不同数量的客户请求下的变化。

其次 Web 服务一般都将 HTTP 协议作为底层协议，就从 HTTP 层来看，可以说 Web 服务与一般的 Web 网页服务器相差并不很大，性能的制约因素也在很大程度上是相同的。要想使得 Web 服务 达到很好的性能，那么至少在 HTTP 即网络应用层的性能要表现很好。而且 Web 网页服务器和 Web 服务的工作模式类似，于是就可以借鉴 Web 网页服务器的测试方法来对 Web 服务进行测试。Web 网页服务器的性能测试相对来说已经很成熟了[12] [13]。Web 网页服务器的性能测试很看重服务器对于处理突发多用户请求的能力。同样在本文测量的过程中也需要测试 Web 服务的负载极限，同时查看平均响应时间随着客户数量的增加是如何变化的。

Web 服务虽然是通过一个统一的描述语言来描述服务接口，但服务的定制和发布都有很大的自由度。于是有人提出了一个框架可以不加改造的适用不同的服务，评价其服务性能[14]。这个框架由客户端，控制器，驱动程序是三个独立的部件组成，驱动程序解释程序接口属性，和客户端进行通讯并通过模型设置。客户端通过控制器将其注册，而控制器聚集最终的性能结果。通过这个标准设计，控制器和驱动程序可以在不同程序之间重复利用。客户端根据 Web 服务 接口描述语言 WSDL 和可模块化的加载测试数据进行设计。在这个模块下对服务性能的测量也主要是两个参数[15]：

SIRT(Web 服务 Interaction Response Time： Web 服务交互相应时间)完成一次成功的 Web 交互所用时间；

STPS(Service Interactions Per Second：每秒进行的服务交互):测试时间之内平均 SIPS 完成次数。我们也用处理次数来指代服务交互次数；

实际情况中，Web 服务性能之所以不能够让人满意，一是因为当 overhead 时间太大，另外就是多个连接时平均响应时间会显著增加。所以我们更需要了解在不同负载情况下 SIRT 时间。由于 SIRT 与 STPS 有很大的相关性，所以并没有需要测量 STPS 值。

2.2 本文采用的方案

总结上一章节所说的相关技术，针对 CI 的应用需求，对 Web 服务工具测试内容重点主要有两个：

Web 服务作为中间件，必定会增加应用的开销。所以首先需要测定的就是采用某项工具封装应用时 overhead 时间。这个时间主要由 SOAP 包的解析和封装产生的。这是 Web 服务工具中核心功能之一，所以专门测试这个时间对于了解 Web 服务性能特性和指导以后的改进方法都是很重要的。

Web 服务工具对应用的支持性能。为了发布一个 Web 服务，总是需要将具体的应用放入到一个工具提供的标准环境或者说是容器当中去。这个容器对应用支持的程度有很多具体因素影响，不同工具的支持方法也各不相同。我们能做的就是不去考虑具体的如何支持应用的方法，将其当作一个整体进行考虑。

测试内容为以上两项，不过测试侧重的是负载特性测试，也就是在不同

负载下 Web 服务的具体性能表现。在 CI 环境下，一个有用的服务往往被很多人调用，特别是目前一些公司在尝试将软件以网络服务的形式提供给用户而不是将程序下载到用户端使用，的情况下，保证 Web 服务面对大量请求也能够保持良好性能的需求就显现出来了。事实上，要想将 Web 服务技术推广开来，Web 服务必须能够对多用户进行良好支持。

总结起来测试参数为：

RTT（Round Trip Time）：如图 2.1 所示，是指客户端程序从发出请求到得到响应的时间。这个时间可以整体衡量这个服务系统的性能。

OHT（Overhead Time）：如图 2.1 所示，这个参数记录的是 Web 服务中间件耗费时间。这个时间并不能直接测得，是通过间接测量的手段来得到的。

PRT (Process Time)：如图 2.1 所示，这个参数记录的是 Web 服务应用处理请求花费时间，。

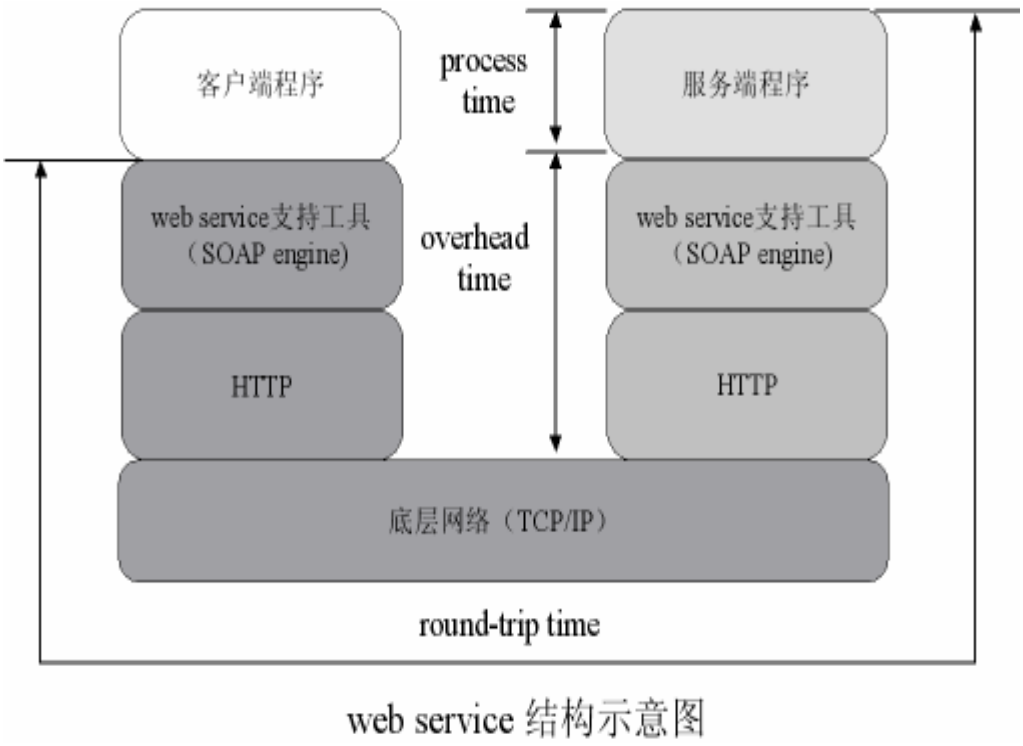

图2.1 Web服务结构示意图

2.3 测试方法及测试前提

本文是以测试 Web 服务工具包为核心，以被测工具包为工具，构建一个 Web 服务应用，然后通过一个客户程序向此 Web 服务提出请求，传递参数，经过服务器计算运行后将最终结果返回客户程序。

为了测试负载特性，客户程序分别以不同的数量在很短的时间内向服务器发送大量请求，然后记录下这些报发送，到达，返回等时间。通过这些记录，就可以比较清楚的分析出这个 Web 服务的负载特性，进而可以比较不同的 Web 服务工具包。

不过在实际测试的时候，需要面对以下几个问题：

网络延迟：本文测试的内容所关心的是 Web 服务完成情况，并不需要涉及网络延迟，如图 2.1 所示。而且网络延迟也是很难估计和控制的，为了尽可能的消除网络延迟对测量结果的影响，所以测量是客户端及服务端处于 100M 以太局域网内，且选择网络状况良好时期进行测量。

客户端程序延迟：为了便于分析，一般都认为客户端程序可以在很短的时间内发送大量的请求，从服务器来看，可以认为是一个阶跃输入。但实际情况是如果发送的请求太多，那么客户程序需要一定的时间才能够发送出去，此时最早发送出去请求的响应已经回到了客户端，所以并不能够简单的将生成和发送请求的时间可以忽略。不过实际测量结果显示生成和发送所有请求所花费的时间是服务端实际完成所有请求时间的十几分之一，一定程度上可以忽略这个时间。不过为了准确期间，分析时采用的是客户端请求全部发出后到所有请求得到响应这段时间的数据。

除了以上这些问题，在测量时也需要保证测试程序所在计算机没有无关进程干扰，以保证测量的准确性。

2.4 被测工具及测试平台

2.4.1 被测工具

目前 Web 服务 主要通过基于 C/C++和 JAVA 两种编程语言的工具来实现。不管是哪种编程语言，都已经有相应的工具包来定制和发布 Web 服务。常用的发布 Web 服务方法有：采用 Java 编程的 Axis+Java 与采用 C 编程的

Gsoap+C。Axis 提供一个可以封装一般应用和类为 Web 服务的容器和相应的支持环境，Gsoap 是 C 语言编写的针对 Web 服务的工具包，可以为程序提供用于定制和发布 Web 服务的环境，同时还有很多支持性的函数库。所以本文测试的 Web 服务工具就为 Axis 与 Gsoap 工具包。

虽然两种工具大相径庭，但其核心功能确实一样的，那就是充当 SOAP 引擎。SOAP 是一个基于 XML 的用于应用程序之间通信数据编码的传输协议。最初由微软和 Userland Software 提出，随着不断地完善和改进，SOAP 很快被业界广泛应用。在其发展过程中，W3C XML 标准工作小组积极促成 SOAP 成为一个真正的开放标准。SOAP 被广泛作为新一代跨平台、跨语言分布计算 Web 服务的重要部分。通常情况下，SOAP = HTTP + RPC + XML。即：SOAP 以 HTTP 作为底层通信协议，以 RPC 作为交互方式，以 XML 作为数据传送的格式。在 Web 服务体系中，客户端与服务端或者服务端与服务端均是靠基于 SOAP 协议的 SOAP 包来通讯。

2.4.2 Axis 简介

Axis 是 Apache 组织推出的 SOAP 引擎，Axis 项目是 Apache 组织著名的 SOAP 项目的后继项目。虽然 Axis 核心功能是 SOAP 引擎，但完成的功能并不仅仅是一个引擎的功能。 它还包括：

一个独立运行的 SOAP 服务器；

一个 Servlet 引擎的插件，这个 Servlet 引擎一般都是 Tomcat；

对 WSDL 的扩展支持；

一个将 WSDL 的描述生成 JAVA 类的工具；

一些示例代码；

还有一个监控 TCP/IP 包的工具 TCPMonitor；

Axis 一共有两种发布 Web 服务的方法[16]：

使用即时发布 Java Web 服务(JWS)

对即时发布的支持是 Axis 的特色之一，使用即时发布使用户只需有提供服务的 Java 类的源代码，即可将其迅速发布成 Web 服务。每当用户调用这类服务的时候，Axis 会自动进行编译，即使服务器重启了也不必对其做任何处理，使用非常简单快捷。

使用定制发布 Web 服务 Deployment Descriptor(WSDD)

即时发布是一项令人激动的技术，它使 Web 服务的开发变得如此简单；然而即时发布并不总是最好的选择，比如有些应用系统是第三方提供的，我们没有购买源代码，只有.class 文件，但我们又希望将这个应用系统的一些功能对外发布成 Web 服务，使其能够在更大范围内产生作用，这个时候即时发布技术就无能为力了。此外，即时发布技术并不灵活，无法进行更多的服务配置，这使得它并不能满足一些特定系统的需求。

本文测量中采用的是第一种发布模式也就是即时发布。

2.4.3 Gsoap 简介

Gsoap 是一个开源的项目，用它可以方便的使用 C/C++地进行 SOAP 客户端和服务器端编程，而不必了解 XML 和 SOAP 协议的细节。这样使用者就可以专注于自己的 Web 服务 客户端或服务器端的编写，而不用纠缠与其它细节。它以 HTTP 协议为基本的通信协议，以 XML 文件形式请求远程服务，再以 XML 文件的形式返回执行结果。

Gsoap 编译工具提供了一个 SOAP/XML 关于 C/C++ 语言的实现，从而让 C/C++语言开发 web 服务或客户端程序的工作变得轻松了很多。Gsoap 包含的 WSDL 生成器可以为你的 web 服务生成 web 服务的解释，解释器及导入器可以使用户不需要分析 web 服务的细节就可以实现一个客户端或服务端程序，编译器可以 生成 SOAP 的代码来序列化或反序列化 C/C++的数据结构[17]。和 Gsoap 相比，其他绝大多数的 C++ web 服务工具包通过提供一组 API 函数类库来处理特定的 SOAP 数据结构，这样就使得用户必须改变程序结构来适应相关的类库。与之相反，Gsoap 利用编译器技术提供了一组透明化的 SOAP API，并将与开发无关的 SOAP 实现细节相关的内容对用户隐藏起来。Gsoap 的编译器能够自动的将用户定义的本地化的 C 或 C++数据类型转变为符合 XML 语法的数据结构，反之亦然。这样，只用一组简单的 API 就将用户从 SOAP 细节实现工作中解脱了出来，可以专注与应用程序逻辑的实现工作了。此外，Gsoap 编译器可以集成 C/C++和 Fortran 代码（通过一个 Fortran 到 C 的接口），嵌入式系统，其他 SOAP 程序提供的实时软件的资源和信息；可以跨越多个操作系统，语言环境以及在防火墙后的不同组织。开发简单和跨平台这两个特点对于 CI 的开发是十分有利的。

2.4.4 测试平台

测试环境：清华大学 FIT 楼同一网段内 100M 以太网；

测试平台：

硬件：服务器：Pentuim（R）4 CPU 3.00GHz　　1Ｇ内存

客户端：Pentuim（R）4 CPU 2.80GHz　　1G 内存

软件：LINUX 系统　Fedora Core 6

测试软件：

LINUX+ Gsoap+C：

采用 Gsoap2.7.1e；

下载地址：http://www.cs.fsu.edu

http://Gsoap2.sourceforge.net

LINUX+Apache+Tomcat+Axis+Java ：

采用 Apache HTTP Server 2.2.4；

Tomcat4.1.36；

Axis-1_2；

J2sdk1.5

下载地址：http://apache.org/

第3章 Web 服务性能测试结果和数据分析

测试数据一共可以分为两个部分：一是静态负载测试数据，一是动态负载测试。静态负载测试主要测量在一定的负载情况下 RTT，PRT，OHT，RPT 等参数的分布特性，动态负载测试测量的是这些参数随着负载数增加的变化规律。

在本章节中有一些概念需要定义一下：

设 n 维向量 var，var(i)表示第 i 个向量元素，向量元素中最大值为 k，最小值为 m，则

对测试数据（向量形式）的归一化处理为：

$$\text{var}(i) = (\text{var}(i) - m)/(k - m) \qquad \text{公式 3-1}$$

平均值计算方法为：

$$ave = \sum_{i} \text{var}(i)/n \qquad \text{公式 3-2}$$

3.1 静态负载测试

分别采用两种工具：Axis 和 Gsoap 定制发布服务，采用多个客户端同时发送请求的方式进 RTT(Round-Trip Time), RPT(Response Time), OHT (Overhead Time) ,PRT(Process Time)的测量。

3.1.1 RTT 测试

对测试结果做出频率分析。本文采取的静态负载数分别为 200，1500，5000，主要是基于 Web 服务处理能力方面的考虑：200 个负载是低负载，5000 个负载已经接近实际服务器的极限负载，然后取中间 1500 个负载作为过渡情况进行察看。

基于 Gsoap 工具包的 Web 服务

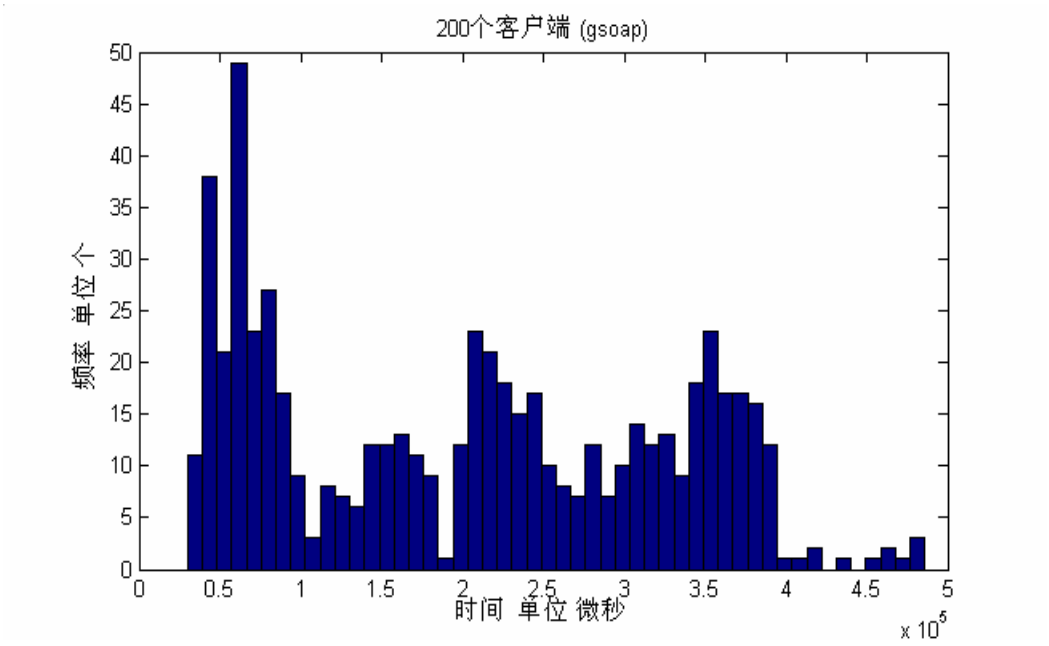

图3.1　200个客户端基于Gsoap的Web服务RTT直方图

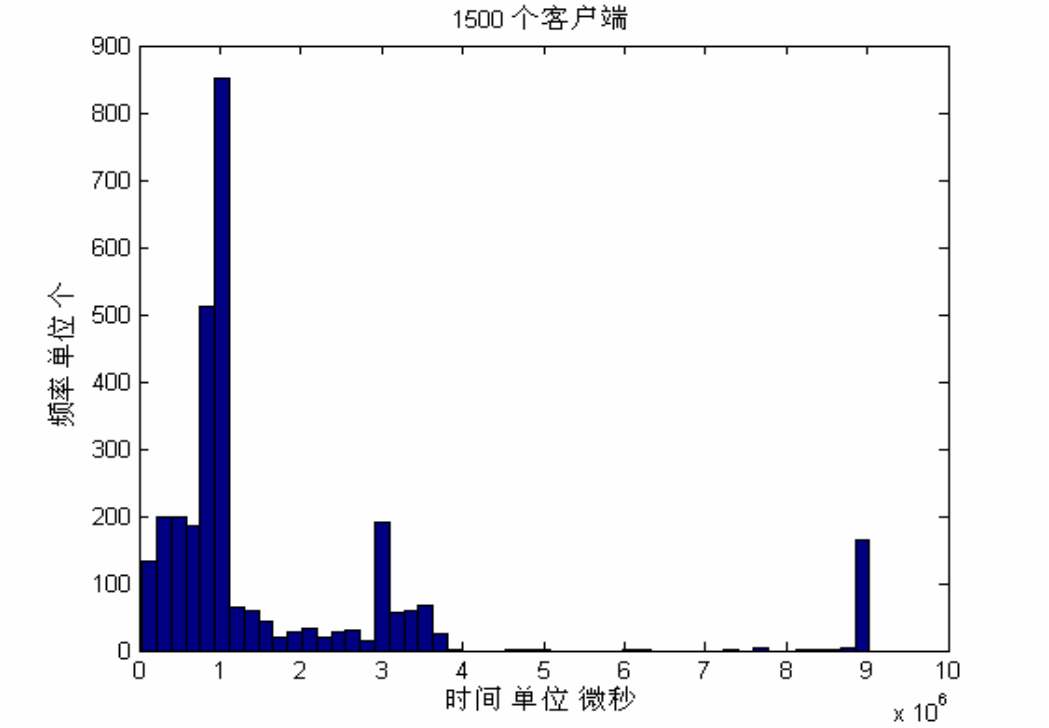

图3.2　1500个客户端基于Gsoap的Web服务RTT直方图

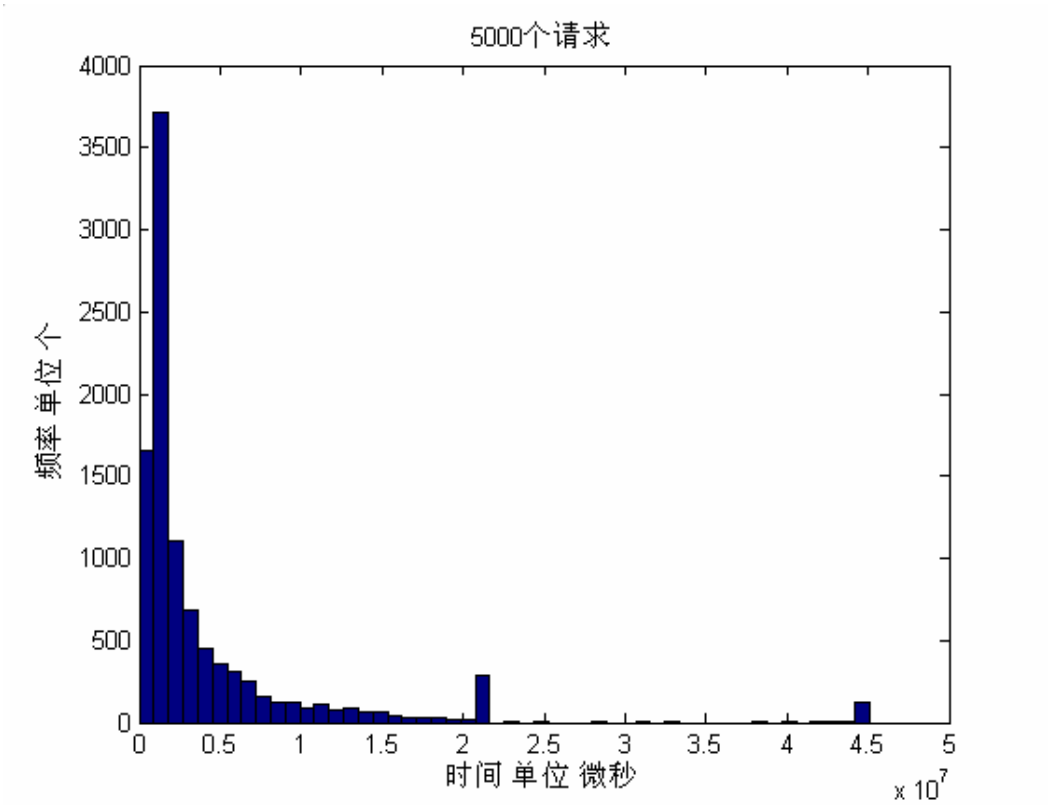

图3.3 5000个客户端基于Gsoap的Web服务RTT直方图

由图 3.1，3.2，3.3 可知，基于 Gsoap 工具包定制的 Web 服务 当同时处理 200 个请求的时候处理平均时间较短，从图上来看分布比较散。随着请求数量的增加，当同时处理 5000 个请求时，平均的响应时间显著增加，但时间相对集中。将数据归一化后计算标准方差，见表 3.1。

表3.1 Web服务（Gsoap）RTT标准方差

客户端数	*200*	*1500*	*5000*
标准方差	0.2512	0.2338	0.1616

由表可见，虽然 RTT 时间随着同时处理的请求数的增加显著增加，但分布随之相对集中，能够比较好的平衡处理每一个请求，性能表现相当稳定。

基于 Axis+Java 的 Web 服务

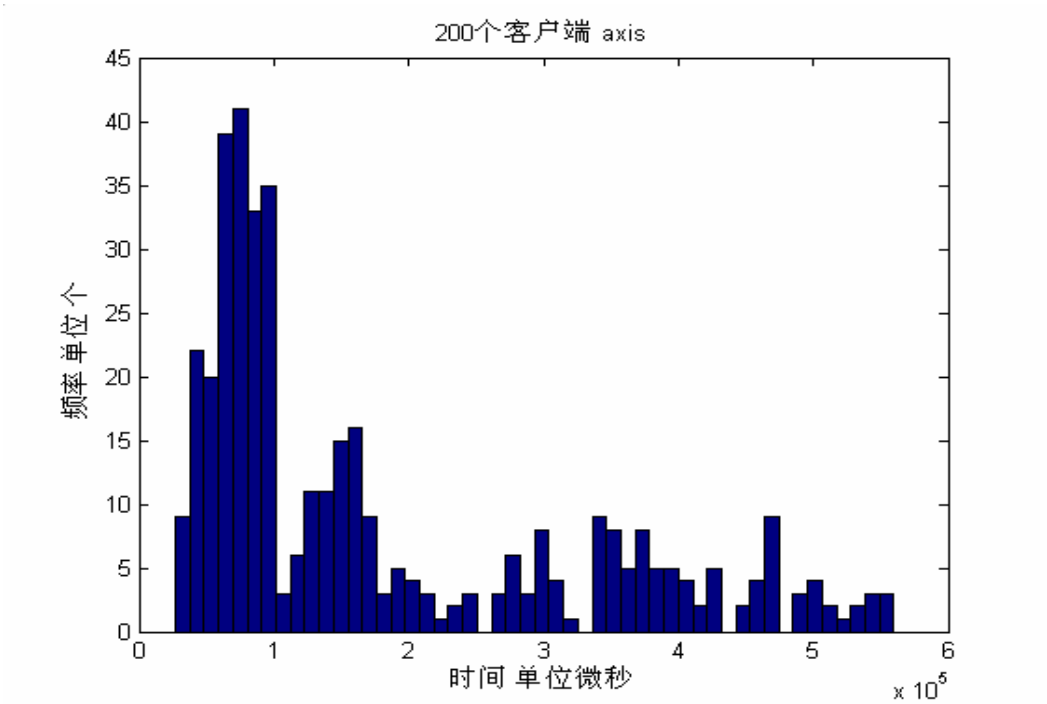

图3.4　200个客户端基于Axis的Web服务 RTT直方图

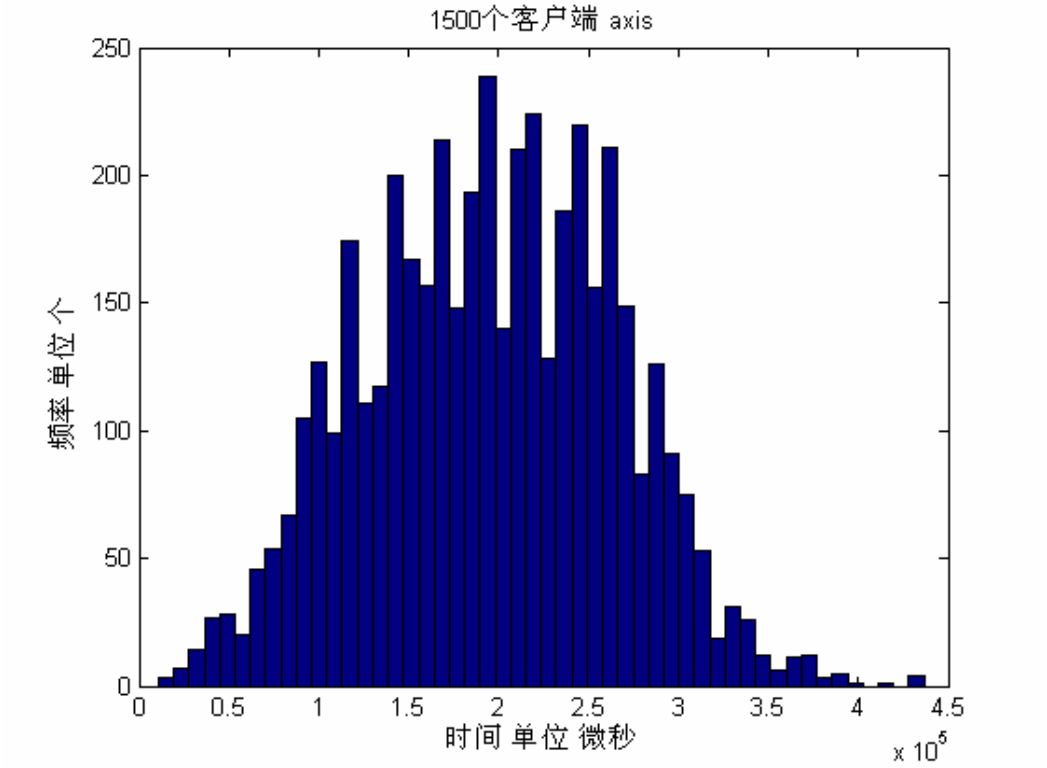

图3.5　1500个客户端基于Axis的Web服务RTT直方图

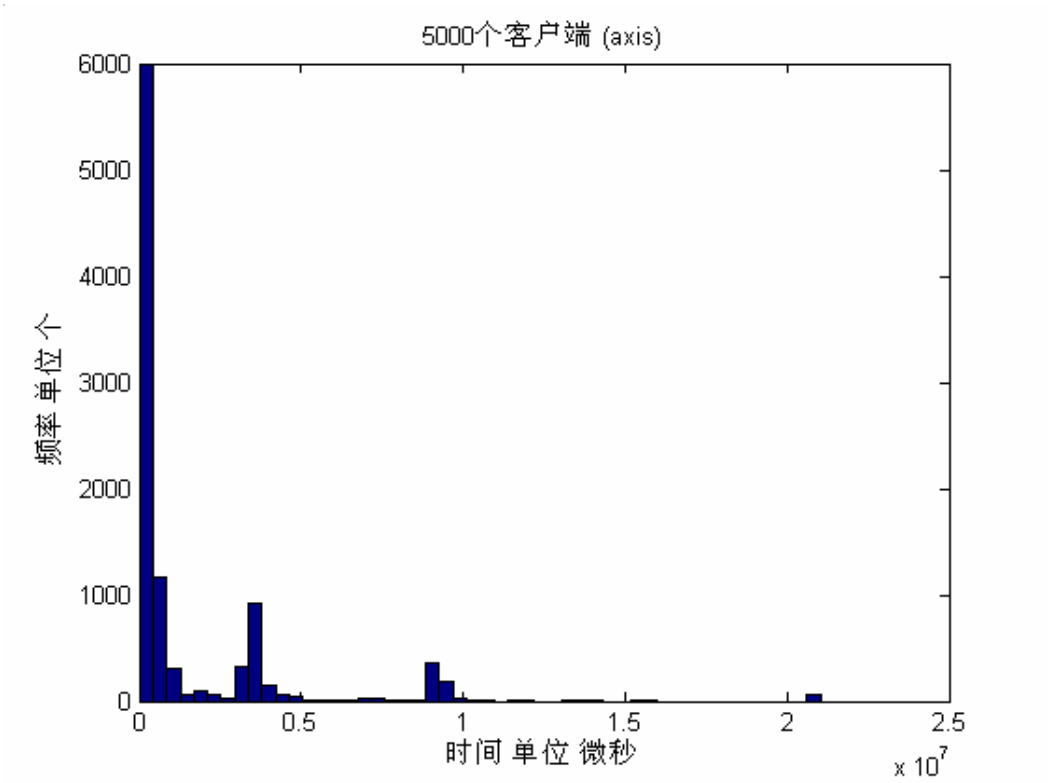

图3.6 5000个客户端基于Axis的Web服务RTT直方图

同样，由图 3.4，3.5，3.6 可知，RTT 时间随着同时处理个数的增加，也呈现出时间显著增加同时相对集中的情况，这个和基于 Gsoap 定制的 Web 服务有相同的趋势。将数据归一化计算标准方差，见表 3.2。

表3.2 Web服务（Axis）RTT标准方差

客户端数	*200*	*1500*	*5000*
标准方差	0.2694	0.1637	0.1361

从数据中上可以看出基于 Axis 定制的 Web 服务测量的 RTT 时间要小于基于 Gsoap 的 Web 服务，同时方差也要比较小，性能更加稳定。对于平均响应时间随负载的变化情况将在动态测试这节中进行详细解释。

3.1.2 PRT 测试

对测试结果做出频率分析。为了便于与上面的 RTT 测量结果最比较，同时也有上文测量 RTT 选择负载数同样的考虑，也采用 200，1500，5000 三个负载数。

需要提出的是在 AXIS+JAVA 环境下，对时间的测量只能够精确到毫秒。在实际测量环境下，有多个因素会影响到最终测量结果，如网络延迟，CPU 状况等，微秒级的测量结果实际上已经不准确了。同时本文对测量结果的分析也更加看重时间趋势，所以测量精度到毫秒对本文最终结论并没有重要影响。

基于 Gsoap 工具包的 Web 服务

通过测试，发现 PRT 时间随着负载的增加并没有显著增加，在时间上的分布图也极为类似。如图 3.7，3.8，3.9 所示。

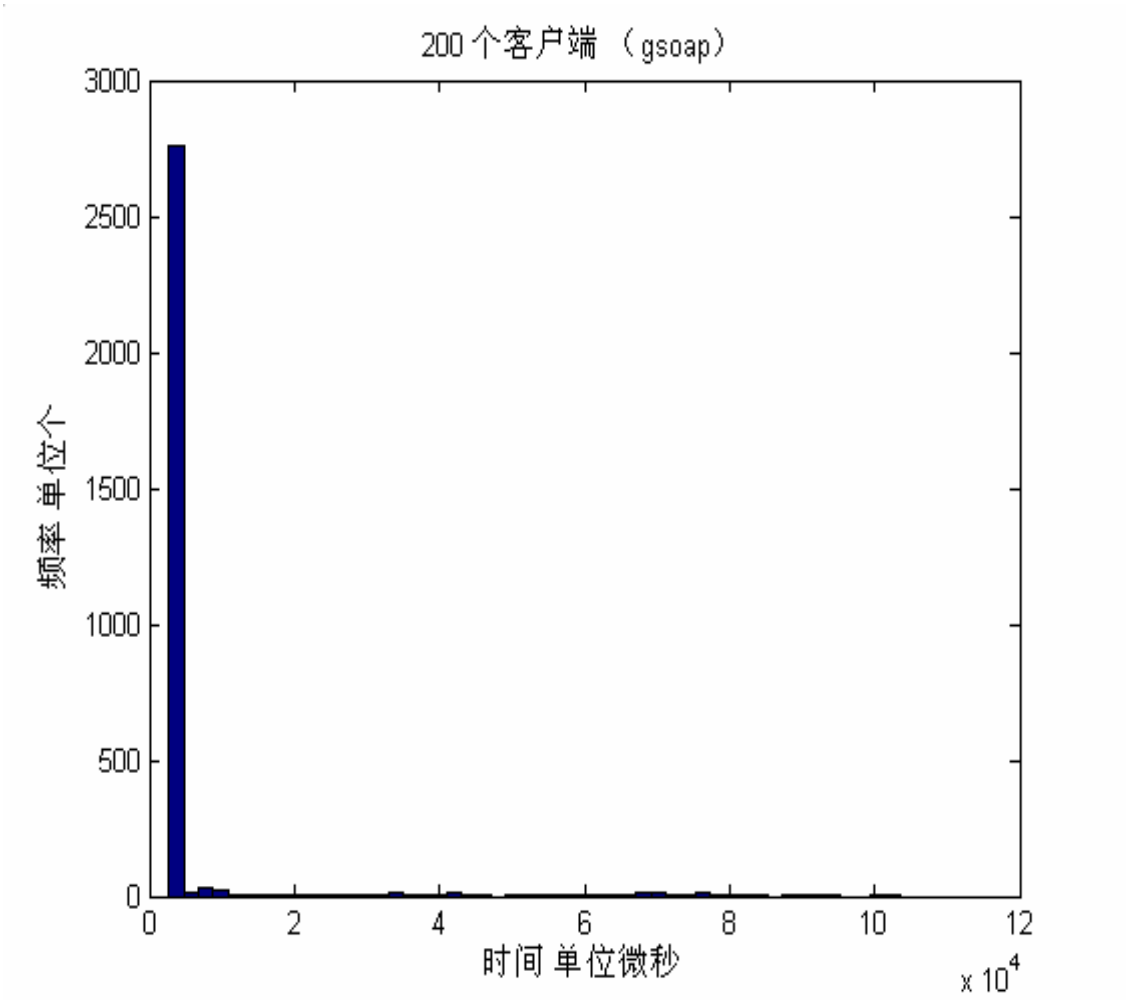

图3.7　200个客户端基于Gsoap的Web服务PRT直方图

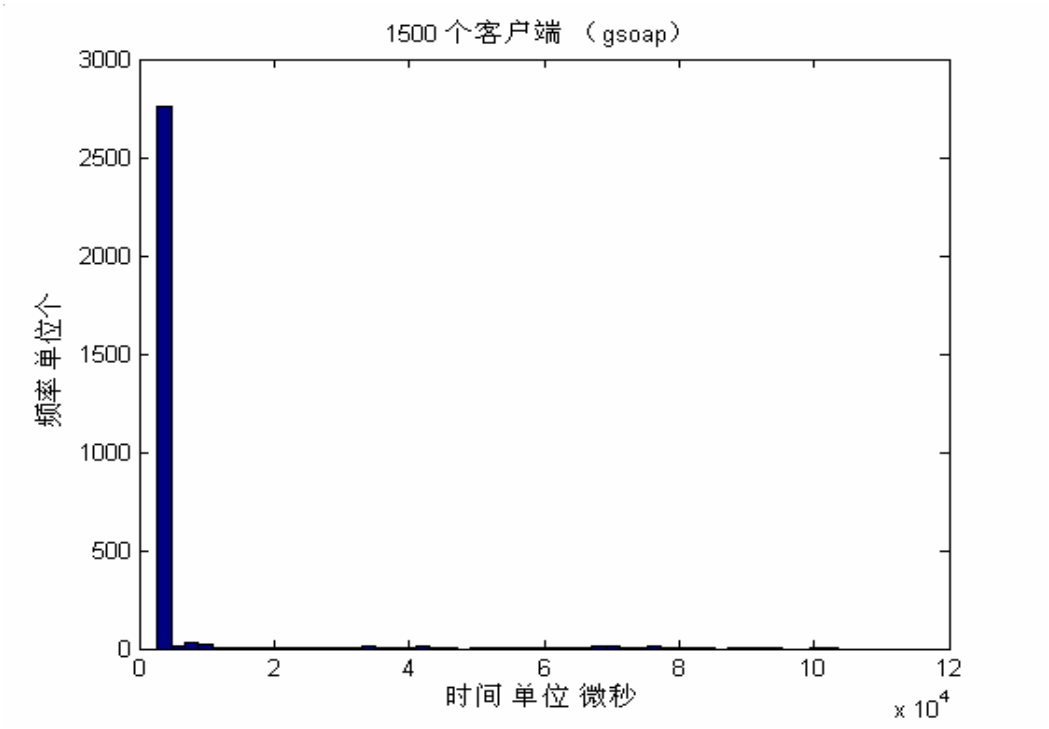

图3.8　1500个客户端基于Gsoap的Web服务PRT直方图

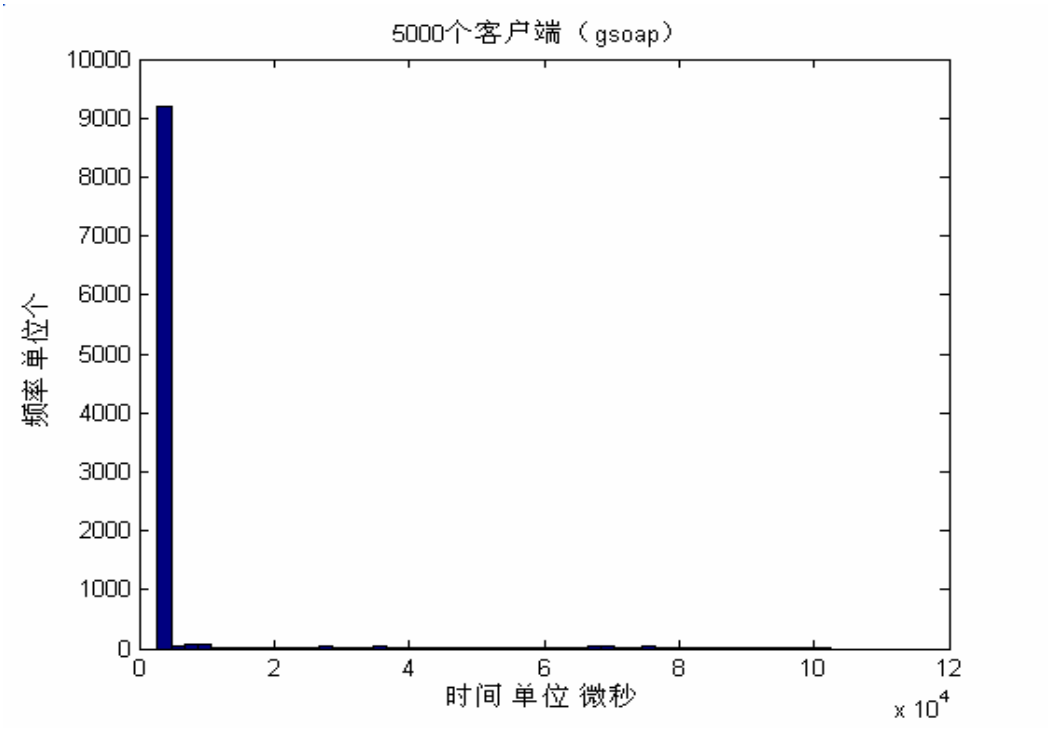

图3.9　5000个客户端基于Gsoap的Web服务PRT直方图

由图可知，应用完成服务的时间是相当短，而且很稳定的集中于一个时间段。同时可以看出应用完成服务的时间与负载数关系不大。标准方差见表3.3.

表3.3　Web服务（Gsoap）PRT标准方差

负载数	*200*	*1500*	*5000*
标准方差	0.0835	0.1275	0.1305

从这个表中可以看到，完成服务的时间是随着负载的增加离散化，也就是说完成服务的时间不稳定。

基于 Axis+Java 的 Web 服务

同样通过数据分析，发现采用 Axis 定制的服务随着负载有明显的增加，但分布方式却完全类似，如图 3.10，3.11，3.12 所示。

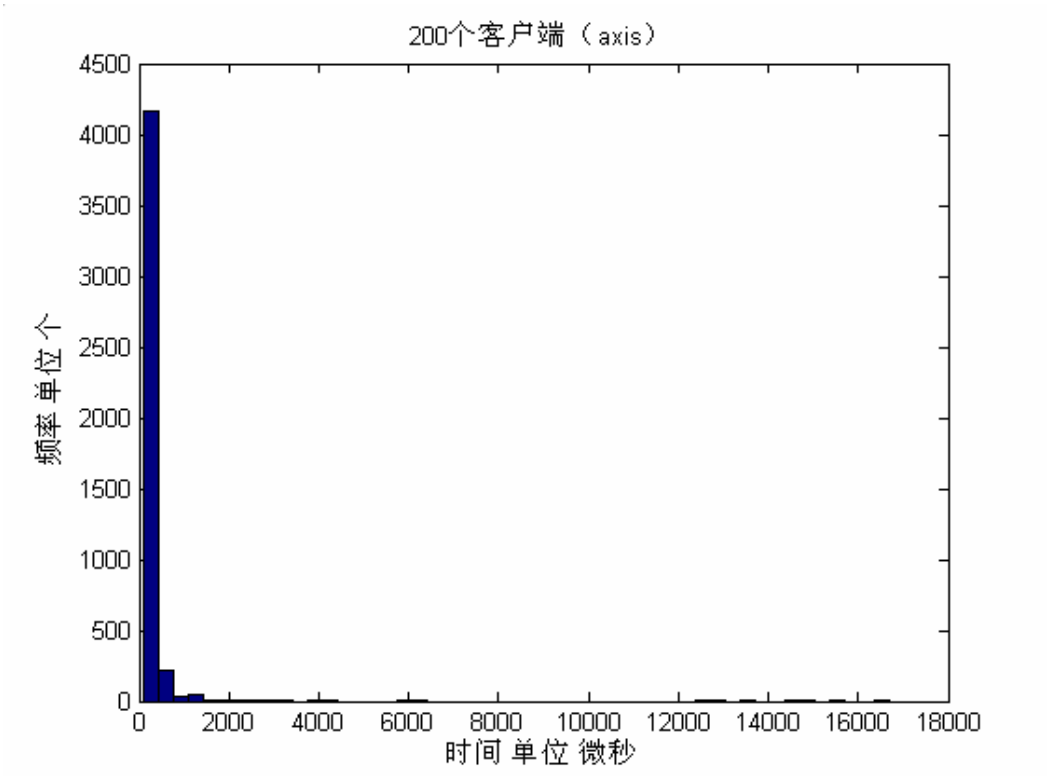

图3.10　200个客户端基于Axis的Web服务PRT直方图

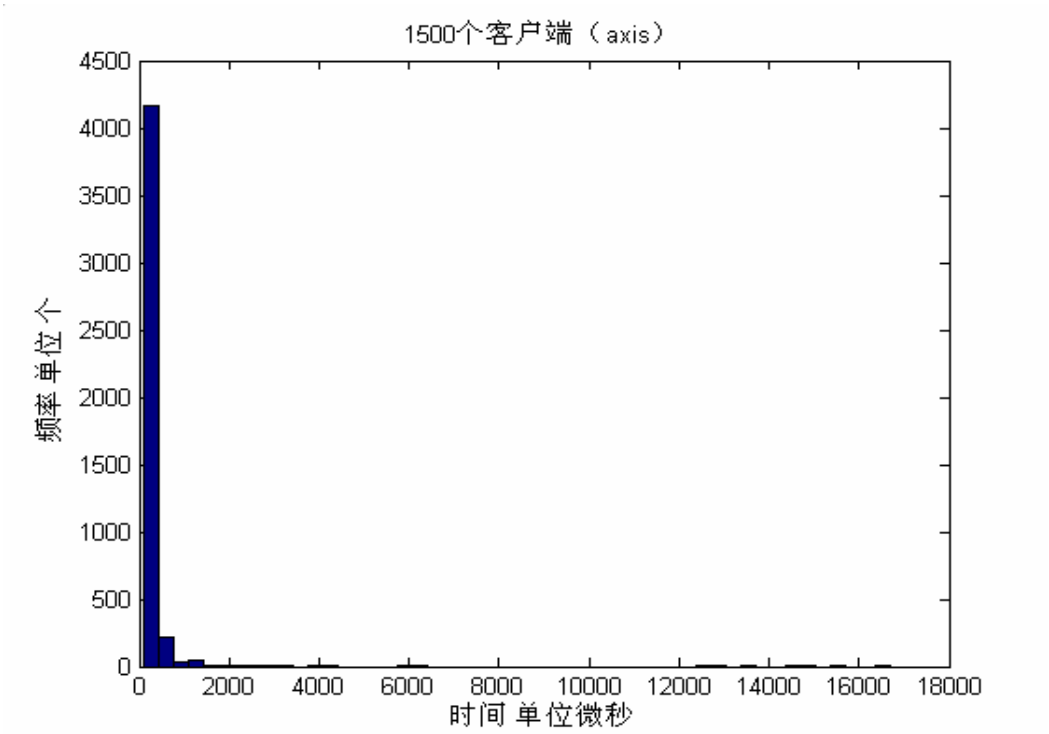

图3.11 1500个客户端基于Axis的Web服务PRT直方图

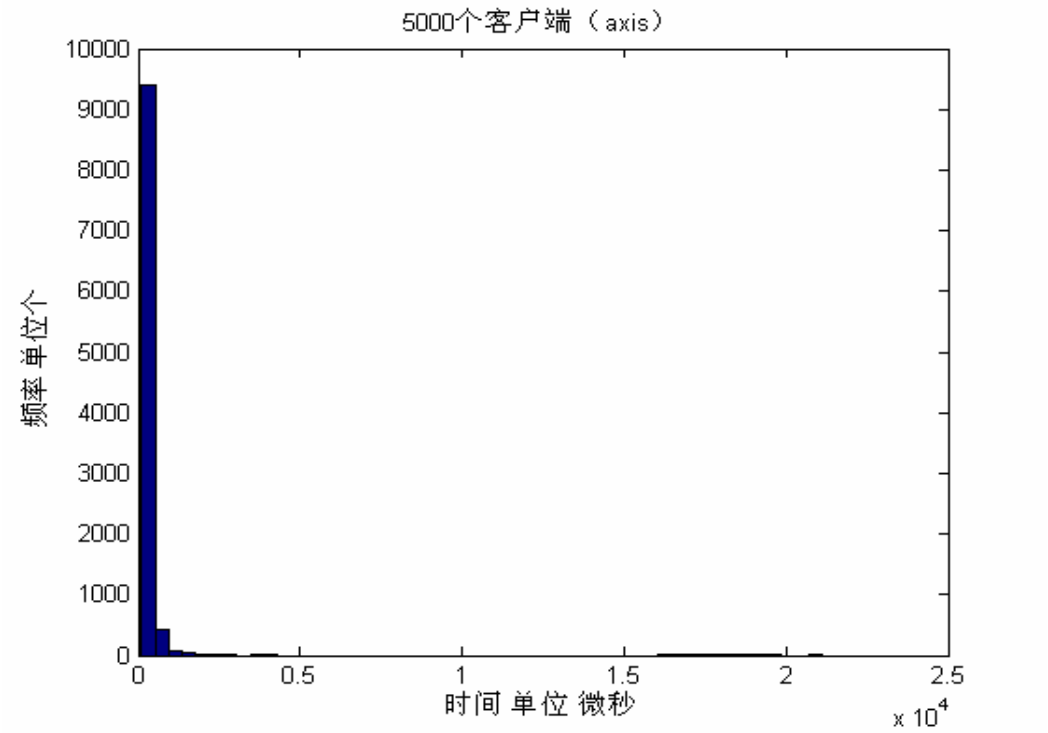

图3.12 5000个客户端基于Axis的Web服务PRT直方图

由图 3.10，3.11，3.12 可以看出，在一定负载下，采用 Axis+Java 定制的 Web 服务在完成服务的时候性能稳定，完成时间基本上都在一个时间段内。标准方差见下表：

表3.4　Web服务（Axis）PRT标准方差

客户端数	*200*	*1500*	*5000*
标准方差	0.0747	0.0408	0.0603

可以看出，方差随着时间变化不大，但都很小，也就是说 Web 服务性能稳定，同时从上面的图中能看到平均处理时间随着负载的增加有明显的增加，这个与采用 Gsoap 定制的 Web 服务有所不同。具体分析将在下一节中进行。

3.2 动态负载测试

动态测试主要测试的是 Web 服务的负载特性。分别采用两种工具：Axis 和 Gsoap 定制发布服务，采用多个客户端同时发送请求的方式对 RTT (Round Trip Time) , RPT(Response Time), OHT(Overhead Time) ,PRT(Process Time)等参数测量。发送请求数的不同，最终测得的参数也不同。通过分析这些参数随负载增加而变化的规律，就可以了解到基于不同技术的 Web 服务的负载特点。

3.2.1 RTT 负载特性

不同负载下 Web 服务平均 RTT 时间显著不同，如图 3.16 所示：

从图中可以看到，初期采用 Axis 发布的 Web 服务的 RTT 时间明显要小于采用 Gsoap 工具定制的 Web 服务，而且如果负载在一定数量之下，ＲＴＴ随负载的增加变化不大，但超过一定数量之后负载数的增加就会明显影响到 RTT 时间，RTT 事件会显著增加，呈指数增长。而基于 Gsoap 定制的服务，不管负载多大，RTT 时间随着负载数的增加基本上都是线性增加，斜率随着负载数的不同略有不同：在负载较小的时候斜率较大，RTT 增加比较快速，而在负载较大的时候斜率反而变小，RTT 增加速度减缓。

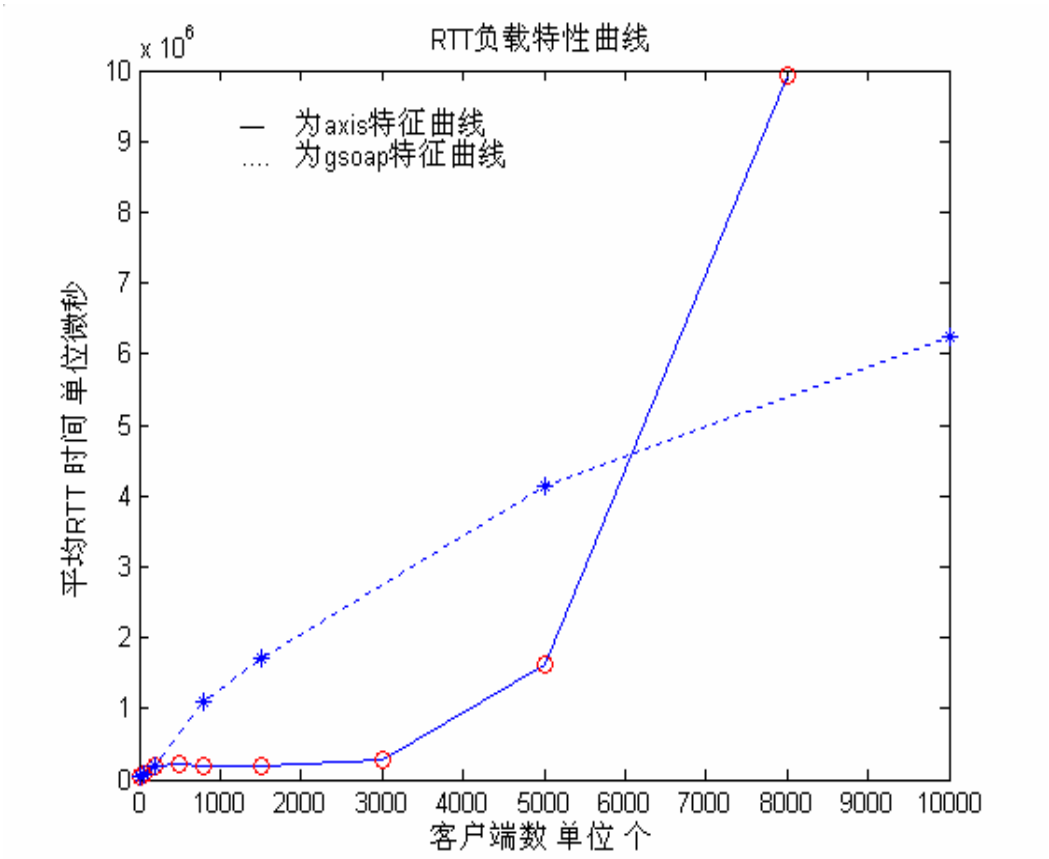

图3.13 RTT动态负载曲线

从图 3.13 可以看到，初期采用 Axis 发布的 Web 服务的 RTT 时间明显要小于采用 Gsoap 工具定制的 Web 服务，而且如果负载在一定数量之下，RTT 随负载的增加变化不大，但超过一定数量之后负载数的增加就会明显影响到 RTT 时间，RTT 事件会显著增加，呈指数增长。而基于 Gsoap 定制的服务，不管负载多大，RTT 时间随着负载数的增加基本上都是线性增加，斜率随着负载数的不同略有不同：在负载较小的时候斜率较大，RTT 增加比较快速，而在负载较大的时候斜率反而变小，RTT 增加速度减缓。

很明显，基于 Axis 和 Gsoap 两种不同工具的 Web 服务，RTT 随负载变化的趋势恰恰相反。在负载较小的时候基于 Axis 的 Web 服务性能表现良好，RTT 时间短而且增加速度缓慢；而当负载较大，超过一定界限的时候，基于 Gsoap 的 Web 服务就更加适用，其 RTT 时间相比要短，对负载的敏感程度也远没有基于 Axis 的 Web 服务强烈。

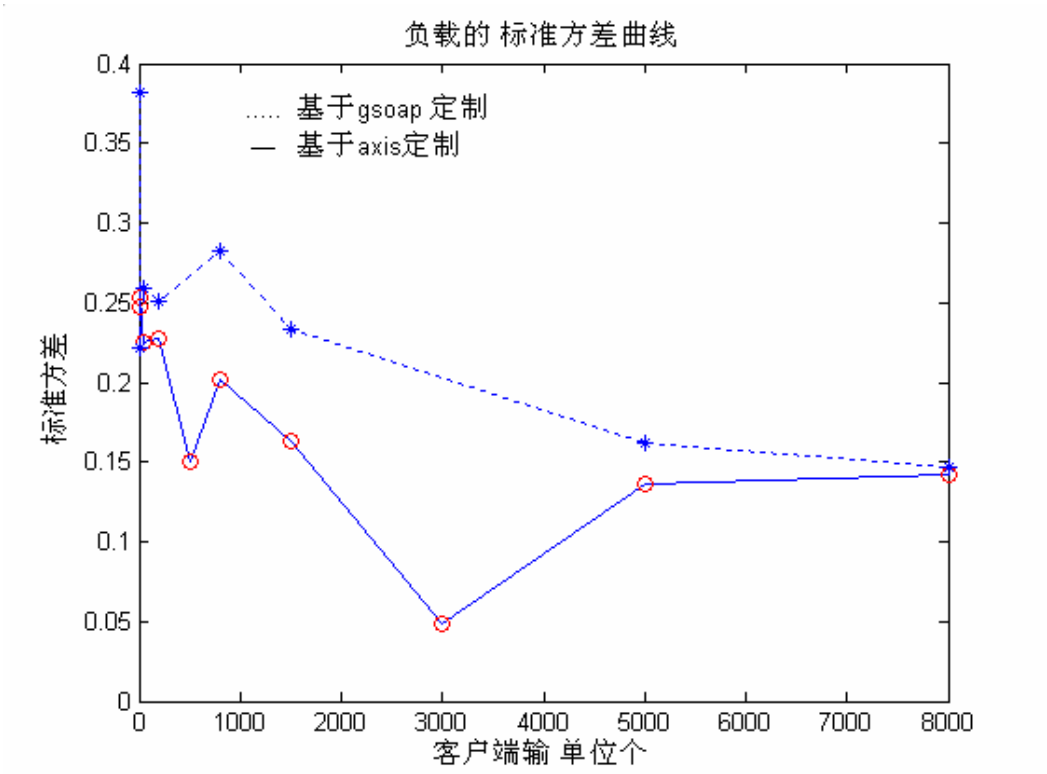

图3.14 RTT动态负载标准方差曲线

图 3.17 显示的是 RTT 时间随着负载增加的标准方法的变化。由上图可以看到，基于 Axis 定制的 Web 服务的标准方差在大部分负载状况下都要小于基于 Gsoap 定制的 Web 服务。标准方差越小，就说明 RTT 时间分布相对的越集中，Web 服务性能表现的就越稳定。所以可以说基于 Axis 定制的 Web 服务的性能表现要更加稳定。

3.2.2 PRT 负载特性

客户将自己的请求以 soap 包的形式发送到 Web 服务，Web 服务会有一个 soap 包解析器（soap engine）将 soap 解析，提取出相应参数递交给应用程序，也就是用户想要发布的服务。这个服务根据递交的参数计算后将结果返回。发布 Web 服务，用户只需编写完成服务的应用程序，然后将这个应用程序放置在 Web 服务容器中，这个应用程序就可以当作 Web 服务进行发布。容器不仅仅完成了对 soap 包的解析和封装，它还有一个作用就是管理和支持应用程序的执行。这个功能直接影响着 Web 服务的最终性能。PRT（process time）就是用来描述这项功能表现状况的参数。在不同负载下，平

均 PRT 时间会有很大的不同，理想中的 PRT 应该是快捷和稳定的，而且应该是负载不敏感的。

图 3.15 表现的是基于两种不同工具的 Web 服务 PRT 时间随负载的变化曲线。

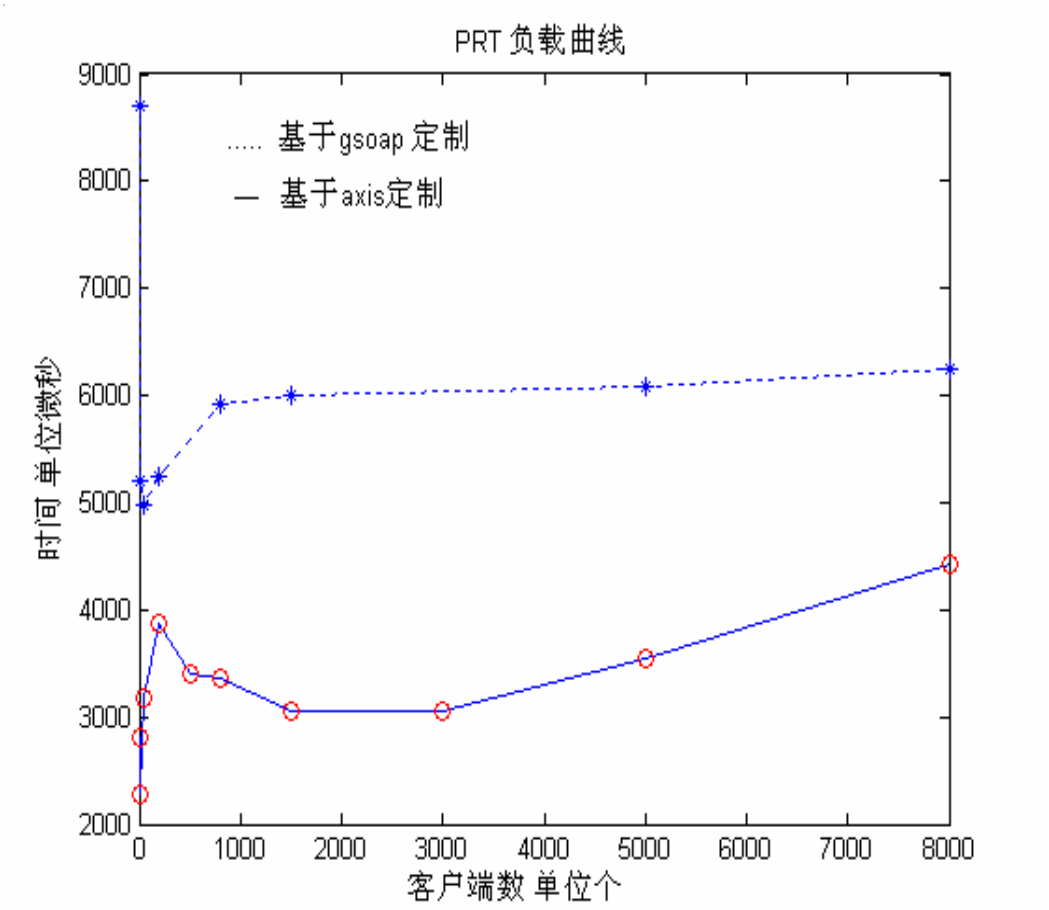

图3.15 PRT动态负载曲线

从图 3.15 可以看到，采用 Axis 定制的服务 PRT 时间明显要小，也就是说采用 Axis 定制的 Web 服务性能较好。同时可以看出，在负载较小的时候，采用 Axis 定制的 Web 服务 PRT 时间很小，随着负载的增加而显著增加，到达一定程度后基本稳定，而采用 Gsoap 定制的 Web 服务 在负载较小的时候 PRT 时间较大，随着负载的增加会显著减小，同样，到达一定程度后也趋于稳定。

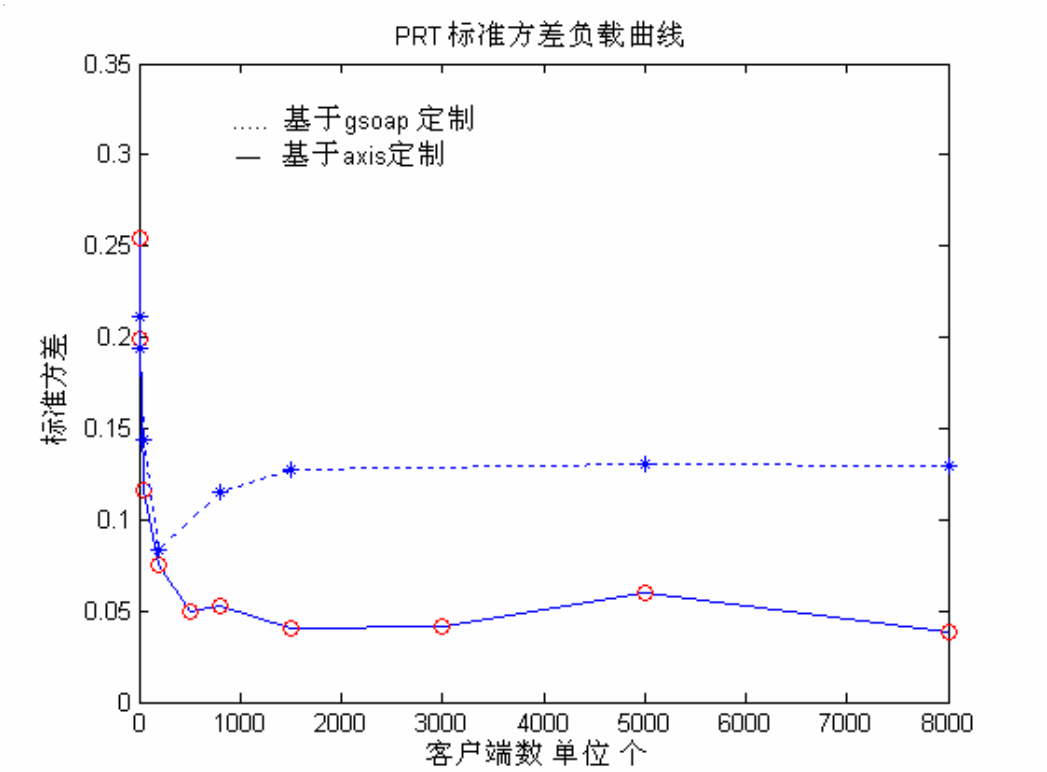

图3.16 PRT动态负载标准方差曲线

从 3.16 标准方差图中可以看出，低负载情况下，两种工具定制的 Web 服务，其 PRT 标准方差都比较大，但随着负载的增加显著减小，所不同的的是采用 Gsoap 工具定制的 Web 服务的 PRT 标准方差在负载超过一定限度后又有所增加，而后趋于稳定；基于 Axis 定制的 Web 服务其 PRT 标准方差一直减小，超过一定限度后基本稳定。

总的来说，这两种方法定制的 Web 服务随着负载的增加 PRT 缓慢增加，对应用程序的支持相当稳定。不过不管从哪方面来说，才用 Axis 定制的服务对应用程序的支持要更好。

3.2.3 OHT 负载特性

OHT（overhead time）这个参数记录的是 Web 服务 中间件耗费时间。这个时间主要消耗在对 soap 包的解析封装和与服务应用程序的通讯方面。由于这个时间并不是直接消耗在对外提供服务，而是消耗在通讯交互等辅助方面，而且是不可或缺的，所以叫做 overhead time。这个时间参数体现的是

Web 服务工具对通讯交互方面的功能。尽可能的压缩这一段时间是 Web 服务工具设计者努力的方向。

图 3.17 显示了 OHT 在不同负载下平均值的变化：

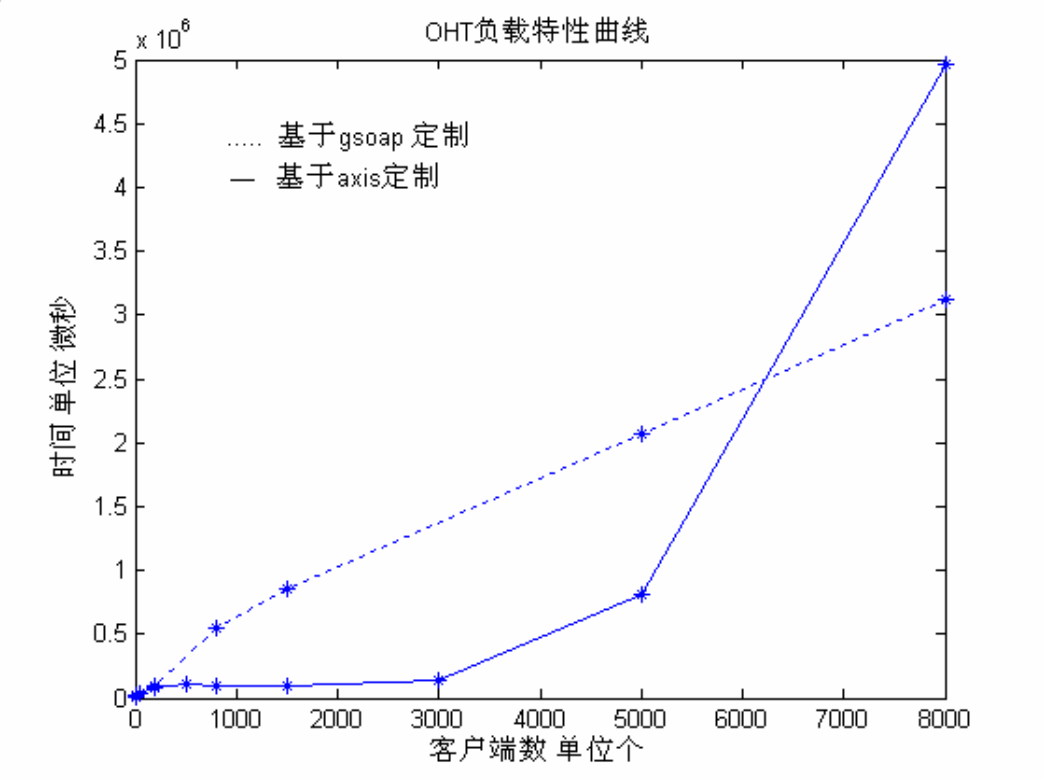

图3.17 OHT动态负载曲线

从图中可以看出，基于 Gsoap 工具的 Web 服务 OHT 与负载显著呈线性关系，在低负载情况下，Gsoap 工具表现出来的 OHT 要大于 Axis 工具表现出来的 OHT。而在重负载或者过负载情况下，Gsoap 的 OHT 要优于 Axis 的表现。采用 Axis 发布的 Web 服务的 OHT 时间明显要小于采用 Gsoap 工具定制的 Web 服务，而且如果负载在一定数量之下，OHT 随负载的增加变化不大，但超过一定数量之后负载数的增加就会明显影响到 OHT 时间，OHT 时间会显著增加，呈指数增长。

很明显，基于 Axis 和 Gsoap 两种不同工具的 Web 服务，OHT 随负载变化的趋势恰恰相反。在负载较小的时候基于 Axis 的 Web 服务性能表现良好，OHT 时间短而且增加速度缓慢；而当负载较大，超过一定界限的时候，基于 Gsoap 的 Web 服务就更加适用，其 OHT 时间相比要短，对负载的敏感程

度也远没有基于 Axis 的 Web 服务强烈。这个情况同 RTT 表现出来的情况十分相似。

第4章 结论及展望

4.1 Web 服务性能评估结论

在上一章节中，对 Web 服务分别通过静态和动态进行性能测试和分析。

在静态测试中，通过在低负载，过负载和中间负载三种情况下对 RTT，PRT 参数分布状况的分析，可以看到在不同情况下 RTT 及 PRT 的分布规律。标准方差越小，意味着在这个负载下 RTT 或者 PRT 时间越集中，性能表现越稳定。通过标准方差和直方图可以看到，基于 Axis 的 Web 服务不论是 RTT 还是 PRT 也不管在什么负载下，性能都要比基于 Gsoap 的 Web 服务稳定。

动态测试是测试 Web 服务随负载数量的不同性能表现得变化。这个对于 Web 服务来说是一个很重要的指标。了解了一种 Web 服务负载特性，就能够很好的针对需求来定制服务，使其既不浪费，又能够满足需求。在这项测试中可以看到，基于 Axis 的 Web 服务在负载数量较小的情况下有很大的优势：不论是性能稳定程度还是 RTT，PRT 或者 OHT 时间都有很大的优势。但从上面的分析中可以看到，当负载数量大于一定限度后，基于 Axis 的 Web 服务的各项性能指标会以很快的速度随着负载数的增加而增加。而基于 Gsoap 的 Web 服务从一开始各项指标就差不多以线性随着负载数增加，当负载到达一定程度的时候基于 Gsoap 的 Web 服务性能将超过基于 Axis 的 Web 服务。从这个角度来看，Gsoap 工具适合于构建访问量很大的 Web 服务，而对于小型的访问量不大的 Web 服务，采用 Axis 工具构建是一个很好的选择，而且就从可移植性和构建建议程度来说，Axis+Java 也要比 Gsoap 有优势。总的来说，Axis 与 Gsoap 各有特点，如何选择工具需要从实际需求考虑。

通过测试可以看到，Web 服务技术就目前来说有很多优点，相当适合于 CI 的构建，但就其表现出来的性能来说距离 CI 的需求还有一定的距离，不论采用哪种工具，在访问量较大的情况下，支持时间（overhead time）的开销不容忽视，也就是说用户等待的时间大部分花在了与服务没有直接联系的

消息传递上面。同时往返时间随着负载的增加也增长较快，很难承担大访问量的 Web 服务构建。

4.2 展望

本文对最常用的两种构建 Web 服务的工具进行了测试，得到了很有意义的结果。但就从评判 Web 服务工具在 CI 平台下性能表现这个方面来说还有两项工作需要亟待解决：

首先，本文虽然提出了不少参数用来描述 Web 服务性能，进而评判这两种构建工具，但并没有将这两种工具所有有用的特定都描述清楚。在 CI 的构建过程中，程序重复利用性，可移植性这两个特点对于技术和工具的推广至关重要。这是因为为了构建 CI，首先需要将目前现有的大量应用程序转变为 Web 服务，程序的重复利用可以使现有应用程序很容易的被封装为 Web 服务而不需要很多额外的工作，其次 CI 环境的复杂性也要求 Web 服务构建工具好的移植性，以减少开发工作量。就实际使用来说，使用 Axis 来定制 Web 服务要远远简单于采用 Gsoap 定制 Web 服务。程序员几乎可以不作任何改动的就将原有程序封装为 Web 服务然后将其发布。同时 Java 相对 C/C++来说对平台的依赖性也很小，有很好的可移植性。而采用 Gsoap 工具定制的服务或者客户端当移植到其他电脑上时需要重新进行编译才能够运行。所以如何能够用简单易行的参数来描述这两个特性，是以后进一步研究的内容。

其次，虽然对两种工具进行了比较，测得了各项参数。但如何将这些成果和实际应用联系在一起？这就需要将实际对 Web 服务的需求用参数的方式描述出来，例如在 CI 环境中，需要将某一个天文台开发成一个能满足若干人使用的 Web 服务，那么为了能够让这些用户满意，这个 Web 服务的性能特性是什么？为了让其有一定的抗压能力，其负载曲线又应该是什么？在 CI 环境下，如何描述一个服务的实际性能需求，然后又能够针对性地选择工具和方法进行开发？这将是未来构建 CI 的一个亟待解决的问题。

插图索引

表格索引

参考文献

[1] LIGO – Laser Interferometer Gravitational-wave Observatory. http://www.ligo.caltech.edu

[2] LSC – LIGO Scientific Collaboration. http://www.ligo.org

[3] Revolutionizing Science and Engineering Through Cyberinfrastructure. Report of the National Science Foundation Blue-Ribbon Advisory Panel on Cyberinfrastructure, 2003, 4～13

[4] NEES – Network for Earthquake Engineering Simulation. http://www.nees.org

[5] OSG – Open Science Grid. http://www.opensciencegrid.org

[6] NVO – US National Virtual Observatory. http://www.us-vo.org

[7] TeraGrid. http://www.teragrid.org

[8] Eric Newcomer Gregg Lomow. Understanding SOA with Web 服务. 徐涵译. Newcomer,E. 2006,5～23

[9] W. K. Chan, S. C. Cheung, and K. P. H. Leung. “Towards a Metamorphic Testing Methodology for Service-Oriented Software Applications,” in Proceedings of the First International Workshop on Services Engineering (SEIW),2005

[10] A. Ng, S. Chen, and P. Greenfield, An evaluation of contemporary commercial SOAP implementations, in Proceedings of the Fifth Australasian Workshop on Software and System Architectures, Melbourne, 2004

[11] Giovanni Denaro, Andrea Polini, Wolfgang Emmerich. Early Performance Testing of Distributed Software Applications, in WOSP 04, January 14-16, 2004, Redwood City, CA

[12] 武海平,蒋东兴,程志锐,康晓宁.服务器通用性能测试系统的设计与实现,小型微型计算机系统, 2003, 24 (2)

[13] 王惠,陈莘萌,基于 WWW 的 Internet 网络计算模式的性能分析[J].计算机工程与应用. 2001,14(37):72～73

[14] Giovanni Aloisio, Massimo Cafaro, Italo Epicoco, Daniele Lezzi. The GSI plug-in for gSOAP: Enhanced Security, Performance, and Reliability, Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'05), 2005

[15] Liming Zhu, Ian Gorton, Yan Liu. Model Driven Benchmark Generation for Web 服务, IW-SOSE'06, May 27–28, 2006, Shanghai, China, 2006

[16] http://ws.apache.org/axis/java/index.html

[17] http://gsoap2.sourceforge.net/

致　　谢

首先，由衷地感谢我的导师，尊敬的曹军威研究员。他给了我研究工作上的悉心指导；同时他也有着优良的研究作风，对我的影响非常之大，对我以后的研究也将产生很积极的作用。

也要感谢实验室的张文师兄。本文研究过程中，给予我大量的帮助和支持，同时也有不少工作是和他讨论完成，这让我能够很快的度过前期准备，完成课题。他的扎实的专业功底和宽泛的知识基础给我留下很深的印象。

感谢所有关心和支持我的人。

声　　明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签　名：_____日　期：_____