基于 CORBA 的中间件管理系统的设计与实现*

曾 嵘, 杨树强, 贾 焰, 张 文
(国防科学技术大学 计算机学院, 湖南 长沙 410073)

摘 要: 大规模事务处理系统实现了配置数据的添加/删除/修改, CORBA 服务对象、主机的监测和实时状态显示, CORBA 服务对象的运行维护以及部署等功能。

关键词: CORBA 服务对象; 配置; 监测; 运行维护; 部署; 冗余服务
中图法分类号: TP915.07 文献标识码: A 文章编号: 1001-3695(2005)02-0118-03

Design and Implementation of CORBA-based Middleware Management

ZENG Rong, YANG Shu-qiang, JIA Yan, ZHANG Wen
(School of Computer Science, National University of Defense Technology, Changsha Hunan 410073, China)

Abstract: The system implements the following functions: adding/deleting/updating of the configuration data, inspecting and the state displaying of CORBA-based service objects and server, maintaining and disposing of CORBA-based service objects.
Key words: CORBA-based Service Object; Configurate; Inspect; Maintain; Dispose; Redundant Service

随着个人计算机的普及以及网络通信技术的发展, 分布计算[1]逐渐成为计算技术的主流。为了使用户能够透明、有效地共享分布在网络上的信息资源和计算资源, 分布计算中间件成为实现分布计算的关键技术之一; 并且, 随着大规模事务处理、军事抗毁等应用需求的推动, 越来越多的分布式应用采用了冗余服务技术来提高系统的性能和可靠性。

实际应用中, 由于分布式计算的规模较大, 中间件众多, 加之冗余服务技术的应用给系统的管理带来了很多难题, 我们开发了一个大规模事务处理系统。该系统采用 CORBA[2] 来构建事务处理中间件, 使用符合 CORBA 3.0 标准的 StarBus 4.0[3] 作为系统开发和运行平台, 应用中间件层次的冗余服务技术来提高系统的性能和可靠性。目前该系统包含 20 多台主机, 每台主机上驻留着若干类 CORBA 服务对象和它们的冗余对象, 整个系统中的服务对象有 100 多个, 如此众多的主机和服务对象使管理和维护成了一个难题。手工管理存在着许多问题, 如主机或服务对象故障不容易被发现, 服务对象的部署和调整都很困难, 用户对服务对象在系统中的分布没有一个直观的了解等。系统还使用了 StarBus 的负载平衡服务 AFLS(Application level Fault-tolerant and Load-balancing Service)[4] 来平衡系统中各冗余服务对象之间的负载。服务对象的冗余和负载平衡提高了系统的利用率和吞吐率, 达到了一定程度上的容错, 但是当某类服务对象的冗余对象失效时, AFLS 和用户是觉察不到的, 这时服务对象冗余数量的不足就会降低系统的性能, 当冗余数量减少为 0 时, 系统将不能正常运行。

由此可见, 分布式计算中间件的管理和维护工作是十分重要的。目前一些计算机研究机构和公司都提出了相应的管理协议和算法, 并实现了一些管理系统。但是这些管理系统并不适用于我们开发的大规模事务处理系统。近年来, OMG 组织提出了基于 CORBA 的构件模型 CCM(CORBA Component Model), 有效地解决了分布式计算中间件管理难的问题。但是, 面对复杂庞大的 CCM 规范, 目前还没有一个完善、成熟的管理系统可以使用。鉴于以上所述, 我们针对大规模事务处理系统的特点, 设计并实现了一个 CORBA 服务对象的配置管理与运行维护系统。

收稿日期: 2004-03-15; 修返日期: 2004-06-07
基金项目: 国家自然科学基金资助项目(60003001); 国家“863”计划资助项目(863-306-ZD02-01-2)

1 系统结构

本系统按照功能可以分为六个模块: 库管理模块、服务对象和主机监测模块、服务对象运行维护模块、服务对象可执行文件管理模块、服务对象部署模块、服务对象启动/关闭模块。这六个功能模块的逻辑关系可由图 1 表示。

每个模块的功能如下:

(1) 库管理模块。它负责将主机和服务对象的配置信息和状态信息添加/删除/修改到对象配置和状态信息库中。

(2) 服务对象和主机监测模块。它负责监测指定主机上指定类型的 CORBA 服务对象的状态和负载, 以及主机的运行状态。

(3) 服务对象运行维护模块。它负责将服务对象当前的冗余数量自动调整到用户配置的冗余数量。

(4) 服务对象可执行文件管理模块。它负责管理各类不同版本的服务对象的可执行文件和它们的信息。

(5) 服务对象部署模块。它负责将服务对象的可执行文件部署到指定主机上, 部署将按照用户配置信息自动进行。

(6) 服务对象启动/关闭模块。它负责启动/关闭服务对象。

2 系统设计与实现

本系统有着友好的 GUI 界面, 用户可方便添加、修改、删

除配置信息, 直观地查看服务对象的运行状态和分布情况。GUI 是在 Windows 操作系统下利用 Visual C++ MFC[5] 开发实现的, 其中, 服务对象和主机监测模块、服务对象运行维护模块、服务对象启动/关闭模块由驻留在主机中的 CORBA 服务对象来实现, 其他模块由本地的几个类来实现。

2.1　库管理模块

在服务对象的管理中有很多用户配置信息和服务对象的状态信息, 库管理模块就是将这些信息添加、修改、删除到对象配置和状态信息库中。本系统我们采用现在较为流行的 XML[6] 技术, 使用多个结构化 XML 文档建立对象配置和状态信息库来存储数据。该模块的具体实现是通过 DOM(Document Object Model) [6] 接口, 使用 LoadXml, Insert, Query, Delete, Update, SaveXml 等类来操作 XML 文档, 以完成对相关的配置信息和状态信息的添加、修改、删除工作。DOM 是一个对象化的 XML 数据接口, 它由一组对象组成。在结构化的 XML 文档中, 信息存储格式是按层次化的树型结构组织的。当 DOM 处理 XML 文档时, 首先要加载 XML 文档并在内存中生成 DOM 节点树。DOM 节点树的结构如图 2 所示。通过 DOM 接口, 可以对内存中的节点树按指定方式进行遍历, 找到相应节点后可以对数据进行读取、添加、修改以及删除等操作。

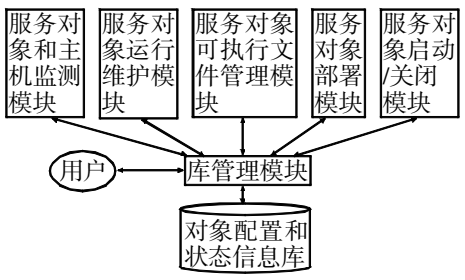

图 1　系统结构图

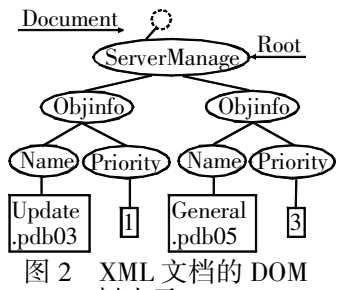

图 2　XML 文档的 DOM 树表示

XML 格式相对于其他信息存储格式的优点是: XML 允许开发者建立它们自己的、用于存储信息的标记结构; XML 格式以文本为基础, 这使得它能被更容易地阅读; XML 格式便于传输, 利用它与其他组织交换数据非常方便, 等等。

2.2　服务对象和主机监测模块

对于一个大规模事务处理系统来说, 主机和服务对象的数目众多, 因此对系统进行定时监测并报告主机状态、服务对象的分布和负载是非常重要的, 有利于及时地纠错和对现有的运行规模进行调整。

服务对象和主机监测模块主要由 InspectObj、LoadCalculate、两个 CORBA 服务对象、一个服务方请求级截获器 LoadInterceptor 和一个 InspectNode 类组成。InspectObj 负责服务对象的动态注册与去除、服务对象的状态监测与负载的定时获取。刻画系统负载的参数一般有: CPU 利用率、内存利用率、进程个数、连接数、请求数量。在本系统中以服务进程接收处理的请求数量来刻画系统负载。CORBA 2.3 规范引入的截获器机制为我们获取请求数提供了有力支持, 截获器可以截获 ORB 内核的执行过程, 具体来讲可以截获发送和接收的请求及应答, 这使得可以获取请求数而又不影响应用的透明性。LoadInterceptor 是我们在 ORB 内核注册一个服务方请求级截获器。由它截获请求的接收与完成, 并通知 LoadCalculate 完成请求的计数。InspectNode 负责定时监测主机的状态。服务对象和主机监测模块的流程如图 3 所示, 其中粗线条表示数据流向。

InspectObj 定期向 LoadCalculate 获取负载抽样, 有两种方式定义某一时刻的负载: 一种方式将当前正在处理的请求数作为负载, 然而考虑图 4 所示情况, 系统取样周期为 T, $t1$ 时刻请求处理全部处理完且直到 $t2$ 时刻才有新的请求到达。尽管上一周期完成了 10 个请求, 并马上又将有 10 个新的请求到达, 而负载的采样值仍然为 0。由此可以看出时间点的负载很难真正反映系统负载, 必须从时间单位上去考虑。为此将负载定义如下:

取样时刻负载值 = 过去时间间隔内处理完毕的请求数 + 当前活动请求数

这样定义既考虑了工作量的完成又考虑了负载的延续, 是一种更为合理的方案。

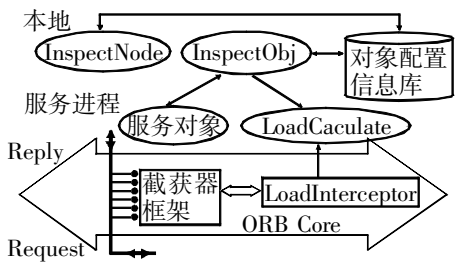

图 3　流程图

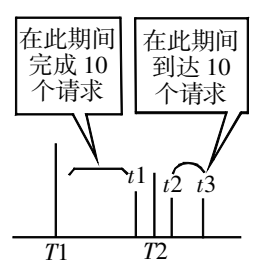

图 4　系统请求负载

具体实现流程是: 当启动一个服务进程时, 创建了一个服务对象和一个 LoadCalculate, 然后将服务对象和 LoadCalculate 的对象引用注册到 InspectObj, InspectObj 再将这些信息添加到对象配置和状态信息库中。LoadInterceptor 在规范中定义的四个截获点通知 LoadCalculator 执行以下操作:

receive_request_service_contexts: 活动请求数加 1
send_reply: 活动请求数减 1, 完成请求数加 1
send_exception: 活动请求数减 1, 完成请求数加 1
send_other: 活动请求数减 1, 完成请求数加 1

InspectObj 根据记录在对象配置和状态信息库中用户输入的配置信息, 定时监测指定的服务对象是否存在, 若不存在, 则删除对象配置和状态信息库中它的信息; 若存在, 则定期调用 LoadCalculate 的 get_loads( ) 方法时执行: 将活动请求数与完成请求数之和作为负载值返回, 并将完成请求数清 0, 然后 InspectObj 将得到的负载值添加到对象配置和状态信息库中。用户可以通过对象配置和状态信息库, 查看每个服务对象在所有主机上的分布情况和负载值, 以及在每台主机上所有服务对象的运行情况。

2.3　服务对象运行维护模块

服务对象运行维护模块负责调整服务对象的冗余数量, 主要由驻留在每个主机上名为 StartObj 的 CORBA 服务对象和一个本地的 CompareRedundance 类组成。该模块的设计实现是以一个服务进程只包含一个 CORBA 服务对象为前提条件的。CompareRedundance 负责定时将服务对象当前的冗余数量与用户配置的冗余数量相比较, 若服务对象当前的冗余数量小于用户配置的冗余数量, 则通过 StartObj 启动相应数量的服务对象, 使服务对象当前的冗余数量等于用户配置的冗余数量; 若服务对象当前的冗余数量等于或大于用户配置的冗余数量, 则不做动作。

2.4　服务对象可执行文件管理模块

在大规模事务处理系统中, 服务对象的数目和种类很多, 而且同一类服务对象可能存在多个版本, 因此手工管理这些对象将十分困难。服务对象可执行文件管理模块通过一个对象库对服务对象的可执行文件进行管理, 简化了管理工作。对象

库是本地主机上的一个目录。对象库中保存着不同版本的各类服务对象的可执行文件;在对象配置和状态信息库中有对象库中所有服务对象可执行文件的完整描述信息,如功能标志、版本标志、运行环境、初始化参数等。该模块主要由 InsertObj, DeleteObj, UpdateObj 等类组成。InsertObj 类用来将服务对象可执行文件加入到对象库中,并在对象配置和状态信息库中记录相应的信息;DeleteObj 类用来删除对象库中的一个服务对象可执行文件和其在对象配置和状态信息库中的信息;UpdateObj 类用来修改服务对象可执行文件在对象配置和状态信息库中的信息。

2.5 服务对象部署模块

服务对象部署模块主要由 SendObj, DeployInfo 等类组成。SendObj 类主要是将用户选定的服务对象可执行文件以类 Ftp 方式通过网络传送到指定的主机上,以完成服务对象的自动部署工作,发送成功后系统自动将服务对象的信息、发送地址以及发送时间存入对象配置和状态信息库中,作为日志以便查阅。通过 DeployInfo 类,用户可以将手工部署的服务对象的信息输入到对象配置和状态信息库中,还可以更改和查看整个系统中服务对象可执行文件部署情况的信息。

2.6 服务对象启动/关闭模块

服务对象启动/关闭模块主要由驻留在每个主机上的 StatrObj 和 StopObj 两个 CORBA 服务对象组成。StatrObj 负责服务对象的启动;StopObj 负责服务对象的关闭。用户可以在本地通过本系统启动或关闭其他主机上的一个或多个指定的服务对象,同时系统会自动将这些服务对象的信息、启动或关闭的时间存入对象配置和状态信息库中,作为日志以便查阅。服务对象的关闭目前只是针对它离线的情况,在线的关闭一个正在工作的服务对象会出现很多问题。

3 结束语

面对大规模事务处理系统难以管理的问题,我们设计并实现了一个基于 CORBA 的服务对象配置管理与维护系统。本文详细介绍了系统各功能模块的设计和实现。本系统在应用过程中获得了较好效果,尤其是其良好的监控管理环境以及直观的视图极大地简化了系统管理员的工作。现在,由于一些大型的分布式应用系统要求服务对象一天 24 小时不间断地工作,这就对我们的管理维护提出了一些新的难题,比如怎样在线地关闭掉一个服务对象,旧的服务对象怎样在线地被新的服务对象所替换,等等。在下一步工作中,我们将对这些问题进行探讨和研究。

参考文献:

[1] Coulouris, J Dollimore, T Kindberg. Distributed Systems: Concepts and Design[M]. England: Pearson Education Limited, 2001.

[2] Michi Henning, Steve Vinoski. 基于C++ CORBA 高级编程[M]. 北京: 清华大学出版社, 2000.

[3] StarBus 3.0 程序员指南手册[M/CD]. 长沙: 国防科学技术大学计算机学院 613 教研室, 2002.10-16.

[4] 黄杰, 陈琳, 钱芳, 等. 面向 OLTP 应用的服务对象管理系统[J]. 计算机科学, 2000, 10(增刊).

[5] Scott Stanfield, Ralph Arvesen. Visual C++ 开发人员指南[M]. 北京: 机械工业出版社, 2001.

[6] 曾春平, 等. XML 从入门到精通[M]. 北京: 北京希望电子出版社, 2002.97-107.

作者简介:

曾嵘(1978-), 女, 硕士生, 主要研究方向为分布计算; 杨树强, 男, 研究员, 主要研究方向为分布计算和数据库; 贾焰, 女, 教授, 主要研究方向为分布计算和数据库; 张文, 男, 硕士生, 主要研究方向为网络技术。

(上接第 114 页) 内存量的增加将呈现波峰状态, MMXMemoryCopy() 和 memcpy() 的拷贝速度的峰值出现在 10K ~100K 之间, MyMemcpySSE() 和 memcpySSE() 的拷贝速度的峰值出现在 150K ~220K 之间; ②所有拷贝函数在内存量约等于 500KB 时拷贝速度突然下降,下降后拷贝速度基本上保持平稳速度不变; ③当拷贝内存大于 130KB 时,函数 MyMemcpySSE() 的拷贝速度超过其他三个函数的拷贝速度; ④在数据对齐的情况下,当拷贝内存大于 500KB 时,函数 MyMemcpySSE() 拷贝速度比 memcpySSE() 提高约 3%,比 MMXMemoryCopy() 提高约 38%,比函数 memcpy() 提高了约 45%; ⑤在数据没有对齐时,拷贝内存量大于 500KB 时,函数 MyMemcpySSE() 的拷贝速度比 memcpySSE() 提高约 240%,比 memcpy() 提高约 45%,比函数 MMXMemoryCopy() 提高约 38%; ⑥在数据对齐和没有对齐两种情况下 memcpySSE() 的拷贝速度相差很大,而 MyMemcpySSE() 的拷贝速度基本不变,可见函数 MyMemcpySSE() 克服了数据没有对齐对内存拷贝速度造成的消极影响。

5 总结

分析了用 SSE 指令优化内存拷贝函数代码以提高内存拷贝速度的方法,并在此基础上开发了快速内存拷贝函数。实验表明,内存拷贝函数在拷贝数据量大于 130KB 的数据时,其拷贝速度比其他几个现存的内存拷贝函数都快,且函数拷贝速度受数据对齐方式的影响很小。函数在大尺度海量图像管理系统中大大减轻了后台内存拷贝的压力。但是从实验结果看,峰值速度远远大于平稳速度是内存拷贝函数的普遍现象,是否可以通过适当的代码再优化,进一步提高平稳速度是下一步要做的工作。

参考文献:

[1] T Thakkar, T Huff. Internet Streaming Simd Extensions[J]. IEEE Computer, 1999, 32(12): 26-34.

[2] Jory Anick. Optimizing CPU to Memory Accesseson the SGI Visual Workstations 320 and 540[EB/OL]. http://www.joryanick.com/memcpySGI.htm, 2003-07-03.

[3] Nevrax Ltd. fast_mem.cpp[EB/OL]. http://www.nevrax.org/docs/doxygen/nel/fast__mem_8cpp-source.html, 2002.

[4] Agner Fog. How to Optimize for the Pentium Family of Microprocessors[EB/OL]. http://www.agner.org/assem/#optimize, 2004-02-10.

作者简介:

钱昌松(1979-), 硕士生, 主要研究方向为计算机图像处理; 刘志刚(1975-), 讲师, 博士生, 主要研究方向为计算机图像处理等; 刘代志(1960-), 教授, 博士生导师, 博士, 主要研究方向为信号与信息处理, 发表论文 100 余篇。