维普资讯 http://www.cqvip.com

基于CORBA构件模型的应用服务器中容器并发的研究与实现*)

肖和平 黄 杰 吴泉源 韩伟红 王克波

(国防科技大学计算机学院 长沙410073)

摘 要 容器支持高并发访问对于显著减少处理器的闲置时间、增加应用服务器的吞吐量、提高应用服务器的性能具有十分重要的意义。本文从分析StarCCM应用服务器中容器与构件、ORB以及POA的关系入手,从四个方面阐述了容器并发的设计及其实现方法。文中提出了一种高效可适配的线程池模型用于派发构件请求。测试结果表明,我们提出的这种线程池模型能够灵活地适应更大范围的突发请求数目,同时通过限制线程池中线程数目的上限来减少线程无限增长对系统性能带来的负面影响。

关键词 CORBA,容器,分布构件,线程池

Research and Realization of Container Concurrency in Application Server Based on CORBA Component Model

XIAO He-Ping HUANG Jie WU Quan-Yuan HAN Wei-Hong WANG Ke-Bo

(School of Computer, National University of Defense Technology, Changsha410073)

Abstract High concurrency accessing components in container is an important method to reduce idle time of processor and enhance throughput of application server. By analyzing the relationships of container, component, ORB and POA in StarCCM application server, the paper puts forward design and realization of container concurrency from four aspects. It proposes an efficient adaptive thread pool model which is used to dispatch request to component. By practical testing, it proves the thread pool model which we proposed is adaptive and flexible to wide range of concurrent request number. At the mean time, it reduces side effect to system of increasing thread by limiting the maximum size of the pool.

Keywords CORBA,Container,Distributed component,Thread pool

1 引言

随着分布式应用的发展,CORBA作为一种成熟的中间件技术,虽然具有跨操作系统、跨平台、跨语言等强大的互操作性,但其使用的复杂性成为它在企业应用中的主要障碍。

以中间件为核心的分布构件技术以降低中间件平台的复杂性、支持分布式企业应用和代码的二进制重用为目标,已成为中间件技术的主要发展方向。OMG组织提出了CORBA平台上的构件模型(CORBA Component Model,CCM),在经过几年的研究和多次修改之后,于2002年6月推出了CCM 3.0版本,该版本成为2002年12月发布的CORBA3.0标准[1]的一部分。

CCM是继Microsoft公司的MTS/COM+模型和SUN公司的EJB模型之后的第三种服务端构件模型。CORBA分布构件技术是一种在服务器端支持分布式应用开发和运行时管理的中间件技术,其思想是将分布应用的业务逻辑封装于构件之中。CCM构件模型继承了CORBA对象计算模型的各种优良特性,便于面向对象系统的开发,同时能与EJB和COM+构件互操作,并且在面向对象技术的基础上,以二进制代码为其重用的形式,支持二进制代码级的组装、配置和动态部署,从而能将多个相关的构件相互连接,从而组装成更大的应用。

我们基于自主产权的StarBus3.0分布计算平台(ORB),独立开发了符合CCM3.0规范、支持高并发访问的StarCCM-2.0[9]应用服务器。本文从分析应用服务器中容器与构件、ORB以及POA的关系入手,深刻阐述了容器并发的设计及实现方法。在深入分析ORB并发模型的基础上,提出了一种高效可适配的线程池模型用于定制StarBus3.0中的请求派发策略以派发构件请求,并与StarBus3.0中的简单线程池进行了对比测试,结果表明,这种线程池模型有助于应用服务器灵活地适应更大范围的突发请求数,同时通过限制线程池中线程数目的上限来减少线程无限增长对系统性能带来的负面影响,明显改善了应用服务器处理突发请求时的性能。

2 容器、构件、ORB及POA的关系

如图1所示,在CCM模型中,容器是一个建立在ORB[1]、POA[1]和一组CORBA公共服务基础上的服务器端运行框架,用于提供服务器端的构件运行和管理环境,支持对安

*)本课题得到国家自然科学基金(No.90104020)、国家"八六三"高科技研究发展计划基金资助(2001AA113020)、国家"九七三"重点基础研究发展规划项目(G1999032703)资助。**肖和平** 硕士研究生,主要研究方向为分布计算和构件技术。**黄 杰** 博士研究生,主要研究方向为多库系统和故障诊断。**吴泉源** 教授,博士生导师,主要研究方向为分布式计算及人工智能。**韩伟红** 副教授,硕士生导师,主要研究方向为分布式数据库和中间件系统。**王克波** 博士研究生,主要研究方向为分布计算和构件技术。

维普资讯 http://www.cqvip.com

全[6]、事务[4]、持久[5]等系统服务的集成。构件无需参与服务的管理和初始化等工作，能够以一种简单的、定义良好的方式来使用这些服务，甚至可以委托容器来管理服务。

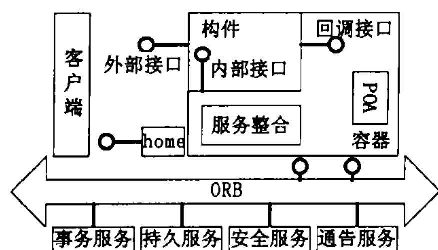

图1 CCM 模型

容器要实现对构件运行时的管理及服务的集成，就必须处于客户与构件之间的必经路径上，以截获 ORB 发送给构件的所有请求。POA 中定义的 ServantLocator 接口[1]为这一机制的实现提供了一种很好的解决方案，该接口定义如下：

```
module PortableServer {
  local interface ServantLocator : ServantManager {
  native Cookie;
  Servant preinvoke(in ObjectId oid,
    in POA adapter,in string operation,
    out Cookie the_cookie)
    raises(ForwardRequest);
  void postinvoke(in ObjectId oid,in POA adapter,
    in string operation, in Cookie the_cookie,
    in Servant the_servant);
  };
};
```

具有 USE_SERVANT_MANAGER 和 NON_RETAIN 策略值的 POA 并不将对象与 servant 之间的关联存储在 AOM(Active Object Map，激活的对象映射)表中[7]，而是在请求到达时，调用向该 POA 注册的 ServantLocator 对象的 preinvoke 方法以获取执行请求所需要的 servant。因此，容器只需要实现一个支持 ServantLocator 接口的 servant 管理类 ExecutorInvoker，并在创建具有相应策略值的 POA 时，向其注册一个 ExecutorInvoker 实例。这样，当构件的请求到达时，POA 将调用容器中 ExecutorInvoker 对象所实现的 preinvoke 方法，从容器获取一个要调度给请求的 servant，从而实现了容器对每一次构件请求的截获。在请求的执行返回之后，POA 将调用 ExecutorInvoker 对象的 postinvoke 方法，以便容器执行一些后期处理函数，比如同步锁的释放、事务的提交等。

由于服务器端的构件实现对象必须挂接在 POA 上，为有效管理多种类型的分布构件实现，我们所采用的设计思想是，容器对每种类型的构件独立使用不同的 POA，使得每一类构件实现对象都能够单独注册在为该类型构件所专用的 POA 上。这样设计的优点是可以为不同类型的构件选用不同的 POA 策略，并可以在不影响其它构件运行的前提下对其进行更新、迁移等操作，除此之外，还可以对不同构件分别配置不同的线程池，以减少容器中不同构件之间不必要的同步开销，提高应用服务器的并发度。

3 容器并发的设计

分布式构件服务程序往往需要接收大量的客户访问，同时处理多个用户的请求或一个用户的多个请求。即使某个构件服务程序本身并不需要多线程，但由于容器是整个构件系统的“大门”，它管理着 ORB 发送给部署在该容器中的所有构件的所有请求，所以，客观上要求容器必须支持高并发访问。

由于容器使用 ServantLocator 机制来截获构件请求，容器的并发就直接体现在 ORB 对 ExecutorInvoker 中 preinvoke 和 postinvoke 方法的并发调用上，因此，选择适当的 POA 线程策略与 ORB 的并发模型是容器并发设计必须解决的首要问题。我们将从四个方面详细阐述容器并发设计的内容。

3.1 POA 线程策略的选择

CORBA 规范2.3及其后继版本中在 POA 上定义了线程策略，其作用是指明 POA 向其注册的 Servant 提供什么样的并发同步语义。策略值 ORB_CTRL_MODEL 意味着可以利用 ORB 底层的多线程机制让 POA 上对象的请求处理在多个线程中并发执行。而 SINGLE_THREAD_MODEL 线程策略值则意味着 POA 将串行化其上的所有请求，以使那些并不是为多线程环境设计的代码可以安全地执行。由此可见，要使容器实现高并发访问，就必须将容器中构件 POA 的线程策略设置为 ORB_CTRL_MODEL。这是因为，容器中可能存在大量由不同客户创建的构件实例，不管构件的实现是否线程安全，对不同构件实例的请求应可以并发执行，而具有 SINGLE_THREAD_MODEL 策略值的 POA 将串行处理这些请求，显然，这降低了容器的并发能力，影响了 CCM 应用服务器的性能。

3.2 ORB 并发模型的选择

确定了构件 POA 的线程策略之后，就需要正确选择层 ORB 的并发模型，以充分利用 ORB 的多线程机制来实现容器的高并发访问。当然，不同的 ORB 实现可能采用不同的并发模型，我们基于 StarBus3.0分布计算平台来阐述容器实现高并发访问所需的并发模型。

ORB 内核的并发分为通信级并发和请求级并发。通信级并发是指 ORB 如何并发处理多个通信端口的通信问题，支持 reactor 和 thread 两种并发模型。reactor 模型实际上是一种单线程模型，而 thread 模型则采用多线程技术，在每个端口上使用一个线程专门负责接收端口上的连接请求，并在每条连接上使用两个线程分别负责接收连接上的请求消息和发送应答消息。能够同时在不同通信端口并发地接收多个客户请求是容器实现高并发访问的基本要求，因此，我们在容器中将 ORB 通信级并发模型设置为 thread。

请求级并发是指 ORB 如何并发地调度执行从通信级解析出来的请求。请求级并发可以通过三种派发策略来描述：SAME_THREAD(在同一线程内派发)、THREAD_PER_REQUEST(在新创建的线程内派发)以及 THREAD_POOL(在线程池内派发)。我们分别将这三种派发策略应用于 StarCCM 中，以测试应用服务器在不同派发策略下的性能，其结果如图2所示。

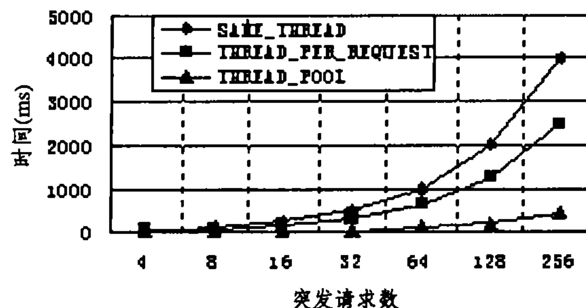

图2 不同派发策略下应用服务器的性能

维普资讯 http://www.cqvip.com

从图2可以看出，随着突发请求量的增加，应用服务器使用 THREAD_POOL 派发策略要明显优于其他两种派发策略，因此，我们将使用线程池技术来派发构件请求。事实上，线程池技术是当前很多流行的应用服务器为提高性能所普遍采用的技术。使用线程池派发构件请求时，应用服务器的并发能力要远远高于同一线程内派发，因而能在很大程度上提高服务器的吞吐率。尽管线程池派发的并发能力不及在新创建的线程内派发，但随着突发请求量的增加，它能够显著减少服务器中线程创建和销毁的次数，因而多数情况下，它比在新创建的线程内派发具有更好的性能。

3.3 高级线程池模型的设计

StarBus3.0采用简单的线程池技术，简单、高效、稳定可靠是这种线程池技术的显著特征，但不能满足 StarCCM 应用服务器的应用需求，主要体现在以下两个方面：

(1)没有对外提供管理接口，难以满足应用服务器的管理监控工具对服务器运行时的监控与管理需求。

(2)线程池一旦被创建，就不能修改其尺寸(即线程池中线程的数目)。应用服务器高效运行所需的最佳线程数将随着业务计算的复杂度以及突发请求数量的变化而变化，用户选择合适的线程池尺寸非常困难，因此，固定不变的线程池尺寸很难满足高性能应用服务器的需求。选择构件化的 HelloWorld 例子，分别使用不同的线程池尺寸对 StarCCM 应用服务器的性能进行测试，其结果如图3所示。

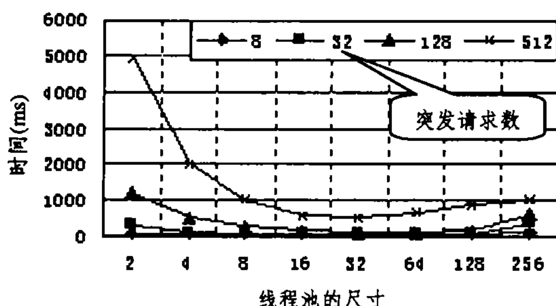

图3 线程池尺寸对应用服务器性能的影响

从图3可以看出合理配置线程池尺寸对于大量突发请求处理的效率有非常明显的提高，但是一旦线程池尺寸选择不合理(过大或过小)就会严重影响服务器的性能，甚至可能导致应用服务器无法处于稳定状态[8]。

因此，我们需要设计高级的线程池模型来定制 POA 的请求派发策略，用来派发构件的请求，以解决简单的线程池技术所存在的问题。为此，我们提出了一种可适配的线程池模型。

该模型由请求队列和线程池两个部分组成，对外提供以下几个可控参数以满足应用服务器的管理监控工具对应用服务器运行时性能监控的需求：

(1)队列标志长度 L_a(线程池将根据这个值来确定是否需要动态创建新的线程)

(2)队列最大长度 L_b

(3)线程池的最小尺寸 S_a

(4)线程池的最大尺寸 S_b

(5)线程池中线程的生存时间 T。

这些参数可以通过部署工具进行初始配置，也可以在运行时通过管理监控工具进行动态调整，从后面的算法描述中可以看出它们的含义及其作用。请求队列采用具有头尾指针的单向链表，使用两把不同的互斥锁分别同步对头尾指针的访问，当请求队列为空时，头尾指针指向同一个非 NIL 结点，可以证明，这种设计方案在请求队列含有未处理请求的情况下，请求的入队和出队完全可以并发执行。

当请求到达时，该线程池模型使用算法1来调度新到达的请求(L_c 表示请求队列中当前未处理的请求数；S_c 表示线程池当前拥有的线程数)。

算法1

```
while Lc≥Lb do
  等待线程池中线程通知 Lc 的变化
  endwhile
  将请求插入队列尾部
  Lc=Lc+1
  if Lc==1 then     通知线程池中等待请求的线程
  endif
  if (Lc≥La 且 Sc<Sb)或 Sc<Sa then
     创建一个新的线程
     将该线程的空闲等待时间设置为 T
     Sc=Sc+1
  endif
```

线程池中的每个线程使用算法2来处理每一次请求(T_w 表示线程空闲的时间)：

算法2

```
while Lc≥1 do
  if 得到请求 then
     Lc=Lc-1
     通知请求队列，Lc 已发生变化
     执行请求
     if Sc>Sb then
     //由于监控工具能够在运行时对线程池模型的可控参数进行
       动态调整，因此，运行过程中可能出现线程池中的当前数超
       过了线程池的最大尺寸 Sb。
         线程退出
         Sc=Sc-1
     endif
  endif
  //由于可能有多个线程竞争请求，因此，当前线程并不一定能够获
    得队列中请求的处理权
endwhile
Tw=0
等待请求队列通知，超时时间为 T
计算 Tw 的值//计算线程空闲的时间
if Tw<T then goto 1 //得到队列通知
if Sc>Sa then //线程等待超时
   线程退出 Sc=Sc-1
else goto 1
```

3.4 serialize 构件请求的串行化方法

CCM3.0规范规定了构件支持 multithread 和 serialize 两种线程模型[2,3]，它们决定了容器为构件分发请求的策略。multithread 线程模型说明构件是线程安全的，容器允许多个线程同时进入构件实现，对这种构件的访问，是无需容器辅助的。而 serialize 线程模型则意味着构件实现不是线程安全的，容器必须为其提供串行化机制，以阻止多个线程同时进入同一个构件实例。

容器可以选择在 ORB 或 POA 上同步来串行化 serialize 构件的请求。这种方案实现非常简单，只需在创建构件 POA 时，将同步策略设置为 ORB 或 POA 上同步即可。但是，POA 上同步会串行化同一构件所有实例的请求，ORB 上同步会串行化容器中不同 serialize 构件所有构件实例的请求，这降低了容器的并发性，严重影响了应用服务器的性能。

为此，我们提出了一种高效的支持容器高并发访问的串行化方法：

(1)容器在创建构件实例时，为其构造一把互斥锁，该互斥锁的生命周期与相应构件实例的生命周期相同。

(2)容器在 preinvoke 方法的调用中，为 ORB 返回调度请求的 servant(非 NULL 值)之前，锁定相应构件实例的互斥锁。由于 CORBA 规范保证在同一线程中调用处理同一请求的 preinvoke 和 postinvoke 方法，因此，可以将构件实例的

维普资讯 http://www.cqvip.com

互斥锁通过 preinvoke 方法中的 cookie 参数(该参数为字节流参数,其中可以放进任何信息)传递到 postinvoke 方法中。

(3)请求执行完成之后,容器在 postinvoke 方法中释放当前构件实例所对应的互斥锁。

这种方案对容器中不同构件以及同一构件的不同实例的请求完全并发执行,甚至对同一构件实例的请求可以并发通过容器,只是在使用 servant 真正执行请求时才对其进行串行化。测试表明,其实现是高效的。

4 容器并发的实现及性能测试

根据上述设计,我们基于 StarBus3.0分布计算平台,在 StarCCM 应用服务器中实现了容器的并发访问。定制的高级线程池模型虽然对请求队列的访问进行了优化,但同时也增加了对可控参数的额外控制,这是否会严重影响应用服务器的性能?为此,我们在 Windows XP 系统中,使用经典的构件化 HelloWorld 例子,从两个方面将其与 StarBus3.0中的线程池进行了对比测试。

一方面,在完全相同的条件下(采用相同的线程数,执行相同数目的突发请求,高级线程池中的线程数也不动态增长),测试结果如图4所示。

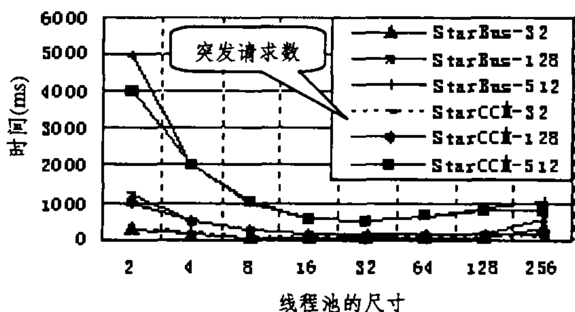

图4 相同条件下线程池性能的对比测试

从图4可以明显看出,在相同条件下,定制的高级线程池模型比 StarBus3.0中的简单线程池具有更好的性能。这说明我们对请求队列的访问优化是卓有成效的,它完全抵消了对线程池的额外控制所带来的性能开销。

另一方面,在缺省情况下(StarBus3.0缺省的线程池尺寸为10;定制的高级线程池缺省的参数设置为:$S_a=10$,$S_b=128$,$L_a=16$,$L_b=1024$,$T=60s$),测试结果如图5所示。

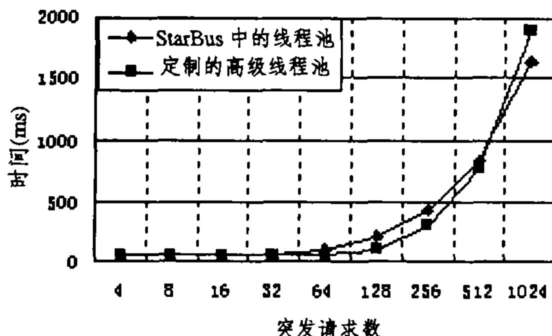

图5 缺省情况下线程池性能的对比测试

从图5可以看出,多数情况下,高级线程池处理突发请求的性能明显优于 StarBus3.0中的简单线程池,特别当突发请求数在128左右时,高级线程池处理请求的时间只有简单线程池的一半。只有当突发请求数量很大时,由于高级线程池增加了动态创建大量线程所需的开销以及过多的线程所导致的其他开销,其性能才有所降低。当然,可以通过应用服务器的管理监控工具对高级线程池模型的可控参数进行适当调整,比如,通过增加 L_a 或减少 S_b 来降低动态创建线程的数目,从而避免性能降低情形的出现。这充分说明,高级线程池模型中线程数能够随着突发请求数目的变化在$[S_a,S_b]$范围内动态调整这一特性,降低了构件部署人员选择理想的线程池尺寸的难度,减少了由于线程池尺寸选择的不当对服务器性能所产生的负面影响,增强了应用服务器适应不同突发请求数的能力,从而提高了应用服务器的性能。

结束语 StarCCM 应用服务器的研发得到了国家自然科学基金的和国家863高科技项目基金的支持。容器是 StarCCM 应用服务器的核心,是整个构件系统的"大门"。本文论述了一个支持高并发访问的容器的并发设计及其实现,这对于显著减少处理器的闲置时间、增加应用服务器的吞吐量、提高应用服务器的性能具有十分重要的意义,是 StarCCM 应用服务器真正实用化的一项极端重要的工作。

参考文献

1 Object Manage Group. Common Object Request Broker Architecture: Core Specification v 2.4.0—v 3.0.2. OMG, 2002.12

2 Object Management Group. CORBA Component Model Specification, 3.0 edition[S], OMG document: ptc/2002-0.-.5 3.0 edition, June 2002

3 Pharoah A, Siegel J, Brooke C. Creating Commercial Components CORBA™ Component Model (CCM) Technical White Paper. http://www.componentsource.com/BuildComponents/WhitePapers/CORBAWhitePaper.asp. March 2002

4 Object Management Group. Object Transaction Service Specification 1.3 edition[S]. OMG document: ptc/2002-08-07 1.3 edition, Aug. 2002

5 Object Management Group. Persistence Status Service Specification 2.0 edition[S]. OMG document: ptc/2002-09-0. 2.0 edition, Sept. 2002

6 Object Management Group. Security Servce Speicification 1.8 edition[S]. OMG document: ptc/2002-03-11 1.8 edition, March 2002

7 Henning M, Vinoski S 著. 徐金梧,等译. 基于 C++CORBA 高级编程[M]. 清华大学出版社,2000

8 段雪峰,张新家,戴冠中. 中间件服务性能建模与分析. 计算机应用,2004,24(1):105~107

9 StarCCM 应用服务器源代码. http://StarCCM.sourceforge.net