并行构件非功能属性研究

彭云峰 王瑞平
(安阳工学院计算机科学与信息工程学院 安阳 455000)

摘 要 扩展了CCA并行构件体系结构,首先定义了一个并行构件非功能属性的一个最小集合,然后实现了对这些属性进行管理的非功能构件。定义了与这些非功能属性相关的接口。并行构件可以有选择地提供这些接口。并行构件向非功能构件注册自己的属性,然后通过非功能接口向非功能构件提供相关信息。为了最优化并行构件的实现,构件开发者需要实现属性管理中属于特定构件的部分,这部分内容在非功能接口中实现。非功能构件调用并行构件的非功能接口,对并行构件的非功能属性进行统一的管理。这种方法提高了并行构件运行的性能,并且为管理并行构件的执行提供了一种简便的方法。

关键词 并行计算,并行构件,非功能属性,CCA,属性管理
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.08.046

Research of Parallel Component Non-functional Attributes

PENG Yun-feng WANG Rui-ping
(School of Computer Science and Information Engineering, Anyang Institute of Technology, Anyang 455000, China)

Abstract This paper proposed an extension of CCA component architecture. We defined a minimal set of non-functional attributes of parallel component. We implemented non-functional components for managing these attributes. This paper defined some interfaces related to these non-functional attributes. Parallel components can provide these interfaces optionally. Parallel components register their attributes to non-functional components. They provide their attribute information to non-functional components though non-functional interfaces. Component developers implement the management part which is specific to certain components. Non-functional components management the non-functional attributes of parallel components uniformly. Our way improves the performance of parallel component applications, and provides an easy way for the management of parallel component execution.

Keywords Parallel computing, Parallel component, Non-functional atrribute, CCA, Atrribute management

1 引言

在并行计算领域,提高并行计算软件的生产率正逐渐成为一个具有挑战性的课题。一方面,大规模并行应用的开发仍然处在手工阶段,使这些应用难以维护和重用;另一方面,新的硬件体系结构也不断出现,这就需要并行软件具有较高的生产率和较好的可移植性。在开发串行软件时,采用构件技术已经成为比较流行的趋势[1-3]。已有的普通构件技术如COM[4]、CORBA[5]和EJB[6]一般不面向并行软件。而基于这些技术的并行构件技术逐渐出现。并行构件是用来组成高性能科学计算应用的软件构件。美国一些国家实验室和学术机构共同建立了通用构件体系结构(CCA)论坛来研究并行构件问题[7]。在这个体系结构中,用不同语言编写的构件可以通过科学接口定义语言SIDL[8]进行交互。文献[9]介绍了一个为自适应并行构件设计的中间件平台,这个平台提供了一种对资源进行建模和监测的机制。Furmento Nathalie在文献[10]中为高性能应用提出了一种层次化的构件体系结构,它分离了构件的抽象和实现。文献[11]提出了一种对并行构件组合进行性能预测建模的方法。这些研究为并行构件的研究做出了重要贡献。但是对于一个并行构件,它可能具有除了科学计算功能以外的很多重要属性,这些属性在并行构件的执行环境是异构平台时显得特别重要。同时,并行构件具体实现的不同使得对它们属性的管理比较困难。

CCA并行构件体系结构作为一种比较完善、规范的并行构件体系结构,受到了广大并行构件技术研究者的重视。本文提出了一种对CCA并行构件体系结构的扩展。这个扩展的体系结构提供了对CCA功能构件的非功能属性的统一的管理。这个管理被分为两个部分,即非功能构件和非功能接口。其中一些非功能构件定义了这个管理中被所有功能构件共享的部分。非功能构件对于并行应用程序的功能构件来说

到稿日期:2013-10-23 返修日期:2014-01-03 本文受2012国家科技重大专项立项(2012ZX04011-012),国家自然科学基金河南人才培养联合基金项目(U1204613)资助。

彭云峰(1982-),男,博士,讲师,主要研究领域为并行计算与软件工程,E-mail:peng_ayit@163.com;王瑞平(1976-),女,硕士,讲师,主要研究领域为软件工程、计算机应用。

是公共的构件。功能构件是真正完成科学计算功能的构件，它们可以有选择地提供一些非功能接口。因为每个功能构件都有自己独特的地方，或者是数据分布，或者是算法实现等内容，所以只靠非功能构件无法对所有的功能构件实现统一的管理。而通过非功能接口中的方法实现非功能属性管理中属于特定功能构件的部分，可以有效地解决这个问题。本文提供了对功能构件的 9 个非功能属性的管理，它们是性能、资源需求、部署、调度、负载均衡、自适应、执行、事务、安全和访问，相应地定义了 9 个非功能构件和 9 个非功能接口。测试结果显示了提出的管理方法的有效性，它可以提高并行构件应用程序的性能，同时能对并行构件程序的执行进行高效的管理。

本文第 2 节介绍了扩展的并行构件体系结构；第 3 节介绍了非功能属性的管理机制；第 4 节介绍了非功能属性管理的例子并进行了测试；最后总结全文。

2 扩展的并行构件体系结构

首先定义了一个并行构件非功能属性的最小集合，包括 9 个非功能属性。这些属性对并行构件的性能和执行控制有着重要影响，它们是性能、资源需求、部署、调度、负载均衡、自适应、执行、事务、安全和访问。将并行构件分为功能并行构件和非功能并行构件。一个功能构件(functional component，FC)在一个并行构件程序中执行特定的科学计算功能。一个非功能构件(non-functional component，NC)定义为对功能构件的 9 个非功能属性进行管理的构件。一个非功能构件对于整个并行应用程序来说是一个公共构件，因为它管理的是这个程序中所有功能构件的某个非功能属性。对于 CCA 并行构件体系结构中的一个并行构件，它可以定义 provides 接口和 uses 接口。一个 provides 接口包含了提供它的构件所要实现的方法。uses 和 provides 接口对应，当一个构件需要使用其它构件的 provides 接口时，它本身必须包括一个与之对应的 uses 接口，这样可以降低构件之间的耦合，提高构件的复用性。两个构件可以通过对应的 provides 接口和 uses 接口连接起来。现有的并行构件技术仅仅关注于并行构件的功能型接口，即那些实现科学计算功能的方法。但是对于一个并行构件，还有许多非功能的属性能够影响它的运行，比如构件的性能、资源需求、部署、调度等等。为了对并行构件的这些非功能属性进行管理，定义了一些可选接口，每个接口都包含用来控制一个非功能属性的方法。为了优化构件的实现，这些方法都是特定于某个并行构件的，它们应该由并行构件的开发者来实现，属于实现科学计算功能的功能构件的 provides 接口。定义通过 uses 接口调用这些接口的非功能构件，但是非功能构件本身也有自己的 provides 接口，用户通过这些接口来启动非功能构件。由于非功能属性和底层运行环境有很大关系，定义了两种管理异构平台的资源管理构件，图 1 显示了所定义的扩展体系结构。

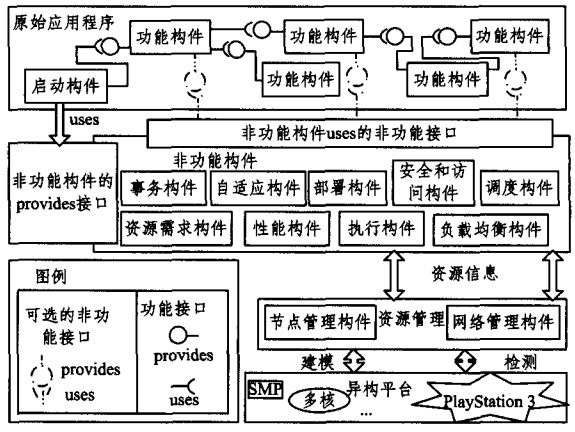

图 1 扩展的体系结构

3 非功能属性管理

在描述非功能属性的管理之前，先给出本文使用的底层运行资源管理机制。为了有效地对并行构件程序运行底层的异构平台上的资源进行管理，先将平台上的资源以 C++对象的形式进行建模。定义两个类，节点类描述的是一个节点上的资源，网络类描述的是网络资源。节点资源包括 CPU、内存、硬盘等等。网络资源包括带宽、网络地址等。定义两种资源管理构件，即节点管理构件和网络管理构件。每一个资源都有一个唯一的 ID，一个节点管理构件负责注册它所在节点上的所有资源。同样地，每一个服务器节点也都有一个网络管理构件，它可以通过简单的测试得到网络的资源信息。比如，一个 ping-pong test 可以得到一个简单消息的传递时间。

本文实现的非功能属性管理被分为非功能接口和非功能构件两部分。为并行构件定义一些非功能接口，这些接口是可选的，一个并行构件并不一定要实现这些接口中的一个。这些接口可能有不同的具体实现。它们属于功能构件的 provides 接口，即由功能构件实现，被相应的非功能构件调用，属于这些非功能构件的 uses 接口。功能构件向非功能构件注册这些接口，非功能构件使用这些非功能接口来管理功能构件的属性。下面分别介绍设计的 9 个非功能构件和它们相应的非功能接口。

3.1 性能构件

定义一个性能构件，它是一个非功能构件，通过一个性能接口连接到一个功能构件上。性能接口就是一个非功能接口，包含一个 sourcep 方法，这个方法能够给出一个功能构件的源代码地址。性能构件接着分析源代码，计算代码中不同的运算量和通信量。对于循环，它计算迭代的平均数量；对于一个 switch 或一个分支语句，它计算所有情况的平均值。通过与标准机器和网络状态下的基准程序的对比，它会建立起一个粗略的性能模型。这个模型包含所有提供了 sourcep 方法的功能构件。性能构件接着将这个模型存储到和它相连的一个模型数据库中。在性能接口中还有一个 dataget 方法，它可以获得功能构件在运行时的性能数据。在 dataget 方法中调用了 TAU[12] 构件。同时，性能构件可以调用第 3 节中定义的资源管理构件来获得底层资源的信息。结合从性能接口

得到的性能数据，性能构件可以为特定的功能构件建立起精确的性能模型，这个模型会代替数据库中的粗略模型。性能构件提供了一个预测接口，该接口包含一个 predictive 方法，这个方法可以为一些功能构件的组合提供性能预测。这个预测是基于模型数据库中的性能模型进行的，同时也用到了资源管理构件提供的资源信息。图 2 显示了性能构件的结构。

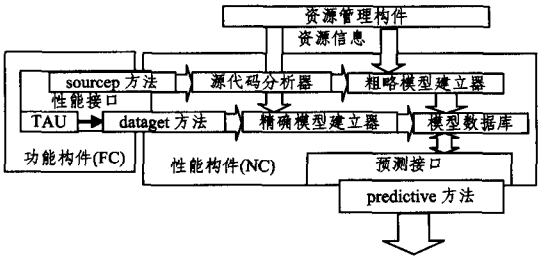

图 2　性能构件

3.2　资源需求构件和部署构件

资源需求构件也是一个非功能构件，它可以调用一个功能构件的资源需求接口，从而获得这个构件的基本资源需求。这个需求信息可以发送给一个部署构件。部署构件也是一个非功能构件，它可以产生对功能构件的部署策略。这个部署策略的产生包括两步：第一，部署构件要从资源管理构件获得底层资源的信息；然后，它会根据构件的资源需求信息和底层资源情况产生出每一种可能的部署策略。第二，为了选择较好的部署策略，本文采用了一种启发式的策略选择方法。对于每一种部署策略，部署构件都调用性能构件进行一次性能预测。这个预测是针对提供了性能接口这个非功能接口的构件的组合的。未提供这个接口的构件默认为是对整体性能影响不大的构件。基于这些预测，可以得出最佳的部署策略。这个最佳策略作为调用一个功能构件的部署接口时的参数，来部署构件程序。图 3 显示了资源需求构件和部署构件的结构。

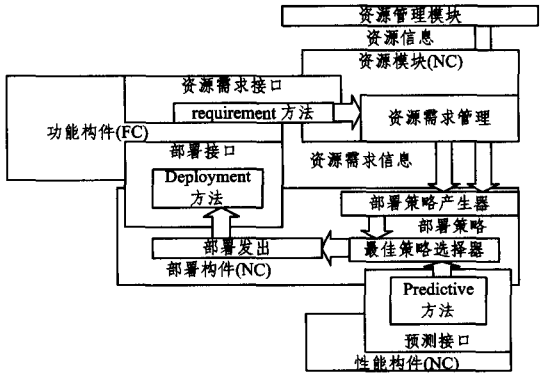

图 3　资源需求构件和部署构件

3.3　调度构件

定义了一个调度构件来控制功能构件上任务的调度。如果一个功能构件提供了调度接口，那么这个接口中的一个 criterion 方法会给出这个功能构件的调度标准。这个标准可以是时间、优先级、实时性或吞吐量。一个 taskgq 方法给出了这个功能构件中的一个任务组队列。这些任务组必须按照这个队列的顺序执行，但是一个任务组内部的任务可以以任意的顺序进行。调度构件使用任务组队列和调度标准生成一个调度策略。首先，它必须根据 criterion 方法给出的标准确定每一个任务组中的任务顺序。对于一个时间标准，要尽量减少总的执行时间。每一个任务都被赋予一个时间权重，权重最高的任务首先被调度。对于一个优先级标准，优先级最高的任务最先被执行。对于一个实时性标准，在规定时间内没有及时反应的任务将被抛弃。对于一个吞吐量标准，调度构件将尽量减少每一个任务组的执行时间，任务组中最短的任务将被执行。安排完任务组内的任务执行顺序后，任务组将被按任务组队列的顺序依次执行。调度构件通过调度功能构件的调度接口将这个调度策略发送给功能构件，调度接口中的一个 scheduler 方法执行这个调度策略。图 4 给出了调度构件的结构。

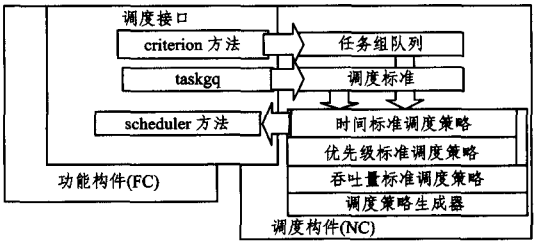

图 4　调度构件

3.4　负载均衡构件和自适应构件

如果想要为一个功能构件提供负载均衡机制，需要为这构件添加一个负载均衡接口，这是一个非功能接口。一个负载均衡构件负责调用这个接口，它是一个非功能构件。一个负载均衡构件提供了一个开始接口，这个接口中有一个 start 方法。用户可以设置 start 方法的参数，比如一个 dtime 参数设置了两次负载探测的间隔时间，high and low threshold 参数设置一个 CPU 的最大负载和最小负载的阈值。如果 CPU 的负载超过了最大阈值，就称这个 CPU 为过载 CPU；如果 CPU 负载低于最小阈值，就称这个 CPU 为空闲 CPU，sharen 参数设置了共享负载的 CPU 的最多个数。若要对一个功能构件提供负载均衡功能，首先要通过均衡接口将功能构件和负载均衡构件连接起来。负载均衡构件周期性地调用节点资源管理构件，根据得到的资源负载信息，它可以生成一个负载均衡策略。负载均衡构件将过载 CPU 上运行的负载迁移到空闲 CPU 上。这种策略有两种限制，首先空闲 CPU 的数量不能超过 sharen 参数的值，其次被选中的空闲 CPU 所在的节点资源必须满足被迁移资源的要求。负载均衡构件通过调用相关功能构件的负载均衡接口，将负载均衡策略作为接口参数传递给功能构件。负载均衡接口中的方法执行实际的负载调度。图 5 显示了一个负载均衡构件的结构。

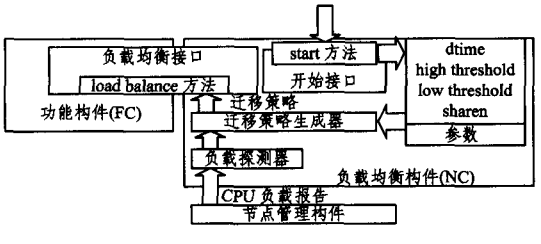

图 5　负载均衡构件

定义一个自适应构件。作为一个非功能构件，它具有与负载均衡构件类似的机制，不仅检测底层运行资源的变化，而且监控一些用户定义的事件。它也有一个开始接口，这个接口的 event 参数设定了被监控的事件。当被监控的事件发生时，自适应构件生成一个自适应策略并发送给相关的功能构件。定义 4 种自适应策略，分别是改变并行度、改变数据划分、构件迁移和改变实现。改变并行度策略表示改变功能构件的并行度。数据划分策略代表改变并行构件的数据划分。构件迁移策略表示将构件迁移到另一个服务器节点上。改变实现策略表示使用另一种功能相同但实现不同的构件替代当前构件。自适应接口是一个非功能接口，实现这个接口的功能构件会从这个接口的 self-adaptive 方法得到自适应的策略，然后执行。图 6 显示了自适应构件的结构。

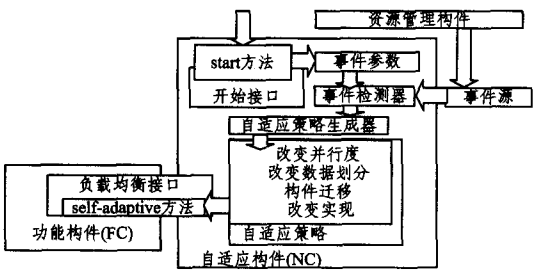

图 6　自适应构件

3.5　管理功能构件执行的构件

以上提出的非功能构件的主要作用都是用来提高并行构件程序执行的性能。而下面要介绍的执行构件、事务构件、安全和访问构件是用来对并行构件进行运行时的管理。执行构件作为一个非功能构件，可以调用一个功能构件的执行接口。这个接口属于一个非功能接口，执行构件调用这个接口中的 suspend 方法来暂停一个进程的执行并保存它的状态。这个方法向执行构件返回暂停操作的结果和被暂停进程的 ID。执行接口内的一个 continue 方法，它可以使用这个 ID 重新激活这个进程。这个接口内有一个 exit 方法，它可以停止功能构件的执行。图 7 显示了执行构件的结构。

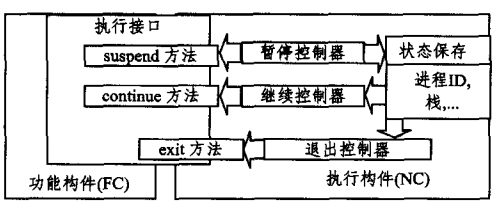

图 7　执行构件

提出的非功能构件还包括一个事务构件，它可以控制功能构件的一些事务操作。事务接口是一个非功能接口，它包含两种方法，事务构件可以调用一个 state 方法获得功能构件的当前状态；事务构件还能够调用一个 transactional 方法并得到返回结果。这个方法执行一些事务操作，比如更新被多个功能构件共享的数据。如果所有共享数据的构件的更新操作都成功了，事务构件向所有相关的功能构件发送数据更新成功的广播。这些功能构件将执行接下来的操作。如果一些更新操作失败，广播消息就是一个失败的通知，所有的功能构件将回滚到数据更新操作前的状态。图 8 显示了事务构件的结构。

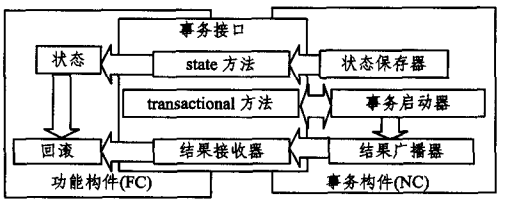

图 8　事务构件

安全和访问构件是一个非功能构件，它可以控制对一个功能构件的安全访问。一个功能构件可以提供一个安全和访问接口。这个接口中的 set 方法可以用来设置不同用户的访问权限。这些访问权限以一个列表的形式存储在一个文件中。访问权限可以是允许、阻止，或者是安全和访问接口中特定方法所定义的安全访问级别。安全和访问构件提供了一个应用接口，它包含一个 applyf 方法，这个方法有一个 clist 参数。用户如果想对多个功能构件实施访问控制，可以把这个参数设置成这些功能构件名字的列表。安全和访问构件调用安全和访问接口中的 access 方法来控制对功能构件的安全访问。图 9 显示了一个安全和访问构件的结构。

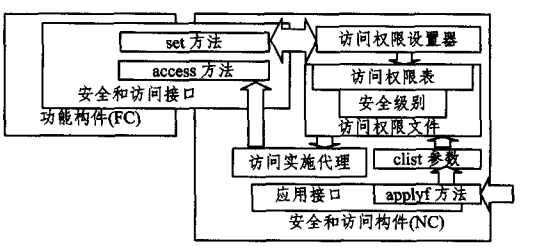

图 9　安全和访问构件

以上所提到的非功能接口中的具体方法的实现是特定于具体的功能构件的，但是非功能构件是被所有功能构件共享的，所以在本文提出的扩展的并行构件体系结构中，仅仅实现了所有的非功能构件。非功能接口中的方法仅进行了定义，具体的实现依赖于功能构件的内部。

4　实验结果和分析

所定义的功能构件的非功能属性中，对自适应、负载均衡、资源需求、部署、性能、调度这 6 个属性进行管理是为了提高并行构件的性能。对其它 3 个属性的管理是为了对并行构件的运行进行控制，例如，终止、暂停一个并行构件的运行。通过 5 个实验测试对这 9 个属性进行管理。

实验所用平台是一个异构的集群，它由 3 部分组成，这 3 部分通过千兆以太网连接。第 1 部分由 38 台 SMP 服务器组成。一台 SMP 服务器的处理器是 2 x Intel Xeon 2.8GHz CPU，512k 的二级 Cache，2G 的内存。操作系统是 fedora Linux 9。第 2 部分由 5 台多核服务器组成。其中两台多核服务器采用 4 x Quad-Core Intel(R) Xeon(R) Processor E7320/2.13GHz 处理器和 2X2MB 二级 Cache。两台多核服务器使用 4 x Quad-Core AMD Opteron(tm) Processor 8347/1.9GHz处理器和 4096k 的二级 Cache。一台服务器使用 8 x

Quad-Core AMD Opteron(tm) Processor 8347/1.9GHZ 处理器和 4096k 二级 Cache，8G 内存。操作系统是 Red Hat Enterprise Linux Server release 5.2。异构集群的第 3 部分由配备了 IBM CELL BE 处理器的 16 台 PlayStation 3(PS3)组成，操作系统是 Fedora 9 for Linux ppc64。

首先使用一个小的应用测试提出的负载均衡机制。程序包含一个主构件和 5 个从构件。主构件向从构件发送不等的任务，一些从构件得到比其它从构件远远多的任务。每个构件都部署在一台 SMP 服务器上。一个任务包含一些简单的计算。从构件上实现了负载均衡接口，并在测试时通过负载均衡构件对这个程序的运行进行管理。运行结果如图 10 所示，可以看出，提出的负载均衡构件提高了并行构件程序的性能。

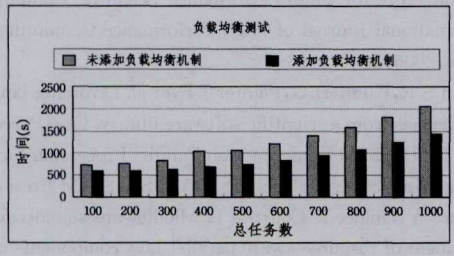

图 10　负载均衡测试

设计管理自适应属性的应用程序测试。在这个程序中，一个计算构件在一个本地数组上进行一定量的数学计算。数组的大小是 10000 的整数倍。构件在执行时被初始化为 4 个进程，每个进程放在一台 SMP 服务器上运行。计算构件进程首先检查本地数组的大小是否为 10000。如果不是 10000，则会触发一个事件，比如说数组的大小是 $10000\times n$，这个事件会被自适应构件捕捉到。自适应构件会产生一个自适应策略，这个策略规定计算进程会再生成 $(4n-4)$ 个新进程，然后前面定义的 adaptivef 方法会执行这个生成新进程的操作，给每一个进程发送 10000 个数据。在计算完毕后，结果会被重新收集到初始化时生成的 4 个进程中。图 11 给出了测试的结果，自适应策略提高了计算的并行度。本实验的自适应机制取得了比原始程序好得多的性能改进。

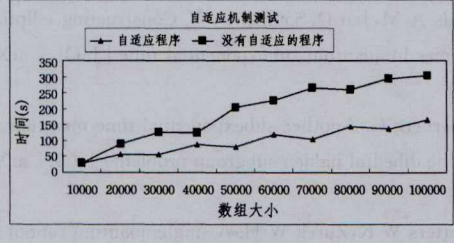

图 11　自适应机制测试

在测试调度构件时使用的是时间标准。测试时首先建立一个复合构件，构件内包含一些计算任务组。根据在单个 CPU 上执行的时间，每个任务被给予一个时间权重。调度构件调用复合构件上调度接口内的 taskgq 方法得到任务组的信息，然后根据时间标准产生一个调度策略。调度接口中的 scheduler 执行这个策略。这个测试被放在一台 SMP 服务器上进行，图 12 给出了这个测试的结果。

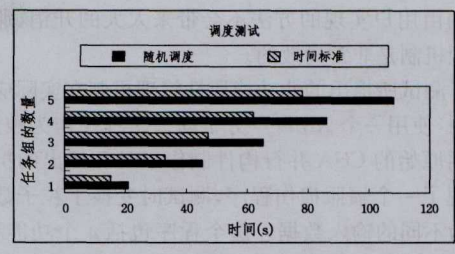

图 12　调度测试

在实验中同时测试所提出的性能构件、资源需求构件和部署构件。测试首先建立一个由 4 个构件组成的图像处理程序。一个初始化构件初始化测试环境，一个分发构件读入一些图片并把它们发送给一些处理构件的进程。这些进程对这些图片进行中值滤波，结果由一个收集构件接收。对分发构件和处理构件添加了性能接口、资源需求接口、部署接口这 3 个非功能接口。定义分发构件的内存需求是大于 4G，处理构件的资源需求是它需要一个处理频率大于 2.4GHz 的 CPU。使用满足这两个资源需求的服务器进行性能预测，并生成一个对这两个构件最优的部署。其它两个构件采取随机部署的方式。图 13 显示了实验的结果。

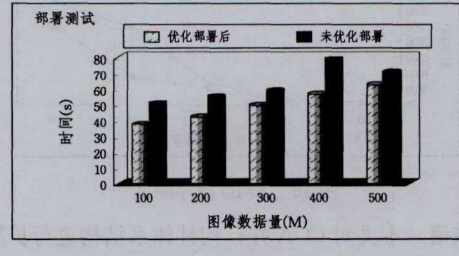

图 13　部署测试

在实验中同时测试执行构件、安全和访问构件、事务构件。实验实现了一个具有执行接口、安全和访问接口、事务接口的功能构件，这个构件运行时有 5 个进程。它首先检查当前用户的访问权限，每一个进程更新一个共享数据的本地拷贝，定义这些更新为一个事务。接着这个功能构件运行一些简单的计算，用来模拟实际应用中的特定科学计算功能。在这个计算过程中，在前面定义的一个非功能构件(本文称作执行构件的非功能构件)的控制下，这个计算可以暂停运行，等待 20 秒后继续执行。为了对照，还测试了没有实现这 3 个接口的功能构件。这个构件在计算完成后休眠 20 秒。测试结果如图 14 所示，所有的控制功能都能成功完成。

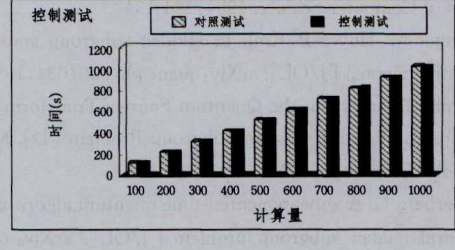

图 14　控制功能测试

在这个控制测试中，所有的事务操作都成功了。假设这是本文提出的事务机制最常遇到的情况。通过实验可以看出，由这些非功能接口带来的额外开销是非常小的。如果这

些接口内由用户实现的方法不会带来太大的开销，那么这种执行管理机制是非常高效的。

为了测试所提出的非功能属性管理机制在实际应用程序上的作用，使用一个 XMD[13] 分子动力学程序作为被测对象，分别使用原始的 CCA 并行构件制作工具和提出的扩展体系结构实现了一个辐照损伤程序，测试时变换了粒子数和迭代次数作为不同的输入数据。这个程序包括 4 个功能构件，一个驱动构件调用一个读入命令构件读入输入数据，发送给一个临近表构件和一个集成构件。临近表构件生成临近表并进行作用力的计算，集成构件分发粒子到不同的域。临近表构件有 1 个性能接口、1 个资源需求接口和 1 个部署接口共 3 个非功能接口。这个构件属于计算密集型构件，本测试定义它需要一个频率大于 2.5GHz 的 CPU 来执行计算。非功能构件之一的资源需求构件得到这个资源需求并发送给部署构件，部署构件根据性能预测的结果生成最佳部署策略并执行。使用一个用原始的 CCA 并行构件制作工具制作的具有同样分子动力学模拟功能的构件程序作为对照。测试结果如图 15 所示。从图中可以看出，提出的非功能属性管理机制在实际应用程序上起到了提高程序性能的作用。

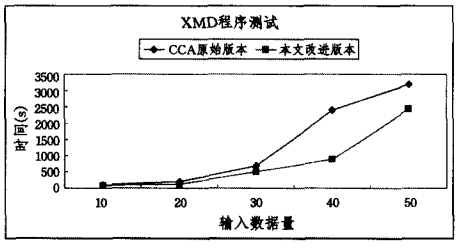

图 15 XMD 程序测试

结束语 本文对 CCA 并行构件体系结构进行扩展，提出了一种对于并行构件的 9 种非功能属性进行统一管理的机制，将这种管理机制中被功能构件所共享的部分实现为 9 个非功能构件；还定义了特定于某个功能构件的管理部分，这部分被实现为 9 个可选的非功能接口；实验部分给出了一些实现这些管理功能的具体的并行构件程序的例子。实验表明，提出的管理机制提高了并行构件的性能，同时对并行构件的执行管理也非常有效，且不会带来过多的开销。

参 考 文 献

[1] 唐龙业，王宇，王志坚. 面向服务的构件规约[J]. 计算机科学，2013，40(2)：180-185

[2] 张振领，贾仰理，谢圣献，等. 基于协议的实时构件行为一致性验证[J]. 计算机科学，2012，39(6)：125-128

[3] 周静静，盛鑫芽. CCA 环境下构件化线性解法器的设计[J]. 计算机科学，2011，38(5)：127-128

[4] Abernethy R，Morin R，Chahin J. COM/Dcom Unleashed[M]. IN，USA：Sams Indianapolis，1999

[5] Vinoski S. CORBA：integrating diverse applications within distributed heterogeneous environments [J]. Communications Magazine，IEEE，1997，35(2)：46-55

[6] Burke B，Monson-Haefel R. Enterprise JavaBeans 3.0[M]. CA，USA：O'Reilly Media，Inc，2006

[7] Bernholdt D E，Allan B A，Armstrong R，et al. A Component Architecture for High-Performance Scientific Computing [J]. International Journal of High Performance Computing Applications，2006，20(8)：163-202

[8] Kohn S R，Kumfert G，Painter J F，et al. Divorcing language dependencies from a scientific software library[C]// Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing，2001. Philadelphia，PA，USA：SIAM Press，2001：10

[9] Mahéo Y，Guidec F，Courtrai L. Middleware support for the deployment of resource-aware parallel Java components on heterogeneous distributed platforms[C]// Proceedings of the 30th EUROMICRO Conference，2004. Washington，DC，USA：IEEE Computer Society，2004：144-151

[10] Furmento N，Mayer A，McGough S，et al. A Component Framework for HPC Applications[C]// Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing，2001. London，UK：Springer-Verlag，2001：540-548

[11] Zhao Lei，Jarvis S A，Spooner Daniel P，et al. Predictive Performance Modelling of Parallel Component Compositions[J]. Cluster Computing，2007，10(2)：155-166

[12] Malony A，Shende S，Trebon N，et al. Performance Technology for Parallel and Distributed Component Software[J]. Concurrency and Computation：Practice and Experience，2005，17(3)：117-141

[13] XMD-Molecular Dymamics for Metals and Ceramics. Sourceforge[EB/OL]. http://xmd.sourceforge.net/，2013-03-01

(上接第 185 页)

量子多项式时间算法的探索具有重要的意义。

参 考 文 献

[1] Shor P. Algorithms for Quantum Computation：Discrete Log and Factoring [C]// Proceedings of the 35th Symposium on Foundations of Computer Science. 1994：124-134

[2] Ettinger M，Hoyer P，Knill E. Hidden subgroup states are almost orthogonal [J/OL]. arXiv：quant-ph/9901034，1999

[3] Murphy J. Analysing the Quantum Fourier Transform for finite groups through the Hidden Subgroup Problem [D]. Montreal：McGill University，2001

[4] Kuperberg G. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem [J/OL]. arXiv：quant-ph/0302112

[5] Regev O. Quantum computation and lattice problems [C]// Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science. 2002：520-529

[6] Regev O. A subexponential time algorithm for the dihedral hidden subgroup problem with polynomial space [J/OL]. arXiv：quant-ph/0406151

[7] Childs A M，Jao D，Soukharev V. Constructing elliptic curve isogenies in quantum subexponential time [J/OL]. arXiv：1012：4019

[8] Kuperberg G. Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem [J/OL]. arXiv：1112.3333

[9] Wootters W K，Zurek W H. A single quantum cannot be cloned [J]. Nature，1982，299：802-803

[10] Barnum H，Caves C M，Fuchs C A，et al. Non-commuting mixed states cannot be broadcast[J]. Phys. Rev. Lett，1996，76：2818-2821

[11] Duan L M，Guo G C. Probabilistic cloning and identification of linearly independent quantum states [J]. Phys. Rev. Lett，1998，80(22)：4999-5002

[12] Pati A K. Quantum superposition of multiple clones and the novel cloning machine [J]. Phys. Rev. Lett，1999(83)：2849-2852