HPC并行程序开发环境容错设计

● 郑 磊 江南计算技术研究所 无锡 214083
● 周明忠 江南计算技术研究所 无锡 214083

摘 要

本文基于语言应用和高可用运行控制环境，面向并行应用开发和HPC系统使用，对于为最终用户构建高可用的HPC开发使用环境进行了初步尝试和探讨。

关键词 高性能计算机 并行程序开发和使用环境 容错

1. 引 言

21世纪初，在应用需求的驱动下，高性能计算机（HPC）再度成为人们关注的焦点。长期以来，HPC系统传统上的两大难题一直没有得到根本解决：首先，HPC系统缺乏一个高效、实用的并行应用软件开发环境，高端计算应用开发周期长、效率差；其次，HPC系统缺乏一个友善的使用环境，对使用者的计算机专业技能要求较高，提高了其使用门槛。目前，高端计算应用已经成为HPC系统发展的最薄弱的环节。

高性能计算界一直以“峰值运算速度”来衡量HPC技术水平的发展，为了追求更高的峰值运算速度，HPC系统规模越做越大，其系统可用性也遇到了前所未有的挑战。此外，HPC用户环境软件系统自身也随之变得更加庞大和复杂，其开发、移植和维护的代价居高不下。如何充分发挥HPC系统的性能，尽可能地隐藏系统体系结构的细节、特点和内部错误，并有效控制并行开发环境软件系统自身的开发和维护成本，为用户提供一个友善的、高可用的HPC并行程序开发环境，一直是高性能计算机领域的一个重要研究课题。

2. 并行程序开发环境

2.1 功 能

HPC并行程序开发环境是针对HPC系统特点、基于网络的集成式用户并行程序开发和HPC系统使用环境，它为并行程序的设计开发及调试运行、性能分析与优化的各个阶段提供服务和帮助，为用户使用HPC系统提供一个友善的交互机制。用户通过HPC并行开发环境能够方便地了解并行计算机，有效地进行并行程序的设计、开发、编译、运行、调试，以及数据的可视化、程序的性能监测和性能分析等。

2.2 整体架构

通常情况下，HPC系统主要由三部分组成（如图1所示）：

- 主机系统：用于并行任务的计算；
- 非开放式服务节点（群）：主要用于编译并行程序，进行系统管理和作业管理；
- 用户客户端的个人微机：为用户操作的主要平台。

HPC并行程序开发和使用环境基于非开放式服务节点（或服务节点集群）和用户客户端个人微机实现。最终用户在安装Windows/Linux/UNIX操作系统的个人微机上通过网络使用该环境。

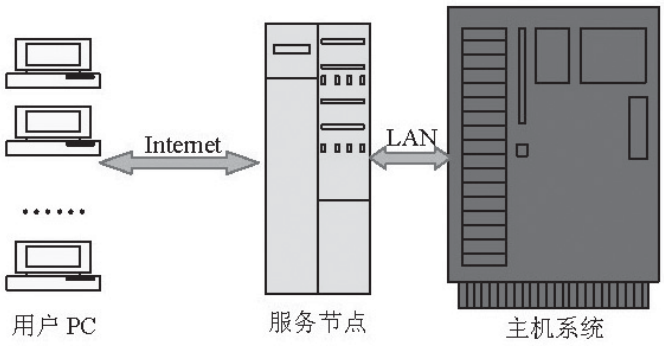

图1 HPC系统组成

用户开发并行应用和使用HPC系统所涉及到的各种功能要求高度集成，用户无需离开HPC并行程序开发和使用环境即可完成全部的开发与使用过程。开发环境要求与当前主流开发工具在风格上保持一致，自身具有高度的友善性和可维护性，整体风格清晰、明快。用户可以通过网络从任意一台PC机使用该开发环境。

并行程序开发和使用环境由客户组件与服务组件两部分构成，服务组件一般驻留在服务节点上，通过基于Web Service技术为客户端组件提供服务；客户组件驻留在用户PC上（如图2所示）。

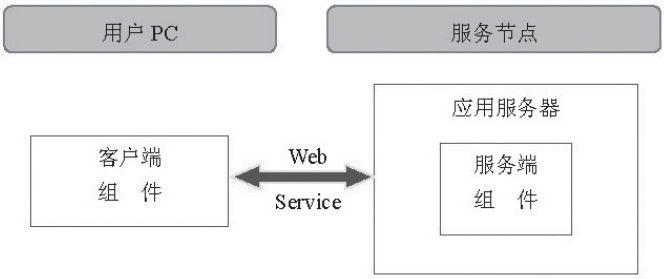

图2 开发使用环境结构

2.3 故障分析

在并行程序开发环境的实际运行过程中，可能会遇到以下四种故障情形：

- 客户端故障：开发环境客户端组件故障、用户PC死机等；
- 网络故障：用户PC与服务节点之间的网络故障；
- 服务节点故障：应用服务器崩溃、服务节点硬件失效/操作系统崩溃等；
- HPC主机系统故障：主机系统硬件失效/操作系统崩溃、主机系统与服务节点连接中断等。

上述四种故障情形中，最后一种故障的容错处理由主机系统负责，主机系统恢复运行后，环境必须能够与之自动重新联结并提供服务，以不影响用户使用；而前三种故障是并行程序开发和使用环境必须考虑并予以解决的问题。因此，并行程序开发和使用环境的设计必须要考虑到：

- 用户PC和开发环境服务端组件可能崩溃；
- 用户PC和服务节点间的不可靠网络连接；
- 服务节点或应用服务器可能崩溃；
- 操作系统或语言对环境服务支持会失效。

3、容错处理模式

在HPC系统容错体系中，并行程序开发和使用环境的容错位于语言应用和高可用运行控制的基础之上，直接面向用户的应用开发，建立容错的应用开发使用环境。并行开发环境基于下述原则进行容错设计：

- 尽可能捕获导致故障的原因；
- 根据获取的信息快速诊断，进行必要的纠正，并从最近的运行现场还原应用；
- 若无法自动纠正还原，则将可能的故障原因通报用户和管理员，等待人工处理。

针对不同服务组件的功能特点和可用性要求，并行开发环境组件设计主要基于三种应用模式：无保留状态的请求响应模式、异步提交—事后查询模式和保留状态的会话模式[1-2]。

3.1 无保留状态的请求响应

在该模式下，服务端不维持调用者的状态，只是处理单一请求，一次服务响应完成后客户状态即被清除。该模式的优点是具有很好的伸缩性，可以支持大量用户的调用，特别是没有容错方面的问题，尤其适用于简单查询类操作（如作业队列信息查询、目录/文件信息查询等功能）。若服务端组件返回错误信息，客户端组件只需重新请求服务即可。其缺点是，在该模式下若传递客户状态的参数过多，网络性能可能会有所降低。

在实际系统中，通过采用无状态的Session Bean组件实现这一模式。由于一个无状态的Session Bean从来不钝化，因而其生命周期只有两种状态：不存在（non-existent）状态和方法就绪（method-ready）状态。客户方通过接口参数调用该组件，根据返回值判别调用是否成功。若调用失败，则由调用者根据错误原因决定是重新调用还是向上层返回错误原因。

3.2 异步提交/事后查询

由于用户PC和服务节点之间的网络连接不可靠，用户PC、服务节点和应用服务器均有出现崩溃的可能。要确保实现高可用性，就需要具备有效的状态恢复机制。如果一个组件失败，系统在处理新

的请求之前，必须先恢复该组件所保持的状态。根据这一特点，并行程序开发和使用环境中的一些执行命令类的服务适宜于采用异步提交/事后查询的模式。

在实际系统中，通过采用无状态的Session Bean组件启动命令执行器、命令执行器捕获输出信息直接修改数据库信息和采用Entity Bean组件完成命令返回数据查询/操作来实现这一模式。由于相关信息在数据库中持久存储，即使当用户PC/服务节点/应用服务器崩溃或停止运行、用户PC与服务节点网络中断，其数据仍然保留在数据库中，不会丢失。

适合采用该模式的功能模块主要包括：程序编译、作业提交和作业输出信息查询等命令执行类功能模块。在这些功能模块中，将服务请求与结果查询分开，且操作结构在服务端持久存储，能够有效地克服网络故障和服务节点故障。其缺点是，流程略显复杂，执行效率有所下降。

3.3 保留状态的会话

HPC开发环境中的某些功能模块（如并行调试）必须以交互方式运行，是一个复杂的多请求、多事务处理过程。服务端组件不但要时刻维持用户的相关状态，还要维持与HPC主机系统内部的不间断网络数据连接。在这种情形下，必须使用保留状态的会话模式。

在实际系统中，通过采用有状态的Session Bean组件实现这一模式。在该模式下，如果出现Bean异常中止、网络连接终端或服务节点重新启动，客户状态能否维持直接关系着服务系统的可用性。对此，可采用两种实现级别：

- 完全依赖高端EJB容器的Session级别保留恢复功能，某些高端EJB容器可以通过不断保存当前活动组件的状态，为有状态的Session Bean提供故障恢复功能；
- 主动保留/恢复关键信息和资源，利用数据库系统完成对某些关键信息的保留。

适合采用前一级别模式的功能模块主要包括用户登录、注销等，若有故障发生，即使从安全角度考虑，也没有必要对其进行刻意恢复，客户端需要重新进行用户身份验证；适合采用后一级别模式的功能模块包括并行调试等交互式功能模块，若有故障发生，客户端重新连接后，服务端可以根据持久存储的信息资源从上个保留点为用户恢复服务。

4、结束语

本文对如何为用户提供一个高可用的应用环境进行了初步探讨，所做工作对未来HPC并行程序开发和使用环境的研发具有一定的启发和借鉴作用。从长远看，如何更加紧密地与主机容错系统进行融合，更好地对最终用户隐藏系统可能产生的错误将是今后研究的重点。

参考文献

[1] 柴晓路. Web服务构架与开放互操作技术. 北京：清华大学出版社，2002

[2] The Middleware Company. J2EE Best Practices. http:// www.middleware.com. 2003-01