第33卷 第18期 计 算 机 工 程 2007年9月
Vol.33 No.18 Computer Engineering September 2007
·开发研究与设计技术· 文章编号：1000—3428(2007)18—0279—04 文献标识码：A 中图分类号：TP391

并行文件系统管理工具的设计与实现

伍卫国[1]，陈长虹[1]，张 虎[1]，钱德沛[1]，胡雷钧[2]
(1. 西安交通大学计算机系，西安 710049；2. 浪潮北京电子信息股份有限公司，北京 100085)

摘 要：通过对常用并行文件系统体系结构的分析，提出了一种适用于某一大类并行文件系统的管理框架。以该管理框架为基础，实现了PVFS 并行文件系统的管理工具，并为用户提供了单一映像的管理控制台。功能测试结果表明，该管理工具的引入降低了并行文件系统管理的复杂性。

关键词：并行文件系统；文件系统管理工具；并行虚拟文件系统(PVFS)

Design and Implementation of Management Tools for Parallel File System

WU Wei-guo[1], CHEN Chang-hong[1], ZHANG Hu[1], QIAN De-pei[1], HU Lei-jun[2]
(1. Dept. of Computer, Xi'an Jiaotong University, Xi'an 710049;
2. Langchao(Beijing) Electronic Information Industry Co., Ltd., Beijing 100085)

【Abstract】 The structure of common file systems is analyzed and a management framework applicable to a group of parallel file systems is suggested. Based on the management framework, management tools for PVFS file systems are implemented and a single image management platform is provided. Functional evaluation shows that the file system management tool can effectively reduce complexity in file system management.

【Key words】 parallel file system; file system management tool; parallel virtual file system(PVFS)

1 概述

并行文件系统通常考虑3类问题：

(1)透明性。并行文件系统所管理的存储资源分布在服务器系统内的若干节点上，但用户在使用这些资源时，希望其如同单机文件系统一样方便；

(2)可靠性。分布式系统环境通常要求具有高可靠性，而数据的高可靠性是整个系统可靠性的基础；

(3)动态性。由于分布式系统的节点数量可能随时随地变化，因此并行文件系统管理的文件对象也是随时变化的。

并行文件系统较单机文件系统复杂，其管理通常采用脚本语言配置的方式，但该方式易用性较差。因此，设计一套使用方便、高效率的管理工具十分必要。

2 典型的并行文件系统结构

并行文件系统分为两类：商业并行文件系统和研究型并行文件系统。商业并行文件系统有xFS[1]、HFS[2](high performance file system)、GPFS[3](general parallel file system)和GFS[4](global file system)等。这类系统可以提供高性能I/O，但往往依赖于特定的平台以及特定的设备。研究型并行文件系统有PPFS[5](portable parallel file system)、Galley[6]和PVFS[7](parallel virtual file system)等，它们往往侧重某一方面，例如PPFS侧重于自适应缓冲和预取机制的研究；Galley则偏向于优化磁盘存取和文件的组织机制。

2.1 典型文件系统

(1)xFS 无集中式服务器系统，该系统中的节点在提供文件服务方面的地位均等，即任何一个节点都能够存取、缓存和控制任意的数据块。xFS 的节点由客户端、存储服务器、垃圾清扫器和管理器组成。该文件系统具有位置独立、性能良好和可扩展的特性。

(2)GPFS 是并行共享磁盘文件系统，用在 RS/6000 SP 和 Linux 集群中。GPFS 允许共享访问跨多个 SP 节点的多个磁盘驱动器的文件。该文件系统节点使用特有的硬件设备，通过交换机与共享磁盘阵列连接。

(3)GFS 是一种基于 Linux 操作系统的集群文件系统，允许位于一个 SAN(storage area network)中的多个服务器对 SAN 设备中的文件进行读写。GFS 集群节点通过 Fibre Channel 或共享的 SCSI 设备共享存储器。

(4)Galley 是一种运行于 IBMRS/6000 组成的集群和 IBM SP-2 上的并行文件系统。其设计目标是为应用程序提供更强大的接口，让用户程序显式地控制文件访问的并行方式。该系统采用客户/服务器结构，其节点分为两类：(1)I/O 处理机(IOP)，作为服务器运行；(2)计算节点机(CP)。其中，每个 IOP 管理一个磁盘，并以三维的结构来存储文件；CP 上的应用程序通过库函数与 IOP 通信以取得需要的数据。

(5)Lustre 是一种面向对象储存系统，由客户端、存储服务器和元数据服务器(meta data server, MDS)组成。对象存储设备(object storage target, OST)具有 CPU、内存和存储介质，提供以对象为单位的访问。MDS 管理文件系统的命名空间、

基金项目：国家“863”计划基金资助项目(2004AA111110，2002AA104550)

作者简介：伍卫国(1963－)，男，副教授，主研方向：高性能计算，嵌入式系统；陈长虹，硕士研究生；张 虎，博士研究生；钱德沛，教授、博士生导师；胡雷钧，高级工程师

收稿日期：2006-09-20 **E-mail**：wgwu@mail.xjtu.edu.cn

目录层次结构和数据的分布。OST 是智能型的，数据分布的工作由其完成，从而降低了 MDS 的负载。

(6)Google FS[8]是一个可扩展的系统，支持对大量数据的大型的、分布式的访问。一个Google FS集群由一个Master和大量的Chunk Server构成，Master维护文件系统的元数据，Chunk Server维护数据的存储信息。

2.2 并行文件系统的组成

并行文件系统都具有多个用于并行数据读写的数据 I/O 服务器节点，不同点体现在元数据服务器的数量和硬件依赖性等方面。按元数据服务器的数量可以将并行文件系统分为 3 类：(1)不存在元数据服务器类，例如 xFS 系统采用对称结构；(2)一个元数据服务器类，如 Lustre、PVFS 和 Google FS 等，这种结构最常见，但元数据服务器容易成为系统的故障点和瓶颈；(3)多个元数据服务器类，如各种增加了元数据镜像功能的 PVFS 文件系统。按照硬件依赖性可以分为两类：1)依赖特定硬件类，如 GPFS、GFS 等；2)不存在硬件依赖性类，如 PVFS、Lustre 等，这类系统一般是源代码开放的，可运行在 Linux/Unix 中。

xFS 等使用对称结构的并行文件系统由于结构差异较大，很难提出一种统一的模型来管理；大部分商业文件系统都依赖相应的硬件，比如磁盘阵列等，对其进行监控存在困难，因此本文对管理系统适用的目标系统作如下限定：存在一个或者多个元数据服务器；不存在硬件依赖性。研究对象为 Lustre、PVFS 和 LCTS-PFS 等，其结构特点如图 1 所示。

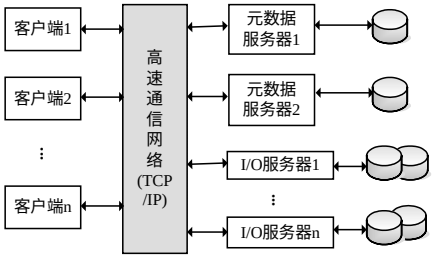


图 1 一类并行文件系统模型

(1)元数据服务器。元数据服务器管理系统的元数据。元数据主要用来描述文件的相关属性，如文件类型、存储位置、文件大小、创建与修改时间以及权限等，此外，管理的扩展文件属性还包括目录结构、服务器数目、服务器地址以及文件系统大小等。其主要涉及的管理操作有服务实体管理、服务器状态检测、故障恢复和数据同步等。

(2)I/O 服务器。I/O 服务器存储文件的数据信息。I/O 服务器的主要任务是响应由客户端发出来的读/写请求。在服务器中除保存文件的数据信息外，还保存文件的大小、修改与访问时间等信息，这些信息通过一定方式与元数据服务器保持同步。其主要涉及的管理操作有状态检测、监控、故障通知和数据同步等。

(3)客户端。客户端是文件系统的使用者，系统一般提供两种使用方式：I/O 库访问及使用标准的 POSIX 访问语义。其涉及的管理操作主要包括访问流量实时统计、配置更新和故障检测等。

3 并行文件系统管理框架

并行文件系统管理框架设计的核心是如何对监控信息进行采集处理，包括节点的监控信息采集；监控信息汇集后的分析处理、显示。为了采集节点的监控信息，要在每个节点上部署和配置管理代理，管理员通过控制台来监控和管理所有节点的监控信息。

3.1 管理框架设计

管理框架是指监控数据的通信框架模型，具体是指为了采集文件系统节点的监控信息而参与通信和协作的实体的集合以及这些实体之间的相互关系。本文采用的通信框架模型是基于管理/代理模式的两层通信模型，该模型包含两种通信实体：(1)管理代理，部署在被管理节点中，负责采集节点的监控信息；(2)管理端，即管理系统中为管理员提供交互界面的控制台(console)，管理端通过访问管理代理采集被管理节点和被管理实体的监控信息。该模型包含如下定义：

(1)被管理节点：部署了并行文件系统的计算机节点。

(2)被管理实体：在被管理节点上运行的组成并行文件系统的运行实体，实体一般表现为一个守护进程，比如数据 I/O 守护进程，元数据服务器进程等。

(3)管理代理：管理代理是在被管理节点上运行的守护进程，它的主要功能是收集并行文件系统各个被管理节点中被管理实体的状态信息，并对控制台发出的请求做出响应，发回执行结果，是管理系统的核心部分。

管理框架如图 2 所示，在管理过程中，管理员通过控制台向管理代理发出管理命令，管理代理接受到管理命令后，验证管理权限，执行相应命令，最后将执行结果发回到管理端。

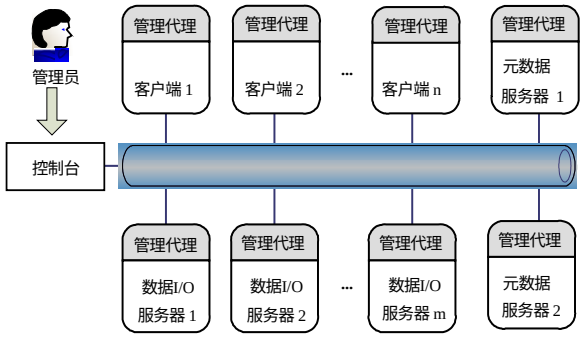


图 2 并行文件系统管理框架

由图 2 可知，并行文件系统管理工具的设计分为管理代理设计和管理控制台设计两部分。

3.2 管理代理设计

为了提高管理代理的通用性，使其能在多种并行文件系统中运行，本文采用管理插件和 XML 配置文件相结合的设计方式。其中，管理插件采用 Linux 的动态共享库编写，提供标准的调用接口；在 XML 配置文件中则定义了被管理实体及其状态获取和设置接口。采用这种方式，当代理应用到新的并行文件系统中时，管理控制台的修改量较小。管理代理的结构与主要模块如图 3 所示。

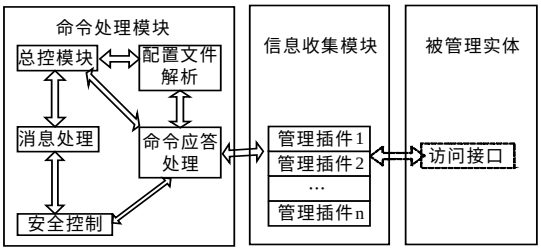


图 3 管理代理模块结构

由图 3 可知，管理代理由命令处理模块和信息收集模块组成。命令处理模块负责配置文件解析、安全控制和命令应答处理等。信息收集模块供命令处理子模块调用，当收到状态查询或者设置命令时，调用管理插件中的接口函数。管理代理的命令处理流程如下：

```
BindSocket(AGENT_PORT);                    //监听管理代理端口
while (1){packet = ReadPacket();           //接受一个管理数据包
  if (!packet.IsValid()) continue;         //验证包是否正确
  if (!packet.CheckSecurity()) continue;   //验证包的安全性
  BeginThread(CommandDispatchThreadFunc, packet); }   //启动
//命令处理线程
   void CommandDispatchThreadFunc(packet)
  //命令处理线程主函数
{ CMD_TYPE cmdType = GetType(packet);      //分析包的类型
  if (cmdType == null) return;
  //类型错误返回
  dispatchPlugin = LoadPlugin(packet, cmdType);     //加载相应
//处理插件
   if (dispatchPlugin.DoCommand(ack) == SUCCESS)        // 执行
//处理动作
        ack->SetCode(OK);                  //处理成功设置为 OK
  else  ack->SetCode(ERR);                 //处理不成功设置为 ERR
  SendBack(ack);     }                     //发回处理结果
```

3.3 管理控制台设计

管理控制台用来处理用户的交互，将用户输入的管理项目和参数编码成命令字发送给管理代理，并将操作结果返回给用户。大多数并行文件系统仅仅提供了命令行式的管理控制台，这种管理方式需要用户对并行文件系统本身有较深的理解并掌握操作系统相关命令才能正确使用，而且使用时出错的几率比较高。对此，本系统设计了两种管理控制台：(1)基于 console 方式的管理控制台，用户需要通过 ssh 或者 telnet 登录主机，然后启动字符方式的界面，对系统参数进行配置，最后交给本管理主机的管理代理。这种方式的主要优点是配置速度快，具有较高的安全性，适合中高级用户使用；(2)目前流行的 B/S 方式，即用户通过浏览器访问 Web 服务器，通过浏览器上的交互界面对系统进行配置。这种方式的易用性好，对用户的要求低，适合初中级用户使用。缺点是配置较复杂，编程难度大。

4 PVFS 管理系统的实现

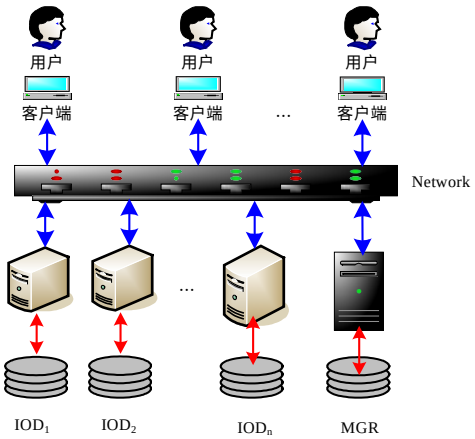


图 4 PVFS 组织结构

PVFS 组织结构如图 4 所示，其节点分为 3 类，即元数据服务节点(MGR)，数据 I/O 节点(IOD)及客户端。管理工具必须具有以下功能：(1)MGR 配置功能。该功能包括工作目录设置、元数据信息存放目录的设置、目录的拥有者和权限等；(2)IOD 配置功能；(3)客户端配置功能。该功能包括 PVFS 文件目录与 Linux 文件目录映射文件；(4)系统进程的状态查询、服务启停功能；(5)文件系统的整体管理功能，即监视文件系统资源的使用，剩余空间大小等。

根据需求，管理工具模块结构如图 5 所示，主要分为如下 3 部分：

(1)系统配置：管理节点配置和文件系统整体配置；

(2)资源监视：监控所有节点运行状况以及文件系统的整体资源状态；

(3)管理控制：控制所有节点上实体运行状态，完成文件系统的启动/停止功能。

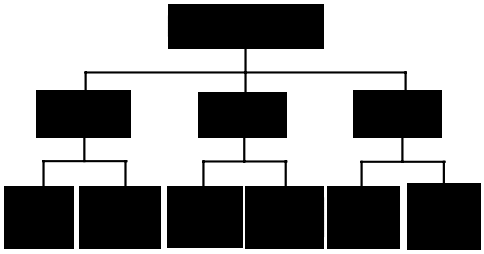
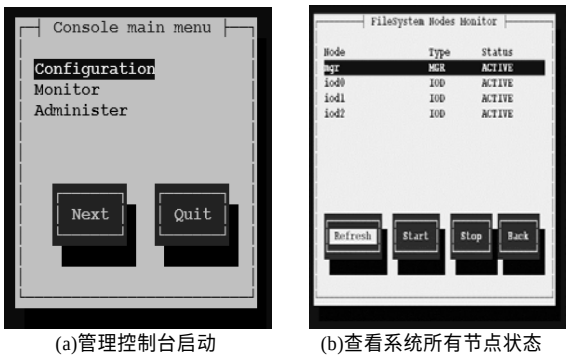


图 5 PVFS 管理工具模块

针对不同需要，本文构建了两种方式的控制台界面，即 Console 界面和 Web 界面。Console 控制台采用 Newt 库实现，通过 socket 和管理代理通信，本地需要配置所有被管理节点信息。各种文件系统的配置功能差别较大，对不同的系统，配置功能部分需要重新设计；Web 管理界面则采用 JSP/Servlet 开发，利用 Java 语言可移植性好的特点，方便系统在各种软硬件环境下运行。

4.1 Console 界面

图 6 是 PVFS 管理工具 Console 界面的代表性截图。图 6(a)是执行 startconsole 后显示的主界面，包含了 Configuration、Monitor 和 Administer 3 项功能，分别对应配置、监视和控制 3 个管理项目；图 6(b)是查看系统节点上所有实体时的界面，显示了系统节点的运行状态信息。



(a)管理控制台启动　　(b)查看系统所有节点状态

图 6 PVFS 管理工具 Console 界面

4.2 Web 界面

图 7 是 Web 控制台的代表性截图。图 7(a)是 MGR 管理控制台中的元数据服务器配置界面，主要用来配置 mgr 运行的各种参数，比如用户组、元数据目录等；图 7(b)是文件系统的总体管理控制台，主要功能包括显示系统各个部分的状态信息、启动/停止文件系统服务等。



(a)MGR 管理控制台-MGR 配置



(b)文件系统总体管理控制台

图 7　PVFS 管理工具 Web 界面

5　结束语

本文分析了典型的并行文件系统结构，对其一般结构特征进行了归纳；提出了一种可以适应多种结构的并行文件系统管理框架，并以 PVFS 为例，设计实现了其图形化管理工具。测试结果表明，Console 界面和 Web 界面都可以正确完成系统需要的管理监控功能，降低了该文件系统的管理难度。

参考文献

1 No J, Thakur R, Choudhary A. High-performance Scientific Data Management System[J]. Journal of Parallel and Distributed Computing, 2003, 63(4): 434-447.

2 Bordawekar R, Landherr S, Capps D, et al. Experimental Evaluation of the Hewlett-Packard Exemplar File System[J]. ACM SIGMETRICS Performance Evaluation Review, 1997, 25(3): 21-28.

3 Schmuck F, Haskin R. GPFS: A Shared-disk File System for Large Computing Clusters[C]//Proceedings of the Conference on File and Storage Technologies, Monterey, CA. 2002: 231-244.

4 Preslan K, Teigland D, O'Keefe M, et al. An Overview of The Global File System[DB/OL]. (2002-05). http://www.ch-open.ch/html/events/LinuxConference/PDF/1H_GFS.pdf.

5 Anderson T E, Dahlin M D, Neefe J M, et al. Serverless Network File Systems[J]. ACM Transactions on Computer Systems, 1996, 14(1): 41-79.

6 Nieuwejaar N, Kotz D. The Galley Parallel File System[J]. Parallel Computing, 1997, 23(4): 447-476.

7 Haddad I F. PVFS: A Parallel Virtual File System for Linux Clusters [EB/OL]. (2002-12). http://www.linuxjournal.com/article/4354.

8 Ghemawat S, Gobioff H, Leung S T. The Google File System[C]// Proceedings of the 19th ACM Symposium on Operating Systems Principles, NY, USA. 2003.

(上接第 252 页)

(2)VV&A 人员检查关键测试的执行；(3) VV&A 人员只接收重要测试的数据。其次 VV&A 人员评估集成测试计划的完备性和恰当性。开发者必须测试作为系统可接受性测试一部分的集成结构。这种测试以成员两两集成开始。测试序列不断重复扩展，直到典型的通信途径和想定元素被测试。最后 VV&A 人员校核硬件安装、操作、网络接口和集成输入数据。

阶段 6：确认决策阶段。这一阶段是围绕“运行联邦、准备输出”阶段展开的。这一阶段的活动主要是 VV&A 人员检查主要的产品以及每一步的证据收集。当制定确认决策时，确认人员将询问 VV&A 人员的意见。VV&A 人员将所有相关的和反常的信息输入给确认人员。如果 VV&A 人员的建议和确认人员的意见一样，那么将把全部证据进行总结，并呈交给确认权威考虑。否则，需要进行额外的调查。当数据缺乏时，如果出现不期望的结果，那么 VV&A 人员需要进行更详细的测试。

阶段 7：事后执行并归档阶段。这一阶段是围绕“分析数据、评估结果”阶段展开的。这一阶段的活动主要是 VV&A 人员参与事后分析，完成和记录所有的 W&A 活动。VV&A 人员应准备两方面的文档。第 1 个方面的文档是与开发者相关的报告，包括对可接受性测试过程的详细的记录以及联邦执行每一场景的能力。VV&A 人员是这一报告重要的检查者和写作者。另一个方面的文档是 W&A 人员自己的最终报告，包括 VV&A 活动的总结和完成。VV&A 的文档应该包含充分的信息，以允许联邦 VV&A 过程的重新建立，支持可重用性，或至少部分重用。从这个意义上来说，VV&A 历史文件应该是联邦相关的历史文件中不可缺少的部分。

4　结束语

基于 MDA 的联邦开发过程看起来和原有的 HLA 仿真系统开发过程很类似，但是这里有一个至关重要的不同，就是联邦的领域模型和应用模型在联邦开发过程中是可执行的，并且其过程是可逆的，而在以往开发基于 HLA 仿真系统的方法当中，从模型到模型的转换主要是手工完成的。这样开发人员可以迅速地验证系统的基本功能，进一步优化 PIM 和 PSM，从而提高开发效率。通过 MDA 技术，把验证、确认和认证进一步地扩展到 HLA 仿真系统开发的全过程，为联邦开发的可靠性、可用性以及功能和描述的一致性方面提供保证，从而提高仿真系统描述真实世界的准确性和可信度。这也就是说，基于 MDA 架构，真正能够做到对联邦模型进行与联邦开发同步的 VV&A。

参考文献

1 Graffagnini J, Youngblood S, Lewis R. An Overview of the Verification, Validation, and Accreditation(VV&A) Process for the HLA FEDEP[C]//Proceedings of the Summer Computer Simulation Conference. 1999.

2 Dobey V T, Lewis R O. Verification, Validation, and Simulation: T&E, VV&A or Both[C]//Proceedins of the Summer Computer Simulation Conference. 1998.

3 Jean G, Simone Y, Lewis R. A VV & A Process of the HLA FEDEP[C]//Proceedings of the Summer Computer Simulation Conference. 1999.

4 徐耿保, 曾莲芝. 论仿真技术的重点和其中的热点与难点[J]. 计算机仿真, 2003, 20(2): 1-4.

5 徐学文, 陶国强, 马春燕, 等. 国外校核、验证和确认(VV&A)的研究和实施[C]//2001 年军事运筹学会议. 2001.

6 李　俊. HLA 仿真系统的校核验证与确认研究[D]. 长沙: 国防科学技术大学, 2003.

7 吴晓燕, 许素红, 刘兴堂. 仿真系统 VV&A 标准/规范研究的现状与军事需求分析[J]. 系统仿真学报, 2003, (9): 1081-1084.