计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(10):11−19 [doi: 10.15888/j.cnki.csa.006026]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 ALUA 的多路径存储系统①

范长军, 胡志成, 杨佳东

(中电海康集团有限公司, 杭州 310013)

摘　要: 随着网络存储技术的不断深入发展与广泛应用, 存储多路径技术成为信息系统容灾与故障恢复方案中的核心技术之一. 在现有存储多路径技术的基础上, 提出了一种基于 ALUA 的多路径存储子系统总体框架, 并重点在逻辑卷控制器属主管理、ALUA 属性与路径选择策略配置和 I/O 重定向等模块进行了创新性的设计与实现. 经过应用测试, 该系统很好地解决了存储系统的路径故障迁移和故障恢复问题, 并智能地实现了路径间 I/O 的负载均衡, 极大地提高了性能和可靠性.

关键词: 多路径; ALUA 属性; I/O 重定向; 负载均衡; 故障迁移; 故障恢复

引用格式: 范长军,胡志成,杨佳东.基于 ALUA 的多路径存储系统.计算机系统应用,2017,26(10):11–19. http://www.c-s-a.org.cn/1003-3254/6026.html

Multi-Path Storage System Based on ALUA

FAN Chang-Jun, HU Zhi-Cheng, YANG Jia-Dong

(CETHIK Group Co. Ltd., Hangzhou 310013, China)

Abstract: With the rapid development and wide application of SAN technology, multi-path technology becomes one of the key technologies to achieve disaster tolerance and high performance. Based on the existing techniques, an improved multi-path storage sub-system is presented. And then improvement is made in four modules, i.e., LUN ownership management, configuration of ALUA, path selection strategy, and I/O forwarding implementation. The test results show that this scheme can be a very good solution to the path failover and load balance problems, greatly enhancing the I/O performance.

Key words: multi-path; ALUA attributes; I/O forwarding; load balancing; failover; failback

随着移动互联网、社交网络等新型 IT 技术的发展, 可用数据总量呈几何级增长, 人类进入大数据时代. 大数据中蕴含的知识是宝贵的社会财富, 但其价值的体现依赖于对数据的可靠存储和高效处理, 这对存储系统的可用性、可靠性和存取效率提出了新的挑战, 为此, SAN (Storage area network) 网络存储技术应运而生, 并得到大力发展和广泛应用.

在 SAN 系统中, 有两个基本实体, 发起端和目标端. 发起端主动发起 I/O 命令请求存储资源, 目标端响应存取命令. 发起端可以是应用服务器、个人 PC 机等各种有存储需求的实体, 目标端主要是磁盘阵列、磁带库等存储实体, 两端由光纤通过 (网络、FC 等) 交换机连接起来形成 I/O 路径, 构成发起端与目标端通信的桥梁. 在 I/O 子系统中, 单路径 I/O 传输错误是导致系统单点失效的根源, 它难以满足现代存储对系统可用性和可靠性的要求; 同时, 在单路径场景中, 数据只能拥堵在一条 I/O 通道中传输, 成为性能的瓶颈. 为了提高性能和可靠性, 在从主机到交换机到存储控制器的链路中, 一般采用冗余的部件, 使对存储设备的 I/O 操作可以在多条路径上执行, 由此发展出 I/O 多路

① 收稿时间: 2017-01-16; 采用时间: 2017-02-26

径技术, 以期解决故障切换和负载均衡问题.

目前, 国内外都有多路径技术的研究工作, 主要集中在一些存储厂商, 如 EMC、NetApp、HP、IBM 等, 一些大学、研究机构也在参与[1]. 本文在现有存储多路径技术的基础上设计和实现了一种基于 ALUA 的多路径存储子系统, 它支持 I/O 的故障迁移和负载均衡等功能, 同时具备高性能、可靠性以及高扩展性.

1 研究现状

在现代网络存储环境中, 为了提高存储系统的可靠性, 通常会使用冗余的物理组件 (如适配器、电缆或交换机等) 连接服务器主机与存储设备, 所以在主机系统到存储阵列控制器间会存在多条物理链路 (SAN 存储网络多路径连接示意图如图 1 所示). 通过这些物理链路, 存储阵列上的块设备被映射给主机系统, 在默认情况下, 主机系统将每条路径映射的块设备都认作一个实际的物理盘, 而本质上它们只是通向同一个物理盘的不同路径. 主机端在对这一虚拟的块设备进行访问时, 可以在多个物理链路中选择一个下发指令, 即便出现其中一条路径断掉的单点故障, 还有其它物理路径可以保证主机业务的正常执行, 提升了存储系统整体的可靠性. 同时, 多路径软件也可通过自动监控和动态分流数据 I/O, 实现传输速率的提高.

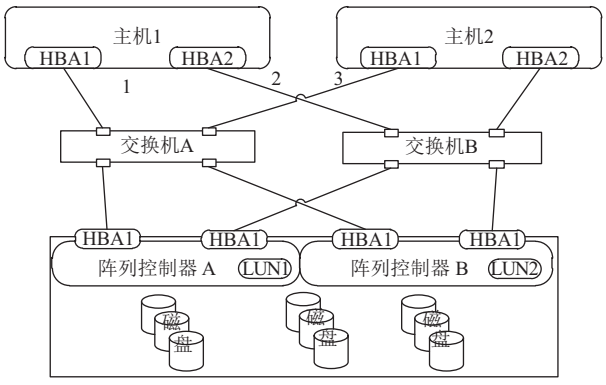

图 1 SAN 存储网络多路径连接示意图

鉴于多路径技术的诸多优点, 各大存储厂商如 EMC、华为等也将其纳入标准配置, 对该技术的应用和研究也一直比较热门. 目前多路径技术研究主要集中在两方面, 一方面是主机端多路径软件的设计与实现, 如文献[2,3]分别介绍了现有的各种主机端多路径实现方法, 文献[4-7]分别在不同的层面, 用不同的方式设计和实现了主机多路径; 另一方面是多路径软件的特定应用, 如文献[8]将不同模式的磁盘阵列与不同多路径软件搭配使用, 文献[9]将多路径技术与虚拟化、集群等软件配合使用, 文献[10-12]分别将多路径应用到电台广播和地震资料处理集群中.

这些多路径方案在各自的领域针对不同应用场景能够很好的满足需要, 但是都存在各自的问题. 首先, 多路径技术涉及诸多方面, 包括主机系统和存储系统等, 它们相互配合共同实现相应功能, 而现有方案往往只着眼于主机端的多路径软件, 没有从整体上考量. 其次, 一部分方案是将现有的多路径技术应用到某一个特定的领域, 并未对多路径技术进行升级和优化.

为应对上述问题, 本文设计并实现了一整套的多路径存储子系统, 它具备如下特点:

- 高性能, 高度负载均衡和高聚合带宽;
- 高扩展性, 支持链路、交换机和 HBA 卡等的扩展;
- 高可靠性, 链路冗余和路径选择策略避免了单点故障.

2 多路径存储子系统架构

2.1 多路径存储子系统设计思想

在多路径典型应用场景中, 存储阵列中的物理磁盘通过存储控制器上运行的 RAID 程序组成 RAID 组, 再将 RAID 组中的存储池划分为存储单元, 称为 LUN (Logical unit number, 又称逻辑卷), 存储设备将 LUN 提供给主机进行 I/O 访问. 存储控制器是多路径 I/O 的目标端, 对于多路径的故障切换和负载均衡起着至关重要的作用. 存储设备一般包含两台或更多的存储控制器, 以达到冗余配置, 每台存储控制器又有多个存储端口, 以方便连接前端主机或存储网络交换机, 从而形成多条链路. 当采用多条链路时, 多条传输路径可同时传输数据块, 以解决单条链路的传输能力有限的问题.

本文提出的方案对存储系统进行多路径组网, 实现下述一整套的流程. 主机启动时系统通过扫描, 发现每条路径上的块设备, 并记录路径状态等相关信息. 对于每个块设备 (LUN), 主机端多路径软件下发命令, 查询它的所属控制器, 并将主机到其控制器属主之间的路径作为优先选择路径, 即主路径. 如果主机到其所属控制器之间有多条路径, I/O 就会被多路径软件分摊到各条路径上, 以期提高存储系统的整体性能. 如图 1 所示, 当主路径 (如路径 1) 由于控制器 A 宕机而失效后,

存储端自动将业务通过 failover 切换到控制器 B, 随之主机端多路径软件将 I/O 切换到备用路径 (路径 3), 避免因单点故障而造成业务中断. 当控制器 A 的故障得以解除或修复后, 存储业务通过 failback 切换回控制器 A, 并由主机端多路径将 I/O 从备用路径切换回主路径.

总体而言, 多路径技术涉及从主机到存储阵列之间的各类实体. 逻辑卷所属控制器的设定会影响其在 ALUA 机制下目标端口组的配置, 此两者与主机端路径选择策略相配合, 共同实现多路径功能. 因此, 多路径存储子系统的设计需要从系统论角度来考虑, 由总到分、自顶向下地进行, 并充分考虑各个模块之间的相互促进与制约, 在保证故障转移功能的基础上, 尽最大努力实现负载均衡, 从根本上改善系统 I/O 性能.

2.2 多路径存储子系统架构组成

本文提出了一种新的基于 ALUA 的多路径存储子系统架构, 包括系统管理模块、ALUA 属性配置模块、路径选择策略管理模块和 I/O 重定向模块等. 系统管理模块负责整个存储阵列的存储资源管理, 其中与多路径相关的是 LUN 的控制器属主信息, 它是实现不同 I/O 路径优先级的关键因素; ALUA 属性配置模块就是据此来设置其对应的目标端口和目标端口组的, 它与路径选择策略管理模块一起为 I/O 规划下发路径, 以保证不同 I/O 路径间的负载均衡和单条路径 I/O 性能的最优化; 当进行了上述的设定之后, I/O 只能通过主存储控制器下盘, 分发到冗余存储控制器的 I/O 需要重定向模块将其转发给所属控制器. 具体的多路径存储子系统架构如图 2 所示.

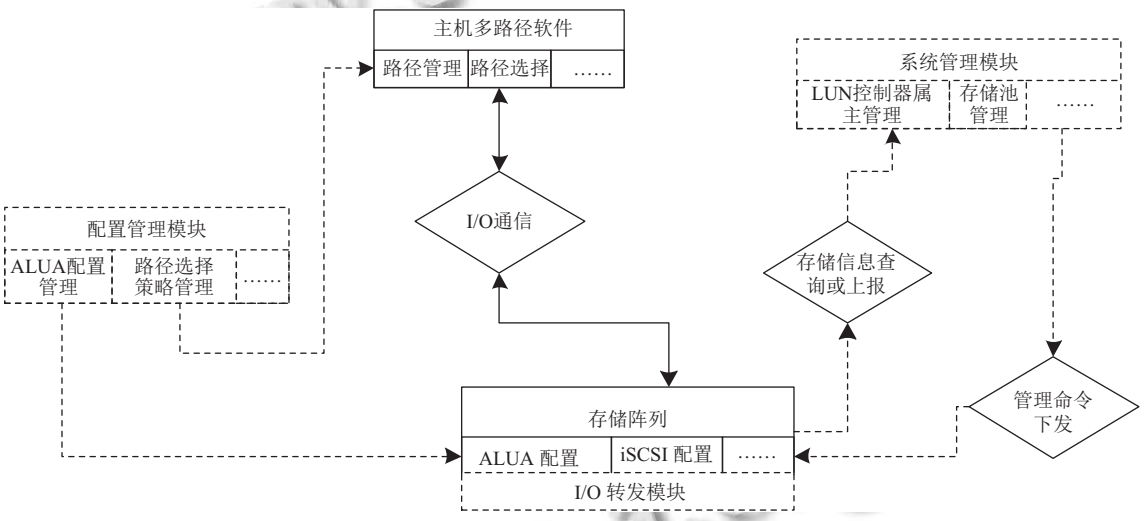

图 2 多路径存储子系统架构

具体而言, 在系统管理模块, 本文提出了一种逻辑卷控制器属主的自动分配和智能均衡方法, 通过收集各存储控制器上的 I/O 信息来动态分配 LUN 到各存储控制器上; 其次, 在底层存储中, 基于 ALUA 机制提出了一种 LUN 目标端口和目标端口组的配置方式, 根据 LUN 的控制器属主信息, 来规划路径的优先级; 再次, 在底层存储中, 开发了 I/O 重定向模块, 在 I/O 从非所属控制器下发时, 通过内部端口将 I/O 转发到所属控制器; 最后, 在路径选择策略管理模块中, 根据底层存储 ALUA 属性的配置, 设置了相应的路径管理策略, 并开发了新的路径选择算法. 通过对上述方案的设计与开发, 更好地实现了 I/O 路径的故障转移和负载均衡功能.

3 各模块的设计与实现

3.1 LUN 控制器属主管理

在多控制器存储阵列中, 主机在访问存储设备逻辑单元 (LUN) 时, 由于 LUN 存在控制器属主的概念, 下发给不同 LUN 的 I/O 将选择不同的路径, 这会造成各控制器 I/O 负载和性能的不同. 为了充分利用存储资源, 提高性能和可靠性, 实现故障切换和负载均衡, 需要一种有效的配置方案.

目前, 多控制器存储阵列的 LUN 属主和多路径配置方案多种多样, 各厂商都有自己的多路径软件和 LUN 属主配置子系统, 种类繁多、配置复杂, 并且存在以下问题: 传统的 LUN 控制器属主的配置都是通过系统管理界面人为手动操作的, 由于难于全面掌握各控制器的

状态信息, 管理员很难有效地进行所属控制器的划分. 这有可能导致 I/O 在一台控制器上拥堵或饥饿的极端情况, 极大地损耗了性能, 对可用性也造成了一定的影响.

为解决上述问题, 本文提出了一种多控制器存储设备逻辑卷属主管理方法. 在双控制器场景下, 创建 LUN 之前, 首先收集各控制器的 I/O 统计信息, 并根据设定的算法决定相应的控制器属主, 然后据此通过 ALUA 进行存储端口和端口组的配置, 为 LUN 提供路径访问的优先级机制. 最后, 当新创建的 LUN 投入使用后, 反馈并更新各控制器的 I/O 统计信息. 具体参看图 3.

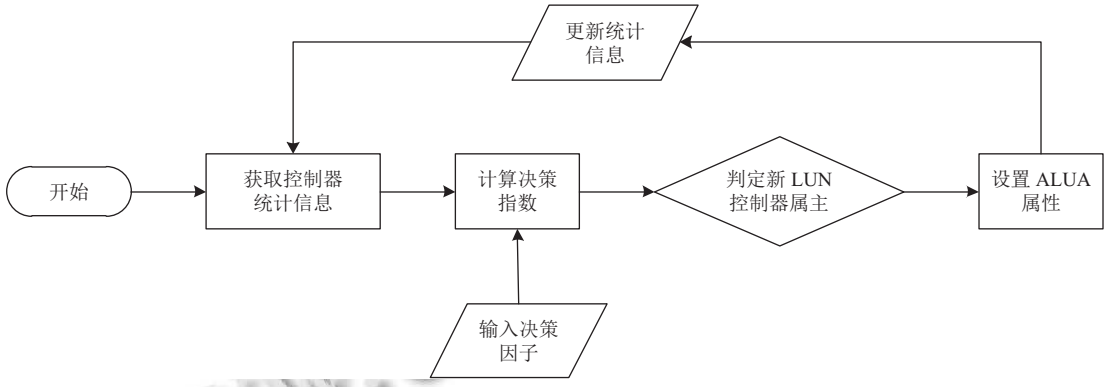

图 3 LUN 控制器属主管理流程

具体而言, 首先采集各个控制器的基本 I/O 信息, 包括带宽、IOPS 等; 然后通过一定的统计方法计算求取均值、方差和积分等性能表征参数; 接着, 将给定的决策因子与表征向量进行加权平均计算, 得到每个控制器的决策指数; 最后, 选取指数较大的控制器为新建卷的控制器属主.

根据存储的特性, 选取各个控制器的带宽、IOPS、LUN 数目、LUN 总容量、总 I/O error 率、故障率、I/O 突发率、总 I/O 量、I/O 集中度、吞吐量等来作为性能指标, 计算并表示其性能表征参数为 $X=\{x_1, \ldots, x_i, \ldots, x_n\}$. 为了消除数值量纲不同造成的影响, 将数据进行归一化处理. X_i 与 Y_i 分别表示双端控制器各自的第 i 个参数.

$$X_i = X_{i\_observe} / \left(X_{i\_observe} + Y_{i\_observe}\right) \quad (1)$$

$$Y_i = Y_{i\_observe} / \left(X_{i\_observe} + Y_{i\_observe}\right) \quad (2)$$

权重系数$\upsilon=\{\omega_1, \ldots, \omega_i, \ldots, \omega_n\}$可由管理员根据应用场景的不同而设置不同的值, 以决定主要的影响因素. 例如, 两个控制器的 LUN 的总带宽均值相同, 但是其方差有较大差别, 而且一边的 I/O 突发率明显高于对端, 如果新建的 LUN 主要应用于平滑的 I/O 顺序读写, 则应该将 I/O 突发率和方差对应项的权重系数提高.

$$\upsilon \cdot X = \Sigma \omega_i \cdot x_i = \omega_1 \cdot x_1 + \ldots + \omega_i \cdot x_i + \ldots + \omega_n \cdot x_n \quad (3)$$

进一步地, 当配置信息的数据积累到一定程度时, 可以用线性回归的方法来学习得到决策因子, 自适应地得到各项参数, 增加子系统的智能性和自动化程度.

两个控制器通过加权求和公式 (如公式 (1) 所示) 得到一个决策指数, 比较两个决策指数, 数值较大的控制器为新建逻辑卷的所属控制器.

分配好新建 LUN 的控制器属主后, 接下来在安装了 LIO(Linux-IO target) 模块的存储端控制器上进行端口组的划分和 ALUA 的配置.

3.2 ALUA 属性配置管理

ALUA(Asymmetric logical unit access) 的全称是非对称逻辑单元访问, 是 SPC-4 协议的一个重要组成部分, 主要应用于某个存储资源 (LUN) 具备多条访问路径的环境下. 在本文, I/O 性能在 LUN 所属控制器对应的路径上会更好, 不同路径具备不同的访问特性, 使用 ALUA 来规定路径访问的优先级, 确保在正常情况下读写指令都是通过最优路径进行下发, 以此达到最优的访问体验, 并通过目标端口组 TPG(Target port group) 来对这些路径进行分类管理.

一个目标端口组 TPG 包括若干目标端口, 同一个 TPG 中的目标端口在访问逻辑单元 LUN 的时候具有相同的属性. 主机可以通过 SCSI 命令来获取某一个 LUN 对应的 TPG 及其相关属性, 并通过这些信息来组织访问 LUN 的路径.

ALUA 有 4 种访问模式, 这些工作模式是在主机下发 inquiry 指令时, 存储端上报给主机的, 主要目的是告诉主机, 存储处于哪个工作模式上. 具体的描述如表 1 所示.

表 1 SCSI 命令中的 TPGS 域

编码	描述 (Report target port groups, 简称RTPG; Set target port groups, 简称STPG)
00b	SCSI目标设备不支持非对称逻辑单元存取, 或者取决于具体厂商, RTPG或STPG命令均不支持
01b	仅支持implicit非对称逻辑单元存取, 支持RTPG命令, 而不支持STPG命令, 并不需要STPG命令即可改变目标端口的非对称逻辑单元存储状态
10b	仅支持explicit非对称逻辑单元存取, 既支持RTPG命令, 也支持STPG命令, 需显示调用STPG命令才能改变目标端口的非对称逻辑单元存储状态
11b	既支持implicit也支持explicit非对称逻辑单元存取, RTPG或STPG命令均支持

ALUA 的访问状态有 4 种, 这些状态是通过响应主机下发的 SCSI 指令 REPORT TARGET PORT GROUPS 上报给主机的,对应的位是“ASYMMETRIC ACCESS STATE”.

SPC-4(SCSI Primary Command-4) 对存储端口划分为四种状态:

1) 主动优化 (Active/optimized): 目标端口能立即访问 LUN.

2) 主动非优化 (Active/un-optimized): 仅能响应相应的命令, 可转化为 Active/optimized 状态.

3) 备用 (Standby): 仅能响应相应的命令, 可转化为 Active/optimized 状态.

4) 不可用 (Unavailable): 仅能响应受限的命令集, 不可转化为其它状态.

设定端口为某一访问状态, 相应的存储控制器就仅能响应对应的命令集, 限定了目标端口访问 LUN 的权限. 对于双控制器存储阵列, 根据端口状态的划分, 可以有三种不同的多路径访问方式: A/A(Symmetric Active/Active, 对称双活), A/P (Active/Passive, 主备模式) 和 ALUA(Asymmetric Active/Active, 非对称双活). 为了性能和实现的简易型, 本文选择 ALUA.

为了有效地约束 I/O 下发路径, 防止 I/O 在一台控制器上拥堵或饥饿, 结合上述的 LUN 控制器属主的设计, 本文利用 LIO 提供的 ALUA 机制来配置目标端口和目标端口组.

为进行端口组属性的配置, 开发了相应的系统接口, 将目标端口按照所属控制器进行分组, 为 LUN 控制器属主的目标端口组提供更高的路径优先级, 从而提高 I/O 效率和可用性.

具体而言, 对于特定的 LUN, 将目标端口按照所属控制器进行分组, 划分到不同的目标端口组 TPG 中. 将所属控制器的所有目标端口设为 Active/optimized 状态, 形成一个目标端口组, 将备控制器的所有目标端口设为 Active/un-optimized 状态, 形成另一个目标端口组. 当有 I/O 读写请求时, 主备控制器都可以接受对 LUN 的访问, 但流经备控制器的 I/O, 会被重定向模块转发到主控制器, 最终 I/O 只能通过主控制器下盘.

此外, 由于所属控制器端路径和非所属控制器端路径的性能不同, 当一端控制器因故宕机, 其上的 LUN 故障迁移 (也即 failover) 到对端控制器时, I/O 切换到非优化的路径下发, 性能随之下降. 为了解决此问题, 将对应 TPG 的 type 属性设置为 both(Implicit and Explicit), 其中的 Explicit ALUA 属性赋予主机端更改备端控制器对应端口组存取状态的权限, 以实现多路径软件对存储端 TPG 属性的修改, 达到提升路径优先级的目的.

3.3 路径选择策略管理

3.3.1 路径组织策略

针对网络存储, 各大厂商都有自己的多路径实现方法, 为了提高响应速度和总体性能, 需要在主机端开发相应的多路径软件来组织 I/O 路径和进行 I/O 路径的选择. 但是由于主机系统的种类繁多, 针对不同类型的操作系统都需要开发一套多路径软件, 这极大地增大了维护成本, 降低了扩展性. 本文充分利用各大系统平台提供的原生多路径软件, 通过其提供的通用框架来设计路径组织和选择策略.

在各类主机多路径软件中, 路径组把具有相同特征的路径组织在一起, 以便于管理 I/O 路径. 在某一瞬时只有一个路径组处在活跃状态, 在路径切换时只能从当前活跃的路径组里选择最优的路径. 不同的主机系统平台内嵌的多路径软件的组织策略各不相同, 但都实现了常用的模式. 在应用服务器领域, Linux 类型系统的主机占有绝对优势, 下面以 Linux 主机为例.

Linux 系统主机自带的多路径框架 DM-Multipath 的组织策略分为五种:

1) multibus 策略, 同一个路径组包含一个块设备对应的所有路径.

2) failover 策略, 一个路径组只含一条路径.

3) group_by_serial 策略, 同一个路径组内只含序列号相同的路径.

4) group_by_prio 策略, 同一路径组内只含优先级相同的路径.

5) group_by_node 策略, 同一路径组内只含节点名相同的路径.

为了负载均衡的目的, 本文选择 group_by_serial 模式来配合存储端 ALUA 属性的配置, 这样 I/O 的切换被限定在同一控制器的路径组内, 只有在进行 failover 故障迁移时, 才在不同的控制器间切换, 实现了对 I/O 路径的管控.

3.3.2 路径选择

在主机下发读写指令时, 先根据优先级从多个路径组中选一个, 然后在这个路径组里根据设定的路径选择算法进行选择. DM-Multipath 内置的路径选择算法有三种, 分别是 Round-robin(轮询算法)、Queue-length (排队深度算法) 以及 Service-time(服务时间算法).

轮询算法通过双向循环链表来组织所有路径, 当选择新路径时, 会顺次选取下一个节点对应的路径返回. 触发路径切换的条件是, 每条路径固定分发一定次数的 I/O, 当前路径发完规定次数的 I/O 后, 就会自动切换到下一条路径. 但是, 该算法没有考虑到路径处理能力、响应时间等的不同.

队列深度算法参考每条路径上的 I/O 负载, 通过动态记录每条路径上排队等待处理的 I/O 数量来衡量路径的优劣. 具体而言, 当有 I/O 分发到某条路径上时, 对应的队列深度就增加; 当队列中的 I/O 下发到底层驱动上时, 队列深度就减少. 同上, 此算法也没有考虑不同路径物理特性的不同.

服务时间算法预估 I/O 请求发送到每条路径后进行处理所需要的周转时间, 服务时间最短的那条路径为最优路径. 具体而言, 将路径上总的排队 I/O 块大小除以此路径的吞吐量即可得到服务时间.

为了更好地在路径间均衡 I/O 负载, 本文提出并实现了一种新的路径选择算法——hikdata-selector 算法. 在选择一条路径时, 不仅仅单一地参考响应时间、相对吞吐量、排队 I/O 块大小和重复次数等, 而是将它们综合起来考虑.

在选取一条路径时, 先考虑路径的响应速度, 选择响应时间短的路径作为最优路径; 当两条路径的响应时间一样时, 再计算相对服务时间, 优选耗时少的路径; 如仍无法确定, 还可继续参考路径的相对吞吐量; 当相关指标都相同时, 可随机选一条路径返回. 具体流程可参看图 4 所示.

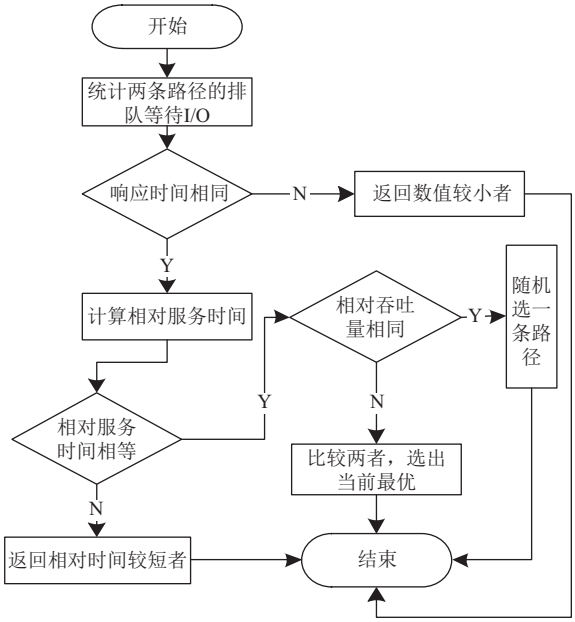

图 4 hikdata 路径比较算法

路径由双向循环链表组织起来, 选择的具体流程是: 首先, 从备选路径链表里选取前两条路径, 并根据上述算法进行比较, 得到当前最优路径; 随后再从路径链表里取出下一条路径, 与当前最优路径进行比较, 选出较好的一条来更新当前最优路径, 循环往复, 直到整条链表全部判断完毕, 选出的就是最佳路径. 总体流程如图 5 所示.

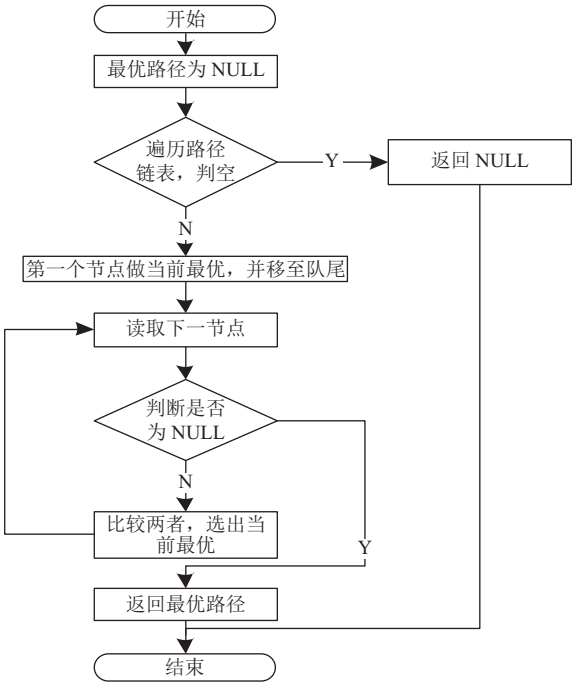

图 5 hikdata 路径选择流程

3.4 I/O 重定向模块

根据上文的设定, 多路径软件优选主控制器端路径, 备控制器端路径闲置. 当出现主存储控制器 (如: A 控) 的网卡损坏或者网络环境异常等故障时, 主机系统与主控之间的路径断开, 主机多路径软件会将 I/O 切换到备控 (如 B 控) 的路径下发指令, 由于主控没有因故障宕机, 所以不会执行 failover 操作, 备控中相应的块设备仍然是只读的, 从主机端下发到备控的读写指令将无法在备控下盘. 为了保证业务不中断, 需要将备控接收到的 I/O 通过内部端口重定向到主控, 才能保证整个 I/O 过程的完整. 为此, 基于 LIO 开发了 I/O 重定向模块, 其整体架构如图 6 所示.

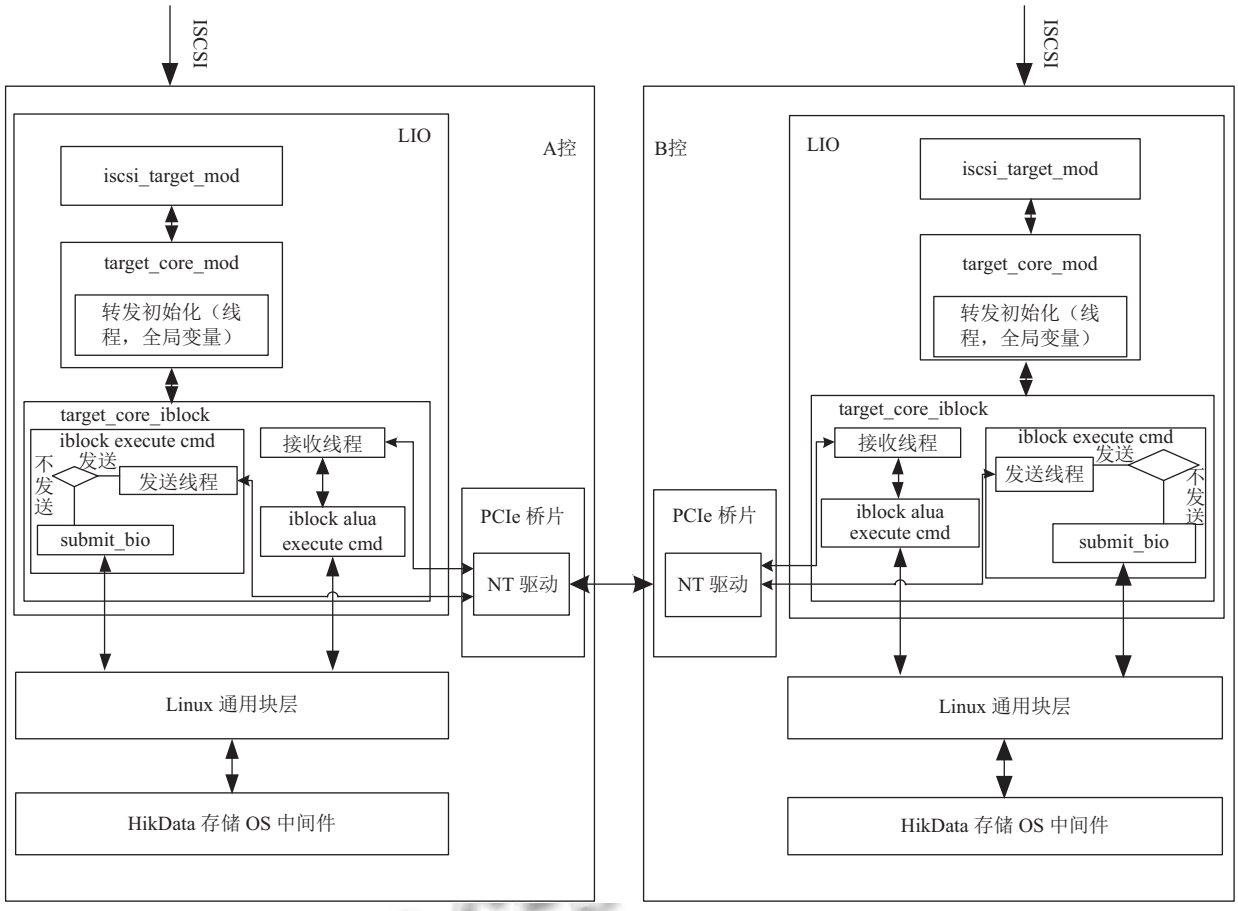

图 6 I/O 重定向模块系统架构

整个 I/O 重定向可以分为初始化模块和转发模块. 初始化模块主要负责初始化 I/O 重定向所需的全局变量, 分配全局空间, 并创建转发线程. 转发模块由发送子模块和接收子模块组成. 发送子模块负责接收本端 iSCSI 命令数据, 在需要转发的情况下, 将命令数据通过 PCIe 非透明桥 (NT) 驱动的发送接口发送到对端, 接收子模块负责接收对端发送来的命令数据, 并在本端执行 I/O 指令, 再将执行结果通过 NT 接口发送回对端, 对端发送子模块将结果返回给上层, 完成一次完整的 I/O 重定向过程.

在本文, 在 LIO 子系统中实现 I/O 重定向, 减少了对存储端 Linux 存储协议栈的改动, 降低了实现难度, 也降低了 I/O 重定向对于整个系统带来不稳定的风险. 现有的 I/O 重定向, 如 iSCSI 重定向只能适用于前端是 iSCSI 协议的场景, 扩展性较差. LIO 支持将块设备通过多种协议 (如 FC、InfiniBand 等) 映射到主机端, 通过本方法实现的 I/O 重定向, 可以有效兼容前端不同的网络存储协议, 提高适用性. 另外, 本 I/O 重定向模块通过调用 NT 驱动接口实现与对端控制器之间的数据收发, 速度更快, 更可靠.

4 实验评估

根据本文提出的多路径存储子系统模型, 采用自主研发的双控制器存储阵列平台搭建了多路径存储子系统运行环境. 针对多路径技术主要实现故障转移和负载均衡功能, 设计了两种组网方式来进行测试, 并选择存储业界广泛使用的 IOMeter 作为 I/O 测试工具. 测试硬件由两台应用服务器、一台双控制器存储阵列和两台万兆交换机组成, 如图 7 所示.

组网一仅包括 7 条路径 (P1-P7), 如图 7 所示, 主机 1 访问 LUN2 时, 路径有 P1-P3-P6 和 P1-P4-P5-P6 可选, 而且两路径的 I/O 在存储后端全由控制器 A 处理.

首先, 测试 I/O 路径切换. 经实测, 在上述场景下, I/O 指令优选 P1-P3-P6 路径下发. 通过拔掉网线的方法来模拟 P3 出现故障的场景, 此时路径重定向为 P1-P4-P5-P6, 但存储设备并没有执行 LUN failover. 将 P3 线路插回, 再模拟 P6 出现故障的场景, 此时存储业务转由控制器 B 处理, I/O 重定向到 P1-P3-P5-P7 路径, 但是没有立即执行 LUN failover. 过了一段时间, 通过控制器 B 处理的 I/O 累计到了设定的量级时, 才执行了 LUN failover. 上述实验充分说明了, 此多路径子系统能够有效地实现故障切换功能.

接下来, 启动目标端, 设置 LUN1、LUN3 为远程访问盘, 并将其分别通过 HBA1、HBA2 的两个万兆网口映射出去. 针对主机 1 与主机 2, 分别启动 iSCSI 发起端连接到目标端. 主机 1 登入 LUN1, 主机 2 登入 LUN3, 如此每个主机可得到两个相同的远程盘, 经测试, I/O 指令优先通过所属控制器对应的路径下发.

随后, 先通过发起端测试单链路的性能和系统负载, 再加载多路径模块, 对多路读写性能和系统负载进行测试. 相应的测试数据如图 8 所示. 其中多路单链路是指通过多路径模块访问时, 只使用一条有效链路传输. 双链路是指通过多路径模块访问时, 采用 hikdata-selector 的负载平衡算法, 两条链路并行传输. 单链路是指不通过多路径模块而进行单一链路的访问. 从图 8 中可以看出, 三者之中双链路的读写性能最好.

主机与存储控制器配置:
CPU: Intel Xeon CPU E5-2643 3.40GHz 12 核
网络: Intel Ethernet 10-Gigabit X540-AT2
内存: 32 GB
存储容量: 48 TB

10 Gb E 交换机配置:
接口: 48 个10G BASE-T以太网电接口
CPU: 1.5 GHz, 4 核
内存: 4 GB

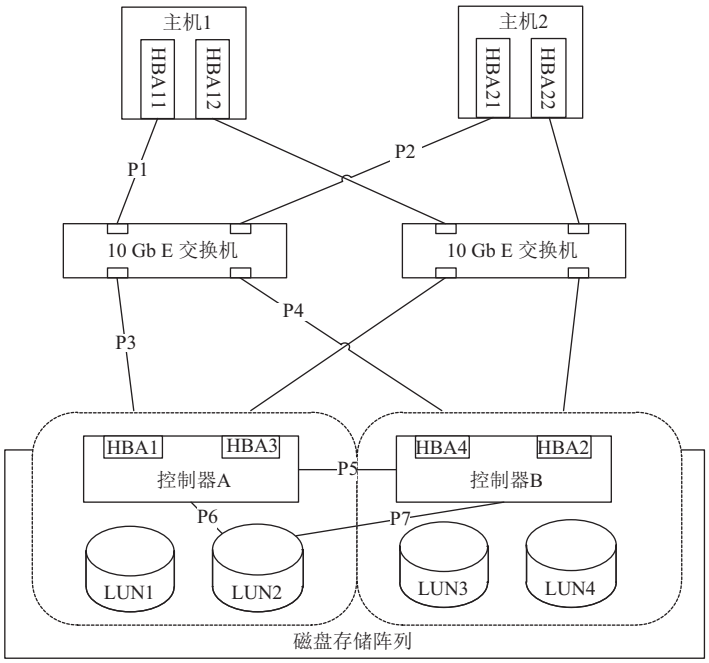

图 7 测试场景示意图

最后, 针对本文提出的方案, 在组网二 (包括图 7 所有路径) 上进行如下测试: 在双控存储阵列上规划创建 20 个卷, 分别模拟不同的用途, 比如, 邮件服务器存储、数据库存储、视频流存储、FTP 存储等, 分别对应小文件随机读写、均匀顺序读写、突发大文件读写等, 通过提出的 LUN 控制器属主管理方法将其分配到不同的控制器上, 并每二十秒测试一次存储端网络 HBA 卡的吞吐量, 持续 3 个小时, 然后求取这些测试

值的均值和变异系数 (CV). 在对照组中, 按照传统做法随机分配 LUN 的所属控制器, 并采用经验证效果较好的 Service-time 选路算法.

将上述流程中不同用途的卷的创建顺序打乱, 重新分配所属控制器, 进行下一次测试. 循环往复测试 6 次, 测试结果如表 2 所示. 从测试数据的对比, 可以看出本文提出的方案在四块 HBA 卡的速率基本一致, 较好地实现了负载均衡的要求, 并比对照组有更好的性能.

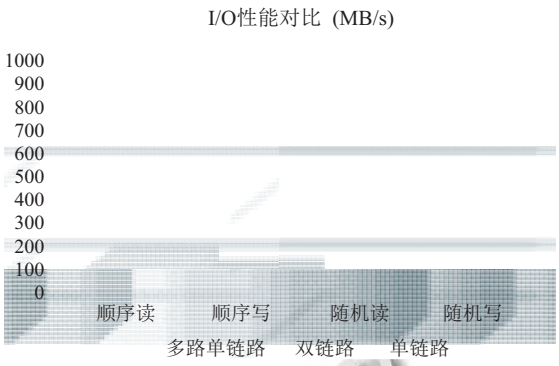

图 8 多路径 I/O 性能测试结果

表 2 多路径 I/O 吞吐量测试结果

编号	Hikdata方案(MB/s)						对照组(MB/s)					
	HBA1	HBA2	HBA3	HBA4	均值	CV(%)	HBA1	HBA2	HBA3	HBA4	均值	CV(%)
1	605.14	593.55	577.61	610.43	596.68	2.11	609.03	592.49	598.37	566.95	591.71	2.61
2	801.37	790.91	760.7	775.46	782.11	1.97	797.68	778.94	761.32	810.75	787.17	2.38
3	411.25	398.83	400.82	377.59	397.12	3.07	407.28	395.46	368.34	372.97	386.0125	4.14
4	780.56	802.42	759.27	771.49	778.43	2.02	743.38	783.64	774.67	754.51	764.05	2.08
5	505.11	485.37	497.29	513.64	500.35	2.07	489.48	502.91	472.92	528.19	498.385	4.05
6	675.69	687.32	705.46	664.38	683.21	2.22	665.19	649.13	684.74	677.28	669.095	2.01

5 结语

在网络存储系统中, 在主机与存储系统之间利用多路径技术进行容灾已成为常例, 既能保证数据业务不因故障而中断, 又可提升系统总体 I/O 传输效率和吞吐量. 为此, 该技术得到了业界的普遍重视, 对它的研究和应用越来越广泛. 但是目前多路径技术方面的研究工作大都局限于主机端多路径软件的设计开发或特定领域的应用, 没有从系统的角度进行考量.

本文提出了一种基于 ALUA 的多路径存储子系统设计与实现方案, 提出了新的多路径存储子系统的体系架构, 并分别在 LUN 控制器属主管理、ALUA 属性与路径选择策略配置和 I/O 重定向等模块进行了创新性的设计与实现. 通过应用效果分析, 本方案能够很好地保证系统的高可用性, 在系统容灾、故障转移及 I/O 分流等方面具有较好的应用效果, 并且有效地进行了 I/O 的负载均衡, 优化了 I/O 子系统的整体性能.

参考文献

1 和军. 基于 Windows I/O 多路径的网络存储容灾技术分析与实现[硕士学位论文]. 成都: 电子科技大学, 2009.
2 魏勇. 详解 Linux 下存储设备多路径管理. 科技风, 2013, (18): 22–23. [doi: 10.3969/j.issn.1671-7341.2013.18.016]
3 焦繁. 论软件 SAN 存储多路径的实现方法. 中国报业, 2012, (24): 30–32.
4 郭晓金, 王超. 一种新的 Linux 主机冗余存储路径设计与实现. 电视技术, 2012, 36(3): 70–73.
5 胡耀义. 一种多路径环境下的智能选路算法设计与实现. 铁路计算机应用, 2016, 25(10): 13–15, 20. [doi: 10.3969/j.issn.1005-8451.2016.10.004]
6 蔡斌, 谢长生, 任劲. SCSI 子系统中间层多启动互连多路径 I/O 的存储方式的研究. 小型微型计算机系统, 2005, 26(8): 1420–1426.
7 阚闯, 戚玮玮. 一种新结构的 DM-multipath 与动态负载平衡. 计算机应用, 2008, 28(2): 289–291.
8 王超. 多路径软件与不同模式磁盘阵列搭配使用方法研究. 电光系统, 2011, (4): 45–49.
9 胡耀义, 陶宏才. 一种基于 Hyper-V 和 WSFC 集群的主机多路径网络存储架构方案. 铁路计算机应用, 2012, 21(6): 80–82.
10 王炎. 多路径 I/O 技术在电视台 SAN 存储网络中的应用探讨. 广播与电视技术, 2013, (3): 86, 88, 90, 92–93.
11 金弟, 庄锡进, 曹晓初, 等. 基于多路径地震资料处理集群存储系统. 计算机研究与发展, 2012, 49(增刊): 42–46.
12 金弟, 庄锡进, 王启迪, 等. 存储框架模型在地震资料大数据中的应用. 计算机系统应用, 2016, 25(2): 45–51.