第 32 卷 第 6 期 计 算 机 工 程 2006 年 3 月
Vol.32 № 6 Computer Engineering March 2006
·软件技术与数据库· 文章编号：1000—3428(2006)06—0067—03 文献标识码：A 中图分类号：TP311.1

蓝鲸分布式文件系统的物理资源管理模型

黄 华[1,2]，张敬亮[1,2]，张建刚[1]，许 鲁[1]
（1. 中国科学院计算技术研究所，北京 100080；2. 中国科学院研究生院，北京 100039）

摘 要：蓝鲸分布式文件系统可以管理数百个存储节点，向上千个应用服务器提供远程文件访问服务，提供超大规模的系统容量。其物理资源管理模型能有效地管理分布的存储资源，形成统一的地址空间，动态分配各种资源，缩短查找和跟踪路径。此模型为整个蓝鲸分布式文件系统提供了统一的资源管理机制，是文件系统并发访问的基础。

关键词：文件系统；分布式文件系统；资源管理；蓝鲸

Model of Physical Resource Management in Blue Whale Distributed File System

HUANG Hua[1,2], ZHANG Jingliang[1,2], ZHANG Jiangang[1], XU Lu[1]
(1. Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080;
2. Graduate School of Chinese Academy of Sciences, Beijing 100039)

【Abstract】 The blue whale distributed file system can manage hundreds of storage nodes and provide remote file access service to thousands of client nodes. Its model of physical resource management efficiently manages the storage resources distributed among multiple nodes, has a global address space, dynamic allocates various objects, and reduces the searching and tracking path. This model provides a uniform resource management mechanism and is the basis of parallel operations in blue whale distributed file system.

【Key words】 File system; Distributed file system; Physical resource management; Blue whale

中国科学院计算技术研究所国家高性能计算机工程研究中心研制开发的蓝鲸分布式文件系统(Blue Whale Distributed File System，BWFS）是一个能够管理上百个存储节点，向数千个应用节点提供文件服务的大规模系统软件。BWFS 为并行计算及信息处理等软件在集群环境下的数据交换和存储提供了坚实的基础和卓越的性能。BWFS 的物理资源管理模型能有效地管理分布的存储资源，根据不同需要动态分配各种文件系统的对象。此模型使用精心设计的数据结构来描述物理资源，使得系统可以快速分配和查找资源。

1 BWFS 的体系结构

BWFS 是一个大规模的集群文件系统，它可以管理上百个存储节点上的千万亿字节的存储空间，向数千个客户端提供远程文件访问服务，其体系结构如图 1 所示（图中 AS 表示应用服务器，MS 表示元数据服务器，AD 表示管理服务器，BS 表示绑定服务器，SN 表示存储节点，实线表示它们之间通过千兆以太网连接）。

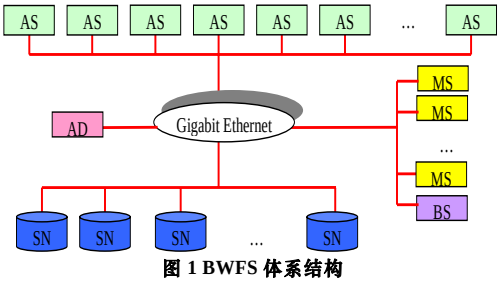

图 1 BWFS 体系结构

BWFS 可以配置若干个元数据服务器，所有文件系统元数据相关的操作都由元数据服务器完成，而文件系统的数据直接在应用服务器和存储服务器之间传递，无须经过元数据服务器转发。所有存储服务器的存储资源形成统一的地址空间，形成共享磁盘的结构。绑定服务器协调多个元数据服务器之间的并行操作，保证元数据操作的一致性。绑定服务器决定在某个时刻某个活跃元数据绑定在某个特定的元数据服务器上，此后与此元数据相关的操作都由该元数据服务器完成。管理服务器负责整个系统的配置管理，负责全局资源的调配，控制整个系统的启动和停止等。整个集群在管理服务器上实现单点管理，其它节点在启动以后向管理服务器汇报状态，接收配置信息，服从系统的统一管理。相比 NFS，BWFS 的这种带外（out-of-band）数据管理模式缩短了数据传输的路径，有效提高了系统的性能。

BWFS 支持在线添加存储节点，也支持在线添加元数据服务器，并能无缝地将这些新增节点纳入到原系统中。这种动态扩充能力使得 BWFS 具有很好的扩展性，使之可以随着应用需求的增长随时扩充系统的数据存储能力和数据吞吐能力。BWFS 拥有这些优点都是基于一个优异的物理资源管理模型，下面将介绍此模型。

2 BWFS 的物理资源管理

文件系统是管理物理磁盘空间的一种系统软件，它向用户提供一组基于文件（普通文件、目录文件和其他特殊文件

基金项目：国家“863”计划基金资助项目(2002AA1120010)
作者简介：黄 华(1978—)，男，博士生，主研方向：分布式文件系统，海量存储等；张敬亮，硕士生；张建刚，博士、副教授；许 鲁，博士、研究员、博导
收稿日期：2005-03-20 **E-mail**：huanghua@ict.ac.cn

的统称）的访问接口，简化数据读写操作。文件系统物理资源管理的主要职责就是记录它所管理的设备中任何一段有效空间的使用情况，以及如何定位某一个文件系统的对象（比如文件索引节点，inode）在设备中的存储位置等。

根据物理资源的不同用途，文件系统一般将存储空间划分成系统块、索引节点块和数据块等。系统块一般用来存放文件系统内部使用的重要配置信息和数据结构（比如超级块等），它们一般在文件系统创建时就被确定存放在固定的位置，而且被一直占用，不再释放。因此，本文不讨论有关系统块的管理问题。索引节点一般用来存放文件的属性信息，包括文件的大小、各种时间戳以及文件数据块的存放位置等。数据块用来保存文件数据、目录文件的内容或者内部的间接指针等。这两种不同用途所占用的存储空间会随着文件操作不断被释放，再占用，再释放等循环往复。对于本地文件系统（比如 EXT2）来说，物理资源管理的功能大致就是确定物理存储空间中的某一部分是否已经被占用，如何根据索引节点号确定索引节点的存放位置等。

BWFS 是一个大规模的集群文件系统，能管理上百个存储节点上的千万亿字节的存储空间。物理资源管理是整个文件系统的基础，它为整个系统提供一种访问物理资源和使用物理资源的接口。为了简化文件系统的设计，也为了能够更好地管理分布的存储资源，我们将所有系统可用的物理存储空间统一编址，形成一个巨大的逻辑地址空间。BWFS 通过该逻辑地址存取系统中的物理资源，形成共享磁盘的结构。将此逻辑地址称为全局逻辑地址（GLA），它是一个 64 位无符号整数。BWFS 中有专门的一个模块来管理 GLA，称之为全局逻辑地址管理器（GLAM），它主要负责将系统中的物理存储空间映射到 GLA。系统中可以访问的 GLA 不一定是连续的，这取决于 GLAM 是否将物理存储空间映射到连续的 GLA。

虽然所有的物理存储空间形成了统一的逻辑地址空间，但是由于这些物理存储资源来自不同的存储节点，可能采用不用的磁盘、不同的控制器、不同的网络连接等，因此表现出不同的读写性能。为了区分这些不同的属性，并向资源的使用者（文件系统上层以及应用程序）提供根据不同属性分类的存储空间，我们引入资源组（RG）的概念。一个资源组是一类具有相同属性或者相似属性的物理存储资源的集合。为了简化整个资源组的管理，规定一个资源组是全局逻辑地址中的一段连续空间。BWFS 中的每一个 RG 都有一个对应的资源组管理器（RGM），它负责管理该 RG 的属性和资源的使用情况。同时，BWFS 中还有一个全局资源组管理器（GRGM）负责协调各个 RGM 之间的工作。在图 2 中，GLAM 将 3 个存储节点上的 6 个物理存储设备映射到从 0~A6 的全局逻辑地址，然后 GRGM 又将它们分成 3 个 RG，分别是 RG1，RG2 和 RG3。

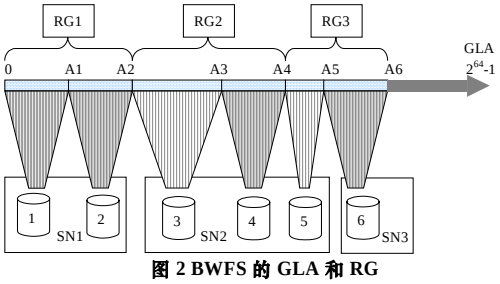

图 2 BWFS 的 GLA 和 RG

通过这种模型，每一个需要访问分布物理存储资源的节点，都可以通过 GLA 存取数据，而无须关心数据实际存放的物理位置。负责进行数据传输的模块可以通过查询 GLAM 获得 GLA 和实际物理位置之间的关系。在实际的数据存取时，元数据服务器可以根据用户的需求或者当前系统的负载状况等因素决定某一个文件索引节点和数据块存放的资源组，更好地满足应用性能及可靠性的需求。

3 资源组管理器

一个资源组是一段连续全局逻辑地址所表示的具有相同或者相似属性的物理存储空间。每一个资源组都有一个资源组管理器来负责对其上可用的存储空间进行管理，主要是进行索引块和数据块的分配与回收，以及索引块和数据块的定位查找。

在 BWFS 中，每一个 RG 都被赋予一个唯一的资源组号。我们约定，每一个 RG 最大管理 2TB(2^{41})的存储空间，并且 RG 使用 GLA 表示的起始地址一定是 2T 的整数倍。在这样的情况下，使用 RG 的起始地址的高 23 位作为资源组号。

3.1 数据块管理

数据块管理的主要任务就是分配和回收该 RG 中的数据块。BWFS 的数据块可以用来存放索引节点、文件系统使用的间接指针、目录项和文件的数据等。一般的本地文件系统，比如 Linux 中的 Ext2/3 文件系统使用位图（跟踪数据块的使用情况。BWFS 管理的物理资源总量十分庞大，单纯使用位图进行数据块分配与回收的效率十分低下，同时难以进行各种高级分配策略，比如连续分配等。因此我们在使用位图的基础上，加入了多层次统计信息，实现快速分配和各种高级分配策略。

图 3 表示一个 RG，它从全局逻辑地址 x 开始，共有 n(n≤2T)个字节长，每一个方框表示 4 096 个字节。RG 的起始部分是一个超级块，它存放有关此 RG 的属性信息，比如 RG 的长度、RG 的读写性能等；还有此 RG 中空间分配状态的信息，比如空闲数据块数目等。接着超级块的是 256 个统计块和 m 个位图块。在此之后都是被管理的数据块。

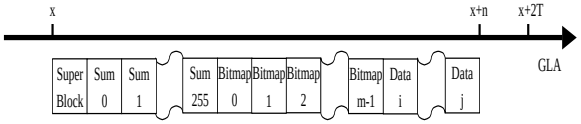

图 3 BWFS 中 RG 的数据块管理

每一个位图块实际管理 8 192 个数据块的分配情况。如果每一个数据块按照 4kB 计算的话，一个位图块实际管理 32MB 的存储空间。位图块中有它所管理的空间中空闲数据块数目、第 1 个空闲数据块的相对位置、最大连续空闲块的相对位置等，有助于加速分配的信息。为了提高系统的可用性和一致性，又将这些信息复制，存储在同一位图块的前后两半部分。在系统出现故障需要恢复的时候，可以对比前后两半部分的内容进行必要的恢复。

每一个统计块统计 256 个对应的位图块所表示的资源分配情况。它包含一个有 256 个元素的数组和其它一些信息。数组的每个元素记录对应位图块表示的空间中空闲块的数据和此段空间中最大连续空闲块的数目。如果统计块对应的位图块不存在，也即超过了本 RG 的最大范围，那么统计块中所有资源总数都是 0，表示没有空闲块。

超级块中有一个含有 256 个元素的数组，用来统计统计

块所描述的分配信息，采用同样原理进行工作。在实际应用中，一般是将超级块和统计块缓存在内存中，其内容即使被修改了也不会立刻同步到磁盘，而每次修改位图块都需同步写回磁盘。位图块中保存着整个 RG 中数据块的分配情况，统计块的内容和超级块中的统计信息都是基于此计算出来的。

在进行数据块分配时，首先查找超级块中的统计信息，确定满足要求的统计块。再查找统计块，确定符合要求的位图块。最后分析位图块中的位图信息，分配合适的数据块。根据用户的不同需求，比如分配地址连续的数据块或者其它要求，就可以依据各个层次的统计进行匹配。在进行数据块分配时，首先更新位图块，并同步到磁盘，再更新统计块和超级块。在进行数据块回收时，直接定位位图块，修改位图信息，同步写回磁盘，然后逐级更新统计信息。所有的统计信息在停止服务之前才同步到磁盘。这种层次结构的模型可以明显减少磁盘操作的次数，加速分配、释放和检索的效率。

RG 的数据块管理只涉及 RG 内部的资源。RG 的实际存储位置和它的起始 GLA 之间没有必然的联系，它们之间的映射关系由管理员确定。我们可以通过改变这种映射关系，动态迁移 RG 中保存的数据，实现各个存储节点之间的负载平衡，而这一切对资源的使用者都是透明的，也就是说无须改变文件的目录结构、索引节点信息等。

3.2 索引节点管理

一般的文件系统都使用索引节点（inode）来描述一个文件。索引节点保存着文件系统内部使用的文件信息，每一个文件对应一个索引节点。根据不同文件系统的需要，索引节点的大小可以从几个字节到 512 个字节变化，甚至更大。索引节点保存在物理存储设备上，占用一定量的磁盘空间。有些文件系统（比如 EXT2/3）在创建时就决定数据块和索引节点占用磁盘空间的比例，有些（比如 JFS[5]）则根据需要动态分配索引节点所占用的磁盘空间。

BWFS 管理众多的存储节点上的海量存储空间。如果分配固定比例的存储空间给索引节点，在系统只含有少量大文件的情况下，可能会浪费大量的分配给索引节点的存储空间；在系统含有大量小文件的情况下，可能会导致索引节点的存储空间不够用。因此我们必须实现动态分配索引节点。

BWFS 将索引节点存储在数据块中。为了满足海量存储容量和一些高级文件技术的需要，BWFS 的索引节点为 512B，每一个数据块存放 8 个索引节点。

为了既能实现快速动态分配索引节点，又能实现快速跟踪索引节点的存储位置，我们设计了三级指针模型，如图 4 所示。图中最上层是超级块，其中含有 8 个一级指针，分别指向 8 个一级数据块；每个一级数据块又含有 256 个二级指针，指向 256 个二级数据块；每个二级数据块含有 256 个三级指针，指向 256 个真正的索引节点位图块；一个索引节点位图块管理 8 192 个索引节点，存储有这些索引节点的存储位置、统计信息、分配情况等数据，如图 4 的最下面的放大图。按照这样的层次关系，每一个 RG 都能管理 4G（2^{32}）个索引节点。

在实际使用中，一个 RG 不可能保存所有 4G 的索引节点，也无需这样。将全体正在使用的索引节点分配到多个 RG，每一个 RG 存储不重叠的一部分。这样在每一个 RG 上，这棵树状的结构中将出现很多空指针，表示对应的索引节点没有被分配，或者没有存储在此 RG 中。将这一棵树的所有叶子节点顺序线性连接起来，就是一个稀松文件。该稀松文件保存着索引节点位图块，每个位图块占用 4kB 的数据块，管理 8 192 个索引节点，顺序映射。同时，为了加速查找，在每一个索引节点位图块中保存有分配的统计信息，在图 4 的每一个上层数据块中，都会统计该间接块所管辖的索引节点位图块的索引节点分配信息。

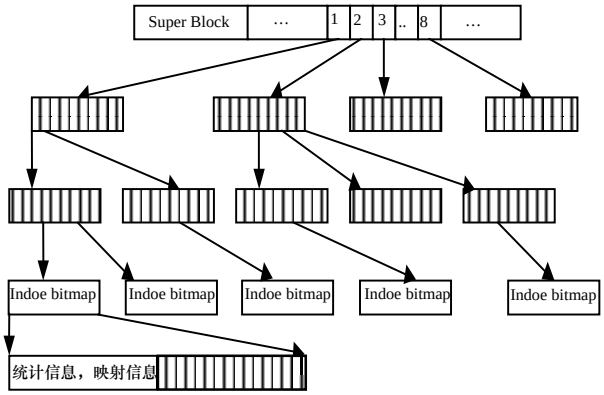

图 4 BWFS 中 RG 的索引节点管理

在进行索引节点分配时，和数据块管理采用同样的原理，无须从头至尾扫描所有的位图以确定如何分配，而是逐级查找统计信息，快速确定空闲索引节点的位置。如果在对超级块遍历以后发现没有满足要求的空闲索引节点，那么就从 GRGM 申请分配索引节点。GRGM 以 8 192 个索引节点为一组分配给 RG。在获得这一组索引节点后，RG 只需通过简单的线性映射来分配各级指针，记录这些资源，并分配数据块保存它们。之后 RG 就可继续提供索引节点分配服务了。

在进行索引节点回收时，申请索引节点回收服务的元数据服务器首先通过 GRGM 查找这些索引节点由哪个 RG 管理，然后再向该 RG 申请回收服务。RG 接收到索引节点回收请求时，根据三级指针，快速定位到相应的位图块，修改其中的位图信息和统计信息。如果在回收一些索引节点以后发现某个位图块描述的索引节点全部是空闲的，就可以将这些索引节点归还给 GRGM，并且释放它们占用的存储空间。

在进行如图 4 所示数据结构的更新时，同样是最先更新最底层的数据，并且同步到磁盘。然后再逐级更新各级统计信息，将它们缓存在内存中，并在系统下线时一次性更新到磁盘。由于已将三级统计信息缓存至内存，可以通过 4 次 4kB 大小的内存检索和两次 4kB 大小的磁盘操作，实现索引节点的分配和回收；同时可以通过 3 次内存比较，确定某一个索引节点的存储位置，因为在二级数据块中保存有索引节点的存储位置信息。

在 BWFS 中，索引节点的存储空间是动态分配的，索引节点的存储位置和索引节点号之间没有任何联系，因此通过改变映射关系可以改变索引节点的存储位置，却无须改变文件系统中的目录项数据结构。利用这个特性，BWFS 可以轻松实现索引节点在 RG 之间的动态迁移和动态负载平衡。

4 总结

蓝鲸分布式文件系统的物理资源管理模型能够有效地管理大量分布的物理存储资源，形成共享磁盘的存储结构，动态分配各种文件系统对象，提高分配和回收的效率，减少查找定位的路径。同时，该模型有效地支持系统进行各种资源分配策略、进行负载平衡以及提高可用性和扩展性等。

(下转第 72 页)

因此，为了提供充分的审计信息，本文提出了增强型的轻量级审计思想。

首先，可以把 UNIX 系统中的所有文件分 3 类：第 1 类是系统安全程序，这些程序执行普通的日常任务，执行频度很高，并且这些程序的行为结果是可知的，因此，对这类程序的运行只需要审计到命令级就可以了。在缺省条件下，所有的/bin、/sbin/等系统中的大多执行简单管理任务的可执行文件应被认为安全程序。另外用户专用的或多用户共享的可执行程序，只要通过安全审核也可以提交管理员注册为安全程序。第 2 类是 setuid 程序，由于这些程序以 root 权限运行，有可能会导致安全问题。第 3 类为未注册程序，这些程序的运行是不能预先知道的。对于这些不同类别的程序处理方法也应有所不同：

（1）对于注册的安全程序只产生基于命令的审计记录（同原先的轻量级审计），因为它们的运行信息已经是先验了解的。

（2）对于 setuid 程序若其有单独的日志，并且信息的粒度足够细，则产生命令级的审计记录，否则认为该进程是不可信的，需对该进程进行详细审计。

（3）非注册的程序产生的进程其运行结果是不可知的，因此被认为是不可信的，也需要详细审计该进程。

（4）进程审计是递归的，即不可信的进程中如果再次出现了安全子进程，则安全子进程的结果又是已经先验了解的，又可被精简为命令级的输出。

只要能够保证注册的安全程序不被篡改，增强型轻量审计收集信息就是充分的。因此有必要采取安全措施，如为每个安全程序产生一个指纹，以发现它们是否被窜改。

并且，如果系统每个用户尽可能将自己主目录的常用可执行文件进行安全检查，提交给管理进行安全注册，就会使轻量级的审计输出减少到与 pacct 基本相当的规格。

4 增强型轻量级的算法及实现

基于安全相关的轻量级审计算法分解成两个子例程，互相调用，子例程 1light_weigh_collect 用于产生父进程为 pid 所有进程产生轻量级输出，而 Detailed_audit_process 会产生当前进程的详细 BSM 记录，但如果又有子进程，处理方法又使用子例程 1。例如：light_weight_collect(1)表示从进程 pid=1 处开始产生审计输出；

子例程 1：轻量级输出算法(调用参数为进程号 pid)：

```
light_weitht_collect (int pid)：{
While (读取下一条进程号是 pid 的 BSM 审计记录!=NULL) {
IF(审计事件是 AUE_fork) {
将 AUE_fork 记录输出；
查找其对应的 AUE_execve 审计记录并判断：
IF (当前记录用户执行的是 setuid 的特权程序)
AND（该特权程序没有自己单独的日志或有日志产生的记录不够详细 {
将当前审计记录输出；
detailed_audit_process（当前记录.pid）
}
ELSEIF (如果用户执行的是非安全程序) {
将当前审计记录输出：
detailed_audit_process ( 当前记录.pid);
}
}  ELSE {直接将当前审计记录输出}
} }
```

子例程 2：当前进程详细审计收集算法（调用参数是审计记录的 pid）

```
detailed_audit_process( int pid){
While(非空) {
读取下一条审计记录进程号为 pid 的记录，并输出。
IF（当前的审计记录是 AUE_fork 事件）{
查找其对应的 AUE_execve 记录
light_weight(AUE_execce.pid)
}} }
```

5 结论

Stefan 提出的轻量级审计算法存在一定的缺陷，主要在于提供的信息不充分，只能用于描述用户的行为，缺乏描述进程行为的能力；另一方面，BSM 审计输出存在极大的冗余，绝大多数的记录与系统安全无关，只是反映了系统的一些管理工作或运行状态的细节。本文提出的增强型的轻量级处理算法以审计安全相关为出发点，把可预知其执行结果的操作系统可执行程序(或已验证的安全程序)从其它可执行程序中区分出来，并把它们产生的审计输出减少到 1 条记录，而所有可能出现问题的进程将会得到充分的审计，所有的信息都会包含在审计输出中，从而合理地解决了审计粒度与数据量的之间的矛盾。

参考文献

1 Axelsson S, UlfLindqvist.An Approach to Unix Security Logging [EB/OL]. http://www.ce.chalmers.se/old/staff/ulfl/pubs/nissc98a.pdf,1998.

2 Bishop M, Wee C, Frank J. Goal-oriented Auditing and Logging [EB/OL]. http://seclab.cs.ucdavis.edu/awb/AuditWorkBench.html.

3 Wetmore B. Paradigms for the Reduction of Audit Trails[D]. Davis: University of California, 1993.

4 Bishop M. A Standard Audit Trail Format[C]. Proceedings of the 18th National Information Systems Security Conference, Baltimore, Maryland, USA, 1995-10:136-145.

5 Li Zhenmin. A Unified, Correalated Logging Archetecture for Intrusion Detection[EB/OL]. http://www.ncassr.org/projects/sift/papers/uclog.pdf.

(上接第 69 页)

参考文献

1 Sandberg R. Sun Network Filesystem Protocol Specification[R]. Technical Report, Sun Microsystems, Inc., 1985.

2 Schmuck F, Haskin R. GPFS: A Shared-disk File System for Large Computing Clusters[C]. Proceedings of the Conference on File and Storage Technologies (FAST'02), 2002-01: 231–244.

3 Braam P. The Lustre Storage Architecture[EB/OL].http://www.lustre.org.

4 吴思宁，贺　劲，熊　劲等. DCFS 机群文件系统服务器组的设计与实现[C]. 2002 全国开放式分布与并行计算学术会(DPCS2002), 2002.

5 Juan I, Santos Florido. Journal File Systems[EB/OL]. http://www.linuxgazette.com/issue55/florido. Htm, 2000-07.