高技术通讯 2016 年 第26 卷 第3 期:235 ~245

doi:10.3772/j.issn.1002-0470.2016.03.003

BW-RAID 系统的分布式异步版本识别机制①

王 慧②*** 郭明阳* 董欢庆* 许 鲁*

(*中国科学院计算技术研究所 北京 100190)

(**中国科学院大学 北京 100049)

摘 要 为解决 BW-RAID 系统中数据冗余模式从镜像向 RAID4 转换时的节点间缓存数据版本识别问题,提出了一种分布式异步版本识别机制。该版本识别机制在镜像卷中的一个逻辑块被更新时,为其生成一个新版本;在冗余模式转换时,通过比较镜像节点间的版本判定某一逻辑块在两个镜像节点缓存的数据是否一致,如果数据一致将其迁移到数据存储卷,否则暂存入镜像节点各自的磁盘缓存以保证系统的冗余一致性。实验表明,该机制在系统正常、降级、故障恢复状态下均能准确、有效识别一致数据;顺序写评测中存储负载小于1%,平均带宽提升最高达25.43%;Open-mail 负载重播评测中存储负载低于40%,应用重播结束时写更新数据均完成冗余模式转换,实现了提高空间利用率的目标。

关键词 网络 RAID, 数据一致性, 分布式异步版本, 泛化时间戳, 版本识别

0 引 言

RAID 是廉价磁盘冗余阵列(redundant array of inexpensive disk),是存储系统的基础和关键部件。RAID 技术大大提高了存储系统的存储容量、可靠性和性价比。蓝鲸分布式集群 BW-RAID(Blue Whale network RAID)[1] 采用镜像与纠删码协作的系统架构,其优势在于既能发挥较高的读写性能又节省存储空间。在工业界类似的混合冗余架构被广泛应用,比如,EMC-Isilon[1,2]、Panasas[3] 均实现了对小文件使用副本冗余,对大文件使用 parity 保证可靠性;HDFS-DiskReduce[4] 则实现了应用写三副本,异步转为纠删码冗余方式保存。上述系统中混合冗余机制均是文件系统级的实现,BW-RAID 则是在通用块层实现了混合冗余。

BW-RAID 采用 Mirror 与 RAID4 混合的冗余方式容忍单点故障。集群中的存储资源划分为若干冗余组(RAID-set),每个 RAID-set 实现对数据的冗余存储。一个 RAID-set 覆盖多个数据卷和一个校验卷,其中包括两类功能节点,即数据存储节点(data node, DN)和校验存储节点(parity node, PN);一个数据卷的写更新数据首先存储在由 DN 和 PN 的缓存组成的镜像中,PN 的后台线程异步地对冷数据进行集中 RAID4 计算,并将完成冗余计算的数据存储到 DN 的数据卷,将校验码更新到 PN 的校验卷,实现数据冗余模式转换。针对冗余模式转换过程中如何解决镜像中的缓存数据一致性问题,本文提出了一种分布式异步版本识别机制,以保证冗余组一致。

1 背 景

RAID-set 的最小宽度为 2,包括两个数据卷和一个校验卷,覆盖两个 DN 节点和一个 PN 节点,布局见图 1(a)。为减少不必要的网络数据传输和磁盘 IO,RAID-set 在进行冗余模式转换时 PN 冗余计

① 863 计划(2013AA013205)和中国科学院重点部署课题(KGZD-EW-103-5(7))资助项目。

② 女,1981 年生,博士生;研究方向:网络存储和分布式系统冗余一致性;联系人,E-mail: wanghui@nrchpc.ac.cn

(收稿日期:2015-10-20)

算和 DN 数据迁移均引用各自缓存中的数据副本，但此时正在进行转换的逻辑块在 DN 与 PN 缓存中的数据对象若不一致将导致 RAID4 的数据与校验不一致，即影响 RAID-set 的冗余一致性。然而在分布式环境中由于存在网络延迟、节点间独立请求调度等因素，互为镜像的缓存中数据不一致的现象难以避免。比如图 1(b)所示情形，DV_1 中的逻辑块 B 在 DN_1 缓存中已更新为[B:V_2]，然而远程写被阻塞，PN 缓存中仍为[B:V_1]，此时若对逻辑块 B 进行冗余模式转换导致 RAID4 不一致。因此缓存数据一致性感知(或数据对象一致性识别)是保证冗余一致性的关键。解决数据对象一致性识别问题的常用方法有以下几种：

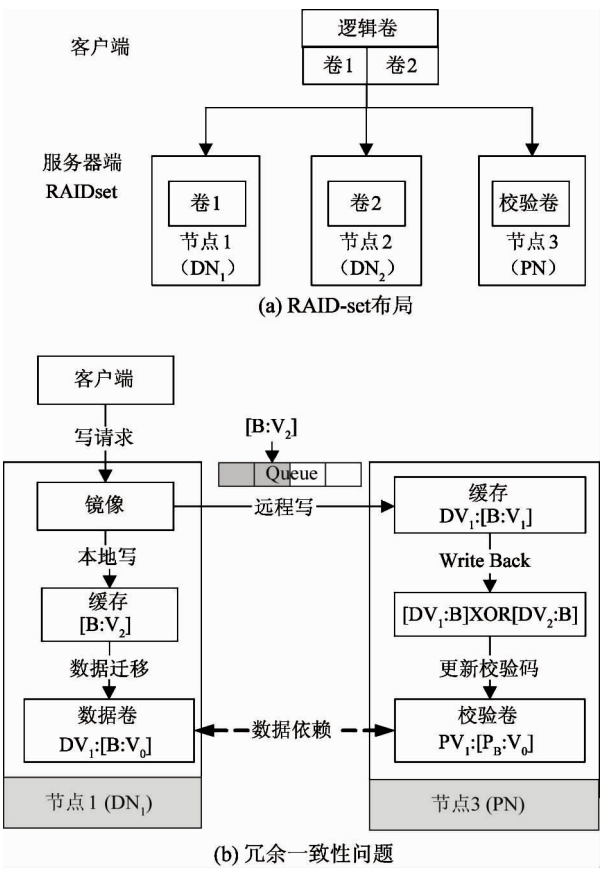

图 1 RAID-set 结构及冗余一致性问题

(1) 内存字符比较：Linux 系统中的 cmp 工具、CMSD 系统[5]等使用此方法比较数据一致性，该方法的不足是，将数据加载到内存增加了磁盘 IO，在分布式环境中需要将数据传输到执行比较的节点又有网络开销。

(2) 数据指纹比较：采用 SHA-1、MD-5 等算法计算数据对象的数据指纹，使用数据指纹索引数据内容，可以实现一致性确定，比如文件系统中的冗余数据识别[6-14]、内存寻址系统[15]、版本管理系统 GIT[16]等均使用该方法，其优点是精确度高，但是数据指纹计算的时间开销、维护的空间开销都较高。

(3) 数据版本识别：(a)物理时间戳，通过比较数据对象产生的物理时间戳识别其一致性，其优点是时间由操作系统维护的，无需单独的模块支持，该方法在理论上可行，然而在分布式环境中要求各节点的时间须完全同步，难以实现；(b)逻辑时间版本，其本质是请求序列号，通过比较版本判断数据更新发生的序列关系确定一致性，该方法可以不依赖于物理时间。比如版本管理系统 SVN[17]，使用自增量作为其版本，Lamport timestamp[18]通过记录更新操作发生的偏序关系判断节点间的数据更新一致性，Dynamo[19]采用了该类算法解决并发访问中的数据一致性问题。逻辑时间版本方式易于实现，其最显著的特点是节点间数据同步时需要同时同步其版本号，目前通用块层成熟的传输协议(如 Iscsi)不支持该功能。(c)物理时间与逻辑时间结合的版本，每隔固定时间更新并同步节点间的全局逻辑时间，各节点在全局逻辑版本为基础独立更新本地逻辑时间，该方法允许节点间存在合理的时钟偏移，这可以降低时钟同步开销，并且逻辑时间同步及更新与数据同步可以解耦，WACE[20]使用了该方法解决缓存集群中的缓存一致性问题。

上述方法难以满足 BW-RAID 对数据对象一致性识别的要求，为此，本文提出了一种分布式异步版本识别机制解决该问题，该版本机制支持物理时间与逻辑时间结合的实现方式。

2 分布式异步版本识别

2.1 概念

主节点(master serving node, MSN)：缓存镜像中接收客户端请求的存储节点，正常情况下默认是数据节点(DN)。

从节点(slave serving node, SSN)：缓存镜像中

接收镜像写请求的存储节点,正常状态下默认是校验节点(PN)。

关联请求:主节点写请求和从节点与之对应的镜像写请求为一对关联请求。

2.2　基于精确时间的版本识别

2.2.1　精确时间版本定义

假设系统开始运行时主、从节点时间同步,主节点串行的接收对一个数据卷中同一逻辑块地址的写更新。我们对同一逻辑地址的写请求按发生次序使用自然数编号,主、从节点的请求分别表示为 $Q_i(i=1,2,3,\cdots)$, $q_j(j=1,2,3,\cdots)$。

根据同步镜像特点则有:

推论 1:主节点写请求 Q_i 与从节点访问相同地址的写请求 q_j 为关联请求的充要条件是 $i=j$。

识别关联请求的关键是探寻 $i=j$ 的充分条件。由数理逻辑可知:

$$i=j \Leftrightarrow i+1>j \,\&\&\, j+1>i$$
$$\overline{i=j} \Leftrightarrow \overline{i+1>j \,\&\&\, j+1>i} \Leftrightarrow i+1\leqslant j \,\|\, j+1\leqslant i \tag{1}$$

只需要探寻 $i+1\leqslant j \,\|\, j+1\leqslant i$ 的必要条件。由此结合访问某逻辑地址的当前写请求与其后继写请求的时间属性,我们给出了写请求版本定义,该版本同样是写请求所包含数据对象的版本。

定义 1:存储节点上一个逻辑数据卷中的任意逻辑块 B,在当前节点对其数据进行更新的写请求 Q_i 的版本表示为 $<A_i, C_i, A_{i+1}>$:

(1) A_i:Q_i 在当前节点发生的精确系统时间;

(2) C_i:Q_i 在当前节点完成的精确系统时间;

(3) A_{i+1}:写请求 Q_{i+1} 在当前节点发生的精确系统时间。

2.2.2　精确时间版本识别

假设主、从节点访问相同逻辑块 B 的任意两个写请求,Q_i:$<(A_i,C_i,A_{i+1}>$, q_j:$<a_j,c_j,a_{j+1}>$,我们以 Q_i 与 q_j 为参照论证关联请求版本识别方法。论证分为三步:论证 $i+1\leqslant j$ 的必要条件;论证 $j+1\leqslant i$ 的必要条件;论证 $i=j$ 的充分条件。

(1) $i+1\leqslant j$ 的必要条件:

$i+1\leqslant j$ 时,Q_{i+1},Q_j,q_j 在主、从节点的逻辑时序如图 2(a)。参数说明:[d] 为网络传输延迟;[δ] 为 DN 与 PN 当前时钟误差;[Δm] 为写缓存时间。

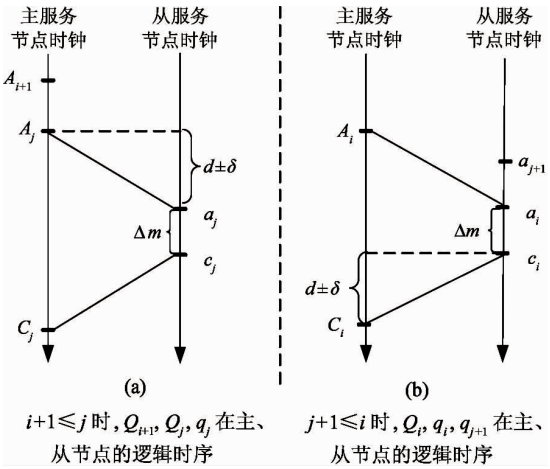

图 2　关联请求逻辑时序

由时序图可得推导式:

$$\begin{aligned}
&i+1\leqslant j\\
\Rightarrow &A_{i+1}\leqslant A_j\\
\Rightarrow &A_{i+1}\leqslant a_j-d\mp\delta\ (\because A_j=a_j-d\mp\delta)\\
\Rightarrow &A_{i+1}\leqslant c_j-\Delta m-d\mp\delta\ (\because a_j=c_j-\Delta m)\\
\Rightarrow &A_{i+1}<c_j+\delta
\end{aligned} \tag{2}$$

即 $A_{i+1}<c_j+\delta$ 是 $i+1\leqslant j$ 的必要条件。

(2) $j+1\leqslant i$ 的必要条件:

$j+1\leqslant i$ 时,Q_i,q_i,q_{j+1} 在主、从节点的逻辑时序如图 2(b),则有推导式:

$$\begin{aligned}
&j+1\leqslant i\\
\Rightarrow &a_{j+1}\leqslant a_i\\
\Rightarrow &a_{j+1}\leqslant c_i-\Delta m(\because a_i=c_i-\Delta m)\\
\Rightarrow &a_{j+1}\leqslant C_i-\Delta m-d\mp\delta(\because c_i=C_i-d\mp\delta)\\
\Rightarrow &a_{j+1}<C_i+\delta
\end{aligned} \tag{3}$$

即 $a_{j+1}<C_i+\delta$ 是 $j+1\leqslant i$ 的必要条件。

由式(1)、(2)得推导式:

$$(i+1\leqslant j)\,\|\,(j+1\leqslant i)\Rightarrow(A_{i+1}<c_j+\delta)\,\|\,(a_{j+1}<C_i+\delta) \tag{4}$$

即 $(A_{i+1}<c_j+\delta)\,\|\,(a_{j+1}<C_i+\delta)$ 是 $(i+1\leqslant j)\,\|\,(j+1\leqslant i)$ 的必要条件

(3) $i=j$ 的充分条件:

式(4)的逆否命题成立,即有

$$\overline{(A_{i+1} < c_j + \delta) \parallel (a_{j+1} < C_i + \delta)} \Rightarrow \overline{(i+1 \leqslant j) \,||\, (j+1 \leqslant i)} \Leftrightarrow (i+1 > j) \&\& (j+1) > i \Leftrightarrow i = j \tag{5}$$

由此得:**定理 1**: $(A_{i+1} \geqslant c_j + \delta) \&\& (a_{j+1} \geqslant C_i + \delta) \Rightarrow i = j$

□

证毕。

2.2.3 节点故障恢复

BW-RAID 中节点故障恢复时,在接替节点数据恢复写请求与应用写请求并存。我们需要论证集群恢复过程不会发生误判,即:(1)恢复到新节点的数据不会误判为某一版本的旧数据;(2)恢复到新节点的数据不会误判为某一版本的新数据。结合定理 1,对于上述两种情况只需证明 $\overline{i = j}(i \neq j) \Rightarrow (A_{i+1} < c_j + \delta) \parallel (a_{j+1} < C_i + \delta)$。

首先,定理 1 的前提是一个数据卷中同一逻辑地址的写请求串行地进入存储节点,通过调研 dm-raid1、md-raid1 等常用的镜像实现方式,我们发现在镜像恢复过程中会阻塞对正在恢复的逻辑地址的应用写更新,直到该逻辑地址恢复完成,即对于同一逻辑地址数据恢复请求与应用写请求是串行的。其次,镜像恢复时将恢复窗口内地址的最新版本数据同步到接替节点。最后,新节点接入后集群恢复以 RAID-set 为单元并发执行。我们以一个 RAID-set 中的数据节点恢复和校验节点恢复为例分别论证。

(1) 数据节点恢复

假设 RAID-set 中的 DN_1 故障(见图 1),替换节点为 DN'_1,恢复开始时 DN'_1 与 PN 时钟同步;DV_1 中的任意逻辑块 B,在 DN'_1 的数据恢复写请求 Q_i: $< A_i, C_i, A_{i+1} >$, PN 的写请求 q_j: $< a_j, c_j, a_{j+1} >$, Q_i 与 q_j 所包含数据对象不一致。

证明:

1) DN'_1 的恢复数据不会误判为 PN 的任意旧版本数据(即 $i > j$)。

假设逻辑块 B 在 T_r 时刻开始恢复,此时应用写请求 q_i 在 PN 节点已经完成,恢复线程将 q_i 对应用的数据同步到 DN'_1,该同步恢复请求即为 Q_i, 对应的逻辑时序见图 3(a),则有:

$$\begin{aligned} & j < i \Rightarrow a_i \geqslant a_{j+1} \\ & T_r \geqslant a_i \Rightarrow T_r \geqslant a_{j+1} (\because a_i \geqslant a_{j+1}) \end{aligned} \tag{6}$$

上述推导式结合时序图可得:

$$\begin{aligned} & C_i = T_r + \Delta m + d \mp \delta \\ & \Rightarrow T_r = C_i - \Delta m - d \mp \delta \\ & \Rightarrow C_i - \Delta m - d \mp \delta \geqslant a_{j+1} (\because T_r \geqslant a_{j+1}) \\ & \Rightarrow C_i + \delta > a_{j+1} \end{aligned} \tag{7}$$

因此则有:$j < i \Rightarrow C_i + \delta > a_{j+1}$, 即 DN'_1 的恢复请求不会误判为 PN 的任意旧版本请求。

2) DN′1 的恢复数据不会误判为 PN 的任意新版本数据(即 $i < j$)。

该情况下一个逻辑块的数据恢复完成后新的应用写更新发生,即 DN'_1 节点的 Q_i 对应某一旧版本,而 PN 的 q_j 为恢复完成后的某一新版本,在 DN'_1 与之对应的是 Q_j, 时序见图 3(b),此时则有:

$$\begin{aligned} & i < j \Rightarrow A_{i+1} \leqslant A_j \\ & A_j = C_j - \Delta m \\ & \Rightarrow A_j = c_j - \Delta m - d \mp \delta (\because c_j = C_j + d \pm \delta) \\ & \Rightarrow A_{i+1} \leqslant c_j - \Delta m - d \mp \delta (\because A_{i+1} \leqslant A_j) \\ & \Rightarrow A_{i+1} < c_j + \delta \end{aligned} \tag{8}$$

根据上述推导式可知 $i < j \Rightarrow c_j + \delta > A_{i+1}$, 即 DN'_1 的恢复数据不会误判为 PN 的任意新版本数据。

3) 结论:综合 DN 恢复时的两种情况则有 $i \neq j \Leftrightarrow i < j \,||\, j < i \Rightarrow (c_j + \delta > A_{i+1}) \,||\, C_i + \delta > a_{j+1}$。证毕。□

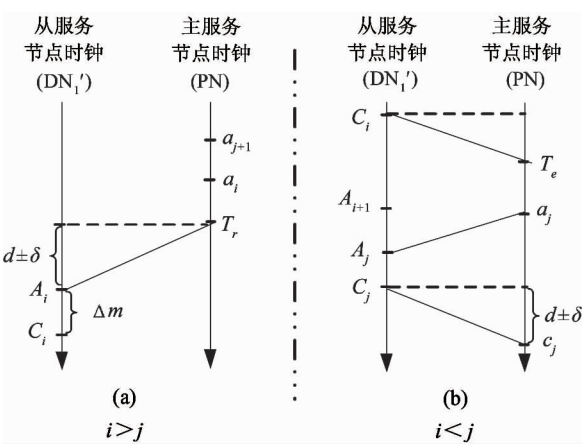

图 3 DN′恢复的请求时序图

(2) 校验节点恢复

假设 RAID-set 中 PN 故障(见图 1),接替节点为 PN′,恢复开始是 PN′与两个 DN 时钟均同步;DV_1 卷中的任意逻辑块 B,在 DN_1 有写请求 Q_i: $< A_i, C_i, A_{i+1} >$,在 PN′有恢复请求 q_j: $< a_j, c_j, a_{j+1} >$; Q_i 与 q_j 对应不同的数据对象。

证明:

1) PN′的恢复数据不会误判为 DN_1 的任意旧版本数据(即 $j > i$)。

假设 DN_1 在 T_r 时刻恢复逻辑块 B 数据到 PN′,PN′的恢复请求 q_j: $< a_j, c_j, a_{j+1} >$ 对应恢复时刻的最新数据,在 DN_1 与 q_j 对应的应用写请求 Q_j: $< A_j, C_j, A_{j+1} >$ 较 Q_i 晚到达存储节点。逻辑时序关系见图 4(a),则有推导式:

$$\begin{aligned} &A_{i+1} \leqslant A_j < T_r (\because j > i) \\ &\Rightarrow A_{i+1} < c_j - \Delta m - d \mp \delta (\because T_r = c_j - \Delta m - d \mp \delta) \\ &\Rightarrow A_{i+1} < c_j + \delta \end{aligned} \tag{9}$$

即有 $j > i \Rightarrow A_{i+1} < c_j + \delta$,那么 PN的恢复数据不会误判为 DN_1 的任意旧版本数据。

2) PN′的恢复数据不会误判为 DN_1 的任意新版本数据(即 $j < i$)。

根据镜像恢复逻辑在逻辑块 B 恢复时刻 DN_1 的最新更新对应 Q_j,而 Q_i 为恢复完成后到达的某一应用写请求,逻辑时序见图 4(b),有推导式:

$$\begin{aligned} &a_{j+1} \leqslant a_i (\because j < i) \\ &\Rightarrow a_{j+1} \leqslant c_i - \Delta m (\because a_i = c_i - \Delta m) \\ &\Rightarrow a_{j+1} \leqslant C_i - \Delta m - d \mp \delta (\because c_i = C_i - d \mp \delta) \\ &\Rightarrow a_{j+1} < C_i + \delta \end{aligned} \tag{10}$$

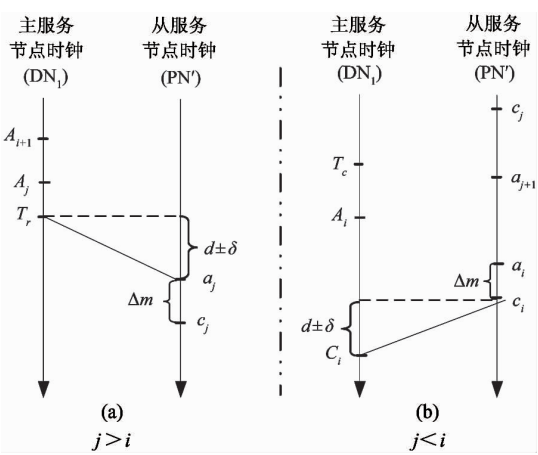

图 4　PN 节点恢复请求时序图

即有 $j < i \Rightarrow a_{j+1} < C_i + \delta$,亦即 PN的恢复数据不会误判为 DN_1 的任意新版本数据。

3) 结论:综合 PN 恢复时的两种情景则有 $i \neq j \Leftrightarrow i < j \,||\, j < i \Rightarrow (c_j + \delta > A_{i+1}) \,||\, C_i + \delta > a_{j+1}$。证毕。□

(3) 结论

综合集群故障恢复时 RAID-set 中 DN 恢复与 PN 恢复两种情况,对于任意两个数据对象不一致的恢复写请求与应用写请求均满足推导式:

$$i \neq j \Leftrightarrow i < j \,\|\, j < i \Rightarrow (c_j + \delta > A_{i+1}) \,\|\, C_i + \delta > a_{j+1} \tag{11}$$

由此证明集群恢复过程中新节点的镜像恢复请求不会误判为正常服务节点的任一不相关联的写请求。进而根据数理逻辑则可知集群恢复状态下定理 1 同样成立。

2.3　应用局限性

定理 1 表明通过比较镜像节点间请求的精确物理时间戳来识别关联请求的方法理论上可行,实际应用中仍存在以下局限性:

局限 1:保证数据与版本一致须将数据更新与版本更新绑定,当缓存数据合并时,精确物理时间版本合并违背定义 1 语义。

局限 2:定义 1 与定理 1 依赖于后继应用写请求到达的精确物理时间,因而版本识别须在后继写请求发生之后进行,否则请求版本不完整。

针对上述两方面局限性,本文提出了泛化时间戳版本理论。

2.4　基于泛化时间戳的版本机制

2.4.1　泛化时间戳版本定义

定义 2:对于存储节点一个逻辑数据卷中的任意逻辑块 B,在本节点对其数据进行更新的任意写请求 Q_i 的泛化时间戳版本为 $< A'_i, C'_i, A'_{i+1} >$:

(1) A'_i:Q_i 在本节点发生时间不早于 A'_i;

(2) C'_i:Q_i 在本节点完成时间不晚于 C'_i;

(3) A'_{i+1}:后继写请求 Q_{i+1} 在本节点发生时间不早于 A'_{i+1}。

推论 2:假设任意节点的写请求 Q_i 的精确时间版本为 $< A_i, C_i, A_{i+1} >$,对于任意时间三元组 $< A'_i, C'_i, A'_{i+1} >$,若同时满足 $A'_i \leqslant A_i$, $C'_i \geqslant C_i$,

$A'_{i+1} \leqslant A_{i+1}$，则 $< A'_i, C'_i, A'_{i+1} >$ 为 $< A_i, C_i, A_{i+1} >$ 的合法泛化时间版本。

2.4.2 泛化时间戳版本识别

假设一个数据卷中任一逻辑块 B 在主、从节点的两个写请求为 Q_i,q_j，对应的泛化时间版本分别为 $< A'_i, C'_i, A'_{i+1} >$，$< a'_j, c'_j, a'_{j+1} >$，则有：

定理 2：$(a'_{j+1} \geqslant C'_i + \delta) \&\& (A'_{i+1} \geqslant c'_j + \delta) \Rightarrow i = j$

证明：

假设 Q_i，q_j 的精确物理时间属性分别为 (A_i, C_i, A_{i+1})，(a_j, c_j, a_{j+1})，由定义 2，则有

$$A'_i \leqslant A_i \&\& C'_i \geqslant C_i \&\& A'_{i+1} \leqslant A_{i+1}$$

$$a'_j \leqslant a_j \&\& c'_j \geqslant c_j \&\& a'_{j+1} \leqslant a_{j+1}$$

$$A'_{i+1} \geqslant c'_j + \delta$$

$$\Rightarrow A_{i+1} \geqslant c'_j + \delta (\because A_{i+1} \geqslant A'_{i+1})$$

$$\Rightarrow A_{i+1} \geqslant c_j + \delta (\because c'_j \geqslant c_j)$$

$$a'_{j+1} \geqslant C'_i + \delta$$

$$\Rightarrow a_{j+1} \geqslant C'_i + \delta (\because a_{j+1} \geqslant a'_{j+1})$$

$$\Rightarrow a_{j+1} \geqslant C_i + \delta (\because C'_i \geqslant C_i)$$

因此有

$$(a'_{j+1} \geqslant C'_i + \delta) \&\& (A'_{i+1} \geqslant c'_j + \delta)$$

$$\Rightarrow (a_{j+1} \geqslant C_i + \delta) \&\& (A_{i+1} \geqslant c_j + \delta)$$

$$\Rightarrow i = j (\because \text{定理 1})$$

证毕。□

2.5 应用局限性解决

2.5.1 版本合并方法

基于定义 2 提出了版本合并方法解决局限 1，使得存储层数据合并时，新旧版本同时合并。版本合并须保证定理 2 适用，须满足两个条件：

(1) 合并后版本是新数据的合法泛化时间版本，即符合定义 2 语义。

(2) A_i 对应同一逻辑块写请求 Q_{i-1} 的后继写请求的泛化发生时间。

版本合并方法：假设任意逻辑块 B 当前版本为 $< A_i, C_i, A_{i+1} >$，后继写请求版本为 $< A_{i+1}, C_{i+1}, A_{i+2} >$，逻辑块 B 数据合并时，两个版本项按如下规则合并：$\{A_i = \text{MIN}(A_i, A_{i+1})$；$C_{i+1} = \text{MAX}(C_i, C_{i+1})$；$A_{i+2} = A_{i+2}$；$\}$ $< A_i, C_{i+1}, A_{i+2} >$ 作为新版本覆盖旧版本。

证明：

假设两个待合并的泛化时间版本对应的精确时间版本分别为 $< a_i, c_i, a_{i+1} >$ 和 $< a_{i+1}, c_{i+1}, a_{i+2} >$，根据定义 2 则有 $A_i \leqslant a_i \leqslant a_{i+1}$，$C_{i+1} \geqslant c_{i+1}$，$A_{i+2} \leqslant a_{i+2}$，再根据推论 2 可得 $< A_i, C_{i+1}, A_{i+2} >$ 是合并后数据的合法版本。

版本合并前，A_i 是 Q_{i-1} 后继写请求的泛化发生时间，每次版本合并均保留 A_i，即新版本可以保证当前版本中请求到达时间为 Q_{i-1} 的后继写请求的泛化到达时间。

因此新版本符合定理 2 适用的两个条件。

证毕。□

2.5.2 后继写请求时间扩展

推理 1：假设任意逻辑块 B 的某一版本 $< A_i, C_i, A_{i+1} >$，若进行版本比较时 $A_{i+1} = \text{NULL}$，则取 $A_{i+1} = \text{SYS}_{\text{current.}}$。

证明：

假设 Q_{i+1} 发生的精确时间为 A_{next}（A_{next} 或趋于 ∞），则有 $A_{i+1} \leqslant A_{\text{next}}$，因此 A_{i+1} 时间符合定义 2 语义，即该版本合法且定理 2 适用。

证毕。□

2.6 小 结

论证结果表明，利用物理时间戳版本解决 BW-RAID 中镜像节点间的数据版本识别问题是理论可行的，通过对物理时间戳做泛化扩展可以满足实际系统的需求，基于此形成了基于泛化时间戳的分布式异步版本识别机制。该版本机制优点是：(1) 易于实现，不需要冗余组内的节点时间保持完全同步，允许节点间存在合理的时间误差，时间同步难度和开销降低。(2) 时间复杂度低，其实现的时间复杂度主要是版本生成及更新时检索版本项的时间消耗，本文的原型系统中将使用基树 (Radix tree) 管理版本项，查找操作的时间复杂度为 $O(h)$，h 为基树高度，在 64 位 x86 平台的 Linux 操作系统中基树最大高度仅为 16。

3 原型实现及评测

3.1 原型系统实现

本文以 BW-RAID 原型系统为基础，实现了版

本子系统,该系统包括四个功能模块:(1)组内全局版本管理模块,定时更新全局逻辑版本号,并向组内各成员同步全局版本;(2)本地系统版本更新模块,接收并更新全局逻辑版本号,定时更新本地版本序列号,将全局版本与逻辑版本组合产生节点当前系统版本;(3)请求版本生成及更新模块,写请求到达节点时为其生成版本,对于已存在旧版本的逻辑块,执行版本合并;(4)版本识别模块,锁定待比较逻辑块版本,并执行版本比较;该模块包含 inline 和 offline 两个版本识别线程;inline 线程在校验节点执行冗余计算时与数据卷对应的数据节点交换版本,并进行对应逻辑块的版本识别;offline 线程对冗余计算时写入 PN 暂存的逻辑块进行后台识别。

BW-RAID 系统中引入分布式异步版本识别机制使得 RAID-set 冗余模式转换可以与应用数据更新异步。冗余模式转换过程是先更新校验码,然后同时对即将转换的逻辑块进行版本比较,若版本一致则将 DN 缓存中的数据迁移到数据存储卷;若版本不一致 PN 的数据暂存入本节点磁盘缓存中,监测到一致版本后 DN 再进行数据迁移,并释放在 PN 占用的磁盘缓存资源。

本文对配置分布式异步版本识别机制的 BW-RAID 原型系统进行了评测。测试环境包括一个客户端和三个存储节点,节点物理配置见表 1。本文在三个存储节点上搭建了只有一个 RAID-set 的简易集群,在客户端为每个数据卷配置主、从访问路径,可以分别通过 DN 和 PN 读写数据,默认主路径为活跃路径;客户端节点作为组内时钟服务器。

表 1　硬件环境配置

配置	存储服务器/客户端
CPU	Intel(R) Xeon(R) CPU E3-1230 V2 @ 3.30GHz
内存	12GB　DDR3
网卡	1 万兆网卡:Intel Corporation Device 154d/10000
盘阵	13 块 3T 盘组成的 raid5(存储节点)
交换机	万兆交换机 Brocade ICX 6450-24

3.2　数据可用性验证

数据正确性是指 BW-RAID 系统中配置分布式异步版本识别机制后,客户端写入逻辑卷的数据与从该卷读出的数据须一致,即可用性。综合 RAID-set 的状态,可用性包括三方面:(1)客户端写入到存储端的数据可用;(2)数据节点故障时数据可用;(3)节点故障恢复完成后,恢复到新节点的数据可用。

测试方法:客户端将逻辑卷格式化为 ext3 文件系统并挂载,复制 2GB 大小的文件和 1.5GB 大小的目录(包含 3 个子目录)到挂载目录;分别通过主路径(master path)和从路径(slave path)访问存储节点,计算复制文件与源文件的 md5 校验值,对比一致性,执行 diff 操作对比复制目录与源目录的一致性。通过主、从路径读 RAID-set 中数据卷的请求流向见图 5。

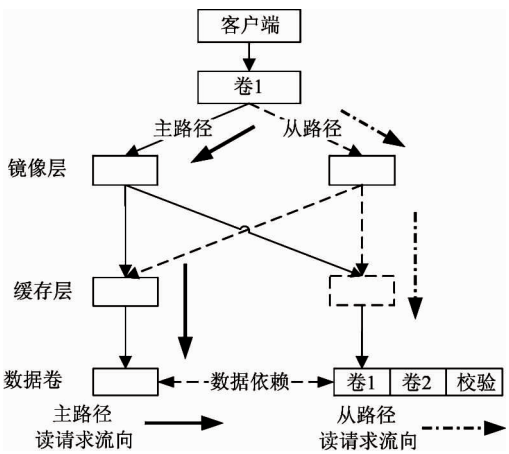

图 5　读请求流向

综合数据可用性所涉及的三方面,进行以下测试:

(1) 冗余组正常状态下,从 DV_1、DV_2 各自主路径读取数据,比较一致性;

(2) DN_1 故障,由 DV_1 卷从路径读取数据,比较一致性;

(3) DN_1 故障恢复完成后,由 DV_1 主路径读取数据,比较一致性;

(4) PN 故障恢复完成后,将 DV_2 或者 DV_1 的访问路径切换到从路径读取数据,比较一致性。

对于上述 4 种测试,复制文件、目录与源文件、目录均一致,说明异步版本机制可以保证数据可用性,进而达到保证冗余组一致性的目标。

3.3 有效性验证

有效性评价测试使用基准测试工具 FIO 进行存储压力测试和应用负载重播。存储系统参数配置见表 2。

表 2 系统参数

系统参数	数据小盘容量	360GB
	资源管理粒度	顺序写:64kB Trace:4kB
版本参数	全局版本更新间隔	30s
	本地版本更新间隔	5ms

本节评测中以 sync 模式的 BW-RAID 作为对比基准,其特点是冗余模式时阻塞写 IO 保证节点间缓存同步以解决一致性问题,具体过程是:首先阻塞客户端对逻辑块 B 所在条带的写更新,并等待该条带上所有写更新完成;然后计算、更新 RAID4 校验码,并进行数据迁移。

3.3.1 存储压力评测

存储压力评测,在逻辑卷上执行指定请求粒度的顺序写测试。测试分别在 sync 和 async 两种模式的 BW-RAID 中进行,评价指标为校验节点的 IO 负载开销和系统吞吐量。

(1) 测试方法:在客户端以 direct 方式向逻辑卷顺序写入 128GB 数据,每 5 秒记录一次带宽作为实时带宽。分别对 4kB、64kB、1024kB 三种请求粒度进行测试对比。sync 模式在校验存储节点不产生 IO 开销,只记录带宽;对 async 模式,同时记录校验节点冗余模式转换时写本地磁盘缓存的 IO 数量,及版本比较时锁定版本导致阻塞的 IO 数量。

校验节点 IO 负载(图 6)的结果表明:(1)三种粒度的顺序写测试中版本识别率均高于 99%,最高达99.69%;(2)由于 inline 线程中版本比较不一致在 PN 产生的 IO 开销为0.31% ~0.85%;async 模式下 4kB 粒度用例中不一致版本比例最高,原因在于缓存资源管理粒度是 64kB,同一缓存块被多次更新,多次被回写,导致 DN 与 PN 缓存的数据在某些时刻存在不一致的情况。其它两个用例中每个缓存块被写一次,即只有一个版本,此时不一致版本属于漏判。冗余计算过程中 inline 线程不能识别的版本转入后台识别模式,三个用例的结果显示不一致数据对象均完成版本识别,迁移到了共享数据卷,在 PN 所占用的磁盘缓存资源也释放。由此可以得出分布式异步版本识别机制在顺序写应用中可以通过版本识别实现识别一致数据对象的目标。

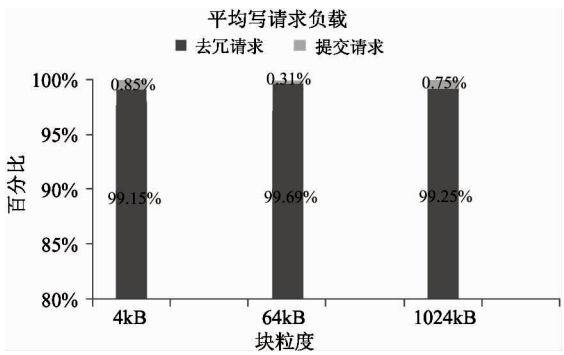

图 6 校验节点 IO 负载

(2)结果分析:

(1)4kB 粒度用例,两种模式实时带宽接近,async 模式平均带宽比 sync 模式下降 2.70%,其原因是资源管理粒度为 64kB,async 模式下存储节点最多 16 个请求竞争同一逻辑块版本执行版本合并操作,因此带宽略有降低(图 7)。

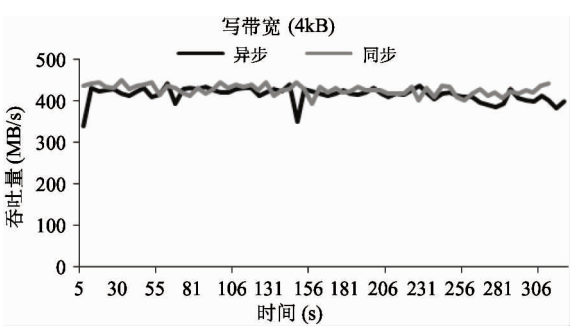

图 7 4kB 实时带宽

(2)64kB 粒度用例,async 实时带宽明显优于 sync 模式,平均带宽提升 25.43%,校验存储节点 IO 负载仅增加0.31%(图 8)。

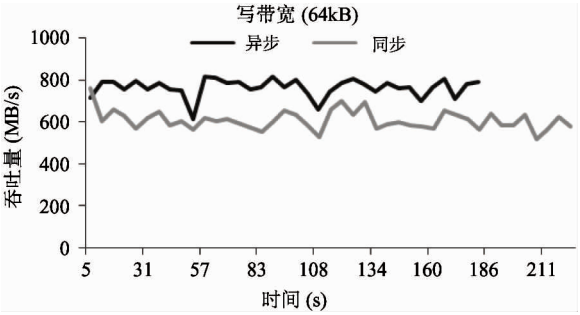

图 8　64kB 实时带宽

(3)1024kB 粒度用例,async 模式实时带宽抖动明显,峰值带宽高于 sync 模式,通过分析我们发现此时因版本比较而阻塞的请求数量明显上升,分别是 4kB 和 64kB 的 14 倍和 17 倍,阻塞请求增加导致系统吞吐量不稳定(图 9)。

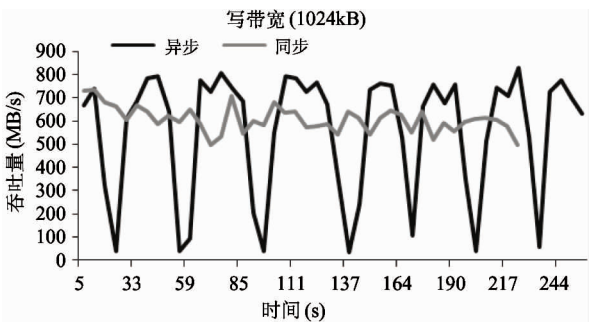

图 9　1024kB 实时带宽

顺序写平均带宽对比见表 3

表 3　顺序写对比

块粒度	平均写带宽(MB/s)		
	异步	同步	异步/同步(%)
4kB	414.6	426.1	-2.70
64kB	762.2	607.6	25.43
1024kB	614.4	610.4	0.64

3.3.2　应用负载重播评测

应用负载重播,在 async 模式下重播应用 trace 验证版本识别机制的有效性,评测指标:(1) inline 线程与 offline 线程结合是否可以识别所有版本;(2) inline 线程识别的版本比例及校验存储节点数据小盘 IO 负载。

测试方法:在 async 模式下重播 open-mail 应用中写请求比例高于 50% 的 1# - 3# trace,三个 trace 中小于 8kB 的写请求均比例均高于 90%,故在本小节中选取了 4kB 的资源管理粒度。

结果表明三个 trace 重播完成后,所有数据均完成冗余模式转换,即随着测试的进行版本均可以识别。上述三个应用中 inline 线程不能识别的版本接近 40%,那么相对于 sync 模式 PN 平均约增加了 40% 的 IO 负载(图 10)。

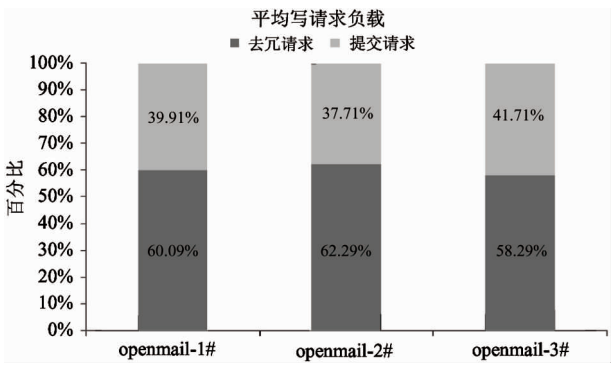

图 10　校验节点 IO 负载

根据本文版本识别理论,同一逻辑块重复被写,并且每次写间隔小于系统设置的最小可识别间隔是导致版本不可识别的主要原因。对此我们也分别分析了三个 trace 的覆盖写请求比例及对同一逻辑块覆盖写时间间隔(见表 4)。

表 4　覆盖写比例

负载	重复写逻辑块比例(%)	重复写请求比例(%)	请求间隔 <=5ms 比例(%)
1#	55.66	82.99	9.27
2#	59.02	84.36	12.62
3#	62.50	85.65	12.76

三个 trace 中均有 55% 以上的逻辑块被重复写,并且覆盖写请求比例均高于 80%。然而重复写请求间隔小于配置的可识别间隔(5ms)的比例仅在 10% 左右,评测结果与应用特点不完全相符。原因在于本文使用了基准测试工具 FIO 重播,其特点是仅按照请求原有序列模拟 IO,并不考虑原有请求实

际到达的相对时间,即忽略了应用原有请求时间属性。应用重复写比例高,请求间隔被忽略导致不可识别的版本比例升高。

4 结 论

本文针对 BW-RAID 异步冗余计算面临的冗余一致性问题,以 WACE 为研究基础提出了一种分布式节点数据版本识别方法,该方法以泛化分布式时间戳思想为基础实现了分布式镜像节点中同一逻辑块的数据对象一致性比较,其主要特点是:(1)可以转换为物理时间与逻辑时间结合的版本实现方式;(2)组内每个节点可以独立维护本节点系统版本且系统版本更新与 IO 解耦合,组内节点间版本传递开销降低,适用于 Iscsi 等通用块层传输协议;(3)可以通过比较版本确定逻辑块(或请求)所包含数据是否相同,满足 BW-RAID 冗余模式转换对数据一致性识别的要求。实验表明,该方法可以有效识别镜像关联请求,降低组内网络开销。

然而应用负载评测表明在覆盖写密集的应用中校验节点 IO 开销明显增大,同时随着系统数据容量增大维护版本产生的资源开销也随之增加,对于提高版本识别比例、降低版本维护资源开销等问题需要做进一步的研究及验证。

参考文献

[1] 那文武, 柯剑, 朱旭东等. BW-netRAID: 一种后端集中冗余管理的网络 RAID 系统. 计算机学报, 2011, 34(5): 912-923

[2] Noble M. EMC Isilon OneFS-A Technical Overview . http://www. emc. com/collateral/hardware/white-papers/: EMC Corporation. 2012

[3] Welch B, Unangst M, Abbasi Z, et al. Scalable performance of the Panasas parallel file system. In: Proceedings of the 6th USENIX Conference on File and Storage Technologies, Berkeley, USA, 2008. 17-33

[4] Fan, B, Tantisiriroj W, Lin X, et al. DiskReduce: RAID for data-intensive scalable computing. In: Proceedings of the 4th Annual Workshop on Petascale Data Storage, New York, USA, 2009. 6-10

[5] 李玮玮. BW-RAID 系统中数据版本管理的研究: [硕士学位论文]. 北京: 中国科学院计算技术研究所, 2011. 9-40

[6] Kulkarni P, Douglis F, LaVoie J, et al. Tracey. Redundancy elimination within large collections of files. In: Proceedings of the General Track: 2004 USENIX Annual Technical Conference, Boston, USA, 2004. 59-72

[7] Policroniades C, Pratt I. Alternatives for detecting redundancy in storage systems data. In: Proceedings of the General Track: 2004 USENIX Annual Technical Conference, Boston, USA, 2004. 73-86

[8] Jain N, Dahlin M, Tewari R. TAPER: Tiered approach for eliminating redundancy in replica synchronization. In: Proceedings of the 4th USENIX Conference on File and Storage Technologies, San Francisco, USA, 2005. 281-294

[9] Zhu B, Li K, Patterson H. Avoiding the disk bottleneck in the data domain deduplication file system. In: Proceedings of the 6th USENIX FAST, San Jose, USA, 2008. 69-282

[10] Koller R, Rangaswami R. I/O deduplication: utilizing content similarity to improve I/O performance. In: Proceedings of the 8th USENIX Conference on File and Storage Technologies, San Jose, USA, 2010. 211-224

[11] Ungureanu C, Atkin B, Aranya A, et al. HydraFS: a high-throughput file system for the HYDRAstor content-addressable storage system. In: Proceedings of the 8th USENIX Conference on File and Storage Technologies, San Jose, USA, 2010. 225-238

[12] Kruus E, Ungureanu C, Dubnicki C. Bimodal content defined chunking for backup streams. In: Proceedings of the 8th USENIX Conference on File and Storage Technologies, San Jose, USA, 2010. 239-252

[13] Srinivasan K, Bisson T, Goodson G, et al. iDedup: latency-aware, inline data deduplication for primary storage. In: Proceedings of the 10th USENIX Conference on File and Storage Technologies, San Jose, USA, 2012. 299-312

[14] Lillibridge M, Eshghi K, Bhagwat D. Improving restore speed for backup systems that use inline chunk-based deduplication. In: Proceedings of the 11th USENIX Conference on File and Storage Technologies, San Jose, USA, 2013. 183-198

[15] Strzelczak P, Adamczyk E, Herman-Izycka U, et al.

Concurrent deletion in a distributed content-addressable storage system with global deduplication. In: Proceedings of the 11th USENIX Conference on File and Storage Technologies, San Jose, USA, 2013. 161-174

[16] Scott C, Liu H. Sgit community book. http://gitbook.liuhui998.com: Git Community. 2011

[17] Ben C S, Brian W, Fitzpatrick C. Michael Pilato. Svnbook. http://svnbook. red-bean. com: Creative Commons, 2008

[18] Lamport L. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 1978, 21(7): 558-565

[19] DeCandia G, Hastorun D, Jampani M, et al. Dynamo: amazon's highly available key-value Store. In: Proceedings of the 21st ACM Symposium on Operating Systems Principles, Stevenson, USA, 2007. 205-220

[20] 司承祥. 集中共享存储环境中缓存集群的关键技术研究:[博士学位论文]. 北京:中国科学院计算技术所, 2011. 25-28

Distributed and asynchronous version identification mechanism for BW-RAID system

Wang Hui[*,**], Guo Mingyang[*], Dong Huanqing[*], Xu Lu[*]

([*]Institute of Computing Technology, Chinese Academy of Science, Beijing 100190)

([**]University of Chinese Academy of Sciences, Beijing 100049)

Abstract

To solve the BW-RAID system's problem of cache data version identification during its data redundant mode conversion from mirror to RAID4, a distributed and asynchronous version identification mechanism is presented. This version identification mechanism generates a new version for a logical block in a mirrored volume when its data are updated, and during the redundant mode conversion, checks whether this logic block's data cached in two mirroring notes are consistent by comparing the versions between mirroring nodes, and if consistent, makes the data move to the data volume, otherwise stores them to disk caches temporarily to guarantee system redundant consistency. It was proved by experiments that the mechanism can identify consistent data for all blocks in any state, including normal, degraded, and recovery states. The sequential writing tests showed it improved the average bandwidth up to 25.43% with the storage overhead less than 1% compared to cache synchronization mode. The open-mail workloads replay tests showed, the storage loads were less than 40%, and the blocks updated finished redundant conversion as each workload replay ended. The proposed mechanism is essential to guaranteeing redundant consistency while improving storage space utilization in BW-RAID.

Key words: network RAID, data consistency, distributed and asynchronous version, time-interval-based timestamp, version identification