ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.18, No.6, June 2007, pp.1456−1467
DOI: 10.1360/jos181456

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种基于有限编码的多副本分簇管理方法*

周 婧[1,2+], 王意洁[1,2], 李思昆[1]

[1](国防科学技术大学 计算机学院,湖南 长沙 410073)

[2](并行与分布处理国家重点实验室(国防科技大学),湖南 长沙 410073)

A Multi-Replica Clustering Management Method Based on Limited-Coding

ZHOU Jing[1,2+], WANG Yi-Jie[1,2], LI Si-Kun[1]

[1](School of Computer, National University of Defense Technology, Changsha 410073, China)

[2](National Key Laboratory for Parallel and Distributed Processing (National University of Defense Technology), Changsha 410073, China)

+ Corresponding author: E-mail: jingle77@126.com

Zhou J, Wang YJ, Li SK. A multi-replica clustering management method based on limited-coding. *Journal of Software*, 2007,18(6):1456−1467. http://www.jos.org.cn/1000-9825/18/1456.htm

Abstract: In this paper, according to the resource management problems brought by a large number of replicas, a multi-replica clustering management method based on limited-coding is proposed. In this method, according to the process of creating new replicas from existent single replica, replicas are partitioned into different hierarchies and clusters. Then replicas are coded and managed based on the user-defined limited-coding rule consisting of replica hierarchy and replica sequence, which can also dispose the alteration of clusters caused by dynamic adjustments on replicas (replica addition or replica removal) effectively. After that, a management model of centralization in local and peer to peer in wide area is adopted to organize replicas, and the cost of reconciling consistency can be greatly depressed combining with defined minimal-time of update propagation. The relevance between the coding rule and the number of replicas, and the solutions to replica failure and replica recover are discussed. The results of the performance evaluation show that the clustering method is an efficient way to manage a large number of replicas, achieving good scalability, not sensitive to moderate node failure, and adapting well to applications with frequent updates.

Key words: data replication; peer-to-peer distributed storage system; clustering; data consistency

摘 要: 针对大量数据副本所带来的资源管理问题,提出一种基于有限编码的多副本分簇管理方法.在该方法中,根据单副本复制产生新副本的过程对副本分级和分簇,通过定义“副本级别+副本顺序”的编码规则对划分后的副本进行编码和组织,并依据编码规则对由于副本的动态调整(增加或撤消)而引起的簇的动态变化进行有效管理.通过该方法,在大量副本之间建立局域集中、广域对等的管理模式,再结合定义的“最小更新传播时间”可以降低大量副

* Supported by the National Natural Science Foundation of China under Grant No.69903011 (国家自然科学基金); the National Basic Research Program of China under Grant No.2002CB312105 (国家重点基础研究发展计划(973)); the Foundation for the Author of National Excellent Doctoral Dissertation of PR China Under Grant No.200141 (高等学校全国优秀博士学位论文作者专项资金)

Received 2006-01-06; Accepted 2006-06-06

本的一致性维护开销.讨论了方法中编码规则与副本规模之间的关系,以及副本失效和恢复时的解决方法.性能测试结果表明,该方法能够有效组织大规模的数据副本,具有较好的可扩展性,对适度的结点失效不敏感,适合更新频繁的应用.

关键词: 数据复制;P2P 分布存储系统;分簇;数据一致性
中图法分类号: TP311 文献标识码: A

数据复制技术是通过合理放置数据对象的副本,并根据数据访问模式的变化动态地调整副本放置方案,从而提高分布式系统的性能、可靠性及可用性.P2P 分布存储系统对数据复制进行了广泛的研究[1−3],先前的研究重点集中在大规模分布式系统的信息分布(副本放置)[4,5]和信息发现(副本定位)[6,7]上,对大规模分布式系统中信息共享所带来的问题关注较少.随着全球数据资源共享不断呈现出爆炸性增长的趋势,分布式系统中的数据副本数量也在不断地增长,副本的管理面临着挑战.

系统中存在大量的数据副本会增加更新传播延迟,导致副本负载不平衡.现在的分布式系统一般设计为多主本(multi-master)系统,即允许任意副本在任何时刻都可以发布更新,然后在后台交换更新.Gray[8]认为,多主本复制算法会带来 $O(N^2)$的更新冲突(N 是系统中副本的数量),更新冲突的增加会导致用户视图不一致,并且当系统支持临时更新应用时,还会增加更新撤消(undo)和重做(redo)的次数,这样会消耗大量的计算资源.当系统中的数据访问等概率产生时,这个问题是多主本系统固有的.

现有的副本管理方式主要分为主从方式[9]、层次方式[10]、P2P 方式[11]以及基于图形拓扑结构的管理方式[2]等,将零散的副本组织起来之后再通过组播[12]加速更新消息的传播,同时弱一致性[8]松弛数据一致性的要求,提高了系统包容通信失效和结点失效的能力,但是上述这些技术不能完全满足大规模数据副本的管理需要.

分布服务面临两个潜在的且相互冲突的挑战:高可用性和强数据一致性.我们的设计目标是在最大化系统可用性的同时提高数据一致性.在任何时刻,系统中的副本都是可读写的,系统能够无阻塞地创建和撤消副本,并加速更新传播,为此,提出一种基于有限编码的副本分簇管理方法(multi-replica clustering management method based on limited-coding,简称 RCLC).

在 RCLC 中,首先对副本分簇,然后结合自定义的副本编码规则进行管理,充分考虑了副本动态增加和撤消时的处理方法以及该管理方法下副本动态更新所导致的一致性维护过程.实验结果表明,该方法可以实现对大量副本的有效组织,简化副本管理和一致性维护的复杂性,确保副本的最终一致.

1 副本编码和分簇

1.1 相关概念

1.1.1 活动副本(live replica)

在大规模分布式系统中,副本一经产生之后并不是永久存在的,系统将根据数据访问模式的变化动态地调整副本放置方案,增加新的副本或者撤消某些结点上的副本.同时,当副本所在结点发生系统故障时,会导致结点暂时或长期失效.在 RCLC 中,系统内所有副本的集合表示为 S_R,方法中所考虑的副本定义为:

定义 **1**. 活动副本.数据对象在系统的不同物理位置上存在 1 个或多个具有相同逻辑标识的拷贝,这些拷贝称为该数据对象的副本;至少有 1 个用户可以访问到的副本是活动副本.副本 r 是活动副本记为 $L(r)$.

对于上述定义,有几点需要特别说明:

- $\forall r(L(r)\Leftrightarrow r\in S_R)$;
- 当出现网络划分时,副本不能被全部用户访问,但是总有用户可以访问到副本,因此,该状态下的副本仍然是活动副本;
- 结点失效分为临时性(temporary)失效和永久性(permanent)失效两种.但在失效发生时刻并没有有效的机制可以立即做出判断,因此一旦结点失效,则其上的数据副本 r'不在讨论范围之内,即 $r'\notin S_R$;

- 若结点从失效中恢复,其上的数据副本将作为新的副本重新加入副本集合;
- 若系统根据一定的策略决定删除副本 r',则 $r' \notin S_R$.

1.1.2 复制源(replication source)

目前,P2P 分布存储系统中采用的复制技术大都基于单一数据副本:当需要在结点 A 上为某个数据对象增加一个新的副本 r_k 时,总是从现有的众多副本中选择一个最佳副本 $r'_k(L(r'_k))$,然后生成副本 r'_k 的拷贝,并传送到结点 A,从而形成新的副本 r_k.下面的定义就是基于上述的单副本复制.

定义 **2**. 复制源.对于单副本复制,系统生成某个已有副本 r'_k 的拷贝,并传送到系统中的另一结点,从而形成新的副本 r_k,则 r'_k 是 r_k 的复制源,记为 $RS(r_k)=r'_k$.

副本及其复制源副本之间形成的复制关系具有下述特性:

- 自反性:$\forall r \in S_R \rightarrow RS(r)=r$,即所有的副本都可以认为是自身复制产生的;
- 反对称性:$\forall r_1(r_1 \in S_R \wedge \exists r_2(r_2 \in S_R \wedge RS(r_1)=r_2 \wedge RS(r_2)=r_1)) \rightarrow r_1=r_2$,

这是因为复制是由现有副本产生新副本的一种衍生(父子)关系,如果复制关系是对称的,则违背了事物存在的自然法则;

- 非传递性:$\forall r_1(r_1 \in S_R \wedge \exists r_2(r_2 \in S_R \wedge RS(r_1)=r_2 \wedge \exists r_3(r_3 \in S_R \wedge RS(r_2)=r_3))) \not\rightarrow RS(r_1)=r_3$,

这是因为副本之间是弱一致的.例如,副本 r_2 是通过复制副本 r_3 生成的,之后,副本 r_3 发布了新的更新,而这些更新在还没有被副本 r_2 接收到时,复制副本 r_2 生成副本 r_1,显然,副本 r_1 与 r_3 内容不同,简单地从复制源的定义上就可以判断 $RS(r_1) \neq r_3$.

1.1.3 最小更新传播时间

在多主本系统中,多个更新可能在不同副本上同时发布,然后按照任意次序传播到其他副本上.该类系统的复制算法必须确保每个副本都可以接收到所有副本发布的全部更新,并依据一定的规则对这些更新排序和应用,从而实现副本的最终一致.但是,M 个并发更新之间的全排序数量为 $O(M!)$,当副本规模增大且更新频繁时,$O(M!)$将变得很大,不可能通过彻底的检索确定最合适的更新排序.因此,为了简化并发更新的确认、排序问题,定义了最小更新传播时间.

定义 **3**. 最小更新传播时间($UPT_{\min}$).对于 $\forall r_1 \forall r_2(r_1 \in S_R \wedge r_2 \in S_R)$,理想状态(不存在网络堵塞、消息丢失等)下,副本 r_1 发布的更新传递到副本 r_2 的最小时间.

在更新发布频繁的情况下,通过合并 $UPT_{\min}$ 内局域范围发布的更新,可以减少系统中并发更新的传播数量,降低一致性维护的开销,见第 2.3 节.

1.2 簇的划分

P2P 分布系统中数据副本的数量和生存状态都不是一成不变的,RCLC 适应这种动态变化,对系统内的所有副本分簇,基本思想是:

(1) 将副本与其复制源划分到同一簇内;每个簇内的复制源被标识为该簇的簇首副本,其他副本为平凡副本,假设某个簇的簇首副本为 r,则标识该簇为 $C(r)$.

(2) 复制副本 r_i 产生新副本 r_j;如果 $C(r_i)$存在,则 r_j 加入簇 $C(r_i)$;否则,创建新的簇 $C(r_i)$,r_j 加入簇 $C(r_i)$.

(3) 系统中的活动副本被划分到嵌套的簇中;对于外层的簇而言,将其嵌套包含的簇整体作为一个活动副本;根据簇的嵌套关系确定簇的级别,最先产生的簇级别最低,被嵌套的簇级别高于外层的簇,嵌套于同一外层簇的多个簇具有相同的级别,并且是对等的.例如,副本 r_l,r_m,r_n 之间存在关系:(i) $RS(r_m)=r_n$ 和(ii) $RS(r_l)=r_m$;由(i)创建簇 $C(r_n)$,$r_m,r_n \in C(r_n)$;由(ii)创建簇 $C(r_m)$,$r_l,r_m \in C(r_m)$,则在 RCLC 中,将簇 $C(r_m)$ 作为一个活动副本,嵌套包含于簇 $C(r_n)$.上述关系可以表示为

$$\forall C(r_x)(L(r_x) \wedge \exists C(r_y)(L(r_y) \wedge r_x \in C(r_y))) \rightarrow C(r_x) \in C(r_y).$$

(4) 限制每个簇内副本的数量;当簇内副本的数量超过阈值时,簇溢出,需要另行处理,见第 2.1 节.

(5) 根据副本及其复制源之间的复制关系将副本分簇,而一旦副本划分到一定的簇中,复制关系就失去了意义,取而代之的是平凡副本与簇首之间的关系.这是因为随着系统的运行,处于簇首位置的“复制

源”可能被替换为其他副本.

簇首副本和平凡副本只是针对某个具体簇在逻辑上加以的区分,某些副本在某个簇中是平凡的,在另一簇中则可能是簇首.如图 1 所示,图中每个点表示一个副本,黑色同心圆表示的副本为簇首,灰色点表示的副本为平凡的.从整体上来看,系统中所有的副本被分为 5 个簇,标识为 a 的簇内嵌套有一个子簇 b,簇 b 的簇首对于簇 a 而言只是平凡副本.

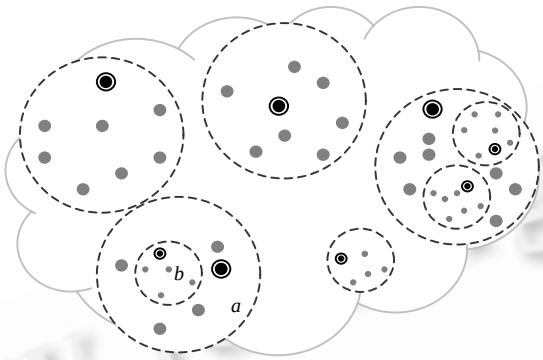

Fig.1 An illustration of replicas clustering

图 1 副本分簇示意图

副本分簇之后,采用一种局域集中、广域对等相结合的方法组织副本并解决更新冲突,使其在可扩展性、负载均衡和性能等方面均有获益.

为了便于描述,这里约定:$R_h(r)$表示副本 r 是某个簇的簇首;$R_c(r)$表示 r 在某个簇中是平凡的;$C(r,r')$表示副本 r 和 r'属于同一个簇,且 r 是该簇的簇首.

引入函数 f 计算 $C(r)$簇中平凡副本的数量 $num_{R_c}(C(r))$:

$$f(r' \mid r' \in S_R)=\begin{cases}1, & L(r')\wedge C(r,r')\\ 0, & (r'=r)\vee(L(r')\wedge\overline{C(r,r')})\end{cases} \tag{1}$$

$$num_{R_c}(C(r))=\sum_{\forall r'\in S_R} f(r') \tag{2}$$

1.3 副本编码规则

副本编码规则是定义在二进制编码集合与副本集合之间的一种映射.通过该映射为每个副本分配副本标识,并且根据副本标识便可获知副本之间在分簇管理模式中的对应关系.

假设二进制编码集合为ς,则对于映射 $f:\varsigma\to S_R$,具有:

- f 为满射——任一活动副本都应该具有副本标识(二进制编码);
- f 为内射——任一活动副本只能具有一个副本标识,且每个二进制编码只能分配给一个活动副本;
- f 为双射——f 既是满射,又是内射.

副本标识是一个二进制字符串,在 RCLC 中由两部分组成:副本级别编码+副本顺序编码.副本级别编码是为了表示副本分簇之后所蕴含的层次关系,副本顺序编码则是为了表示平凡副本与簇首之间的继承关系以及簇内副本之间的顺序关系.图 2 给出了副本 r_1,r_2 和 r_3 的编码示例,其中第 16,17 位(深灰色标注)为副本级别编码,第 0~15 位为副本顺序编码.

假设用于标识副本级别的二进制编码有 l 位,表示单个簇内的副本顺序编码为 n 位,由于副本需继承其簇首有含义部分的编码,因此,副本顺序编码共有 $d=2^l\times n$ 位,副本最多可以被分为 2^l 级.副本顺序编码与其级别编码的对应关系为:级别为 j 的副本编码的第 $j\times n$~$((j+1)\times n-1)$位为其在本簇内的顺序编码;第 $i\times n$~$((i+1)\times n-1)$位为其第 $i(0\le i<j)$级外层簇首的簇内副本顺序编码;第$(j+1)\times n$~$(d-1)$位编码无含义,默认为全零.例如:图 2 中未用任何颜色

标注的是副本在本簇内的顺序编码;浅灰色标注的是平凡副本从簇首副本继承下来的编码;灰色标注的编码无含义.

	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
r_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
r_2	0	1	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	0
r_3	1	0	0	0	0	0	0	1	1	0	1	0	1	1	1	1	1	0

Fig.2 An illustration of replicas coding

图 2 副本编码示意图

在 RCLC 中,副本标识的分配遵守下列约定:

- 系统中每个数据对象的第 1 个副本的编码为全零的二进制字符串;
- “副本级别编码”是按照其产生顺序进行分配的,如果副本 r 所在簇的簇首级别为 k,则 r 级别为 k+1;
- “副本顺序编码”由副本所属簇的簇首进行分配;
- 簇的级别用其簇首的级别进行标识,簇首维护簇内所有副本的信息,包括副本编码、副本所在结点的物理信息、副本的状态等;
- 簇首的编码是按其在低一级簇中作为平凡副本分配得到的;
- 平凡副本在相应位上继承其簇首编码中有含义部分的编码.

仍以图 2 中的 3 个副本为例,根据图中给出的编码可分析得到以下信息:副本 r_1 的级别为$(00)_{10}$=0,副本 r_2 的级别为$(01)_{10}$=1,副本 r_3 的级别为$(10)_{10}$=2;副本 r_1 属于最外层簇,副本 r_2 编码的第 0~3 位继承副本 r_1 编码的第 0~3 位,副本 r_3 编码的第 0~7 位继承副本 r_2 编码的第 0~7 位.由以上信息可知:3 个副本之间的关系为 $r_1,r_2\in C(r_1)$, $r_2,r_3\in C(r_2)$且 $C(r_2)\in C(r_1)$;簇 $C(r_1)$的级别为 0,簇 $C(r_2)$的级别为 1;副本 r_2 的编码是在簇 $C(r_1)$中作为平凡副本分配得到的.

2 基于有限编码的副本分簇管理

2.1 增加副本

系统通过复制某个已经存在的副本来增加新的副本,由于副本标识中对副本级别进行了编码限制,同时,每个簇内的副本数量有限,因此,增加副本会导致已有簇的分裂,这种分裂甚至可能波及到多个级别,算法 1 给出 RCLC 增加副本的算法流程(假设系统在结点 A 上增加某数据对象的副本 r).

- 选择新增副本加入的簇:如果新增加副本(r)的复制源(r')已经是最高级别的副本,则新副本通过其复制源所在簇的簇首(r'')加入到簇 $C(r'')$中,这时,副本 r 与副本 r'是对等的;当副本 r 加入的簇饱和时,簇首(r_i)负责向低一级簇首(r_j)发送簇溢出(cluster-overflow)消息;依此类推,直到发现合适的簇为止.
- 确定新增副本的逻辑角色:如果副本 r 通过其真正的复制源(r')或副本 r'的簇首加入到簇中,则副本 r 作为平凡副本加入;否则作为簇首加入.这是因为增加副本 r 时,已经建立的簇管理结构中的部分簇中的副本个数已经达到了上限,如果不将这些簇进行拆分,则后续增加的副本总会重复副本 r 的操作,增加了计算和通信开销;而在结点 A 上增加副本说明在这一段时间内该结点对该数据的读访问频繁,进而写访问的频率也可能随之提高,因此,将副本 r 作为簇首处理可以缩短更新的传播深度.
- 为新增副本分配编码以及系统内副本编码的维护:簇首 r_h 在给新增加的副本 r 分配编码时,需要顺序搜索簇内的编码,直到发现没有进行分配的编码,这主要是因为副本撤消或副本失效时会导致已分配编码的无效.新增加的副本 r 复制簇首 r_h 中记录的本簇副本的位置信息以及祖先簇首(簇 $C(r_h)$所嵌套于的所有外层簇的簇首)的位置信息;同时,簇首 r_h 通知簇内的其他平凡副本新增副本 r.

另外,增加副本可能导致某些副本(S_{temp})的簇首发生改变,新簇首(r)需要向 S_{temp} 中的副本进行广播,通知该变化.当 S_{temp} 中的副本接收到广播时,相应地修改自身的编码,保持与副本 r 的一致性.另外,S_{temp} 与 S'_{temp} 中的副本需要修改本地记录的簇内副本信息,以保持与调整簇后簇内副本的一致性.

算法 **1**. RCLC 增加副本算法.

Step1. 结点 A 复制已经存在的副本,产生新的副本.

If $\exists r'(r'\in S_R)$ // 定位到活动副本 r'并将其作为复制源

Then 结点 A 复制副本 r'(假设副本 r'的级别编码为 $L_{2^l-1}L_{2^l-2}...L_0$),生成副本 r

Step2. 判断复制源 r'是否为最高级别的平凡副本.

If $L_{2^l-1}L_{2^l-2}...L_0=11\ldots1$ // r'为最高级别的副本,不能作为任何簇的簇首

Then 令 $L'_{2^l-1}L'_{2^l-2}...L'_0=L_{2^l-1}L_{2^l-2}...L_0-1$; // $L'_{2^l-1}L'_{2^l-2}...L'_0$ 记录副本 r 将要加入的簇的级别

副本 r'返回副本 $r''|C(r'',r')$的信息 // r''为副本 r'簇首的第一候选副本

Else 令 $L'_{2^l-1}L'_{2^l-2}...L'_0=L_{2^l-1}L_{2^l-2}...L_0$; // r'不是最高级别的副本,可以作为簇的簇首

副本 r'返回自身的信息

Step3. 选择副本 r 的簇首.

副本 r 接收到副本 r'返回的信息,向其中包含的副本 $r_i=r'(r_i=r'')$发送加入簇 $C(r'')$的请求;

While $(R_h(r_i)\wedge num_{R_c}(C(r_i))=2^n)$ Do // 副本 r_i是簇首且本簇内副本的数量已经饱和

$L'_{2^l-1}L'_{2^l-2}...L'_0=L'_{2^l-1}L'_{2^l-2}...L'-1$;

副本 r_i 向副本 $r_j|C(r_j,r_i)$发送簇溢出信息; // 副本 r_i 向自身的簇首副本发送簇溢出信息

$r_t=r_i;r_i=r_j$ // 递归操作,直至为副本 r 找到可以加入的簇

Step4. 副本 r 加入系统,为其分配编码.

副本 r_i 分配给副本 r 编码,并记录其相应信息;

If $(L'_{2^l-1}L'_{2^l-2}...L'_0=L_{2^l-1}L_{2^l-2}...L_0)\|(L'_{2^l-1}L'_{2^l-2}...L'_0=L_{2^l-1}L_{2^l-2}...L_0-1)$

Then $R_c(r)$ // 如果副本 r 以其真正的复制源 r'或副本 r'的簇首 r''加入到簇中,则副本 r 作为平凡副本加入系统

Else $R_h(r)$; // 否则副本 r 作为簇首副本,接下来为副本 r 分配其簇内的平凡副本

$k=\sum_{i=0}^{i=2^l-1}2^{L'_i}$; $S_{temp}=\varnothing$; // 簇 $C(r_t)$嵌套于簇 $C(r_i)$且副本的数量已经饱和

For 副本 $r_x\in\{r_{temp}|C(r_t,r_{temp})\}$(假设 r_x 的副本顺序编码为 $e_{d-1}e_{d-2}\ldots e_0$)

If $\left(\sum_{i=0}^{i=n-1}2^{e_{(k\times n+i)}}>2^{n-1}\right)$ // 按照编码从小到大的顺序,将簇 $C(r_t)$内编码大于 2^{n-1} 的副本作为簇 $C(r)$的平凡副本

Then $r_x\in\{r_{temp}|C(r,r_{temp})\}$;

$S_{temp}=S_{temp}\cup\{r_x\}$; $S'_{temp}=\{r_{temp}\,|\,C(r_t,r_{temp})\}-S_{temp}$;

2.2 撤消副本

簇首负责维护簇内其他副本的信息,并且起着承上启下的作用,簇首和平凡副本在 RCLC 副本管理中的逻辑地位不同,因此,对这两种类型副本的撤消要区别对待.算法 2 给出了 RCLC 撤消副本 r 的算法流程.

- 新簇首的选取:当被撤消的副本是簇首时,由于需要继续维护簇的管理模式,因此必须产生新的簇首,即在算法中从 S_{temp} 中选择一个副本 r_{new_h} 作为新的簇首.新簇首的选择可以根据副本的活跃程度、物理位置等因素进行选择,或者采用一些经典的选举算法,这里不展开讨论.
- 簇首信息的维护:在选择出新簇首 r_{new_h} 后,其簇首(算法中的副本 r_j)不是增加副本 r_{new_h} 的信息,而是将副本 r 的编码所对应的副本信息进行修改,这是因为副本 r_{new_h} 沿用了副本 r 的编码.
- 系统内副本编码的维护:新簇首 r_{new_h} 需要向 $S_{temp}-\{r_{new_h}\}$中的副本进行广播,但 $S_{temp}-\{r_{new_h}\}$中的副本不需要修改自身的编码.这同样是因为 r_{new_h} 沿用了 r 的编码,从而不用连带地更改所有平凡副本的编码,简化了副本管理的复杂性.$S_{temp}-\{r_{new_h}\}$中的副本同时修改本地记录的簇内副本信息.

算法 **2**. RCLC 撤消副本算法.

If $\forall r'(r' \in S_R \wedge \overline{C(r,r')})$

Then $R_c(r) \wedge \overline{R_h(r)}$; // 副本 r 不是任何簇的簇首副本,只是平凡副本

If $\exists r_i(r_i \in S_R \wedge C(r_i,r))$ // 副本 r_i 是副本 r 的簇首副本

Then 副本 r_i 删除副本 r 的相关信息 // 对于平凡副本的删除,只需其簇首副本删除相关信息

Else $r''=r$; // 对于簇首副本的删除,需要选择新的簇首

令 $S_{temp}=\{r_{temp}|r_{temp} \in S_R \wedge C(r'',r_{temp})\}$;

While $S_{temp} \neq \varnothing$ Do

从 S_{temp} 中选择一个副本 r_{new_h}; // 副本 r_{new_h} 作为新的簇首

If $\exists r_j(r_j \in S_R \wedge C(r_j,r''))$ // 副本 r_j 是副本 r''的簇首副本

Then 副本 r_j 修改副本 r''的信息为副本 r_{new_h} 的相应信息

副本 r_{new_h} 复制 r''上记录的簇管理信息;$r''=r_{new\_h}$;令 $S'_{temp}=\{r_{temp}|r_{temp} \in S_R \wedge C(r'',r_{temp})\}$

副本 r_{new_h} 向 $S_{temp}-\{r_{new_h}\}$中的副本进行广播; // 通知原 $C(r'')$内的平凡副本其簇首发生变化

$S_{temp}=S'_{temp}$; // 如果 r_{new_h} 原本是另一个簇 C'的簇首,则同样需要为簇 C'选择新的簇首

2.3 副本一致性维护

通过分簇以及编码管理,系统中大量的副本在逻辑上不再是完全分散的,系统中大量的副本被直接或间接地划分到级别为 0 的簇中.级别为 0 的簇可以看作是以系统为虚拟簇首的平凡副本,因此从系统全局来看,系统中副本数量不多、具体到每一个簇来讲,副本数量有限,从而极大地降低了副本一致性维护的开销.

基于上述分簇方法的更新一致性维护过程是:

- 副本 r 上发布的更新总是发送到副本 $r'|C(r',r)$;
- 副本 r'接收到 r 的更新请求,将预定间隔时间 $UPT_{\min}$ 之内本簇接收到的所有更新合并为一个新的更新,然后发送到副本 $r''|C(r'',r')$;以此类推,直到 r''的级别编码为 00…0;
- 级别编码为 00…0 的副本之间是对等的,采用时间戳向量方法[13]或临时历史记录(casual history)方法[3]确认更新顺序,通过反熵(anti-entropy)方法[14]进行更新交换.如果级别编码为 00…0 的副本只有一个,则更新确认变形为宿主确认方法[15];
- 副本 r 在向副本 r'发送更新的同时,获取副本 r'中副本 r 未知的更新.另外,一旦副本 r'从副本 r''处接收到新的更新,也会主动地传送给副本 r.也就是说,在更新传播的过程中,采用拉(pull)与推(push)相结合的方法,加速了更新传播速度,缩短了副本处于不一致的时间.

系统中副本数量大,更新发布密集,引入最小更新传播时间 $UPT_{\min}$ 带来的好处是:在局部副本范围内提前检测到冲突的更新,降低更新冲突率;合并互不冲突的多个更新,减少系统中流动的更新数量,从而减少一致性维护的开销.

在 $UPT_{\min}$ 时间间隔内,每个簇首会接收到簇内多个平凡副本发送的更新请求,由第 1.1 节中的定义 3 可知,在该时间内任何更新都不可能从一个副本传递到另一个副本,因此,相互冲突的更新互不为对方副本所知.发现冲突的更新则立即拒绝,并且合并局域内互不冲突的更新不会破坏副本的最终一致性.$UPT_{\min}$ 是一个经验值,也可以根据具体系统的要求加以设定.

3 讨 论

3.1 副本数量与副本编码

副本编码位数是由系统设定的,并且一旦设定就不能更改.究竟副本编码采用多少位可以满足系统对标识副本的需要,下面就这两者之间的关系进行讨论.基于第 1.3 节中的假设,当每个簇都达到饱和状态时,每个簇最

多包含 2^n+1 个副本,该编码体制中可以表示的副本数量 num_{total} 为

$$num_{total}=2^n+2^n\cdot 2^n+...+\underbrace{2^n\cdot(2^n\cdot(...))}_{2^l\text{个}2^n}=\sum_{i=1}^{2^l}(2^n)^i \tag{3}$$

随着 l 值的增加,副本簇的级别呈指数增长,增加了更新传播的深度,降低了更新传播的效率;l 值和 n 值太小,不能满足表示副本的需要,当 n 值太小时,在增加副本时还可能频繁地引起簇的分裂;当 n 值太大时,增加了簇内一致性维护的复杂性,使得簇内和簇间没有达到平衡.表 1 给出 n 和 l 取不同值时对应的 num_{total} 值.由表 1 可以看出,当 l=2,n=4 时,该编码规则可以表示的副本最大数量为 65 536,完全能够满足系统的需要.

Table 1 Relationship between coding rule and the amount of replicas

表 **1** 副本编码规则与副本数量对应表

l \ n	n=3	n=4	n=5	n=6
l=1	72	272	1 056	4 160
l=2	$\approx 2^{12}$	$\approx 2^{16}$	$\approx 2^{20}$	$\approx 2^{24}$
l=3	$\approx 2^{24}$	$\approx 2^{32}$	$\approx 2^{40}$	$\approx 2^{48}$

由上述可知,增加副本会引起簇的划分,撤消副本会引起副本级别的变化,并且在极端情况下,这种分裂或变化会波及到多个级别.在该编码规则下,副本被分为 $2^l=2^2=4$ 级,因此,极端情况下的分裂不会占用系统过多的计算资源.另外,在一致性维护过程中,更新是按照簇的级别依次传播的,过多的级别会增加更新传播延迟,按照该编码规则,更新传播过程不会降低一致性维护的性能.

3.2 结点失效

与撤消副本不同,结点失效常常是突发的、不可预料的,系统不可能像撤消副本那样,在进行预处理之后副本才丧失其功效,因此需要寻求另外的解决方法.结点失效会对更新发布和更新传播带来一定的影响,与副本被撤消相同的是,不同类型副本的失效带来的影响也是不同的,需要分别处理.

如果副本 r 不作为任何簇的簇首,当其所在的结点失效时,只会导致自身的不一致,而不会破坏系统中其他活动副本的一致性;否则,$S=\{r'|\forall r'(r'\in S_R\wedge C(r,r'))\}\neq\varnothing$,即副本 r 是某个簇的簇首,其所在结点失效会引发簇内发布的更新不能传播,同时也接收不到其他簇发布的更新.这时,必须根据一定的策略选取新的簇首 r_{new_h},算法 3 给出了新簇首 r_{new_h} 的投票选取算法.这里与撤消副本相同,r_{new_h} 仍然沿用副本 r 的编码,其余操作等同于撤消副本中的操作.另外,当失效结点级别为 0 时,还需要在系统中广播定位同级别的其他副本.

算法 **3**. RCLC 新簇首投票选取算法.

Step1. 判断簇首失效.

For $\forall r'(r'\in S)$

If 副本 r'没有接收任何副本发送的“选取新簇首”消息且副本 r'上发布了新的更新

Then 副本 r'试图将更新发送到簇首 r;

If 簇首 r 没有响应

Then 副本 r'通过周期性的心跳检测簇首 r 的存活状态;

If 簇首 r 超过一定时间阈值一直没有响应

Then 副本 r'认为簇首 r 已经失效;

副本 r'向本地记录的簇内副本信息中包含的所有副本发送“选取新簇首”的消息 $VoteHead(r',T(r'))$; // 其中 $T(r')$为副本 r'发起该消息的时钟

Step2. 簇内副本投票选取新簇首.

If 副本 $r''|r''\in S$ 接收到副本 r'发送的 *VoteHead* 消息

Then If $VoteHead(r'',T(r''))$ // 副本 r''自身也发起了“选取新簇首”

Then If $(T(r'')>T(r'))\vee((T(r'')=T(r'))\wedge(ID(r'')<ID(r')))$ // $ID(r')$表示副本 r'的编码

Then 返回 *ack*=1; 副本 r''退出新簇首的竞选;

Else 返回 ack=2;

Else If 副本 r''在此之前已经接收到其他副本发送的 *VoteHead* 消息

Then 返回 ack=0;

Else If 副本 r''同时还接收到副本 r''' 发送的 *VoteHead* 消息

Then $r_{temp} \leftarrow ((T(r''')>T(r'))\vee(T(r''')=T(r'))\wedge(ID(r''')<ID(r')))?r':r'''$;

返回副本 r_{temp} 响应值 ack=1;返回其他副本响应值 ack=2;

Else 返回 ack=1;

Step3. 新簇首的确定.

时间 T_0 后,副本 r'根据接收到的其他副本的返回消息进行判断;

If $\exists_{r''\in S}(ack(r'')=2)$ // $ack(r'')$表示副本 r''返回的响应消息

Then 副本 r'退出新簇首的竞选;

Else 计算 $v=\sum_{r''\in S}(ack(r'')=1)\Big/\sum_{r''\in S}(ack(r'')=0\vee ack(r'')=1)$;

If v≥0.5 Then 副本 r'当选为新簇首 r_{new_h};

3.3 结点恢复

结点失效在分布系统中是经常发生的,虽然这些结点可能只是暂时失效,但是在失效发生时刻,并不能有任何机制可以对暂时失效和长久失效作出明确判断.因此,一旦结点失效发生,在进行副本管理时,该结点上的副本就不再予以考虑,该副本所分配的编码会被重新分配.

如果原本放有副本 r 的结点从失效中恢复,则 r 再次成为活动副本,r 定位到某个活动副本 r',通过 r'加入到系统中.副本重新加入系统的过程为:副本 r 与副本 r'进行一致性比对,如果副本 r'上保留了所有副本 r 在失效期间未接收到的更新记录,则副本 r 接收并应用这些更新;否则,副本 r 复制副本 r',与副本 r'达成一致.副本 r 将副本 r'作为其复制源,即 $RS(r)=r'$,以下流程与增加副本相同.

结点频繁的失效和恢复会经常打破已经建立的副本管理体制,增加系统内消息传播的数量;尤其是从长时间失效中恢复的副本会引起整个数据对象的传播,占用大量网络带宽,加重网络负载.第 4.4 节中将就结点失效率对 RCLC 方法性能的影响进行测试.

4 模拟测试

OptorSim[16]是一个用于测试复制算法的模拟器.模拟的分布存储系统由一个资源代理器和多个存储结点构成.资源代理器负责接收外部数据请求,并将数据请求发送到各个存储结点.每个存储结点包括计算单元、存储单元和副本管理器,具有一定的计算能力和存储能力.副本管理器负责副本的选择和动态调整.

在我们的性能测试中,系统由 1 000 个存储结点组成,每个数据对象为 10GB,数据对象集合的大小为 150GB.假定初始状态时每个数据对象只有 1 个主拷贝,且随机分布在所有结点上.我们采用根据访问频率增加副本的复制策略,即如果某个结点对某个数据对象的访问频度超过设定的阈值,就需要在该结点增加一个该数据对象的副本;如果结点有空闲的存储空间,则不撤消任何副本;否则,撤消访问频率最低的副本.

我们的模拟性能测试涉及 5 类外部访问请求和 5 类外部更新请求,每类访问请求访问的数据量及其产生概率见表 2,每类更新请求涉及的数据量及其产生概率见表 3.

Table 2 Schedule of accesses

表 2 外部访问请求

Read access type	Total size (GB)	Probability (%)
1	10	20
2	20	20
3	30	35
4	40	20
5	50	5

Table 3 Schedule of updates

表 3 外部更新请求

Write access type	Total size (MB)	Probability (%)
1	1	20
2	10	25
3	100	30
4	1000	22
5	10000	3

4.1 UPT_{min}的获取

UPT_{min} 的设定是算法性能测试的基础,因此首先需要获取该值.我们模拟了在网络带宽分别为 1MB,10MB,100MB,1GB 和 10GB 的网络配置中,理想状态下不同类型更新的最小传播值,如图 3 所示.从模拟结果可以看出,不同类型的更新在不同的网络配置中,UPT_{min} 的值都不低于 0.1 秒.因此,在下面的模拟中设定 UPT_{min}=0.1s.

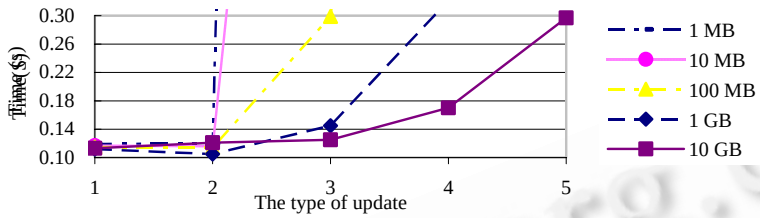

Fig.3 Minimal time of update propagation

图 3 最小更新传播时间

4.2 副本数量对算法性能的影响

模拟器在每个结点存储空间为 20GB,40GB,60GB,80GB 以及 100GB 分别运行 10 000 个外部访问请求,假设在运行过程中,没有结点失效和网络划分.每个数据对象的副本根据我们的方法被分簇,并按照 l=2,n=4 的编码规则获得一个编码.运行结束时,所有数据对象的最大副本数量和最小副本数量分布如图 4 所示.

通过上述的模拟可以看出,当结点存储容量较大时,副本数量最多可以达到 900 个,这主要是因为我们采用的复制策略与结点存储容量密切相关.考察针对不同副本数量 RCLC 的副本一致性维护开销,并与各个副本自治(uniform)管理以及将副本组织为树形(tree)和网状(mesh)拓扑结构等几种方法进行对比,如图 5 所示.模拟器运行结束条件为完成 5 000 个更新的传播,所有副本最终一致.Uniform,Tree 和 Mesh 方法在进行一致性维护时,通过反熵进行更新交换传播,且两个副本之间进行反熵的频率为更新发生频率的 1/2.模拟结果表明:当副本数量较少时,Uniform 方法更新传播开销最小;当副本数量较多时,RCLC 性能最佳,且当副本数量达到一定的数量时,更新传播时间只有微弱的变化;Tree 方法和 Mesh 方法性能较差.

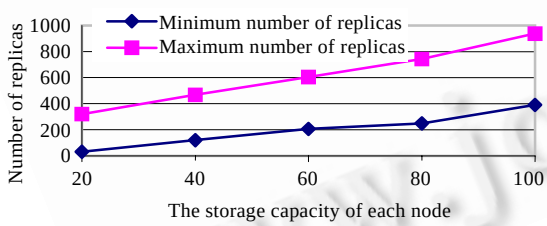

Fig.4 Distribution of the number of replicas

图 4 副本数量分布

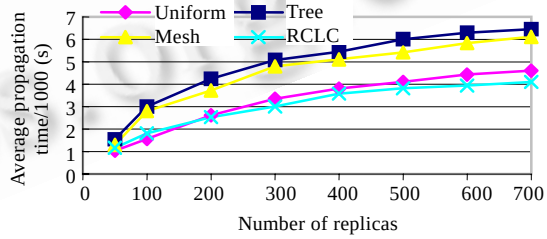

Fig.5 Effects of the number of replicas on the performance of different methods

图 5 副本数量对算法性能的影响

4.3 更新频率对算法性能的影响

当副本数量为 500 时,更新频率的变化对算法性能的影响如图 6 所示.运行结束条件同上,设定副本结点之间进行反熵的频率为 1.在更新总量明确的前提下,更新频率越高,副本每次交换更新时交换的更新数量相对就越多,一致性维护过程加速.RCLC 合并 UPT_{min} 时间内的小规模更新,大规模的副本进行分簇后,使得在广域范围内进行反熵的副本结点对的数量并不多,所以,更新频率越高其优越性越明显.对于 Tree 方法,由于每个副本发布的更新必须首先传送给根结点上的副本,因此,当更新频率很高时,根结点由于在短时间内接收到大量的更新

而性能下降,继而导致整体性能变差.

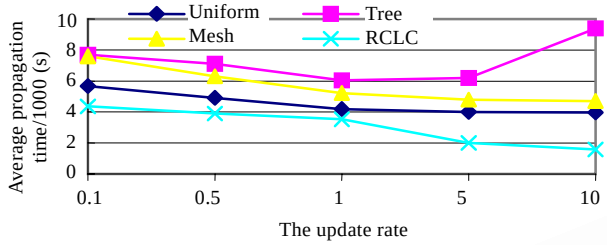

Fig.6 Effects of the update rate on the performance of different methods

图 6 更新频率对算法性能的影响

4.4 结点失效对算法性能的影响

当副本数量为 500 时,结点失效率对算法性能的影响如图 7 所示.模拟器运行结束条件同上,两个簇之间进行反熵的频率为更新发生频率的 1/2.RCLC 详细考虑了结点失效时的处理方法,低结点失效率对算法的性能影响不大,但是高失效率引起簇内和簇间结点的频繁调整,算法性能急剧降低.

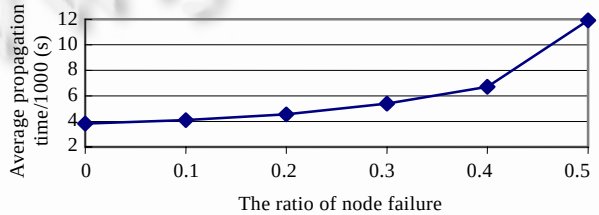

Fig.7 Influence of nodes failure on the performance

图 7 结点失效对算法性能的影响

5 结 论

针对大规模分布式环境中存在大量数据副本的现状,根据数据复制的本质,RCLC 采用分簇的方法组织副本,并通过自定义的编码规则对动态变化的副本进行管理;充分考虑了副本的动态添加、动态删除和一致性维护过程,并对编码规则与副本规模之间的关联、副本失效及其恢复进行了讨论.RCLC 实现了 3 个目标:① 支持大量的数据副本,系统可用性最大化;② 在某些结点不可用时支持副本的动态添加和撤消;③ 在结点失效和突然退出的情况下确保副本的最终一致.模拟实验结果也表明,该方法具有较好的可扩展性,对适度的结点失效不敏感,适合更新频繁的应用.

References:

[1] Dahlin M, Gao L, Nayate A, Venkataramani A, Yalagandula P, Zheng J. PRACTI replication for large-scale systems. Technical Report, TR-04-28, Austin: University of Texas at Austin, 2004.

[2] Saito Y, Karamanoli C, Karlsson M, Mahalingam M. Taming aggressive replication in the pangaea wide-area file system. In: Proc. of the 5th Symp. on Operating Systems Design and Implementation. New York: ACM Press, 2002. 15–30. http://portal.acm.org/citation.cfm?id=844131

[3] Kang BBH. S2D2: A framework for scalable and secure optimistic replication [Ph.D. Thesis]. Berkeley: University of California, 2004.

[4] van Renesse R, Schneider FB. Chain replication for supporting high throughout and availability. In: Proc. of the 6th Symp. on Operating Systems Design & Implementation (OSDI 2004). 2004. http://www.cs.cornell.edu/fbs/publications/ ChainReplicOSDI.pdf

[5] Ranganathan K, Iamnitchi A, Foster I. Improving data availability through dynamic model-driven replication in large peer-to-peer communities. In: Proc. of the 2nd IEEE/ACM Int'l Symp. on Cluster Computing and the Grid (CCGRID). Washington: IEEE Computer Society, 2002. 376−381. http://citeseer.ist.psu.edu/ranganathan02improving.html

[6] Hildrum K, Kubiatowicz JD, Rao S, Zhao BY. Distributed object location in a dynamic network. In: Proc. of the 14th Annual ACM Symp. on Parallel Algorithms and Architectures. New York: ACM Press, 2004. 41−52. http://portal.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=564877

[7] Zhao BY, Huang L, Stribling J, Rhea SC, Joseph AD. Tapestry: A resilient global-scale overlay for service deployment. IEEE Journal on Selected Areas in Communications, 2004,22(1):41−53.

[8] Gray J, Helland P, O'Neil PE, Shasha D. The dangers of replication and a solution. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Montreal, 1996. 173−182. http://portal.acm.org/citation.cfm?id=233330&coll=portal&dl=ACM

[9] Kistler J, Satyanarayanan M. Disconnected operation in the coda file system. ACM Trans. on Computer Systems, 1992,10(1):3−25.

[10] Kubiatowicz J, Bindel D, Chen Y, Czerwinski S, Eaton P, Geels D, Gummadi R, Rhea S, Weatherspoon H, Weimer W, Wells C, Zhao B. OceanStore: An architecture for global-scale persistent storage. ACM SIGARCH Computer Architecture News, 2000,28(5):190−201.

[11] Clarke I, Sandberg O, Wiley B, Hong TW. Freenet: A distributed anonymous information storage and retrieval system. In: Proc. of the ICSI Workshop on Design Issues in Anonymity and Unobservability. Berlin: Springer-Verlag, 2000. 25−31. http://citeseer.ist.psu.edu/clarke00freenet.html

[12] Golding RA, Long EDD. Design choices for weak-consistency group communication. Technical Report, UCSC-CRL-92-45, Santa Cruz: University of California, 1992.

[13] Golding RA. Modeling replica divergence in a weak-consistency protocol for global-scale distributed data bases. Technical Report, UCSC-CRL-93-09, Santa Cruz: University of California, 1993.

[14] Petersen K, Spreitzer MJ, Terry DB. Flexible update propagation for weakly consistent replication. In: Proc. of the 16th ACM Symp. on Operating Systems Principles. New York: ACM Press, 1997. 288−301. http://portal.acm.org/citation.cfm?id=266711

[15] Terry DB, Theimer MM, Petersen K. Managing update conflicts in Bayou, a weakly connected replicated storage system. In: Proc. of the 15th ACM Symp. on Operating Systems Principles. New York: ACM Press, 1995. 172−183. http://portal.acm.org/citation.cfm?id=224057.224070&coll=&dl=ACM&CFID=15151515&CFTOKEN=6184618

[16] Bell WH, Cameron DG, Capozza L, Millar P, Stockinger K, Zini F. Optorsim: A grid simulator for studying dynamic data replication strategies. Int'l Journal of High Performance Computing Applications, 2003,17(4):403−416.

周婧(1977－),女,江苏徐州人,博士生,主要研究领域为网络计算,虚拟现实.

王意洁(1971－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为网络计算,数据库技术,移动计算.

李思昆(1941－),男,教授,博士生导师,CCF 高级会员,主要研究领域为虚拟现实与可视化,嵌入式系统与 SoC 设计方法.