基于哈希树的分布式目录同步方法

鲍爱华[1]， 袁晓萍[1]， 陈 锋[2,3]， 刘 鹏[1]
(1. 解放军理工大学 指挥信息系统学院，江苏 南京 210007；2. 第二军医大学 网络信息中心，上海 200433；
3. 国防科技大学 信息系统与管理学院，湖南 长沙 410073)

摘 要：在云存储应用中，用户通常需要在多个终端上对其工作目录副本进行修改，如何在分布式目录副本之间进行高效的数据同步是一个重要问题。设计实现了一个面向私有云存储的分布式目录同步系统 HTD2Sync。系统以文件哈希值为依据进行并发同步冲突检测，能够在文件同步过程中过滤大量伪冲突；针对目录副本同步中的 2 种冲突类型和 6 种冲突场景，给出了对应的冲突消解方法；引入有序哈希树对用户目录副本的内容和结构进行建模，给出了有序哈希树的创建和更新方法。提出了一种基于有序哈希树的分布式目录副本同步方法，对其流程和核心操作步骤进行了说明，给出了有序哈希树比对算法 COMPARE_OHT。实验表明，HTD2Sync 系统能够快速感知私有云存储终端的文件变化，在云端文件变化感知方面也具有较高的效率。

关键词：哈希树；目录同步；冲突检测；最终一致性
中图分类号：TP393 **文献标识码**：A **DOI**：10.3969/j.issn.1009-3443.2013.04.080

Distributed directories synchronization method based on Hash tree

BAO Aihua[1]， *YUAN Xiaoping*[1]， *CHEN Feng*[2,3]， *LIU Peng*[1]
(1. College of Command Information System, PLA Univ. of Sci. &Tech., Nanjing 210007, China;
2. Network Information Center, The Second Military Medical University, Shanghai 200433, China;
3. School of Information System and Management, National University of Defense Technology, Changsha 410073, China)

Abstract: In cloud storage application scenarios, users usually need to modify their working directory copies in different terminals, and then it becomes an important issue for efficient data synchronization between distributed replications of the working directory. Optimistic replication and the eventual consistency theory was used as reference, a distributed working directory synchronization system named HTD2Sync was designed and implemented. In HTD2Sync, file hash value was imported as foundation to detect conflicts in concurrent file synchronization, so that most of the pseudo-conflicts were filtered; two conflict types and six conflict scenarios in directory replication synchronization were analyzed, and the corresponding resolution methods were proposed; Ordered Hash Tree (OHT) was used to represent the character of user working directory replication, and the create and update methods of OHT were introduced; the distributed directory synchronization method based on OHT was proposed, and meanwhile, its progress and core operation steps were elaborated, and algorithm to compare OHT named COMPARE_OHT was also proposed. Experiments show that HTD2Sync can detect client file changes quickly, and also has a good performance in detecting cloud file changes.

Key words: Hash tree; directory synchronization; conflict detection; eventual consistency

收稿日期：2013-04-08
基金项目：江苏省自然科学基金资助项目(BK2010131)
作者简介：鲍爱华，博士，讲师，主要研究云计算、云存储、语文 Web 服务等，nudtbaoah@gmail.com
通信作者：鲍爱华

?1994-2018 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

随着云计算概念的普及，云存储作为其重要的分支，近年来得到了广泛的关注。目前已经出现了许多面向个人的公共云存储服务，如 DropBox[1]、SkyDrive 和金山快盘等。但出于对数据安全的顾虑，持有核心数据的组织（如创新型企业、军队等）往往难以使用公共云存储服务。构建自主管理数据的私有云存储则成为更为合理的选择。虽然私有云主要面向组织内部成员，但仍然面临用户多终端办公和多用户协同办公的工作场景，私有云应当在多个终端间维持用户数据副本的一致性，使用户能够在多个终端间漫游使用并同步云端数据。在这种情况下，如何对分布在不同终端上的文件进行同步和一致性维护，成为构建私有云面临的重要问题。

私有云中的分布式目录同步存在许多挑战。首先，分布式目录同步不仅要达到不同副本的一致性，更重要的是正确识别用户的最新数据，实施合理的变化，避免用户所需的数据被覆盖，这在多用户协同工作场景下尤为重要；其次，私有云用户终端环境复杂，随时面临着系统宕机、网络故障或人为破坏问题。此时传统单机模式下的锁机制已难以在分布式数据并发同步中发挥作用，需要采用新的同步机制。另外，文件同步还需要尽可能少地占用终端资源和网络带宽，避免影响用户现有业务。

针对上述问题，本文设计实现了一个面向私有云存储的分布式文件同步系统 HTD2Sync，该系统采用哈希树快速比较云端与终端目录中的文件变化，以文件哈希值为依据进行并发同步冲突检测，通过冲突消解实现目录副本分布式一致性。HTD2Sync 系统允许用户在不同终端上对文件进行并发操作，在同步过程中以保留用户最新变化操作、实现数据最终一致性[2]为目标，以乐观复制[3]思想为基础，允许在不同终端上存在不同副本，并在冲突消解中尽可能多地保留多个副本，从而防止用户数据丢失，提高多用户协作的可靠性。

1　分布式目录同步冲突检测与消解方法

1.1　分布式同步文件副本冲突检测

分布式环境下的数据副本同步方法与系统的控制结构直接相关，目前常见的控制结构包括 Master-Slave 结构、Client-Server 结构和 P2P 对等结构[4]。Master-Slave 结构对于写操作实施强一致性，数据写操作需要由 Master 节点协调存储该数据的 Slave 节点实施，比较具有代表性的有 Google GFS[5]、Hadoop Zookeeper[6]等；Client-Server 结构由 Client 节点对其存储的数据进行更新，一般以实现副本最终一致性为目标，允许同步过程中存在不一致的情况；P2P 对等结构中，虽然节点自身也能更新数据，但由于总控节点存在，分布式副本的同步由节点之间广播同步，最终达成数据一致性。

分布式目录副本同步系统 HTD2Sync 采用 Client-Server 结构，其服务器职能由私有云总控节点承担，客户端职能则集成在私有云客户端中。图 1 给出了 HTD2Sync 的拓扑结构，可以看出，任何客户端都可以通过下行同步的方式从中心服务器（集群）获取用户数据副本，并独立为用户提供数据读取和更新操作；客户端通过上行同步的方式将本地的文件变化同步至中心服务器，其他终端则通过下行同步获取用户文件副本的变化，并依据变化发生顺序确定本地副本修改方式，进而保持分布式文件副本的一致性。由于每个客户端节点都可以对其持有的副本进行独立操作，因而在进行上行同步时必然会出现冲突。能否准确检测这些冲突并进行消解处理，是分布式文件目录同步中最为关键的问题。

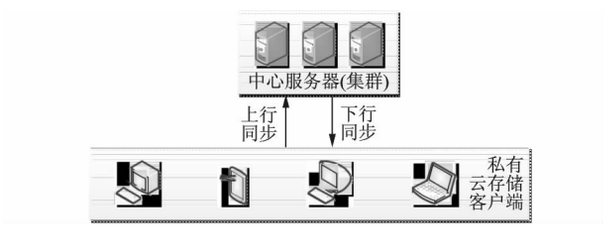

图 1　HTD2Sync 系统拓扑结构

Fig. 1　HTD2Sync system topology

定义 1（文件）　文件是 HTD2Sync 系统中的同步对象，分为中心节点文件（Master file，MF）和客户端节点文件（Client file，CF）2 种类型。形式上，$MF=\langle p,h_c\rangle$，$CF=\langle p,h_c,h_s\rangle$。其中，$p$ 代表文件相对路径，h_c 代表当前文件内容哈希值，h_s 代表客户端该文件最近一次同步时文件内容哈希值。特别地，如果文件为目录，则其哈希值为文件相对路径的哈希值；若文件（目录）不存在，则其哈希值为 ϕ。

在上述定义中，CF 的 h_s 记录了客户端最近一次成功同步时文件的哈希值，如果在后续同步过程中发现 CF 的 h_c 与 h_s 不同，则说明在上次同步完成后客户端对该文件进行过修改；如果 CF 的 h_s 值与

?1994-2018 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

中心节点 MF 的 h_c 不等,则说明客户端在上次成功同步后,存在其他客户端对该文件进行了修改,并将修改操作同步到服务器上。因此,h_s 实际上相当于一个衡量标准,用于在同步过程中确定客户端或中心节点的文件变化。

在一般的文件系统中,文件变化操作类型较多,如 CreateFile、CreateDirectory、WriteFile、DeleteFile、Rename、RemoveDirectory 等等,但在 HTD2Sync 系统中,将文件变化操作抽象为 Create、Delete 和 Modify 3 种类型。其中,Create 操作是指创建新文件或目录,Delete 指删除文件或目录,而 Modify 则泛指对现有文件内容进行修改的操作。对于较为复杂的文件操作,通过这 3 个变化操作的组合来表示。由于 HTD2Sync 系统中最为关注的是文件自身数据的同步,因此在分析文件变化时忽略文件属性的变化。

从客户端文件 CF 的角度来看,记 cf 为相对路径为 p 的文件,那么在初始状态下 $\mathrm{cf}=\langle p,\phi,\phi\rangle$。在用户创建该文件后,假设创建后文件的哈希值为 h_1,则 $\mathrm{cf}=\langle p,h_1,\phi\rangle$;一旦客户端成功地将该文件同步到中心节点,则 $\mathrm{cf}=\langle p,h_1,h_1\rangle$;如果此时文件 f 被删除,则 $\mathrm{cf}=\langle p,\phi,h_1\rangle$。从中心节点文件 MF 的角度看,记 mf 为相对路径为 p 的文件,那么在初始状态下 $\mathrm{mf}=\langle p,\phi\rangle$,一旦有客户端节点通过上行同步创建该文件后,假设创建后的文件哈希值为 h_1,则 $\mathrm{mf}=\langle p,h_1\rangle$;如果该文件再次在上行同步操作中被删除,则 $\mathrm{mf}=\langle p,\phi\rangle$。

定义 2(上行同步冲突) 在上行同步过程中,假设相对路径为 p 对应的客户端文件 $\mathrm{cf}=\langle p,h_c,h_s\rangle$、中心节点文件 $\mathrm{mf}=\langle p,h_c\rangle$,那么该文件存在上行同步冲突,如果 $\mathrm{cf},h_s\neq\mathrm{mf},h_c\ \wedge\ \mathrm{cf},h_c\neq\mathrm{mf},h_c$。

定义 3(下行同步冲突) 在下行同步过程中,假设相对路径为 p 对应的客户端文件 $\mathrm{cf}=\langle p,h_c,h_s\rangle$、中心节点文件 $\mathrm{mf}=\langle p,h_c\rangle$,那么该文件存在下行同步冲突,如果 $\mathrm{cf},h_c\neq\mathrm{cf},h_s\ \wedge\ \mathrm{cf},h_c\neq\mathrm{mf},h_c$。

图 2 给出了上行同步冲突和下行同步冲突的示例。对于上行同步而言,由于客户端文件 CF 在成功同步到中心节点后,本地文件的 h_s 会立即更新,并与中心节点对应的哈希值保持一致,所以只要发现本地文件的 h_c 与中心节点文件的 h_c 不同,即说明在本地 2 次同步过程中,存在其他客户端对中心节点文件进行了修改,因此会出现文件上行同步冲突,否则会造成其他客户端的文件修改操作丢失。如果中心节点文件此时的 h_c 与本地文件的 h_c 相等,说明 2 个文件内容仍然相同,因此这个上行同步冲突实际上是伪冲突,在 HTD2Sync 系统中予以忽略。

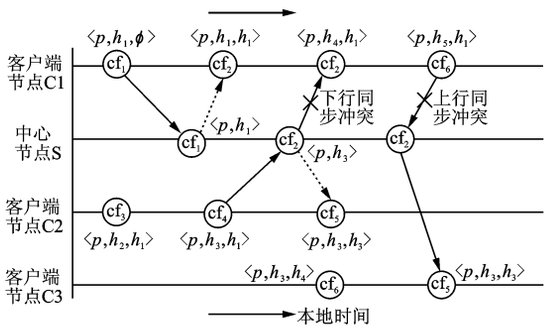

图 2 HTD2Sync 系统文件同步与冲突检测

Fig. 2 The file synchronization and conflict detection of HTD2Sync

对于下行同步而言,如果客户端文件 CF 的 h_c 和 h_s 不等,说明客户端在上次文件成功同步后对该文件进行了修改,因而在下行同步时也需要产生冲突,否则会导致客户端的文件修改操作丢失。但是,如果此时中心节点文件的 h_c 与本地文件的 h_c 仍然相等,说明本地文件修改后的内容已经与中心节点保持一致,因此这个下行同步冲突也是一个伪冲突,在 HTD2Sync 系统中予以忽略。

从上述冲突检测过程可知,虽然 HTD2Sync 系统通过记录 h_s 的方式保存历史同步信息,但它也会通过文件内容来过滤那些形式上成立但实质上可以不处理的同步冲突,从而减少冲突次数,提高分布式目录同步的效率,这是采用类版本控制的冲突检测方法难以做到的。另外,由于上述冲突检测过程没有依赖任何时间信息,因此无论私有云的多个终端以在线或离线的方式操作文件,HTD2Sync 系统都能够准确地识别可能的冲突,避免分布式目录同步过程中造成的文件变化操作遗失。

1.2 冲突消解方法

分布式同步冲突的处理方法有多种,HTD2Sync 系统进行冲突消解时,以尽可能多地保护不同终端的数据为优先准则。系统将冲突划分为并发写冲突和删除写冲突 2 种类型。所谓并发写冲突,就是指在同步发生时,客户端和中心节点分别由不同客户端节点对文件进行了写操作(包括 Create 和 Modify);所谓删除与冲突,就是指在同步发生时,客户端和中心节点中有一个节点对文件进行了删除操作(Delete),而另一节点对文件进行了写操作(包括 Create 和 Modify)。HTD2Sync 系统对文件同步冲突的处理也根据这 2 种情况进行,具体的

?1994-2018 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

冲突消解方法如表 1 所示。

表 1　HTD2Sync 系统同步冲突消解方法

Tab. 1　The synchronization conflict resolution methods of HTD2Sync

同步类型	冲突类型	场景	冲突消解方法
上行同步	并发写冲突	中心节点与客户端都发生文件写操作，即：$mf=\langle p,h_1\rangle, cf=\langle p,h_2,h_3\rangle, h_1\neq h_3, h_1\neq h_2$	将中心节点文件 mf 重命名，接受 cf 的更新操作
	删除写冲突	中心节点可能发生删除操作，客户端发生写操作，即：$mf=\langle p,\phi\rangle, cf=\langle p,h_1,h_2\rangle, h_1\neq\phi$	接受客户端写操作，在中心节点创建文件
		中心节点发生写操作，客户端发生文件删除操作，即：$mf=\langle p,h_1\rangle, cf=\langle p,\phi,h_2\rangle, h_1\neq h_2$	在中心节点上抛弃客户端删除操作，维持 mf 不变
下行同步	并发写冲突	中心节点与客户端都发生文件写操作，即：$mf=\langle p,h_1\rangle, cf=\langle p,h_2,h_3\rangle, h_1\neq h_2, h_2\neq h_3$	将客户端文件 cf 重命名，接受中心节点的文件变化
	删除写冲突	中心节点可能发生删除操作，客户端发生写操作，即：$mf=\langle p,\phi\rangle, cf=\langle p,h_1,h_2\rangle, h_1\neq\phi, h_2\neq h_3$	在客户端抛弃中心节点的删除操作，维持 cf 不变
		中心节点发生写操作，客户端发生文件删除操作，即：$mf=\langle p,h_1\rangle, cf=\langle p,\phi,h_2\rangle, h_1\neq\phi, h_2\neq\phi$	接受中心节点的写操作，在客户端创建文件

2　基于哈希树的分布式目录同步方法

2.1　有序哈希树

检测目录副本的变化有许多方法，如目录扫描比对、跟踪系统文件操作等，这些方法各有优势，应用场景也不同。本文采用一种有序哈希树(ordered Hash tree，OHT)来检测客户端和中心节点目录副本的变化。其主要优势在于不需要记录时间信息，而且可以快速获取文件变化，这在目录文件和子目录层次多的情况下尤为明显。有序哈希树是与用户客户端或中心节点目录副本相对应的树，表示该目录的特征，其节点可以用五元组〈p，h，Parent，FirstChild，NextSibling〉表示，其中，p 表示该节点对应的文件相对路径，h 表示该节点存储的 Hash 值，Parent、FirstChild、NextSibling 分别代表该节点在 OHT 中的父节点、头子孙节点和后继兄弟节点。图 3 给出了有序哈希树的示例。

有序哈希树的构建算法如表 2 所示。其中，记与目录(文件)f 对应的有序哈希树为 OHT(f)，记 OHT(f)的根节点为 RN。从 OHT 构建算法可知，对于用户文件副本而言，OHT 中对应节点的哈希值即为文件数据自身的哈希值(步骤 1)；对于目录而言，则首先分别创建其直属子目录或子文件的对应节点(步骤 2)；然后将其直属子节点的哈希值和文件名进行拼装，并将拼装结果的哈希值作为其对应节点的哈希值(步骤 3)。由于有序哈希树中子节点按照文件名进行了排序(步骤 2.2～2.4)，因此只要目录的某个子文件内容或目录结构发生变化，其对应的 OHT 节点的哈希值必然发生变化；对于用户的目录副本而言，如果 OHT 根节点 RN 的哈希值发生变化，也就意味着用户目录空间发生了文件变化。这也是 HTD2Sync 系统中基于 OHT 快速侦测文件变化的原理。

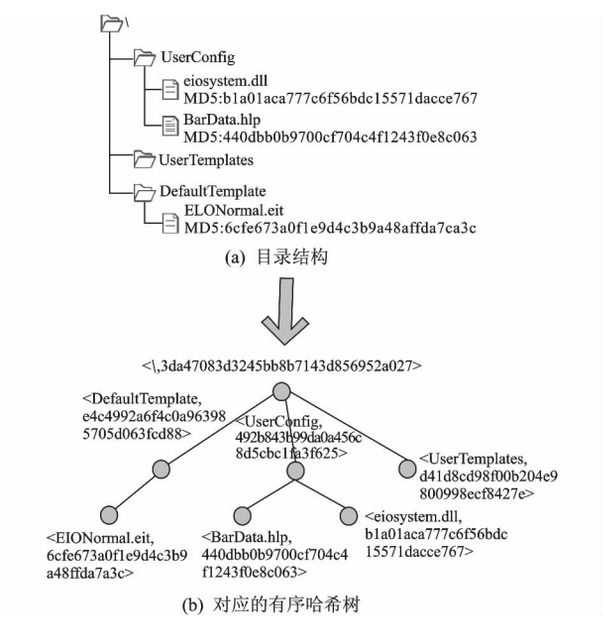

图 3　有序哈希树示例

Fig. 3　Example of OHT

由于 OHT 反映了用户目录副本的特征，因此在目录副本双向同步时，需要获取最新的 OHT 进行比对才能发现完整的文件变化。由于 OHT 的构建需要扫描完整目录空间，并计算文件的哈希值，因此在同步过程中实时构建 OHT 需要消耗较多的系统资源，这并不是合理的选择。在 HTD2Sync 系统中，采用“一次构建、逐步更新”的方式来获取与当前目录副本匹配的 OHT，具体方法为：

(1)对于客户端节点而言，HTD2Sync 系统启动时，根据 BUILD_OHT 算法生成用户目录副本当前有序哈希树 oht；客户端通过 Notify 机制监听系统对目录副本的修改操作，将所有修改操作归结为

?1994-2018 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

Create、Modify 和 Delete 3 种类型,并对 oht 中的相关节点和哈希值进行更新。

表 2 有序哈希树构建算法 BUILD_OHT

Tab. 2 Algorithm of the construction of Ordered hash tree construction

算法输入:用户客户端或中心节点的文件(目录)副本 f
算法输出:有序哈希树 OHT(f)

BUILD_OHT(f)
1 令文件 f 的相对路径为 p,为文件 f 创建有序哈希树的节点 $rn=<p$, NULL, NULL, NULL, NULL>,若 f 是目录,则进入 2;否则,假设文件 f 的哈希值为 h,令 rn.$h=h$,进入 4;
2 若 f 是目录,对于其任意的直属子目录或文件 f',创建与 f' 对应的有序哈希树 OHT(f') = BUILD_OHT(f'),进行如下处理;
2.1 令节点 node=OHT(f').RN,若 rn.FirstChild = NULL,则 rn.FirstChild = node,进入 2;若 rn.FirstChild≠NULL,则记 chNode=rn.FirstChild;
2.2 设文件 f' 的相对路径为 p',若 $p'<$ chNode.p,则令 rn.FirstChild=node,node.NextSibling=chNode,进入 2;
2.3 若 chNode.NextSibling=NULL,chNode.NextSibling=node,进入 2;否则令 nextNode=chNode.NextSibling;
2.4 若 $p'<$ chNode.p,则令 chNode.NextSibling=node,node.NextSibling=nextNode;否则令 chNode=nextNode,进入 2.3。
3 对于节点 rn 的所有子节点 chNode,按顺序依次拼接 chNode.h 和 chNode.p,形成字符串 s,rn.h=Hash(s);
4 令 OHT(f).RN=rn,退出。

(2)对于中心节点而言,由于用户目录副本在初始化状态下为空,因此与该目录对应的有序哈希树 oht 为空。在客户端节点通过上行同步提交变化操作后,中心节点根据冲突消解后的变化修改中心节点的目录副本,并将变化归结为 Create、Modify 和 Delete 3 种类型;中心节点根据这些变化操作,对 oht 进行局部更新,使其与最新的目录副本保持一致。

由于 OHT 的更新采用根据变化按需局部更新的方式,因此消耗系统资源将远小于其重建过程,且涉及更新的 OHT 节点数目等同于变化文件的目录层级(如 Create('\book\sheet.xlt')操作,目录层级为 3,更新 OHT 也仅涉及文件'\'、'\book'、'\book\sheet.xlt'对应的节点),因此能够跟随文件变化操作进行实时更新。

2.2 基于 OHT 的分布式目录同步流程

HTD2Sync 系统采用 Client-Server 控制结构,其客户端节点之间并不直接交互,因此系统实现分布式目录同步需要通过客户端节点与中心节点(云端)的双向同步完成,即客户端周期性地将本地的文件变化同步到中心节点,同时分析中心节点上来自其他客户端的同步变化,并拉取到本地,从而实现与其他客户端目录副本的同步。HTD2Sync 系统目录双向同步的流程如图 4 所示。

HTD2Sync 系统的双向同步过程中的有序哈希树包括 3 种类型:

(1)客户端当前 OHT。客户端当前 OHT 存储在客户端内存中,在 HTD2Sync 客户端启动时通过 BUILD_OHT 算法进行初始化,并通过操作系统 Notify 机制获取目录副本文件变化,然后基于该变化对内存中的 OHT 持续进行局部更新,从而得到反映当前用户目录副本特征的 OHT。

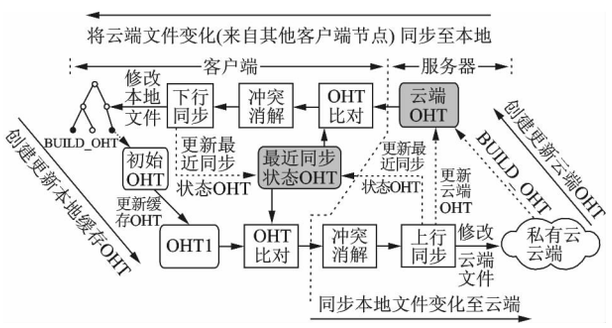

图 4 HTD2Sync 系统目录双向同步流程

Fig. 4 The process of directory bidirectional synchronization in HTD2Sync

(2)最近同步状态 OHT。最近同步状态 OHT 由客户端维护并进行持久化存储,它反映了该客户端最近一次与中心节点同步完成后的用户目录副本特征。在 HTD2Sync 系统的双向同步过程中,每当成功同步了上行或下行文件变化,都需要对该 OHT 进行更新。如果客户端发现最近同步状态 OHT 发生损坏或未初始化,则从内存中复制当前 OHT,持久化保存为最近同步状态 OHT。

(3)云端 OHT。云端 OHT 是反映用户在中心节点上对应目录副本特征的有序哈希树,由中心节点创建和更新,并进行持久化存储。在私有云用户创建并初始化时,其对应的云端 OHT 为空有序哈希树(即只包含哈希值为 ϕ 的根节点),在客户端通过上行同步对用户中心节点目录副本进行修改时,云端 OHT 同时进行局部更新,从而使其与最新的中心节点目录副本保持一致。

?1994-2018 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

HTD2Sync 系统的分布式目录副本同步流程以上述 3 类 OHT 为基础，其中，上行同步流程主要使用到客户端当前 OHT 和最近同步状态 OHT，下行同步流程主要依赖云端 OHT 和最近同步状态 OHT。最近同步状态 OHT 实际上是一个比对标准，它辅助 HTD2Sync 客户端节点发现本地和云端用户目录副本的文件变化，并进行同步处理使双方保持一致，最终实现分布式目录同步。

无论是上行同步或下行同步，有序哈希树 OHT 的比对都是一个关键步骤，它决定着 HTD2Sync 系统感知客户端与云端文件变化的速度，也影响着整个双向同步效率。客户端当前 OHT 或云端 OHT 与最近同步状态 OHT 的比对算法 COMPARE_OHT 如表 3 所示。

表 3　有序哈希树 OHT 比对算法 COMPARE_OHT

Tab. 3　Algorithm of OHT comparing

算法输入：有序哈希树 oht，客户端最新同步状态有序哈希树 oht_s

算法输出：有序哈希树 oht 和 oht_s 差异比对结果，即文件变化集合 l

COMPARE_OHT(oht, oht_s)

1　令 l 为 oht 和 oht_s 比对所得的文件变化集合，令 RN 为 oht 的根节点、RN_s 为 oht_s 的根节点，若 $RN.h = RN_s.h$，进入步骤 3；

2　令 node=RN. FirstChild，$node_s = RN_s$. FirstChild，对 node 和 $node_s$ 进行步骤 2.1～2.7 的处理：

2.1　若 node、$node_s$ 均为 NULL，进入步骤 3；

2.2　若 node= NULL ∧ $node_s \neq$ NULL，则令 subOHT 为 oht_s 中根结点为 $node_s$ 的子树，执行子过程 REMOVE_SUB_OHT(subOHT, l)，nodes=nodes. NextSibling，进入步骤 2.1；

2.2　若 node≠NULL ∧ $node_s$ = NULL，则令 subOHT 为 oht 中根节点为 node 的子树，执行子过程 ADD_SUB_OHT(subOHT, l)，node=node. NextSibling，进入步骤 2.1；

2.3　若 node. p < $node_s$. p，则令 subOHT 为 oht 中根节点为 node 的子树，执行子过程 ADD_SUB_OHT(subOHT, l)，node=node. NextSibling，进入步骤 2.1；

2.4　若 node. p> $node_s$. p，则令 subOHT 为 oht_s 中根结点为 $node_s$ 的子树，执行子过程 REMOVE_SUB_OHT(subOHT, l)，nodes=nodes. NextSibling，进入步骤 2.1；

2.5　若 node. p= $node_s$. p ∧ node. h = $node_s$. h，则进入步骤 2.7；

2.6　若 node. p= $node_s$. p ∧ node. h≠$node_s$. h，则令 subOHT 为 oht 中根节点为 node 的子树，subOHT$_s$ 为 oht_s 中根结点为 $node_s$ 的子树，执行子过程 UPDATE_SUB_OHT(subOHT, subOHT$_s$, l)，进入步骤 2.7；

2.7　令 node=node. NextSibling，nodes=nodes. NextSibling，进入步骤 2.1。

3　算法运行结束，返回 l。

ADD_SUB_OHT(subOHT, l)

A1　令 RN 为 subOHT 的根节点，记文件 $f = \langle RN.p, RN.h, \phi \rangle$，$l$. push($f$)；令节点 node=RN. FirstChild；

A2　若 node=NULL，则子过程执行完毕，退出；若 node≠NULL，则令 subOHT为 subOHT 中根结点为 node 的子树，执行子过程 ADD_SUB_OHT(subOHT, l)；

A3　令 node=node. NextSibling，重复执行步骤 A2。

REMOVE_SUB_OHT(subOHT,l)

R1　令 RN 为 subOHT 的根节点，令节点 node=RN. FirstChild；

R2　若 node=NULL，则进入步骤 R4；若 node≠NULL，则令 subOHT为 subOHT 中根结点为 node 的子树，执行子过程 REMOVE_SUB_OHT(subOHT,l)；

R3　令 node=node. NextSibling，重复执行步骤 R2；

R4　记文件 $f = \langle RN.p, \phi, RN.h \rangle$，$l$. push($f$)；子过程执行完毕。

UPDATE_SUB_OHT(subOHT, subOHT$_s$, l)

U1　令 RN 为 subOHT 的根节点、RN_s 为 subOHT$_s$ 的根节点，令 node=RN. FirstChild，$node_s = RN_s$. FirstChild，若 node、$node_s$ 均为 NULL，则记文件 $f = \langle RN.p, RN.h, RN_s.h \rangle$，$l$. push($f$)，子过程执行完毕，否则进入步骤 $U2$；

U2　令 l_s=COMPARE_OHT(subOHT, subOHTs)，l. pushAll(l_s)，子过程执行完毕。

有序哈希树 OHT 比对算法 COMPARE_OHT 由一个算法主控流程和 3 个子过程组成，其中，3 个子过程分别对应着文件的 Create、Delete 和 Modify 变化侦测过程，算法总控和子过程之间通过递归调用进行组合，共同完成 2 颗树的遍历和比对。其中较为特殊的是 ADD_SUB_OHT 和 REMOVE_SUB_OHT，前者首先为子 OHT 树的根节点生成文件变化，然后实施遍历（步骤 A1～A3）；后者则首先实施递归遍历，生成文件变化删除子节点对应的文件，最后生成文件变化删除根节点对应的文件（步骤 R1～R4）。这么处理的主要原因在于，创建文件或目录需要从根目录开始依次进行，最后创建子文件，否则会造成找不到根目录的异常；文件（目录）删除操作正好相反，需要从子文件开始删除，最后删除根目录，否则会造成非空目录删除异常。由于 HTD2Sync 系统的目录副本冲突检测以文件的形式进行分析，因此 COMPARE_OHT 算法的返回结果也是一个文件集合，表示 2 个 OHT 所代表的目录副本间哪些文件发生了变化，同时变化前后的文件哈希值也随之返回（通过哈希值 ϕ 的运用可以表示 Create 和 Delete），用以进行冲突检测。

与全盘扫描或目录副本元数据比对方式相比，

?1994-2018 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

基于OHT的目录副本比对算法COMPARE_OHT在效率上有其优势。由于OHT节点包含了对应文件(目录)的特征信息,因此通过哈希值比较就可以直接发现其自身或子节点是否发生变化,这使得算法在比对过程中能够直接分析已经变化的节点,跳过没有变化的节点(步骤2.5),从而可以提高比对速度。假设用户目录副本中目录层次为D,每个目录平均包含的子目录(文件)数目为W,如果用户目录副本中存在一个文件发生变化,那么使用算法COMPARE_OHT发现该文件变化的时间复杂度为$O(D \cdot W)$,而传统完整元数据比对的时间复杂度为$O(W^D)$。由于COMPARE_OHT算法是多项式时间复杂度,因此在用户目录副本中元数据数目增大时,该算法有着较好的适应性。

2.3 系统实现与分析

根据前面所述的分布式目录同步方法,本文设计实现了HTD2Sync原型系统,系统组成结构如图5所示。

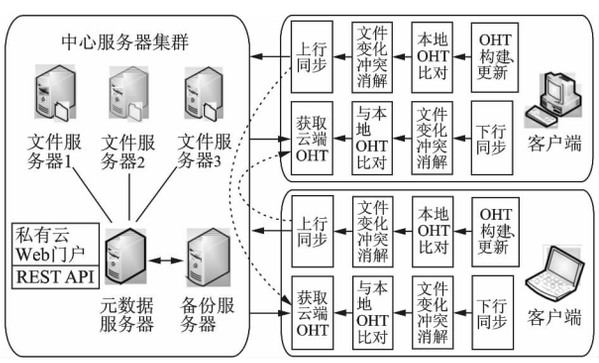

图5 HTD2Sync系统结构

Fig. 5 The structure of HTD2Sync

HTD2Sync系统采用Client-Server结构,中心服务器由元数据服务器、备份服务器和若干文件服务器组成。其中:元数据服务器负责管理用户文件元数据,在用户目录副本同步过程中维持云端目录空间OHT,并以此协调客户端的文件上行同步和下行同步,使分布在不同终端上的目录副本最终保持一致;文件服务器提供文件存储服务,客户端从元数据服务器获得文件元数据后,可以直接连接文件服务器进行文件上传和下载,实现控制流和数据流分离;备份服务器作为元数据服务器的影子节点,在元数据服务器出现故障时,自动接替元数据服务器提供服务,从而提高系统的可靠性。HTD2Sync系统客户端负责本地目录副本OHT的创建、更新,并以此为工具发现本地或云端发生的文件变化,在冲突消解后通过上行或下行同步,使本地目录副本与云端保持一致。HTD2Sync系统的部分运行界面如图6所示。

(a) 客户端同步任务

(b) HTD2Sync系统Web门户

图6 HTD2Sync系统运行界面

Fig. 6 The snapshots of HTD2Sync

为了对HTD2Sync的系统特性进行分析,本文将系统中通过OHT比对发现目录副本差异的方法更换为采用目录副本元数据完全比对发现差异的方法,形成对比测试系统CmpSync。在实验中,采用开源通信开发框架ACE作为实验对象(8 406个文件,321个目录,共计164 MB),将其复制5份(Ace1, Ace2, …, Ace5)后拷入客户端工作目录,通过以下5种工作场景测试CmpSync和HTD2Sync之间的差异:

场景A:在系统运行过程中,编辑文件\Ace1\ace\os_include\os_aio.h,统计文件变化分析时间;

场景B:在系统关闭的情况下,编辑文件\Ace1\ace\os_include\os_aio.h,然后运行系统,统计文件变化分析时间;

场景C:在系统运行过程中,将文件夹\Ace1重命名为\Ace6,统计文件变化分析时间;

场景D:在系统关闭的情况下,将文件夹\Ace2重命名为\Ace7,然后运行系统,统计文件变化分析时间;

场景E:将客户端工作目录拷贝到新的客户端中,运行系统,在新客户端中统计文件变化分析时间。

实验在100 M局域网环境下进行,客户端采用

?1994-2018 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

Intel 1.86G 双核 CPU、2GB 内存，运行 Windows XP 操作系统。主要实验数据如表 4 所示，对其中的数据进行分析可知：

(1)场景 A 和 C 中，HTD2Sync 和 CmpSync 均能够很快完成文件变化分析，且时间差异较小。主要原因在于二者在运行过程中均利用了操作系统 Notify 机制，这使得二者能够直接发现变化的文件，进而进行处理操作。

(2)场景 B 和 D 中，HTD2Sync 能够更快发现文件变化，原因在于 HTD2Sync 在生成目录空间当前 OHT 后，能够通过哈希值比较直接发现目录\Ace1 和\Ace2 的变化，而 CmpSync 则需要将当前目录元数据与历史数据进行完整比较后才能完成变化分析；场景 B 和 D 的文件变化分析速度要远慢于场景 A 和 C，主要原因在于，系统启动时需要耗费时间构建当前目录的快照(如 OHT)，并且需要与历史数据进行对比分析后才能发现文件变化。

(3)场景 E 中，HTD2Sync 文件变化分析速度要远高于 CmpSync，原因是 HTD2Sync 在创建完当前 OHT 后，仅需要比较当前 OHT 头节点的哈希值与云端 OHT 头节点哈希值，即可发现文件没有发生变化；而 CmpSync 则需要将云端元数据(约 21M)下载到本地后，才能进行完整比对，其中元数据网络下载和本地对比分析需要耗费较多的时间。

表 4　对比实验结果

Tab. 4　Results of compare experiments

ms

分析对象	文件变化分析时间					
	场景	A	B	C	D	E
HTD2Sync		2	8354	1180	10683	8371
CmpSync		1	12545	1059	11984	18927

通过以上实验分析可以看出，HTD2Sync 系统在缺乏系统通知的情况下，能够以较高的效率完成客户端文件变化分析；云端文件变化感知方面，在缺乏历史信息的条件下(如场景 E)，由于 OHT 比对可以仅分析哈希值发生变化的节点，在变化较少时可以避免大量的元数据网络传输，因此文件变化分析的效率同样较高。

3　相关工作

对于分布式目录副本同步而言，发现客户端、云端文件变化是实现数据同步的前提。目前常见的方法包括纯目录状态比对[7]和跟踪系统文件操作过程发掘文件变化[8]2 种方式。前者完全以修改前后目录的文件目录结构为素材，通过逐一比对发现文件变化操作，采用这一方式比较典型的系统有 RSync[9]和 SyncButler[9]等；后者则通过 Hook、文件过滤驱动或系统 Notify 机制等技术，直接监控系统运行期间的文件操作，进而提取出文件变化，采用这一方式比较典型的系统有 DropBox、金山快盘等。这 2 种方式各有优劣，前者不依赖任何系统运行状态信息，可以发现任意操作用例下目录的文件变化操作，但在系统持续运行过程中可能会造成系统开销过大；后者由于可以直接获取变化通知，因此在资源占用上具有优势，但如果监控自身由于各种因素暂停或失效(如 Notify 机制中的缓冲区问题)，就会造成文件变化遗失。

在分布式目录副本同步中，对文件数据冲突进行检测和处理是实现分布式一致性的关键问题。SVN 版本管理工具在面对工作副本时，以抢占方式获取并修改全局版本号，这样系统可以很方便地识别工作副本的冲突，但系统难以区分多个不同节点上的工作副本。Amazon 云计算基础设施 Dynamo[10]中能够同时保存多个不同的副本，它采用 Vector Clock 来标识每个副本，在用户读取到多个副本时，根据 Vector Clock 来判定最新的副本数据，从而实现分布式数据同步。但随着节点的增多，Vector Clock 列表也会逐渐增大，冲突检测的复杂度也逐步增大；在私有云存储系统中，用户终端的时钟无法确保统一，这也会使 Vector Clock 自身失效。文献[11]将版本管理和 Vector Clock 结合起来，提出了一种适用于云存储的分布式目录同步方法，将 Vector Clock 中的时间戳替换为全局统一的版本号，从而快捷地进行冲突检测和同步。但是，该方法中使用版本号来标识文件内容，如果客户端发生版本号信息修改或损坏的情况，就会产生大量的冲突。另外，在版本号不同但文件内容实际相同的情况下(例如人工在不同节点间拷贝数据)，也会产生不必要的冲突。

与上述工作相比，HTD2Sync 系统在感知文件变化时，将目录比对和系统通知这 2 种方式结合起来，在系统初始化时通过目录比较发现当前变化，在系统持续运行时通过系统通知的方式获取变化，确保不遗失任何用户目录中的文件变化操作，并降低系统运行时的资源占用。在进行目录比对时，HTD2Sync 系统采用有序哈希树 OHT 对目录内容和结构进行建模，在比较分析时可以直接发现发生变化的节点，从而提高目录副本差异比较的速度。同时，HTD2Sync 系统参考借鉴了乐观复制和最终一致性思想，在文件同步过程中将文件 Hash 值引

?1994-2018 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

入分布式副本冲突检测,在冲突检测过程中比较本地和云端文件的哈希值,在客户端历史同步信息缺失的情况下避免了大量伪冲突的出现,提高了分布式目录副本同步的效率。

4 结 语

针对私有云存储应用中分布式目录副本同步问题,本文设计实现了一个面向私有云存储的分布式目录副本同步系统HTD2Sync。系统支持私有云用户在不同终端上对文件进行并发操作,允许在不同终端上存在多个副本,防止用户数据丢失,提高多用户协作的可靠性。系统以文件哈希值为依据进行并发同步冲突检测,能够在文件同步过程中过滤大量伪冲突,提高冲突检测效率。在满足最终一致性的前提下,分析了目录副本同步中的2种冲突类型和6种冲突场景,给出了对应的冲突消解方法。系统引入了有序哈希树表示用户目录副本特征,给出了有序哈希树的创建和更新方法,提出了基于有序哈希树的分布式目录副本同步方法,对其流程和核心操作步骤进行了说明,并给出了有序哈希树比对算法COMPARE_OHT。实验与分析表明,HTD2Sync系统能够在缺失历史信息的情况下快速感知私有云终端的文件变化,并能够在云端文件变化感知过程中避免大量元数据的网络传输,从而保证云端文件变化感知的效率。

本文提出的分布式目录副本同步方法同样适用于其他分布式数据同步和冲突检测消解的应用场景。在下一步工作中,将进一步考虑在双向同步过程中引入增量备份的可行性,从而降低系统对网络带宽的要求,提高文件双向同步的效率。

参考文献:

[1] HENDRICKSON M. Dropbox:the online storage solution we've been waiting for[EB/OL]. (2008-03-11). [2010-03-10]. http://techcrunch.com/2008/0-3/11/dropbox-the-online-storage-solution-weve-been-waiting-for.

[2] VOGELS W. Eventually consistent[J]. Communications of the ACM -Rural Engineering Development, 2009,52(1):40-44.

[3] SAITO Y, SHAPIRO M. Optimistic replication[J]. ACM Computing Surveys, 2005,37(1):42-81.

[4] 周 婧. P2P分布存储系统中海量数据的数据一致性维护技术研究[D]. 长沙:国防科技大学, 2007.

[5] Sanjay Ghemawat, HOWARD G, LEUNG Shun-Tak. The Google file system[C]. SOSP'03 Proceedings of the Nineteenth ACM Symposium on Operating systems principles, New York: ACM, 2003.

[6] ZooKeeper: A distributed coordination service for distributed applications[EB/OL]. [2013-02-01]. http://zookeeper.apache.org/doc/t-runk/zookeeperOver.html.

[7] Navendu Jain, DAHLIN M, TEWARI R. Taper: Tiered approach of eliminating redundancy in replica synchronization[C]. Proc of FAST'05, Berkeley: USENIX Association, 2005.

[8] BALASUBRAMANIAM S, BENJAMIN C. Pierce-What is a file synchronizer[C]. Proc of MobiCom'98, New York: ACM, 1998.

[9] PIERCE B C, VOUILLON J. What is in unison(MS-CIS-03-06)[R]. Philadelphia, Pennsylvania: Department of Computer and Information Science, University of Pennsylvania, 2004.

[10] Giuseppe De Candia, Deniz Hastorun, Madan Jampani, et al. Dynamo: amazon's highly available key-value store[C]. SOSP'07 Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles, New York: ACM, 2007.

[11] 李 强,朱立谷,曾赛峰,等. 分布式目录同步的冲突处理与副本管理研究[J]. 计算机研究与发展, 2012, 49(增刊1):257-262.
LI Qiang, ZHU Ligu, ZENG Saifeng, et al. A new conflict handing and copies management algorithm for distributed directories synchronization[J]. Journal of Computer Research and Development, 2012, 49(zl): 257-262. (in Chinese).

(责任编辑:徐金龙)

?1994-2018 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net