文章编号：2095－1248(2015)04－0071－07

私有云环境下基于 HDFS 的 PDM 数据备份策略

夏秀峰[a,b]，王 涛[b]
（沈阳航空航天大学 a. 辽宁省通用航空重点实验室；b. 计算机学院，沈阳 110136）

摘要：企业“大数据”时代的到来，以及集中式存储、访问和维护大数据带来的困难，兼顾企业数据涉密性的实际需求，使得企业利用私有云来存储 PDM 数据成为必然趋势，而如何备份企业私有云中的海量数据也成为一个新的问题。以 Hadoop 分布式文件系统 HDFS 为企业数据存储平台，提出了一种集中式的数据块级增量备份策略，详细描述了备份系统的体系结构、相关概念定义及备份分组算法。该策略能够解决系统故障、人为误操作、存储介质故障等原因导致的企业 PDM 数据丢失的问题。实验结果表明，分组算法通过充分利用网络带宽，实现了数据的高效备份。

关键词：PDM 数据；企业私有云；HDFS；备份策略
中图分类号：TP311 **文献标志码**：A
doi：10.3969/j.issn.2095－1248.2013.04.12

Backup strategy for PDM data in private cloud based on HDFS

XIA Xiu-feng[a,b], WANG Tao[b]
(a. Liaoning General Aviation Key Laboratory;
b. College of Computer Science, Shenyang Aerospace University, Shenyang 110136, China)

Abstract: In the enterprise “Big Data” era, private cloud to store enterprise PDM data becomes an inevitable trend to deal with the difficulties of centralized storage, access and maintenance caused by “Big Data” and to meet the actual needs of secret characteristic of enterprise data. And how to back up large-scale data in enterprise private cloud has become a new problem. With Hadoop Distributed File System (HDFS) as the storage platform for enterprise data, a centralized block-level incremental backup strategy has been proposed, which gives the backup systematical structure, relevant definitions and grouping algorithm. The backup strategy can solve the problems of enterprise PDM data loss caused by system failures, human mistakes, storage media failure, etc. Experimental results show that the grouping algorithm can realize efficient data backup with sufficient network bandwidth.

Key words: PDM data; enterprise private cloud; HDFS; backup strategy

作为装备制造企业数据管理的核心系统，产品数据管理（Product Data Management，PDM）是管理所有与产品相关的信息和过程的软件，其主要功能包括产品文档管理、产品结构管理和工作流管理等[1]。企业“大数据”时代的到来，以及集中式存储、访问和维护大数据带来的困难，兼顾企业数据涉密性的实际需求，利用企业私有云存储 PDM 数据将是必然趋势。例如，2010 年李伯虎

收稿日期：2014－10－20
基金项目：航空科学基金（项目编号：2013ZG54032）
作者简介：夏秀峰（1964－），男，山东青岛人，教授，主要研究方向：数据库理论与技术，E-mail：xiaxiufeng@163.com；王涛（1987－），男，山东莱芜人，硕生研究生，主要研究方向：管理信息系统与数据库，E-mail：594608907@qq.com。

等人在文献[2-3]中提出了云制造的概念,其作为一种基于知识、面向服务、高效低碳的网络化智能制造新模式,强调产品全生命周期中各类制造资源的整合与高度共享。

企业私有云存储利用云计算、大数据管理等多项技术充分整合企业已有资源,可以有效解决海量数据存储访问、高并发、高扩展等问题,能有效降低存储成本,减少 PDM 系统的实施周期,提高生产效率[4]。存储在私有云中的 PDM 数据,由于人为误操作、存储介质损坏、软件错误、自然灾害等多种因素导致重要数据的丢失,会给企业带来无法弥补的损失。作为一种数据安全策略,备份是避免数据丢失的最基本方法[5]。

在传统的集中式环境下,要实现文件系统中 PDM 物理文档数据的增量备份,所花费的查询时间代价、PDM 文档签出对比的时间代价很大。因此,现在大多数大型制造企业只能采用一周一次的完全备份策略,这样的备份策略不仅时间粒度大,而且备份时间特别长。在企业私有云环境下,PDM 数据的备份问题迄今为止未见相关研究成果。

本文以 Hadoop[6-7] 分布式文件系统 HDFS (Hadoop Distributed File System)[8] 作为私有云存储平台,研究 PDM 数据备份的策略。根据 HDFS 一次写入、多次读取、不支持在已上传文件的任意位置进行修改等特点,快速准确定位到一日内更新过的文件,进而提出一种以日为粒度的增量备份方案。

1 PDM 数据的企业私有云存储

近几年,国内制造型企业经过多年的积累、产品型号数量的不断增加、“构型”机制的逐步实施,以及 MBD/MBE[9] 概念的提出和应用,使存储在 PDM 中的数据越来越多,逐步呈现出“大数据”特征,使存储与处理成本不断增加、并发性和访问速度逐步下降、备份时间越来越长。

如何解决制造型企业海量 MBD 数据的存储和共享问题,是企业由制造向智造转型的重要研究内容之一[10-12]。因此,为解决制造型企业面临的大数据存储问题,兼顾企业高度保密性,私有云存储[13]是个很好的解决方案。文献[2]中提出基于企业私有云的 PDM 系统架构可采用四层结构模型:基础设备层、存储管理层、应用接口层和访问层,如图 1 所示。

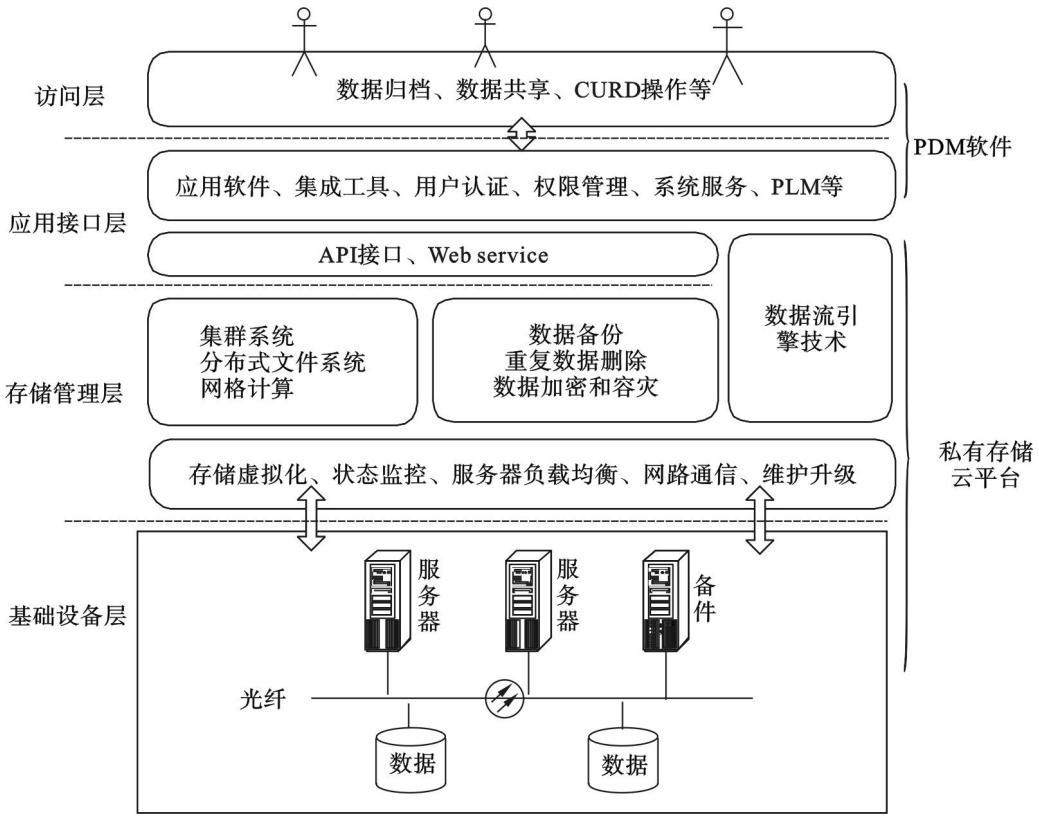

图 1 私有云存储四层结构模型

在上述基于企业私有云的 PDM 系统结构的基础上,本文以 Hadoop 分布式文件系统 HDFS 作为私有云存储平台,体系架构如图 2 所示[14]。

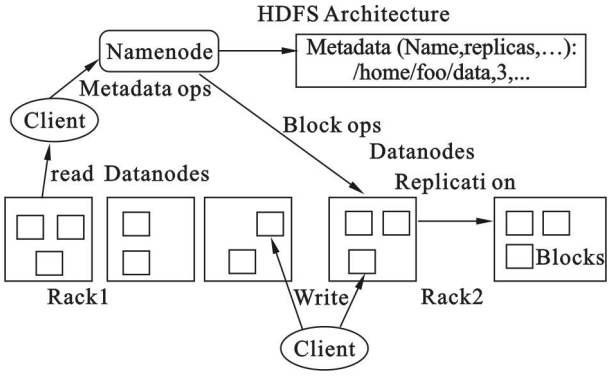

图 2　HDFS 架构图

HDFS 集群包含一个 Namenode 节点和多个 datanode 节点。客户端(c1ient)通过与 Namenode 和 Datanode 交互来访问整个文件系统。HDFS 在读取一个文件时默认的算法是采用顺序读取方式,具体过程为:客户端向 Namenode 节点申请读操作,Namenode 通过申请后会向客户端返回一个包含文件所有数据块信息的链表,客户端按链表顺序选择合适的数据节点,进行与数据节点的连接和数据块的读取。当一个数据块读取完成后,客户端关闭与该数据节点的连接,并为下一个数据库选择数据节点,直到所有数据块都读取完成。

2　私有云存储 PDM 数据增量备份方案

本文提出的以日为时间粒度(备份时间粒度是指两次备份之间的时间间隔)的备份方案采用的是以一周为周期,周一到周六进行增量备份,周日做一次全备份。而要实现以日为时间粒度的增量备份,则需要查找到当日 HDFS 中新建的文件和更新后重新上传或者修改过属性的文件。要完成文件的备份,则需要获取文件在 HDFS 中的元信息,包括:文件名称、文件最后修改时间、文件大小、文件源路径、组成文件的数据块 ID、数据块(包括副本)所在的节点信息。

同时,我们把上述元信息中的文件名称、文件大小、最后修改时间、文件源路径作为文件的备份元数据,并把这些元数据信息以数据库的方式进行组织,以便恢复时快速查询。对于 HDFS 而言,只要能够备份出系统的文件目录树和具体的文件,配合文件的备份元数据信息就可以实现数据的恢复。

2.1　体系结构

目前,常用的集中式备份系统体系结构有:Host – based、LAN – Based 和基于存储区域网 SAN(Storage area network)的 LAN – Free 等多种结构[15]。Host – Based 是传统的数据备份结构,已经不适合现在大型企业的数据备份要求;LAN – Based 备份结构的优点是节省投资、磁带库共享、集中备份管理,而缺点在于对网络传输压力大。LAN – Free 备份结构彻底解决了 LAN – Based 方式占用网络带宽的问题,但是其成本高、难度大、恢复能力弱,且只能针对于 SAN 这一特定的存储架构。

考虑到 LAN – Free 的缺点,并且由于实行冷备份,网络带宽资源全部用于备份,LAN – Based 方式中网络传输压力大的问题也将不存在,因此采用 LAN – Based 的集中式备份体系架构。其体系架构如图 3 所示。

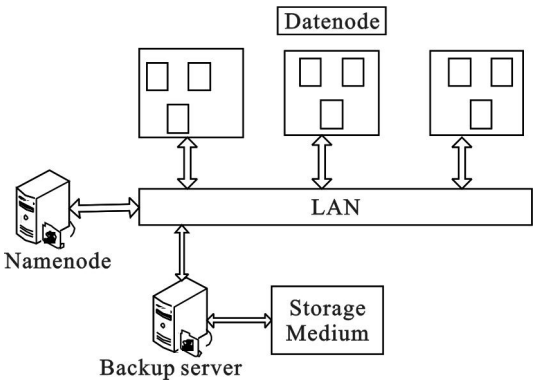

图 3　数据备份体系结构

整个系统分为三部分:名字节点 Namenode、数据节点 Datenode 和备份服务器 Back server。各部分的功能如下:

Namenode 保存文件存储系统中所有文件的相关信息、管理文件元数据、负责管理和协调各个 Datenode。但 Namenode 不负责存储文件,也不负责具体的文件上传下载工作,这样就大大减轻了 Namenode 的负担,避免 Namenode 成为系统的瓶颈。

Datenode 是文件存储的基本单元,以数据块的形式保存着 HDFS 中文件的内容和数据块的校验信息。备份恢复工程中,直接与备份服务器建

立数据上传和下载链接。

Backup server 是备份恢复系统的核心,负责具体备份恢复工作。备份文件时,备份服务器首先从 Namenode 上获得备份文件的相关信息,然后和存储文件的 Datenode 建立连接,下载文件,在将文件存储在备份介质上之后,记录备份文件的相关信息。

2.2 备份分组算法思想及相关概念

备份分组算法的思想就是:在多线程下载备份任务的前提下,为充分利用节点的网络带宽,使所有数据节点在同一时刻都有正在下载的数据块。为进一步描述备份分组算法,下面给出几个基本概念定义描述。

定义 1:备份任务。设一个文件 file 由 block1,block2…blockn n 个数据块组成,如式(1)所示,对该文件所有数据块的下载任务之和称为一个备份任务,用 Task_m(m 为任务编号)表示,形式化描述如式(2)所示:

$$\mathrm{file} = \sum_{i=1}^{n} \mathrm{Block}_i \tag{1}$$

$$\mathrm{Task}_m = D(\mathrm{file}) = D\left(\sum_{i=1}^{n} \mathrm{Block}_i\right) = \sum_{i=1}^{n} D(\mathrm{Block}_i) \tag{2}$$

式(2)中,备份任务 Task_m 描述该文件所有数据块的下载任务,$D(\mathrm{Block}_i)$ 为对数据块的下载,称为备份子任务。在进行备份操作之前,对文件最后修改时间与当天日期进行匹配,如果一致,则确定一个备份任务。

因为数据块在 HDFS 数据服务器中存在多个副本,所以需要确定下载哪个数据节点上的数据块,即确定备份子任务。确定备份子任务的基本思想是:根据数据块(包括副本)在数据节点上的实际物理分布情况,按照数据节点上映射的数据块数量平均化策略,依次排除数量多的数据节点上的数据块副本,从而筛选出需要下载的数据块。

定义 2:备份任务组。所有的备份任务确立后(设任务数量为 N),在备份子任务定义的基础上,把 n 个数据节点 dataNode 上同时进行的备份子任务的集合称为备份任务组。因为备份任务组是备份子任务的集合即数据块备份的集合,因此在不同的时间段会有不同的任务组形成。形式化描述如式(3)所示:

$$\mathrm{TaskGroup} = \left\{ \begin{matrix} D(\mathrm{dataNode}_1\_\mathrm{file}_i.\,\mathrm{block}_p), \cdots, \\ D(\mathrm{dataNode}_n\_\mathrm{file}_j.\,\mathrm{block}_q) \end{matrix} \right\} \quad (i,j \in (1,N)) \tag{3}$$

式(3)中,n 个数据节点上同时下载的数据块可以全部属于一个文件,也可以属于不同的文件,即使任意两个数据块属于同一个文件,即 $i=j$,而 $p \neq q$ 时,按照定义 1 的描述,筛选数据块确立备份子任务的过程也能保证各数据节点所映射的数据块的唯一性。根据数据节点与数据块 ID 的映射,判断数据节点上是否还有数据块,若有则数据节点随机选择一个数据块进行下载,从而确定一个任务组。

备份任务组确立以后,需要进行备份任务组的初始化工作,即首先获取备份子任务的信息,包括数据块的 ID、数据块所在的数据节点、数据块所属文件源路径和下载数据块的目标路径。可用如式(4)所示的四元组 $\mathrm{Info}_{D(\mathrm{block}_i)}$ 来描述。

$$\mathrm{Info}_{D(\mathrm{block}_i)} = [x,y,z,t] \tag{4}$$

其中,x 为下载数据块的 ID,y 为所在的数据节点,z 为所属文件的源路径,t 为下载的目标路径,z 与 t 是镜像关系,即在备份中心文件的逻辑组织方式与在 HDFS 中相同。设 C_{blockIDs} 是备份子任务中要下载的数据块 ID 集合;$C_{\mathrm{dataNodes}}$ 是数据节点集合;$C_{\mathrm{sourcepaths}}$ 是备份任务中要下载的文件源路径集合;$C_{\mathrm{tartetpaths}}$ 是备份任务要下载到的目标路径集合,则初始化的过程如下。首先,建立三个映射关系:

C_{blockIDs} 到 $C_{\mathrm{dataNodes}}$ 的映射关系:$R_{\mathrm{blockIDsto\ DataNodes}} = \{[\langle x,y\rangle \mid x \in C_{\mathrm{blockIDs}}, y \in C_{\mathrm{dataNodes}}\}$;$C_{\mathrm{blockIDs}}$ 到 $C_{\mathrm{sourcepaths}}$ 的映射关系:$R_{\mathrm{blockIDstoSources}} = \{\langle x,z\rangle \mid x \in C_{\mathrm{blockIDs}}, z \in C_{\mathrm{sourcepaths}}\}$;$C_{\mathrm{sourcepaths}}$ 到 $C_{\mathrm{tartetpaths}}$ 的映射关系:$R_{\mathrm{SourcestoTargets}} = \{\langle z,t\rangle \mid z \in C_{\mathrm{sourcepaths}}, t \in C_{\mathrm{targetpaths}}\}$。

然后,一旦一个任务组内的数据块 IDx 已定,根据映射关系 $R_{\mathrm{blockIDstoDataNodes}}$ 和 $R_{\mathrm{blockIDstoSources}}$ 可确定其所在的数据节点 y 和所属文件的源路径 y;根据映射关系 $R_{\mathrm{blockIDstoSources}}$ 和 $R_{\mathrm{SourecestoTargets}}$ 可确定其下载的目标路径 z。

备份窗口是指在不影响系统正常业务的情况下,用于备份的时间段。在保证数据全部备份完成的前提下,所花费的最少时间称为最小备份窗

口。最小备份窗口与读请求响应时间、网络传输速度以及磁盘读写速度有关。以下给出最小备份窗口的明确定义和估算公式。

定义 3:最小备份窗口。完成所有备份任务所需要的时间,其估算公式如式(5)所示:

$$T = m \times \left(\frac{n \times S_{\text{block}}}{v_1} + \frac{S_{\text{block}}}{v_2} + \frac{p \times S_{\text{block}}}{v_2} \right) + T_0 \qquad (5)$$

其中,n 为 HDFS 中数据节点个数;$p(1 \leqslant p \leqslant n)$为不同备份任务组内任务的个数,$p$ 随着备份的进程而变化的;m 表示备份任务组的数量;S_{block} 表示 HDFS 的数据块大小(默认为 64M);v_1 为网络传输速率(在百兆交换机的情况下,v_1 的均值为 11M/S);v_2 为磁盘读写速度;T_0 为读请求相应时间。其中,读请求响应时间 T_0 可以忽略。根据映射关系 $R_{\text{blockIDstoDataNodes}}$ 能够得到其反映射关系 $R_{\text{DataNodestoblockIDs}}$,从而计算出下载所有的备份任务所需要建立的备份任务组的数量 m,即所有数据节点映射的数据块数量的最大值,其计算公式如式(6)所示:

$$m = \text{Max} \left\{ \begin{matrix} N_{\text{dataNode}_1.\ \text{blockID}}, \cdots, N_{\text{dataNode}_i.\ \text{blockID}} \\ \cdots, N_{\text{dataNode}_n.\ \text{blockID}} \end{matrix} \right\} \qquad (6)$$

式(5)中,为了能够计算,取 p 的最大值 n,因此理论值会比实际测量值大。

2.3　分组算法思想描述

根据上述思想,分组算法的自然语言描述如下:

(1)遍历 HDFS 文件目录,将文件最后修改日期与当天日期进行匹配,如果一致则确定一个备份任务并放入备份任务集合中,建立备份源路径集合与目标路径集合的映射、数据节点与数据块 *ID*(包括副本)的映射和数据块 *ID* 集合到备份源路径的映射。

(2)根据节点与数据块 *ID*(包括副本)的映射,确定数据节点中对应的数据块数量,然后根据某一数据块 A 所在的数据节点中总的数据块数量,选择数量少的数据节点作为数据块 *ID* 集合到数据节点集合映射中 A 的象,从而建立起数据块 *ID* 集合到数据节点集合的映射,建立映射过程即为备份子任务筛选数据块的过程。

(3)根据数据块 *ID* 集合到数据节点集合的映射,得到其逆映射关系即数据节点到数据块 *ID* 集合的映射,计算数据节点映射的数据块数量的最大值 n。

(4)根据数据节点到数据块(不包括副本)*ID* 集合的映射,建立一组备份任务组,若 $n=0$,则转到(7)。

(5)根据建立的映射关系,得到备份任务组内每个子任务信息,初始化备份任务组,进行数据块的下载。

(6)一组备份任务所有备份子任务完成后,将每个数据节点映射的数据块 *ID* 删除,n 减 1 并转到(4)。

(7)算法结束。

形式化语言描述如下:

输入:HDFS 根目录 dirpath;

初始化:

Set C_{files}←new HashSet(FileInfos);

//更新的文件集合(备份任务集合)

$R_{\text{SourcetoTarget}}$←new HashMap(string,string);

//源路径到目标路径的映射

$R_{\text{blockIDstoSources}}$←new HashMap(Long,Srting);

//数据块 *ID* 到源路径的映射

$R_{\text{dataNodestoAllblockIDS}}$ ← new HashMap(String, HashSet<Long>);

//数据节点到数据块 *ID* 的映射

$R_{\text{blockIDstoDataNodes}}$←new HashMap(Long ,DataNodeInfo);

//数据块 *ID* 到数据节点的映射

$R_{\text{dataNodestoblockIDs}}$←new HashMap(String, HashSet<Long>);

//数据节点到筛选后数据块 *ID* 映射

maxNum←0;

//数据节点中数据块(筛选后)数量的最大值

dataNodeNum←0;

//数据块数量不为 0 的数据节点个数

建立映射:

(1) C_{files}←createFilesHashSet(dirpath);

(2) $R_{\text{SourcetoTarget}}$ ← createSourcetoTargetHashMap(C_{files});

(3) $R_{\text{blockIDstoSource}}$ ← createBlockIDstoSourcesHashMap(C_{files});

(4) $R_{\text{dataNodestoAllBlockIDs}}$ ← createDataNodestoAllBlockIDsHAshMap(C_{files});

为备份子任务筛选数据块的过程(建立数据节点到筛选后数据块 *ID* 的映射):

(1) for $i=0$ *to* C_{files}. *size do*

(2) locatedBlock←namenode. getBlockLocations(C_{files}. Get[i]. getFilePath(),0,C_{files}. get[i]. getFileSize());

//获得文件的数据块信息

(3) locatedBlockList←locatedBlock. getlocatedBlocks();

//数据块链表信息

(4) for $j=0$ to locatedBlockList. size do

(5) a←R. dataNodestoAllblockIDs. get(locatedBlck[j]. getLocations()[0]. getHost()). size();

//将第一个数据块副本所在的数据节点中数据块数量赋值给 *a*

(6) b← R. dataNodestoAllblockIDs. get(locatedBlock[j]. getLocations()[1]. getHose()). size();

//将第二个数据块副本所在的数据节点中数据块数量赋值给 *b*

(7) c←R. dataNodestoAllblockIDs. get(locatedBlock[j]. getLocations()[2]. getHose()). size();

//将第三个数据块副本所在的数据节点中数据块数量赋值给 c

(8) ←M min(a,b,c);

(9) if $m==a$ then

(10) $R_{blockIDstoDataNodesHashMap}$. put(locatedBlock,getBlcok()

. getBlcokID(),locatedBlcok. getLocations()[0]);

//若 *a* 为最小值,则把第一个数据块所对应的数据节点放入数据块 *ID* 到数据节点的映射中

(11) end if

(12) if $m==b$ then

(13) $R_{blockIDstoDataNodesHashMap}$. put(locatedBlock,getBlcok()

. getBlcokID(),locatedBlcok. getLocations()[1]);

//若 *b* 为最小值,则把第一个数据块所对应的数据节点放入数据块 *ID* 到数据节点的映射中

(14) end if

(15) if $m==c$ then

(16) $R_{blockIDstoDataNodesHashMap}$. put(locatedBlock,getBlcok(). getBlcokID(),

locatedBlcok. getLocations()[2]);

//若 *c* 为最小值,则把第一个数据块所对应的数据节点放入数据块 *ID* 到数据节点的映射中

(17) end if

(18) end for

(19) end for

(20) $R_{dataNodestoblockIDs}$ ←creatDataNodestoblockIDsHashMap(C_{files},$R_{blcokIDstoDataNodes}$);

创建任务组并下载:

(1) maxNum←max($R_{dataNodestoblockIDs}$); //获得数据节点中数据块(筛选后)数量的最大值

(2) dataNodeNum ← getDataNodeNum($R_{dataNodestoblockIDs}$);//获得数据块数量不为 0 的数据节点数

(3) for $i=0$ to maxNum − 1 do

(4) for $j=0$ to dataNodeNum − 1 do

(5) if $R_{dataNodestoblockIDs}$. get(dataNode[j]). size >0 then

TaskGroup[i]. add(dataNode[j]. block[i]);//

(6) $C_{blockID}$. add(dataNode[j]. block[i]);

(7) end if

(8) end for

(9) run(TaskGroup[i]. $R_{blockIDstoSources}$, $R_{blockIDstodataNodes}$,$R_{SourcestoTargets}$);//进行任务组的下载

(10) end for

3 实验及结果分析

实验环境为三台机架服务器和一个磁盘阵列。三台机架服务器中,一台为 12 核、24 线程、16G 内存,型号为 A620r − G;另两台机架服务器为 24 核、48 线程、64 G 内存,型号为 A840r − G;磁盘阵列有 10 T 容量,型号为 DS200 − N10。在服务器上,搭建 Hadoop 云环境:1 个 namenode 节点,1 个 snamenode 和 8 个 datanode 节点。Hadoop 软件版本为 1.2.1,Java 版本为 jdk1.7.0_40。

实验分别以 5.09 G、11.86 G、14.11 G、30.25 G 的备份任务量为例。备份介质为普通台式机磁

盘,其读写速度在 90 M/S – 150 M/S 之间,取 128 M/S,得出分组算法中最小备份窗口的理论与实测值对比,如图 4 所示。

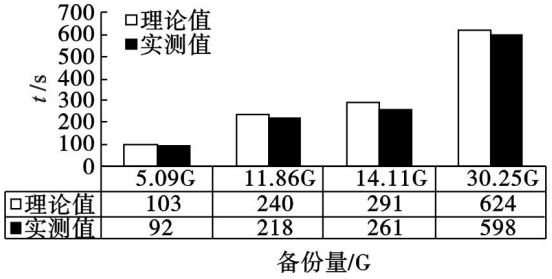

图 4　最小备份窗口估算值与实测值对比

从图 4 中可以看到,估算值与实测值相差不大。4 次比较误差依次为 11 秒、22 秒、30 秒、26 秒。在备份任务量大的情况下,可以用公式(5)进行估算。

分别对单线程下载及多线程并行下载前提下的 HDFS 默认算法和分组算法进行时间效率对比,如图 5 所示。

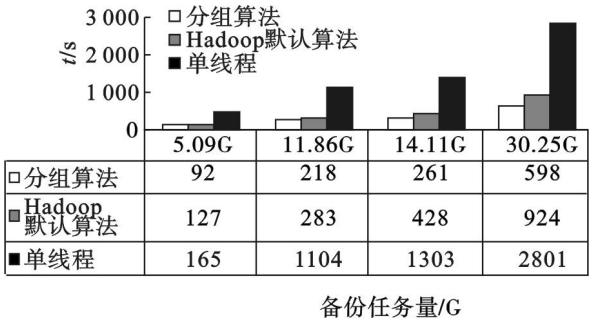

图 5　备份时间效率对比

对于单线程执行备份,由于 HDFS 读数据块流程是顺序的,节点带宽只能利用 1/8(8 个节点),花费的时间随任务量增加基本呈线程增长,远比多线程要高得多。多线程方式下,Hadoop 默认算法随着备份任务的进行,读请求集中在某几个数据节点,导致 8 个数据节点的带宽得不到充分利用。而备份分组算法定位到数据块,得到备份任务组里所有文件的数据块信息,按 N 个数据节点里的数据块建立备份子任务组,使所有节点时刻都有任务,达到了带宽近 100% 的利用。从图 4 中看出,在任务量小的情况下,分组算法与默认算法时间差距不大,随着任务量的增加,分组算法的优势越来越明显,在海量数据面前有着较高的下载效率。因而,该增量备份策略能够解决日益增长的企业海量 PDM 数据的备份问题。

4　总结

本文在基于企业私有云存储 PDM 数据的前提下,针对 PDM 数据如何备份这一新问题,提出了一种以日为粒度的增量备份方案。该方案能够将企业私有云中的 PDM 数据,以 HDFS 中相同的目录组织方式,高效地备份到集中式的环境中,能够解决企业海量 PDM 数据的备份问题,同时在恢复时,只要按照备份中心的目录进行恢复即可。

参考文献(References):

[1] Miller E. PDM Today[J]. Computer Aided Design, 1995,14(2):32 – 41.
[2] 李伯虎,张霖,王时龙. 云制造—面向服务的网络化制造新模式[J]. 计算机集成制造系统,2010,16(1):1 – 7,16.
[3] 李伯虎,张霖. 再论云制造[J]. 计算机集成制造系统,2011,17(3):449 – 457.
[4] 夏秀峰,赵小磊,孔庆云. MBE 与大数据给 PDM 带来的思考[J]. 制造业自动化,2013,10(35):71 – 72.
[5] LI W J,LI L X,LI F L,et al. Design and implementation of backup system based on P2P network[J]. Journal of Inform ation Engineering University,2010,11(3):351 – 355.
[6] Apache Hadoop. Welcome to ApecheTM Hadoop(r)[EB/OL]. 下载地址:http://Hadoop. apache. org/.
[7] 栾亚建,黄翀民,龚高晟. Hadoop 平台的性能优化研究[J]. 计算机工程,2010,36(14):262 – 264.
[8] Konstantin Shvachko, Hairong Kuang and Sanjay Radia. The hadoop distributed file system[C]. Proceedings of the 26th IEEE Symposium on Mass Storage Systems and Technologies(MSST 10). Lake Tahoe, May 3 – 7,2010. 1 – 10. IEEE,c2010.
[9] 梅中义. 基于 MBD 的飞机数字化装配技术[J]. 航空制造技术,2010(18):42 – 43.
[10] 姜红明,张丰华,吴慧杰. MBD 技术实施研究[J]. 制造业自动化,2012,134(12):1 – 3,12.
[11] 余定方. 面向飞机全生命周期的 MBD 数据管理解决方案探讨[J]. 航空制造技术,2010(23):124 – 127.
[12] Yin Haijun, Ning Junyi, Han Zhiren. Study on application technology of lightweight model oriented to no – drawing manufacturing[C]. 2012 2nd International Conference on Chemical, Material and Metallurgical Engineering, ICCMME 2012, December 15, 2012 – December 16,2012:3798 – 3801.
[13] Ferraiolo D, Kuhn R, Sandhu R. RBAC standard rationale:Comments on "A critique of the ANSI standard on role – based access control"[J]. IEEE Security &Privacy,2007,5(6):51 – 53.
[14] 杨帆. Hadoop 平台高可用性方案的设计与实现[D]. 北京:北京邮电大学网络技术研究院,2012:7 – 12.
[15] 王昌旭,许榕生. 企业数据备份系统分析与研究[J]. 计算机应用软件,2008,25(10):122 – 123.

(责任编辑:刘划　英文审校:宋晓英)