高技术通讯 2013 年 第23 卷 第12 期:1219 ~1229

doi:10.3772/j.issn.1002-0470.2013.12.002

大规模时空数据分布式存储方法研究①

钟运琴② *,** 方金云* 赵晓芳*

(*中国科学院计算技术研究所 北京 100190)

(**中国科学院大学 北京 100190)

摘 要 提出了一种基于 Hadoop 云平台的时空数据分布式存储方法,以应对空间应用中出现的无法满足高并发用户在线实时访问和空间信息服务中断等大数据存储瓶颈问题。该方法运用时空数据切分与布局机制使数据均匀分布于集群中以确保存储与访问负载均衡;运用时空对象重组织机制提高数据的时空临近性以匹配时空应用存取模式;运用热点时空对象分布式缓存机制以降低磁盘 I/O 访问延迟。利用该方法实现了基于 Hadoop 云平台的时空数据分布式存储中间件原型系统 exHDFS,实验结果表明该方法能高效地满足数据密集型空间应用存储需求。

关键词 云平台,空间应用,数据管理,时空数据存储,时空索引,时空数据缓存

0 引 言

时空数据是一种结构复杂、多层嵌套的具有空间和时态特性的高维数据,它有效记录了事物的空间位置和时空变化过程,并准确地表达了事物的历史、当前和未来状态,如城市变迁、疾病扩散、环境变化、地质演化、移动对象位置变更等[1]。随着对地观测和位置服务技术的发展,人们通过专业传感器设备采集了大量异构多源时空数据如高分辨率遥感数据、基础地理数据、航拍影像、GPS 轨迹数据等。在线空间应用如网络地理信息系统(WebGIS)、基于位置的社交网络服务(LBSNS)的普及,使得普通用户也可以和专业人员一样通过桌面浏览器和移动终端发布、浏览、查询时空数据[2],他们也成为数据生产者,这进一步加剧了数据规模的增长[1]。当前主流的时空数据存储方法由于单机空间数据库的吞吐性能与扩展能力有限,已无法满足高并发用户近在线实时访问,而且极易发生单节点失效问题造成空间信息服务中断。因而我们需要设计高效的时空数据存储方法以满足高并发用户访问海量数据的需求。本研究针对这种现实情况进行了大规模时空数据存储方法探索,提出了一种基于 Hadoop[3] 云平台的时空数据分布或存储方法,并通过实验证实了该方法的有效性。

1 研究背景与相关研究

时空数据存储是空间应用后端系统的核心,其性能决定着整个系统的服务能力。最新的时空数据存储方法大都基于空间数据库管理系统(spatial database management system,SDBMS)而设计的,它们的存储能力很大程度上依赖于底层 DBMS 性能[4]。大规模时空数据和高并发访问对基于 SDBMS 的时空数据存储方法是很大的挑战。从可扩展性上看,SDBMS 通过垂直扩展的方式,即通过增加性能更高的硬件如更快的 CPU、大容量内存和高速磁盘等来提升系统性能,但是单机环境下硬件的性能扩展有限,因而 SDBMS 的并发访问和 I/O 吞吐性能无法持续提升。尽管使用并行数据库系统可以提升数据存储性能,但是其扩展能力也很有限[5],而且其商业版本需要支付高昂的授权费用和配备昂贵的硬件设备以提升性能。从可用性上看,当并发用户少和数据量小时,单机 SDBMS 可满足应用需求,但随着数

① 863 计划(2011AA120300,2011AA120302),中国科学院知识创新工程重要方向项目(KGCX2-YW-174)和中国科学院研究生科技创新与社会实践资助专项资助项目。

② 男,1984 年生,博士生;研究方向:时空数据管理,地理信息系统,分布式存储系统;联系人,E-mail:zhongyunqin@ict.ac.cn

(收稿日期:2012-12-26)

据规模和并发用户数持续增长,系统极易出现 I/O 拥塞导致存取速度变慢,甚至发生单点失效问题导致服务中断。

面向互联网级数据密集型应用(如在线地图服务和 LBSNS 等)的集群云平台属于存储与计算紧耦合型结构,每个节点均具有独立的存储设备和计算单元,节点之间采用无共享架构水平扩展。在 Yahoo! 等商业公司的支持下,Apache 组织发布了 Google 云平台的开源实现:Hadoop 平台。Hadoop 包括分布式文件系统(hadoop distributed file system, HDFS)[6]、Hadoop MapReduce、键值数据库 HBase、数据仓库 Hive 等。目前学术界和工业界都在不断地丰富 Hadoop 生态系统,利用其良好的扩展性、容错性和并行处理能力等,研发了面向不同应用领域的数据处理系统。同时还有研究将 MapReduce 与 DBMS 融合使 MapReduce 支持在线处理[7],如 HadoopDB[8] 在线分析处理半结构化数据,RCFile[9] 基于 Hadoop 专门为数据仓库分析型应用而设计的文件结构。

时空数据的分布式存储管理研究尚处于起步阶段,目前仍未出现成熟方案和产品,近年来已有利用分布式集群存储管理空间数据的研究成果。Samet 等研究了基于 SDBMS 的空间数据存储索引方法,将数据分割与分组去耦处理以提高空间数据组织索引效率[10]。文献[11]利用分布式文件系统存储 Web-GIS 应用数据,提出了基于分布式文件系统的地图数据存储策略并优化了小文件 I/O 性能。Zhong 等提出了利用分布式键值存储系统管理栅格地图瓦片的方法[12]。文献[5]提出了基于小规模 MapReduce 集群的空间窗口查询方法,但处理时未引入空间索引机制,而使用暴力扫描法对每个要素对象进行空间运算,因此在针对大规模数据集时由于其 I/O 访问延迟大,造成整个查询处理代价过高。文献[13]提出了在对等(Peer-to-Peer)分布式环境下的空间索引方法 SD-Rtree,它改进了在新增数据场景下的索引树结点分裂与数据节点的资源分配算法,当结点分裂时重新分配一个节点存储该结点所覆盖空间范围内的数据,从而实现动态负载均衡。Liao 等提出了在 HDFS 上构建多维索引使之支持全局最近邻查询[14]。MD-HBase[15] 基于键值存储系统构建了多维四叉树索引以支持位置相关的用户兴趣点查询,MD-HBase 的空间位置数据存储在开源的键值对存储系统 HBase 中。CareDB[16] 是一个用户兴趣偏好和地理位置感知的数据管理系统,用于支撑个性化的 LBS 推荐服务。GeoFeed[17] 是一个基于位置感知的新闻推荐系统,它通过将社交推荐信息加上位置标签和空间属性,用于支持空间范围查询给用户推荐其邻近地理区域内的新闻信息。以上这些方法都只涉及到静态空间数据,而未能有效存储动态时空数据,无法有效提供高吞吐量、低延迟数据访问,不支持时空查询,因而无法满足时空应用需求。迄今尚未有很好的公开成果深刻阐述基于 PaaS(platrorm as a service)云计算平台的 大规模时空数据存储方法研究。

在这种情况下,本文提出了基于 Hadoop 云平台的大规模时空数据存储方法以提升时空存取性能。该方法运用三个关键机制来克服现有方法的不足。一是地理空间位置感知的时空数据切分与布局机制,该机制基于地理时空局部性和数据块阈值加权原则将海量时空数据按照地理空间进行逻辑划分,子区域内的对象组织成符合阈值条件的物理数据块并分配全局唯一的标识号,通过时空布局算法将它们均匀分布于集群中以确保数据存储和访问负载均衡。二是时空对象重组织机制,该机制充分考虑时空对象邻近性和磁盘存储局部性原理,将时间与空间位置上邻近的时空对象按照空间填充曲线次序存储在数据块内连续的磁盘页中。根据 Tobler 地理学第一定律和空间应用特点,空间上越邻近的对象其关联性越强,当访问了某特定区域内的时空对象时,其邻近时空范围内的对象被访问的可能性也很大。因此,当数据对象被访问时,可以将其邻近位置上的数据顺序传输至客户端,减少随机 I/O 次数,充分利用磁盘的顺序 I/O 以提高数据访问吞吐率。三是热点时空对象分布式缓存机制。由于大部分空间应用是读多写少存取模式,当高并发用户访问时,将经常被访问的热点时空对象预加载至分布式缓存中可有效降低访问延迟。本研究实现了基于 Hadoop 云平台的时空数据存储中间件原型系统 extHDFS,它充分利用集群的存储与计算能力,有效消除了时空存取模式与 Hadoop 之间的鸿沟。通过将真实空间应用部署至 extHDFS 上的实验结果表明,extHDFS 能有效支撑数据密集型空间应用。本文方法对基于其它 PaaS 平台构建扩展性好、可用性高的时空数据存储系统具有一定的借鉴意义。

2 基于地理感知的时空数据切分与布局机制

我们设计的具有地理位置感知的时空数据切分

机制将大规模时空数据分割成等额大小的数据块集合,并将它们均匀分布至集群节点上。该机制根据地理时空范围对数据进行分区,将空间位置与时间序列邻近的时空对象组织成同一逻辑分区并存储在相同数据块中,而且为数据块大小设定一个阈值,每个节点维护一个或者多个逻辑分区。该机制的切分思想是:首先将预设时间间隔内的整个地理空间划分成若干逻辑区域;然后将逻辑子区域内的对象组织在符合阈值上限的数据块中,若超过阈值,则将逻辑子区域递归划分直至符合阈值条件为止。该机制可保证空间区域内对象的邻近性和磁盘的存储局部性,即确保时空邻近的数据对象存储在连续的磁盘块中,并使同一个子区域内的数据只占用一个块单元,从而可利用磁盘顺序 I/O 性能,减少随机 I/O 以提高数据存取性能。

为将时空数据有效存储于 Hadoop 分布式文件系统(HDFS)中,首先设计了一种能够唯一标识时空对象和时空区域的虚拟编码方法,该编码方法能准确表达数据的时空临近性,本文后面的时空对象块内组织结构也用该编码方法进行标识。

2.1 时空数据虚拟编码方法

本文中的时空数据是以图层(Layer,L)为单位组织的,将具有相同几何特性和表达属性的同一类时空对象集合组织在一个图层中,如移动点、道路、位置区域图层等。实际应用常会涉及不同图层,多个图层数据处理结果叠加得到最终结果。时空对象具有 4 个属性:标识符(Identifier,ID)、最小包围矩形(minimum bounding rectangle,MBR)、空间坐标值(x,y)和时间 T。图层具有 3 个属性:图层名、MBR 和分辨率尺度(Scale,S)。本文设计了虚拟编码方法为划分后子区域及其包含的时空对象进行编码为图层内子区域及其时空对象提供唯一标识。

2.1.1 虚拟编码相关定义

定义 1(虚拟编码):虚拟编码(virtual code,VC)是一种由四进制字符串组成的编码,编码长度为 n 的 VC 可以形式化表示为 $VC = c_1c_2\cdots c_n$,其中 $c_i \in \{0,1,2,3\}, i \in \{1,2,\cdots,n\}$,字符 c_i 表示每级划分的四叉子区域字符表示。

定义 2(编码长度):在一个包含 $2^s \times 2^s$ 个空间对象的图层,其每个对象的 VC 编码长度为 s。

定义 3(区域边长):假设有 $2^s \times 2^s$ 的区域,均分子区域大小为 $2^k \times 2^k$,则子区域 VC 长度为 k 其边长为 2^{m-k}。

定义 4(子区域大小):假设一个图层区域大小为 A,那么其划分子区域大小为 $4^{-m}A$,其中 m 表示子区域边长。

2.1.2 编码过程

将图层内的时空对象位置信息编码成唯一的 VC,并且能通过 VC 指示对象所在的子区域编码。假设空间对象的位置用坐标(x,y)表示,所在的图层区域大小为 $2^s \times 2^s$,则 VC 形式化为

$$\begin{cases} x = (x_1x_2\cdots x_s)_2 \\ y = (y_1y_2\cdots y_s)_2 \\ VC = c_1c_2\cdots c_s = (y_1x_1)(y_2x_2)\cdots(y_sx_s) \end{cases} \tag{1}$$

$$VC = \sum_{i=1}^{s}(2y_i + x_i) \times 4^{s-i} \tag{2}$$

如式(1)所示,首先将坐标(x,y)转换成对应的二进制形式,二进制串的长度等于该图层的区域边长 s,式(2)表示对应的编码数值。

图 1 所示的是区域边长为 2 的对象编码过程示意图。图 1(a)表示将坐标(x,y)转换成二进制形式;图 1(b)显示空间对象的 VC 编码串;图 1(c)显示了用四进制表示的时空对象的 VC 值。例如,坐标为(1,2)的对象的 VC 值为“21”;坐标为(2,2)的对象 VC 值为“30”。

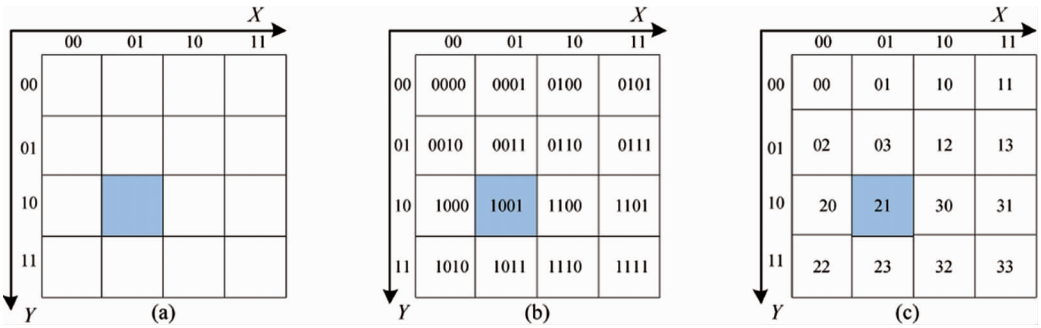

图 1 *VC* 编解码过程示意图

2.1.3 解码过程

解码过程是将时空对象的 VC 值解码成相应的坐标值,用于在正确的地理坐标位置上表达该对象,以得出真实空间值。时空对象的空间坐标值和其 VC 值一一对应,解码过程可用下式计算得出:

$$\begin{cases} x = \sum_{k=1}^{s} (c_k \bmod 2) \times 2^{s-k} \\ y = \sum_{k=1}^{s} (c_k div 2) \times 2^{s-k} \end{cases} \quad (3)$$

由式(3)所示,s,k 分别表示该图层与其子区域 VC 值的长度,(x,y) 表示对象坐标值,c_k 表示 VC 的第 k 位值,其中 $x,y \in [0,2^{s-1}]$ 且 $c_k = \{0,1,2,3 \mid k = 1,2,\cdots,s\}$,如图 1(c)所示,VC = 21 的对象解码的空间坐标为(1,2)。

2.2 基于改进型四叉树的划分策略

时空数据以图层为单位采用四叉划分方法分割成若干个逻辑子区域,划分条件由图层的最小包围矩形(MBR)和数据块阈值共同决定,四叉划分算法的具体执行步骤如下:

(1)首选计算整个图层区域内的所有时空对象大小之和,包括块内局部索引和所有时空对象的长度之和。

(2)若区域内所有对象长度之和不超过阈值,则停止划分该区域,并将区域内的对象组织成一个数据块;否则继续将该区域递归地均匀划分为四个更小的子区域。

(3)分别计算每个子区域内所包含的时空对象长度之和,并调用步骤(2)对新的子区域进行判断,直至所有子区域内的对象长度之和符合阈值条件。

采用以上步骤将图层划分为若干个子区域,并把每个子区域内的时空对象组织成一个独立数据块后,最后将所有数据块均匀分布至 HDFS 数据节点上。通常设定数据阈值与 HDFS 块大小一致,如设置为 64MB,目的是为了保证划分后的子区域内的时空对象存储在一个数据块中,从而确保区域内的数据块存储在同一数据节点上。阈值的设置要避免过小或过大,因为阈值过小会造成 HDFS 主节点需要维护过多的元数据信息,造成主节点元数据检索效率变低而导致数据存取性能下降;阈值过大则会导致同一子区域内的时空数据需要传输至两个或两个以上节点,造成数据分布不均匀,而且访问时需跨节点访问以还原数据块,因而会引入额外开销增加访问延迟。

图 2 是在时间间隔为 $[t_s,t_e]$,区域长度为 4 的图层数据被划分成若干个逻辑分区的示意图,实线框表示划分后子区域内的数据组织成一个物理块,虚线框代表在该子区域所包含的时空对象。根据划分规则,无论子区域或者对象都是用四进制虚拟编码表示,尽管划分后的子区域边长可能不相同,但是同一子区域内的对象具有相同的编码前缀,如图 2 所示的四进制数字串表示时空对象前缀编码,即区域编码。在实际应用中,根据四叉划分策略将数据分割成若干个符合阈值的数据块,并按照空间区域和时间范围将数据均匀分布在集群中的不同节点上。

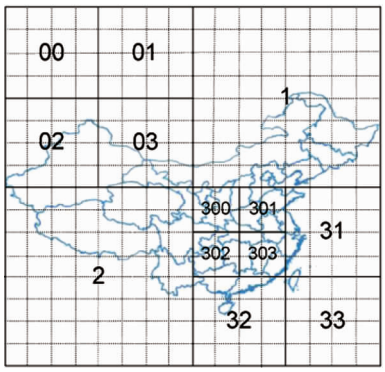

图 2 四叉划分示意图

3 时空对象重组织机制

在地理空间应用中,客户端访问某个区域内的对象时,也会有很大的概率访问其邻近对象。考虑时空存取模式,本文设计了基于空间填充曲线的时空对象块内组织方法将空间位置上邻近的对象存储在连续的磁盘页中以保证对象邻近性与存储局部性。当客户端访问某区域时,系统将顺序读取该区域内空间位置和时间上连续的邻近对象,减少随机存取 I/O 操作次数以加速数据访问效率。

由于 Hilbert 填充曲线具有保持空间邻近性的特点,本文将同一区域内的对象按照 Hilbert 次序存储可以保证对象的空间邻近性与数据内容的存储局部性。根据区域内的对象数目选择不同阶数的 Hilbert 遍历次序,若区域内有 $2^n \times 2^n$ 个对象,则选择 n 阶 Hilbert 曲线;并依据对象的 VC 值进行 Hilbert 排序遍历。如图 3 所示,不同子区域内时空对象以 Hilbert 次序存储组织,图中蓝线表示 1 阶 Hilbert 次序;红色表示 2 阶 Hilbert 次序。

本文设计了专门的数据块存储结构用于组织同一区域内的对象及其局部时空索引信息,该数据块结构包含两个部分:局部索引和实际数据内容部分。

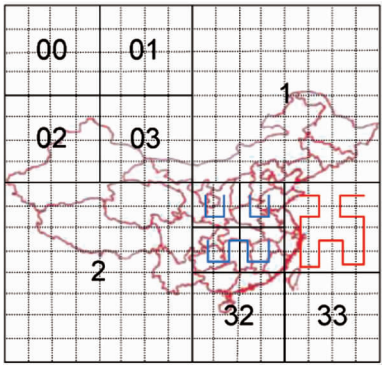

图 3　Hilbert 次序组织数据

如图 4 所示,数据块的前半部分为局部索引,后半部分为数据内容。数据块的首部是包含空间区域的时间戳、VC 值和 MBR 信息,其中 VC 值为四进制虚拟编码、MBR 用四个双精度浮点数表示,占 32 字节。索引部分依次是第一个对象的坐标值(x,y)和区域长度值,分别用四个字节整型表示;接下来依次是根据 Hilbert 次序从该区域第一至最后一个对象的长度和偏移量,数据部分依次存储对应的空间对象的数据内容。数据块的长度为局部索引和数据部分的长度之和,其大小不超过划分阈值。

VC Sequence	Region MBR(32B)	X-value of 1st obj(4B)	Y-value of 1st obj(4B)	*K*-value of region(4B)	Index
offset of 1st obj(4B)	length of 1st obj(4B)	offset of 2nd obj(4B)	length of 2nd obj(4B)	offset of 3rd obj(4B)	
length of 3rd obj(4B)	offset of rest objs…	length of rest objs …	offset of 2^{2k}obj(4B)	length of 2^{2k} obj(4B)	
real data of 1st obj	real data of 2nd obj	real data of 3rd obj	data of rest objs …	data of the 2^{2k} obj	Data

图 4　时空数据块内存储结构

根据虚拟编码方法,由已知对象的坐标信息就可以计算其对应的唯一的 VC 值。利用 VC 值的前缀定位相应的数据块后通过所请求对象和第一个对象的相对偏移位置就可以得到该对象在数据块中的序号,根据序号查询其索引信息就可以得到该对象的块内偏移和长度,最后访问实际数据内容。假如某区域内的第一个对象的坐标为(x_0,y_0),则坐标为(x_i,y_i)的数据对象在数据块内索引部分的序号为 $s_{ij}=(x_i-x_0)+(y_j-y_0)\times 2^k$,其中 $x_0\leqslant x_i\leqslant x_0+2^k-1$ 且 $y_0\leqslant y_j\leqslant y_0+2^k-1$。因此坐标为($x_i$,$y_i$)的空间对象索引信息可以通过读取索引部分中第 s_{ij} 个值得到。一个数据对象的索引条目信息需要 8 字节空间,而一个区域内的对象数目为 2^{2k},因此所有对象的索引长度之和为 2^{2k+3} 字节,数据块的前半部分占 $(2^{2k+3}+k+36)$ 字节。

除了在存储结构方面提高 I/O 存取性能外,本文还设计基于空间分划与范围叠加技术构建层次化的分布式索引结构以提高时空数据检索效率。该分布式索引由全局索引和局部索引两级索引结构组成,全局索引用于定位时空对象所在的节点和块位置;而局部索引用于确定对象在数据块内的偏移位置。大图层区域到子区域的映射关系由全局索引维护,由于子区域内数据对象存储在同一个数据块中,子区域到时空对象的映射信息由局部索引维护。全局索引在数据划分过程中所建立,是改进型的分布式四叉树结构。当图层被一分为四地划分时,四叉树的根结点表示整个图层,其 4 个孩子结点分别表示其 4 个子区域,由根结点到叶子结点依次建立四叉索引结构直至所有子区域满足条件为止。全局四叉树叶子结点指向其子区域的数据块,全局索引与 HDFS 目录结构相对应,且数据块用 VC 值命名,全局树结点所占用的空间极少,其大小等于相应的目录名长度。局部索引是对象在数据块内的索引,基于位图空间填充曲线索引结构而建立,局部索引按照 Hilbert 次序将时空对象索引信息依次写入数据块的索引部分。

图 5 是时空数据在 HDFS 中的存储结构示意图,图中最上面部分显示的是改进型全局四叉树索引结构。全局索引在 HDFS 中采用目录结构表示,目录名和块名就是 VC 编码串,由于全局索引属于 HDFS 命名空间的组成部分,因此可随系统启动时加载至主节点(NameNode)内存中,以便检索时快速定位到相应的数据块(在图 5 中用矩形框表示,后缀名是“.stb”)。局部索引作为元数据的一部分被

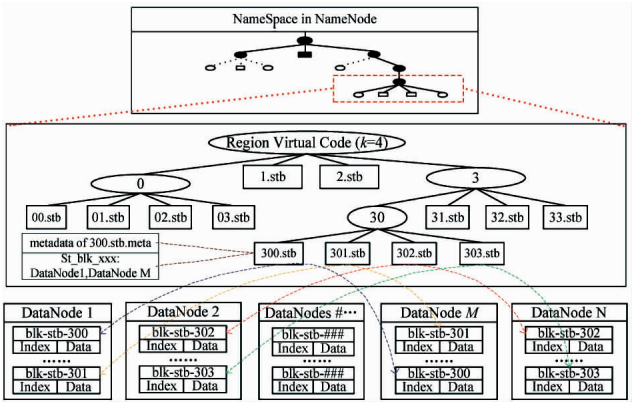

图 5　时空数据在 HDFS 重组织结构示意图

封装在数据块的首部,当访问数据块时,其块内的局部索引就会被加载至数据节点(DataNode)内存。图 5中间的框图显示了与图 2 中相对应的图层经过划分后在 HDFS 中的存储目录结构图,最上层根目录名用图层的 VC 编码值表示,每个目录里有且仅有 4 个子目录或者数据块文件,分别对应空间区域的 4 个子区域,子目录或数据块按照从左到右以 0 到 3 进行编号。最下面的示意图展示了由全局索引叶子结点所指示的最终数据块在 HDFS 集群中的分布情况,划分后的每个子区域只占用一个数据块。extHDFS 采用 HDFS 数据块副本备份机制提高系统可用性和容错性,每个数据块均有三个副本,而且这些副本分布在不同的节点中以提高数据的可靠性。extHDFS 主节点记录了各个节点所维护数据块的元数据信息,设置了检查点并采用整体复制的策略确保数据块副本之间的一致性使系统从错误状态中快速恢复数据块。当发生节点失效或其它错误时,extHDFS 会将访问请求重定向至存有副本块的节点,启动副本节点工作确保服务不被中断。

本文的分布式时空索引机制专为 Hadoop 云存储系统而设计,全局索引通过目录结构来表示,并将索引信息存储在主节点命名空间中。每个子区域内的时空对象组织成一个数据块,不同时间范围内的数据组织成不同时间戳的数据块。当 extHDFS 集群启动时,全局索引和数据块映射信息会被加载到主节点内存。客户端读取数据时,首先遍历主节点中的全局索引树,根据对象的时间范围和空间区域查找数据块内的局部索引定位相应的数据内容;然后客户端直接和数据节点交互访问数据内容,同时将相应的目录结构和块映射信息缓存至客户端;随后访问邻近区域对象时,客户端不再需要和主节点反复交互,直接查找自身维护的部分索引视图,并和相应的数据节点通信读取数据内容。由于大部分空间应用是读多写少存取模式,客户端缓存的只是主节点的部分全局索引信息的副本,全局索引更新周期较长,因此短期内的数据访问不需要考虑客户端和主节点之间的元数据一致性,只需将主节点上的索引信息推送给客户端。由于将属于同一子区域内的时空对象的局部索引信息和数据内容组织成一个数据块,数据块长度阈值保证了数据对象的局部索引和数据内容存储在同一个节点上,从而确保块内的局部索引遍历和数据存取操作只需在一个节点上就可以完成,避免了索引信息和数据内容分离导致跨节点访问,因此减少了客户端与集群数据节点之间的交互次数,大大降低了网络通信量和访问延迟。

4 热点时空对象分布式缓存机制

当高并发用户访问时,若直接从 Hadoop 分布式文件系统中存取数据势必需要大量的磁盘访问,导致很大的 I/O 访问延迟。由于大部分空间应用的数据访问遵循“读多写少”的存取模式,因此,本文提出时空对象分布式缓存机制将常被访问的热点时空数据驻留在分布式内存池以降低访问延迟。当客户端请求数据时,先从分布式缓存中读取相应的数据对象,如果命中则将数据内容传输至客户端;若缓存命中失效,则从分布式文件系统的磁盘数据块中读取所需数据,并将邻近区域内的数据对象也预加载至分布式缓存,从而支撑高并发用户实时访问。

extHDFS 的分布式缓存机制是基于内存池技术构建的对象缓存系统,它在集群中的每个节点上开辟一块大小可配置的内存区域作为内存池,再用网络将所有内存池连接形成大缓存。extHDFS 缓存机制对上层应用透明,客户端只需要调用全局统一的存取接口,而不必关心数据对象所在的具体驻留位置。具体实现中采用内存型键值存储系统如 Memcached 作为对象缓存服务端,同时根据空间应用模式在客户端实现数据分布策略,即将时空对象以键值对的形式利用一致性哈希算法均匀分布至集群节点内存池。extHDFS 热点时空对象缓存机制主要包含三个部分:时空热点模型、缓存对象生命周期管理策略和本地化缓存策略。时空热点模型用于确定热点空间区域和热点时空对象,该模型通过空间密集度与对象访问频率两个参数加权确定热点区域,再将热点区域内的数据对象全部预取至分布式缓存。缓存对象生命周期管理策略用于加载和替换对象。本地化缓存策略用于降低网络传输延迟,它将本地的热点数据块从磁盘加载至本地内存池中;若本地内存池容量不够,再通过网络传输至其它节点内存池中;若其余节点内存容量不够,则采用 LRU 策略替换本地内存中的数据。相对于海量数据规模而言,单个节点内存池容量相对较小,因而分布式缓存总容量也十分有限。为有效确定热点空间区域,将热点对象预取至分布式缓存,本文提出了一种基于空间区域要素数目和对象访问频率加权的热点数据模型。该模型借鉴空间插值思想将热点对象与邻近的 8 个对象,即将当前热点对象及其 Hilbert 次序后八个对象一起加载至缓存。由于时空对象的“热

度”会随时间推移而发生变化,所以需要将热点对象动态地载入缓存或从缓存中换出至磁盘。

假设整个集群分布式缓存总容量为 c 兆字节(MB),时空对象大小为 s 千字节(KB),其长度范围为[min,max],即 $\min \leqslant s \leqslant \max$;$\chi$ 表示访问频率在前 χ 位的对象;μ 表示当前对象的邻近对象数目,缓存对象热点模型由下式确定:

$$c = \frac{\chi \times (\mu + 1) \times s}{1024}, \min \leqslant s \leqslant \max \qquad (4)$$

根据时空热点模型确定热点对象后,其邻近的 μ 个对象也会被动态地加载至分布式缓存。空间应用中的时空对象如遥感瓦片或 GPS 轨迹片段,大部分在 1KB 至 1024KB 之间,即 $1 \leqslant s \leqslant 1024$,因此将“热度”排名在前面的 χ 个对象作为热点数据,根据所配置的分布式缓存容量可以动态调整确定 χ 值和 μ 值。假设由 16 个节点构成的集群,每个节点开辟 512MB 空闲内存池,则整个分布式缓存容量 c 为 8192MB;若将 μ 值设为 8 则表示将热点对象邻近的 8 个对象预取至缓存,理论上可以加载 9 280 至 932 065 个热点区域对象至缓存中,即表示可以将热度排名在前 χ($4640 \leqslant \chi \leqslant 464000$)的热点对象及其邻近 8 个对象全部加载至分布式缓存,此时分布式缓存中共有对象数目为 83 520 至 8 388 585 个。

由于数据块内的时空对象数目是确定的,因此在集群数据节点上分别启动一个后台线程跟踪客户端的对象访问行为,并记录日志条目 <对象坐标:对象 *VC* 值:访问频次>,定期统计对象在该时间段内的访问频率,最后按照访问频率计算对象的热度以进行热点排序。由于同一空间区域内的数据对象具有唯一的 VC 值,而且具有相同的编码前缀(区域编码),因此计算排好序的时空对象 VC 值的最大公共前缀就可确定热点空间区域,即将具有相同最大公共前缀的空间对象所在的区域作为热点区域,且最大公共前缀编码就是区域编码 VC 值。通过后台线程周期性统计对象访问频率,根据访问频率计算时空对象热度和热点区域,并动态地调整热点对象和热点区域排名。利用 LRU 算法周期性地替换驻留在缓存但热度排名下降的区域,一旦确定替换某个区域,则将该区域内的对象全部替换,并将新的热点区域内的对象加载至分布式缓存。由于集群系统中的网络带宽是稀缺资源,为减少分布式文件系统与分布式缓存之间的数据传输量,并尽量避免数据对象跨节点传输,extHDFS 的本地化缓存策略将本地数据块内的热点对象优先放置在本地内存池,当本地缓存容量不足时则将数据通过网络传输至属于同一机架的其它节点内存池。

5 实验结果分析与性能评估

5.1 实验测试平台

实验物理集群由 8 台 Dell PowerEdge 服务器组成(Node1 - Node8),每个节点硬件配置为 two quad-core Intel CPU 2.13GHZ,4GB DDR3 RAM,15000r/min SAS 300GB 硬盘。每个节点均部署 Linux CentOS 5.5(内核 2.6.18),JDK-1.6.0_35,Memcached server-1.4.15。云平台 Hadoop-1.0.3、HBase-0.94.1、Cassandra-1.1.5 和 MySQL-5.5.28 cluster 均部署在 8 个节点集群上,Zookeeper 部署在 7 个节点上用于选举 HBase 主节点,Node3 为 Hadoop 主节点,Node6 为 MySQL 集群主节点,工业级压力测试标准套件 Loadrunner-9.10 部署在 Windows 2003 服务器上。extHDFS 的对比系统主要包括原始 HDFS(orgiHDFS)、数据库管理系统 MySQL cluster 以及两个典型的 NoSQL 数据库有对等结构的 Cassandra 和主从式结构的 HBase。

5.2 测试数据集与测试条目

测试数据集包含栅格数据集、矢量数据集和 GPS 轨迹数据集,数据总规模约 1.21TB。栅格数据集为若干城市地图与遥感数据,共计 126 982 186 个瓦片对象,其中大约有 93.8% 的瓦片大小在 1KB 至 100KB 之间。矢量数据集包含 98 123 686 个要素对象。GPS 轨迹数据集包含 3 293 372 个移动点对象。测试条目分为两大类:I/O 存取和系统综合处理性能。I/O 存取性能测试包括空间应用中的 4 个常用操作:批量加载、顺序写入、随机读取、顺序读取;系统综合处理性能测试包括系统并发事务处理性能、系统响应与错误恢复性能、系统吞吐率扩展性能。

5.3 时空数据 I/O 存取性能

5.3.1 批量加载性能

实际空间应用通常需要将时空数据批量入库,本实验将 3 组数据集分别批量加载至 6 个对比系统中,并测试相应的执行时间。采用系统提供的接口加载测试数据,其中存入 extHDFS 的时间包括建立索引的时间,而存入其它系统则不包括索引建立时间。引入批量加载性能因子 λ,其中 λ = 加载数据量(GB)/批量加载执行时间(min),加载因子 λ 越大说明系统的批量加载性能越好。

图 6 所示的是分别加载 64GB、256GB、1024GB

数据至不同系统中的执行时间，extHDFS 的批量加载性能均优于其它系统。当数据量比较小（64GB）时，extHDFS 的性能优势不太明显，主要原因是其它系统的 I/O 性能未达到瓶颈，extHDFS 数据分布过程中的网络传输延迟抵消了其部分并行 I/O 优势，其加载因子 λ 为其它系统的 1.2 ~ 3.7 倍。获益于时空数据组织和分布机制，当数据量越大时，extHDFS 的优势越明显，其它系统的执行时间均随数据量呈线性增加，而 extHDFS 的加载时间要少得多。当数据量为 1024GB 时其加载因子 λ 是其它系统的 2.9 ~ 10.8 倍。因此 extHDFS 的并行 I/O 能力可有效支持空间应用中大数据批量加载需求。

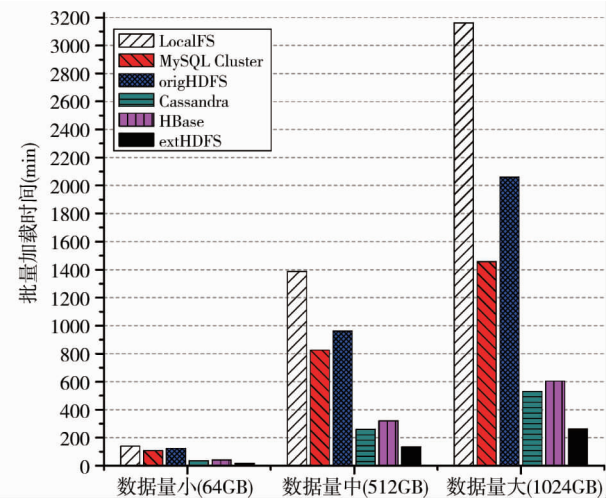

图 6 批量加载性能

5.3.2 顺序写入性能

空间应用中多数用户通常只关注特定区域内的数据，因此支持众多并发用户将小数据顺序写入至存储系统中是一项重要功能。顺序写入操作首先遍历主节点中的全局索引和数据节点中的局部索引查找数据的插入位置，然后顺序写入数据到相应主副本节点与备份节点上。本文采用 $MBR(x,y)$ 表示区域，其中 (x,y) 表示二维空间坐标范围。

图 7 给出了不同系统顺序写入 6 个不同大小区域内的时空数据的性能对比。顺序写入小区域时，MySQL 集群性能比原始 HDFS 好，而写入大区域时则相反，主要由于 MySQL 的复杂事务处理机制损耗了部分存储性能。从所有情况看，extHDFS 的顺序写入性能最好，其次是 Cassandra 和 HBase；extHDFS 的平均写入性能比 Cassandra 高 12.9% ~ 75.3%，比 HBase 高 16.2% ~ 83.1%，主要由于 extHDFS 可将时空邻近对象组织到同一个块分区中，因此先将时空对象顺序写入缓冲区后再把整块分区数据刷新至磁盘中持久存储，提高了数据写入性能。

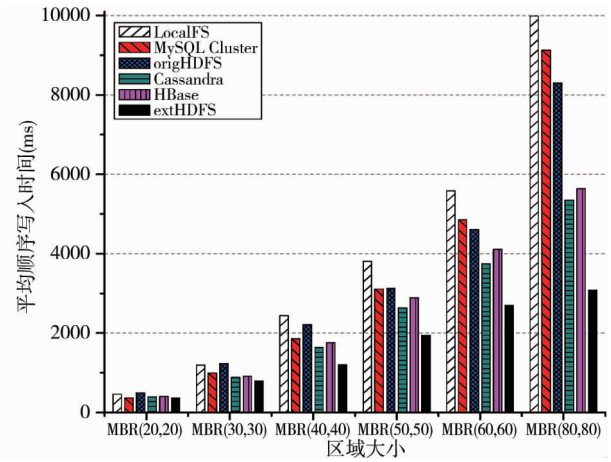

图 7 顺序写入性能

5.3.3 随机读取性能

空间应用中常见的两种数据访问模式包括顺序读取和随机读取。顺序读取的特点是一次性读取较大时空范围内的数据，而随机读取的特点则是每次只访问小范围内的数据或指定位置的时空对象。不同场景需要不同的访问模式，因此提升顺序读取和随机读取性能以满足不同应用场景下的数据访问需求十分重要。本实验引入矩形访问窗口，用 MBR（row，column）表示行长度为 row、列宽度为 column 的窗口，当窗口比较小时为随机读取，如 MBR(1,1) 表示随机读取一个时空对象；当窗口比较大时则为顺序读取，如 MBR(80,80) 表示长度和宽度均为 80 单位的区域。

图 8 显示了不同系统访问 6 组小窗口时的随机读取性能。由图可知，extHDFS 具有比较明显的性能优势，它的平均随机读取性能比原始 HDFS 提高了 1.39 ~ 2.17 倍，比 MySQL 集群提高了 22.8% ~

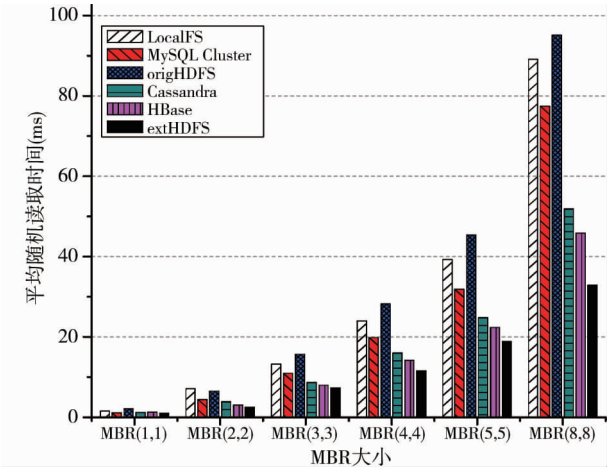

图 8 随机读取性能

98.1%,比 Cassandra 提高了 21.7% ~58.2%,比 HBase 提高了 18.9% ~47.6%。原始 HDFS 由于缺乏索引其随机读取性能甚至比单机文件系统要慢,MySQL,Cassandra 和 HBase 的性能不如比 extHDFS 的主要原因是它们的数据分布策略未考虑时空临近特性,而且缺乏缓存机制,造成了较大的访问延迟。extHDFS 由于具有时空索引和缓存机制,当定位到访问区域内容后,将其邻近范围内的数据预取至缓存中,避免了磁盘访问,从而快速返回所请求的时空对象集合。

5.3.4 顺序读取性能

本实验总共做了 6 组顺序读取操作性能测试,图 9 所示的是顺序读取 6 组不同区域内时空数据的执行时间。由实验结果可知,extHDFS 的顺序读取性能在所有测试用例中均优于其它系统,而且当访问区域越大时,其性能优势越明显。当顺序读取窗口 MBR(20,20)至 MBR(80,80)内数据时,extHDFS 比原始 HDFS 提高了 1.27 ~2.13 倍。MySQL cluster 由于其 I/O 吞吐性能较差,加上其强事务处理机制以及 SQL 解析开销,其性能略逊于原始 HDFS。当访问区域较小时,Cassandra 性能优于 HBase,而 extHDFS 比 Cassandra 性能提高了 82% ~97%。当区域大于 MBR(50,50)时,HBase 的顺序读取性能明显高于 Cassandra,此时 extHDFS 比 HBase 提高了 0.89 ~1.18 倍。从 TB 级数据中顺序读取数据时,其它系统由于数据均需通过中心节点传输,加上复杂的锁机制,造成在高并发用户访问越大的区域时,其性能呈下降趋势,而 extHDFS 具有并行 I/O 性能优势,通过将数据均匀分布于集群中可将顺序访问负载均摊至不同节点,而且它的副本策略使不同用户所请求的相同数据可在多个节点上获取,避免单个节点访问压力过大造成单点失效问题,并确保系统发生故障时能从错误状态中迅速恢复提供数据访问服务。

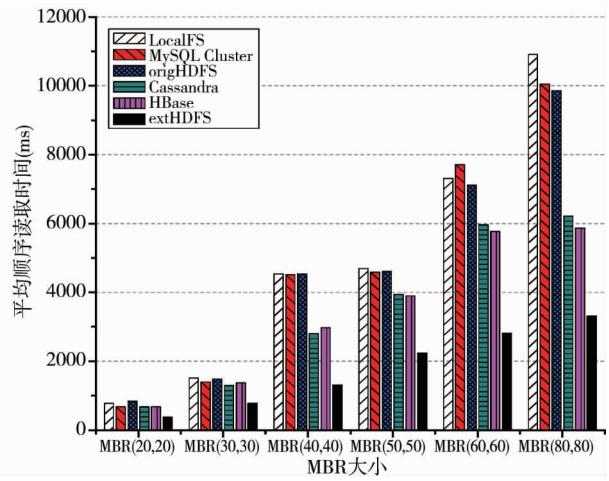

图 9 顺序读取性能

5.4 系统综合处理性能

5.4.1 系统并发事务处理性能

事务处理性能是衡量在线系统服务能力的重要指标之一,本实验用 Loadrunner 测试并发用户数从 20 增至 180 时不同系统运行 300min 所通过的并发访问事务数。从图 10 所示的系统事务处理性能对比图可以看出,具有时空扩展功能的 extHDFS 的事务处理性能优势明显,它处理的事务数几乎随用户数的增多而呈线性增长。基于传统文件系统与 DBMS 集群的方法在持续高并发用户请求下,其访问延迟变大,所通过的事务数呈下降趋势。由于原始 HDFS 的随机存取性能较差,Cassandra 和 HBase 未对时空数据存储访问特点进行优化,它们的事务处理性能皆逊于 extHDFS。从实验结果看,extHDFS 比其它系统的时空事务处理性能提高了 20% 至 90% 左右。因此,extHDFS 能够支持高并发用户访问时空数据服务。

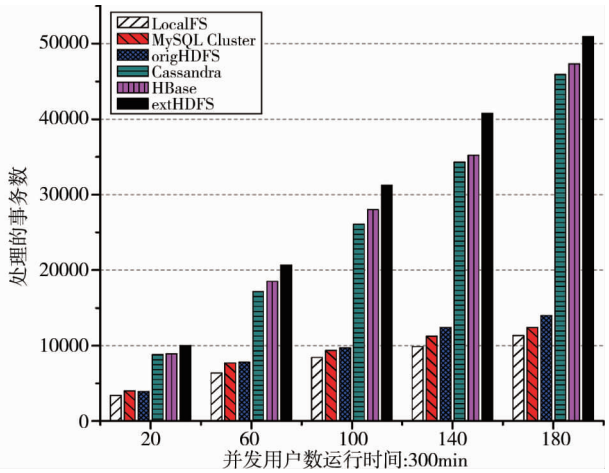

图 10 系统并发事务处理性能

5.4.2 系统响应与错误恢复性能

在线空间应用具有数据密集型和 I/O 密集型两大特点,因此系统响应性能优劣直接影响到服务质量。本实验通过 Loadrunner 加压访问负载测试 100 个并发用户访问不同系统中的时空数据时的响应性能,测试时间为 300min,每隔 60min 记录各个系统的平均响应时间。

图 11 显示了不同系统的平均响应性能,extHDFS 的响应时间稳定在 0.8 ~1.2s 之间;而其它对比系统的响应时间均在 1.5 ~3.2s 之间波动,且随系统运行时间变长,它们的响应延迟变大。由于 extHDFS 的分布式缓存机制将热点数据对象预加载

至内存池中,客户端从缓存中访问数据避免了磁盘 I/O 存取操作,因而能够实现低延迟访问,测试结果表明,extHDFS 的响应性能比其它系统提高了 67% 至 158.7% 左右,其平均响应时间均在毫秒级。当发生节点失效或者其它错误时,extHDFS 会自动将失效节点上的服务请求重定向至备份节点上访问副本数据,实验中采用人为方法关闭服务器和切断网络模拟节点发生故障,结果表明 extHDFS 可在秒级时间恢复响应并继续提供数据访问服务。因此,extHDFS 不仅能满足在线空间应用低延迟响应要求也具有良好的错误处理能力。

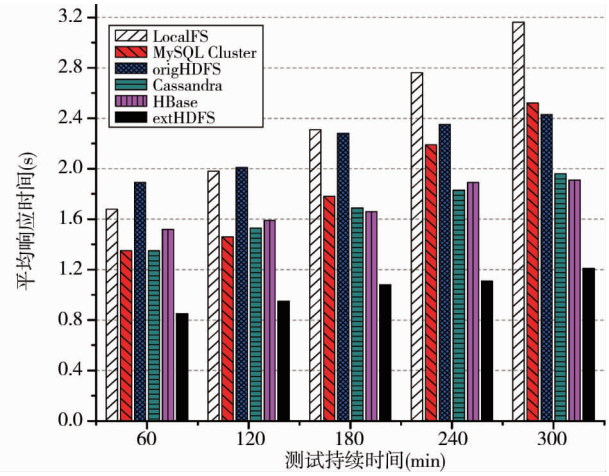

图 11 系统响应性能

5.4.3 系统吞吐率扩展性能

本实验测试当 100 个并发用户访问时,系统吞吐率随集群规模变化的扩展性能。如图 12 所示,相对单机文件系统的基准线而言,各个系统的吞吐率均随集群规模的增大而有不同程度地提高。MySQL 集群和原始 HDFS 的吞吐量增幅较少,均不到 9%,原因是它们的 I/O 存取能力不足,而且不完全匹配空间应用访问模式,系统阻塞导致处理的请求数目逐渐减少,使得系统的吞吐率增幅变缓甚至呈下降趋势。在 5 个节点以下时,Cassandra 的吞吐性能要优于 HBase,而当集群节点数目超过 5 时,HBase 的吞吐率超过了 Cassandra,主要是由于 HBase 继承了底层 HDFS 的存储优势,而 Cassandra 由于是对等(P2P)结构,其额外通信开销过大导致其吞吐性能在集群规模越大的情况下反而下降。由于 extHDFS 专门针对时空数据特性进行了优化使之符合空间应用存取模式,在集群节点数目由 1 增至 8 时,它的吞吐性能比 HBase 提高了 21.8% 至 68.3%,比 Cassandra 提高了 12.8% 至 73.9%。extHDFS 的具有良好的随机读取性能和扩展性,其吞吐性能随集群规模呈线性增长趋势,其加速效率(加速比/节点数)达到了 49.7% ~71.6% 左右。因此,extHDFS 的高扩展能力可确保系统的吞吐性能和可用性。

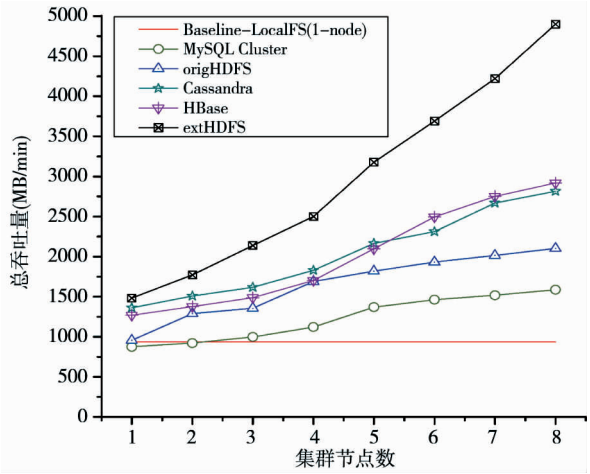

图 12 系统吞吐量性能

6 结 论

针对时空数据规模呈爆炸性增长和高并发用户访问在线空间应用所造成的时空数据存储性能瓶颈问题,提出了大规模时空数据分布式存储方法和地理位置感知的时空数据切分与布局机制、时空对象重组织机制和热点时空对象分布式缓存机制。利用该方法实现了基于 Hadoop 云平台的时空数据分布式存储中间件原型系统 extHDFS。实验结果表明,extHDFS 在大数据存储和高并发用户低延迟访问方面比典型的 NoSQL 数据库和空间数据库具有更优异的性能表现,它的时空数据 I/O 存取性能、并发事务处理性能、系统响应与错误恢复性能、系统吞吐率扩展性能均优于其它对比系统。因此,本文方法中的三个关键机制共同作用能高效地满足数据密集型在线空间应用的数据存储需求。下一步将研究频繁更新场景下的并行时空查询处理难题。

参考文献

[1] Coleman D, Georgiadou Y, Labonte J. Volunteered geographic information: The nature and motivation of produsers. *International Journal of Spatial Data Infrastructures Research*, 2009, 4: 332-358

[2] Sarwat M, Bao J, Eldawy A, et al. Sindbad: a location-based social networking system. In: Proceedings of ACM International Conference on Management of Data, SIG-

MOD'12, Scottsdale, Arizona, USA, 2012. 649-652
[3] Apache Hadoop. http://hadoop.apache.org/.
[4] Güting R. H. An introduction to spatial database systems. The VLDB Journal, 1994, 3(4): 357-399
[5] 张书彬,韩冀中,刘志勇等. 基于 MapReduce 实现空间查询的研究. 高技术通讯. 2010, 20(7): 719-726
[6] Shvachko K, Kuang H, Radia S, et al. The Hadoop Distributed File System. In: Proceedings of the 26th Symposium on Mass Storage Systems and Technologies, MSST'10, 2010. 1-10
[7] Condie T, Conway N, Alvaro P, et al. MapReduce Online. In: Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, NSDI'10, San Jose, CA, USA, 2010. 21-36
[8] Azza A, Kamil B, Daniel A, et al. HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. *The VLDB Journal*, 2009, 2(1): 922-933
[9] He Y, Lee R, Huai Y, et al. RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems. In: Proceedings of the 27th IEEE International Conference on Data Engineering, ICDE'11, 2011. 1199-1208
[10] Samet H. Decoupling partitioning and grouping overcoming shortcomings of spatial indexing with bucketing. *ACM Transactions on Database Systems*, 2004, 29(4): 789-830
[11] 刘旭辉,韩冀中,韩承德等. 基于集群系统的空间数据并行处理策略研究. 高技术通讯, 2009, 19(10): 991-997
[12] Zhong Y, Sun S, Fang J, et al. A novel method to manage very large raster data on distributed key-value storage system. In: Proceedings of the 19th IEEE International Conference on Geoinformatics, Hongkong, China, 2011. 1-7
[13] Mouza C, Litwin W, Rigaux P. Large-scale indexing of spatial data in distributed repositories: the SD-Rtree. The VLDB Journal, 2009, 18: 933-958
[14] Liao H, Han J, Fang J. Multi-dimensional index on Hadoop distributed file system. In: Proceedings of the 5th International Conference on Networking, Architecture and Storage, Macau, China, 2010. 240-249
[15] Nishimura S, Das S, Agrawal D, et al. MD-HBase: a scalable multi-dimensional data infrastructure for location aware services. In: Proceedings of the 12th IEEE International Conference on Mobile Data Management, Lulea, Sweden, 2011. 7-16
[16] Justin L, Mokbel M, Khalefa M. The CareDB context and preference-aware database system. In: Proceedings of the 37th International Conference on Very Large Data Bases, VLDB PersDB Workshop, 2011. 3(2): 1529-1532
[17] Bao J, Mokbel F, Chow C. GeoFeed: A location-aware news feed system. In: Proceedings of the 28th International Conference on Data Engineering, ICDE'12, 2012. 54-65

A distributed storage scheme for Big spatio-temporal data

Zhong Yunqin[*,**], Fang Jinyun[*], Zhao Xiaofang[*]
([*] Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)
([**] University of Chinese Academy of Sciences, Beijing 100190)

Abstract

A novel scheme for distributed spatio-temporal data storage based on the Hadoop cloud platform is proposed to solve the storage bottleneck problems of current main spatio-temporal data storage methods such as low efficiency in online geospatial applications and service interruption. The scheme has three significant mechanisms. Firstly, an efficient data partitioning and placement approach is introduced to distribute big data across cluster nodes evenly and to guarantee load balancing. Secondly, a spatio-temporal data object reorganization approach is adopted to improve the geographic proximity and to meet the access patterns of geospatial applications. Thirdly, a distributed hotspot object caching approach for frequently accessed spatio-temporal data is used to reduce disk I/O access latency. The exHDFS, an intermidiate prototype system based on the Hadoop cloud platform for distrilouted spatio-temporal data storage, was designed and implemented by using the new scheme. The results of the comprehensive experiments show that the extHDFS outperforms the comparisons and thus it could meet the storage requirements of data-intensive geospatial applications.

Key Words: cloud platform, spatial application, data management, spatio-temporal data storage, spatio-temporal index, spatio-temporal data caching