文章编号：1671-1742(2015)01-0052-07

一致性哈希算法在数据库集群上的拓展应用

赵 飞，苏 忠
（空军指挥学院网络中心，北京 100097）

摘要：在数据库集群的研究中，可扩展性是一个重要的性能指标。为实现在数据高速增长或部分集群服务器故障情况下，数据依然能够快速、可靠、安全的分布到新的集群服务器节点上的目的，就必须合理设置数据划分的策略。将Key-Value存储技术中使用的一致性哈希算法思想借鉴运用到并行分析型数据库集群中，提出针对大规模结构化类特殊数据的一致性哈希划分方法，并在MapReduce框架下设计具体的数据划分方案。最后，以TPC-DS作为测试基准，与同类系统进行性能对比测试，实验结果表明方案不仅有良好的划分性能，且扩展性也较好。

关 键 词：计算机应用技术；数据库；并行分析型数据库集群；一致性哈希算法；数据划分
中图分类号：TP311.133.2 **文献标志码**：A

0 引言

数据库集群处理系统中，数据分布存储在多个节点上，为提高系统的并行处理数据的性能，需要采用合适的数据划分策略（算法）对数据进行划分，并合理地将数据分布在集群的各个节点上。数据划分策略不仅要从数据访问和处理的性能方面考虑，还要从数据增长以及集群的可扩展性方面考虑，以便实现高性能、低成本、高可扩展性的分布式数据存储与管理。

1 数据划分的基本概念

数据划分（data partition）是一种分而治之的技术，通过将大数据对象划分成众多可管理的小块，从而为大数据的存储管理提供可伸缩的性能[1]。在并行分析型数据集群中，数据表往往很大，对于这些表，可按照某个或者某些字段水平地划分若干段，每个段称为一个分区，划分所用的字段称为分区键，根据划分算法和分区键则可确定表中某个行所在的分区。目前相关研究中的数据划分算法可分为3种基本的类型。

（1）范围划分，这是一类最基本的划分算法，即按照表中一个或一组字段的值的范围决定表中元组所属的分区。虽然简单，范围划分却在很多情况下非常有效，在MySQL、Oracle等关系数据库管理系统以及很多数据仓库、分布式数据库、并行数据库管理系统中都有对范围划分的支持。

（2）列表划分，针对分区键是离散值且数据重复率较高的场景，将分区键上值相同的元组划分到相同的分区，并对分区键的离散值建立分区列表。列表分区的优势在于可以迅速定位分区键上的离散值存储位置，等值查找效率高，但列表分区的通用性较差，且当数据在分区键上分布不均匀时出现严重的数据倾斜。

（3）哈希划分，采用分区键和哈希函数确定分区的算法。在哈希划分中，首先将分区编号，通过哈希函数计算分区键上的哈希值，并将哈希值映射到各个编号的分区下，从而实现元组的划分。哈希划分的过程由系统自动完成，不需要用户去指定如何分区，用户需要确定的仅仅是分区键和分区的数量，相对于前两种划分方法，哈希划分的操作性较强。

2 一致性哈希算法

在数据划分中，无论何种划分算法，分区键的选取都非常重要。对数据进行分区时，需要对被分区数据的各个维进行特征分析，从而选取合适的分区键和划分算法。在以上3类划分算法中，哈希划分的相关研究较多，具有代表的是在基于内存的Key-value存储引擎中，采用一致性哈希算法进行数据划分[2]，而最早的系统实现则为Amazon Dynamo[3]。

一致性哈希算法（Consistent Hash）最早由David Karger等[2]年提出，当时主要是为了应对互联网中的热点（Hot Pot）而设计，一致性哈希解决了简单哈希算法在分布式哈希表（Distributed Hash Table，DHT）中存在的动态伸缩等问题。

在一致性哈希算法中，将哈希值空间（哈希值范

收稿日期：2014-12-26
基金项目：国家自然科学基金资助项目（61071065）

围)看做一个虚拟的环形区间，以哈希值范围为 0～$(2^{32}-1)$的哈希函数 H 为例，其逆时针方向组织的环形哈希空间如图 1 所示。

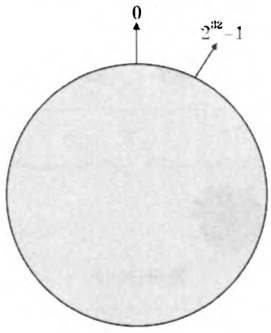


图 1　环形哈希空间

在分布式哈希表问题中，一致性哈希算法被用来确定数据应该存储在哪个 Cache 中。算法中，将 Cache 的标识通过哈希函数计算得到哈希值 $hc = H(\text{Key}_{\text{Cache}})$并映射到环形哈希空间上，并将环形的哈希空间中的两个 hc 之间的哈希空间分配给其中的一个 Cache。

在判断数据存储在哪个 Cache 的过程中，首先用数据对象(object)的 Key 值代入哈希函数，得出哈希值 $h = H(\text{Key}_{\text{obj}})$。然后根据 h 在环形哈希空间中的对应位置得出 Key 对应的数据对象应该存放在哪一个 Cache 中。如图 2 所示为 4 个数据对象(Key 值分别为 Key1～Key4)在 3 个 Cache 的情况下通过一致性哈希算法得出数据分布的示意图，其中，object1 的哈希值 Key1 在 Key A 和 Key C 之间，故其分配到 Cache A 中，object2 到 object4 依次类推分别对应到其他 Cache 中。

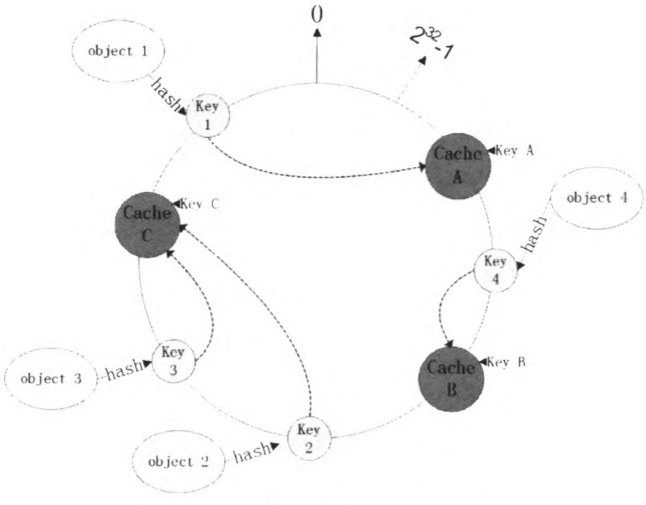


图 2　一致性哈希算法数据分布示意图

假如图 2 中所示的 3 个 Cache 分别存储在 3 台服务器上，那么 4 个数据对象就被划分存储到了这 3 台服务器中。如果要增加服务器，即增加 Cache 时，只需计算新增 Cache 的哈希值 hc 并映射到哈希空间中，将原有的哈希空间切分出一部分给新的 Cache 即可，如图 3 所示，当加入 Cache D 时，Cache C 与 Cache B 之间的哈希空间就被一分为二，原来对应到 Cache C 的数据对象 object2 就按算法划分到新增的 Cache D 上。

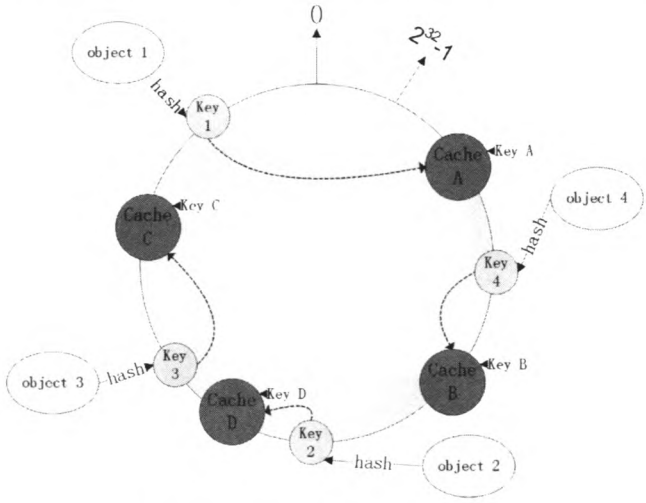


图 3　一致性哈希增加 Cache 示意图

减少 Cache 的操作过程则相反，需要将原有的两个 Cache 对应的哈希空间合并。

一致性哈希算法解决了分布式哈希表动态伸缩的问题，但以上的基本一致性哈希算法容易导致负载不均衡，可能出现个别 Cache 对应过多数据对象的情况，如图 4 中所示的 Cache C，多个数据对象集中聚集在一个 Cache 上。

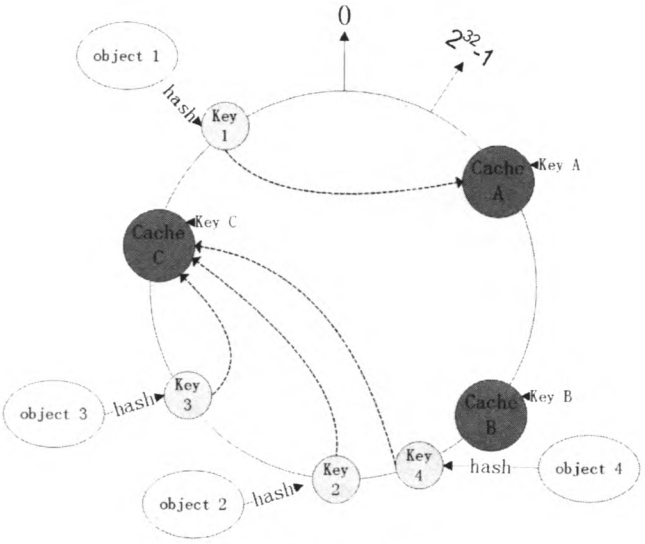
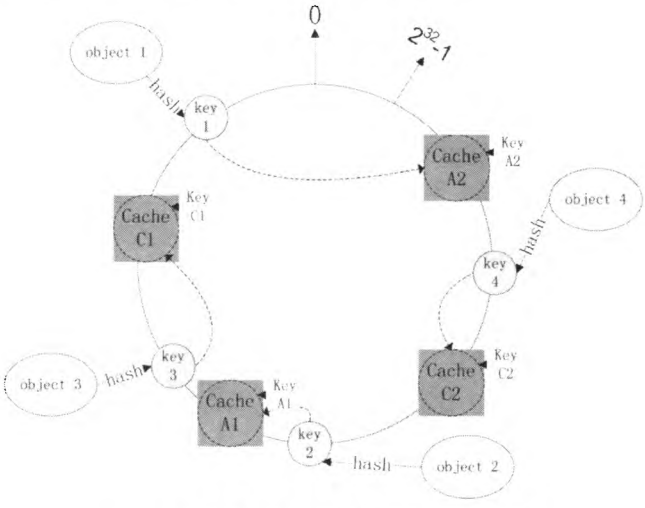


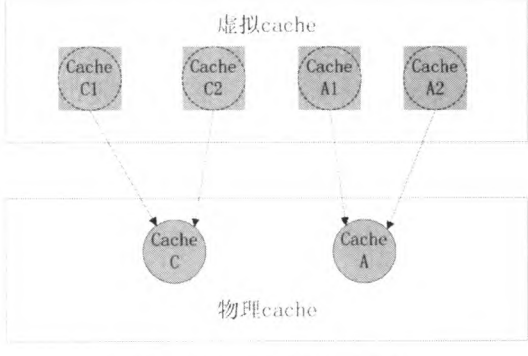
图 4　分布不均衡的示意图

为了解决负载均衡问题，在实际应用中对一致性哈希算法进行了改进，增加了虚拟节点。所谓虚拟节点，就是增加很多虚拟的 Cache，虚拟 Cache 的数量远

大于服务器的数量，将数据对象通过哈希函数 H_1 映射到哈希空间中的虚拟 Cache 上，之后在通过虚拟 Cache 与服务器之间的映射函数 H_2 映射到物理服务器上进行存储。如图 5(a)所示为引入虚拟节点之后，数据对象映射到虚拟节点的情况，图 5(b)所示为数据再次通过映射表映射到物理服务器上的情况。



(a) 引入虚拟节点后，数据对象到虚拟节点的映射

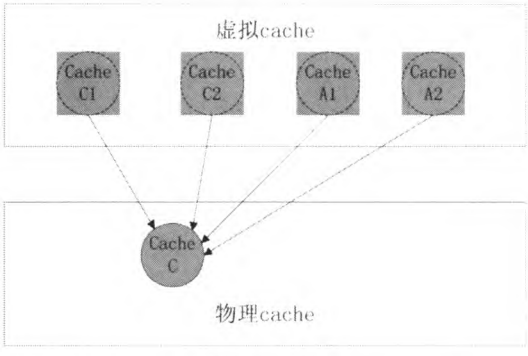


(b) 虚拟节点映射到物理节点

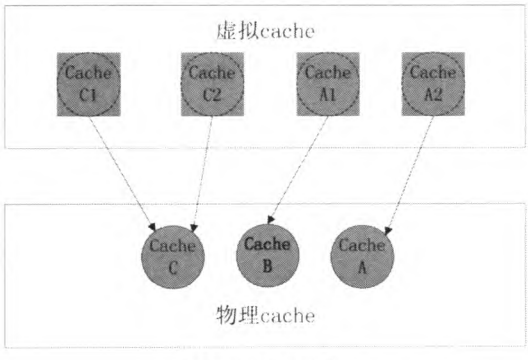
图 5 引入虚拟节点后数据划分示意图

传统的哈希算法通过将哈希函数计算的哈希值对分区数取模，从而将数据划分到各个分区中，这样划分存在容错性和可扩展性的问题。当哈希空间中增加新的分区或者减少分区时，就需要将已经被划分的所有数据重新哈希划分一遍，才能保证分区的正确性和通用性。

在一致性哈希算法中，数据对象按照哈希值被连续地划分到虚拟的节点(分区)中，虚拟节点再次通过映射表或者另外一个哈希函数对应到物理服务器上。当分布式存储系统增加物理节点时，只需修改虚拟节点到物理机器的映射表或者哈希函数，将一部分虚拟节点映射到新增的物理节点即可，而当减少节点时，则只需修改虚拟节点到物理节点的映射表或者哈希函数，将原本映射到被删除物理节点上的虚拟节点重新映射到删除后的其他物理节点即可，增加和删除物理节点的操作如图 6 所示。



(a)增加物理节点



(b)删除物理节点

图 6 一致性哈希算法中物理节点伸缩原理

相对于传统哈希算法，一致性哈希算法的优点体现在如下方面：

(1)可扩展性。一致性哈希算法保证了增加或减少服务器时，数据存储的改变最少，相比传统哈希算法大大节省了数据移动的开销。

(2)更好地适应数据的快速增长。采用一致性哈希算法分布数据，当数据不断增长时，部分虚拟节点中可能包含很多数据、造成数据在虚拟节点上分布不均衡，此时可以将包含数据多的虚拟节点分裂，这种分裂仅仅是将原有的虚拟节点一分为二、不需要对全部的数据进行重新哈希和划分。虚拟节点分裂后，如果物理服务器的负载仍然不均衡，只需在服务器之间调整部分虚拟节点的存储分布。这样可以随数据的增长而动态的扩展物理服务器的数量，且代价远比传统哈希算法重新分布所有数据要小很多。

目前，一致性哈希算法主要应用在分布式 Key-Value 存储系统中。而分布式 Key-Value 存储系统属于 NoSQL 数据库研究范畴[4]，主要用于存储高并发的事务型数据，将数据按其 Key 值的不同，存放在不同的服务器上，例如大型网站通常使用 Key-Value 存储系统存放页面的缓存，通过将一些变化不频繁、但会

被大量重复访问的页面放入缓存,从而减少了 HTTP 服务器对磁盘和后台事务型数据库(MySQL、Oracle 等)的访问压力。

3　一致性哈希算法在并行分析型数据库集群中的拓展应用

在数据库中研究中,关于结构化数据的分析称为分析型数据库,其研究开始于 20 世纪 90 年代,研究者提出了数据仓库的概念[5]。数据仓库为联机分析处理(OnLine Analytical Processing,OLAP)[6]和决策支持系统(Decision Support System, DSS)[7]提供结构化的数据环境,其主要作用是从 OLAP 系统和其他多种数据源中收集和存储结构化的数据并对上层的分析和挖掘任务提供支持。传统数据仓库系统主要是基于单机系统,像 IBM、Microsoft、Oracle 等公司都有相应开发的单机数据仓库系统,开源的单机数据仓库系统有 Infobright 等[8]。随着各行各业信息化程度的提高,尤其是基于智能平台的新媒体的不断深化和拓展,导致结构化数据在数据量方面呈爆炸式增长、产生的速度也不断加快,海量信息的情报分析难度在不断的加大。而由于基于单机的分析型数据库在扩展性、可靠性、性价比方面存在有诸多限制,已经很难满足大量结构化数据的查询分析需求,由此就逐渐出现了适应大规模数据分析的并行计算和集群系统,这里把针对大规模结构化数据分析的并行数据库集群称之为并行分析型数据库集群。

与分布式数据库系统一样,数据划分对于并行分析型数据库集群至关重要,而目前基于一致性哈希的数据划分算法在结构化大数据分析中的应用研究还没有。将一致性哈希算法的思想拓展应用在结构化大数据的数据划分中,设计和实现基于 MapReduce 的数据划分方案[9],并通过实验结果对比,验证数据划分方案的可行性。

3.1　基本划分方法

在结构化大数据划分中,对于大数据表,划分时选择一个分区键,分区键和元组间形成类似 Key-Value 的关系。对于分区键中包含字符串、日期/时间、二进制大对象、大整数等类型字段的情况,如果直接比较不同分区键值的计算复杂度较大,通常需要很多条 CPU 指令才能完成。这种情况下,可使用哈希函数计算出每个元组在分区键上的哈希值作为新的分区键,哈希值通常是固定长度的整型。对于分区键是基本数值型的情况,则将原分区键直接作为新的分区键。称新的分区键为"数值型分区键",称数值型分区键的值空间为"数值型分区空间",简称"分区空间"。

指定数值型分区键之后,即可将数值型分区空间划分为若干的分区,数据表中的元组按照数值型分区键被划分到分区当中,如图 7 所示。分区的概念类似于一致性哈希算法中的虚拟节点。

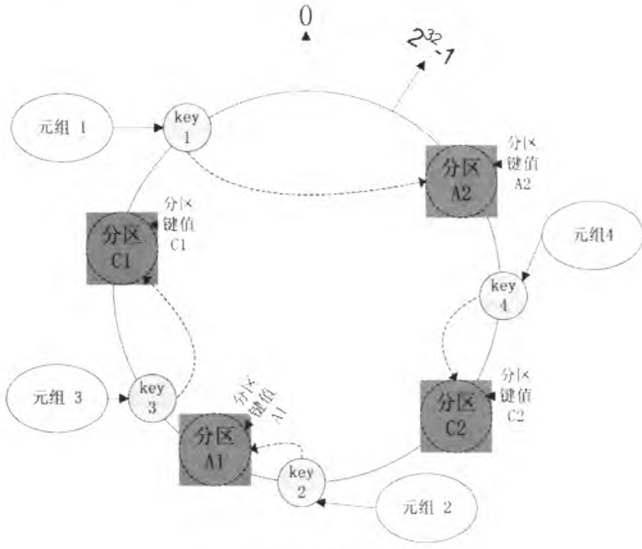


图 7　数值型分区空间及元组划分

在将分区空间划分为分区时,需要先对数值型分区键进行采样、估算出合适的分区边界,保证每个分区的大小基本均衡。之所以增加采样的过程,是因为在 Key-Value 存储系统中,Key 通常是自然界中随机性较好的值,而在结构化大数据分析中,所用的分区键并不一定是随机、均匀分布的,可能出现少数分区键对应大量数据元组的情况。划定分区之后,再将数据划分到各个分区中即可完成对数据元组的划分。

3.2　数据划分方案设计

确定分区方法后,需要设计具体的数据划分方案。目前的结构化大数据大量存储于 HDFS(Hadoop Distributed File System)中[10],HDFS 作为数据存储系统保证了存储的容错性和可扩展性,而 MapReduce 虽然不适合交互式和实时的查询处理,但却非常适合完成 ETL(Extract Transform and Load, 数据库管理系统中,数据进入存储引擎之前的抽取、转换格式和加载过程)任务[11]。文中,假设结构化大数据已经存在于 HDFS 中,并使用 MapReduce 来完成数据的采样和划分。

如图 8 所示,基于 MapReduce 的数据划分方案包括 4 个流程:

(1)输入划分所需的参数。参数包括采样数 n 和分区数 N。n 表示在 HDFS 上的数据表文件(按行存

储的文本文件，每个字段之间用特定字符分割，类似CSV文件格式）上总共采集多少行作为样本，N 表示数据划分最终要生成多少个分区（数据块），每个数据块的大小要基本均衡，这些数据块将被导入到集群的存储系统中。

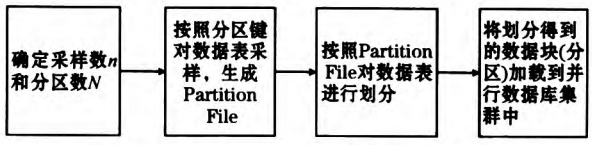


图 8 基于 MapReduce 的数据划分流程

(2)采样生成 Partition File。对于采集到的每个行，获取或计算出其数值型分区键。采样到 n 个分区键后，对其进行排序，并按照这 n 个分区键的值将分区空间划分为 N 段，每段中包含的分区键个数基本相同。将划分后的 $N-1$ 个分区端点写入 Partition File。

(3)启动一个 MapReduce Job 对数据表进行一致性哈希划分。MapReduce Job 按照 Partition File 中的分区端点将数据表文件划分为 N 个分区、即 N 个子文件，每个分区中都包含数值型分区键值连续的一部分数据元组。

(4)读取划分后的数据分块、加载到并行分析型数据库集群中。

其中数据采样的详细流程如图 9 所示。数据表文件存储在 HDFS 上，采样器通过自定义的输入格式对数据表文件进行采样。自定义的数据输入格式主要负责读取数据表文件数据块中的行（元组），并获取或计算出元组的数值型分区键，组成[分区键]→[元组]的 Key-Value 形式，供采样器处理。存储在 HDFS 上的数据表文件在物理上以很多 HDFS 文件块的形式存在，采样器会从每个 HDFS 文件块中采集 $c = n/BN$ 行作为样本，其中 BN 表示 HDFS 文件块的个数。采集样本之后，采样器会如上述第二步所述生成 Partition File。

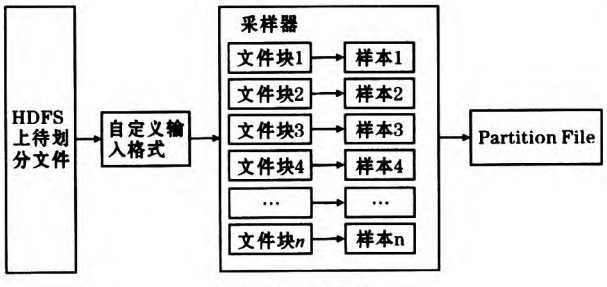


图 9 数据采样流程

产生 Partition File 之后，在上述第三步中，使用一个 MapReduce Job 实现对数据的划分。如图 10 所示。Map 节点通过自定义的输入格式从 HDFS 上的数据表文件中获取数据块并读取其中的数据，Map 读取到的数据是经过自定义输入格式处理后的[分区键]→[元组]的键值对形式，Map 将元组按照 Hadoop 所提供的 Total Order Partition 分区方式分发到 Reduce 端。Map 的个数等于数据表文件的数据块个数，而 Reduce 个数等于要划分的分区数 N。根据 Total Order Partition，每个 Reduce 收到的元组的数值型分区键都处于一个值连续的分区内，Reduce 将自己收到的元组写入分区文件中，数据划分的第三个步骤即完成。

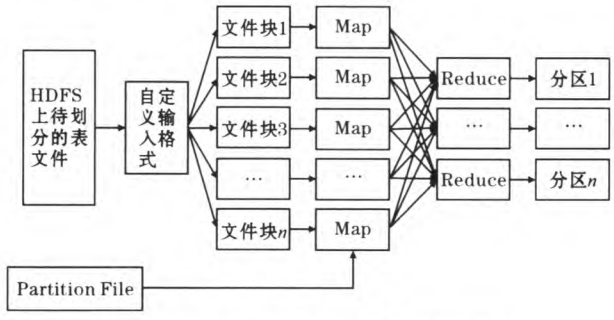


图 10 基于 MapReduce 的数据划分过程

最后，将所产生的分区文件分别下载到并行分析型数据库集群的各个节点上，并将其导入列存储引擎当中，数据划分即完成。

3.3 数据划分测试结果

设计和实现数据划分方案之后，将所实现的基于 MapReduce 的一致性哈希划分方法与同类系统（Presto[12]）中的数据划分方法进行了性能对比测试。

3.3.1 测试环境

数据划分集群配置如表 1 所示。

表 1 数据划分集群配置

Hadoop 版本	Apache Hadoop 1.2.1
Hadoop 集群节点数	1 个 Namenode，5/10 个 Datanode
划分的数据表文件大小	38.15GB（TPC-DS-100GB 数据集中 store_sales 表，为数据集中最大的一张表）
集群节点间的网络连接	千兆以太网

每个 Hadoop 节点的配置信息如表 2 所示。

表 2 Hadoop 单节点配置信息

CPU	4 核2.53 GHz
内存	16GB
磁盘	500GB，最大连续读写 70MB/s
网卡	千兆以太网卡
操作系统	CentOS Linux 6.4 64 位

3.3.2 测试方法

在 Presto 中 DDL 部分的功能依赖于 Hive 的实

现,所支持的数据划分与 Hive 是一致的[13]。Hive 没有直接提供对哈希划分和范围划分的支持,但在建表时可以指定一个字段作为分区键,导入数据时,用户可将分区键上值相同的元组导入一个分区中,相当于提供了对列表划分的支持。对于导入同一分区的数据,还可以按照某个字段的值进行聚簇和排序。但列表划分限制很多,仅在分区键具有少量离散值且数据在分区键上分布均匀时可用,为和文中的数据划分方法进行比较,利用 Hive 提供的分区支持,简洁实现了范围划分。

3.3.3　测试结果

测试采用 TPC-DS 作为测试基准,利用其生成器生成了 100GB 数据集,并采用其中最大的一张事实表(store_sales)作为测试数据,其数据量为 38.15GB。测试时,在 5 个 DataNode 和 10 个 DataNode 的两种规模的集群上对 Hive/Presto 和文中的数据划分方法进行了对比测试。结果图 11 所示。

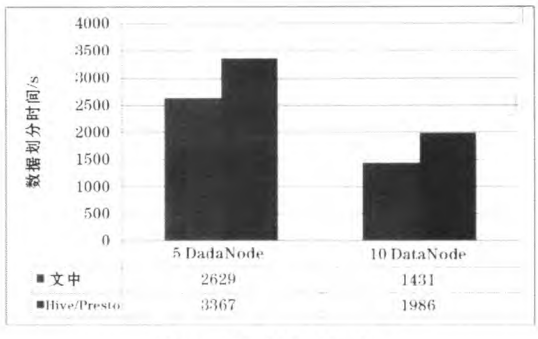


图 11　数据划分时间对比

从测试结果可以看出,在 5 个 DataNode 规模的集群中,文中的数据划分方法用时比 Hive/Presto 低 21.9%,当集群规模为 10 个 DataNode 时,文中数据划分方法用时比 Hive/Presto 低 27.9%. 可见文中的数据划分方法具有较好的性能,且随着集群规模增加,文中数据划分方法对相同规模数据的划分时间随之下降、扩展性较好。

当数据已经被划分时,如果集群中 DataNode 增加,由于文中的数据划分方法和 Hive/Presto 的划分方法都是将划分结果保存在 HDFS 中,因此扩展的代价是相同的。

4　结束语

将目前 Key-Value 存储技术中使用的一致性哈希算法的思想吸收借鉴运用到了并行分析型数据库集群的数据划分当中,提出了基于一致性哈希算法的数据划分思想,并通过 MapReduce 实现了大规模结构化数据划分方法;在具体数据划分方案的设计和实现中,充分利用了 HDFS 分布式存储的低成本、高可靠和高可扩展性以及 MapReduce 的离线数据处理能力,实现了数据按照分区键进行哈希划分。最后通过实验测试,验证所提出的数据划分方案在实际的结构化大数据划分中的可行性。

参考文献:

[1]　王珊,萨师煊. 数据库系统概论[M]. 北京:高等教育出版社, 2006.

[2]　Karger D, Lehman E, Leighton T, et al. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web[C]. Proceedings of the twenty-ninth annual ACM symposium on Theory of computing. ACM, 1997: 654 - 663.

[3]　DeCandia G, Hastorun D, Jampani M, et al. Dynamo: amazon's highly available key-value store [C]. ACM SIGOPS Operating Systems Review. ACM, 2007, 41(6): 205 - 220.

[4]　NoSQL DEFINITION[EB/OL]. http://nosql-database.org/.

[5]　王珊. 数据仓库技术与联机分析处理[M]. 北京:科学出版社,1998.

[6]　Greenfield D, Lyon G F, Vogl R. System and method for online analytical processing[P]. U.S. Patent 6,684,207,2004-1-27.

[7]　Ghodsypour S H, O'brien C. A decision support system for supplier selection using an integrated analytic hierarchy process and linear programming [J]. International journal of production economics, 1998, 56: 199 - 212.

[8]　Slezak D, Eastwood V. Data warehouse technology by infobright[C]. Proceedings of the 2009 ACM SIGMOD International Conference on Management of data. ACM, 2009: 841 - 846.

[9]　Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107 - 113.

[10]　Shvachko K, Kuang H, Radia S, et al. The hadoop distributed file system[C]. Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on. IEEE, 2010: 1 - 10.

[11] Stonebraker M, Abadi D, DeWitt D J, et al. MapReduce and parallel DBMSs: friends or foes [J]. Communications of the ACM, 2010, 53 (1): 64-71.

[12] Presto definition[EB/OL]. http://prestodb.io.

[13] Thusoo A, Sarma J S, Jain N, et al. Hive: a warehousing solution over a map-reduce framework[J]. Proceedings of the VLDB Endowment, 2009, 2(2): 1626-1629.

The Expansion Research of Consistent Hash Algorithm on Parallel Analysis Database Cluster

ZHAO Fei, SU Zhong

(Air Force Command College, Network Center, Beijing 100097, China)

Abstract: In the research of parallel analytical database cluster, scalability is an important performance indicator. In order to divide the data fast, reliably and safely to the new cluster node in the case of data's rapid growth or some cluster servers' breakdown, we must set data partition strategy reasonably. In this paper, we study the consistency hash algorithm which is in common use in Key-Value storage technology and then put forward a method that uses the algorithm in the large-scale structured data partition. After that, we design a program in the MapReduce framework and then use TPC-DS as the benchmark to validate the method. The experimental results show that the scheme not only has good performance of the data partitioning, but also has better scalability.

Key words: computer application technology; database; parallel analysis database cluster; consistent hash algorithm; data partitioning