2) 主节点在逻辑划分阶段对表文件进行逻辑划分,划分为多个 split;
3) 针对每一个 split,调用相应 API 查询内存中的统计信息表,获取每个 split 的数据区间;
4) 根据获取的 split 数据区间以及查询条件,判断该 split 是否满足查询条件:若不满足,则直接丢弃;
5) 若满足,则由后续操作符继续进行处理,直至查询结束.

4 split 内过滤

split 内过滤主要是从减少数据读取产生的磁盘 I/O 的角度进行优化. split 内过滤主要通过对 split 建立索引,在查询执行阶段,通过查询索引只读取满足查询条件的数据,减少不必要的数据扫描产生的磁盘 I/O.split 内过滤主要包含两个部分:聚集索引和非聚集索引.

4.1 聚集索引

聚集索引主要是借鉴了 Hadoop++的技术.简述如下.

存于 HDFS 上的表文件在进行处理时会被划分为一个个 split,将每个 split 中的记录按索引属性进行排序,然后针对索引属性建立聚集索引,将聚集索引存放于数据之后存于 HDFS 中.其结构如图 5 所示.

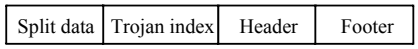


Fig.5 Structure of clustered index
图 5 聚集索引结构

Split Data 为该 split 的数据,Trojan Index 为针对此 split 的索引属性建立的聚集索引,其存放于数据之后.Header 存放索引的一些元信息,比如索引大小等.Footer 存放 split 的大小,主要用于重新划分新的 split,将原来的数据和生成的索引划分为一个新的 split.

如图 6 所示,Header 包含以下 5 个字段:DataSize 为该 split 数据的大小,IndexSize 为索引本身大小,Max 为索引属性在该 split 的最大值;Min 为索引属性在该 split 的最小值,RecordNum 为该 split 的 record 数量.Footer 中,SplitSize 和 FooterSize 分别表示 split 的大小和 Footer 的大小.

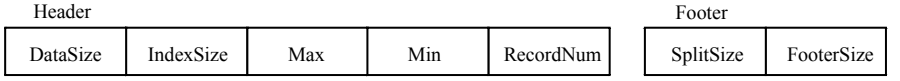


Fig.6 Structures of Header and Footer
图 6 Header 和 Footer 内部结构

- 建立索引

聚集索引可以通过 SOH 本身的计算框架建立,比如 MapReduce,Spark 等.这里介绍聚集索引的建立过程,此处以 MapReduce 为例.

1) 针对存放于 HDFS 的表文件,Map 过程会将其划分多个 split.split 中的每个键值对通过 *map* 函数进行处理,处理前⟨*key*,*value*⟩为{*pos*,*record*},经过 *map* 函数处理后为{*SplitID*+*a*,*record*},此处 *pos* 为该行在 split 中的偏移量,*a* 为索引属性的值.以图 7 中第 1 个数据记录为例,其 *map* 函数处理前的⟨*key*,*value*⟩为{0,{3223,1212,2300}},经过 *map* 处理之后,转化为{1+3223,{3223,1212,2300}};
2) 通过自定义函数 *Partitioner*,按 SplitID 进行划分,确保相同 Split 的数据交给同一个 Reducer 进行处理;
3) Reducer 处理阶段按照 SplitID 进行分组,并对于每个 split 按照 *a* 进行排序.如图 7 中按照 *Order_ID* 将该 split 的记录进行排序;
4) 针对 *a* 属性生成聚集索引.如图 7 所示,索引为 Trojan Index 记录了 *order_id* 的值和对应的偏移量.Header 记录数据和索引的元信息,Footer 记录新 split 的大小.

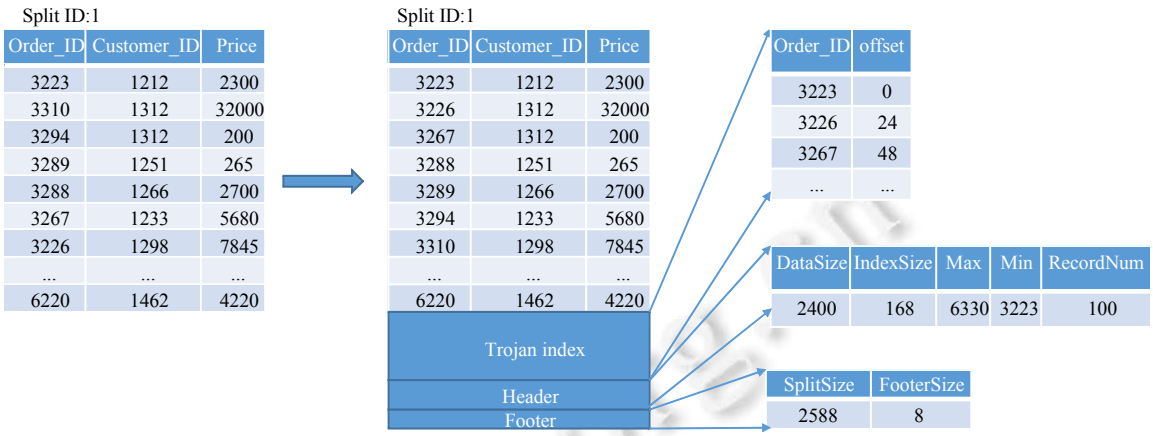


Fig.7 An example of constructing an index

图 7 建立索引过程示例

索引建立时间取决于所依赖的计算框架,但是因为本文主要针对于交互式查询,一般会将索引提前建好,因此并不影响查询的性能.由于 HDFS 数据“一次写入,多次读取”的特点,对于索引维护的需求并不是很大,因此,Hadoop++没有实现索引维护.

查询过程以 MapReduce 计算框架为例,其他计算框架同理,不再一一说明.

1) MapReduce 首先读取文件末尾处最后一个 Footer 字段,通过读取 splitsize 字段获取该 split 的大小,然后将该 split 划分出来;然后,读取倒数第 2 个 Footer,划分倒数第 2 个 split;依此类推;
2) 每一个新的 split 会交给一个从节点处理,读取 header 获取索引的元信息,比如索引大小等;
3) 读取索引,通过索引确定满足条件的记录的偏移量,查询并读取满足查询条件的记录.

4.2 非聚集索引

Hadoop++只实现了聚集索引,而没有考虑非聚集索引.在实际应用中,我们很多时候需要实现非聚集索引,比如查询的过滤条件不只是涉及到数据表的一个属性,而是该表的多个属性均有过滤条件.因此,本文设计并实现了非聚集索引.一个表可以同时拥有一个聚集索引和多个非聚集索引,以支持不同的查询要求.

非聚集索引的结构如图 8 所示.

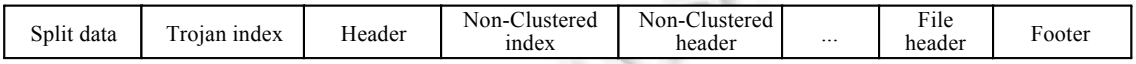


Fig.8 Structure of non-clustered index

图 8 非聚集索引结构

Split Data 为数据,Trojan Index 为聚集索引,Header 保存聚集索引的元信息,NonClustered Index 是非聚集索引,主要是保存未排序的属性的偏移量.NonClustered Header 保存非聚集索引的元信息.NonClustered Index 和 NonClustered Header 可以有多个,即可以同时建立多个非聚集索引.File Header 主要记录该 split 在哪些属性上建立了索引.Footer 记录该 split 的大小,用于划分 split.

- 建立索引

非聚集索引建立过程与聚集索引类似,但在数据排序等过程有所不同,具体过程如下.

1) split 中的每个键值对通过 *map* 函数进行处理,$\{pos,record\}\rightarrow\{SplitID+a+pos,record\}$;
2) 按照 SplitID 进行划分,确保相同 Split 的数据交给同一个 Reducer 进行处理;
3) Reducer 处理阶段,按照 SplitID 进行分组,并对于每个 split 按照 *pos* 进行排序.确保数据建立索引之前

和建立索引之后的相对顺序不变;

4) 针对属性 a 建立非聚集索引.

- 查询过程

非聚集索引查询过程与聚集索引有较大的不同:因为聚集索引的 split 是按照索引属性排序的,因此对于范围查询只需判断查询条件与 split 索引属性值范围的重叠范围,然后从头到尾扫描即可;而数据对非聚集索引属性是无序的,因此不能采用聚集索引的此种方法.

我们根据非聚集索引的特点和 SOH 数据扫描的特点制定了非聚集索引的扫描策略,我们假设 split 值的范围是[c,d],查询条件的查询范围是[a,b],这里有 6 种情况.

(1) 当 $c\leqslant a\leqslant d$ and $b\geqslant d$ 时,加载索引,扫描起始位置为[a,d]区间中所有值的偏移量最小值,终止位置为[a,d]中区间所有值的偏移量最大值;

(2) 当 $c\leqslant a\leqslant d$ and $c\leqslant b\leqslant d$ 时,加载索引,扫描起始位置为[a,b]区间中所有值的偏移量最小值,终止位置为[a,b]中区间所有值的偏移量最大值;

(3) 当 $a=b$ 时,加载索引,扫描起始位置为值 a 的偏移量最小值,终止位置为值 a 的偏移量最大值;

(4) 当 $a\leqslant c$ and $c\leqslant b\leqslant d$ 时,加载索引,扫描起始位置为[c,b]区间中所有值的偏移量最小值,终止位置为[c,b]中区间所有值的偏移量最大值;

(5) 当 $a\leqslant c$ and $b\geqslant d$ 时,全表扫描;

(6) 当 a,b 不满足以上 5 种情况时,丢弃该 split.

非聚集索引的查询过程基于以上的扫描策略,具体查询过程如下.

1) 首先读取文件末尾 Footer 字段,划分 split.类似聚集索引的方法将所有的 split 划分出来;

2) 读取 File Header 字段,判断查询条件中的属性是否建立了索引,并确定非聚集索引属性的偏移量;

3) 读取 NonClustered Header,获取非聚集索引的元信息.通过 Header 中记录的该 split 值的范围和查询条件中的查询范围确定扫描策略;

4) 根据制定的扫描策略扫描数据.

非聚集索引有效性分析:首先,根据索引,可以避免扫描一些完全无关的 split;其次,当 split 范围和查询范围重叠时,我们可以缩小扫描区域,避免读取整个 split 的记录;最后,当某个属性存在重复值时,通过索引可以定位到符合点查询条件的所有记录的偏移量范围,避免全表扫描.非聚集索引解决了查询包含多个属性过滤条件的问题.后续通过实验验证了非聚集索引的效率,证明了其可用性与有效性.

5 实验分析

5.1 实验设定和数据集

实验是在 Openstack 平台上进行的,我们使用了 8 个节点用于实验,其中,1 个为主节点,7 个为从节点.每个节点拥有 2 个 CPU 和 2GB 的内存,操作系统为 CentOS 6.5.每个节点拥有 200GB 的硬盘容量.系统源代码采用的是 Java.在数据集选择上,我们采用了通用的标准数据集 TPC-H,该数据集是模拟订单和仓库管理随机产生的数据,本实验选用的数据量为 10G.Hadoop 版本为 2.5.2.

5.2 数据区间粒度对于查询性能的影响

本节主要介绍数据区间粒度(即第 3.1 节中讨论的 k 值大小)对于查询性能影响的实验.数据区间粒度直接影响统计信息表的大小和查询的效率,如何确定数据区间粒度,是一个非常重要的问题.本实验通过调整数据区间粒度来观察查询的时间,从而确定出比较优的数据区间粒度,为后续实验奠定基础.

由图 9 可知:对于 TPC-H 数据集,查询时间随着数据区间划分个数增长而逐渐减少,到数据区间为 160 时基本趋于平缓.因此对于 TPC-H 数据集来讲,数据区间个数取值为 160 时是比较合理的.对于不同的数据集,最优的数据区间粒度有所不同,我们可以针对不同的数据集对数据区间粒度进行调整得到最优的数据区间粒度.

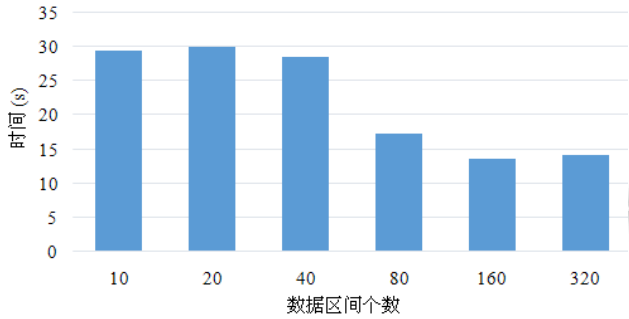


Fig.9 Influence of data ranges on query performance

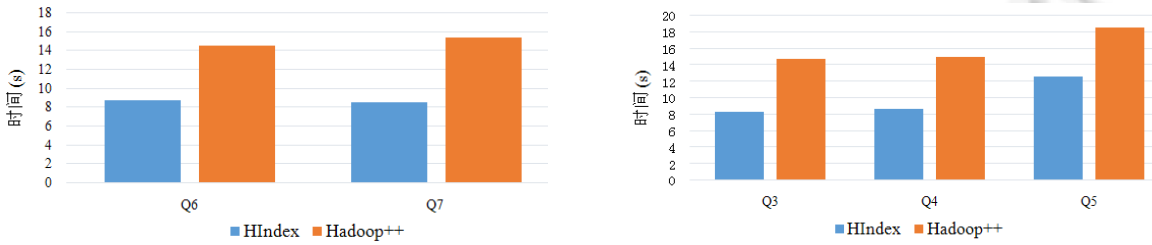
图 9 数据区间粒度对查询性能的影响

5.3 聚集索引性能测试

本节主要介绍聚集索引性能测试实验.在本实验中,我们的对比系统选择的是德国 Saarland 大学 Dittrich 等人提出的 Hadoop++系统.本实验从等值查询和短范围查询(查询区间小于相应的属性值域范围 5%)两个方面来测试.因为长范围查询涉及到大量的数据计算,基本已经接近于全表扫描,属于分析型查询的范畴,已经超出了本文的应用范围,此处不再做分析.

(1) 等值查询

图 10(a)是等值查询性能对比实验的结果.如图所示,HIndex 在 Q6,Q7 两个等值查询中性能比 Hadoop++有了明显的提升,查询性能分别提升了 40%和 44.2%.



(a) 等值查询性能对比实验 (b) 短范围查询性能对比实验

Fig.10 Performance comparison of clustered index

图 10 聚集索引性能对比

我们在表 1 中对比了 HIndex 和 Hadoop++两个系统 Map Task 启动的个数.Hadoop++两个查询启动 Map Task 数量均为 122,而 HIndex 系统整个查询只需要一个 Map Task 来处理,极大地优化了查询性能.表 1 的结果展示了 HIndex 在 split 层过滤的有效性.

Table 1 Number of Map Task for evaluating equality queries

表 1 等值查询启动 Map Task 个数对比表

Map Task 个数	Q6	Q7
HIndex	1	1
Hadoop++	122	122

(2) 短范围查询

图 10(b)展现的是短范围查询的实验结果.HIndex 相比于 Hadoop++在 3 个短范围查询 Q3,Q4,Q5 中查询速度分别提升了 43.5%,42.4%和 32.6%.

从表 2 中可以看出,HIndex 在 Q3,Q4,Q5 这 3 个查询中分别只需要启动 1,1,3 个 Map Task 去处理,优化效果

非常明显.

Table 2 Number of Map Task for evaluating short-range queries

表 2 短范围查询启动 Map Task 个数对比表

Map Task 个数	Q3	Q4	Q5
HIndex	1	1	3
Hadoop++	122	122	122

由于等值查询和短范围查询的结果较少,其满足查询条件的记录只存在于少量的 split 中,因此,大量的 split 是不需要启动 Map Task 去处理的.HIndex 可以完全避免启动 Map Task 所花费的开销,减少查询处理的代价,使得查询性能得到了提升.

5.4 非聚集索引性能测试实验

因为 Hadoop++系统并没有实现非聚集索引,因此,此实验直接与 Hadoop 进行对比.

(1) 等值查询

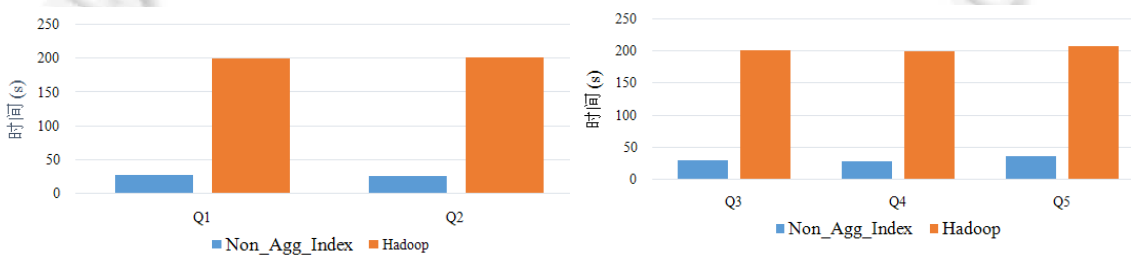
图 11(a)表明:非聚集索引在等值查询方面相较于 Hadoop 有很大优势,其在 Q1,Q2 性能分别提升了 7.22 倍和 7.5 倍.

(2) 短范围查询

在短范围查询方面,非聚集索引依然体现了明显的优势.如图 11(b)所示,HIndex 在 Q3,Q4,Q5 这 3 个短查询上查询速度相较于 Hadoop 分别提高了 7.12 倍、7.22 倍和 5.75 倍.

(3) 查询性能分析

对于等值查询和短范围查询,聚集索引显示出了巨大的优势.这是因为这两种的查询结果一般都比较少,如果此时还采用 Hadoop 全表扫描的方式,则会产生大量的磁盘 I/O,极大地增加了查询时间.通过非聚集索引,则可以提前过滤掉不需要的 split 和 split 内无用的数据,减少了磁盘 I/O,达到查询优化的目的.



(a) 等值查询 (b) 短范围查询

Fig.11 Performance comparison of non-clustered index

图 11 非聚集索引对比实验

6 结束语

本文针对基于 HDFS 的 SOH 系统在处理选择型查询或交互式查询时遇到的性能瓶颈提出了一种多层索引技术 HIndex.该索引包括 split 层和 split 内部两层过滤,实现了聚集索引和非聚集索引.实验结果显示,HIndex 可以有效地提高 SOH 系统查询处理效率.对于未来工作的展望:我们已开始在 Spark SQL 上集成本文所提出的索引技术.此外,我们将研究如何根据数据集规模和统计特征,自动确定数据区间粒度的选择(k).

References:

[1] Huai Y, Chauhan A, Gates A, Hagleitner G, Hanson EN, O'Malley O, Pandey J, Yuan Y, Lee RB, Zhang XD. Major technical advancements in apache hive. In: Proc. of the SIGMOD Conf. 2014. [doi: 10.1145/2588555.2595630]

[2] He YQ, Lee RB, Huai Y, Shao Z, Jain N, Zhang XD, Xu ZW. RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems. In: Proc. of the ICDE. 2011. [doi: 10.1109/ICDE.2011.5767933]

[3] Melnik S, Gubarev A, Long JJ, Romer G, Shivakumar S, Tolton M, Vassilakis T. Dremel: Interactive analysis of Web-scale datasets. Proceedings of the VLDB Endowment, 2010,3(1):330−339.

[4] Dittrich J, Quiane-Ruiz JA, Jindal A, Kargin Y, Setty V, Schad J. Hadoop++: Making a yellow elephant run like a cheetah (without it even noticing). Proceedings of the VLDB Endowment, 2010,3(1):518−529.

[5] Richter S, Quiané-Ruiz JA, Schuh S, Dittrich J. Towards zero-overhead static and adaptive indexing in Hadoop. In: Proc. of the VLDB. 2014. [doi: 10.1007/s00778-013-0332-z]

[6] Eltabakh MY, Özcan F, Sismanis Y, Haas PJ, Pirahesh H, Vondrak J. Eagle-Eyed elephant: Split-Oriented indexing in Hadoop. In: Proc. of the EDBT. 2013. 89−100. [doi: 10.1145/2452376.2452388]

[7] Gankidi VR, Teletia N, Patel JM, Halverson A, De Witt DJ. Indexing HDFS data in PDW: Splitting the data from the index. Proceedings of the VLDB Endowment, 2014,7(13):1520−1528. [doi: 10.14778/2733004.2733023]

[8] Lu P, Chen G, Ooi BC, Vo HT, Wu S. ScalaGiST: Scalable generalized search trees for MapReduce systems. Proceedings of the VLDB Endowment, 2014,7(14):1797−1808. [doi: 10.14778/2733085.2733087]

[9] Xin RS, Rosen J, Zaharia M, Franklin MJ, Shenker S, Stoica I. Shark: SQL and rich analytics at scale. In: Proc. of the SIGMOD Conf. 2013. 13−24. [doi: 10.1145/2463676.2465288]

[10] Chen GJ, Wiener JL, Iyer S, Jaiswal A, Simha RLN, Wang W, Wilfong K, Williamson T, Yilmaz S. Realtime data processing at facebook. In: Proc. of the SIGMOD Conf. 2016. [doi: 10.1145/2882903.2904441]

[11] Hayward R. Average case analysis of heap building by repeated insertion. Journal of Algorithms, 1991,12:126−153. [doi: 10.1016/0196-6774(91)90027-V]

何龙(1988－),男,河北石家庄人,硕士,主要研究领域为大数据管理技术.

陈晋川(1978－),男,博士,副教授,主要研究领域为大数据管理技术,不确定性数据管理技术.

杜小勇(1963－),男,博士,教授,博士生导师,CCF 会士,主要研究领域为数据库系统,大数据管理,智能信息检索,知识工程.