基于 MapReduce 的 Skyline-join 查询算法

孙大烈，李建中

（哈尔滨工业大学 计算机科学与技术学院，150001 哈尔滨，sdl@hit.edu.cn）

摘　要：Skyline 查询是一种非常耗时的操作，而涉及多个表的 Skyline 查询（Skyline-join 查询）则会给数据库系统带来更多的负载，从而影响整个系统的响应时间. 为解决这个问题，提出了基于 Google 设计的 MapReduce 并行处理框架的 Skyline-join 查询处理算法，采用分片剪枝的方法降低复杂度，进而提高查询性能. 在 Amazon 的云计算平台（EC2）上进行的实验表明，该算法可以有效减少冗余操作和网络数据传输，基本不受节点个数以及数据量的影响，具有很好的可扩展性.

关键词：Skyline 查询；MapReduce；分布式算法；云计算

中图分类号：TP311.133.1　　**文献标志码**：A　　**文章编号**：0367－6234(2012)01－0103－04

MapReduce-based Skyline-join processing

SUN Da-lie, LI Jian-zhong

(School of Computer Science and Technology, Harbin Institute of Technology, 150001 Harbin, China, sdl@hit.edu.cn)

Abstract: Skyline query is one of the most expensive operators in the database system. Some Skyline queries involving multiple tables, which are called Skyline-join queries, are even more costly to evaluate. Therefore, in this paper, we adopt Google's MapReduce, a parallel processing framework, to handle Skyline-join queries. A novel parallel algorithm is proposed to prune the dataset progressively and hence the network transfer cost is reduced. The algorithm is evaluated on Amazon's EC2 and the experiments verify its efficiency.

Key words: Skyline query; MapReduce; distributed algorithm; cloud computing

给定一个感兴趣的属性集合，Skyline 查询返回这样的元组，这些元组在任何一个属性上都不受其他元组制约[1]. 例如，一名学生想要寻找一个合适的住处，他可能提交这样的查询："返回价格便宜并且离学校距离近的公寓". Skyline 查询在决策系统中很有价值（用处）. 由于它的重要性，研究人员已经开始在商用数据库管理系统（DBMS）中实现 Skyline 查询[2-3].

现有的研究工作大多假设 Skyline 查询只局限于一个数据表，也就是说，所有待查的属性都来自于同一张数据表. 然而，这种假设在互联网环境中不再成立，因为此时查询处理需要来自多源的数据. 例如，数据库 cheapoair.com 提供机票预订服务，BookInHotels.com 提供酒店预订服务. 假设用户提交这样的查询" 请列出所有在 5 月 11 日起飞的最廉价的航班，以及距离机场最近的四星级酒店". 这种查询与 Skyline 查询有共同的特点，但是它需要从多张数据表提取数据信息. 在本文中，称这种类型的查询为 Skyline-join.

处理 Skyline-join 查询的一个简单而原始的方法是，先连接所有相关的数据表，然后再应用现有的 Skyline 算法. 然而，这种简单的算法往往效率低下，不能提供及时的结果. 因此，本文提出一种新的基于 MapReduce 框架[4-5] 的分布式并行算法. 该算法可以应用在计算机集群环境中，通过将计算分布在不同的节点上来提高处理速度.

1　Skyline-join 查询

如前所述，不同于现有的操作，Skyline-join 操作会涉及多张数据表. 对于某单一数据表，如果当 $0 \leqslant i \leqslant d$ 时，$v_i > v'_i$ 并且 $\exists j(0 \leqslant j \leqslant d) \wedge (v_j > v'_j)$，这里定义 $\geqslant$ 和 $>$ 为偏序关系，则元组

收稿日期：2011－02－24.
基金项目：国家自然科学基金资助项目(61033015).
作者简介：孙大烈(1963—)，男，副教授；
李建中(1950—)，男，教授，博士生导师.

$t_i = (v_0, v_1, \cdots, v_d)$ 制约元组 $t_j = (v'_0, v'_1, \cdots, v'_d)$. Skyline 查询返回那些不受其他元组制约的元组. 若 $Q = <\{A_1, A_2, B_1, B_2\}, \varnothing, \{A_3, B_3\}>$ 取自多张数据表,则引发了新问题——多数据表 Skyline 查询. 为处理这种查询,本文引入一种新的查询操作,称为 Skyline-join. 最近,Skyline-join 查询引起了其他研究者的兴趣[6].

设有数据库 D,其中数据表集合为 T, Skyline-join 表示为 $<S, C, J>$,其中 S 为 Skyline 查询中的属性集合, C 为用户提交的查询条件集合,J 为连接属性集合. $S(T)$, $C(T)$ 和 $J(T)$ 分别为对于某数据表 T 的 Skyline 查询属性集、条件属性集以及连接属性集. 为讨论简便起见,这里假设 $J \cap S = \varnothing$. 事实上,$J \cap S \neq \varnothing$ 只是算法的一种特殊情况. 对于某查询 $<S, C, J>$, Skyline-join 检索空间定义为:

定义 1 Skyline-join 检索空间.

T_{total} 是查询 $<S, C, J>$ 的 Skyline-join 检索空间,当且仅当

a) $T_{\text{total}} = T_0 \infty T_1 \infty \cdots \infty T_k, T_i \in J$;

b) $\forall \text{att}_i \in (S \cup C) \rightarrow \exists T_i (T_i \in T_{\text{total}}) \wedge (\text{att}_i \in T_i)$;

c) 不存在 T'_{total} 满足上述两条性质,且其中数据表的个数少于 T_{total}.

Skyline-join 查询在其相应检索空间里进行处理,检索空间中不同元组之间的制约关系定义为:

定义 2 Skyline-join Domination.

给定 Skyline-join 查询 $<S, C, J>$ 及其检索空间 T_{total},对于 T_{total} 中的两元组 $t_1 = (v_0, v_1, v_2, \cdots, v_d)$ 和 $t_2 = (v_0, v_1, v_2, \cdots, v_d)$,$t_1$ 制约 t_2 当且仅当 $\forall \text{att}_i (\text{att}_i \in S) \rightarrow (v_i \succeq v'_i) \wedge (\exists \text{att}_j (\text{att}_j \in S) \wedge (v_j \succ v'_j))$.

为表述方便,本文用 $t_i \succ_{\text{dom}} t_j$ 表示元组 t_iSkyline-join 制约元组 t_j,并用 $t_i \succ_A t_j$ 来表示元组 t_i 在属性集合 A 上 Skyline-join 制约元组 t_j.

定义 3 Skyline-join 结果集.

对于 Skyline-join 查询 $Q = <S, C, J>$,检索空间为 T_{total},其结果集 R 有以下性质:

性质 1 $\forall t_i (t_i \in R) \rightarrow \neg \exists t_j (t_j \in T_{\text{total}}) \wedge (t_j \succ_{\text{dom}} t_j)$.

性质 2 对于元组 $t_i \in R, t_i = (v_0, v_1, \cdots, v_d)$,若存在 $\text{att}_i \in C$,则 v_i 必须满足相应查询属性条件.

为方便讨论简单,假设查询条件 $C = \varnothing$ 并只考虑数值型属性,域值范围 [0, 100],并用“MAX”作为 Skyline 条件.

2 MapReduce 框架

MapReduce 是由 Google 提出的处理海量数据的并行处理平台. 它可以应用在计算机集群之上,通过将数据和计算分布在不同的节点来取得高性能. MapReduce 也是一个灵活的编程框架,它提供了两个接口函数:Map 和 Reduce. 用户可以实现自己的 Map 和 Reduce 函数来完成相关的处理任务.

Map 和 Reduce 函数的输入必须是一对(key, value). 其中 key 用来表示数据的键值,而 value 则代表实际的数据. Map 函数可以抽象为

$$\text{Map}(k_1, v_1) \text{ -> } \text{list}(k_2, v_2).$$

对每一个输入的(key, value)对,Map 函数进行相应的处理,并输出一串新的(key, value)对. 这些新的(key, value)对被传送到不同的 Reduce 函数进行进一步的处理. 而 Reduce 函数可以抽象为

$$\text{Reduce}(k_2, \text{list}(v_2)) \text{ -> } \text{list}(v_3).$$

Reduce 函数收到来自不同 Map 函数的(key, value)对. 在进行处理之前,它先对这些(key, value)对按照它们的 key 值进行排序,具有相同 key 值的对被集成为(key, list(v)),即一个 key 值对应一个 value 数组. (key, list(v)) 作为输入被传入 Reduce 函数,然后按照用户的处理逻辑生成结果. 通常结果是一串值 (表示为 list(v)),并被写为分布式文件系统,如 GFS[7] 和 BigTable[8].

图 1 列出了基于 MapReduce 的统计词频的伪代码. 在 Map 函数中,每次对一行的文本进行分析,将单词切分出来. 对每一个单词 w 产生一个(key,value)对,即 (w, 1),表示该单词已经出现了一次. 在 Reduce 函数,对同一个函数的统计被集成在一个(key, list(value))中. 因此,只要遍历该数组,就可以知道这个单词出现的次数.

```
public void map(LongWritable key, Text value, Context context){
  String line = value.toString();
  StringTokenizer tokenizer = new StringTokenizer(line);
  while (tokenizer.hasMoreTokens()) {
  word.set(tokenizer.nextToken());
  context.write(word, one);
  }
}
public void reduce(Text key, Iterable < IntWritable > values, Context context){
  int sum = 0;
  for (IntWritable val: values) {
    sum + = val.get();
  }
  sum + = val.get();
}
```

图 1 MapReduce 词频统计算法

3　基于 MapReduce 的查询方法

本文用表 1，表 2 作为例子来阐述相关的算法.

表 1　表 *R*

ID	A_1	…	A_m

表 2　表 *S*

ID	B_1	…	B_n

其中表 R 和表 S 使用 ID 属性进行连接，而对于其他属性（$A_1 \cdots A_m$ 以及 $B_1 \cdots B_n$），Skyline-join 查询使用 Max 作为条件，要求返回在这些属性上不被其他记录支配的记录.

使用 MapReduce 来处理 Skyline-join 查询可分为：1）一个 MapReduce 任务被用来产生表的连接结果；2）连接的结果被用来作为另一个 MapReduce 任务的输入，而该任务产生多个表的 Skyline-join 结果.

3.1　产生连接结果

在第 1 个 MapReduce 任务中，Map 函数从分布式文件系统读取两个表的数据，对于每一个记录，Map 函数产生一个（key，value）对，其中该记录的 ID 值被用作为 key，而其他值被作为 value. 这样一来，能够在 ID 上连接的表 R 和表 S 的记录将被发送到同一个 Reduce 函数. 该 Reduce 函数将含有相同 ID 的记录集合在一起，采用基于内存的连接算法产生相应的表连接结果. 在表连接结果产生之后，Reduce 函数再调用基于内存的 Skyline 算法，比如 BNL 算法. 被其他记录支配的记录被舍去，因为它们不可能成为最终的 Skyline-join 结果. 其他记录被写入分布式文件系统. 图 2 给出了相关伪码.

```
public void map(LongWritable key, Text value, Context context){
   Tuple tuple = new Tuple(value);
   context.write(tuple.getID(), tuple.toString());
}
public void reduce(Text key, Iterable<Text> values, Context context){
   ArrayList<Tuple> RData, SData, Result;
   for(Text val: values){
      Tuple tuple = new Tuple(val);
      if(tuple is from R){
               for(Tuple t: SData){
          Tuple joinResult = join(t, tuple);
          Result.add(joinResult);
        }
        RData.add(tuple);
      }
      else{
        //similar processing logic as R's tuple
      }
   }
   ArrayList<Tuple> finalresult = getSkyline(Result);
   Write finalresult to Distributed File System
}
```

图 2　MapReduce 的 Skyline-join 算法

图 2 是针对两个表的 Skyline-join. 对于更多表的 Skyline-join 查询，该算法依旧适用. 唯一的区别就是当处理有 n 个表的 Skyline-join 查询时，需要有 n 种 Map 函数，每种函数读取一个表的数据. 同时在 Reduce 函数，需要对 n 个表进行连接操作.

3.2　产生 Skyline-Join 结果

第 2 个 MapReduce 任务读取上一个任务产生的结果，然后采用空间划分的方式来计算 Skyline 结果. 假设 Skyline-join 查询要求在 x 和 y 属性上得到不能被支配的元组，图 3 给出了一种可能的空间划分方法. 该方法将 x 和 y 平均划为 4 个区域，从而整个空间被划为 $4 \times 4 = 16$ 的子空间. 如果一个记录位于子空间 $S_{(a,b)}$，那么它的 x 属性的值位于 x 的第 a 个区间，而它的 y 属性位于 y 的第 b 个区间.

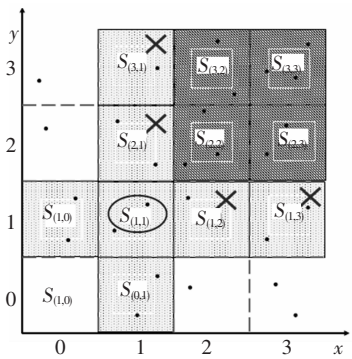

图 3　空间划分

在第 2 个 MapReduce 任务中，Map 读取上一个任务的结果，然后为每一个记录判断其所在的子空间. 假设下一个记录 t 在空间 $S_{(a,b)}$，那么 Map 为它生成的（key，value）对为 $(h(a, b), t)$. 其中 h 为一个哈希函数，给定两个值 a 和 b，$h(a, b)$ 返回一个 Reduce 函数的 ID. 根据该 ID，Map 函数知道该（key，value）对应该传递给哪一个 Reduce 函数. 采用这种方法，同一个子空间的记录都会发送给同一个 Reduce 函数. 该 Reduce 函数调用基于内存的 Skyline 算法，可以计算得出该子空间的 Skyline 结果.

当第 2 个 MapReduce 任务完成后，分布式文件系统存储着每个子空间的 Skyline-join 结果. 然而这并不是最终的结果. 为了得到全局的 Skyline-join 结果，第 3 个 MapReduce 任务被提交，用来合并子空间的 Skyline-join 结果. 在合并之前，本文采用一个预处理过程来删减掉不可能产生结果的子区间.

在图 3 中，假设子空间 $S_{(3,2)}$，$S_{(2,2)}$，$S_{(3,3)}$ 和 $S_{(2,3)}$ 产生了一些 Skyline-join 结果. 那么子空间 $S_{(1,1)}$ 可以被排除，因为即使它产生了一些结果，其结果也必然被上述 4 个子空间中的结果支配，从而不可能出现在最终结果中. 相反，子空间 $S_{(3,1)}$，$S_{(2,1)}$，$S_{(1,2)}$ 和 $S_{(1,3)}$ 不能被排除，因为它们的结果并不能被前面的子空间支配.

预处理之后，剩下的结果被 Map 函数读取，

Map 函数为所有的记录产生同一个 key,使得它们都被发送到同一个 Reduce 函数. 该 Reduce 函数调用传统的 Skyline 算法,如文献[9 – 10],来产生最终的结果.

4 实验结果

为了验证分布式算法的有效性,本文采用 Amazon 的 EC2[11] 平台搭建了一个集群环境,并根据 文献[6]的描述生成了实验数据. 实验数据包括两个数据表 R 和 S. 表 R 包含 3 个属性:员工编号、年龄和工资. 表 S 包含 3 个属性:员工编号、经理编号和公司编号. 所有属性均为整形,并符合均匀数据分布,其中表 R 和表 S 使用员工编号进行连接操作. 在一个 n 节点构成的集群中,每个表的元组数为 1 500 000 n. 每个节点运行 4 个 Map 进程和2 个Reduce 进程.

图 4 展示了随着节点个数的增加,查询处理时间的变化. 可以看到,本文的分布式算法基本不受节点个数以及数据量的影响(节点越多数据量越大). 因此,该算法具有很好的可扩展性. 如果要获得更好的查询性能或者能够处理更多的数据,只需要增加更多的节点即可.

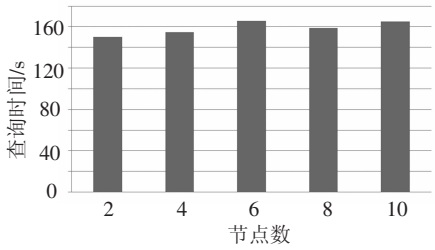

图 4 节点数目的影响

作为比较,图 5 显示了当节点增加数据量不变的情况下,查询的效率变化. 可以看出,本文提出的 Skyline-join 算法非常适合应用在并发处理平台如 MapReduce 之上. 当节点数量增加,查询的速度也随之变快,几乎达到线性递减的趋势.

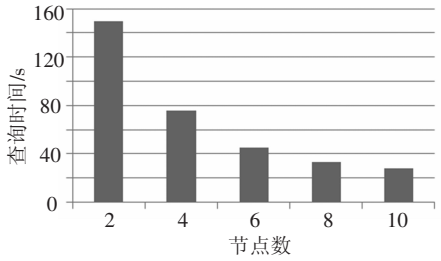

图 5 节点数目的影响

5 结 论

1)通过将数据以及计算分布在不同的节点上,使用并行化的处理机制来提高性能.

2)在 Map 阶段采用分片剪枝的方法来降低复杂度.

3)通过在真实的云计算平台实验表明,该算法具有高可扩展性的特点.

参考文献:

[1] BORZSONYI S, STOCKER K, KOSSMANN D. The Skyline operator[C]//Proceedings of the 17th International Conference on Data Engineering. Washington, DC: IEEE Computer Society, 2001: 421 – 430.

[2] CHAUDHURI S, DALVI N, KAUSHIK R. Robust cardinality and cost estimation for skyline operator[C]//Proceedings of the 22nd International Conference on Data Engineering. Washington, DC: IEEE Computer Society, 2006: 64.

[3] EDER H, WEI Fang. Evaluation of skyline algorithms in PostgreSQL[C]//Proceedings of the 13th International Database Engineering & Applications Symposium. New York, NY: ACM, 2009: 334 – 337.

[4] DEAN J, GHEMAWAT S. MapReduce: Simplified data processing on large clusters[J]. Communication of ACM, 2008, 51(1): 107 – 113.

[5] YANG Hung-Chih, DASDAN A, HSIAO Ruey-Lung, *et al*. Map-reduce-merge: simplified relational data processing on large clusters[C]//Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data. New York, NY: ACM, 2007: 1029 – 1040.

[6] JIN W, ESTER M, HU Z J, *et al*. The multi-relational skyline operator[C]//Proceedings of the IEEE 23rd International Conference on Data Engineering. Washington, DC: IEEE, 2007,1376 – 1380.

[7] GHEMAWAT S, GOBIOFF H, LEUNG Shun-Tak. The Google file system[J]. ACM SIGOPS Operating Systems Review, 2003, 37(5): 29 – 43.

[8] CHANG F, DEAN J, GHEMAWAT S, *et al*. Bigtable: A distributed storage system for structured data[C]//Proceedings of the 7th Usenix Symposium on Operating Systems Design and Implementation. Berkeley CA: USENIX Association, 2006: 205 – 218.

[9] BARTOLINI I, CIACCIA P, PATELLA M. Salsa: Computing the skyline without scanning the whole sky[C]//Proceedings of the 15th ACM International Conference on Information and Knowledge Management. Arlington, Virginia, 2006: 405 – 414.

[10] CHAN C Y, ENG P K, TAN K L. Stratified computation of skylines with partially-ordered domains[C]//Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data. New York, NY: ACM, 2005: 203 – 214.

[11] Amazon. Amazon Elastic Compute Cloud (Amazon EC2)[EB/OL]. http://aws.amazon.com/ec2//192 – 5518875 – 2032964.

(**编辑** 张 红)