一种基于大数据的有效搜索方法

尤川川[1,3] 张桂刚[2,3]
(武汉大学软件工程国家重点实验室 武汉 430072)[1] (清华大学信息技术研究院 北京 100084)[2]
(湖北经济学院信息管理学院 武汉 430205)[3]

摘 要 针对大数据查询效率低下的问题,提出了一种有效的搜索方法。将共享的历史查询结果作为中间结果集,在新的查询请求到达时,首先与历史查询进行匹配,若能实现匹配,则直接将匹配部分的历史查询结果直接作为新查询请求结果的一部分。这减少了大量的对历史查询的重复计算,节省了搜索时间,提高了查询效率。实验对比分析表明,新的基于大数据的查询方法能较好地提高查询效率。

关键词 大数据,搜索,查询网,云数据库
中图法分类号 TP301.6 **文献标识码** A

A Kind of Efficient Search Method Based on Big Data

YOU Chuan-chuan[1,3] ZHANG Gui-gang[2,3]
(The State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China)[1]
(Research Institute of Information Technology, Tsinghua University, Beijing 100084, China)[2]
(School of Information Management, Hubei University of Economics, Wuhan 430205, China)[3]

Abstract This paper proposed an efficient search method to the problem of low efficiency for large dada queries. Using shared history query results as a set of intermediate results, when a new query request arrives, the first match for historical inquiry is directly added to the matching portion of the historical results for directly as part of the new query result of the request if achieving matching. It can reduce the large number of double counting query history, save search time and improve query efficiency. By experimental comparison and analysis show that data based query methods can improve query efficiency.

Keywords Big data, Search, Query network, Cloud database

随着云计算技术的飞速发展,尤其是物联网技术和移动技术的发展,越来越多的数据被人、各种传感设备或者机器所产生。越来越多的应用每个月都会产生TB级别甚至PB级别的数据,与此同时,越来越多的需求每天需要处理几PB、几百PB甚至EB级的数据。如FaceBook、Google+及其国内的人人网等每天需要处理几十PB的社交媒体数据;Google,Yahoo!及百度等每天需要处理上百PB的数据;Twitter与新浪微博等需要处理上亿条的微博记录等。所有这些均对大数据的处理能力和效率提出了更高的需求。

为了较好地处理这些大数据,许多研究机构和公司也开发了相应的新技术或者新架构来面对这些新的需求,主要就是通过提升计算效率的机制即云计算技术来实现对这些大数据的并行处理,其中实现这些并行处理的最基础模型为Google最早用来提高搜索效率的并行编程框架Google MapReduce[1]。后来在Google MapReduce的基础上,Apache提出了属于自己的云计算生态系统架构Hadoop。其中Hadoop生态系统中的Hadoop MapReduce即成为其所对应的并行编程框架。MapReduce的基本核心就是实现了对机器的线性扩张,即随着机器数量的不断增加,可线性地增加其并行计算能力,从而满足用户的查询需求。后来在MapReduce的基础上,又产生了许多新的并行编程框架,它们均对MapReduce框架在一定程度上做了改进,主要有:Twister[2],Haloop[3],Pregel,MapReduce-Merge,Clustera,Dryad[4],Spark[5]以及Hadoop++[6]等等。这些不同的编程模型分别从不同的角度提高了对大数据的处理能力,如Twister及Haloop主要在于提高对于迭代计算的处理。Pregel主要用于提高对图计算的大数据的处理能力;MapReduce-Merge主要在MapReduce的基础上增加了一个合并数据模块,用于改善MapReduce的处理效率等等。最近,由于Hadoop MapReduce第一代在并行共享槽上存在的瓶颈及其它原因,Apache又设计了第二代MapReduce,称之为YARN,用来进一步提高并行处理速度。

虽然针对大数据的处理已经有许多新的技术或者方法,但是如何从PB级这样的大数据中取出满足用户查询需求的一条记录或者合适的网页,仍然需要花费很长的时间,这对于交互式的查询,尤其是即时查询来说,仍然是一个巨大的挑战。虽然现在已经有各种各样的搜索方法[7-9]出现,但是仍然

到稿日期:2012-08-27 返修日期:2012-11-21 本文受国家973计划项目(2011CB302302)资助。
尤川川(1978—),男,博士生,讲师,CCF会员,主要研究方向为复杂网络、软件工程,E-mail:chinayou@vip.sina.com;张桂刚(1978—),男,博士后,副教授,CCF高级会员,主要研究方向为大数据处理。

很难满足这种针对大数据搜索的需求。正是针对这种情况，我们设计了一种新的基于大数据的有效搜索方法。这种新的搜索方法将为大数据的查询，尤其是针对大数据的即时查询提供一种新的思路和方法，对实现交互式的查询具有重要的意义。

1 基于大数据的搜索框架

图 1 展示了一种新的大数据搜索的基本框架。该框架主要根据来自用户的新的查询请求，首先判断历史记录上是否有针对该查询或者部分查询的先例，若有，则可以共享历史查询的结果集合，以减少重新针对大数据的集合进行全部查询带来的时间消耗。

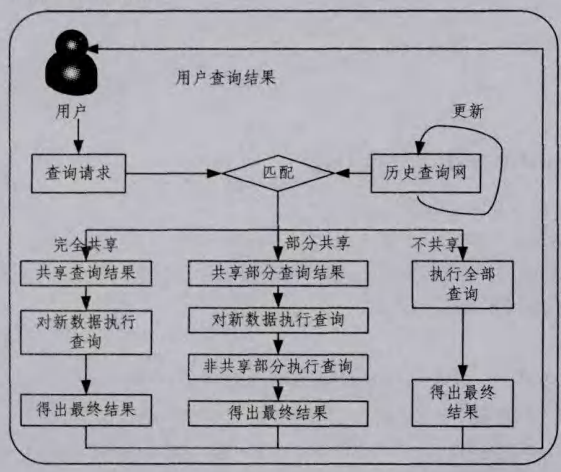

图 1 一种新的大数据搜索框架

该框架包含如下几个部分：

步骤 1 用户首先提出查询请求。

步骤 2 对来自用户的查询请求和历史查询网进行匹配，匹配有如下 3 种情况：

(1)完全共享。若为完全共享，则表明用户新来的查询请求以前同样出现过，这样以前同样的查询所得到的查询结果可以直接为本次查询所利用。其需要执行如下处理即可：

1)共享历史同样查询的查询结果。

2)由于历史查询只是对某段时间以前的数据进行的查询，有可能在该历史查询后又有新的数据记录产生，因此对新增的数据仍然需要执行查询，并得出相关结果。

3)将 1)和 2)的结果进行合并，得到用户所需的最终结果。

(2)部分共享。若为部分共享，则表明用户新来的查询请求以前部分出现过，这样以前同样的查询部分所得到的查询结果可以直接为本次查询所利用。其需要执行如下处理即可：

1)共享历史同样的部分查询的查询结果。

2)由于历史查询只是对某段时间以前的数据进行的查询，有可能在该历史查询后又有新的数据记录产生，因此对新增的数据仍然需要执行查询，并得出相关结果。

3)其中 1)和 2)只是得到了查询相同部分的共享结果，对于查询不相同部分仍然需要执行查询，并得到相应的结果。

4)将 1)、2)和 3)的结果进行合并，得到用户所需的最终结果。

(3)不共享。若为完全不共享，则表明用户新来的查询请求没有任何历史查询记录可供共享，需要重新执行查询。其需要执行如下处理即可：

1)执行全部的查询，并得到相应结果。

2)其中 1)所得到的结果即为用户所需的最终结果。

步骤 3 将用户查询所需的结果反馈给用户。

步骤 4 实现对历史查询网的更新。

2 基于大数据的搜索关键技术

上一节我们分析了一种新的大数据的搜索框架。本节将着重介绍该框架中的几个关键技术，分别为新的查询请求与历史查询网的匹配技术、新增数据查询实现方法及历史查询网的更新方法。

2.1 新的查询请求与历史查询网匹配

图 2 展示了新的查询请求与历史查询网匹配的一个示意图。

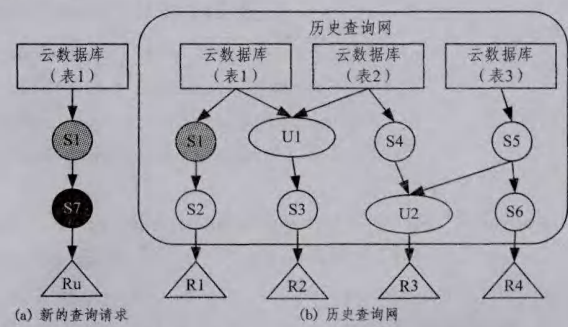

图 2 新的查询请求与历史查询网匹配示意图

大数据的应用一般用云数据库(如谷歌的 BigTable，Apache 的 Hbase 或者其他的云数据库)来存储数据记录，如谷歌用 BigTable 来存储检索关键词所对应的 URL 链接信息等等，Hbase 或者其他的云数据库均可以用于存储 Twitter 或者新浪微博的 Tweets 或者上亿条的微博记录等。

图 2(b)中的 S1，S2，S3，S4，S5，S6，U1 及 U2 为历史查询网中所有的查询条件。同时历史查询网将对应节点 S1，S2，S3，S4，S5，S6，U1 及 U2 的所有的历史查询记录。其中 S1，S2，S3，S4，S5 及 S6 为选择查询节点，U1 和 U2 为联合查询节点，R1，R2，R3，R4 及 Ru 为查询结论节点。而图 2(a)中的 S1 节点在历史查询网中能够找到与之匹配的节点，但是 S7 节点是在历史查询网中不能找到的新的查询请求节点。故通过图 2(a)和图 2(b)的匹配比较分析可知，S1 的查询节点可以共享历史的查询数据结果集，只需要对新增的查询请求节点 S7 执行相应的查询即可，从而可以减少查询时间，提高查询效率。

新的查询请求与历史查询网匹配的匹配算法简要描述如下。

算法 1 新的查询请求与历史查询网匹配

输入：新的查询请求，历史查询网

输出：新的查询请求和历史查询网的匹配情况，即哪些节点匹配成功，匹配不成功的节点为新的查询请求节点。

算法基本思想：

步骤 1 找到与新的查询请求的数据源相同的云数据库表，云数据库(表 1)。

步骤 2 查找云数据库(表 1)下的 S1 节点，找到后做一标记，并跳到

步骤 3：若没有找到，则退出程序。

步骤 3　继续在步骤 2 的基础上往下找节点 S7，找到后做一标记，并跳到步骤 3；若没有找到，则退出程序。

步骤 4　重复步骤 3，直到退出程序。

步骤 5　算法结束。

2.2　新增数据查询实现方法

表 1 展示了新增数据查询实现方法的一个示意图。

表 1　某知名网站的微博记录表

时间	微博记录发送人	微博记录发送地点	微博内容
1 月	A	北京	今天 8 点上班
1 月	B	上海	今天上海小雨
……	……	……	5000 万条
1 月	C	湖南	今天买了新自行车
2 月	D	河北	今天吃了西红柿
……	……	……	1 亿条
2 月	K	山西	考试得了满分
3 月	P	北京	拿到了驾照
……	……	……	9000 万条
3 月	B	上海	又是小雨
4 月	Y	广东	吃了烤鱼
……	……	……	3000 万条
今天	A	北京	今天天气很好

假设用云数据库表来存储所有的微博记录（如使用 BigTable，HBase 或者其他的云数据库）。其中 1 月份有用户在上面发布了 5000 万零 3 条记录；2 月份有用户在上面发布了 1 亿零 2 条记录；3 月份有用户在上面发布了 9000 万零 2 条记录；4 月份到今天有用户在上面发布了 3000 万零 2 条记录。

假设有一个用户需要查询该网站今年的微博记录中包含“自行车”的所有记录，而正好历史上也曾经有人有过同样的查询，但是该人查询的是今年 4 月之前包含“自行车”的所有记录。由于不断有新的微博记录进来，因此我们只能共享该历史查询当时的时间段的所有历史记录，后面新增的数据需要重新进行查询。

云数据库的一个重要特点就是，当有新的记录进来时，所有记录都是以追加的方式补充进来，所以新来的所有微博记录都会按照追加的时间顺序追加在后面，不会像关系数据库那样，其数据可能会插入到历史数据的前面或者中间去。这就为我们追加的记录的获取提供了方便，即只需要找到历史查询当时的最后一条记录即可，新增加部分的微博记录全部按照追加顺序追加在它的后面。因此，可以容易地得到新增加的记录条数，见表 1 阴影部分的数据，即 4 月份到今天有用户在上面发布的 3000 万零 2 条记录。

假设前 3 个月的所有 2 亿 4000 万零 7 条微博记录中包含“自行车”的微博记录条数为 5 万条，那么从这前 3 个月的所有 2 亿 4000 万零 7 条微博记录中搜索包含“自行车”的微博记录条数为 5 万条的所有时间均可以节省下来，只需要从 4 月份到今天有用户在上面发布的 3000 万零 2 条记录中找出满足包含“自行车”的微博记录条数（假设为 8500 条记录）即可，这样大大减小了搜索的范围，可以共享以前的历史查询结果。最后将满足条件的 58500 条满足条件的记录最终反馈给用户即可。

2.3　历史查询网更新方法

图 3 展示了更新后的查询网的一个示意图。

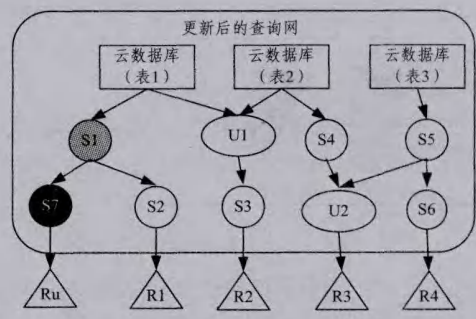

图 3　更新后的查询网

在图 2 中我们分析了新的查询请求与历史查询网匹配的一个关系。分析得知有部分查询可以共享查询结果，但是仍然有新的查询 S7 是历史查询网中所没有的，所以需要将图 2(a)与图 2(b)进行合并，合并后得到的更新后的查询网如图 3 所示。

历史查询网更新算法可以简要描述如下。

算法 2　历史查询网更新

输入：新的查询请求，历史查询网

输出：更新后的查询网

算法基本思想：

步骤 1　找到与新的查询请求的数据源相同的云数据库表，云数据库（表 1）。

步骤 2　查找云数据库（表 1）下的 S1 节点，找到后做一标记，并跳到步骤 3；若没有找到，则将新的查询节点连接到该云数据库（表 1）。

步骤 3　重复步骤 2，直到所有的新节点均已经更新。

步骤 4　算法结束。

3　仿真实验

(1)实验环境搭建

使用 Hadoop 作为实验环境，使用了 4 台机器。其中 1 台 Master 节点，2 台 Slaves 节点，另外 1 台作为 Master 节点的影子节点。4 台机器安装 Linux 操作系统及与 Hadoop 相关的各种组件。

(2)实验方法对比

本实验将本文所设计的大数据查询方法与传统的基于关键词的使用基本 MapReduce 的关键词统方法进行对比。我们具体设计了如下的如题比较机制。

1)简单查询搜索比较。主要使用普通的 MapReduce 计算搜索方法和本文的计算搜索方法实现对大数据查询的比较。

2)复杂查询搜索比较。复杂查询中最为代表性的为 Join 连接查询，它涉及到两个数据集之间进行大数据的计算问题。它的效率直接影响复杂查询能否实行实时查询。

(2)仿真实验

根据上节的实验方法对比的设计，我们设计了如下两组不同的仿真实验。

1)简单查询搜索仿真实验比较

本仿真实验 1 主要验证查询包含“自行车”的搜索时间对比图（使用共享数据和不使用共享数据）。仿真实验结论如图 4 所示。

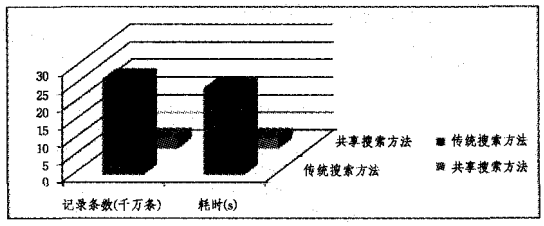

图 4　仿真实验 1 仿真结果

从图 4 可以看出，我们的搜索方法在执行搜索时需首先检查历史查询网，由于以前历史上曾经有过对包含“自行车”的搜索，因此只需要直接对新增部分的记录执行新的搜索即可，以前的记录不需要重新进行搜索，可以直接利用。图 4 的左边展示了新的搜索方法和传统的搜索方法所需要的搜索记录的一个对比，可以看出我们的搜索方法只需要搜索较少的搜索记录。图 4 的右边展示了传统搜索方法和我们的搜索方法在耗时上的一个比较，从该比较可以看出我们的搜索方法由于只需要搜索较少的记录，因此大大地减少了搜索的时间，提高了查询效率。

2)复杂查询搜索仿真实验比较

本仿真实验 2 主要是验证查询两个数据集的 Join 连接。本仿真实验直接从 TPC-H 数据集中选择了两个表进行 Join 连接复杂查询计算。仿真实验结果如图 5 和图 6 所示。

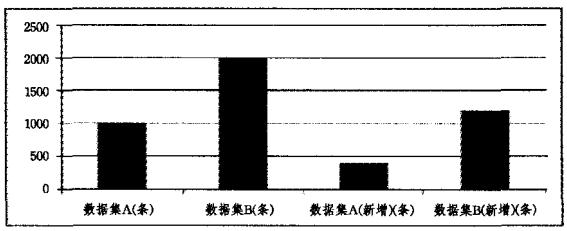

图 5　仿真实验 2 仿真结果(计算记录条数比较)

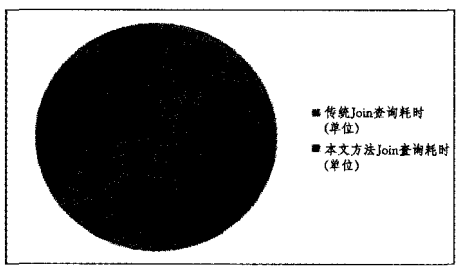

图 6　仿真实验 2 仿真结果(计算时间比较)

从图 5 和图 6 可以看出，使用本文的方法进行 Join 连接的复杂查询，无论计算记录的条数还是最后的计算所花费的时间，均有一定程度的提高。

结束语　本文针对大数据查询效率低下的问题，提出了一种有效的搜索方法。我们的主要设计思想是将共享的历史查询结果作为中间结果集，在新的查询请求到达时，首先与历史查询进行匹配，若能实现匹配，则直接将匹配部分的历史查询结果作为新查询请求结果的一部分。这减少了大量的对历史查询的重复计算，节省了搜索时间，提高了查询效率。最后通过实验对比分析得出，新的基于大数据的查询方法能较好地提高查询效率。

为了进一步提高大数据下的搜索效率，未来的工作主要集中于如下几个方面：

(1)建立一个针对大数据的语义网，增强近似查询的能力。通过语义网，可以实现语义查询的需求。

(2)改善 MapReduce 的计算能力。MapReduce 的 Shuffle 阶段一直是制约 MapReduce 计算能力的一个重要瓶颈，以后要设计一种较为灵活的 Shuffle 处理机制，从而提高其处理能力，实现更为快速高效的搜索。

参 考 文 献

[1] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters[C]// Brewer E, Chen P, eds. Proc. of the OSDI. California: USENIX Association, 2004: 137-150

[2] Ekanayake J, Li Hui, Zhang Bing-jing, et al. Twister: A Runtime for Iterative MapReduce[C]// The First International Workshop on MapReduce and its Applications (MAPREDUCE'10). 2010: 110-119

[3] Bu Y Y, Howe B, Balazinska M, et al. HaLoop: Efficient iterative data processing on large clusters[J]. PVLDB2010, 2010, 3 (1/2): 285-296

[4] Isard M, Budiu M, Yu Y, et al. Dryad: Distributed data-parallel programs from sequential building blocks[J]. ACM SIGOPS Operating Systems Review, 2007, 41(3): 59-72

[5] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: Cluster Computing withWorking Sets[R]. Technology report of UC Berkeley. 2011

[6] Dittrich J, Quian'e-Ruiz J A, Jindal A, et al. Hadoop++: Making a yellow elephant run like a cheetah (without it even noticing)[J]. PVLDB, 2010, 3(1/2): 518-529

[7] 陈国华，汤庸，彭泽武，等. 基于学术社区的学术搜索引擎设计[J]. 计算机科学，2011，38(8)：171-175

[8] 殷哲，曹炬. 带差商信息的云搜索优化算法及其收敛性分析[J]. 计算机科学，2012，39(1)：252-255，267

[9] 杨艺，周元. 基于用户查询意图识别的 Web 搜索优化模型[J]. 计算机科学，2012，39(1)：264-267

(上接第 137 页)

[14] Anisetti M, Ardagna C A, Bellandi V, et al. OpenAmbient: a Pervasive Access Control Architecture[M]. Security in Autonomous Systems, 2006: 160-172

[15] Vassilis K, Loukas H, Dimitris K, et al. A dynamic context-taware access control architecture for e-services[J]. Computers Security, 2006, 25(7): 507-521

[16] Takabi, Amini, Jalili. Enhancing role-based access control model through fuzzy Relations[J]. Information Assurance and Security, 2007, 15(3): 131-136

[17] Zhang S, He D K. Fuzzy model for trust evaluation[J]. Journal of Southwest Jiaotong University, 2006, 14(1): 23-28

[18] 窦文阳，王小明，张立臣. 普适计算环境下的动态模糊访问控制模型研究[J]. 计算机科学，2010，37(9)：63-37

[19] Chen S M, Li W, Liu H C, et al. Multiattribute decision making based on interval-valued intuitionistic fuzzy values[J]. Expert System with Applications, 2012, 39(13): 343-351

[20] Sevastjanow P, Dymova L, Barosiewicz P. A new approach to normalization of interval and fuzzy weights[J]. Fuzzy Sets and Systems, 2012, 198(2): 34-45