大规模数据上的交互式查询

摘要

在本文中，我们将要提出一种支持在大数据上执行交互式查询的框架 BlinkDB。BlinkDB 是一种高度并行化的系统，通过对原数据进行采样，BlinkDB 可以非常迅速地给出查询的近似结果。由于即使在没有完全精确结果的情况下，人们也可以进行正确的决策，因此这样的近似结果也是有意义的。BlinkDB 是一种对 Hive/HDFS 技术栈的延伸与拓展，这些系统支持的选择、投影、表连接以及聚合（SPJA，selection, projection, join and aggregate）操作，BlinkDB 同样可以支持。BlinkDB 能够在保证统计学误差的前提下实时地给出结果，同时它还支持在 PB 级数据和上千台机器的大规模集群当中运行并具有一定的容错能力。我们采用了 TPC-H 指标以及从 Conviva 公司采集的实际数据对该系统进行了实验验证，实验表明在 100 台机器，查询数十 TB 的数据，误差全部控制在 2%到 10%之间的情况下，BlinkDB 运行大部分查询的速度最多能达到 Hive on MapReduce 的 150 倍，达到 Shark (Hive on Shark) 的 10 到 150 倍。

1 绪论

很多公司越来越重视对它们收集来的大量数据进行分析，因为这样可以发现数据中隐藏的规律，从而创造价值。有很多数据分析师们通过在数据上运行一些快速的、探索性的查询来获取他们感兴趣的结论的场景，比如根本原因分析，问题日志分析（比如寻找一个视频网站启动非常慢的原因），还有实时地衡量一个新广告是否有效等等。对着这些查询，实时性比准确性要重要得多。同时，这些查询也是不确定的，并且它们往往涉及到非常庞大的数据。

然而，想要在短时间内在大数据上做查询是一件非常困难的事情，因为硬盘的带宽和内存的大小是有限的，往往单台机器存不下一整份数据；同时，网络通信的延迟也是非常严重的损耗。单个进程的处理能力也是有限的。事实上，一份几 TB 的数据集往往分布在几百台机器上，而单是对这份数据集进行一遍扫描就需要十几分钟的时间，这还没有考虑到在进行大量数据交换的时候网络的损耗。如此长的查询时间将会使得数据分析师在数据上的探索能力受到严重影响。

在本文中，我们提出一种新的支持大数据的近似查询引擎 BlinkDB。BlinkDB 支持 SPJA 形式的查询，并且在 BlinkDB 上运行的聚合查询可以设定误差上界或者运行时间上界。举例来说，考虑一份视频网站的访问历史记录数据，这份数据共有 5 列： SessionID（会话 ID）, Genre（类型）, OS （用户设备的操作系统），City（城市）以及 URL (用户访问的网址)。下面这个查询会返回“western”类型下，每一种操作系统占的记录数量，并且在置信度 95%的情况下保证 10%的相对误差。

```
SELECT COUNT(*)
FROM Sessions
WHERE Genre = 'western'
GROUP BY OS
ERROR 0.1 CONFIDENCE 95%
```

类似地，下面这个查询会在 5 秒钟内给出结果，以及在 95%置信度下估测的相对误差。

```
SELECT COUNT(*), ERROR AT 95% CONFIDENCE
FROM Sessions
WHERE Genre = 'western'
GROUP BY OS
WITHIN 5 SECONDS
```

BlinkDB 的基本原理是预处理出一些原数据的样本，这些样本是通过对历史数据的分析进行挑选的。在执行查询的时候，BlinkDB 会挑出一份最适合该查询的样本，并且在这份样本上执行查询。这些样本都是针对原数据中一些列的分层采样样本。有时候查询涉及到的组有可能出现次数很多，所以 BlinkDB 在这些有偏样本之外，还维护了一份随机采样样本。

注意到要对原数据中所有列的所有子集进行分层采样的话复杂度是指数级的，因此这么做是不可行的，而如果只对历史查询当中出现过的列集合进行分层采样的话，又会限制未来随机查询的处理能力。因此，我们设计了一套优化算法，这个算法可以根据数据分布、过往查询、存储空间限制以及其他一些系统上的参数来给出最佳的用于分层采样的列集合。在执行这套算法以后，BlinkDB 就可以对这些列集合进行分层采样并全部维护起来。这些样本本身是多维的，并且每份样本的大小也不一定相同，这样就使得 BlinkDB 可以根据查询中给定的限制条件选择执行代价最小的样本进行查询。

为了做到向后兼容，BlinkDB 与 HIVE/Hadoop/HDFS 做到了无缝整合。BlinkDB 也可以在 Shark（Hive on Shark）上运行。Shark 是可以在存储层和语言层向后兼容 HIVE 的一个框架，它用 Spark 将数据集缓存在内存中。这样一来，BlinkDB 就可以先把样本缓存在内存中，然后在内存中运行查询，这样只需要几秒就可以完成查询。BlinkDB 是开源的系统，目前已经有一些在线服务提供商对该系统表示有兴趣了。

在本文中，我们将让 BlinkDB 在一个 Amazon EC2 集群上运行，这个集群有 100 台机器。实验中所采用的数据集是一份 10TB 的浏览器会话数据集，该数据集是某个互联网公司的真实数据，其结构与上文当中提到的 Session 数据集非常类似。采用的查询是一个用于分析问题日志的查询集合。我们将同时在 BlinkDB 以及原始的 Hive、Shark 引擎上运行这些这些查询，从而表明我们的系统可以在极短的时间之内给出带有置信区间的预估答案。

2 系统综述

在这一部分中，我们将阐述 BlinkDB 的设计以及关键部分的运行原理。

2.1 系统设计目标和假设条件

BlinkDB 是按照数据仓库的模式设计的，通常数据仓库里面只会有一份大的“实际”表。这张表可能需要跟其他“维度”表进行外键表连接。在实际当中，维度表通常会比实际表小很多，这些表是可以放到集群内存里的。BlinkDB 只会对实际表以及维度表当中用于表连接的列进行分层采样。

另外，由于我们的支持目标是任意查询而不是一组事先给定的查询，所以我们假设这些查询所涉及的的列集合是基本稳定的。事实上，BlinkDB 是基于这样的假设来决定要制作哪些原数据的样本的。我们在一份 Conviva 公司的两年查询历史记录上做的前期分析表明这条假设在实际生产环境当中是成立的。我们不要求对具体涉及列的先验知识，也不要求对查询子句的预测知识。

最后，在本文的实验中，我们只关注一小部分聚集函数：COUNT, SUM, MEAN, MEDIAN/QUANTILE. 不过，事实上这些函数的复合函数我们也可以支持，只要其结果是有解析解的。

2.2 系统架构

系统的整体架构如图 1 所示。BlinkDB 的系统结构是对 Hive 和 Shark 的扩展。Shark 是向后兼容 Hive 的，并且跑在 Spark 上，其中 Spark 是一个分布式计算框架，它将运算中间结果保存在内存中。BlinkDB 向其中增加了两个主要的组件：（1）一个创建和维护样本的组件，以及（2）一个预估查询运行时间和准确度，以及根据限制条件挑选最合适的样本的组件。

2.2.1 样本的创建和维护

这个部分主要负责创建以及维护一组样本，这些样本包括随机采样样本和分层采样样本。当查询涉及到的列分布比较均匀的时候，我们就用随机采样样本回答查询，否则我们就用分层采样样本回答查询。

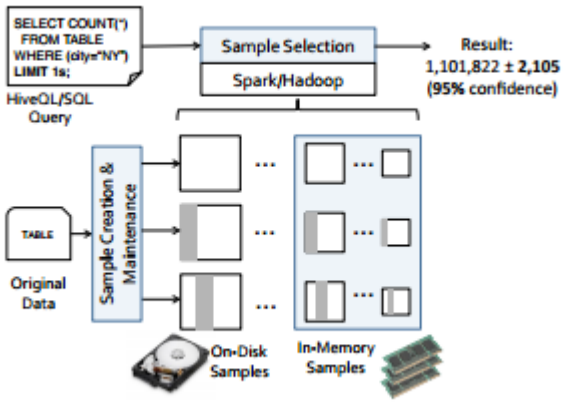

图 1 BlinkDB 的系统结构

样本是根据原始数据的统计数据以及查询的历史记录创建并且更新的。BlinkDB 会创建并维护一份随机采样样本和多份分层采样样本，其中分层采样样本的分层基准列是根据一个组合优化算法决定的。这个算法会找出多组列集合，使得这些列集合（1）对于历史查询数据的效果最好，并且（2）包含尽可能多的离群值，因为这些离群值通常不会出现在随机采样的样本中。当查询数量变化或者是数据有更新的时候，这些样本也会更新。

2.2.2 运行时的样本选择

在运行一个查询时，我们首先根据查询给定的限制条件选择一个或多个最佳的样本来运行查询。这些样本的挑选有两个依据，一是通过一些事先算好的统计数据挑选，二是 BlinkDB 会动态地在多个更小的样本上运行该查询，从而判断该查询的复杂性和选择性，然后根据这些进行挑选。这样的流程可以帮助优化模块找到最佳的查询计划以及最佳的样本数据。

2.3 一个实例

为了演示 BlinkDB 的运行流程，请考虑图 2 的这个例子。这个例子当中的表包含 5 列：SessionID, Genre, OS, City 和 URL。

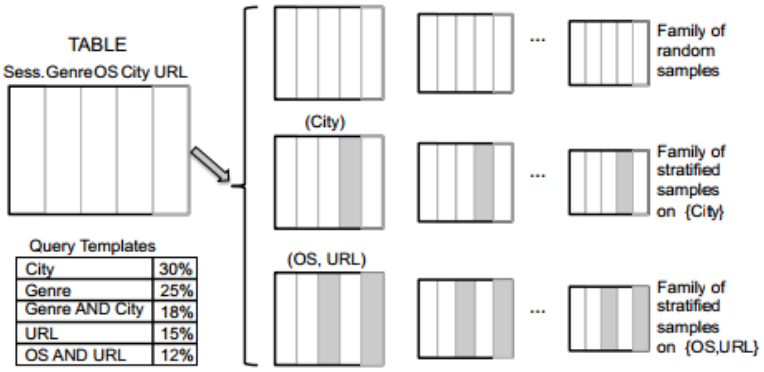

图 2 一个有 5 列的表以及给定查询的例子

图 2 左下角的表显示了一些查询列组合以及它们所占的频率。给定这些查询列组合以及存储空间上限，BlinkDB 会根据这些数据创建一些原数据的样本。这些样本按照“样本族”组织，每个样本族里面包含不同粒度的一组样本。在这个例子里，BlinkDB 决定创建两个样本族：一组是关于 City 列的，还有一组是关于 OS 和 URL 列的。

对于每一个查询，BlinkDB 会选择一个合适的样本族以及一份合适大小的样本来回答查询。总的来说，在 WHERE 和 GROUP BY 子句当中涉及到的列不一定会恰好与一份分层采样样本吻合。为了解决这个问题，BlinkDB 会先在每个样本族当中最小的那个样本上运行这个查询，然后利用得到的结果决定采用哪个样本用来真正运行这个查询。

3 性能测评

本部分主要介绍 BlinkDB 在各种指标上的性能表现。我们的实验数据集是一份来自 Conviva 公司的真实数据，该数据的内容是关于视频网站内容分布的，大小为 17TB。这份数据分布式地存储在 100 台 Amazon EC2 节点上，我们的查询集合是原始历史查询集合当中的一个子集。这份数据包含大约 55 亿行，104 列。

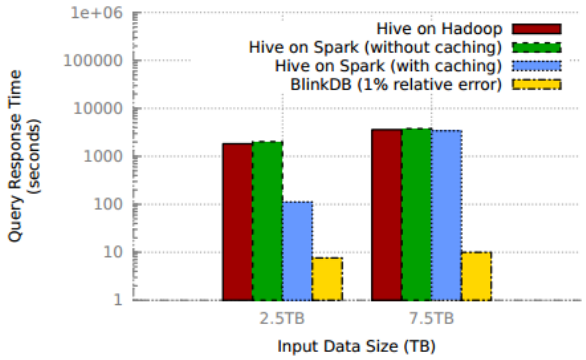

图 3 Hive，Shark（有无缓存），以及 BlinkDB 在简单聚合函数上的响应时间比较（取对数）。

3.1 采样的必要性

我们比较了 BlinkDB 和直接在原数据上运行查询的处理框架之间的性能差异。在这个实验中，我们使用了原数据集的两个子集，其大小分别为 2.5TB 和 7.5TB，这两个数据集都散布存储在 100 台机器上。值得注意的是，之所以要选择两个子集，是因为这样有利于演示并行处理框架和现代高内存集群之间的相互利用关系。具体到这个例子来讲，2.5TB 的数据是可以直接缓存在我们的集群内存中的，而大于 6TB 的数据就必须把一部分数据放在硬盘中。我们采取的查询是一个非常简单的查询：在 date 列上做一个过滤，然后在 city 列上做 GROUP BY，计算的是平均的用户会话时间。通过这个查询我们就可以看出，即使是针对如此简单的查询，采样在降低响应时间方面也是非常重要的。我们比较了 Hive on MapReduce，Hive on Spark（也就是 Shark）这两种精确查询框架所用的时间，以及 BlinkDB 这种近似查询框架所用的时间。其中，Shark 是否使用缓存的选项也被考虑在内，BlinkDB 的相对误差设定为 1%，置信度设置为 95%。其结果如图 3 所示，注意图 3 的 Y 轴是对数坐标。由图可见，无论在何种情况下，BlinkDB 的效率都远超其他几种方式，大约提升了 10 到 150 倍不等。其原因主要是因为 BlinkDB 比起其他几种方式读取的数据量要少得多。对于两种数据大小，BlinkDB 都能在几秒之内返回结果，而其他几种框架都必须花几千秒才能返回结果。在数据大小为 2.5TB 的时候，Shark 的缓存机制起了很大的作用，在使用缓存的时候查询时间只有 112 秒。但是当数据大小为 7.5TB 的时候，很大一部分数据必须存在硬盘上，这个时候缓存机制就不能很好地优化查询时间了。

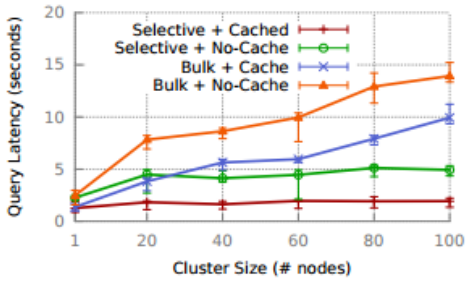

图 4 查询时间随集群大小的变化

3.2 大规模化

在本部分中，我们测试 BlinkDB 在大规模集群下的表现情况。我们使用了两组不同的查询集合，每组查询集合包含 40 条类似于 Conviva 公司历史记录的查询。第一组查询叫做 selective 查询，这些查询的特点是它们只涉及输入数据当中的一小部分。这些查询在实际的查询集合中是比较常见的，只要 WHERE 子句的条件过滤性比较强，那么一个查询就是 selective 的。第二组叫做 bulk 查询，这些查询涉及的数据量很大。第一组查询涉及到的数据可能只存储在几台机器上，而第二组涉及到的数据通常存储在大量的机器上，这样运行第二组查询时的网络通信损耗要比第一组大得多。图 4 显示了这两种查询的查询耗时随着集群大小的变化趋势。每个查询都在 100n GB 的数据上运行，其中 n 是集群的大小。比如集群当中有 10 台机器，那么每个查询就在 1TB 的数据上运行，如果集群有 100 台机器，那么每个查询就在 10TB 数据上运行。并且，对于每组查询，我们都记录两种情况下的耗时：第一种是原数据的样本完全缓存在内存中的情况，第二种是这些样本完全存储在硬盘上的情况。由于在现实情况中，每个样本都有可能是一部分存储在内存中，一部分存储在硬盘中的，因此这两个结果分别显示的是查询耗时的最大值和最小值。

4 实验细节

BlinkDB 的实验主要演示在 100 台机器的 Amazon EC2 集群上的整体实现。在本部分中我们将演示 BlinkDB 对一部分的任意探索性查询的支持。

4.1 实验设置

本实验中，我们采用 BlinkDB 的一份开源实现并部署在 100 台 Amazon EC2 机器上。为了再现现实世界当中的根本原因分析这一情景，我们事先将一份大约 10TB 的视频会话日志表存储在了这个集群里。

为了更加有效地演示这一系统，我们在本实验中提供了一个基于 AJAX 的网页前端，如图 5 所示。用户可以使用这个前段来快速提出查询，并且可以对数据的任意一部分进行可视化。除此以外，我们也提供一个 100 台机器的 Apache Hive 集群同时运行以方便比较二者差异。

图 5 一个根本原因分析的案例，此案例中希望分析 Kensington 的延迟数据中尖刺出现的原因

4.2 实验情景

我们的实验情景让参与者扮演 CalFlix 公司的质量监督工程师角色。CalFlix 公司是一家知名的视频内容传播公司，但是这家公司最近遇到了一些视频质量上的问题。一部分用户报告称他们的网站启动时间特别长，并且缓冲时间也特别长，这使得他们客户的满意度和效益都受到了影响。

为了阻止效益的进一步流失，该问题需要在尽可能短的时间之内查出根本原因。不过，造成这个问题的原因有很多，比如在某个地区的 ISP 或者边缘缓存可能负载过大，或者视频内容当中有一部分访问量很大的部分损坏了。要诊断这个问题，工程师需要对数十列的原数据进行切分，来逐个排除原因。这些维度可能包括客户端使用的操作系统，浏览器，固件版本，设备类型，地理位置，ISP，CDN，访问内容等等。工程师必须通过对这些数据的分析，来找到遇到该问题的用户的共同点。

本实验的参与者可以利用网页前端来交互式地看到原数据当中的一系列指标。第一层的面板展示的是整个数据的统计指标，不过用户可以通过网页来缩小数据范围，只计算一部分列上的这些统计指标。

4.2.1 数据特征

我们的这份 10TB 数据是一份未经过标准化的视频访问数据，包含 100 多列，其中包括会话 ID，起始和终止时间，内容 ID，操作系统版本，ISP，CDN 等等。该数据存储在 HDFS 上，实际存储于 100 台机器上。数据当中的每一条记录都是一个访问记录，例如开始或者停止播放一段视频，缓冲，调整比特率，播放广告等等。原数据中大约有几十亿条这样的记录。

4.2.2 根本原因分析的场景

为了演示几个原因分析的场景，我们在这份数据里面隐藏了一些人造的问题。这些问题如下所示：

1. **响应时间尖刺：**在该情境中，”XYZ” ISP 在“Kensington, CA”的用户在观看“Star Wars Episode I: The Phantom Menace”的时候，会遇到响应时间的突然增加。造成这个问题的原因是这个地区的边缘缓存过载了。
2. **视频播放器的 Bug：**在该情境中，“FireFox 11.0”的用户缓冲时间会特别长，因为他们使用的开源软件“ZPlayer”近期的一份 beta 版是有 bug 的。
3. **内容损坏：**在该情境中，在一台 CDN 服务器上的一份特定的数据副本损坏了。这影响到了访问该部分数据的一些用户。

要诊断这些问题，用户就必须交互地提出探索式询问。这些询问通常包括 WHERE 和 GROUP BY 子句，涉及到原数据当中的一列或者几列。我们希望用户可以在分钟级别的时间之内诊断出以上全部三个问题的根本原因，这样，BlinkDB 在支持大数据交互式提问上面的高效率也就得以演示了。

5 相关工作

已经有很多人研究过近似查询处理（Approximate Query Processing，AQP）这一领域。过去研究所采取的方法主要有两种，一种就是我们所采用的基于采样的方法，另一种就是采用如下文介绍的其他数据结构。

关于利用随机采样和分层采样来给出近似回答的研究已经有很长的历史了。一个时间上比较近的例子是 SciBORQ。SciBORQ 是一个用于科学领域的数据分析框架，它利用一种叫做 impressions 的特殊结构来回答询问。Impressions 实际上也是对原数据的有偏采样，它是根据每条记录与历史查询结果当中的距离来决定采样哪些记录的。与 BlinkDB 不同的是，SciBORQ 不支持误差限制，也不提供相对误差的保证。

除了采样以外，也有很多对原数据的压缩方法，例如小波分析，直方图等等。这些方法都可以回答特定种类的查询。类似地，也可以通过创建数据方块（Data Cube）的方式来快速回答特定种类的查询。不过，虽然这些方法能够保证回答查询的速度，但是它们必须对每一种查询运算都建立一份特殊结构，否则是无法应对任意形式的查询的。另外，这些技术与我们的工作之间是不矛盾的，BlinkDB 完全可以在应对特定的查询的时候使用这些技术来回答，同时对于一般的查询仍然使用采样技术来回答。

致谢

本论文的研究受到了 Qualcomm, Google, SAP, Amazon Web Services, Blue Goji, Cloudera, Ericsson, General Electric, Hewlett Packard, Huawei, IBM, Intel, MarkLogic, Microsoft, NEC Labs, NetApp, Oracle, Quanta, Splunk, VMware 与 DARPA 公司的支持。（合约号 #FA8650-11-C-7136）.

6 参考文献

[1] Apache Hive Project. http://hive.apache.org/.

[2] Conviva Inc. http://www.conviva.com/.

[3] S. Agarwal, S. Kandula, N. Bruno, M.-C. Wu, I. Stoica, and J. Zhou. Re-optimizing Data Parallel Computing. In NSDI, 2012.

[4] S. Agarwal, A. Panda, B. Mozafari, S. Madden, and I. Stoica. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. Technical Report, http://arxiv.org/abs/1203.5485, 2012.

[5] N. Bruno, S. Agarwal, S. Kandula, B. Shi, M.-C. Wu, and J. Zhou. Recurring Job Optimization in Scope. In SIGMOD, 2012.

[6] C. Engle, A. Lupher, R. Xin, M. Zaharia, M. Franklin, S. Shenker, and I. Stoica. Shark: Fast Data Analysis Using Coarse-grained Distributed Memory. In SIGMOD, 2012.

[7] M. Garofalakis and P. Gibbons. Approximate Query Processing: Taming the Terabytes. In VLDB, 2001. Tutorial.

[8] M. L. Kersten, S. Idreos, S. Manegold, and E. Liarou. The Researcher's Guide to the Data Deluge: Querying a Scientific Database in Just a Few Seconds. PVLDB, 4(12), 2011.

[9] L. Sidirourgos, M. L. Kersten, and P. A. Boncz. SciBORQ: Scientific data management with Bounds On Runtime and Quality. In CIDR'11, pages 296–301, 2011.

[10] M. Zaharia et al. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In NSDI, 2012.

[11] K. Zeng, B. Mozafari, S. Gao, and C. Zaniolo. Uncertainty Propagation in Complex Query Networks on Data Streams. Technical report, UCLA, 2011.