流式数据查询系统①

王 栋[1,2], 张 潇[1,2], 武延军[1]

[1](中国科学院软件研究所 基础软件国家工程研究中心, 北京 100190)
[2](中国科学院大学, 北京 100190)

摘 要: 近年来, 随着互联网和物联网的快速发展, 海量的数据在很多应用中都会出现, 而这其中有很大一部分数据是以流数据的形式存在的. 流数据的特点是快速、大量、无序, 并且要求快速的响应. 研究表明, 传统的关系型数据库并不适用于这种流式数据的应用场景, 因此如何开发出一套新型的数据查询系统来满足流式数据的处理需求就成为当前研究的一个热点课题. 本文借鉴当前几个有代表性的流式数据管理系统的优点, 分析流式数据查询系统的关键问题, 综合考虑流数据接口定义、数据预处理, 查询语言定义、查询执行过程, 系统监控、系统界面等问题, 设计并实现一个可用的流式数据查询系统. 最后, 通过采集具体的新闻流式数据验证系统的各项功能和性能, 实验结果表明, 该流式数据查询系统具有良好的数据查询性能.

关键词: 流数据; 连续查询; 滑动窗口; 负载平衡

Streaming Data Query System

WANG Dong[1,2], ZHANG Xiao[1,2],WU Yan-Jun[1]

[1](Institute of Software, the Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100190, China)

Abstract: In recent years, with the rapid development of the Internet and the Internet of Things, the mass of data in many applications will appear, and a large part of the data is in the form of streaming data. Streaming data is characterized rapid, massive, disorderly, and also requires quick response. The research shows that the traditional relational database is not suitable for the application of the streaming data. Therefore, how to develop a new type of data query system is a hot topic in the current research. In this paper, it uses the advantages of several prevalent data management systems, analyzes the key problems of the streaming data query system and consideres the definition of data interface, data preprocessing, query language definition, query execution process, system monitoring, system interface and other issues to design and implement an available streaming data query system. Finally, it evaluates the system by the specific news streaming data. The results show that the streaming data query system has a good data query performance.

Key words: streaming data; continuous query; slide windows; load balance

1 引言

一系列连续且有序的点组成的序列 $S_1,S_2,\ldots S_n$,称为流式数据, 它们按照固定的次序产生, 具有特定的时间间隔, 一般不被人为终止或扰乱[1]. 流式数据与传统的数据之间有很多的不同, 主要表现在数据产生方式、接收数据处理方式、数据管理模式等多个方面. 传统的关系型数据一般都是以持久的形态存在着, 在查询方式上往往一次单次查询就能满足要求, 访问方式上也都是随机的访问, 数据存储在无限的磁盘空间里, 数据更新速率较低, 很少需要实时的查询服务; 反过来, 流式数据之间的关系不再持久, 数据都是以瞬时的流达到系统, 且查询服务一般预置在系统里, 通过

① 基金项目:国家自然科学基金面上项目(61432001);国家自然科学基金重大项目(91218302)

收稿时间:2015-12-30;收到修改稿时间:2016-01-28 [doi:10.15888/j.cnki.csa.005315]

连续的查询满足用户的需求, 在访问方式上也是序列化的访问, 连续不断的数据流也不能完全存储在磁盘空间里, 为了达到实时的响应需求[2], 一般会把最近最新的数据存储在访问速度更快的主存中, 一些暂时不使用的历史数据才会存储到外部磁盘空间里, 除此之外, 流式数据的传输速率也无法做到提前预测, 随着外部环境不断的变化, 这就要求系统能很好的应对这种未知的变数.

智慧城市、智能交通、金融系统、网络监测、电信数据来往、Web 服务日志、天气预测等这些动态环境中产生的信息构成了连续不断的流式数据[3,4]. 依据不同的应用形式, 可以将流式数据区分为事物型和度量型两种类型, 其中事物型流式数据指的是产生于实体之间的交互日志, 比如大型网站的访问点击日志、金融交易的信息记录等; 而度量型流式数据指的是来自监控某个实体的状态, 比如传感器的采集数据, 气象观测数据等. 由此可见, 流式数据已经逐步应用到科学研究、商业应用、居民生活等多个领域, 并且正在发挥着越来越大的作用. 然而由于受流式数据量大、无序等特征的影响, 使得数据管理模式、处理方式以及数据分析等方面均面临着诸多的技术挑战[5].

在 Web 日志在线分析中, 用户访问大型网站(Google、Baidu)时的访问足迹通常都会被网站记录下来, 而这些日志数据构成了连续不断的流式数据. 这些流式数据中包含用户的访问时间、访问协议、访问方式、用户的 IP 等等的数据, 通过流式数据查询技术可以从这些流式日志数据中快速发现用户的关注热点从而有助于提供个性化服务和改善网站的总体性能[6]. 此外基于传感器监控数据的分析在许多场景中起到了非常重要的辅助决策作用, 比如自然环境数据监控[7]就是目前的研究热点之一. 本文正是针对流式数据的应用场景, 设计并实现一个高效可用的流式数据查询系统, 针对流式数据的快速查询在许多应用中都非常有用, 本文的主要贡献包括: 1)提出一种流式数据查询语言, 定义相应的语法规则, 并通过相关的实验证明其有效性; 2)研究并实现了基于内存的流式数据查询系统; 3)通过大量实验数据证明其高效性和稳定性.

2 相关工作

流式数据具有无限量大小的特征, 并且需要及时处理并返回计算结果, 否则一旦数据过时就很难获取到历史数据, 或者即使拿到历史数据对当前的决策意义也不大. 而在有限的主存中无法容纳大容量的流式数据, 为了解决这个矛盾, 现有的计算方法通常通过数据过滤、基于语义的约束、数据近似、概要存储等途径将无限的流式数据映射到可管理的存储空间中进行处理.

最近几年来, 国内外在流数据管理领域开展了许多工作, 也取得了不少研究成果, 对促进流式数据查询的发展起到了很好的推动作用, 目前代表性的数据流管理原型系统有 Aurora[8,9]、STREAM[10]、Fjords 等.

STREAM 是斯坦福大学研制的数据流管理系统, 该系统可支持扩展查询语言, 对标准 SQL 语言的进行了扩展, 在 FROM 子句中加入了流, 还加入了滑动窗口的定义, 从而增加了语言表达能力; 它可以处理大量持续的数据查询, 并且可在资源有限时提供近似的查询结果. Fjords 是加利福尼亚大学开发的传感器数据采集系统, 它主要应用于公路交通状况监测, 通过计算传感器数据获取机车的行驶速度. 系统允许用户发起对数据流的查询, 查询结果支持传感器送来的数据和用户获取的传统数据相结合. Aurora: Aurora 是由美国布兰代斯大学、布朗大学和麻省理工学院联合开发的一个数据流管理系统, 它具有面向流的操作集合并支持实时操作. Aurora 是一种流处理系统, 它采用了类似于工作流的查询语言, 其最重要的结构是 Box 和 Arrow, 每一个 Box 代表一个操作符(Operator); 每一个有向的 Arrow 代表数据流从一个操作符传递到另外一个操作符. 一个 Aurora 的查询定义为由操作符组成的有向无环图[11].

而在国内, 流数据研究的起步稍微晚了点, 目前的参考文献也较少, 在一些学校和研究所正在对流数据展开研究. 复旦大学的研究集中在流数据的管理和挖掘方面, 中科院计算尝试构建一个面向网络信息安全的数据流计算模型, 重点研究网络数据流的获取, 网络数据流建模, 网络数据流查询模型等问题.

随着这几年大数据浪潮的兴起, 数据挖掘、机器学习等学科领域也相继应运而生, 这些学科的核心问题都是对数据进行分析处理, 即随着应用需求的拓展, 以数据为中心的分析手段作用日益明显, 数据分析已经成为一种核心竞争力, 用户迫切希望从大量的流式数据中发现隐藏在其背后的有用信息, 进而能够为辅助决策提供依据. 所以对流式数据的各项研究也变得

意义重大.

3　系统概述

为了实现一个完整的流式数据查询系统, 需要考虑的问题包括: 流式数据接口的设计、查询模块的设计与实现、系统监控的设计以及负载平衡的保证. 为此本系统的实现分为四个模块: 数据采集模块、数据查询模块、系统监控模块、数据管理模块. 系统整体结构图如图 1 所示.

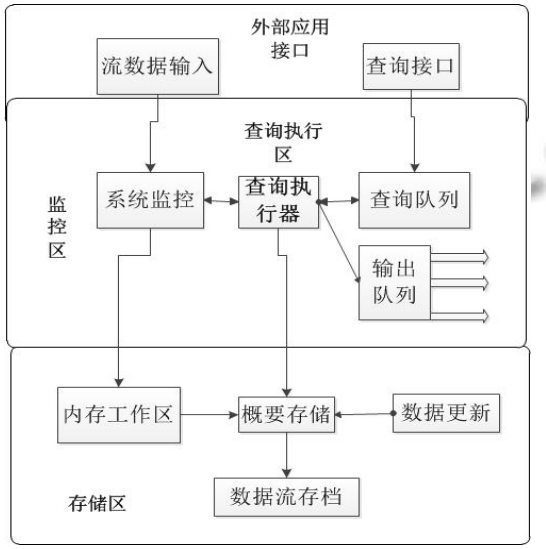

图 1　系统架构图

数据采集模块: 即图中的流数据输入, 该模块是整个数据流系统中信息处理的最前端, 负责向系统发送连续的数据流, 要求所设计的模块能够对指定的数据进行采集并及时传输, 此外还要求该模块能对采集到的数据进行初步的处理以达到系统要求的数据格式.

数据查询模块: 包含查询接口和查询执行区部分, 该模块是系统的核心, 主要完成对流数据的各种查询操作. 查询执行器会从查询队列中获取连续查询语句, 然后对查询语句进行语法分析, 确认无误后对语句进行拆分, 最后执行查询动作, 输出最终的结果.

系统监控模块: 架构图中的监控区, 完成对数据流的监控以及对系统内存、CPU 等资源的监控, 因为数据是在内存中完成查询, 所以要时刻关注系统的内存变化.

数据管理模块: 图中所示的存储区, 当内存中的数据足够多或者需要将一些历史数据存档的时候则由数据管理模块去完成.

4　各模块的算法设计与实现

4.1 数据采集模块

针对不同的应用场景, 数据采集模块的设计各不相同. 目前出现的流数据应用场景包括智能交通、金融系统、网络数据监测、电信数据来往、Web 服务日志、天气预测、新闻流式数据等, 为了不失通用性, 系统支持单独的数据采集模块, 该模块只需负责流式数据接口的定义, 数据流的预处理、数据流在内存中的存储定义等工作, 系统数据流接口完成这些工作之后可将数据传输到后续模块中进行查询、监控、负载平衡等步骤的处理.

为了对系统进行验证, 本文采用的实验数据源是网上的新闻数据, 通过实时采集腾讯、新浪、网易三大门口网站上的新闻数据作为流式数据源输入到系统中. 因为门户网站上的新闻数据是以时间序列不断的产生的, 系统的数据采集进程会不断的去抓取新闻, 形成一条稳定的数据流. 新闻数据抓取流程[12]如图 2 所示.

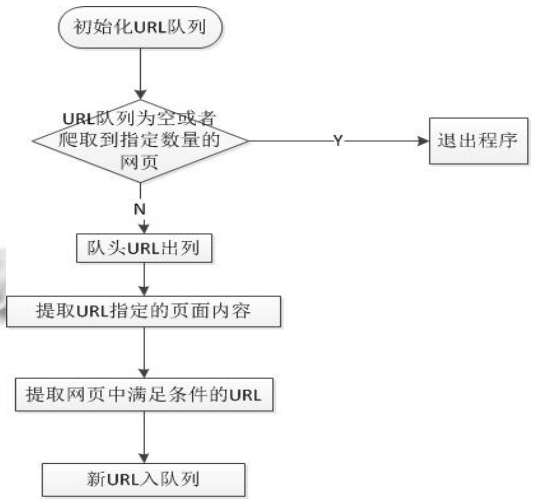

图 2　流数据抓取流程图

系统的抓取进程首先获取到初始化的 URL 队列, 即网站新闻首页的 URL 链接集合, 然后对该队列进行遍历, 对每个 URL 链接对应的页面进行解析, 从中抽取到符合条件的 URL, 把新的 URL 放入到队列中, 直到满足抓取的停止条件. 具体到每个新闻网页, 会从中抽取新闻的标题、时间、来源、正文、评论数等内容. 在这里引入了一个 JAVA 的开源包:

HTMLParser[13]. HTMLParser 是一个 JAVA 写的 HTML 解析库, 它不依赖于其它的 JAVA 库文件, 主要用于改造或提取 HTML, HTMLParser 具有小巧, 快速的优点, 它能超高速解析 HTML 从而能快速便捷的获取网页中的文本内容.

4.2 数据查询模块

数据查询模块是系统的核心, 该模块完成的工作包括查询语言的定义与解析, 查询过程的实现等, 处理流程如图 3 所示.

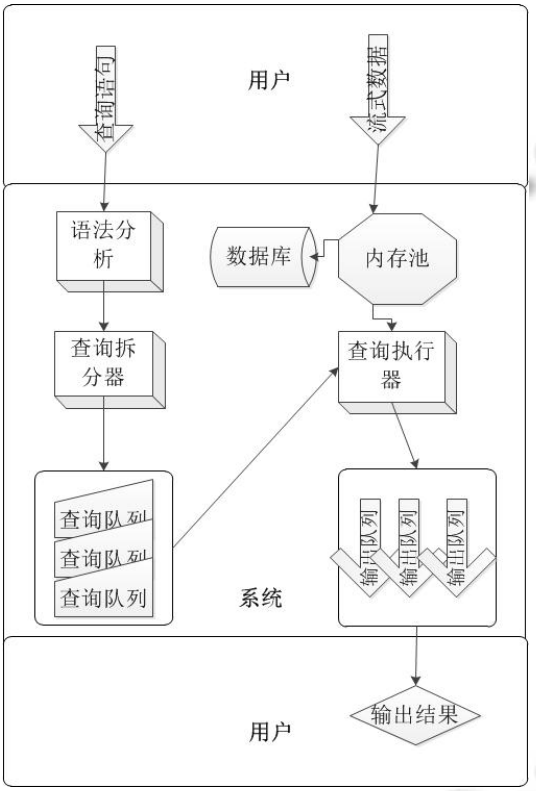

图 3 数据查询流程图

4.2.1 查询语句

系统定义的查询语句由两部分构成, 第一部分为传统的 SQL 语句, 第二部分声明一个滑动窗口[14], 由一个 for 循环实现, 语法结构如下:

```
for(t=init; continue(t), update(t))
{
    window(Stream1, start(t), finish(t));
    window(Stream2, start(t), finish(t));
    …
}
```

根据不同的查询需求, 可将查询分为以下三种:

(1) 快照查询

顾名思义, 快照查询一般是仅执行一次的查询, 在逻辑上类似于传统的关系数据库查询, 比如:

```
Select newsTitle, newsTime  From  News
Where newsWeb=’tencent’ and newsComment > 2000
For(t=0;t==0;t=-1)
{
 window(News, 2015-12-06, 2015-12-07)
}
```

该查询表示查找 2015 年 12 月 6 号和 7 号这两天评论数大于 2000 的腾讯新闻, 因为该查询语句中的 for 循环只累加一次, 故该查询只执行一次.

(2) 标记查询

标记查询的输入是一个时间窗口, 时间窗口的起始端从一个固定的时间点开始, 而结束端随着时间的推移不断向前移动, 比如:

```
Select newsTitle, newsTime From News
Where newsWeb=’tencent’ and newsComment > 2000
For(t=0; t<=200; t++)
{
window(News, 2015-12-06, t)
}
```

该连续查询表示从 2015 年 12 月 6 号开始查找评论数大于 2000 的腾讯新闻. 该查询会连续执行 200 次, 也就是持续 200 天.

(3) 滑动查询

滑动查询的输入也是一个时间窗口, 与标记查询不同的是该类查询的两端都随着时间的推动而不断向前滑动, 比如:

```
Select newsTitle,newsTime From News
Where newsWeb=’tencent’ and newsComment > 2000
For(t=0; t<=50;t+=5)
{
window(News, 2015-12-06+t, 5)
}
```

该查询表示从 2015 年 12 月 6 号开始每隔 5 天查询一次最近五天内评论数大于 2000 的腾讯新闻. 该连续查询持续 50 天, 共执行 10 次.

4.2.2 查询执行过程

当用户输出查询语句, 首先进行语法分析, 如果确认无误, 再进行语义检查, 如果语义也检查成功,

则会生成一个查询命令. 接着查询拆分器会对该查询命令进行拆解, 将其分成两部分, 第一部分是 SQL 语句, 第二部分是滑动窗口, 即 for 循环. 最后查询执行器会拿到拆解后的逻辑去内存中检索出符合条件的结果, 输出给用户.

查询模块的界面如图 4 所示.

图 4　系统查询界面

为了提高查询的执行效率, 也为了支持更多的查询语句, 提高系统的通用性, 系统采用了 yacc 和 lex 这两个工具对输入的 SQL 语句进行解析, 进而进行语法分析和语义检查. Lex 作用就是从 SQL 语句中提取单词, 即提取各种保留字、操作符等语言的元素. yacc 做的工作叫做 syntactic analysis, 即语法分析. lex 提取了单词, 那么剩下的语法表达工作就由 yacc 来完成. 此外, 针对不同的应用场景中不同维度的流式数据, 采可用差异化的存储策略来对数据进行存储, 进而保证对数据的高效查询, 以提高系统在不同环境中的适用性.

4.3 系统监控模块

该系统接收来自外部持续不断的数据流, 然后对这些流数据进行查询处理. 如果输入的数据流速率超过了系统的处理极限时, 系统会负荷过重, 如果不及时进行处理会严重影响系统的性能直至崩溃. 所以需要对整个系统的流量尤其是机器的 CPU、内存进行有效的监控, 在问题爆发前及时预警.

系统每五分钟会监测机器的内存和 CPU 的使用率, 当使用率超过阈值的时候及时通知系统管理员进行相应的处理. CPU 利用率监控图如图 5 所示.

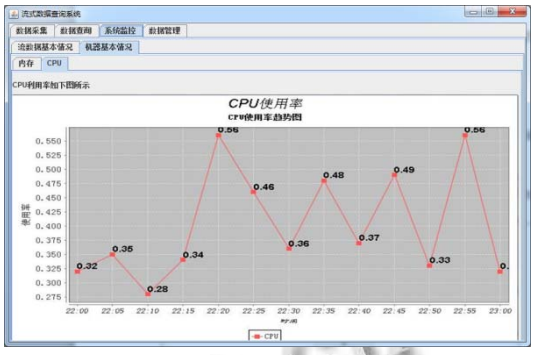

图 5　系统监控界面(CPU)

4.4 数据管理模块

该模块与系统监控模块互相协作, 当系统的负载过大时进行相应的卸载操作, 流程如图 6 所示.

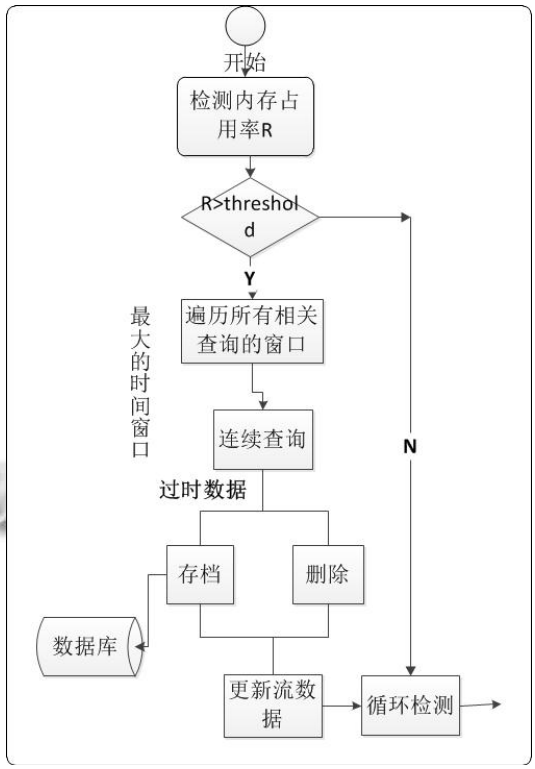

图 6　内存监控流程图

同样的, 第一步系统会有专门的进程来负责负载监控, 以内存为例, 系统的监控程序会每隔五分钟去检查当前内存的使用率, 监测当前系统的内存使用率是否大于预先设定的阈值(80%), 该阈值是系统能够正常高效工作的临界点.

如果第一步的监控进程发现内存的使用率高于

80%了, 接下来就需要进行卸载操作, 所谓的卸载操作即清除内存中的过时的数据或者历史数据, 如何定义历史数据呢? 即内存中不再使用的数据就是历史数据, 或者说超出时间滑动窗口的数据就是历史数据, 比如一个查询是要查找最近 1 个小时内评论数大于 1000 的腾讯新闻, 那么一个小时前的腾讯新闻数据对于这个查询来说就是历史数据了. 进行卸载的时候可以选择两种操作, 一种是直接从内存中把历史数据删除, 另外一种是把历史数据保存到外部的磁盘存储空间, 便于以后分析使用.

在具体确定对哪些数据进行卸载时, 监控进程会去查询队列中遍历所有连续查询的窗口, 从中找到时间窗口的最大的连续查询, 然后对内存中的过时数据进行卸载操作, 之后更新内存中的数据流. 最后进入下一轮循环.

5 系统验证

5.1 实验设计

如上文所述, 实验的数据来源于三大门户网站的新闻数据, 每一条新闻数据都是一个对象, 该对象包括的属性如表 1 所述.

表 1 新闻流式数据属性

属性名	属性描述	属性数据类型	属性说明
newsTitle	新闻标题	String	新闻的标题
newsTime	新闻时间	Date	新闻的发布时间
newsFrom	新闻来源	String	新闻的发布机构
newsComment	新闻评论	Int	新闻的评论数
newsText	新闻正文	String	新闻的正文
newsWeb	新闻站点	String	新闻的来源站点
newsCrawTime	新闻抓取时间	Date	新闻的抓取时间
newsLink	新闻链接	String	新闻的 URL 链接

系统部署的机器硬件环境为内存 12GB, 处理器为英特尔(R) Core(TM) i7-2600 CPU @ 3.40GHz, 四核.

为了验证流式数据的查询性能, 本文会将数据存储一份到数据库中, 然后对两者执行同样的查询语句来比较差异, 数据库采用的版本是 mysql 5.3. 在实验中抓取的新闻数目约为 40 万条.

5.2 实验分析

5.2.1 普通查询

首先比较两者在一般查询上的性能差异, 执行的查询语句是: select * from news where newsComment > 1000. 即查询评论数大于 1000 的新闻. 两者的性能差异如下图 7 所示.

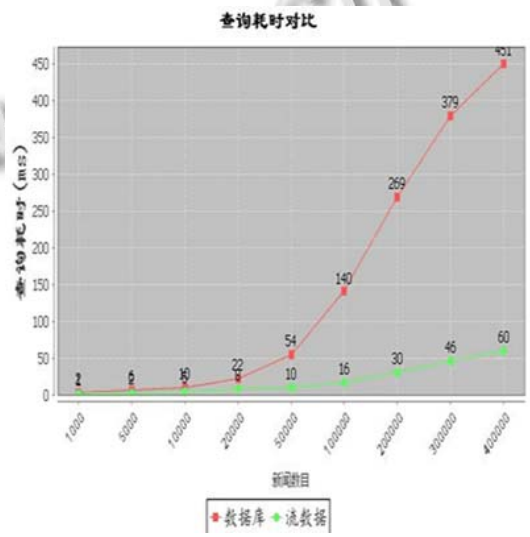

图 7 一般查询耗时对比

图中的横坐标表示新闻的数目, 纵坐标表示查询的耗时, 以毫秒为单位. 从实验的结果可以发现: (1)随着新闻数目的增多, 传统数据库的查询耗时在不断的增大, 且幅度比较明显; (2)相对应的, 基于内存的流数据查询耗时明显要低于数据库的查询耗时, 这也是预料之中的, 因为传统数据库查询有磁盘 I/O 操作, 而这一阶段是比较耗时的; (3)流式数据的查询耗时也会随着新闻数目的增多而相应的增加, 但是我们可以实时的减少在内存当中的数据, 因为对于流式数据来说我们一般只会考虑最近一段时间的数据, 这也是流式数据的一个特点, 通过不断的减少在内存当中的历史数据也会提高查询性能.

5.2.2 聚集查询

聚集查询, 即函数查询, 在这里我们以 max 函数为例, 对数据库和流式数据均执行以下的查询语句: select max(newsComment) from news;即查询评论数最大的新闻, 两者的查询耗时对比如下图 8 所示.

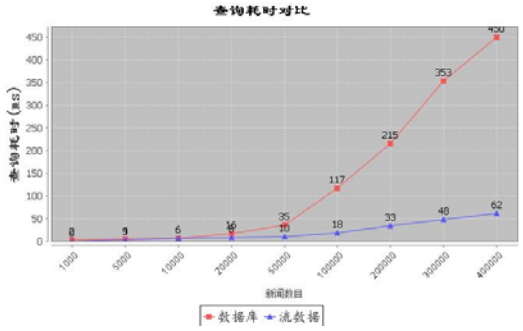

图 8　聚集查询耗时对比

聚集操作也可以称为块操作, 所谓块操作是指只有当全部的数据输入之后才可以进行的操作, 理论上流式数据的输入是无限的, 并不能完成传统意义上的块操作. 但是我们可以对存储在内存当中的那部分数据执行块操作, 而这也是合理的, 因为在大多数的流式数据应用场景当中, 用户只关心当前的数据特征, 过去的数据参考意义不大, 或者说可以通过其他方法对历史数据进行分析.

图 8 展示的是在聚集查询(MAX)下传统数据库与基于内存的流式数据查询的查询耗时对比, 横坐标表示新闻数目, 纵坐标表示查询耗时(以毫秒为单位), 通过上图的实验数据我们发现: (1)随着新闻数目的增多, 聚集查询数据库的查询耗时也以较大的幅度增长; (2)基于内存的流式数据查询相对于数据库查询耗时更低, 且随着新闻数目的增多, 延时增长的速度也更加的缓慢.

在对比基于数据库的查询和基于内存的查询执行效率时, 本文也比较了两者的查询结果. 在同样执行聚集操作(MAX)的情况下, 两者的输出结果如图 9 所示.

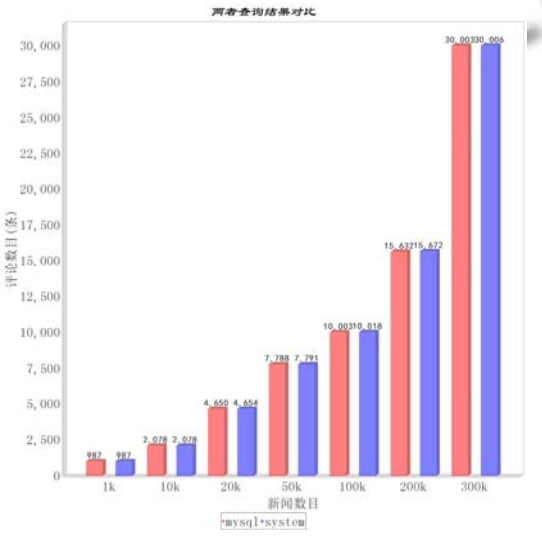

图 9　两者查询结果对比

横坐标表示新闻的数目, 以千为单位, 纵坐标表示的是最大的新闻评论数目. 红色柱状图表示 mysql 的查询结果, 蓝色柱状图代表本系统的查询结果.

从图 9 中的实验结果可以发现: (1)当新闻数目量较小的时候(少于 1 万条); 数据库的查询结果和本系统的查询结果是完全一致的; (2)当新闻数据大于 1 万条时, 基于数据库的查询结果和本系统的查询结果略有差异, 原因在于两者的数据抓取速率并不是完全一致的, 数据来源于网络, 所以数据的采集速率会受到网速的影响, 两个不同的数据抓取进程并不能保证完全的同步, 所以出现了这种较小的差异; (3)经过进一步验证, 两者的查询结果都是同一条新闻的评论结果: 习近平视察“互联网之光”博览会和大佬们说了啥. 链接为: http://news.163.com/15/1217/16/ BB24E0UO 00014 PRF. html, 即来自网易的一条热门新闻.

由此可以看到, 基于内存的数据查询不仅在性能上优于基于磁盘的数据库查询, 而且在查询结果的准确性上也得到了保证.

5.2.3 连续查询

为了验证系统连续查询[15]的能力, 设计窗口查询如下:

```
Select * From News Where (newsComment > 100) and (newsWeb='tencent')
For(t=0; t<=24;t+=1) {
    Window(News, 2015-12-06 00+t, 1)
}
```

该查询表示从 2015 年 12 月 6 号 0 点开始, 每小时查询一次评论数大于 500 的腾讯新闻, 总共执行 24 次. 这段时间的连续查询耗时如图 10 所示.

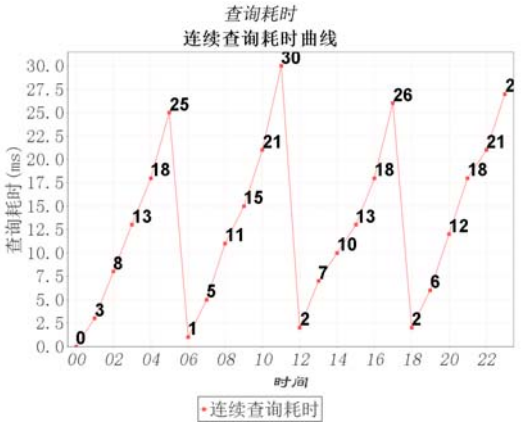

图 10　连续查询耗时

各个小时所包含的新闻条数如下图 11 所示.

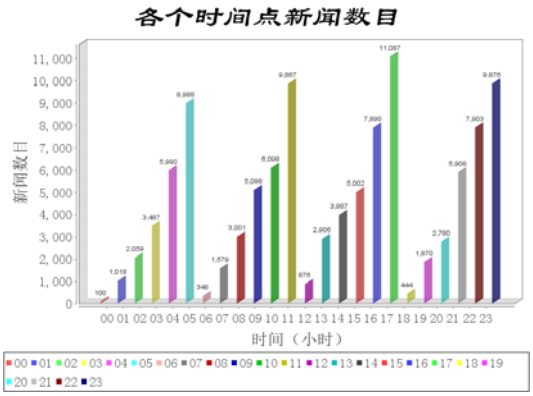

图 11 各时间点新闻条数

系统每六个小时会将内存中的历史数据存档, 所以图 11 中新闻条数会有一个循环的增减过程. 结合图 10 和图 11 的结果, 可以发现系统能很好的执行连续查询的功能, 查询耗时随着新闻数目的增多而延长, 但当内存中的数据量减少时, 耗时也相应的减少, 而这也是基于内存的流式数据查询的一个特点, 因为在大部分的情况下用户只关心当前的数据流情况, 历史数据的参考意义不大.

6 结语

流式数据有其区别于传统关系型数据的特点, 尤其体现在数据量大、变化频繁、要求快速的响应这几个特点上. 而传统的数据库比如 MYSQL、SQL server 等已经不能很好的完成流式数据的相关操作. 在大部分应用流式数据场景中, 需要快速的给出实时结果, 所以在内存中完成查询操作是一种很好的选择, 它省去了磁盘 I/O 的时间, 大大的提高了查询效率; 除此之外, 流式数据一般要求连续的查询, 而关系型数据库本身并不支持这种操作, 所以需要设计一种能很好的完成连续查询的系统, 以便能及时的观察到流式数据的变化更新情况.

本文基于流式数据特点的考量, 设计并实现一个流式数据查询系统, 并通过真实的流式数据验证了系统的可用性和高效性. 相对于基于数据库的查询, 该系统在查询效率上有很大的提高, 并且能够支持连续的查询, 除此之外, 还考虑了系统的负载, 监控等问题, 以此来保证系统的可靠性. 在将来的工作中, 我们将进一步深入优化系统的数据流接口、查询语言、查询过程等模块, 以此设计出一个更加通用、基础的流式数据查询系统.

参考文献

1 Datar M, Gionis A, Indyk P, et al. Maintaining stream statistics over sliding windows. Proc. of the 2002 Annual ACM- SIAM Symp. on Discrete Algorithms. 2002. 635–644.

2 Yi X, Bertino E, Vaidya J, et al. Private searching on streaming data based on keyword frequency. IEEE Trans. on Dependable and Secure Computing, 2014, 11(2): 155–167.

3 侯东风,刘青宝,张维明,邓苏.一种适应性的流式数据聚集计算方法.计算机科学,2010,37(3):152–155.

4 田杰.通用数据流管理原型系统 TTSTREAM 的设计与关键算法研究[学位论文].武汉:华中科技大学,2012.

5 黄浩,杨卫东.数据流上 Ad Hoc 查询的自适应处理算法.计算机工程,2013,(9):74–79.

6 Margara A, Urbani J, van Harmelen F, et al. Streaming the web: Reasoning over dynamic data. Web Semantics: Science, Services and Agents on the World Wide Web, 2014, 25: 24–44.

7 Welsh M. Sensor networks for the sciences. Communications of the ACM, 2010, 53(11): 36–39.

8 Abadi D, Carney D, Cetintemel U, et al. Aurora: A new model and architecture for data stream management.VLDB Journal, 2003, 12(2): 120–139.

9 Arasu A, Cherniack M, Galvez E,et al. Linear road a stream data management benchmark. Proc. of the 30th International Conference on Very Large Data Bases Conference. 2004. 480–491

10 Arasu A, Babcock B, Babu S, et al. STREAM: the Stanford stream data manager. IEEE Data Eng Bull, 2003, 26(1): 19–26.

11 王洪亚,曹姣,金杰.基于 Aurora 系统的持续型查询语言设计与实现.计算机工程与应用,2014(21):133–138.

12 罗刚,王振东.自己动手写网络爬虫.北京:清华大学出版社,2010.

13 htmlparser. http://htmlparser.sourceforge.net/.

14 De Rougemont M, Cao P T. Approximate answers to OLAP queries on streaming data warehouses. Proc. of the 15th International Workshop on Data Warehousing and OLAP. ACM. 2012.121–128

15 Assent I, Kranen P, Baldauf C, et al. Anyout: Anytime outlier detection on streaming data. Database Systems for Advanced Applications. Springer Berlin Heidelberg, 2012: 228–242.