分布式流数据加载和查询技术优化

易 佳[1] 薛 晨[2] 王树鹏[1]
(中国科学院信息工程研究所 北京 100093)[1] (国家计算机网络与信息安全管理中心 北京 100029)[2]

摘 要 分布式流查询是一种基于数据流的实时查询计算方法,近年来得到了广泛的关注和快速发展。综述了分布式流处理框架在实时关系型查询上取得的研究成果;对涉及分布式数据加载、分布式流计算框架、分布式流查询的产品进行了分析和比较;提出了基于Spark Streaming和Apache Kafka构建的分布式流查询模型,以并发加载多个文件源的形式,设计内存文件系统实现数据的快速加载,相较于基于Apache Flume的加载技术提速1倍以上。在Spark Streaming的基础上,实现了基于Spark SQL的分布式流查询接口,并提出了自行编码解析SQL语句的方法,实现了分布式查询。测试结果表明,在查询语句复杂的情况下,自行编码解析SQL的查询效率具有明显的优势。

关键词 大数据,流处理系统,分布式流查询,查询优化,Kafka快速加载
中图法分类号 TP274 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.05.031

Optimization on Distributed Stream Data Loading and Querying

YI Jia[1] XUE Chen[2] WANG Shu-peng[1]
(Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China)[1]
(National Computer Network Emergency Response Technical Team/Coordination Center of China, Beijing 100029, China)[2]

Abstract Distributed stream query is a kind of real-time query computation method based on data stream, which has been widely concerned and developed rapidly in recent years. This paper summarized the research results of the distributed stream processing framework in real-time relational query. There is an in-depth comparison of some products, including the distributed data loading framework, distributed stream computing framework and distributed stream query systems. The paper proposed a distributed stream query model based on Spark Streaming and Apache Kafka, and designed a fast data loading technology based on virtual memory file system, which gets the data loading speed one time faster compare to Apache Flume. On the basis of Spark Streaming, a distributed stream query interface based on Spark SQL was realized, and a method for parsing SQL queries was proposed to implement distributed query in data stream. The experiment results demonstrate that, in the case of complex SQL queries, the method of analyzing SQL by writing code by oneself has obvious advantages.

Keywords Big data, Stream processing system, Distributed stream query, Query optimization, Kafka fast loading

1 引言

随着移动互联网的迅猛发展和大数据时代的到来,大数据的敏捷开发已经成为一种常态[1]。随着数据规模的不断增加,数据变化速度越来越快,希望得到的处理和响应时间越来越短,因此对大数据的快速处理显得越来越重要。因此,基于分布式流数据的实时处理成为了当今热点研究领域之一。

大数据时代下,Hadoop的Mapreduce[2]分布式编程框架虽然在海量数据的计算、存储方面有着天然的优势,但实时性相对较差,时延较高,无法很好地满足快速实时处理的需求。

为了满足海量流数据的快速处理和查询的需求,越来越多的实时大数据流处理框架开始涌现。如Twitter的流处理框架Storm[3],它是一个分布式的、实时的在线处理框架,提供大规模的有效的实时计算,并且允许用户自己编写Bolt和Spout程序执行流计算,支持创建拓扑结构来转换没有终点的数据流,保证Storm的流式实时处理。Apache Samza也是一个分布式的流处理框架,它基于Apache Kafka[4]消息队列提供流消息数据,基于Hadoop Yarn[5]提供高容错、资源隔离和资源管理[6]。然而这些流处理系统的SQL查询都需要用户自行编码实现,系统本身无相应的查询语言可以提供。文献[7]提出目前的分布式流查询系统需要提高易用性和处理能力,增强应用程序的可用性和移植性,在处理系统之上实现具有抽象查询语言的关系查询系统;同时,分布式流查询系统一般通过消息队列进行数据采集和传输,为流处理提供数据保障。业界比较常见的分布式消息队列包括阿里巴巴的RocketMQ[8]、RabbitMQ[9]和Apache的Kafka。

尽管Apache Storm和Apache Samza等流处理框架都提供高效的流计算接口,但由于SQL语句解析和流查询的复杂

到稿日期:2016-08-02 返修日期:2016-12-11 本文受国家自然科学基金(61271275,61202067)资助。
易 佳(1992—),男,硕士,主要研究方向为大数据存储与智能化处理,E-mail:yijia2413@163.com;薛 晨(1986—),男,硕士,工程师,主要研究方向为网络信息安全;王树鹏(1980—),男,博士,高级工程师,主要研究方向为大数据存储与智能化处理。

性,目前还没有成熟的基于流数据的 SQL 查询接口。为提高分布式流处理系统的处理能力,增强可用性和移植性,本文的研究重点是基于分布式流处理系统,实现消息队列的快速加载技术优化和基于流数据的 SQL 查询技术优化。消息队列的快速加载能提高流计算过程中的数据生产效率,从根本上解决流计算过程中数据加载过慢的问题;基于流数据的 SQL 查询优化解决了原生流计算接口无法直接基于 SQL 语句查询的问题,并为 SQL 查询规划的优化提供了方向。

本文基于 Apache Kafka 和 Apache Spark 的 Spark Streaming[10] 组件,实现数据加载和流查询的优化技术。实验表明,优化后基于 Kafka 的加载速度相比常规数据加载策略提速 1 倍以上,且具有较高的可扩展性。同时,优化后的 SQL 查询速度相比于 Spark SQL[11] 的查询速度也有较大提升。

本文第 1 节介绍了大数据时代的背景和分布式流查询技术的现状,以及业界需求的实时处理系统模式;第 2 节介绍了构建分布式实时流查询系统的相关工作和相关组件;第 3 节介绍了分布式流查询系统的结构、组件和各个模块,以及基于分布式流查询系统各个模块的优化工作;第 4 节对流处理系统的测试结果和性能进行了分析;最后总结全文并展望后续工作。

2　相关工作

2.1　分布式流处理框架及其典型应用

近年来,分布式流处理框架的应用越来越广泛,如电商领域阿里巴巴双十一的实时订单服务、金融领域股票市场的高频交易实时分析、微博、Facebook 等社交网站的消息的实时推送服务等。而流数据的 SQL 查询服务也在此基础上得到快速发展。如 Spark SQL 的出现让 Spark Streaming 与 SQL 的结合成为了可能,在 Spark Streaming 的时间窗口内,可以接受 SQL 语句,进行实时的流查询;Intel 公司基于 Spark Streaming 和 Catalyst[11] 框架创建了 Streaming SQL[12] 项目,其目的是为了在数据流上支持 SQL 语句的查询。瑞士洛桑联邦理工学院数据实验室(EPFL Data Lab)基于 Apache Storm 创建了 Squall[13] 项目,在流处理过程中,提供符合 SQL 语法的在线查询操作;近期发展态势迅猛的 Apache Flink[14] 流处理框架也更加注重和完善其 Table API,使之更好地支持结构化数据的查询操作。本文所提的自行解析 SQL 的优化方案是基于 Spark Streaming 流处理框架的一种 SQL 查询优化方案,区别于 Intel 公司的 Streaming SQL 项目,该方案同样适用于其他分布式流计算框架。

典型的分布式流处理框架一般由以下几部分组成:1)数据接入层,负责处理外部数据的采集、接入,对应于数据流传输;2)消息缓存层,对应于消息队列,由于外部数据量大,因此需要中间件来对数据流进行缓存;3)流处理业务层,负责消费消息队列中的数据,处理并得到相应结果,流处理业务包括数据分析、任务调度、语法解析等细节任务;4)集群服务,用于保证整个集群的正确运行,提供流处理的保障。典型的分布式流处理框架如图 1 所示,包括数据接入层、消息缓存层、流业务处理层和集群服务等。

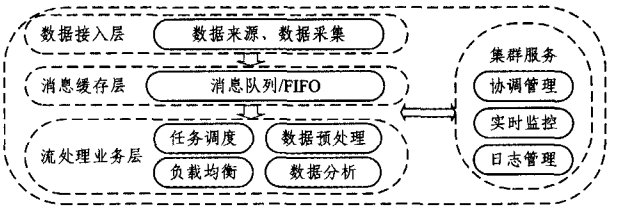

图 1　典型分布式流处理框架的组成

2.2　Spark Streaming

Spark Streaming 支持一个高层次的抽象,即离散流(discretized stream 或 DStream)[10],它是由一系列连续的弹性分布式数据集(Resilient Distributed Dataset,RDD)[15] 组成的。Spark Streaming 通过丰富的 API 和基于内存的高速计算引擎使用户可以结合流式处理、批处理和交互查询等应用。此外,Spark RDD 的数据重用机制具有更高效的容错机制[15]。在计算流程上,Spark Streaming 将流式计算分解成一系列小的批处理作业。所以在 Spark Streaming 程序中,可以基于 Spark SQL 和时间窗口执行 SQL 查询,简化流处理业务的编码。若基于 Spark SQL 的查询无法满足业务需求,则可基于时间窗口和滑动窗口自行设计查询处理的业务逻辑。

2.3　Kafka

由图 1 得知,消息队列对流处理的数据缓存有重要作用。消息采集器将数据收集并放入消息队列中缓存。Spark Streaming 对消息队列 Kafka 提供了原生的支持,在生产环境中,越来越多的公司将 Spark Streaming 和 Kafka 进行整合,构建了分布式流处理系统。

Kafka 是由 LinkedIn 开发的一个分布式的消息系统,它能够发布和订阅数据流,以分布式的形式存储数据流,并且提供实时流计算的接口。它以时间复杂度 O(1) 的方式提供消息的持久化能力,即使是对于 TB 级以上的数据,也能保证常数时间复杂度的访问性能;此外,Kafka 的高吞吐率能使之在廉价的机器上快速传输。生产者向 Topic 写入数据,消费者从 Topic 读取数据[4]。同时,因为 Kafka 是分布式消息队列,其 Topic 也可以分布在多个节点上。

2.4　数据加载工具

数据加载是指将数据从数据源加载到消息队列的过程,对于 Kafka 来说是生产者将数据载入消息队列。

Cloudera 公司的 Flume[16] 是一款高性能的分布式日志收集系统,其 Kafka Sink[17] 支持直接将数据源发到 Kafka 消息队列中。Kafkacat[18] 是一款非 JVM 的 Kafka 生产和消费工具,主要基于 C 语言实现,它可以直接从标准输入流获取消息,然后以管道的形式将消息发送到 Kafka 的 Broker;同时,Kafkacat 也支持指定发送数据到 Broker 节点以及节点的指定分区中,让数据分发更具针对性。

3　系统结构

3.1　关键技术

本文基于 Kafka 和 Spark Streaming 构建了一个分布式流查询系统,主要实现了分布式数据加载和流查询的优化,用到的关键技术如下:

1)设计一种基于 Kafkacat 的数据加载模式,提供增量易扩展的数据加载接口。

2)提出一种基于内存文件系统的数据加载策略,解决数据载入过程中磁盘 I/O 带来的瓶颈和性能问题。

3)提出自行编码解析 SQL 语句的策略,在保证 Spark SQL 同等功能的同时,提升基于原生 SQL 解析的性能,提高分布式数据流查询的速率。

此外,文中提出的基于内存文件系统的数据加载策略不仅适用于 Kafka,也同样适用于其他基于磁盘缓存的分布式消息队列,如 RocketMQ 等;自行编码解析 SQL 查询的方式,同样适用于 Storm、Samza 等流计算引擎。本文以 Spark Streaming 和 Kafka 为例,进行实验和论证。

3.2 分布式流查询系统简介

基于 Kafka 和 Spark Streaming 的分布式流查询系统框架如图 2 所示。

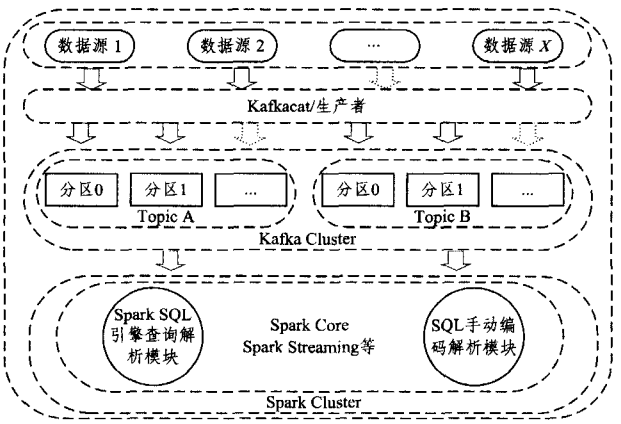

图 2 基于 Kafka 和 Spark Streaming 的分布式流查询框架

数据来源于不同的数据源,这些数据源可以是增量的日志文件,也可以是离线文件,不同的数据节点分布在不同的数据源文件。Kafka 集群对应的生产者为 Kafkacat,它将对应的数据源以消息的形式发送到对应 Topic 的指定分区中。Kafka 提供 Spark Streaming 的流处理消费接口,Spark Streaming 的消费逻辑负责处理和解析数据,得到最终的流查询结果。

图 2 中,Spark Cluster 部分代表的数据消费模块包含两种不同的消费模式:1)利用 Spark 引擎的 Spark SQL 组件,对流消息直接进行查询操作,其优化在于 Spark SQL 引擎的 Catalyst 框架,对查询处理的不同阶段执行逻辑优化、物理优化和代码生成等优化技术[11];2)基于 SQL 查询的实际含义,采用手动编码解析消费的模式,对接收到的数据进行查询操作。两个消费模块可同时并行消费 Kafka 中的数据。

3.3 数据加载

3.3.1 数据采集

将数据源加入到消息队列中需要一个采集的过程。从不同的数据源中采集消息并加入消息队列的方式多种多样,如使用 Apache Flume 开发的 Kafka Sink 插件,配置好 Flume 客户端,通过 HTTP 的方式直接将消息发送到 Kafka 对应的 To-pic中;也可以使用 Kafka 的 Producer API[19],直接接收或读取日志文件,将数据源写入 Kafka 中。经过测试发现,同等情况下,相同配置的 3 台机器,将等量的离线日志信息发送到 Kafka 消息队列中,基于 Kafkacat 的生产者要比基于 Flume 或 Kafka API 的生产者快一倍以上,对于 1 台拥有 3 块 500GB 外置 SAS 硬盘、64GB 内存的机器,创建一个拥有 3 个分区的 Topic,其对应的不同生产者类型的生产者速率如图 3 所示。

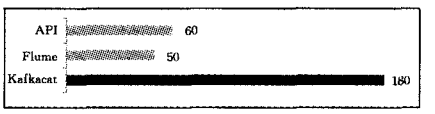

图 3 Kafka 生产工具单机(3 块磁盘)性能对比

对比可知,单机情况下,基于 Kafkacat 的生产者直接以管道的形式从数据源接收消息,然后将接收到的消息通过网络传输发送到 Kafka 对应的 Topic 中,其速度可达 180 MB/s,比 Flume 和 Kafka Producer API 的生产效率快一倍以上。

3.3.2 采集策略

不同的数据采集策略对应不同的消息生产速度。本文主要分析 Kafkacat 的消息采集机制。它具备良好的横向扩展性,同时 2.4 节介绍了 Kafkacat 可以指定 Topic 的分区发送消息。对于拥有 3 个分区的 Kafka Topic,可定制如下数据发送策略:

分区 0:cat file1 | kafkacat -b broker -t topic -p 0

分区 1:cat file2 | kafkacat -b broker -t topic -p 1

分区 2:cat file3 | kafkacat -b broker -t topic -p 2

这样 3 个进程对应同步发送到 3 个不同的分区,速度几乎可以提升 3 倍。而 Kafka API 的程序和 Flume 虽然也可以指定分区发送,但效率没有 C 语言代码高,且代码的可移植性太弱。因此,在创建 Kafka Topic 的分区时,开发者只需获取 Topic 对应的分区数量,然后基于 Kafkacat 以指定分区的方式同步采集数据,即可提高消息采集效率。

对于消息队列而言,重要的是能将消息快速缓存并以先进先出的策略被消费。生产环境中,常常会遇到消息队列出现各种性能瓶颈。如磁盘的优势在于外部的持久化存储,但普通磁盘的 I/O 性能相对较低,而将昂贵的 SSD 只用于缓存消息不是一个明智的行为。常规数据生产情形下,对于 Kafka 的 Topic 来说,其分区是基于磁盘的,最理想的情况是,分区能均匀地分布于各个磁盘上,这样能尽量保证磁盘 I/O 并行性能的最大化,但依旧会存在磁盘数量不足、单盘读写速率过慢的问题。

为了降低因磁盘数量不足、磁盘 I/O 速率过低带来的影响,本文提出了基于内存文件系统[20]的 Kafka 分区设置机制。内存文件系统可以限制申请内存的大小,在内存充足的情况下,一台机器可以申请若干个内存文件系统,并挂载在对应的挂载点。显然,基于磁盘的 I/O 性能会远低于同等条件下基于内存文件系统的 I/O 性能。对应的内存文件系统的创建命令为 mount -F tmpfs -o size=2g tmpfs /mnt/sdb。其中,/mnt/sdb是需要提前创建的文件夹,也是内存文件系统的挂载点,创建 Topic 时,开发者可通过配置文件,指定 Kafka 的分区均匀地分布到这些内存文件系统中;2g 代表内存文件系统可用空间的大小。

考虑到生产机器的内存大小是有限的,可以结合 Kafka 的消息清除策略,设置将 Broker 中的消息在达到固定大小后从前往后删除。只需保证内存文件系统中的消息能在规定时间内被消费完,该内存文件系统就可被循环利用,加速整体流查询速度。

3.4 数据消费

3.4.1 Streaming SQL 消费

生产者将数据源采集到 Kafka 消息队列之后,消费者(流处理业务模块)开始消费数据。截止目前,Spark 官方尚不支持 Spark Streaming SQL,所以本文采用的是基于 Spark Streming 的时间窗口和滑动窗口设置来支持分布式 SQL 查询,同时结合 Spark SQL 和 Spark 的耦合性,让分布式流查询像查询关系型数据库一样查询流消息,最终可将查询结果写入数据仓库或 HDFS 等外部存储系统中。

首先基于时间窗口和滑动窗口创建 Dstream,设置时间窗口和滑动窗口的大小;然后对每一个 RDD 进行 SQL 查询的字段划分;接着通过 Spark SQL 接口进行 SQL 查询,得到分布式查询结果。

Spark SQL 会将一个 SQL 查询翻译成分布式可执行的 Spark 程序。如语句:select src_ip, sum(num) from table group by src_ip,Spark SQL 引擎会把 SQL 查询翻译成执行步骤,提交给 Spark 执行。第一步,读取表,提取 src_ip 和 num 字段并计算 num 字段的和;第二步,设定 shuffle 的 key 是src_ip,它会从第一组节点分组后分发给聚合节点,使得相同的 src_ip 汇集到同一个节点,然后这些节点把每个组的和值加在一起,得到最后的结果。

SQL 语句在 Catalyst 框架内的执行流程为:1)SQL 解析:做简单的词法语法解析,生成对应的逻辑执行计划。2)逻辑执行计划分析:把做完词法语法解析的执行计划进行初步的分析和映射。3)逻辑执行计划优化:基于定义的规则对执行计划进行优化操作。4)物理执行计划树:根据优化后的逻辑执行计划树生成物理执行计划树。5)执行前规划:再次基于规则对物理执行计划树进行优化。6)触发整棵树的计算,得到最终查询结果。

3.4.2 SQL 自行解析消费

由于基于滑动窗口和时间窗口的 SQL 操作受限于 SQLContext 的内部实现等,其不支持一些复杂的嵌套查询操作,因此本文还提出了另一种基于流数据的 SQL 查询方案,即手动编码解析 SQL 语句。理论上,手动编码解析 SQL 语句能支持任何复杂的分布式流查询语句。

考虑以下 SQL 查询:

```
select concat(col1,col2) from table where col3=1024;
```

这是一个简单的查询过滤操作,扫描一个表,当 col3 的值为 1024 时,连接 col1 和 col2 的值并输出。在 Spark SQL 中,可以直接将整条语句作为参数引入执行。而在 Spark Streaming 中,则可以直接用 if 条件来过滤 col3,然后进行字符串连接操作,最后输出过滤拼接后的字符串。

为了计算该查询,基于 Spark SQL 的查询会利用迭代模型的经典评估策略(Volcano model)[21]进行计算。在 Volcano 迭代模型中,一个查询由多个算子组成,每个算子都有 next() 接口,该接口每次只返回一个元组给嵌套树中的下一个算子。基于 Volcano 迭代模型的计算在处理元组时会加入虚函数的调用,大量的虚函数调用会消耗较多 CPU 指令,导致计算减慢。同时,算子之间的传递需要将元组放入内存。而在手动解析的 SQL 语句中,没有一个函数调用能够加速查询的过程。此外,编译器会将手写代码的临时数据存放到 CPU 寄存器中,访问寄存器所需要的 CPU 时间会远少于访问内存所消耗的 CPU 时间。所以,手动编码解析 SQL 语句比执行 Spark SQL 的查询优化的执行效率高。

接下是对实验过程中的结果进行分析和对比,主要对比不同的数据加载工具(Kafkacat、Flume)对加载速率的影响、不同的数据加载载体(内存文件系统、磁盘)对加载速率的影响,以及不同的 SQL 解析执行策略(Spark SQL 解析、自行编码解析)对流查询速度的影响。

4 统计结果及分析

4.1 数据加载

本文所测试的数据加载和分布式流查询对应的软件和硬件环境如表 1 所列。其中 Spark 采用的是最新的 1.6.1 版本。

表 1　系统及软硬件环境

参数项	值
操作系统类型	Redhat Enterprise 6.5 x64
内存	128 GB
CPU	32 核心
硬盘	10 块 2T SAS 盘
Kafka 版本	2.10_0.8.11
Spark 版本	1.6.1
Zookeeper 版本	3.4.8
Flume 版本	1.6.0
网卡	10000Mb/s

为了对比不同加载工具在不同的机器上的加载性能,分别在 1,2,3,5 台机器上测试了 Flume 和 Kafkacat 的加载结果,对比结果如表 2 所列。

表 2　Kafkacat 与 Flume 的加载性能对比

Kafka 节点	Flume 加载/MB/s	Kafkacat 加载/MB/s
1	378	847
2	751	1607
3	1119	2458
5	1764	3922

由表 2 可知,同等条件下,Kafkacat 的加载效率几乎是 Flume 加载效率的 2 倍。同时,随着 Kafka 节点的增多,Kafkacat 和 Flume 的加载速度均呈近似线性增长。

图 4 将 Kafkacat 的加载性能与 Flume 的加载性能进行了直观对比,单台 Kafkacat 的加载速率比 Flume 高 1 倍以上,单位为 MB/s。

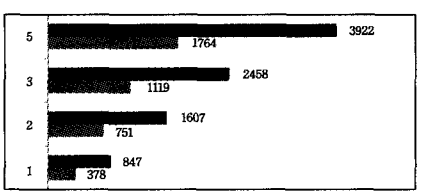

图 4　Kafkacat 与 Flume 的加载性能对比图

此外,本文将同等条件下 Kafakcat 基于内存文件系统的加载策略和基于磁盘的加载策略进行了对比测试,测试环境与表 1 提供的环境一致,同样对比了 1,2,3,5 台机器的测试数据,其结果如表 3 所列。

表 3　Kafkacat 基于内存文件系统和磁盘的加载性能对比

Kafka 节点	磁盘加载/MB/s	tmpfs 加载/MB/s
1	847	901
2	1607	2001
3	2458	2706
5	3922	4611

实验表明，在基于磁盘的加载和基于内存文件系统的加载策略下，随着 Kafka 节点的增多，加载效率近乎呈线性增长。

同等条件下，基于内存文件系统的消息的生产能力与基于 10 块磁盘的 Kafka 分区的消息生产能力相近，甚至于前者的效率更高，图 5 展示了这一对比。

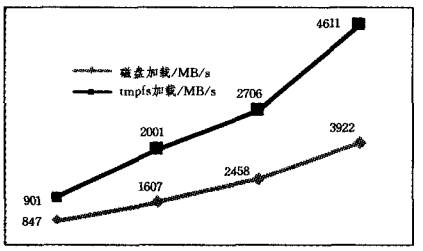

图 5 Kafkacat 基于内存文件系统和磁盘的加载性能对比

由图 5 可知，基于内存文件系统的加载速率略高于基于磁盘的加载效率。结合表 2 中数据加载的实验结果，可以得到如下结论：

1)相比于 Flume 和自行利用 Kafka Producer API 设计的生产者，Kakfacat 更容易配置和横向扩展。

2)同等物理条件下，Kafkacat 的生产速度几乎是 Flume 的 2 倍甚至更多。

3)同等条件下，一台机器上挂载足够数量的内存文件系统对于消息生产的速度等基本等同于挂载 10 块 SAS 外接硬盘的消息生产速度。基于内存文件系统的 Kafka 消息加载策略在节省磁盘的情况下，还能提高加载效率。

4)生产环境中，磁盘加载策略更适用于较长时间的消息缓存，能用来做持久化；内存文件系统加载策略更适用于快速的消费和删除，能弥补磁盘 I/O 上的不足。

所以，在机器没有足够的外接磁盘而流查询系统所需的吞吐率极高的情况下，基于内存文件系统的 Kafka 分区策略和基于 Kafkacat 的生产者是一个非常理想的组合。

4.2 Streaming SQL 与自行解析消费比较

分布式流查询部分，实验选取了 5 台机器进行测试，其配置与 Kafka 加载性能测试的机器一致。实验选取了两条 SQL 查询语句作为分布式查询的依据，语句如下：

语句 A：select log_id,src_ip,count(*) as cnt from syslog where log_id in (923687,34209,959351) group by log_id,src_ip order by cnt desc limit 100；

语句 B：select concat(src_ip,dst_ip) from syslog where src_ip=923687。

对语句 A 和语句 B 分别进行 Streaming SQL 查询测试和 SQL 自行解析查询测试，测试数据均为 10 亿条上网日志记录，日志信息以逗号间隔，共 19 个字段，每条约 176 字节，多次测试之间互不干扰且为同一份实验数据。语句 A 和语句 B 的查询结果如表 4 所列。

表 4 不同语句在不同流查询策略下的速度对比

SQL 语句	查询速度/MB/s
语句 A(自行解析)	4640
语句 A(Spark SQL 解析)	1000
语句 B(自行解析)	4721
语句 B(Spark SQL 解析)	3480

对于语句 A 类型的分组聚合排序的 SQL 查询操作，自行解析 SQL 语句的流查询速度几乎为 Spark SQL 解析操作的 5 倍。而对于语句 B 类型的连接过滤操作，自行解析的速度约为 Spark SQL 解析操作的 1.4 倍。不同类型的两个语句 A 和语句 B 自行解析得到的处理速度几乎一致。

图 6 示出了不同 SQL 查询语句在相同条件下的查询结果。整体上 Spark SQL 查询解析的速率低于自行解析 SQL 查询语句的效率。

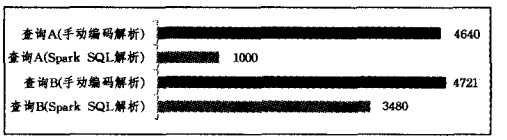

图 6 不同语句在不同流查询策略下的速度对比

综合表 4、图 6 和实验过程，得出如下结论：

1)相比于自行解析 SQL 查询，Spark SQL 提供了更友好的接口，编码更加简单，容错、语法解析均由 Spark 框架来保证，不易出错，且拥有较高的流查询效率。

2)对于复杂的嵌套 SQL 语句查询，分布式 Spark SQL 流查询可能会出现执行过慢、语法支持不够等问题，这时可以通过自行解析 SQL 语句的执行，来得到更高的分布式流查询效率。

3)自行解析 SQL 语句的优势在于，可以充分利用已知的信息，消除虚函数的调用并通过底层硬件进行优化。

4)理论上，任何复杂的 SQL 语句都可以自行解析，提交给 Spark Streaming 执行，得到与 Spark SQL 同等的执行结果，同时可能得到更高的执行效率。

结束语 本文基于分布式流计算处理技术，构建了以 Spark Streaming 和 Kafka 为基础的分布式流查询系统。在此基础上，1)对比了基于 Kafkacat 和 Flume 的加载技术，同等物理条件下，Kafkacat 的加载速率比 Flume 高 1 倍以上，且具有良好的横向扩展性；2)提出了基于内存文件系统的数据加载技术，解决了磁盘 I/O 和磁盘数量带来的数据加载性能瓶颈问题，此技术适用于任何基于磁盘缓存的分布式消息队列；3)提出了自行编码解析 SQL 语句的策略。在分布式流查询过程中，既可以采用接口丰富的 Spark SQL 直接对 SQL 语句进行查询解析，也可以自行编码解析得到更高的查询速率。对于分组聚合等复杂的查询语句，自行解析 SQL 语句能提高 4 倍左右的流查询速度；同时，自行解析 SQL 语句的方法同样适用于其他分布式流计算引擎，具有较高的可移植性。

下一步的研究将侧重于以下方面：1)增加对更多类型的 SQL 语句的自行解析和 Spark SQL 解析对比；2)进一步探讨本文的实验结果，以及与 Spark 2.0 版本的新特性进行对比；3)比较不同的 Spark 版本之间自行解析 SQL 语句和 Spark SQL 解析 SQL 语句的分布式流查询速度。

参 考 文 献

[1] CHEN H M,RICK K,SERGE H. Agile Big Data Analytics Development:An Architecture-Centric Approach[C]// 2016 49th Hawaii International Conference on System Sciences (HICSS). IEEE,2016.

[2] DEAN,JEFFREY,SANJAY G. MapReduce:simplified data processing on large clusters[J]. Communications of the ACM,2008,51(1):107-113.

[3] TOSHNIWAL A,TANEJA S,SHUKLA A,et al. Storm®twit-

ter[C]// ACM SIGMOD International Conference on Management of Data. ACM,2014:147-156.
[4] KREPS,JAY,NEHA N,et al. Kafka:A distributed messaging system for log processing[C]//Proceedings of the NetDB. 2011.
[5] VAVILAPALLI,VINOD K,et al. Apache hadoop yarn:Yet another resource negotiator[C]// Proceedings of the 4th Annual Symposium on Cloud Computing. ACM,2013.
[6] http://samza. apache. org.
[7] WANG C K,MENG X F. Relational Query Techniques for Distributed Data Stream:A Survey [J]. Chinese Journal of Computers,2016,39(1):80-96. (in Chinese)
王春凯,孟小峰. 分布式数据流关系查询技术研究[J]. 计算机学报,2016,39(1):80-96.
[8] RocketMQ[OL]. https://github. com/alibaba/rocketmq.
[9] RabbitMQ[OL]. https://www. rabbitmq. com.
[10] ZAHARIA,MATEI,et al. Discretized streams:Fault-tolerant streaming computation at scale[C]//Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. ACM,2013.
[11] ARMBRUST,MICHAEL,et al. Spark sql:Relational data processing in spark[C]//Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. ACM,2015.
[12] StreamingSQL[OL]. https://github. com/Intel-bigdata/spark-streamingsql.
[13] Squall[OL]. https://github. com/epfldata/squall.
[14] Flink[OL]. http://flink. apache. org.
[15] ZAHARIA,MATEI,et al. Resilient distributed datasets:A fault-tolerant abstraction for in-memory cluster computing[C]// Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. USENIX Association,2012.
[16] HOFFMAN S. Apache Flume:Distributed Log Collection for Hadoop[M]. Packt Publishing Ltd,2013.
[17] Kafka_flume[OL]. http://www. cloudera. com/documentation/kafka/lat-est/topics/kafka_flume. html.
[18] Kafkacat[OL]. https://github. com/edenhill/kafkacat.
[19] KafkaProducer[OL]. https://kafka. apache. org/090/javadoc/indexhtml?org/apache/kafka/clients/producer/KafkaProducer. html.
[20] SNYDER,PETER. tmpfs:A virtual memory file system[C]// Proceedings of the Autumn 1990 EUUG Conference. 1990.
[21] GRAEFE G,MCKENNA W J. The Volcano optimizer generator:Extensibility and efficient search[C]//International Conference on Data Engineering. IEEE Xplore,1993:209-218.

(上接第 171 页)

[2] LAIGLE-CHAPUY Y. Permutation polynomials and applications to coding theory [J]. Finite Fields Appl. ,2007:13(1):58-70.
[3] LIDL R,NIEDERREITER H. Introduction to finite fields and their applications[M]. Cambridge University Press,1986.
[4] MULLEN G L. Permutation polynomials over finite fields [C]// Proc. Conf. Finite Fields and Their Applications in Lect. Notes Pure Appl. Math. ,Marcel Dekker,New York,1993:131-151.
[5] LIDL R,MULLEN G L. When does a polynomial over a finite field permute the elements of the field [J]. American Math Monthly,1988,95(3):243-246.
[6] LIDL R,MULLEN G L. When does a polynomial over a finite field permute the elements of the field? II [J]. Amer Math Monthly,1993,100:71-74.
[7] COULTER R,HENDERSON M,MATTHEWS R. A note on constructing permutation polynomials [J]. Finite Fields Appl, 2009,15:553-557.
[8] MARCOS J E. Specific permutation polynomials over finite fields [J]. Finite Fields and Their Applications 2008,17(2):105-112.
[9] ZIEVE M E. Classes of permutation polynomials based on cyclotomy and an additive analogue[M]// Additive Number Theory. Springer-Verlag,2010:366-361.
[10] HELLESETH T,ZINOVIEV V. New Klooserman sums identities over F_{2^m} for all m [J]. Finite Fields Their Applications, 2003,9(2):187-193.
[11] YUAN J,DING C,WANG H,et al. Permutation polynomials of the form $(x^p-x+\delta)^s+L(x)$ [J]. Finite Fields Appl,2008,14: 482-493.
[12] ZHENG X,ZHU X,HU L. Two new permutation polynomials with the form $(x^{2^k}+x+\delta)^s+x$ over F_{2^n}[J]. Applicable Algebra in Engineering, Communication and Computing, 2010, 21 (2):145-150.
[13] CAO X,HU L,ZHA Z. Constructing permutation polynomials from piecewise permutations[J]. Finite Fields & Their Appllications,2014,26(3):162-174.
[14] ZHENG Y,YU Y,ZHANG Y,et al. Peicewise constructions if inverses of cyclotomic mapping permutation polynomials [J]. Finite Fields & Their Appllications,2016,40(c):1-9.
[15] HOU X. Permutation polynomials over finite fields—A survey of recent advances [J]. Finite Fields & Their Appllications, 2015,32:82-119.
[16] ZHENG Y,YUAN P,PEI D. Peicewise constructions of inverses of some permutation polynomials [J]. Finite Fields & Their Appllications,2015,36:151-169.
[17] YUAN P,DING C. Further results on permutation polynomials over finite fields [J]. Finite Fields & Their Appllications,2014, 27:88-103.
[18] YUAN P,ZHENG Y. Permutation polynomials from piecewise functions [J]. Finite Fields & Their Appllications,2015,35(c): 215-230.
[19] ZHENG Y,YUAN P,PEI D. Large classes of permutation polynomials over F_{q^2} [M]. Springer Science+ Business Media New York,2016.
[20] TU Z,ZENG X,LI C,et al. Permutation polynomials of the form $(x^{p^m}-x+\delta)^s+L(x)$ over the finite field $F_{p^{2m}}$ of odd characteristic [J]. Finite Fields Appl,2015,34(c):20-35.
[21] TU Z,ZENG X,JIANG Y. Two classes of permutation polynomials having the form $(x^{2^m}+x+\delta)^s+x$[J]. Finite Fields Appl, 2015,31:12-24.
[22] ZENG X,TIAN S,TU Z. Permutation polynomials from trace functions over finite fields [J]. Finite Fields Appl,2015,35:36-51.
[23] ZHA Z,HU L. Two classes of permutation polynomials over finite fields [J]. Finite Fields Appl,2012,18(4):781-790.