计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2018,27(3):44−50 [doi: 10.15888/j.cnki.csa.006244]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 Storm 的车联网数据实时分析系统①

张春风[1,2], 申　飞[1], 张　俊[1], 陈　杰[1,2], 刘　静[1,2]

[1](中国科学院 强磁场科学中心, 合肥 230031)
[2](中国科学技术大学, 合肥 230026)

摘　要: 针对传统车联网平台在处理海量数据时存在吞吐量小, 实时性差的问题, 设计了一种基于大数据流处理技术的实时分析系统. 系统分为数据采集、 数据转发、实时分析、数据存储和可视化展示 5 层. 为了满足系统高并发接入以及实时性的需求, 引入 Storm 实时计算系统进行数据的实时分析. 同时, 利用 Kafka 消息队列的异步通信机制将各层之间解耦, 采用 Hbase 进行海量数据存储, 从而提高车联网非机构化数据存储效率. 另外, 针对访问数据库开销大的问题, 采用 Redis 缓存策略, 进一步提高查询效率. 实验证明, 较传统的多线程处理平台, 该系统具有低延迟, 高吞吐, 可拓展等特点, 能够满足车联网大数据流处理要求.

关键词: 车联网; Storm; 实时分析; 低延迟; 高吞吐

引用格式: 张春风,申飞,张俊,陈杰,刘静.基于 Storm 的车联网数据实时分析系统.计算机系统应用,2018,27(3):44–50. http://www.c-s-a.org.cn/1003-3254/6244.html

Real-Time Analysis System of Vehicle Network Data Based on Storm

ZHANG Chun-Feng[1,2], SHEN Fei[1], ZHANG Jun[1], CHEN Jie[1,2], LIU Jing[1,2]

[1](High Magnetic Field Laboratory, Chinese Academy of Sciences, Hefei 230031, China)
[2](University of Science and Technology of China, Hefei 230026, China)

Abstract: To address the large data processing problem on the vehicle networking platform in which the data throughput is small, and its poor real-time feature, this paper proposes a new real-time analysis system based on the big data stream processing technology. The proposed system consists of 5 layers including data acquisition, data forwarding, real-time data analysis, data cache and storage, and visual display. Specifically, it introduces Storm real-time computing system to real-time data processing, which is beneficial to the high concurrent access and can meet real-time requirements of the system. Furthermore, aiming at the problem that the access to the database is expensive, Redis cache strategy is used to improve the query efficiency. Experiments show that the system has low latency, high throughput, and scalability compared with the conventional multithreaded processing platform, which is able to satisfy the requirements of vehicle network data stream processing.

Key words: vehicle network; Storm; real-time analysis; low latency; high throughput

车联网云数据中心与综合服务平台汇聚了车辆位置、状态、速度、加速度、路网等非结构化的车联网数据, 其规模已经达到 TP 甚至 BP 级别. 传统的数据分析技术已经无法满足该级别数据处理的需求, 因此, 引进分布式计算技术和数据存储技术, 构建流式计算处理框架, 对车辆进行实时监控和调度管理迫在眉睫. 目前, 不少流式大数据处理的方案被提出. 其中, Spark Streaming 是 Spark 核心 API 的一个扩展, 不同于 Storm 一次一个地处理数据流, Spark Streaming 在处理前按时间间隔预先将数据流切分为一段一段的批处理作业.

① 基金项目: 国家自然科学基金 (61273323)
收稿时间: 2017-06-19; 修改时间: 2017-06-30; 采用时间: 2017-07-08; csa 在线出版时间: 2018-01-25

因此, Spark Streaming 不是真正意义上的流式计算, 而是批处理, 相比于 Storm, Spark Streaming 存在延迟高, 吞吐量较小等缺点. 另外, Samza 是由 LinkedIn 开源的一个分布式流处理系统, 它依赖于 Hadoop 的资源调度和 Apache Kafka[1,2]. Samza 的流单位既不是元组, 也不是 Dstream, 而是一条条消息, 在数据传递过程中, 消息可能会多次发送, 造成数据冗余. 针对车联网数据处理分析的问题, 以及其低延迟, 增量计算的需求, 本文设计了一种基于 Storm 技术的流式计算系统, 系统具有低延迟, 高吞吐, 分层且可扩展的特性. 利用 Kafka 消息队列将各层之间解耦, Storm 进行数据实时分析, Hbase 和 Redis 对分析结果存储, 从而实现对车辆状态进行实时监控.

1 实时流相关技术

车联网实时分析系统主要由 Boost.Asio、Kafka、Storm、Redis、Hbase 组成. 其中, Boost.Asio 负责与车载终端建立连接, 采集数据. Kafka 负责连接采集层和 Storm. Redis 和 Hbase 负责分析结果的存储. 系统的整个核心实时分析模块由 Storm 担当, 对采集来的数据分析过滤, 实时处理. 下文将介绍 Storm 流式计算框架.

1.1 Storm 流式计算框架

Storm 是一个分布式的、可靠的、容错的数据流处理系统[3]. 同 Hadoop 一样, Storm 可以处理大批量的数据, 并且 Storm 在保证高可靠性的前提下可以让处理进行的更加实时; Storm 同样还具备容错和分布计算的特性, 即可以扩展到不同的机器上进行大批量的数据处理. 除此之外, Storm 同时还有以下的这些特性:

(1) 简单的编程模型. 类似于 MapReduce 降低了并行批处理复杂性, 降低了进行实时处理的复杂性.

(2) 容错性. Storm 会管理工作进程和节点的故障.

(3) 水平扩展. 计算是在多个线程、进程和服务器之间并行进行的. Storm 使用 Zookeeper 进行集群协调, 这样可以充分的保证大型集群的良好运行[3–5].

(4) 可靠的消息处理. Storm 保证每个消息至少能得到一次完整处理. 任务失败时, 它会负责从消息源重试消息.

(5) 快速. 系统的设计保证了消息能得到快速的处理, 使用 ZeroMQ 作为其底层消息队列[6–10].

为了更好的体现 Storm 在流式计算方面独特的优越性, 对比 Spark 计算框架. 如表 1 所示二者的主要区别表现在, Storm 是纯实时的, 来一条数据, 处理一条数据, 后者是准实时, 对一个时间段内的数据收集起来, 作为一个 RDD, 再处理. 而且, Storm 的事务机制、健壮性/容错性、动态调整并行度等方面的表现, 都要比 Spark Streaming 更加优秀.

表 1 Spark 和 Storm 流数据计算框架比较

	Spark	Storm
实时计算1	准实时	纯实时
数据传输	推送	拉取
系统结构	主从	主从
资源利用率	高	高
时延大小	秒级	毫秒级
吞吐量	比较高	高
容错性	一般	强

2 系统架构设计

系统架构如图 1 所示, 主要包含数据采集、数据转发、实时分析、数据存储、可视化展示.

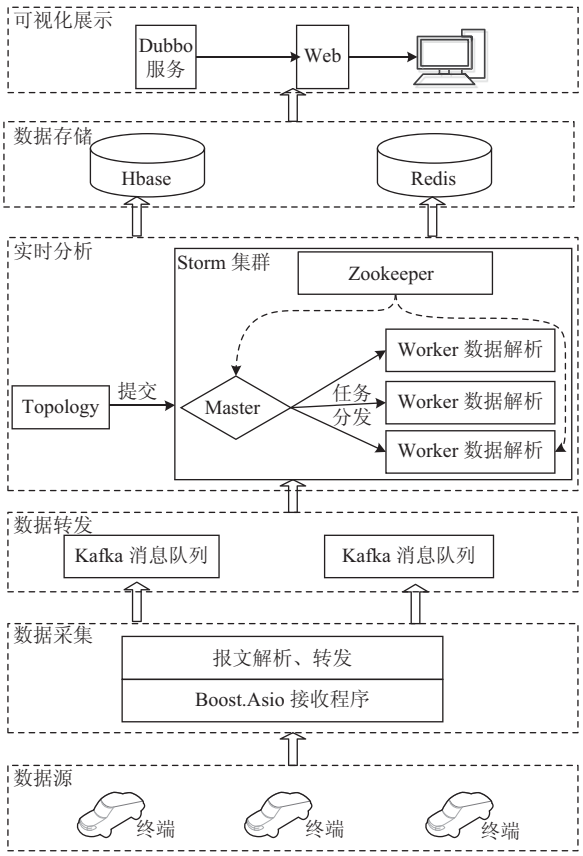

图 1 系统架构图

系统采用层次化结构的设计原理, 每个部分的主要功能如下:

(1) 数据采集: 负责与智能终端 (OBD) 建立 TCP 连接, 验证校验, 获取报文数据.

(2) 数据转发: 对数据类型进行划分, 放在 Kafka 消息队列中, 实现数据的分类管理和高并发接入.

(3) 实时分析: 创建 KafkaSpout, 从 Kafka 中获取数据, 并以数据流的形式发送给 bolt, bolt 负责转化这些数据流, 在 bolt 中完成过滤, 分析计算.

(4) 数据存储: 将实时分析结果存储至 Redis 和 Hbase, 利用分布式文件系统的优势可以实现高并发的读写速度.

(5) 可视化展示: 使用 Dubbo 分布式服务提供实时定位, 轨迹查询和速度报警等服务, 同时利用百度地图动态显示.

2.1 数据采集

数据采集主要负责接收车载终端发送过来的实时车辆信息数据, 车载终端通过无线网络与数据采集层建立通信连接. 数据采集层会维护一个连接请求队列, 面对高并发连接的需求, 模块在开发过程中使用 Boost.Asio 基础网络库作为通信基础, 使用 Boost.Asio 库的异步接口函数来实现全异步的事件处理, 包括 TCP 链接监听、TCP 数据发送、TCP 数据接收. 数据采集层提供车载终端统一的信号接收服务, 避免了数据的重复, 缺失, 从而保证数据采集的质量和可靠性, .

2.2 数据转发

数据转发作为平台各层之间的通信层, 将系统各层之间进行有效地解耦, 提高平台的健壮性. 目前用于消息传递的方案主要包括 RabbitMQ 和 Kafka. 其中 RabbitMQ 是流行的开源消息队列系统, 开发语言为 erlang. Kafka 则是一个分布式的高吞吐量的消息系统[11–13]. 与 kafka 相比, RabbitMQ 协议复杂, 参数较多, 因此其仅适用于数据量较小的场景. 而 Kafka 具有透明、易扩展和吞吐量较高的优点, 更适合处理海量的车联网数据. 基于此, 本系统采用 Kafka 消息队列实现数据缓存与转发, 利用其能够提供消息持久化能力和具有容错性保障的优势, 达到系统数据缓冲的目的.

作为基于 log File 的消息系统, Kafka 更加可靠, 减少了数据丢失的现象. 另外, Kafka 可以记录数据的消费位置, 同时可以自定义消息消费的起始位置, 从而保证了重复消费消息的实现. 而且, Kafka 同时具有队列和发布订阅两种消息消费模式, 与 Storm(实时分析层) 的契合度很高, 充分利用 Linux 系统的 I/O 提高读写速度等. 转发层的架构如图 2 所示.

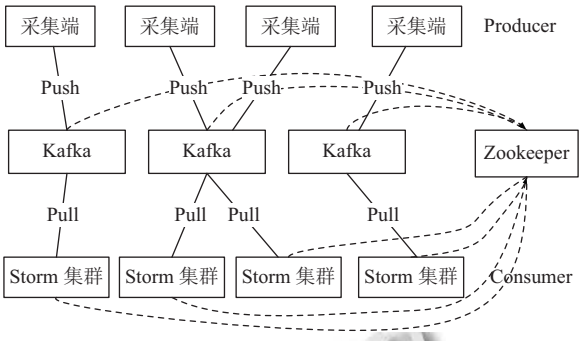

图 2 缓存转发架构

采集层作为 Producer(生产者), 将采集到的车载终端数据以终端标识为区分标准, 建立多个 topic, 用来管理不同种类的消息, 不同类别的消息会记录在到其对应的 topic 池中, 而这些进入到 topic 中的消息会被 Kafka 写入磁盘的 log 文件中进行持久化处理. 实时分析层作为 Consumer(消费者), Storm 集群从 Kakfa 中获取实时流进行处理分析. 数据处理分析的速度可以慢于数据采集的速度, Storm 集群有空余时再拉取那些没拉取到的数据, 从而保证数据不丢失.

2.3 实时分析层

数据实时分析层是系统的核心层. 车载终端所采集的数据是没有被解析的原始数据, 使用单字节、双字节或四字节来进行物理量的表示. 所采集到的数据格式为:

7E 02 00 00 22 01 41 50 48 11 75 00 09 00 00 00 00 00 00 00 03 01 EF 98 78 06 FB B1 08 00 24 00 8C 00 90 16 12 20 16 47 25 01 04 00 01 45 8A FF 7E

因此, 实时分析层需将采集到的车辆实时信息进行过滤、解析、坐标转换. 解析海量数据存在延迟阻塞、高并发等问题. 为了解决这些问题, 本文抛弃了 Java 线程池、无限队列等传统的方法, 突破集中式单节点运算的限制, 采用分布式, 高容错的实时计算系统 Storm. 实时分析拓扑如图 3 所示.

首先, 建立实时分析拓扑图 (Topology) 并提交给 Storm 集群, 由集群中主节点 Master 的守护进程 “Nimbus”分发代码, 将任务分配给工作节点 (Worker) 执行, 同时监控任务和工作节点的运行情况等; Worker 节点上运行的守护进程“Supervisor”, 负责接收 Nimbus 分发的任务并运行, 每一个 Worker 上都会运行着 Topology 程序的一部分. 因此, Topology 程序的运行就是由集群上多个 Worker 一起协同工作的.

Topology 的部分代码如下所示:

```
//建立拓扑图 Topolopy
TopologyBuilder builder = new TopologyBuilder();
//kafkaspout 从 kafka 中获取数据流
builder.setSpout("kafkaspout",
new MyKafkaSpout(kafkaConfig));
//数据解析与转换 bolt
builder.setBolt("msgpredealbolt",
new MsgPreDealBolt()).setNumTasks(1).
shuffleGrouping("kafkaspout");
//电子围栏检测 bolt
builder.setBolt("notasktravelcarbolt",
newNoTaskTravelCar()).setNumTasks(1).
shuffleGrouping("msgpredealbolt");
//超速检测 bolt
builder.setBolt("overspeedbolt",
newOverSpeedBolt()).setNumTasks(1).
shuffleGrouping("msgpredealbolt");
//数据存储入库 HBasebolt
builder.setBolt("HbaseBolt", HbaseBolt()).
setNumTasks(1)
.fieldsGrouping("msgpredealbolt");
//数据存储入库 RedisBolt
builder.setBolt("RedisBolt", RedisBolt()).
setNumTasks(1)
.fieldsGrouping("msgpredealbolt");
```

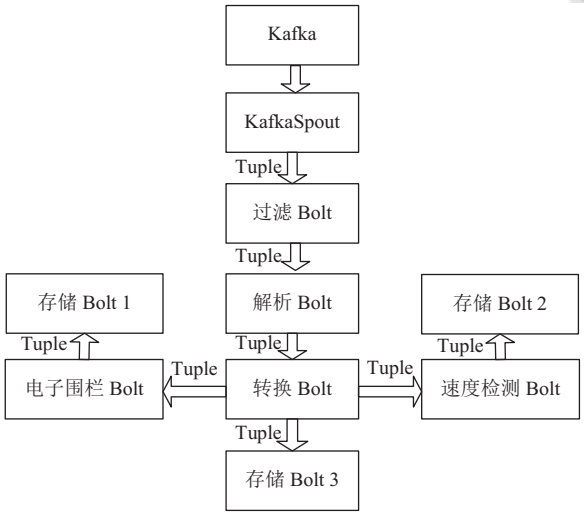

图 3 实时分析拓扑

拓扑中包含 Spout 和 Bolt 两种角色, 系统中 KafkaSpout 从 Kafka 消息队列中获取数据, 通过 nextTuple() 方法以数据流 Tuple 元组的形式发送给下游的 MsgPreDealBolt, Spout 的 ack 和 fail 方法分别在元组被成功处理和处理失败时调用, 保证数据处理的完整性. MsgPreDealBolt 完成过滤工作, 根据指令校验码进行筛选出符合要求的轨迹数据, 按照 FieldsGrouping 的分组策略, 通过 execute() 方法发送到解析模块. 解析模块 GpsDealbolt 主要完成分析处理逻辑, 包括 2 进制的转化, 终端识别, 分析终端发送数据的类型, 并做出相应的处理, 最后按照 FieldsGrouping 的分组策略发送至下一处理模块. 转换模块主要是北斗坐标转换为百度坐标的处理 (方便使用百度地图功能), 从而以次完成数据的解析处理, 转换等工作. Storm 通过实现不同的 Bolt 来完成计算结果的多样化存储. 本系统中对分析结果处理有 HbaseBolt 和 RedisBolt. HbaseBolt 将结果保存到 HDFS 分布式文件系统中, RedisBolt 将计算结果保存到缓存中, 便于查询检索.

2.4 数据存储

基于传统关系型数据库存储的车辆信息表日渐增大, 接近单表存储的上限, 且数据的查询和写入性能会呈现指数级别地下降. 为了实现高性能的并发读写操作, 数据存储层采用硬盘存储和内存存储两种模式. 硬盘存储使用分布式的、面向列的开源数据库 Hbase, 存储离线数据以及将处理后的流数据进行落地. 目前主流的内存数据库有 Memcached 和 Redis. Memcached 是一个高性能的, 具有分布式内存对象的缓存系统; Redis 是一个基于内存的高性能 Key/Value 数据库.

二者主要的区别为: Redis 会周期性地把更新的数据写入磁盘或者把修改操作写入追加的记录文件, 并且在此基础上实现了 master-slave(主从) 同步. 与 Memcached 相比, Redis 的优势在于其具有高效的读写效率以及丰富的数据类型所带来的快速开发. 另外, Redis 作为缓存具有更高的安全性. 因此, 本文选用 Redis 数据库作为系统的缓存, 用于保存整个系统的分析结果, 实现缓存数据的持久化. 在节点宕机或者断电的情况下, 系统仍能够从硬盘中获取备份的数据, 从而保证了系统的健壮性. 以下分别介绍两种模式的具体实现:

(1) 将 Storm 中分析的实时数据存储到 Hbase 中, 为后期的查询和离线分析做数据支持. 采用 HBase 大数据存储框架, 在保证足够的存储空间的前提下, 利用

HBase 的分布式特点来提高数据的存取速度, 解决数据的单点存储隐患, 保障数据的高可用性. 系统中实时车辆信息表如表 2 所示.

表 2 实时车辆信息表

Row Key	Time Stamp	GpsInfo					
		CarId	Lng	Lat	Date	V	...

Hbase 以表 Table 形式存储数据, 每行包括一个 RowKey 和多个 Column Family, 且每行通过 RowKey 唯一标记, 行按照 RowKey 的字典序排列. 实时车辆信息表根据 Hbase 表的设计要求, RowKey 为车牌号的反转+"_"+时间戳, 要尽量缩小 RowKey 的长度, 提高检索效率, columnfamily 要尽量的少, 原因是过多的 columnfamily 之间会互相影响, 所以设计了一个列簇 CarInfo, 在 CarInfo 下分为很多列, 如车牌号, 定位时间, 经纬度等车辆轨迹信息.

(2) 将 Storm 中分析的最新实时数据缓存到内存数据库 Redis 中, 利用 Redis 高性能操作和运算上的优势, 为数据展示层提供既方便又快捷的数据检索. 由于 Redis 数据库容量受到物理内存的限制, 不能用作海量数据的高性能读写, 只能将最新的数据缓存到内存中, 数据展示层首先查询 Redis, 如果数据存在, 直接从 Redis 获取数据, 否则从 Hbase 中获取, 如此以来提高数据的查询检索速度, 优化系统性能.

2.5 可视化展示

可视化展示为数据存储层中所有车辆实时信息提供统一的查询入口, 将车辆轨迹分析结果以可视化的形式展现. 本系统使用 Dubbo 分布式服务框架, 将核心业务抽取出来, 作为独立的服务, 使前端应用能更快速和稳定的响应, 解决了服务器单点故障, 方便后期的拓展和维护. 可视化展示主要包括以下几个方面:

(1) 前端借助百度地图, 将查询车辆的轨迹信息, 包括已经行驶的时间, 行驶过程中的停留点, 速度等在整个地图上动态的显示.

(2) 根据车辆行使的路段, 利用百度地图查询该地段的限速大小, 并与车辆当前速度进行比较, 检测是否超速, 如果超速, 给予驾驶员警告.

(3) 电子围栏, 将车辆的位置信息与规定行使区域实时进行比较, 检测是否超出预定的行驶路线.

(4) 根据车牌号实时定位当前车辆的位置信息, 行驶速度, 急加速等信息, 有效地监控当前车辆状态.

3 实验分析

实验主要验证系统的功能和 Storm 实时分析的效率. 通过部署局域网的 10 台 PC 机, 搭建集群进行测试. 实验环境配置如下: Storm 版本: 0.10.2, 系统版本: centos6.7, JDK1.8.0_45-b14, Kafka2.11-0.10.0.1, Zookerper3.4.9, Hbase1.0.3, , Redis-3.2.3, Dubbo2.8.4, 单机计算机节点配置: 内存大小: 8 G, CPU 型号: intel Core i5, 磁盘 500 G. 本次实验系统部署架构, 集群各个节点的配置和功能描述如表 3 所示. 数据源是网约车平台共享的数据, 将数据源以日志的形式存储在本地硬盘中, 通过读取文件来模拟车载终端发送的大量数据流.

表 3 集群节点配置表

IP	软件部署	功能描述
192.168.56.101	Boost.Asio	采集程序
192.168.56.102	Kafka, Zookeeper	转发层
192.168.56.103	Kafka, Zookeeper	转发层
192.168.56.104	Kafka, Zookeeper	转发层
192.168.56.105	Storm	Nimbus
192.168.56.106	Storm	Supervisor
192.168.56.107	Storm	Supervisor
192.168.56.108	Redis, hbase	存储层
192.168.56.109	Redis, hbase	存储层
192.168.56.110	Dubbo	webApp

3.1 功能测试

首先进行功能的测试, 测试系统能否从终端获取数据, 并利用 Storm 实时解析并可视化的展视. 在 web 端根据车牌号对车辆进行定位查询, 后台从缓存或数据库中获取当前车辆最新的数据, 利用百度地图实时定位, 然后进行可视化展现. 图 4 为实时定位效果的实例展示.

图 4 实时定位图

个人轨迹的查询, 根据车牌号可以查询某一时间段的车辆行驶轨迹. 后台根据时间段和车牌号, 从缓存或数据库拿到车辆轨迹信息, 并在地图上绘制出来, 轨迹查询的效果图如 5 所示.

图 5 轨迹查询图

从效果图可以看出实时分析层已经实现了将采集层采集的数据, 过滤, 解析, 经纬度转换, 存储到数据库中, 并通过展示层可视化的展现出来. 从而说明基于 Storm 的车联网数据实时分析系统的功能基本实现, 对数据的采集, 转发, 解析, 落地存储功能均无问题.

3.2 性能测试

测试系统实时处理的性能, 主要指标是数据处理的吞吐量, 数据处理延迟. 吞吐量反映系统单位时间内处理数据的规模. 对比分析 Storm 集群和 Java 线程池的吞吐能力. 不断增加任务执行的数据量, 记录处理完成所需的时间, 为了提高实验结果的准确性, 测试的数据保持一致, 每项结果是经过 5 次测试取平均值, 对比结果如图 6.

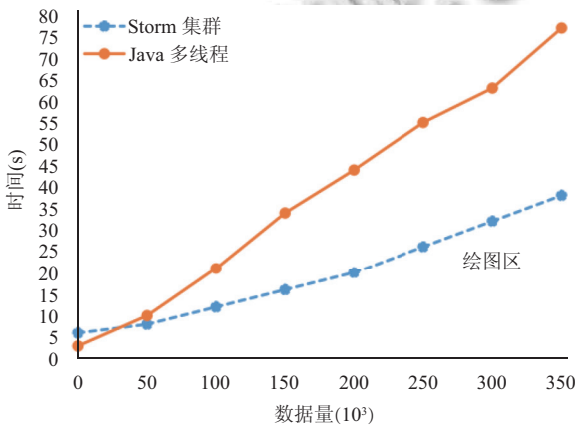

图 6 Java 线程池与 Storm 运行时间对比

当数据规模较小时, 利用 Storm 集群计算需要更长的时间, 这是因为在集群中任务的分发, 数据的传输都要经过网络, 需要消耗部分系统资源和时间. 随着数据规模的增大, 集群的处理能力明显提升, 这是因为 Storm 中计算任务被划分为不同的组件, 在多个 Worker 节点上的 Executor 执行. 因此, 随着数据规模进一步扩大, 单机版的 Java 多线程处理的耗时将更加难以接受, 甚至出现卡顿死机的情况, 而 Storm 集群支持水平扩展, 添加了 Worker 节点, 能够满足更大规模的数据处理要求. 因此, Storm 在流式计算方面的性能远远超过传统的 Java 多线程平台.

Storm 集群和传统 Java 多线程平台在延迟性方面没有可比性. 数据处理延迟与数据处理模块的并行任务数有关. 一般来说, 并行任务数越多, Tuple 等待被处理的时间就越短, 处理延迟越小.

通过实验对比分析, 将 KafkaSpout 的组件数目设置为 2, 分析处理模块的 Bolt 分别设置为 2, 8. 测试结果如图 7 所示, 随着处理的数据量越大, 当处理模块 Bolt 数目为 2 时, 处理延迟越来越大, 这是因为 Spout 不断产生新的数据, 分析处理模块不能及时处理, 导致数据积累, 处理延迟呈上升趋势. 当处理模块的 Bolt 数目为 8 时, 处理延迟都在毫秒级. 因此合理的设置各组件的任务数是优化 Storm 性能的有效途径, 提高 Storm 并行处理能力.

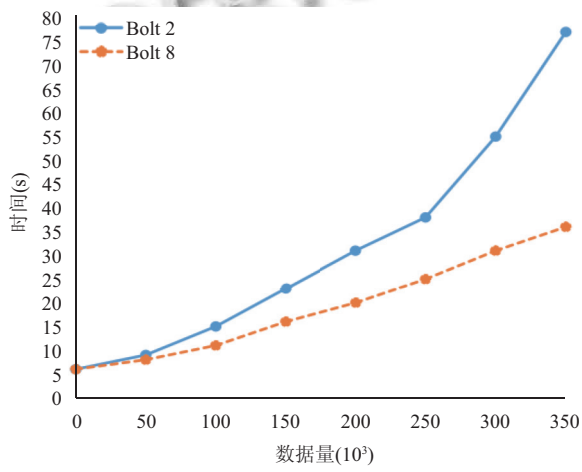

图 7 并行数 Bolt 为 2, 8 处理时间对比

本系统还具备快速部署, 易拓展的优点. 随着业务的发展, 数据量和计算量越来越大, 仅需要增加 Worker 节点便可提高任务的计算能力. 具体地, 新增节

点首先解压 Zookeeper 和 Storm 安装包, 修改配置文件, 然后运行 Zookeeper 和 Storm 集群. 无需修改程序, 在集群启动后, 重新提交 topology 即可完成部署. 随着部署节点的数量不断增加, 系统易拓展的优势将更加明显.

4 结论与展望

为了对海量车联网数据进行实时分析及可视化展示, 本文设计了基于 Storm 的车联网数据实时分析系统, 系统融合了 Kafka 消息队列、Storm 流式计算框架、Hbase 分布式数据库、Redis 内存持久化数据库、Dubbo 分布式框架等技术. 通过测试验证, 与传统的多线程处理平台相比, 系统有高吞吐和低延迟的特性,

实现车辆状态实时监控, 从而提高车辆监管效率. 系统的分布式负载均衡, 调度优化等问题将是我们下一步重点关注的问题.

参考文献

1 周国亮, 朱永利, 王桂兰, 等. 实时大数据处理技术在状态监测领域中的应用. 电工技术学报, 2014, 29(S1): 432–437.

2 戴菲. 基于 Storm 的实时计算系统的研究与实现[硕士学位论文]. 西安: 西安电子科技大学, 2014.

3 李劲松. 一种基于 Storm 的分布式实时增量计算框架的研究与实现[硕士学位论文]. 成都: 电子科技大学, 2015.

4 孙朝华. 基于 Storm 的数据分析系统设计与实现[硕士学位论文]. 北京: 北京邮电大学, 2014.

5 王铭坤, 袁少光, 朱永利, 等. 基于 Storm 的海量数据实时聚类. 计算机应用, 2014, 34(11): 3078–3081.

6 李庆华, 陈球霞, 蒋盛益. 基于数据流的实时处理框架模型. 计算机工程, 2005, 31(16): 59–60, 63. [doi: 10.3969/j.issn.1000-3428.2005.16.023]

7 屈国庆. 基于 Storm 的实时日志分析系统的设计与实现[硕士学位论文]. 南京: 南京大学, 2016.

8 杨素素. 基于 Storm 的城市消防联网远程监控系统的实时数据处理应用. 计算机测量与控制, 2017, 25(3): 55–59.

9 杨婷婷. 基于出租车 GPS 轨迹数据的实时交通状态获取和现有实时路况系统评估[硕士学位论文]. 上海: 华东师范大学, 2016.

10 McCreadie R, Macdonald C, Ounis I, *et al*. Scalable distributed event detection for twitter. 2013 IEEE International Conference on Big Data. Silicon Valley, CA, USA. 2013. 543–549.

11 Namiot D. On big data stream processing. International Journal of Open Information Technologies, 2015, 3(8): 48–51.

12 Maarala AI, Rautiainen M, Salmi M, *et al*. Low latency analytics for streaming traffic data with Apache Spark. 2015 IEEE International Conference on Big Data (Big Data). Santa Clara, CA, USA. 2015. 2855–2858.

13 Nair LR, Shetty SD, Shetty SD. Applying spark based machine learning model on streaming big data for health status prediction. Computers & Electrical Engineering, 2018, 65(1): 393–399. [doi: 10.1016/j.compeleceng.2017.03.009]