计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(12):51−57 [doi: 10.15888/j.cnki.csa.006112]

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

基于 Spark 的实时视频分析系统①

郑　健[1], 冯　瑞[2,3]

[1](复旦大学 计算机科学技术学院, 上海 201203)
[2](复旦大学 上海市智能信息处理重点实验室, 上海 201203)
[3](复旦大学 上海视频技术与系统工程研究中心, 上海 201203)

摘　要: 视频监控技术在交通管理、公共安全、智慧城市等方面有着广泛的应用前景, 且向着智能识别、实时处理、大数据分析的方向发展. 本文针对大规模实时视频监控提出了新的解决方案. 基于 Spark streaming 流式计算、分布式存储及 OLAP 框架, 使多路视频处理在可扩展性、容错性及数据多维聚合分析上具有明显的优势. 系统根据视频处理算法划分为单机处理与分布式处理. 并将视频图像处理与数据分析耦合, 利用 Kafka 消息队列与 Spark streaming 完成对多路视频输出数据的进一步操作. 结合分布式存储方案, 并利用 OLAP 框架实现对海量数据实时多维聚合分析与高效实时查询.

关键词: Spark; 视频分析; 数据分析; 实时计算

引用格式: 郑健,冯瑞.基于 Spark 的实时视频分析系统.计算机系统应用,2017,26(12):51–57. http://www.c-s-a.org.cn/1003-3254/6112.html

Scalable Real-Time Video Analysis System Based on Spark

ZHENG Jian[1], FENG Rui[2,3]

[1](School of Computer Science, Fudan University, Shanghai 201203, China)
[2](Shanghai Key Laboratory of Intelligent Information Processing, Fudan University, Shanghai 201203, China)
[3](Shanghai Engineering Research Center for Video Technology and System, Fudan University, Shanghai 201203, China)

Abstract: The video surveillance technology has a wide application prospect in traffic management, public safety, intelligent city, and is developing towards intelligent recognition, real-time processing, and large data analysis. In this paper, we propose a new system for large-scale real-time video surveillance. The system is based on Spark streaming, distributed storage and OLAP framework so that multi-channel video processing has obvious advantages in scalability, fault tolerance and data analysis of the multi-dimensional polymer. According to video processing algorithm, the processing module is divided into single machine processing and distributed processing. The video processing is separated from the data analysis, and the further operation of the multi-channel video output data is completed by using Kafka message queue and Spark streaming. Combining the distributed storage technology with OLAP framework, the system achieves real-time multi-dimensional data analysis and high-performance real-time query.

Key words: Spark; video analysis; data analysis; realtime computation

当前社会对视频监控的需求日益强烈, 在公共安全、交通管理、智慧城市等领域有着广泛的应用. 相对于传统的单纯视频监控, 结合图像算法与机器学习算法的智能视频监控有着显著的优势[1]. 当前的视频监控系统, 在视频流传输、解码、图像处理与识别等方面有着大量的研究, 但在分布式处理与多路视频数据关联分析等方面的研究较少.

在实际生活中, 视频监控一般应用在较为重要的

① 基金项目: 国家科技支撑计划 (2013BAH09F01);临港地区智能制造产业专项 (ZN2016020103)
收稿时间: 2017-03-20; 采用时间: 2017-04-10

场合, 这对视频监控系统在高可用、低延迟上提出了更高的要求. 由于视频图像这种非结构化数据处理较为复杂, 导致图像处理算法的处理时间相对较长. 如果要实现视频分析的实时性, 需要在图像分析算法或者处理架构上进行改进. 现有的视频监控以单路视频流为单位进行分析, 每路视频流处理后的数据之间完全没有关联. 而在一些场景, 如智能交通领域, 往往同一片区域的视频数据之间能够挖掘出更多的信息[2], 可针对交通数据进行交互式分析、实时挖掘、实时研判, 为千万上亿级别的过车记录实现关联汇总以及实时预警. 如整个交通网络的流量数据分析、交通拥塞时合理路线的推送、套牌车辆识别、黑名单车辆邻近监控点预警等. 而且每路视频处理后的历史数据往往包含着重要信息, 能够从中分析出有价值的信息以进行统计与决策, 如过去 1 年的时间里车辆的行驶轨迹分析等. 在海量数据面前, 传统的关系型数据库已无法实时进行多维数据分析, 这需要采用专门的 OLAP 解决方案[3].

目前, 有些针对视频的分布式解决方案, 如 Natarajan 等人[2]提出的基于 MapReduce 的交通视频分析系统, 但其将视频流先存储至 HDFS 后再利用 MapReduce 进行分布式处理, 时延较大. 而且 Hadoop 适合批处理, 而不适合实时流计算. Huang 等人[4]虽提出利用 Spark 进行图像处理的方法, 但在视频流动态接入、系统高可用与多路视频关联分析等方面没有提出解决方案.

本文基于 Spark[5]、Kafka 消息中间件[6]、分布式存储[7,8]及 OLAP 框架[9], 使多路视频处理在可扩展性、容错性及数据多维聚合分析上具有明显的优势. Spark streaming 相对于其他流式计算框架的突出特点是能够高效并行的实现容错恢复及其对数据分析、机器学习、图计算等工具的无缝集成.

1 系统概述

本系统分为采集层、视频处理层、消息分析层、存储与查询层等. 系统总体框图如图 1 所示.

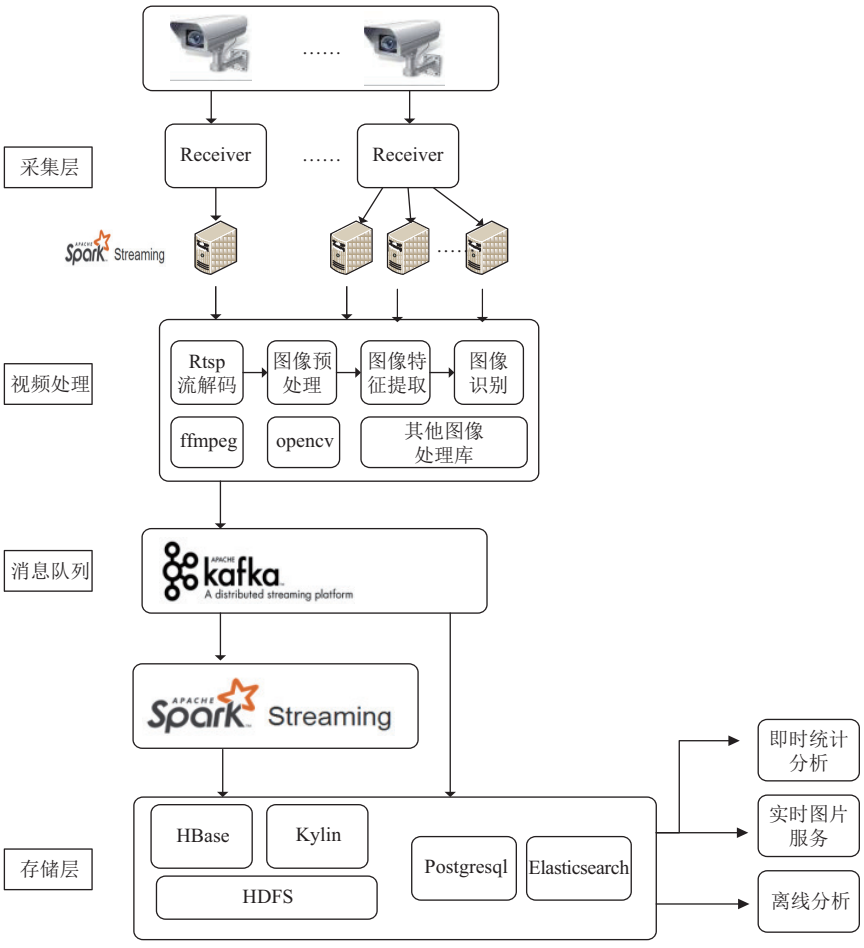


图 1 系统总体框图

采集层: 用于接收来自各个摄像头传入的 RTSP 视频流数据. 利用 YARN 与 Spark streaming, 根据集群资源负载情况, 动态的在相关节点中启动接收器, 接收视频流数据并且解码.

视频处理层: 用于处理解码后的视频帧序列. 将视频分析算法分为两种: 帧间相关与帧间无关. 并对两种情况进行分别处理. 然后调用 opencv 及其他图像处理库进行图像预处理、特征提取与图像识别等操作, 并将处理结果输出至 Kafka 消息队列以进行进一步分析.

消息处理层: 采用 Kafka 与 Spark streaming 进行消息流实时分析. 根据具体场景, 可对单路视频消息独自分析, 或对多路视频消息进行关联分析. 该层针对消息数据的精确一次消费语义进行了实现. 消息处理结果输出至存储层, 以对数据进行实时查询或离线分析.

存储与查询层: 以 HDFS、HBase 及 Postgresql 作为结构化数据及非结构化数据的存储组件, 利用 Elasticsearch 与 Kylin 完成对数据的全文检索及多维聚合分析. 该层能够向外提供实时查询、统计分析、图片检索等服务.

完成视频处理算法的各个子模块后, 在 Spark Job 中利用 PipedRDD 进行操作. pipe 为 RDD 及 Dstream 中的算子, 用于调用外部程序对分区集合中的数据进行处理. 系统调用流程如下:

```
sc.fromCameraStream("rtsp://10.27.31.5/road?fps=15")
.pipe("feature_extraction")
.pipe("object_detection")
```

2 各模块的算法设计与实现

2.1 采集层

2.1.1 视频流动态接入

这里将从摄像头传入的视频流信息集中于分布式平台上处理, 由于各个机器资源配置不同, 导致处理能力各异. 在实际应用中, 可能随时增加或减少摄像头, 所以处理平台需要实现摄像头的动态接入. 本系统基于 YARN 资源管理器, 这里 ReceiverTracker 将根据集群资源状况进行动态调度. 在新增摄像头时, 将 Receiver 接收器分配至空闲的集群节点上, 或在减少摄像头时, 将资源压力较大的节点上的 Receiver 接收器重新分配至其他空闲的集群节点上. 在 YARN 中, 每个 NodeManager 每隔一段时间会通过心跳机制向 Resource Manager 发送心跳信息, 其中包括该节点上的资源使用状况. 通过调用相关接口, 能够获取集群中各个节点的资源使用信息. Spark 在启动 Receiver 时, 先将其封装成一个 RDD, 作为一个 job 进行任务提交, 等待 Resource Manager 为其分配合适的 container, 然后在相应 Node Manager 节点上启动. 本系统对原生的 ReceiverTracker 组件进行了二次开发, 在封装 Receiver 成 RDD 时, 根据 YARN 接口返回的数据修改其 preferredLocation 变量以选择分配节点. 视频流接入流程如图 2 所示.

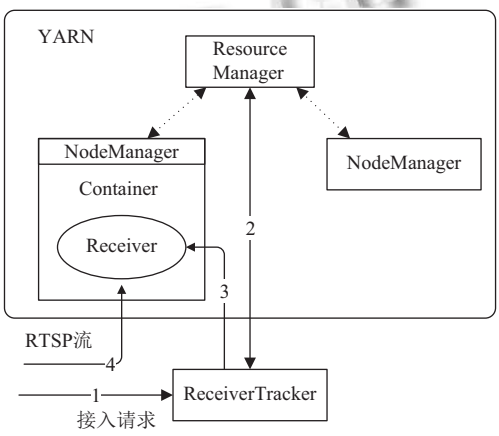
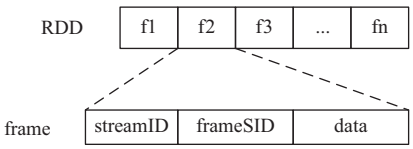


图 2 视频流接入流程

2.1.2 视频流解码

对于从摄像头流入 RTSP 视频流数据, 需要先进行解码, 然后才能进行下一步处理. 这里通过调用 opencv 库中的 VideoCapture 方法进行视频流解码 (需要 ffmpeg 与 gstreamer 库的支持). 在 Receiver 接收器的 onStart 方法中, 调用 VideoCapture 方法将不断流入的 rtsp 视频流数据解码并生成一个个 Mat 数据结构, 这里用 frame 表示, 该结构是 opencv 中进行图像处理操作的基本对象. Spark streaming 的执行流程如图 3 所示. Spark streaming 中的 Receiver 接收器负责接收数据, 然后解码生成图像帧序列并缓存至 currentBuffer 中. Timer 定时器每隔设定的间间隔回调 BlockGenerator, 将 currentBuffer 中缓存的数据流封装成一系列 Block, 放入 blocksForPushing 中, 作为 Dstream 中分区的数据记录. blockPushThread 周期性的从 blocksForPushing 取出 Block 存入存储体系. 由 JobGenerator 为每一批 Block 生成 Job, 交由 Spark 引擎处理. 本系统中, Dstream 中 RDD 的分区数据由图像帧序列组成, 每帧图像 frame 的数据格式如下所示.



其中,

streamID: 视频流编号, 作为每个视频流的唯一标识.

frameSID: 帧序列编号, 用于标记视频流中各个帧序列的顺序关系.

data: 视频帧数据, 包含图像数据的字节数组, 由 Mat 转化而来.

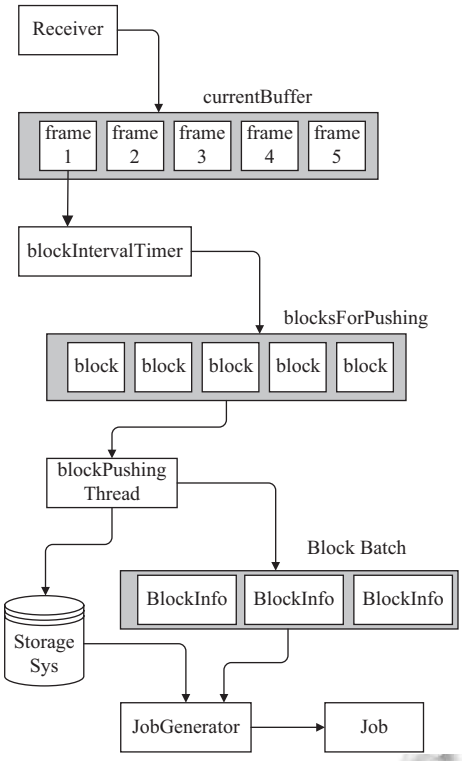


图 3 Spark streaming 执行流程

2.2 视频流处理

解码后的视频数据是由图像帧序列组成. 现有的视频分析算法根据每帧图像之间的关系可分为帧间无关与帧间相关. 对于前者, 可以将各帧图像分布至不同节点上并行处理; 对于后者, 由于帧序列之间相关联, 所以需要进行单节点处理. 视频帧处理过程包括图片预处理、特征提取、目标检测与识别等. 在图像处理过程中, 需要调用 opencv 库或者其他图像处理库. 在 Spark 中, 可以通过利用 RDD 上的 pipe 算子, 能够以管道形式调用外部程序对 RDD 中的数据进行处理.

2.2.1 帧间相关

帧间相关是指视频分析算法需要使用连续的视频帧序列才能完成处理, 即当前帧的处理需要依赖之前的帧序列, 如烟雾检测、目标跟踪等. 这里将每路视频流分配在一个节点中处理. 接收器 Receiver 中的 store 方法将接收到的数据保存在本地节点中, 如果将 RDD 中的分区数置为 1, 将保证数据只在本地节点处理. 由于在处理时当前 RDD 中的视频帧数据时可能需要用到过去时间段 RDD 中的视频帧数据信息, 所以需要使用 Dstream 中的 window 算子及 updateStateByKey 算子进行处理.

2.2.2 帧间无关

帧间无关是指视频分析算法可以针对每帧图像进行处理, 而不需要依赖以前的视频数据, 如车牌识别、人脸识别等. 可以将接收到的视频帧数据分配到若干节点中并行处理, 如图 4 所示. 由于视频帧数据集合作为一个 RDD, 所以能够以分区为单位分配到不同节点中处理, 并发度即为分区数. 处理完成后, 通过 collect 算子收集至 driver 端, 使用 Restful 用于前端显示. 由于 Spark 具有容错性, 所以框架本身能够保证不会出现 RDD 分区中的帧数据丢失.

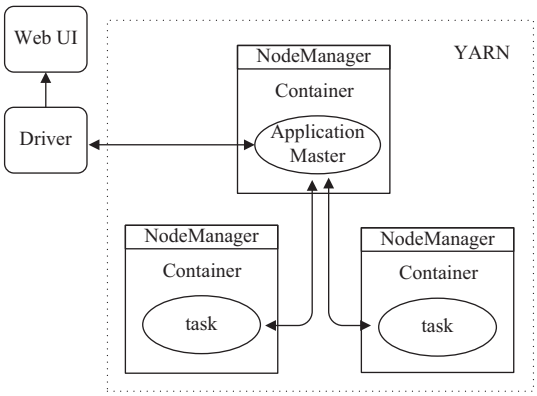


图 4 任务执行图

视频处理层专注于对图像帧数据的分析识别等工作, 而对于识别结果的进一步处理, 以消息数据形式流入消息处理层. 将两个阶段进行耦合, 有利于系统的灵活性与鲁棒性.

2.3 消息处理层

这里采用 Kafka 与 Spark streaming 进行消息流实时分析, Kafka 用于接收视频流处理过程中所产生的消息数据, Spark streaming 用于对 Kafka 中的消息数据进行实时处理. 由于 Kafka 具有高并发、高吞吐量及高可用等优势, 所以经常在流式计算中充当消息中间件. 对视频分析过程中所产生的输出数据集中处理, 有较

大的灵活性和实用价值. 如多路视频间的关联分析, 在交通监控场景下, 可对交通数据进行实时挖掘、实时研判, 为千万上亿级别的过车记录实现关联汇总以及实时预警, 如套牌车分析、黑名单车辆预警、道路流量统计等. 消息分析的框架如图 5 所示. 该模块主要技术细节如下:

① topic 设置: 每路视频对应一个 topic, 每个 topic 包含的 partition 数目依赖具体场景而定. 对于需要顺序消费的场景, partition 数目为 1. 否则可以设置多个分区进行并行消费.

② 消息语义的实现: 在消息处理场景中, 可分为最多一次消费、最少一次消费及精确一次消费. 由于 Kafka 在接收生产者发送的消息时, 可以选择使用确认机制, 所以容易完成生产者对 Kafka 的语义实现. 下面讨论消费者对 Kafka 的精确一次消费语义的实现. 依据是否采用 Receiver, Spark streaming 提供的与 Kafka 集成的接口, 可分为 createStream 与 createDirectStream. 由于后者能够提供更强的语义保证与更高的效率, 所以这里使用 createDirectStream. 这里 Kafka 中需要消费的分区与 RDD 包含的分区一一对应. 消费 Kafka 中的消息时, 主要包括以下步骤: 接收消息、处理消息、输出结果并修改 Kafka 中相应分区的 offset. createDirectStream 能够保证接收消息与处理消息精确一次, 输出结果与修改 offset 的先后顺序将决定最终的语义. 如果先修改 offset, 可能造成部分数据未处理; 如果先输出结果, 可能造成部分数据输出多次. 实现精准一次消费语义, 需要满足: 输出结果幂等或者保证二者作为一个事务操作提交. 以 Postgresql 为例, 每个分区开始消费时需要先从数据库中读取相应的 offset 值放入 TopicAndPartition 中, 并传入 createDirectStream. 处理完毕时需要将两者放在一个事务中提交.

本程序采用 scala 语言编写, 数据库模块采用 ScalikeJDBC 库. 关键代码如下:

```
//读取数据库中所保存的位置信息
val fromOffsets = DB.readOnly { implicit session =>
  sql"select topic, part, off from video_log_offsets".
    map { resultSet =>
      TopicAndPartition(resultSet.string(1),
resultSet.int(2)) -> resultSet.long(3)
    }.list.apply().toMap
}
//从读取的 offset 处, 开始消费
val stream: InputDStream[Long] = KafkaUtils.
createDirectStream[String, String, StringDecoder,
StringDecoder, Long](
  ssc, kafkaParams, fromOffsets,
  (mmd: MessageAndMetadata[String, String]) => 1L)
//事务操作, 保存结果数据与 offset 位置信息
DB.localTx{ implicit session =>
  //保存数据
  val metricRows = sql"""
update txn_data set metric = metric + ${metric}
  where topic = ${osr.topic}
""".update.apply()
  if (metricRows != 1) {
    throw new Exception("...")
      }
//保存 offset 位置
  val offsetRows = sql"""
update t video_log_offsets set off = ${osr.untilOffset}
  where topic = ${osr.topic} and part = ${osr.
partition} and off = ${osr.fromOffset}
""".update.apply()
  if (offsetRows != 1) {
    throw new Exception("...")
  }
}
```

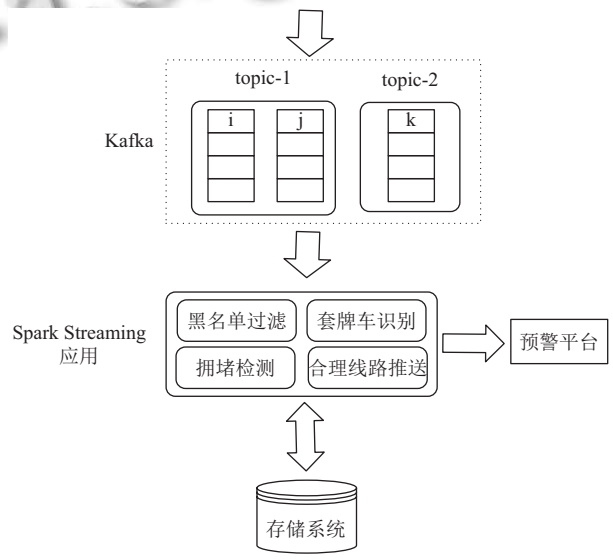


图 5 消息分析框架

2.4 视频信息存储

在存储层, 系统采用 HDFS+HBase+Postgresql+Elasticsearch+kylin 方案. 其中 HBase 用于存储结构化数据, 包括视频数据属性信息. Elasticsearch 与 kylin 用于海量数据的实时检索与多维聚合分析. Elasticsearch 是一种基于 lucence 的高效的分布式全文检索引擎框架. 这里用于多标签查找与全文搜索. Kylin 是一种优秀的 OLAP 框架, 在这里用于对历史数据的实时多维聚合分析. 这对于视频监控领域的分析型场景具有很大实用价值. 视频信息检索分为两个部分: 基于文本的数据查询与基于内容的图像搜索 (如图 6 所示).

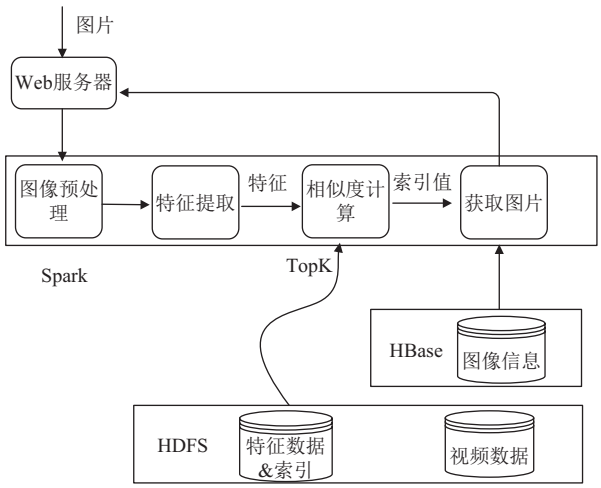


图 6 图像搜索架构图

2.5 容错性分析

视频采集层, 由于 Receiver 具有容错性, 在节点失效或 Receiver 崩溃时, Spark 会在当前节点或是其他节点重新启动.

消息队列层, 由于 Kafka 以分区为单位进行多结点主从分配, 主分区所在节点宕机时会切换至从分区.

生产者发送消息至 Kafka 时会有确认机制, 主从分区全部存储完毕后才对消息进行确认.

在利用 Spark 进行数据计算时, 由于 Spark 引擎本身能够利用 Linearge 及 checkpoint 进行容错恢复, 所以流入的数据在计算过程中不会丢失.

3 实验结果

本实验所运行的服务器集群包含 5 个节点, 集群的硬件配置如下表所示.

实验对比了原系统与本系统的处理性能. 由于日志处理架构为公用部分, 两种系统均可进行接入. 故本实验着重检验了视频处理部分, 并根据实时性、容错性及扩展性对两者进行了对比.

角色	数量	内存	CPU
Master /RM	1	32 GB	Intel(R) Xeon(R) CPU E5-2650 @2.00 GHz
Work /NM	4	8 GB	Intel Core(TM) CPU i3-2120 CPU@3.3 GHz

(1) 实时性

在实验中, 进行 4 路视频实时分析. 包括 2 路帧间相关分析 (包括 camera3 与 camera4) 与 2 路帧间无关分析 (包括 camera1 与 camera2). 分别从 rtsp 流中解码并转换成 opencv 能够处理的 Mat 格式, 调用视频分析算法进行处理. 在帧间相关算法的选择上, 本实验中选取了烟雾检测算法, 该算法根据视频帧序列中的烟雾信息分析其运动特征, 进行烟雾的判别. 在帧间相关算法的选择上, 实验中选取了车牌识别算法, 根据提取的 hog 特征, 依据 SVM+adaboost 分类器进行识别.

原系统的视频流处理模块是采用单机进行处理, 虽然也配置了多个节点, 但节点之间没有关联, 仅仅用于多路视频流的扩展. 不仅无法对帧间无关视频流进行分布式处理, 而且在单点故障上也没有解决.

当算法对单帧图像的处理时间比较长时, 为了保证对流入视频流处理的实时性, 往往采用跳帧的方式进行处理. 这种方式虽然可以缓解这一情况, 但以牺牲视频流的信息为代价. 本系统的分布式处理方案则解决了这一问题. 在实验中, 以每秒能够处理的视频帧作为衡量标准, 实验结果如图 7 所示. 对于帧间无关算法, 由于采用分布式处理方案, 所以每秒处理的帧数很高, 与实时传入的帧数相同. 对于帧间相关算法, 本系统的每秒处理帧数也高于原系统.

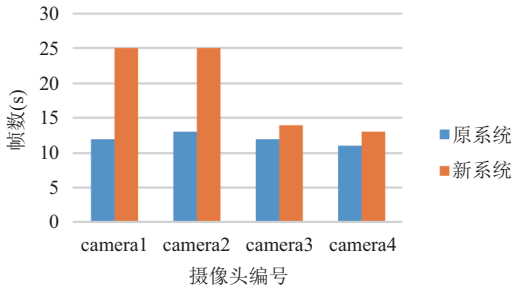


图 7 处理帧数对比图

(2) 容错性

原系统中, 采用守护进程监控所在节点的视频流

处理进程的的运行状态, 如果失效则进行重新启动. 但存在单点故障的情况.

本系统中, 由于 Receiver 具有容错性, 在节点失效或 Receiver 崩溃时, Spark 会在当前节点或是其他节点重新启动. 整个恢复过程只需 1~3 秒.

(3) 扩展性

当新增摄像头时, 原系统需要人工选择节点接入. 对于不同机器的处理能力, 人工往往无法比对. 如果随机选择节点进行接入, 则有可能导致节点负载不均造成处理时延的进一步增加.

而本系统, 基于资源管理平台 YARN, 能够实时获取集群的资源状态, 自动选择合适的节点进行接入并处理, 极大减少了接入时延, 提升了处理效率.

4 结语

本文对基于 Spark 的实时视频分析系统的实现过程进行了详细介绍, 该系统利用当前开源社区主流的分布式组件, 涉及资源管理、计算、存储、消息中间件及大数据即时分析等技术, 具有高可用、实时性等特点. 将传统的对视频数据的单路分析转换为对多路视频信息的关联分析, 通过添加消息分析层, 进一步实时的挖掘不同数据之间的价值, 为更有效的构建分析型业务提供了支撑.

由于 YARN 资源管理器中没有将 GPU 作为资源对象, 而且 Spark 框架也没有提供 GPU 的相关接口, 故这里没有引入. 后续将会进行相关工作以进一步提升处理效率.

参考文献

1 黄凯奇, 陈晓棠, 康运锋, 等. 智能视频监控技术综述. 计算机学报, 2015, 38(6): 1093–1118. [doi: 10.11897/SP.J.1016.2015.01093]

2 Natarajan VA, Jothilakshmi S, Gudivada VN. Scalable traffic video analytics using hadoop mapreduce. The First International Conference on Big Data, Small Data, Linked Data and Open Data. Barcelona, Spain. 2015. 11–15.

3 Apache Software Foundation. Kylin: Extreme OLAP engine for big data. http://kylin.apache.org/.

4 黄文辉, 冯瑞. 基于 Spark Streaming 的视频/图像流处理与新的性能评估方法. 计算机工程与科学, 2015, 37(11): 2055–2060. [doi: 10.3969/j.issn.1007-130X.2015.11.010]

5 Apache Software Foundation. Apache spark, lightning-fast cluster computing. http://spark.apache.org.

6 Apache Software Foundation. Kafka: A distributed streaming platform. https://kafka.apache.org.

7 Apache Software Foundation. HBase: A distributed, scalable, big data store. https://hbase.apache.org.

8 Apache Software Foundation. HDFS: A distributed, scalable File System. https://hadoop.apache.org.

9 Apache Software Foundation. Apache Kylin: Extreme OLAP engine for big data. http://kylin.apache.org.

10 Zaharia M, Chowdhury M, Das T, *et al*. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. 9th USENIX Conference on Networked Systems Design and Implementation. San Jose, CA, USA. 2012.

11 Apache Software Foundation. Spark streaming + Kafka integration guide. http://spark.apache.org/docs/latest/streaming-kafka-integration.html.