计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2020,29(6):64−72 [doi: 10.15888/j.cnki.csa.007454]

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

基于 Storm 的流媒体实时传输系统①

翁小松, 张　征

(华中科技大学 人工智能与自动化学院, 武汉 430074)

摘　要: 为了满足流媒体视频数据在传输过程中对数据的时效性、传输效率及传输的准确性这些高要求, 本文从实时大数据的流处理应用出发, 通过分析流媒体视频数据在实时传输中遇到的难点和所需的关键技术, 采用 Storm 流处理技术实现高性能、低延迟的分布式实时传输系统, 在 Linux 系统上完成 Storm 框架的搭建, 设计并实现了流媒体视频数据的传输拓扑任务, 同时部署 Zookeeper 为框架提供高效可靠的分布式协调服务, 并搭建流媒体服务器用于视频推流后的存储及客户端的点播. 在搭建好 Storm 框架后, 通过了对大规模流媒体视频数据的实时传输测试.

关键词: 流媒体传输; Storm 框架; Linux 系统; 安装部署; 传输测试

引用格式: 翁小松,张征.基于 Storm 的流媒体实时传输系统.计算机系统应用,2020,29(6):64–72. http://www.c-s-a.org.cn/1003-3254/7454.html

Real-Time Streaming Media Transmission System Based on Storm

WENG Xiao-Song, ZHANG Zheng

(School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract: During transmission of streaming video data, there are quite high requirements for timeliness, transmission efficiency, and transmission accuracy. This study focuses on the application of real-time big data stream processing, analyzes the difficulties and key technologies required for streaming media video data in real-time transmission, and achieves a high-performance, low-latency distributed real-time transmission system by the Storm processing technology. Then, we set up the Storm framework on Linux system, design and implement the transmission topology task of streaming video data, while deploy Zookeeper to provide efficient and reliable distributed coordination services for this framework, and finally build a streaming server for video storage and client-on-demand. After setting up the target system, it passed the real-time transmission test of large-scale streaming media data.

Key words: streaming media transmission; Storm framework; Linux system; installation; transmission test

流媒体是指在网络中使用流式传输技术进行下载点播的连续时基媒体, 采用边下载边播放的方式, 缓解了网络带宽的压力和节省了相对传输时间, 因有着良好的时间效应而被广大用户所采纳. 为了符合流媒体在数据传输过程中的稳定性、时效性、可靠性等诸多要求, 已有的流媒体传输系统分别采取了不同的技术和软硬件手段, Fraz 和 Malkani 通过部署高速专用嵌入式处理平台 (DSP), 采用动态 RTP 数据打包技术提高实时视频流系统的性能, 满足系统的高数据吞吐量, 确保了更少的延迟和更好的流媒体质量[1]. 还有基于各种传输协议如 RTSP、MPEG-DASH 等, 基于各种视频压缩技术如 AVS、SVC 等, 改良网络带宽自适应和改良系统资源分配算法等的流媒体传输系统都在一定程度上实现了优秀的视频流传输性能.

20 世纪的 90 年代出现了流处理的概念, 最早应用于数据库技术中, 而分布式流处理系统由原有的分布式系统发展而来, S4、Twitter Storm、Spark Streaming 等技术的发展克服了传统流处理技术在数

① 收稿时间: 2019-11-13; 修改时间: 2019-12-09; 采用时间: 2019-12-20; csa 在线出版时间: 2020-06-10

据传输和资源分配中的不足之处, 由此分布式流处理技术取代了集中式流处理技术[2]. Storm 技术开源于 2011 年, 其有着非常优异的实时性、容错性、鲁棒性、可扩展性等特点, 被广泛应用于金融、交通、电子等服务行业和实时数据计算、实时推荐系统等.

为了将 Storm 在流处理中的优异性能应用到流媒体的传输中, 本文将分析流媒体的视频数据在实时传输中的难点和关键技术, 之后在 Linux 上完成 Storm 框架的搭建, 设计基于 Storm 平台的分布式计算系统和任务拓扑用于流媒体视频数据的实时传输, 并部署 Zookeeper 为框架提供高效可靠的分布式协调服务, 最后, 框架通过了大规模流媒体数据的传输测试, 为框架在实际生产生活中的应用提供参考.

1 需求分析和架构设计

本文的流媒体视频数据传输系统采用大数据流式计算框架 Storm 对视频源数据进行采集、切分、压缩、推流, 最后存储到流媒体服务器中. 本章是针对系统的需求分析, 讲述流媒体传输中的难点和关键技术, 设计整体系统架构 (如图 1 所示) 和各个部分的实现方法.

1.1 流媒体大数据实时传输的需求分析

流媒体视频数据的传输依赖于流媒体技术, 该技术与常规的视频媒体技术之间最大的不同之处在于其可以使流媒体实现边下载边播放, 两者同时进行的实时工作模式, 是一种被广泛应用于视频直播、远程教育、网络电台等的新技术.

实现流媒体视频数据的传输主要有以下几个难点和需要用到的关键技术:

(1) 流媒体传输的实现需要专用的服务器、播放器和合适的传输协议, TCP 协议由于其过多的网络开销不适用于流媒体技术, 所以采用 HTTP/TCP 协议来传输系统的控制信息, 用 RTP/UDP 协议来传输实时的视频数据[3]. 为了能够把服务器的输出重定向到客户机的目的地址, 需要使用上述两种协议来与服务器建立联系.

(2) 进行流媒体传输的视频文件需要使用到视频压缩技术, 将其转换成特定的视频格式, 通常格式的视频文件的容量太大, 在进行网络传输时需要占用过多的资源和花费更长的时间, 而进行压缩后可以有效减少数字视频传输所需的带宽. 由于压缩技术是以消除冗余数据为原则, 会影响到图像质量, 所以需要在处理效率、磁盘空间、视频质量和所需的系统成本之间进行权衡. 在进行格式转换时需要注意在文件中添加“流”信息以便于进行后续的视频的合理切分.

(3) 流媒体视频的传输需要使用缓存技术. 由于网络是动态波动的, 在视频数据分段后每个分组最终所采用的路由是不同的, 导致其到达客户端所使用的的时间也不同, 这时就需要使用缓存技术来保证分组后的数据的时序性, 使得输出做到连续性.

(4) 在 Storm 框架内传递的数据格式是结构化的, 不能直接处理非结构化的视频数据格式. 本文实现一个特定的序列化封装器, 在 Storm 平台上对流媒体视频源数据经采集、切块、分组后, 用于对切分后的视频片段进行对象的序列化和在视频推流阶段的反序列化, 使得视频数据能在 Storm 平台上高效可靠的传输.

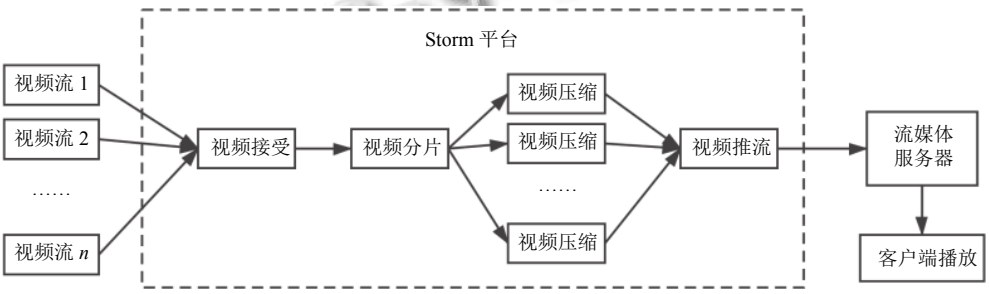


图 1 系统总体架构

1.2 系统整体架构设计

本文所搭建的流媒体视频数据传输系统主要分为两个部分: 用于数据处理和传输的 Storm 平台、用于数据存储和点播的流媒体服务器.

在 Storm 的核心代码任务拓扑 Topology 的主方法入口 main 函数中配置了流媒体的视频源地址及推流地址, 在将任务拓扑提交到 Storm 集群环境中运行后, 由第一部分 Storm 平台负责流媒体视频数据的接

收、分片、压缩和推流等工作, 完成系统的核心处理功能部分. 在 Storm 的第一个 Spout 模块进行源视频的接收和分片, 将处理之后的数据发送到之后的 Bolt 模块进行视频数据的压缩和推送任务, 每个任务分别由一个 Bolt 负责, 减少任务之间的耦合度. Storm 平台上的各个模块之间的协同工作是由其独特的拓扑结构保证的, 本文采用 Storm 的默认调度器来进行任务的资源分配和负载均衡, 提高系统的数据传输效率.

流媒体服务器用于流媒体视频数据在经 Storm 平台推流后的缓存和提供客户端播放器点播. 系统采用开源的 Nginx 轻量级流媒体服务器, 采用 RTMP 协议进行视频数据的传输, 提供视频流的拉取和点播服务, 同时保证了高并发性和稳定性的要求. 同时在视频服务器上集成了 FFmpeg 多媒体视频处理工具用于视频信息的解析、推流等.

流媒体视频数据缓存到流媒体服务器上后, 可以通过客户端的播放器 (VLC) 进行网络串流, 调用流媒体服务器的视频存储端口地址来进行拉流和点播.

2 系统搭建

以 Storm 框架作为流媒体视频数据传输的基础并提供低延迟和高可靠性保证. 本章将围绕 Storm 框架在 Linux 服务器上的搭建流程展开. 介绍 Storm 框架搭建所需的包括硬件环境、软件环境和框架的后台环境配置, 部署 Zookeeper 提供高可用的协调解决方案, 在框架搭建好后启动 Storm 和提交任务拓扑.

2.1 Storm 部署与框架搭建

本文搭建的 Storm 框架的硬件环境配置如下:

华为弹性云服务器上搭建 Ubuntu64 位系统, 并安装可视化界面用于 Storm 集群信息的查看需要, 主节点双核单处理器, 4 GB 内存, 40 GB 硬盘, 副节点单核单处理器, 2 GB 内存, 20 GB 硬盘.

Storm 所需的软件开发环境如下:

操作系统: Ubuntu 7.4.0-1Ubuntu1-18.04.1

JDK 版本: JDK1.8.0_231

Storm 版本: Storm2.0.0

Zookeeper 版本: Zookeeper3.4.14

Python 版本: Python2.7.2

Zeromq 版本: Zeromq4.2.2

Jzmq 版本: JJzmq2.1.0-SNAPSHOT

Maven 版本: Apache-maven-2.5.5

Maven 作为一个项目管理工具用于系统的项目代码管理, 包括依赖包源码的下载编译, 程序的 jar 打包, 必要时还可以充当 bug 调试工具.

在 Storm 的所有 Supervisor 节点中, 需要部署 Zookeeper 分布式应用程序协调服务作为协调者, 在集群运行时发挥如下作用[4]:

(1) Nimbus 和 Supervisor 之间是没有直接的信息传递的, Nimbus 在接收 Storm 集群的任务拓扑后, 将任务信息写入 Zookeeper 中, 提供给 Supervisor 从节点读取这些任务的状态信息, 从而分配资源;

(2) 在 Task 执行失败或 Supervisor 节点宕机时, Zookeeper 可以获得失败信息, 使得 Nimbus 主节点可以根据心跳信息来重启失败的 Task 或 Supervisor.

2.2 Storm 启动和任务提交

Storm 的启动依赖于 JDK 环境, Zookeeper 的部署是必要的, 便于监控整个 Storm 集群的状态信息. Zookeeper 的部署需要使用 Maven 进行编译, 本文所采用的的 Zookeeper3.4.14 版本是编译好的版本, 所以省略这一步骤. 在 Storm 框架及其所需的一系列软件开发环境搭建完毕后, 可以启动 Storm 服务, 需要先运行 Zookeeper 服务, 首先进入 Zookeeper 的安装目录下, 执行如下命令: bin/zkServer.sh start, 启动服务后可以运行 bin/zkServer.sh status 命令查看 Zookeeper 的运行状态. Zookeeper 成功运行后, 进入 Storm 的 bin 目录下, 运行下列命令完成 Storm 框架的启动:

```
./storm nimbus &
./storm ui &
./storm supervisor &
./storm logviewer &
./storm drpc &
```

在这些命令中, Nimbus 主节点的启动应优先于 Supervisor 副节点, 防止集群信息报错, UI 和 Logviewer 服务的启动顺序没有要求, DRPC 用于分布式远程调用 Storm 集群的计算资源, 而省略连接集群中的具体节点的过程, 该项服务和 Logviewer 都是可选项, 只有在具体任务需要时才会发挥作用.

打开浏览器, 进入 localhost:8080 查看 WebUI 界面 (如图 2 所示), 验证 Storm 是否启动成功 (所有 Storm 的进程必须在后台运行, 否则会占用终端控制台). Storm 框架的各个组件的运行情况.

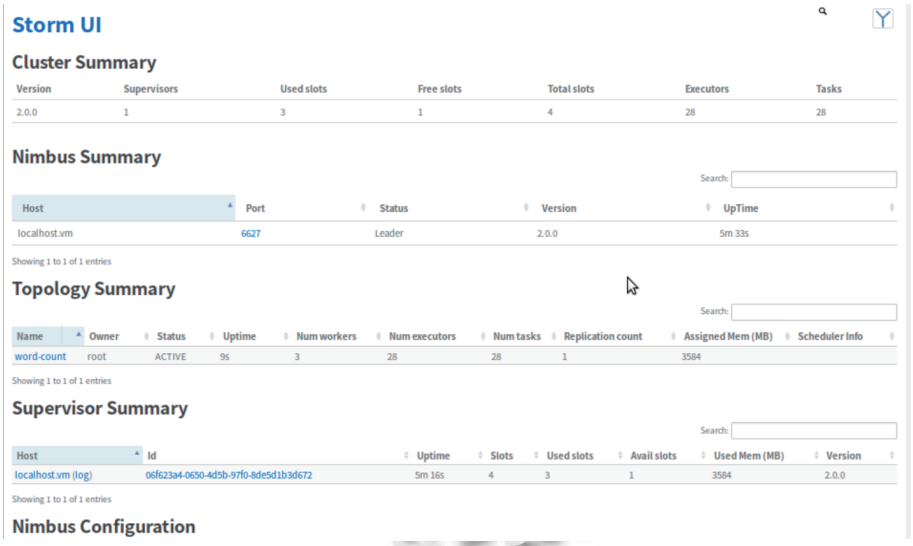


图 2 Storm 的 WebUI 图

在 WebUI 图中, 第一栏显示的是 Storm 的版本信息、Supervisor 启动个数、已使用和未使用的端口数量及端口总数、程序中所定义的 Executors (线程数) 和 Tasks (任务数). WebUI 图中还包括 Nimbus 主节点所在的服务器信息, 集群上所运行的 Topology (任务拓扑) 的相关信息、Supervisor 从节点的服务器信息以及集群的配置信息等, 每一类信息都提供了查看更深层次细化信息的接口按钮.

Storm 任务提交和执行过程示意图如图 3 所示.

Storm 的整体工作流程可以简化为以下步骤: 客户端新建 Topology, 在其中定义 Spout 和 Bolt 的初始并发度, 即初始的 Executors 个数, 并定义各个组件之间的流分组策略, 之后 Client 提交 Topology 到 Nimbus 中; Nimbus 分配任务, 根据 Topology 定义中给定的参数, 下载对应的依赖包的源代码数据, 并将分配好的任务提交到 Zookeeper 上; 子节点 Supervisor 会通过定期查询 Zookeeper 中的信息, 分配具体的 Worker 以及 Executors 执行具体的 Tasks[5].

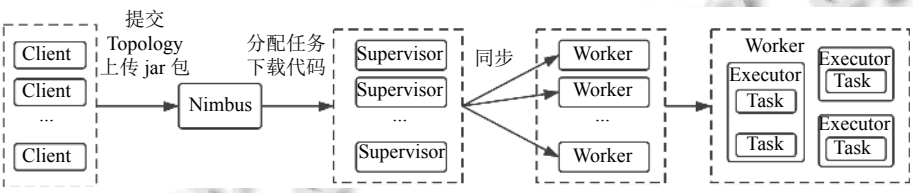


图 3 Storm 任务提交和执行过程示意图

Storm 的 Topology 拓扑任务可以通过 Maven 编译打包成 jar, 相关的配置信息都集中在 pom.xml 文件中, 包括各个依赖包资源及其版本号等, 在 pom.xml 文件所在目录下运行 mvn package assembly:single 命令可以在编译测试后创建 Target 目录并生成一个 xxx-jar-with-dependencies.jar 文件, 这个文件中包含了 Storm 集群环境运行所需的依赖资源和工程源代码[6]. 在打包完成后, 运行如下命令:

storm jar /home/song/storm/xxx.jar cn.storm.topology.WordCountTopology

其中, /home/song/storm/xxx.jar 是程序实现代码经 Maven 命令编译打包成的包含依赖包资源在内的 jar 文件及其所在的相对于终端位置所在的文件路径, cn.storm.topology.WordCountTopology 是程序的主方法入口, 如果测试程序要在集群上运行, 需要在命令后面追加任务的名字, 否则会在本地模拟模式下运行.

2.3 流媒体服务器搭建

在流媒体视频流数据经过 Storm 平台的处理到达推送步骤后, 需要流媒体服务器进行视频流的接收和分发. 本文搭建的 RTMP 流媒体服务器是基于 Nginx

开源项目的轻量级流媒体服务器, 作为流媒体视频数据的存储服务器, 同时提供第三方客户端使用播放器进行视频的网络串流点播[7].

Nginx 的搭建流程如下:

(1) 安装 GCC 和相关 C++工具;

(2) 安装依赖库 libpcre3, libpcre3-dev;

(3) 安装 libssl-dev 和 OpenSSL 工具;

(4) 解压后的 Nginx 重新编译和安装.

如果 Nginx 编译成功, 在/etc/nginx 目录下修改配置文件 nginx.conf, 添加 RTMP 的推流端口 live, 然后在/usr/local/nginx/sbin 目录下启动 Nginx 的主程序, 启动后可以在浏览器中打开 localhost 的特定端口查看启动情况以及服务器的相关信息.

FFmpeg 是一个音视频软编解码和 RTMP 流发送接收的完整解决方案, 本系统的视频数据采用 H.264 进行编解码, 使用 FFmpeg 命令执行视频流的拉流转推任务, 完成对视频数据的相关处理操作.

3 系统测试

在 Storm 框架搭建完毕后, 编写 Topology 拓扑任务用于视频数据的接收、分片、压缩、推流, 并提交 Storm 集群执行, 进行流媒体大数据的传输测试, 使其满足流媒体对于数据传输的实时性、高容错性、数据完整性的要求.

3.1 拓扑任务设计

针对在 Storm 平台上传输的流媒体视频流需要选择一个合理的切块分组方案, 能够方便后续的连续化处理和压缩推流等操作. 视频切分的依据是视频编码时的关键帧信息, 流媒体视频数据中主要包括 3 种类型的编码帧信息: I 帧 (关键帧)、B 帧和 P 帧[8].

系统采用 GOP (2 个关键帧之间的间隔) 作为视频流数据切分的基本单位, 通过对前一个 GOP 的冗余片段进行切分, 将切分后的视频片段顺序发送到 Storm 中, 以此形成连续化的视频组数据流[9].

序列化是流媒体视频数据在 Storm 平台上传输的关键步骤, 它将非结构化的视频格式对象转换为结构化的数据类型, 使得视频数据可以在网络中进行传递, 之后在系统的推流阶段对数据进行反序列化, 将视频数据的类型还原, 完成数据的传输过程. 在 Storm 框架中, 集成了 Kyro 序列化技术, Kyro 序列化处理后的数据占用更少的内存, 比通用的 Java 序列化的效率更高, 耗时更少[10]. 本文在 Kryo 序列化的基础上编写了更加适用于系统的新序列化器, 便于视频数据在 Storm 平台上的传递.

传统的视频格式如 rmvb、mkv 等占用的容量太大, 不利于视频数据在网络中的传输, 所以需要在传输前进行格式的压缩. 对于流媒体视频数据的压缩来讲, 影响视频最终性能的因素有很多, 主要是压缩效率和速率调节方面, 本文在对视频数据进行压缩阶段采用 H.264 视频编码算法, 拥有更节约的码率、更高质量的视频画面和更强的网络适应性, 并提供了差错恢复能力. 视频流切分流程图如图 4 所示.

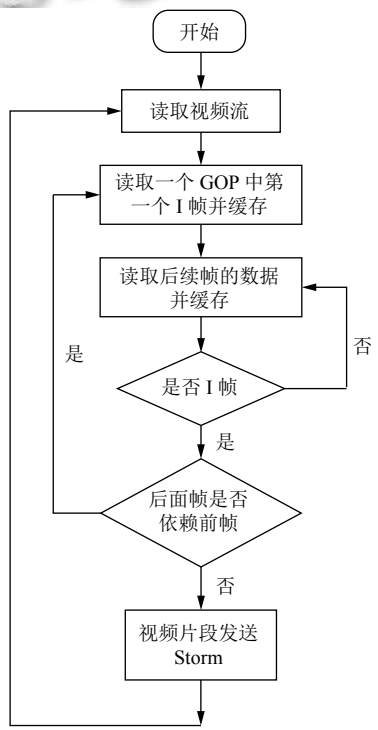


图 4 视频流切分流程图[9]

WorkFrames 封装类型用于对 Storm 接收的视频源经过切分后的数据片段实例化. Storm 平台在读取了流媒体视频流分段数据后, 无法直接使其在系统中进行信息的传递, 自定义的 WorkFrames 数据结构和序列化器 WorkFramesSerializer 如图 5 所示, 用于对视频数据进行分组转换和序列化操作, 使其能进行实时流传递, 其中的 sequenceId 表示 Spout 采集到的视频数据片段的序号, 而 streamId 表示在 Storm 平台上传递的视频流编号, getTuple 方法用于获取 Bolt 中的视频数据, metadata 中的 Object 对象存储了实际序列化后

的流媒体视频数据. WorkFramesSerializer 序列化器中的 write 方法序列化视频数据, 而 read 方法则是反序列化方法, fromTuple 方法和 toTuple 方法用于 write 序列化和 read 序列化的抽象封装, 传递 Storm 中的数据单元.

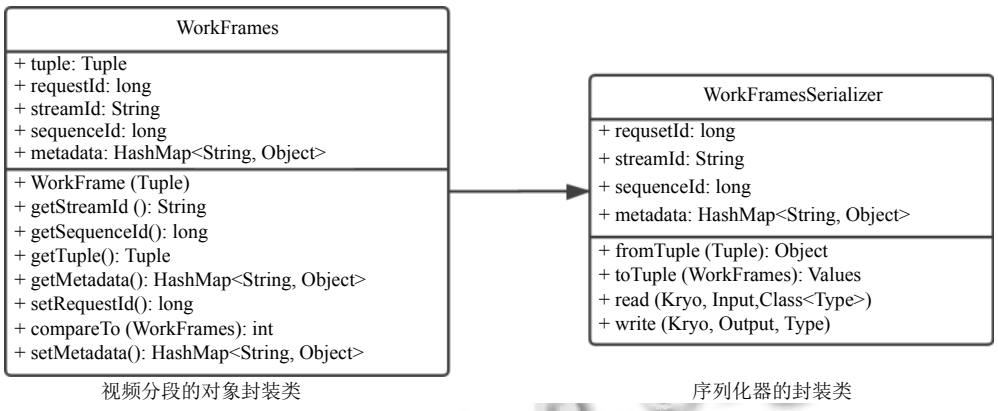


图 5 对象封装类和序列化器封装类

在系统的实时传输作业中, Spout 数据采集器模块负责读取 RTMP 协议的视频流, 并对视频数据文件进行切分, 序列化切分后的视频片段, 使其能在 Storm 平台上进行数据传递, 然后将数据发送给拓扑中的下一个组件. Bolt 组件在接收到 Spout 的数据信息后对其进行并行压缩操作, 最后将反序列化后的视频数据推流到流媒体服务器中 (由 2 个 Bolt 分别处理, 减少逻辑之间的耦合), 提供播放器的拉流点播.

新建 VideoChannelTopology 作为整个拓扑的主类, 在主方法入口 main 函数中添加流媒体视频源地址信息、Storm 集群配置信息、Spout 和 Bolt 各组件模块的创建信息和数据分组模式等. 具体拓扑图如图 6.

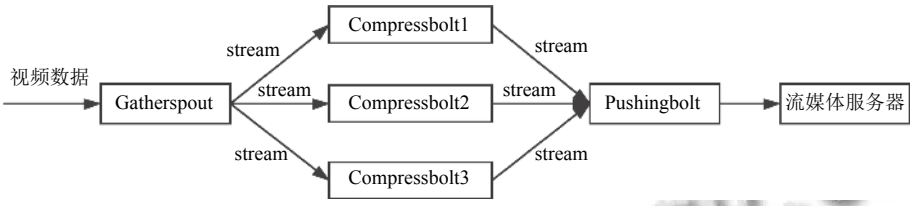


图 6 视频实时传输拓扑图

GatherSpout 是整个拓扑任务的数据采集模块, 负责对流媒体视频流数据的采集, 并完成对视频文件的切块和分组操作, 之后调用基于 kyro 的自定义序列化器 WorkFramesSerializer 对封装后的 WorkFrames 视频对象进行序列化, 将序列化后的数据单元 tuple 并发的发送到下游的多个 CompressBolt 中.

系统的 Storm 平台上一共运行 2 种类型的 Bolt 组件: CompressBolt 和 PushingBolt, 上游的 CompressBolt 采用并发方式运行, 接收从采集器 Spout 组件传来的数据, 调用自定义序列化器的反序列化方法得到具体的视频数据, 然后对视频分段数据采用 H.264 编码算法进行压缩操作, 完成后赋值到新的对象 output 中, 进行二次序列化成 value 类型, 调用 tuple 的 emit 方法传递数据到一个下游 PushingBolt 中, 同时进行 ack 应答, 保证数据的可靠性. 下游的 PushingBolt 则负责进行处理后的视频数据的推送, 使其保存到流媒体服务器上, 由于要保证视频片段分组顺序的正确性, 采用滑动窗口模式来避免在传输过程中的延迟问题, 对视频片段数据采用缓存处理, 保证推送视频服务的稳定性和连续性[11].

流媒体服务器上存储的视频文件可以通过流媒体播放器 (VLC 等) 进行网络串流, 采用 RTMP 实时消息传输协议, 通过配置流媒体服务器的地址 IP 和对应的端口号打开远程链接, 拉取视频数据并播放.

3.2 性能测试

在项目的程序代码编写完毕后, 通过 Maven 命令打包成 jar 在测试通过后提交到服务器的 Storm 集群执行. Storm 的 Topology 拓扑运行细节如图 7 所示.

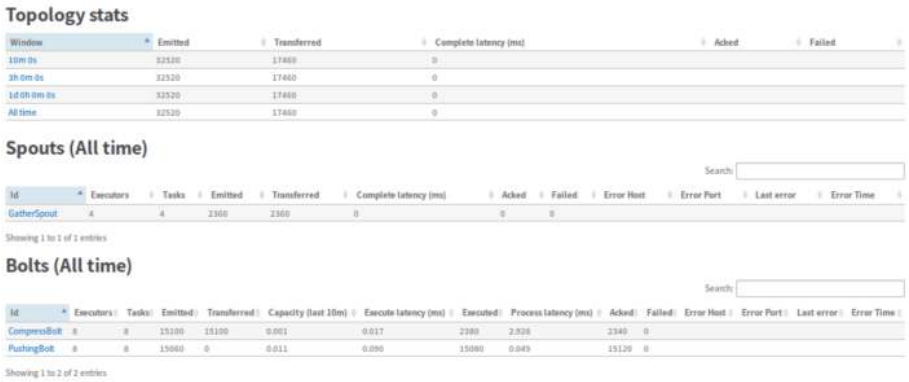


图 7 拓扑运行图

通过在系统集群的 UI 中进行数据量的定量统计和实时性分析, 主要集中在整体拓扑性能和集群中分组件传输效率, 表 1 为截取 10 min、30 min、1 h 的数据分析图, 整理后得出该集群在传输效应方面的数据表.

表 1 拓扑统计信息表

Topology	10 min	30 min	1 h
emitted	330 392	956 980	1849 300
tansferred	177 123	513 140	991 560

在拓扑信息统计表中, emitted 表示窗口发射出去的元组总数, 即输出收集器上调用 emit 方法的次数, transferred 则表示基于时间窗口所有发送至任务的元组数量, 元组发送到流上可能没有组件立即订阅读取, 这会使得其数量少于发射的数量.

在 Spout 的信息统计表 (表 2) 中, emitted 表示该组件上已发射元组数量, transferred 表示发送至其他任务上的元组数量, acked 表示应答的元组数, 在任务设计时 Spout 作为消息收集器不作消息应答, 所以该项数值始终为 0, failed 表示失败的元组数量.

表 2 gatherSpout 统计信息表

GatherSpout	10 min	30 min	1 h
emitted	23 892	74 060	134 000
transferred	23 892	74 060	134 000
acked	0	0	0
failed	0	0	0

在 Bolt 的信息统计表 (表 3、表 4) 中, emitted 和 transferred 分别表示该 Bolt 上发射的元组数量和实际传输元组数量, 由于 pushingBolt 作为拓扑中的最后一个节点, 不需要再将消息发送给下一个组件, 所以其 transferred 的值始终为 0, executed 表示该 Bolt 上处理的元组数量, acked 表示应答的元组数量, 两者数量相等表示消息的可靠性高, failed 表示失败的元组数量.

表 3 CompressBolt 统计信息表

CompressBolt	10 min	30 min	1 h
emitted	153 212	474 480	857 560
transferred	153 212	474 480	857 560
executed	23 886	74 180	134 040
acked	23 922	74 160	133 960
failed	0	0	0

表 4 PushingBolt 统计信息表

PushingBolt	10 min	30 min	1 h
emitted	153 235	483 180	873 000
transferred	0	0	0
executed	153 260	483 060	872 920
acked	153 296	483 100	872 900
failed	0	0	0

通过折线图 (图 8~图 10) 可以看出, 系统在运行过程中的数据处理能力相对平稳, 在 3 个时间点的信息处理量基本符合等比增长规律, 单位时间的系统性能没有遭遇瓶颈. 在集群的数据吞吐量方面, 在综合考虑带宽因素影响的前提下, 基本满足视频数据的传输需要. 在实时性和可靠性方面, 通过几个折线图的横向对比, 消息在流中传输时, 经过处理后由 acked 进行应答, 两者比例接近 100%, 说明消息的每次处理都收到了系统的反馈, 没有出现失败的消息元组.

本文还进行了系统的数据传输量测试, 针对的是 Worker 并发数的变化所引起的对视频处理性能的影响. 具体的测试方法是在 Storm 平台的视频压缩处理模块进行多路视频流的并发传输任务, 测试时在拓扑任务的主方法中配置组件信息, 采集器 Spout 设置为

8 个, 由 8 个 Acker 来保障数据传输的可靠性, 避免消息应答失败导致的数据错误, 每个消息的大小限定为 100 KB, 设置 24 个 CompressBolt 用于视频流数据的压缩任务. 在集群上设置了多个 10 s 时间段的消息记录器, 具体任务测试时设置不同数量的 Workers 任务并发度, 统计集群在每 10 s 所处理的 Tuple 数量, 绘制出不同 Workers 并发数下的 Tuple 处理量的折线图 (图 11), 以此直观的表现系统的集群性能变化情况.

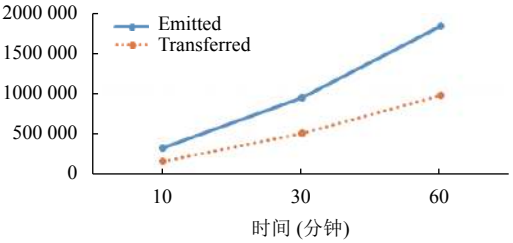


图 8 拓扑信息统计折线图

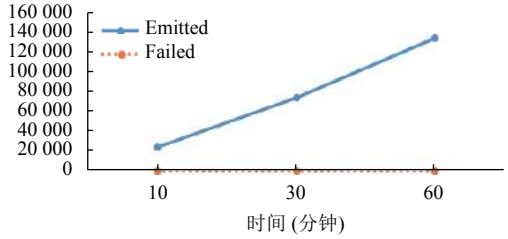
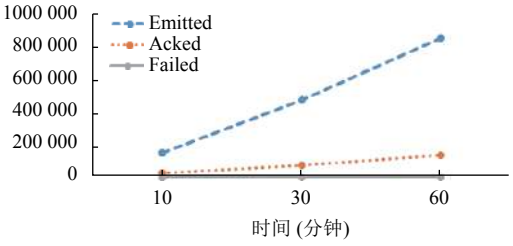
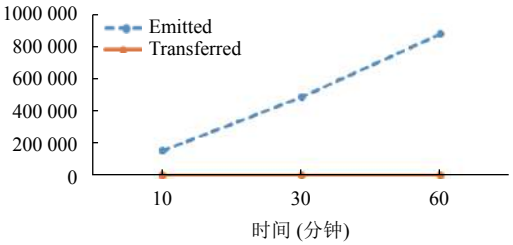


图 9 Spout 信息统计折线图



(a) Compressbolt 信息折线图



(b) Pushingbolt 信息折线图

图 10 Bolt 信息统计折线图

对于视频作业任务而言, 每增加一个 Worker 数, 会增加同时间段内系统的数据处理数和系统平台的线程处理量, 如 Netty 收发线程和心跳线程, 同时会使得原来在线程间内存通信的组件变成网络通信, 降低了系统的吞吐量[12]. Workers 进程数量的增加不会造成系统传输效率的无限增长, 在 Workers 过大时反而会因信息阻塞造成系统性能的下降. 经测试发现, 在 8 个 Workers 并发度的影响下, 系统的传输效率达到了最大化. 影响系统传输性能的因素还有很多: 服务器的硬件条件, Storm 集群的任务调度算法, 拓扑任务的消息复杂度等, 其对系统的影响并不都是线性和正相关的, 这有待于后续更深入的研究.

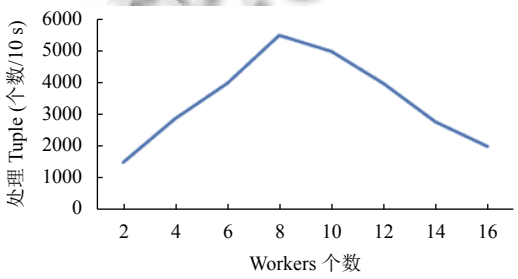


图 11 不同 Workers 数对系统性能的影响

4 结语

流媒体的实时可靠传输是当今网络技术实践中一个重要的研究内容, 被广泛应用于各种视频资源网站的点播和视频直播中. Storm 作为一个免费开源的分布式实时计算框架, 设计用于在容错和水平可扩展方法中处理大量数据, 利用 Storm 可以很容易做到可靠地处理无限的数据流, 很好地满足了大数据在容量不断膨胀扩大, 又对实时性有着高要求的现状[13]. 本文分析了流媒体视频数据传输的原理和需要解决的问题难点及所需的关键性技术, 设计并实现了基于 Storm 的流媒体视频数据传输系统. 之后在云服务器的 Linux 系统中搭建了 Storm 集群, 并部署了 Zookeeper 分布式协调服务. 在 Storm 框架搭建成功后, 编写任务拓扑实现视频的接收、分片、压缩和推流, 搭建流媒体服务器用于流媒体视频数据经 Storm 平台推流后的存储和客户端播放器的拉取点播, 之后进行了流媒体视频大数据的传输测试, 保证了 Storm 框架能完成大数据的实时可靠、高并发量的传输.

参考文献

1 Fraz M, Malkani YA, Elahi MA. Design and implementation of real time video streaming and ROI transmission system

using RTP on an embedded Digital Signal Processing (DSP) platform. Proceedings of the 2009 2nd International Conference on Computer, Control and Communication. Karachi, Pakistan. 2009. 1–6. [doi: 10.1109/IC4.2009.4909244]

2 靳永超, 吴怀谷. 基于 Storm 和 Hadoop 的大数据处理架构的研究. 现代计算机: 专业版, 2015, (4): 9–12.

3 丁维龙, 赵卓峰, 韩燕波. Storm 大数据流式计算及应用实践. 北京: 电子工业出版社, 2015. 15–30.

4 陈敏敏, 王新春, 黄奉线. Storm 技术内幕与大数据实践. 北京: 人民邮电出版社, 2015. 7–9.

5 李川, 鄂海红, 宋美娜. 基于 Storm 的实时计算框架的研究与应用. 软件, 2014, 35(10): 16–20. [doi: 10.3969/j.issn.1003-6970.2014.10.003]

6 Im DH, Cho CH, Jung IG. Detecting a large number of objects in real-time using Apache Storm. Proceedings of 2014 International Conference on Information and Communication Technology Convergence. Busan, Republic of South Korea. 2014. 836–838.

7 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.

8 闻谦. 基于云计算的视频实时转码系统设计与实现[硕士学位论文]. 成都: 电子科技大学, 2018.

9 Bhuiyan RHM. Low delay video transcoding services on distrubuted computing platform [Technical report]. Sweden: Blekinge Institute of Technology, 2016.

10 姚欣, 王劲松. 基于 Apache Storm 的大规模网络流量实时监控系统研究. 天津理工大学学报, 2016, 32(6): 25–29, 47.

11 杨杰, 朱邦培, 吴宏伟. 基于 Storm 的高速公路实时交通指数评估方法的研究与实现. 计算机应用研究, 2017, 34(9): 2707–2713. [doi: 10.3969/j.issn.1001-3695.2017.09.032]

12 Sun DW. Big data stream computing: Features and challenges. Big Data Research, 2015, 1(3): 99–105.

13 牛牧. 基于 Kafka 的大规模流数据分布式缓存与分析平台[硕士学位论文]. 长春: 吉林大学, 2016.