Storm是个实时的、分布式以及具备高容错的计算系统，进程常驻内存，数据不经过磁盘,在内存中处理，Twitter开源的分布式实时大数据处理框架最早开源于github

官网<http://storm.apache.org>

Storm计算模型：

Topology-DAG有向无环图的实现，对于Storm实时计算逻辑的封装即由一系列通过数据流相互关联的 Spout、Bolt所组成的拓扑结构

生命周期:此拓扑启动后会在集群中一直运行直到kill

Tuple元组：Stream中最小数据组成单元

Stream数据流

从Spout传递数据给Bolt、以及上一个Bolt传递数据给下一个Bolt,形成的数据通道即Stream，Stream声明需要指定ID（默认Default）

Spout-数据源：拓扑中数据流的来源，一般会从指定外部的数据源读取元组( Tuple)发送到拓扑( Topology)中，先通过 OutputFieldsDeclarer中的declare方法声明定义的不同数据流，发数据时通过 SpoutOutputCollector的emit方法指定数据流ID(streamId)参数将数据发送出去，Storm线程会不断调用主动从数据源拉取数据通过emit方法将数据生成元组(Tuple)发送给之后的Bolt计算

Bolt-数据流处理组件：拓扑中数据处理由Bolt完成，Bolt通过OutputFieldsDeclarer中的declare方法声明定义的不同数据流，发数据时通过SpoutOutputCollector的emit方法指定数据流ID(streamId)参数将数据发送给其他Bolt，Bolt的execute方法负责接收到个元组(Tuple)数据、真正实现核心的业务逻辑

public class WsSpout extends BaseRichSpout {

Map map;

TopologyContext topologyContext;

SpoutOutputCollector spoutOutputCollector;

String[] strings={"hello zdd bj","zdd nihao world","bj nihao hi"};

Random random=new Random();

public void open(Map map, TopologyContext topologyContext, SpoutOutputCollector spoutOutputCollector) {

this.map=map;

this.topologyContext=topologyContext;

this.spoutOutputCollector=spoutOutputCollector;

}

public void nextTuple() {

List line=new Values(strings[random.nextInt(strings.length)]);

this.spoutOutputCollector.emit(line，数据表示ID);

System.err.println("sput--------------------------------"+line);

Utils.sleep(1000);

}

public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {

outputFieldsDeclarer.declare(new Fields("line"));

}

@Override

public void ack(Object msgId) {//数据完整传递后会被调用，不能保证数据被重复计算，但能保证数据至少正确执行一次

super.ack(msgId);

}

@Override

public void fail(Object msgId) {//数据完整传递失败会被调用

super.fail(msgId);

}

}

public class WsBolt extends BaseRichBolt {

Map map;

TopologyContext topologyContext;

OutputCollector outputCollector;

public void prepare(Map map, TopologyContext topologyContext, OutputCollector outputCollector) {

this.map=map;

this.topologyContext=topologyContext;

this.outputCollector=outputCollector;

}

public void execute(Tuple tuple) {

String line=tuple.getString(0);

String[] words = line.split(" ");

for (String word : words) {

List w = new Values(word);

this.outputCollector.emit(tuple,w);//将Tuple发送给后方的Bolt以便进行数据错误定位

}

}

public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer) {

outputFieldsDeclarer.declare(new Fields("w"));

}

}

public class WcountBolt extends BaseRichBolt {

Map map=new HashMap<String,Integer>();

public void prepare(Map map, TopologyContext topologyContext, OutputCollector outputCollector) {}

public void execute(Tuple tuple) {

String word=tuple.getStringByField("w");

int count=1;

if (map.containsKey(word)){

count=Integer.parseInt(map.get(word).toString())+1;

}

map.put(word,count);

System.err.println(word+"--------------------"+count);

}

public void declareOutputFields(OutputFieldsDeclarer outputFieldsDeclarer){}

}

public class StormStudy {

public static void main(String[] args) {

TopologyBuilder topologyBuilder=new TopologyBuilder();

topologyBuilder.setSpout("wssput",new WsSpout());

topologyBuilder.setBolt("wsbolt",new WsBolt()).shuffleGrouping("wssput");

// topologyBuilder.setBolt("wcountbolt",new WcountBolt()).shuffleGrouping("wsbolt");//轮询平均分配Stream流中的Tuple到每个Bolt中

topologyBuilder.setBolt("wcountbolt",new WcountBolt(),3).fieldsGrouping("wsbolt",new Fields("w"));//设置3个Bolt并发执行，将Stream流中的Tuple按照指定字段分组分配不同的Bolt，allGrouping广播发送每个Tuple到所有Bolt中，globalGrouping全局分组将Tuple分配Bolt编号最小的Bolt

topologyBuilder.setBolt("wcountbolt",new WcountBolt(),Executor线程数量).setNumTasks(Task任务数量).fieldsGrouping("wsbolt",new Fields("w"));

Config config=new Config();

config.setNumWorkers(Worker进程数量);

LocalCluster localCluster=new LocalCluster();

localCluster.submitTopology("wordcount",new Config(),topologyBuilder.createTopology());

}

}

Storm流式处理分为异步和实时请求应答服务（同步）

分布式远程调用DRPC（Distributed Remote Procedure Call）

实时请求处理

Storm安装部署：

安装JDK，Python，ZooKeeper

解压Storm安装包

修改Storm安装目录/conf/storm.yaml配置文件：

storm.zookeeper.servers:

[- "hadoop1"]ZooKeeper服务器

nimbus.host: "hadoop1" nimbus主节点hosts

storm.local.dir "/home/kaku/tmp/storm" 数据文件目录

supervisor.slots.ports: supervisor从节点端口号即Work，一个Work对应一个Slot和端口号

[- 6700]

drpc.servers:

[- "hadoop1"]分布式远程调用服务器

Storm安装目录/bin/storm nimbus >> ./logs/nimbus.out 2>&1 & 后台启动Nimbus，日志输出到nimbus.out

Storm安装目录/bin/storm ui >> ./logs/ui.out 2>&1 & 后台启动Nimbus的UI，日志输出到ui.out

Storm安装目录/bin/storm supervisor >> ./logs/supervisor.out 2>&1 & 后台启动Supervisor，日志输出到supervisor.out

Nimbus服务器存在单点故障问题但由于Nimbus无状态（所有状态信息存放在ZooKeeper中管理）和快速失败（遇到异常自动kill）一般对Nimbus做高可用HA，Supervisor进程无状态和快速失败，Supervisor会重启失败的Worker进程，Worker进程一直失败会被Nimbus分配到其他服务器上执行，Tuple拥有唯一的ID（16位2进制数字，进行异或比较数据传输的完整性）

Storm集群节点可以运行多个Worker进程，一个Topology拓扑对应一个或多个Worker进程，Worker进程可以生成多个Executor线程，每个Executor线程对应一个（默认一个）或多个同一组件（Spout或Bolt）的Task（执行数据处理的最小单元）任务，进程线程数量可在程序中指定

Storm安装目录/bin/storm jar jar包路径名 入口类全类名 jar包参数 Storm集群运行jar包任务

Storm安装目录/bin/storm help rebalance 查看重新负载均衡命令帮助

Storm安装目录/bin/storm rebalance 拓扑名 -n 新Worker数量 -e 自定义组件名=Executor数量 重新指定拓扑任务的并行负载

Storm事务（Transaction）：

强有序顺序流，每个Tuple对应一个TransactionID（从1开始，每个Tuple按顺序加1），一次只能处理一个Tuple无法实现分布式计算

强顺序的Batch流，事务以batch（一批Tuple）为单位，每个batch对应一个TransactionID，batch内部可以并行计算，由于事务处理的一致性所以会有资源浪费的情况

Storm事务设计，将Topology拆分成两个阶段，Processing phase允许并行处理多个batch，Commit phase保证batch强有序，一次只能处理一个batch

Storm通过ZooKeeper存储所有Transaction相关信息