Storm

xumingmingv

内容提要

- 简介
- 编程模型
- 架构模型
- 容错性
- 可靠性
- 伸缩性
- Topology生命周期
- Q & A

Storm是什么

Storm

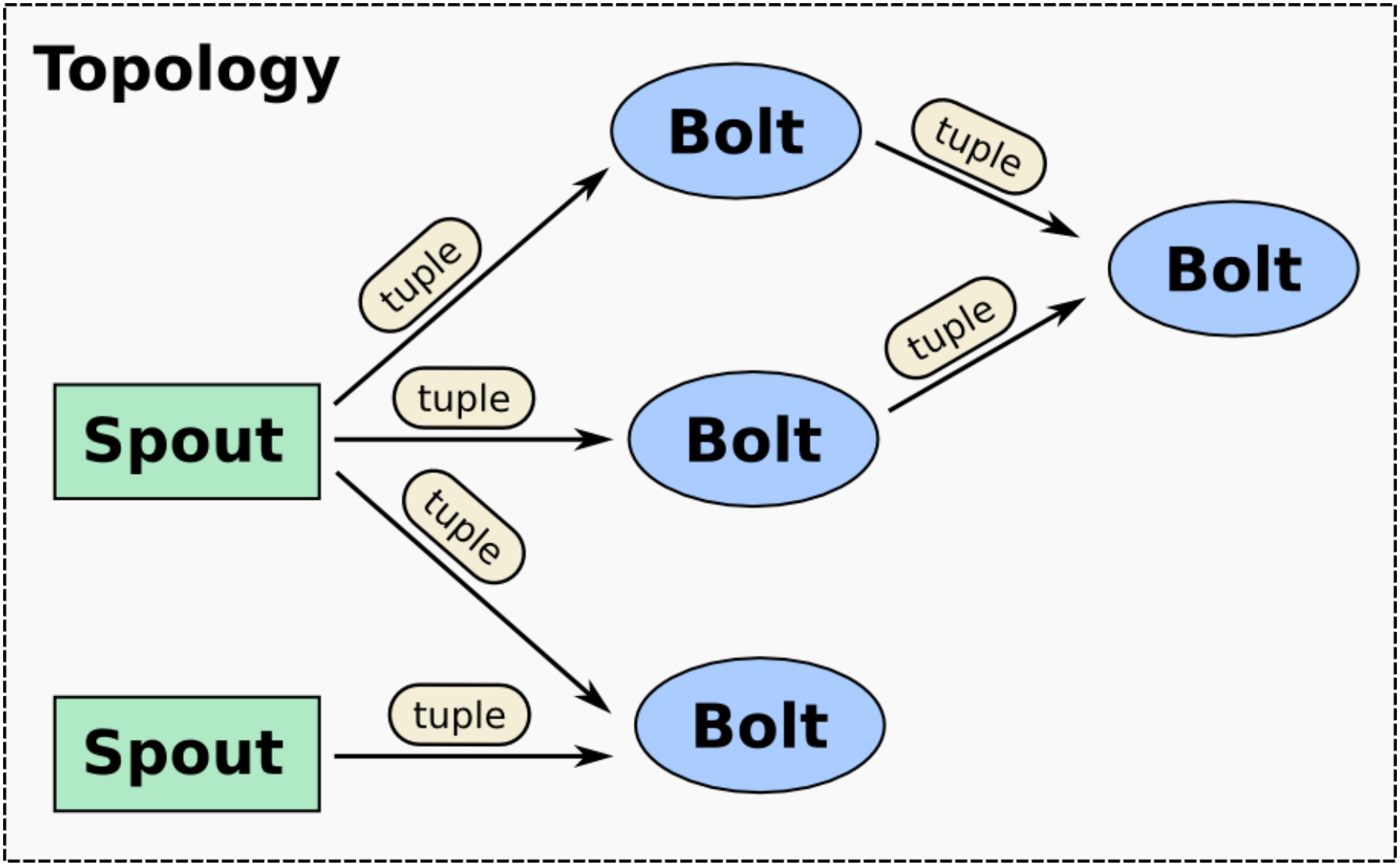
Distributed and fault-tolerant realtime computation

Storm is a free and open source distribu
Storm makes it easy to reliably process
for realtime processing what Hadoop di
simple, can be used with any programm

Hadoop

Storm

Storm编程模型


Topology
Bolt
tuple
Bolt
tuple
tuple
Spout
tuple
Bolt
tuple
Spout
tuple
Bolt

Tuple

- Tuple就是我们要处理的消息，我们可以把它简单的看作一个JSON数组:
  - [{“name”: “james”}, {“age”: 24}]

```
/**
 * Gets the field at position i in the tuple. Returns object since tuples are
 */
public Object getValue(int i);
```

```
tuple.getValue(0) => “james”
```

```
public Object getValueByField(String field);
```

```
tuple.getValueByField(“age”) => 24
```

Spout

- Spout是topology里面tuple的源头，topology里面所有的tuple都是从Spout里面来的。
- Spout的tuple是怎么来的？通常会监听一个网络端口，或者链接到一个消息队列等待消息过来。

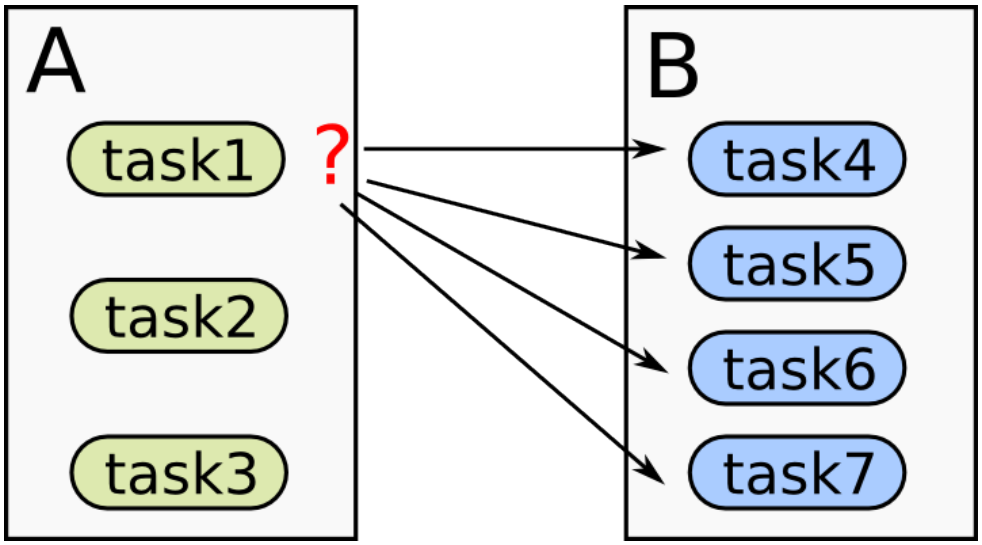
```
/**
 * When this method is called, Storm is requesting that the Spout emit tuples to the
 * output collector. This method should be non-blocking, so if the Spout has no tuples
 * to emit, this method should return. nextTuple, ack, and fail are all called in a tight
 * loop in a single thread in the spout task. When there are no tuples to emit, it is courte
 * to have nextTuple sleep for a short amount of time (like a single millisecond)
 * so as not to waste too much CPU.
 */
void nextTuple();
```

Bolt

- Bolt是实际处理消息的组件，封装着对消息计算逻辑的。
- Bolt会最终以多个task的形式运行于Storm集群中。
- Bolt可以从Spout接收tuple，也可以从另一个bolt接收消息，这样一个bolt接收另外一个bolt的消息，最后会组成一个拓补结构，这也是topology的单词的含义。

```
 * It is required that all input tuples are acked or failed at some point using the OutputCollector.
 * Otherwise, Storm will be unable to determine when tuples coming off the spouts
 * have been completed.</p>
 *
 * <p>For the common case of acking an input tuple at the end of the execute method,
 * see IBasicBolt which automates this.</p>
 *
 * @param input The input tuple to be processed.
 */
void execute(Tuple input);
```

分派策略(Grouping)



分派策略(Grouping)

- **shuffle**：随机派发，保证所有task收到的tuple数大致相等
- **fields**：按照tuple里面某个字段值来分派，保证字段值相同的tuple被派给同一个tuple。
- **all**：每个task都会收到所有的tuple
- **global**：所有的tuple发给同一个task，现在的实现是发给序号最小的那个task

分派策略(Grouping)

- **direct**：直接指定发给哪个task
- **local or shuffle**：如果当前worker里面有下游task，那么直接发给这个task，否则应用shuffle grouping
- **none**：表示我们不关心到底如何分派tuple，目前的实现就是shuffle grouping

分派策略(Grouping)

```
public interface CustomStreamGrouping extends Serializable {

    /**
     * Tells the stream grouping at runtime the tasks in the target bolt.
     * This information should be used in chooseTasks to determine the target tasks.
     *
     * It also tells the grouping the metadata on the stream this grouping will be used on.
     */
    void prepare(WorkerTopologyContext context, GlobalStreamId stream, List<Integer> targetTasks);

    /**
     * This function implements a custom stream grouping. It takes in as input
     * the number of tasks in the target bolt in prepare and returns the
     * tasks to send the tuples to.
     *
     * @param values the values to group on
     */
    List<Integer> chooseTasks(int taskId, List<Object> values);
}
```

Topology

- Topology相当于Hadoop的Job，是Storm里面的工作单元。由Spout和Bolt组成。
  - 除非主动删除，否则topology永远不会停。

WordCountTopology

有一些的句子，要求算出这些句子里面每个单词出现的次数

不考虑分布式的算法

- 把每个句子分隔成一个个单词
- 对每个单词计数

RandomSentenceSpout

```
public class RandomSentenceSpout extends BaseRichSpout {
    SpoutOutputCollector _collector;
    Random _rand;


    @Override
    public void open(Map conf, TopologyContext context, SpoutOutputCollector collector) {
        _collector = collector;
        _rand = new Random();
    }

    @Override
    public void nextTuple() {
        Utils.sleep(100);
        String[] sentences = new String[] {
            "the cow jumped over the moon",
            "an apple a day keeps the doctor away",
            "four score and seven years ago",
            "snow white and the seven dwarfs",
            "i am at two with nature"};
        String sentence = sentences[_rand.nextInt(sentences.length)];
        _collector.emit(new Values(sentence));
    }

    @Override
    public void ack(Object id) {
    }
```

SplitSentence

```
public static class SplitSentence extends ShellBolt implements IRichBolt {

    public SplitSentence() {
        super("python", "splitsentence.py");
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("word"));
    }

    @Override
    public Map<String, Object> getComponentConfiguration() {
        return null;
    }
}
```

这个Spout是调用这个python脚本实现的

split_sentence.py

```
1 import storm
2
3 class SplitSentenceBolt(storm.BasicBolt):
4     def process(self, tup):
5         words = tup.values[0].split(" ")
6         for word in words:
7           storm.emit([word])
8
9 SplitSentenceBolt().run()
```

WordCount

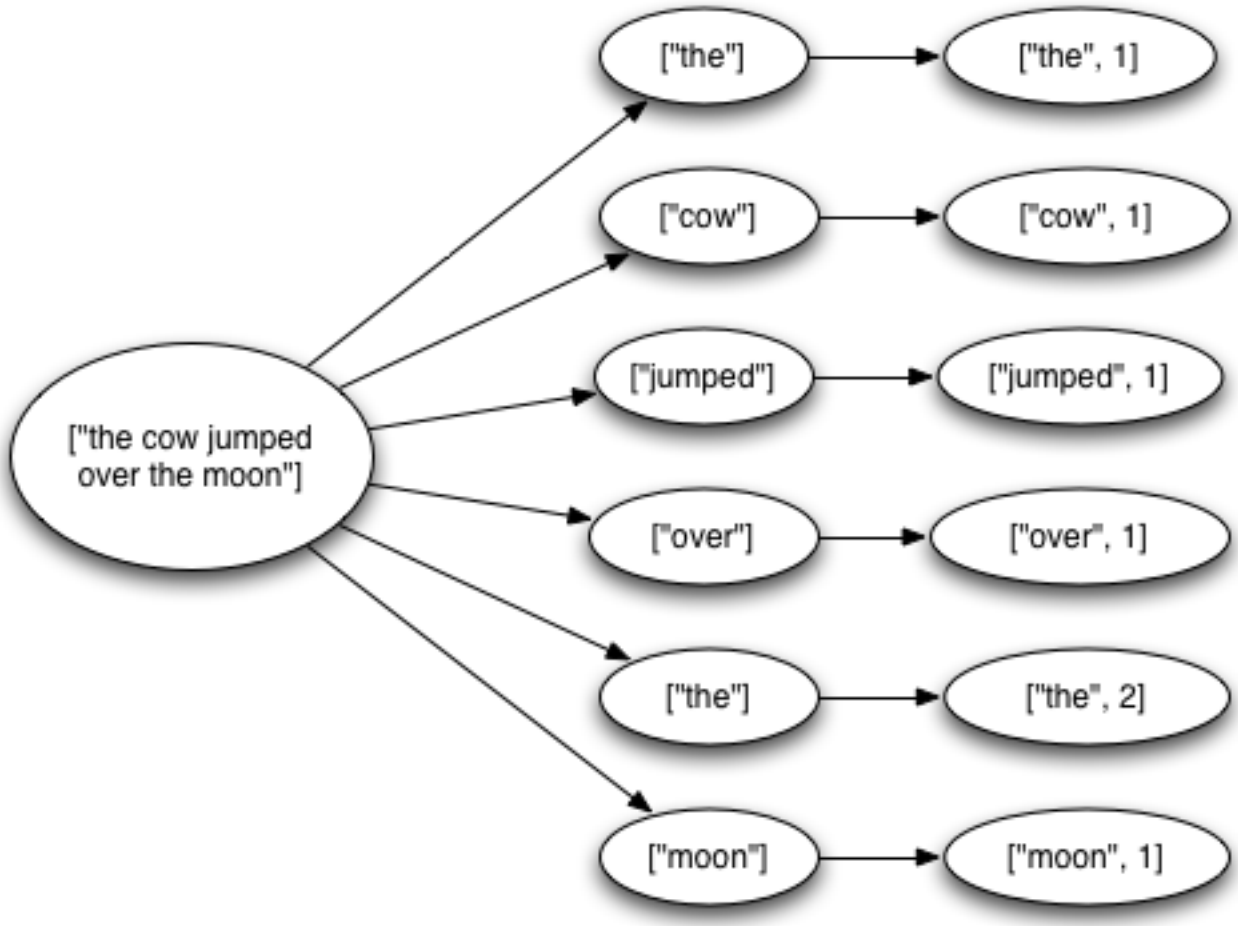
```
public static class WordCount extends BaseBasicBolt {
    Map<String, Integer> counts = new HashMap<String, Integer>();

    @Override
    public void execute(Tuple tuple, BasicOutputCollector collector) {
        String word = tuple.getString(0);
        Integer count = counts.get(word);
        if(count==null) count = 0;
        count++;
        counts.put(word, count);
        collector.emit(new Values(word, count));
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("word", "count"));
    }
}
```

从tuple里面取出要处理的单词

发射tuple给下游


["the cow jumped over the moon"]
["the"]
["the", 1]
["cow"]
["cow", 1]
["jumped"]
["jumped", 1]
["over"]
["over", 1]
["the"]
["the", 2]
["moon"]
["moon", 1]

buildTopology

```
public static void main(String[] args) throws Exception {

    TopologyBuilder builder = new TopologyBuilder();

    builder.setSpout("spout", new RandomSentenceSpout(), 5);

    builder.setBolt("split", new SplitSentence(), 8)
             .shuffleGrouping("spout");
    builder.setBolt("count", new WordCount(), 12)
             .fieldsGrouping("split", new Fields("word"));

    Config conf = new Config();
    conf.setDebug(true);


    if(args!=null && args.length > 0) {
        conf.setNumWorkers(3);

        StormSubmitter.submitTopology(args[0], conf, builder.createTopology());
    } else {
        conf.setMaxTaskParallelism(3);

        LocalCluster cluster = new LocalCluster();
        cluster.submitTopology("word-count", conf, builder.createTopology());

        Thread.sleep(10000);

        cluster.shutdown();
    }
}
```

设置spout

设置bolt

用8个线程来运行这个bolt

从上游的split接收tuple，接收的方式是fields grouping

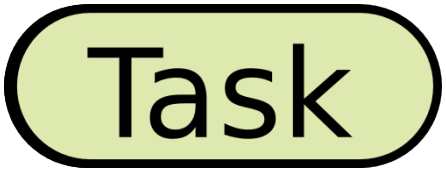
用三个进程来执行这个topology

把这个topology提交到集群

Storm架构模型

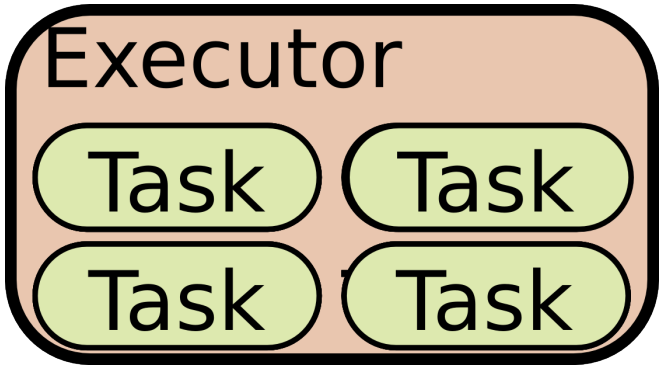
- Task
- Executor
- Worker
- Supervisor
- Nimbus

Task



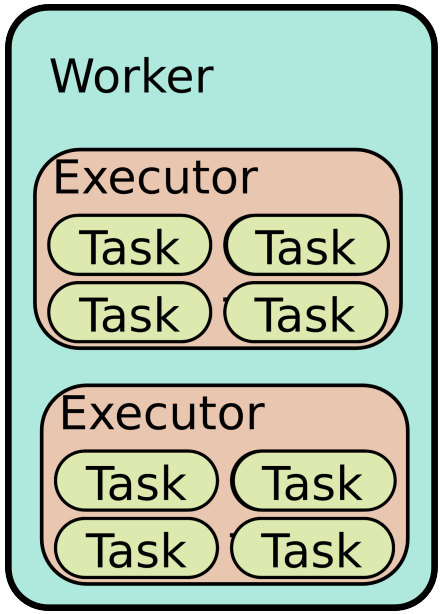
- 不管是Spout还是Bolt，在Storm里面都是以一个个Task的形式去执行的。
- Task其实就是一段代码逻辑
  - 对于Spout就是你的nextTuple方法实现
  - 对于Bolt就是你的execute方法实现
- ComponentConfigurationDeclarer.setNumTasks()

Executor



- 一个Executor对应到一个JVM线程
- 一个Executor里面可以执行多个task，比如[1 3]，表示该Executor里面执行1,2,3这三个task
- 一个Executor里面只能运行一个Spout/Bolt的task

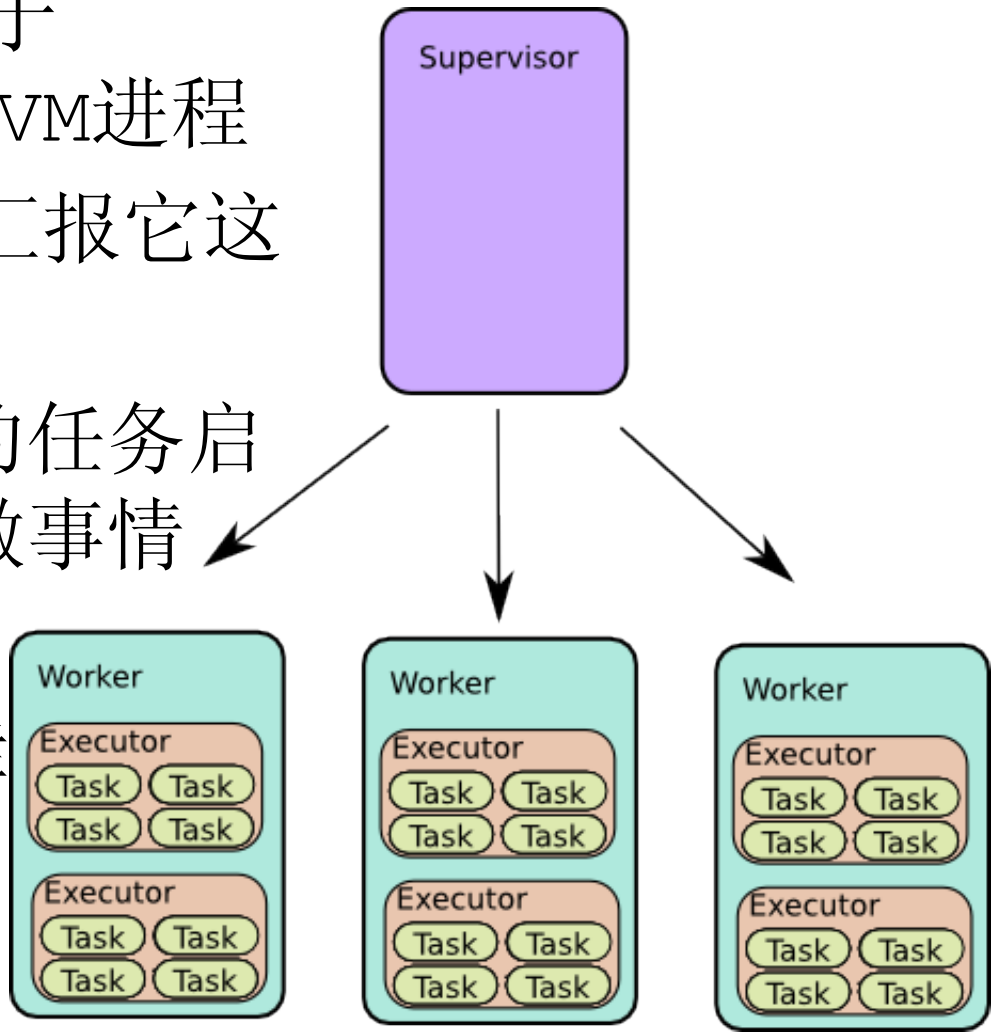
Worker



- Worker对应到一个JVM进程
- 按照分配给自己的任务，executor来执行任务
- Worker控制运行一个topology的总进程数
  - Conf.setNumWorkers()

Supervisor

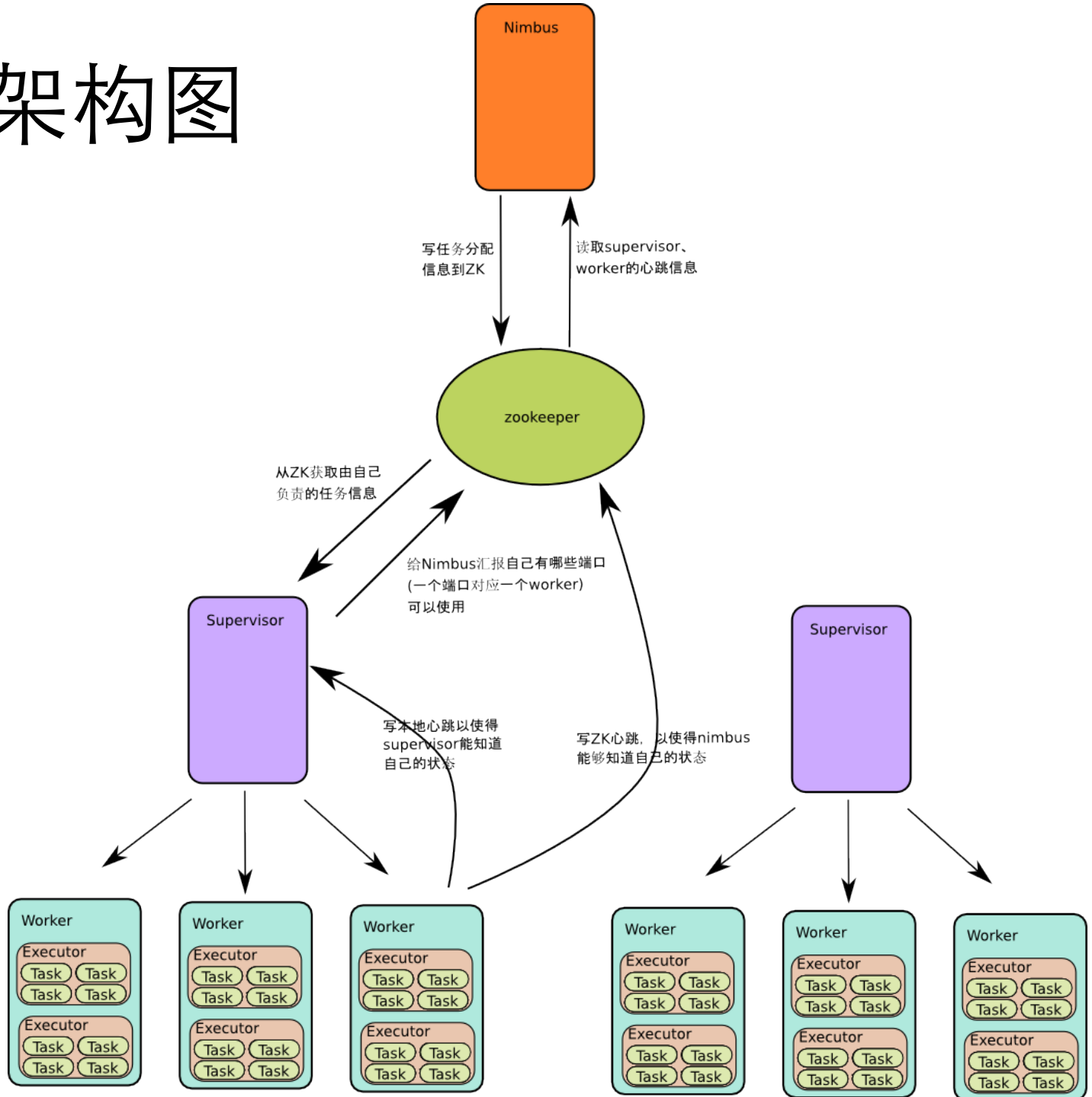
- Supervisor是独立于Worker的另外一个JVM进程
- 它通过ZK向Nimbus汇报它这里有哪几个端口可用
- 按照Nimbus的分配的任务启动指定的worker来做事情
- 监控worker的本地heartbeat，如果挂么重启。



Nimbus

- 接受用户提交的topology
- 根据用户的配置分配worker、executor来执行这个topology，这个任务分配信息会写到ZK
- 监控supervisor、worker的心跳信息，如果有supervisor、worker挂掉，重新分配任务

整体架构图


Nimbus
写任务分配
信息到ZK
读取supervisor、
worker的心跳信息
zookeeper
从ZK获取由自己
负责的任务信息
给Nimbus汇报自己有哪些端口
(一个端口对应一个worker)
可以使用
Supervisor
Supervisor
写本地心跳以使得
supervisor能知道
自己的状态
写ZK心跳，以使得nimbus
能够知道自己的状态
Worker
Executor
Task
Task
Task
Task
Executor
Task
Task
Task
Task
Worker
Executor
Task
Task
Task
Task
Executor
Task
Task
Task
Task
Worker
Executor
Task
Task
Task
Task
Executor
Task
Task
Task
Task
Worker
Executor
Task
Task
Task
Task
Executor
Task
Task
Task
Task
Worker
Executor
Task
Task
Task
Task
Executor
Task
Task
Task
Task
Worker
Executor
Task
Task
Task
Task
Executor
Task
Task
Task
Task

容错性(1)

- 如果集群里面的一台机器挂了怎么办?

所有分配到这台机器上的计算任务会被分配到其它机器上面去。

容错性(2)

- 如果一个Worker挂了怎么办？

Supervisor会去重启这个Worker的，如果这个Worker一启动就挂掉，那么Nimbus得不到这个worker的心跳信息，然后就会把原本分配给这个worker的任务分配到其它Worker

容错性(3)

- 如果Nimbus或者Supervisor挂了怎么办?
  - 因为Nimbus和Supervisor本身是无状态的，所有的状态数据都是保存在zk或者硬盘上面。
  - Nimbus和Supervisor都被设计成快速失败的(fail-fast)，在执行的过程中如果遇到异常，立即主动退出的。
  - 我们一般利用daemontools或者monit这种工具来运行Nimbus，Supervisor，这样一旦检测到进程没了，马上重启一下就好了。
  - 在Nimbus和Supervisor重启的时候，并不影响worker进程，它们继续执行。

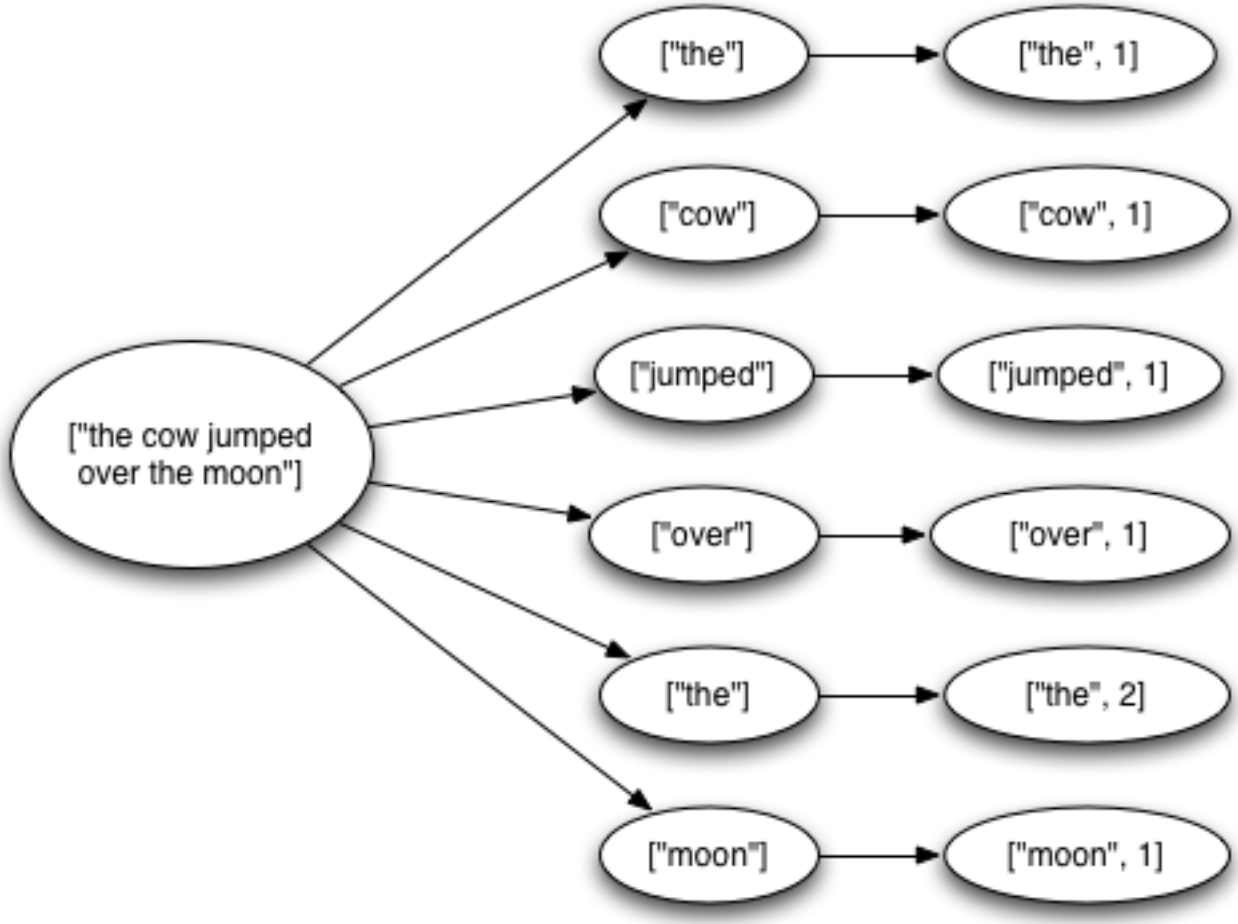
可靠性

“Storm保证所有Spout Tuple会被至少成功处理一次。”

- 一个简单的办法
- Storm的办法

一个Spout Tuple被成功处理是指这个Spout Tuple以及由这个Spout Tuple所引起的所有的Bolt Tuple都被成功处理。

Tuple树


["the cow jumped over the moon"]
["the"]
["the", 1]
["cow"]
["cow", 1]
["jumped"]
["jumped", 1]
["over"]
["over", 1]
["the"]
["the", 2]
["moon"]
["moon", 1]

一个简单的做法

- 保存这个tuple，以及由这个tuple所导致的所有tuple的处理状态:
  - 123456789: success
  - 123456790: success
  - 123456791: fail
  - …

占用空间

- 假设每秒1000万条spout tuple消息，由每条spout tuple会再间接产生100条tuple消息，即每秒的消息量：10亿条消息

8byte(tuple id为long类型) * 1000万 * 100 = 8000000kb = 8000m = 8G

Storm的做法

- 为每个spout-tuple 维护一个20byte的记录
  - **[spout-tuple-id ack-val task-id]**
- Spout每发射一个新spout tuple，添加一条记录
- Bolt发射一个新Bolt Tuple的时候更新ack-val值
  - **new-ack-val = old-ack-val XOR new-tuple-id**
- Bolt每ack一个tuple，再更新ack-val一次，算法同上
- 成功处理的判定条件：
  - **ack-val == 0**

1000万 * 20byte = 200000kb = 200M

伸缩性

- 两个影响topology并行度的因素
  - Worker数量
  - Executor数量

```
storm rebalance -n new-num-workers
                -e component1-id=executor-count
                   component2-id=executor-count
```

Topology的生命周期 – 提交

- Topology的提交：用户执行storm jar命令
  - **storm jar mytopology.jar mytopology**

Topology的生命周期 – 初始化

- Nimbus接收到这个topology，在zookeeper上面为该topology初始化数据，包括心跳目录(worker、executor的心跳信息写在这里)
- Nimbus对这个topology做出任务分配，并把这个分配信息写入zookeeper。
  - 代码配置信息：该topology的jar包、配置信息在nimbus上的目录
  - 谁来运行： executor到worker的映射关系。
    - **{”machine1-6100": [[1 3] [4 6]], ”machine2-6200": [[7 9] [20 25]] }**
  - executor的启动时间。

Topology的生命周期 – Supervisor下载任务

- Supervisor上面有一个线程，每10秒做如下事情：
  - 从zookeeper上面下载任务分配信息，并把它写到本地磁盘 **{"6100": [[1 3] [4 6]], "6200": [[7 9] [20 25]] }**
  - 从zookeeper上看哪些topology被分配给该supervisor运行，如果这个topology的jar包还没下载过，那么下载。

Topology的生命周期 – 启动worker

- Supervisor上面的另外一个线程读本地磁盘的任务分配信息，把这个任务分配信息跟现在正在运行的任务进行比较，该干掉的worker干掉，该启动的worker启动。
- Worker从Supervisor那里拿到任务之后，启动对应的executor线程来执行。
- Worker还会启动一个线程来监控集群的变化（因为它也要给它的下游bolt发送tuple），如果它的下游worker被重新分配了，那么worker会自动重连到新的worker。

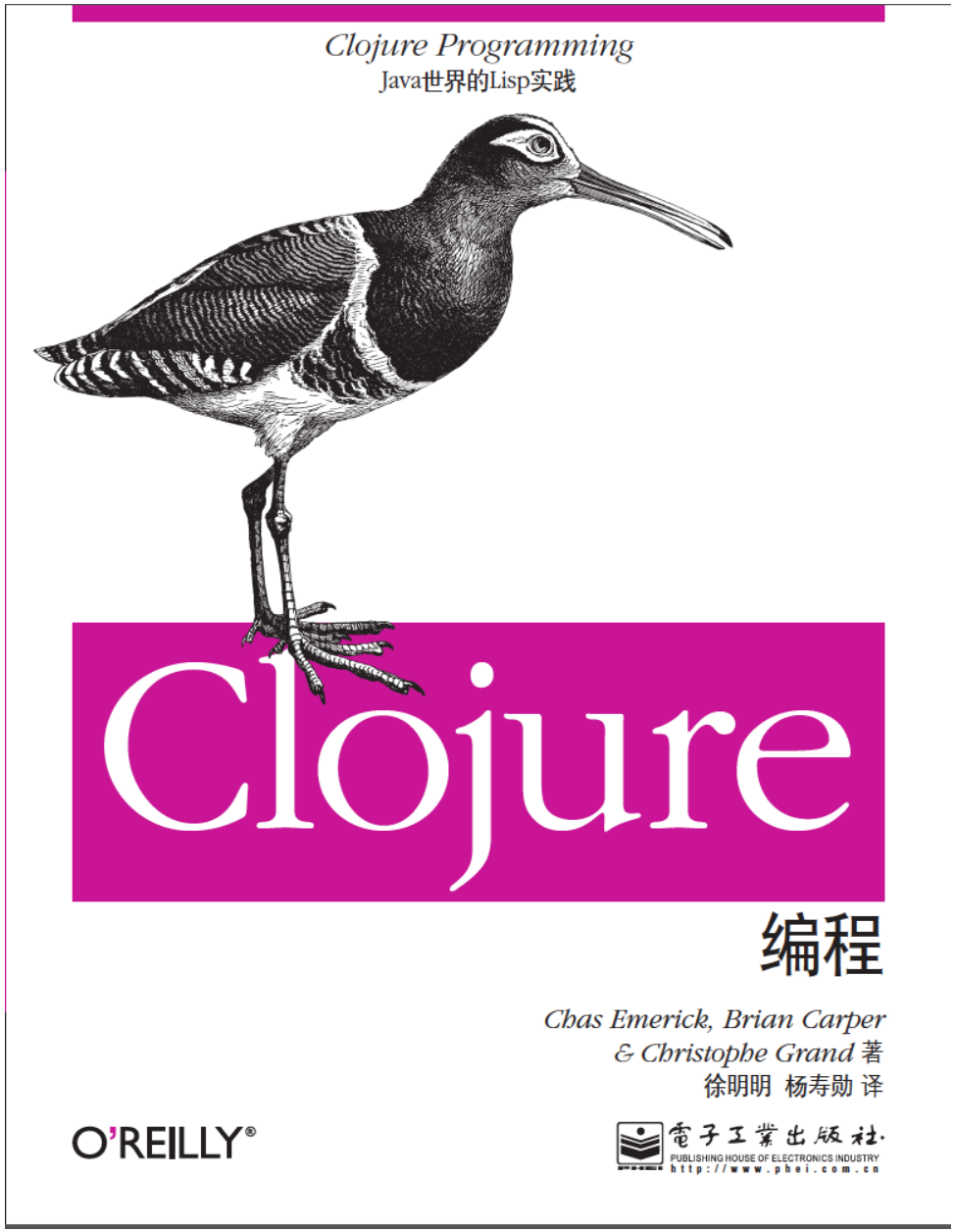
Topology的生命周期 – 启动executor

- Executor线程启动，“启动”它所对应的task，反序列化对应的bolt/spout对象，调用它的prepare方法，准备开始处理消息
- tuple首先是发到worker那里，从tuple里面可以看到它要发给哪个task，而根据task executor的对应关系，把这个tuple转给对应的executor，对应的task去处理
- 在topology的运行过程中，nimbus会监控supervisor、worker、executor的心跳，如果有的worker挂掉了，超时了，对topology的任务进行重新分配。

Topology的生命周期 – 终结

- **`storm kill mytopology`**

《Clojure编程》


Clojure Programming
Java世界的Lisp实践
Clojure
编程
Chas Emerick, Brian Carper
& Christophe Grand 著
徐明明 杨寿勋 译
O'REILLY®
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Q & A