Storm 入门与实践

徐冠鹏

zhujun@taobao.com

2012 年 11 月 3 日

个人简介

- 工作经历
  - 一淘-广告事业部-算法平台支持组
  - 2005 年 7 月毕业于北邮计算机系
  - 同年 10 月加入阿里巴巴，之后一直从事互联网广告相关业务的研发工作
  - 2011 年加入算法平台支持组，负责海量日志数据的实时处理和分析工作
- 个人兴趣
  - 2005 年开始使用 GNU Emacs，并因此了解到了 Lisp
  - 2006 年通过 SICP 学习了 Scheme 的相关知识，并认识到了 Lisp 的强大，其后的编程风格均受其影响
  - 2006 至 2007 年之间为 GNU Emacs 贡献了补丁和文档
  - 2011 年因工作需要接触了 Storm，同时开始学习 Clojure，目前工作中非生产任务利用 Clojure 完成

入门篇

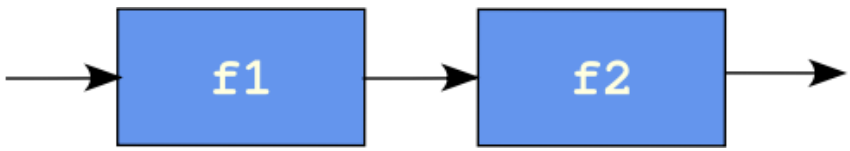
流数据处理的背景

- 早期的流数据处理
  - 单机处理
  - UNIX Shell

```
cat input | grep pattern | sort | uniq -c >output
```

  - 函数式编程语言中的约定界面

```
(def stream1 (map f1 input-stream))
(def stream2 (map f2 stream1))
```



- 分布式流数据处理
  - 集群计算
  - S4
  - Storm

分布式流数据处理的诞生

- 海量数据集
  - 每秒数十万条日志，每天日志总量数十亿
  - 每秒数据上百兆，每天数据 TB 级别
- 实时数据分析需求
  - 淘宝网买家：搜索结果的推荐
  - 淘宝网卖家：效果营销报表
  - 防作弊
  - 数据分析结果生成得越快，用户体验越好

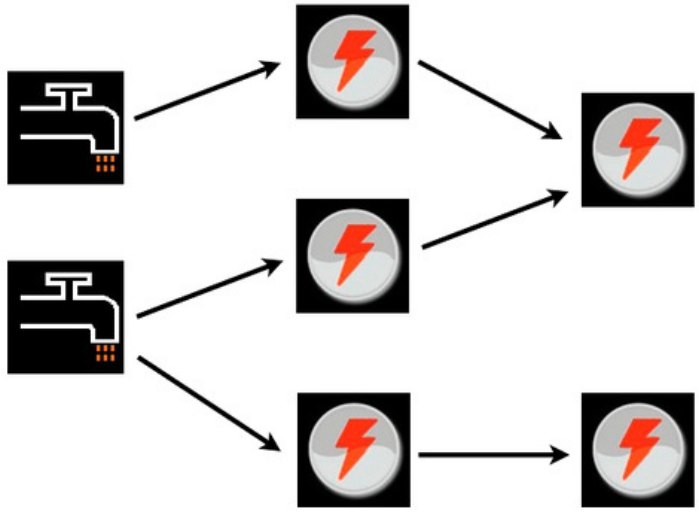
分布式流数据处理与其他处理方式的不同

- 时效性
  - 专有系统 (如搜索引擎): 毫秒级
  - 分布式流处理系统 (如 Storm): 秒级
  - 分布式批处理系统 (如 MapReduce): 分钟级
- 数据处理模式
  - 分布式流数据处理 (如 Storm)：数据来一条、处理一条、输出一条
  - 分布式批处理系统 (如 MapReduce)：大批数据作为输入，大批数据作为输出
- 计算模式
  - 分布式流数据处理 (如 Storm)：增量计算，对现有结果进行更新
  - 分布式批处理系统、全量计算 (如 MapReduce)：计算结果完全依赖于输入

Storm 应用的结构

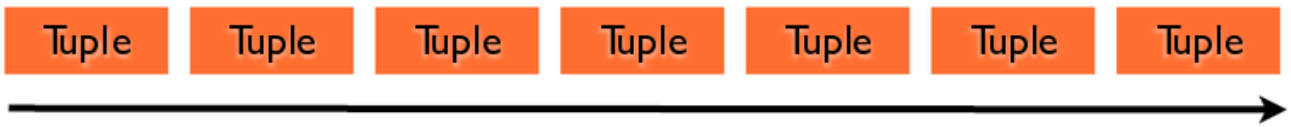
- 数据源
  - 消息队列
  - 是否支持 replay
  - TimeTunnel、Kafka、Metaq
- 处理逻辑 (Storm 应用逻辑)
  - 无状态
    - 改写
    - 过滤
  - 有状态
    - 计数
    - 合并
    - 窗口
- 结果保存
  - 消息队列
  - 分布式数据库 (NoSQL)
    - Redis、HBase

Storm 的概念



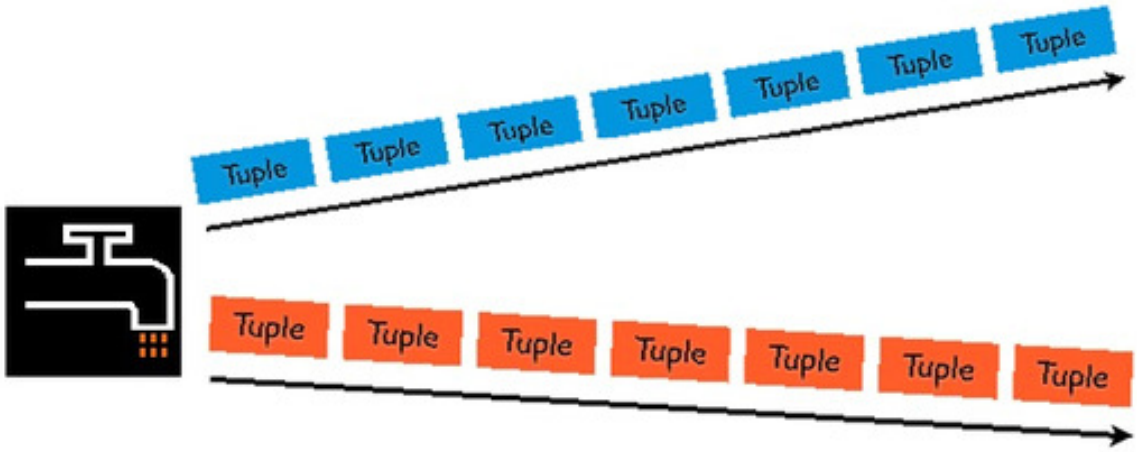
- Spout：读取输入
- Bolt：处理逻辑
- Stream：传输数据
- Topology：完整的 Storm 应用

数据流



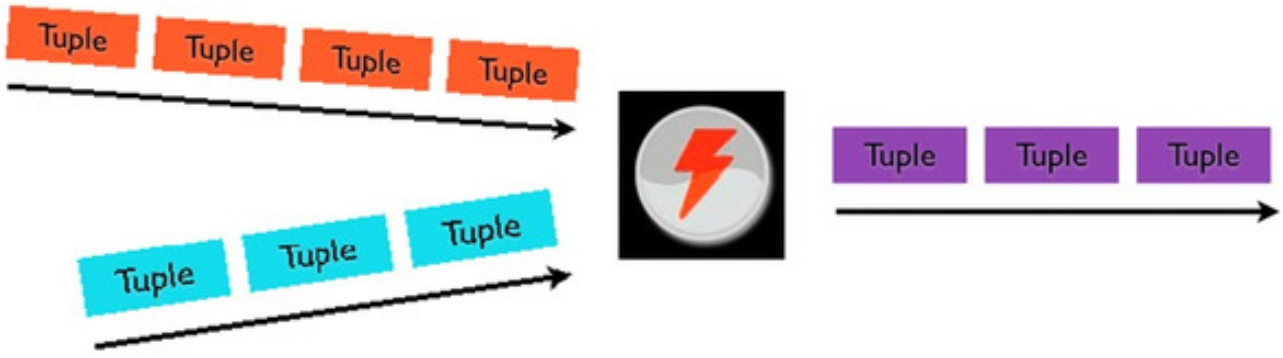
- Tuple：流上的数据
- 一个 Tuple 由多个字段 (Field) 组成

Spout



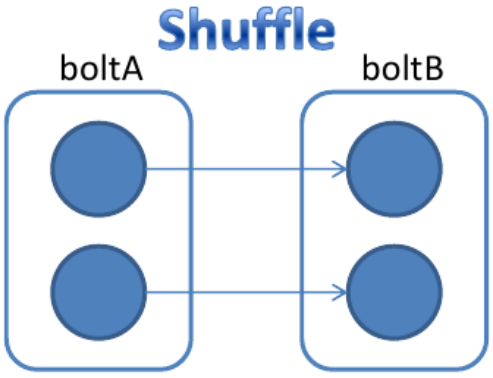
Spout 不断地从数据源读取数据，并发送至数据流

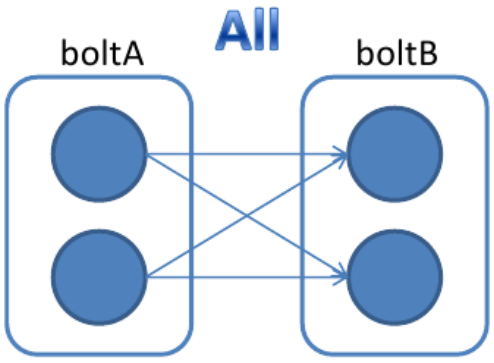
Bolt

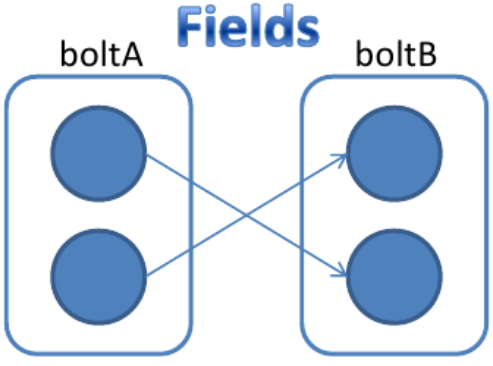


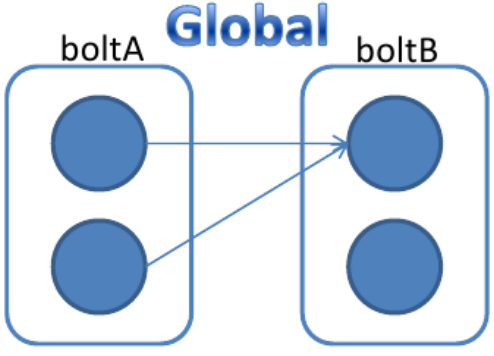
Bolt 接收到上游的一条数据后，做出相应的处理，并向下游发送新数据

消息分组

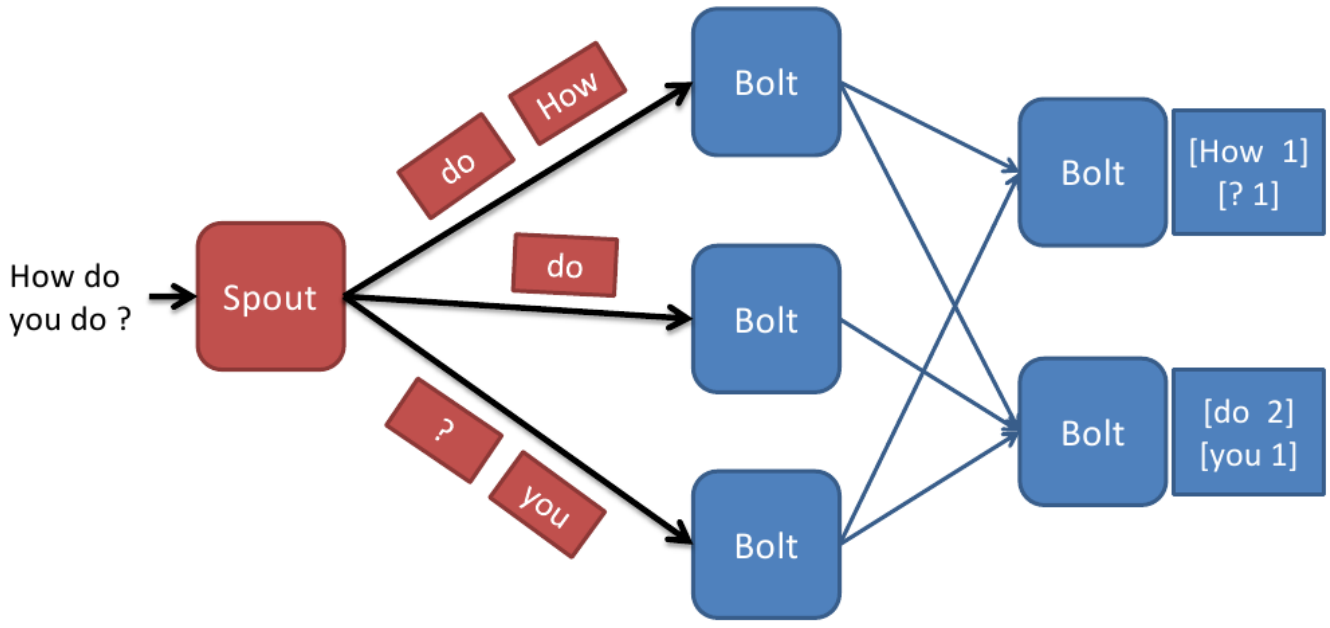

Shuffle
boltA
boltB


All
boltA
boltB

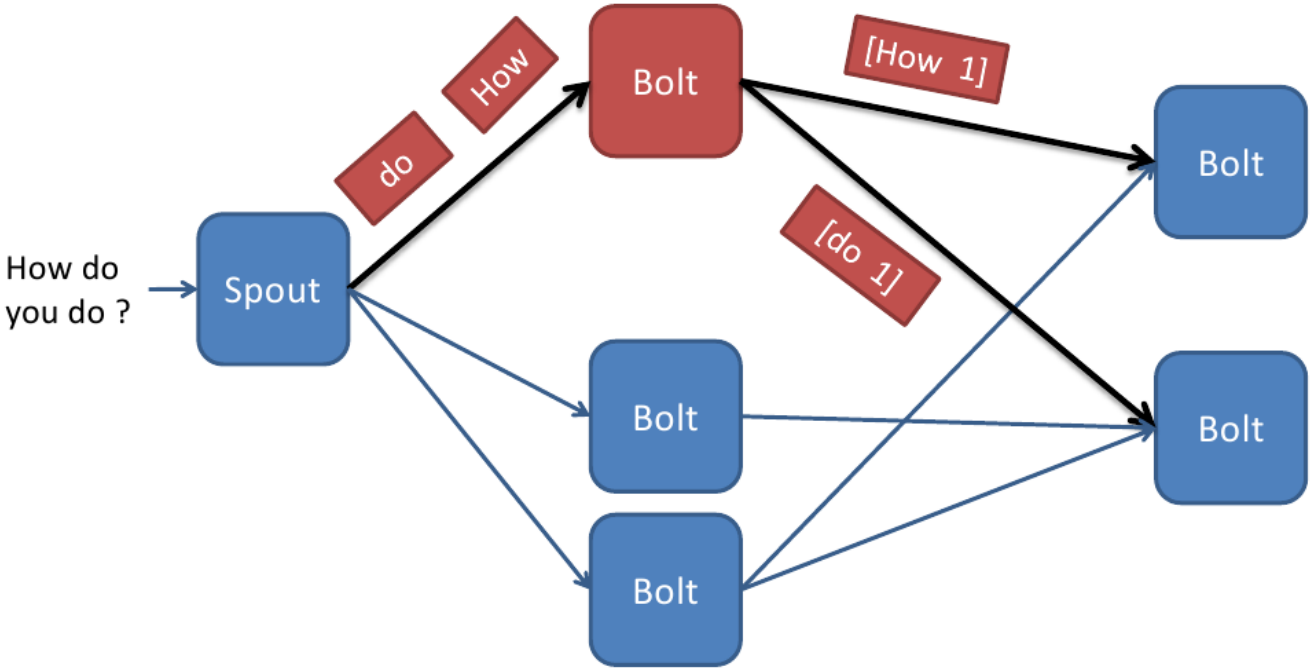

Fields
boltA
boltB


Global
boltA
boltB

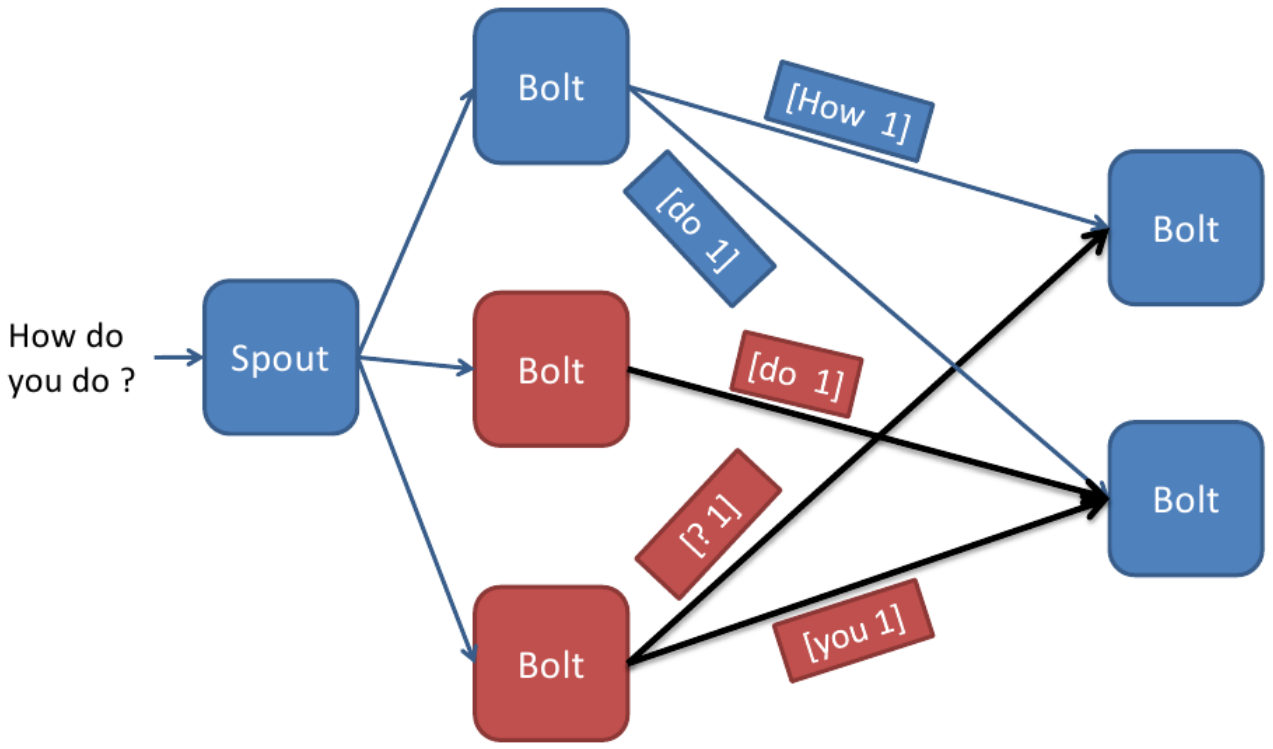
Word Count (1)


How do
you do ?
Spout
How
do
do
?
you
Bolt
Bolt
Bolt
Bolt
Bolt
[How 1]
[? 1]
[do 2]
[you 1]

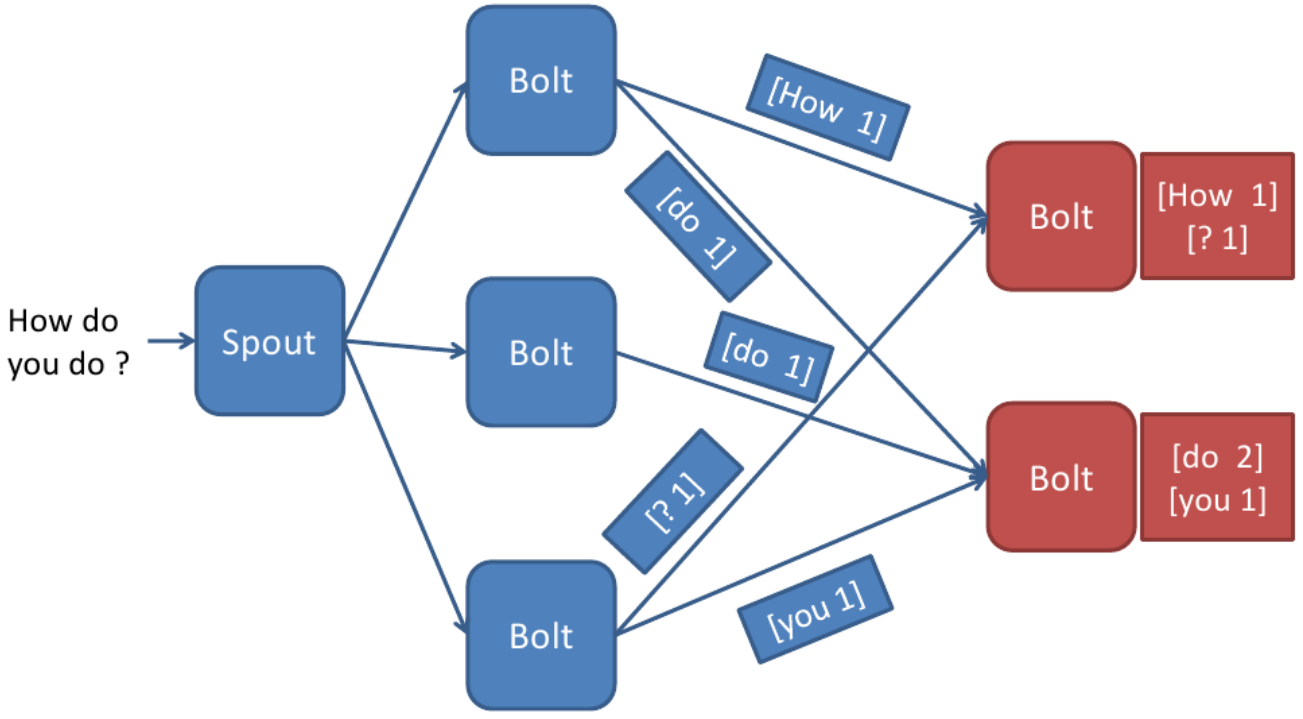
Word Count (2)


How do
you do ?
Spout
do
How
Bolt
[How 1]
[do 1]
Bolt
Bolt
Bolt
Bolt
Bolt

Word Count (3)


How do
you do ?
Spout
Bolt
Bolt
Bolt
Bolt
Bolt
Bolt
[How 1]
[do 1]
[do 1]
[? 1]
[you 1]

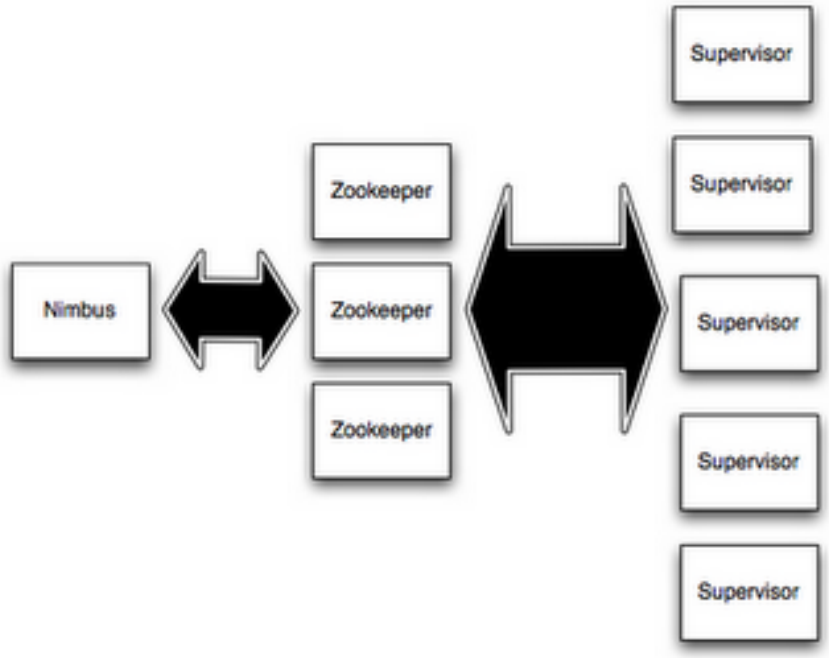
Word Count (4)


How do
you do ?
Spout
Bolt
Bolt
Bolt
[How 1]
[do 1]
[do 1]
[? 1]
[you 1]
Bolt
Bolt
[How 1]
[? 1]
[do 2]
[you 1]

容错

- Spout 或 Bolt 出错退出时，Storm 会安排在集群的其他位置重启任务
- Bolt 处理消息失败时，Spout 可以得到通知
- Upstream backup：利用上游备份容错
- 需要消息队列支持 replay

Storm 集群



- 主从结构
- Nimbus：Storm 集群的 Master，负责启动和停止任务、监控应用运行状态
- Supervisor：监听任务分配情况，启动和停止工作进程
- ZooKeeper：维护集群和应用的状态信息

实践篇

根据流数据处理的特点设计任务

- 数据的到达顺序不可控
  - 数据产生的顺序是 A, B, C
  - Spout 读到数据的顺序是 B, A, C
  - 经过多轮处理，Bolt 接收到数据的顺序是 C, A, B
  - 对时间敏感的处理尤其要小心
- 数据的到达时间不可预测
  - 两条数据流做 Join 操作
  - 第一条数据流的数据已经到达
  - 第二条数据流的数据因为上游故障，经过很长的延迟才到达
  - 影响 Join 操作的结果

状态监控

- 监控、监控、监控！
  - 上下游各个结点的流速
  - 每个环节的处理时间
  - 读写数据库的 QPS 和 RT
  - 当前处理的数据量
- 确认应用正常运行
- 便于排查失败原因
- 寻找系统瓶颈，评估性能，优化应用

Q&A

特别感谢制作过程中我的同事鸿博、蚩尤、吉剑南的帮助