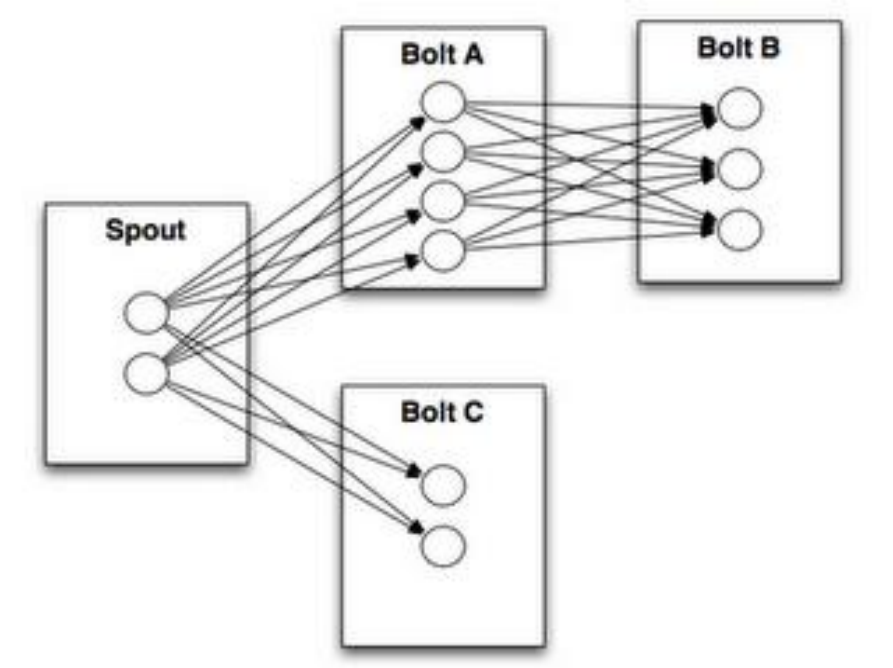
Storm中用到的技术分析

鲁国相

2012-07-16

Storm简介

- 实时的流处理框架
- 实时任务被分解为
  - Spout(数据产生者)
  - Bolt（数据处理者）



  - 一个实时任务中包含的Spout和Bolt以及它们的连接关系合称一个Topology

示例 WordCount

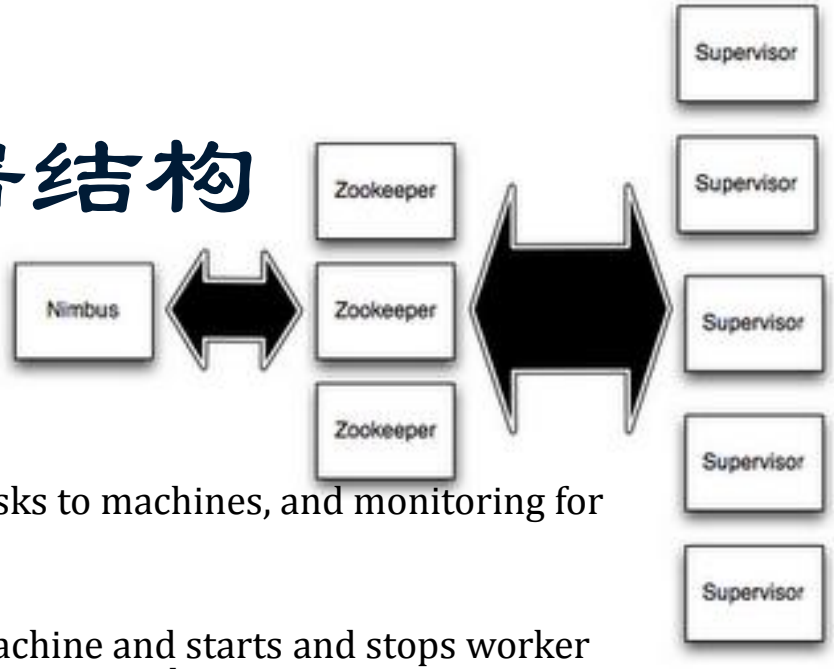
- Spout: RandomSentenceSpout 生成句子
- Bolt1: SplitSentence 将句子切分成单词并输出给下一级Bolt
- Bolt2: WordCount对相同的单词计数并输出

```
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"));
```

Storm部署结构



- 主节点运行Nimbus：
  - distributing code around the cluster, assigning tasks to machines, and monitoring for failures.
- 从节点运行Supervisor：
  - The supervisor listens for work assigned to its machine and starts and stops worker processes as necessary based on what Nimbus has assigned to it.
- 工作进程worker：
  - executes a subset of a topology; a running topology consists of many worker processes spread across many machines.
- Task
  - Worker进程中的一个线程，执行一个Spout或Bolt任务
- Slots
  - 系统总的进程数
- Nimbus 与 Supervisors
  - 通过 Zookeeper 集群交互和协调。
  - 无状态，所有状态都保存在 Zookeeper或本地磁盘。

Storm使用的技术

- 分享目标:
  - 简要介绍Storm使用的技术
  - 了解Storm如何将这些技术组合起来
  - 了解Storm的内部运行机制和构架
- Storm使用的技术
  - Zookeeper
  - Java序列化
  - Thrift
  - ZeroMQ

Zookeeper-1

- Zookeeper是一个针对大型分布式系统的可靠协调系统
- 数据模型：
  - 树形层次结构，类似Unit文件系统，树结点称为znode
  - 节点内可存储少量数据（< 1M）。
  - 根节点为/，节点通过路径引用/zoo/goat, /zoo/cat.
- 操作
  - 提供创建删除znode，读取节点数据和子节点的功能
  - 提供节点内容改变和子节点增删的通知功能
  - 支持短暂节点(*EPHEMERAL*)，创建节点的进程退出后节点自动删除。

Zookeeper-2

- 可配制zookeeper集群保证数据的可靠性，每台机器上都存储一份数据。主节点挂掉后会重新选举新的主节点。
- Storm中的zookeeper
  - 存储supervisor和worker的心跳（包括它们的状态），判断它们是否死亡
  - 存储集群状态和配置
  - Nimbus将分配给supervisor的任务写入Zookeeper

Java序列化

- 将对象保存为二进制文件（PB只序列化数据）
- 可以在另一个java进程中恢复
- 不能很好的解决版本变化
- Storm应用场景：
  - 提交任务后，将Topology序列化并发送给Nimbus，
  - Supervisor，从Zookeeper取得任务后从Nimbus下载序列化文件和jar包，启动worker进程并反序列化得到提交任务时生成Topology对象。

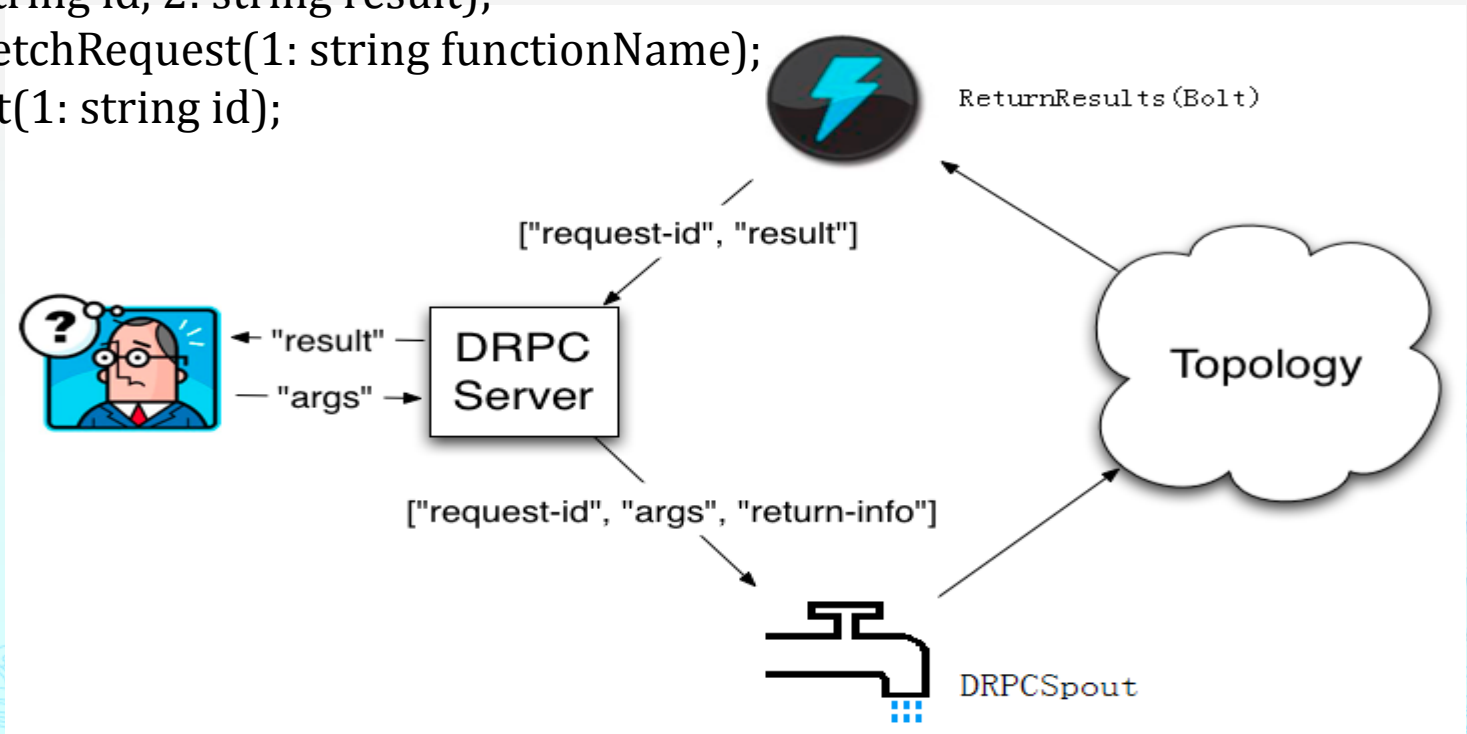
Thrift服务框架

- 一个跨语言的服务部署框架
- 通过一个中间语言(IDL, 接口定义语言)来定义RPC的接口和数据类型，然后通过一个编译器生成服务框架代码
- Storm中的应用
  - 客户端向Nimbus提交Topology的服务
  - Supervisor从Nimbus下载Topology任务（代码和序列化文件）
  - UI从Nimbus获取Topology统计信息

Thrift + Storm实现DRPC

```
service DistributedRPC {
  string execute(1: string functionName, 2: string funcArgs)
throws (1: DRPCExecutionException e);
}

service DistributedRPCInvocations {
  void result(1: string id, 2: string result);
  DRPCRequest fetchRequest(1: string functionName);
  void failRequest(1: string id);
}
```



ZeroMQ

- 嵌入式网络编程库，可作为并发框架连接多个应用程序
- N-to-N的连接，多种模式
- 支持多种语言
- You could throw thousands of clients at one server, all at once.
- Storm应用：
  - Spout,Bolt之间Tuple的传递

Put It All Together

→ Storm

Storm部署流程

- 启动Nimbus
  - storm nimbus
- 启动Supervisor
  - storm supervisor
- 提交任务
  - storm jar xxx.jar com.xxx.XXX topology-name

Nimbus启动流程

- 读取配置信息，启动Thrift服务
- 周期检查supervisor和worker的心跳
- 清理死掉的Topology信息

Supervisor启动流程

- 周期的读取zookeeper中分配的任务，找到分配给自己的任务
- Kill -9 那些死掉任务的worker，删除下载的文件
- 周期的向ZK的/supervisors目录写心跳信息

Supervisor

- 周期的读取zookeeper中分配的任务，找到分配给自己的任务
- Kill -9 那些死掉任务的worker，删除下载的文件
- 周期的向ZK的/supervisors目录写心跳信息

客户端提交Topology

- 调用Nimbus的Thrift RPC，上传topologyjar包和序列化文件
- 调用Nimbus的Thrift RPC的submitTopology方法

Nimbus服务响应submitTopology

- 收到submitTopology请求后，将jar文件、序列化文件、配置文件写入本地磁盘
- 分配Topology任务给supervisor,并保存到ZK的/assignments/[storm-id]下,包括topology代码、序列化文件，被分配给谁等等。

Supervisor检测到分配的任务

- 根据ZK中的信息调用Nimbus的Thrift RPC下载topology jar包和序列化文件
- 启动一个worker(启动一个新java进程，将下载的topology jar包加入classpath)。

Worker

- 在分配的端口上建立ZeroMQ服务(Bolt，ZMQ.*PULL)*
- 建立向下游Bolt的ZeroMQ连接（ZMQ.PUSH,需要不断更新连接）
- 向ZK写心跳
- 对每一个task(及其接收task)建立一个接收队列（*LinkedBlockingQueue*）
  - 建立一个线程从ZeroMQ接收数据并放入对应task的接受队列
- 建立一个传输队列（LinkedBlockingQueue）,用于存储需要从这个进程发送到ZeroMQ的tuple（以及这个tuple要发往的task）
  - 建立一个线程用于发送传输队列中的tuple
- 为每一个Spout或Bolt(Task)任务启动一个线程执行其代码
  - Spout调用nextTuple生成tuple放入传输队列
  - Bolt从task对应的接收队列读取tuple，调用execute,再将新emit的tuple放入传输队列
- LinkedBlockingQueue可能是引起OOM的主要原因
  - 0.8.0用更高效的Disruptor queue 替代LinkedBlockingQueue
  - http://code.google.com/p/disruptor/

Q/A