# 第一章 大数据计算平台Storm

## 概要

### 大数据背景

过去十年中，数据产生的规模如火箭般激增。现在，每秒中有30000G的数据产生，并且数据产生的速度正在加快。我们面临的数据是多种多样的。用户产生的数据包括：博客、微博、社交网络交互内容和照片等等。各大公司的服务器每天在不停地生成日志信息。互连网，拥有少量的数据，数据规模之大，超乎人们想象。

数据的快速增长对商业机构产生了极大的影响。传统的数据库系统，比如关系型数据库，已经面临瓶颈。在大数据的压力下，很多系统都面临崩溃。传统的系统，以及与之相关的数据管理技术，并不能很好地处理大数据。为了解决大数据的挑战，许多新的技术涌现出来。这些新技术，在某些层面上比传统的数据库更为复杂，在有些层面上又更为简单。它们能很好地适应大规模的数据，但是有效地利用这些产品又需要一套新的技术，并没有一劳永逸的解决方案。为了迎接大数据的挑战，我们需要从头开始重新思考数据系统。

过去几年，开源社区贡献了许多大数据相关的技术。本文所用的大数据工具基本都是开源和免费使用的。大数据相关的开源项目包括：批处理计算系统，序列化框架，随机访问非关系型数据库，消息队列系统，实时计算系统。这些开源项目已经比较成熟，有相关的公司为其提供企业级支持。比如，Cloudera为Hadoop）提供支持，DataStax为Cassandra（NoSQL数据库）提供支持。其他项目则是相关公司的产品。例如，Riak（NoSQL数据库）是Basho公司的产品，MongoDB（NoSQL数据库）属于10gen，而RabbitMQ则属于SpringSource（VMWare的子公司）。

考虑到大数据的规模，实时处理成为了许多公司要面对的挑战。现在经常用来进行大数据分析的工具是hadoop，而hadoop通常用作静态数据的大批量处理。另一个非常有效的开源实时计算工具就是storm。Storm由twitter公司开发，现在已经被许多商业机构采用。Storm通常被比作“实时的hadoop”，然而storm远比hadoop来的简单，用它来处理大数据也不会带来新老技术的交替。

大型互连网公司的日常运营会产生TB级的数据，数据的增长速度也越来越快。据了解，（素材）。数据的来源包括了互连网装置可以捕获的所有类型数据，网站、社交媒体、交易型商业数据以及其他商业环境中创建的数据。

### Storm的出现背景

过去10年中，伴随互连网的高速发展，企业产生和积累的数据越来越大，越来越多。随着Hadoop、Google MapReduce等相关技术的出现，对大数据的处理变得容易起来。但是这些系统都是针对大批量数据的处理，并不能满足实时性要求。实时计算系统和批处理系统的模型也有着本质区别。随着企业业务的快速增长，针对大规模数据的实时处理变得越来越有必要。缺少“实时的大数据计算系统”已经成为整个大数据生态系统中的巨大缺失。Storm，正是在这样的背景下出现的，storm的优越性能也能很好地应对这一需求。在storm出现之前，对于实时计算的大数据任务，开发者需要手动维护一个消息队列和消息处理者组成的实时网络。消息处理者从消息队列中取出消息进行处理，然后更新数据库，再发送消息给其他队列。所有这些操作都需要开发者自己实现，这对开发者而言是巨大的负担。开发者需要花费大量的精力去维护消息如何发送，消息发送到哪里，如何部署消息的处理者，如何部署消息处理的中间节点等。而使用storm进行开发，只需要很少的消息处理逻辑，就可快速开发一个实时计算系统。这样开发者就可以专注于业务逻辑的开发，从而提高了大数据实时系统的开发效率。

Storm是免费、开源的分布式实时计算系统，可应用在多个领域：实时分析，在线机器学习，连续计算，分布RPC（远过程调用协议，一种通过网络从远程计算机程序上请求服务），ETL（Extraction Transformation Loading，数据抽取、转换和加载），等等。Storm自2011年开源，受到众多公司青睐，许多互连网公司纷纷采用这一系统，利用storm框架构建各种各样的应用进行大数据分析，包括：twitter，淘宝，百度，爱奇艺，雅虎，奇虎360， Groupon等。引用360在storm上的报告，“360基于storm平台已经构建了大约50个实时应用，包括：网页分析，日志处理，图片处理，音频处理等等。在360中，storm的应用有些特殊，因为我们在数以千记的服务器上部署了storm，但这些服务器并不是专门用于部署storm架构的。令我们惊叹的是，在这些服务器上，storm仅仅占用了很小的cpu、内存和网络资源。利用服务器的闲置资源，storm用近乎为零的代价就提供了巨大的计算能力，并且它是实时的。简直太神奇了。”从中我们可以窥探storm的强大之处。

### Storm是什么

全量数据处理使用最多的是hadoop或者hive。Hadoo作为一个批处理系统，以吞吐量大、自动容错等优点，在海量数据处理上得到了广泛的使用。

Hadoop框架对于数据的处理流程

Hadoop是磁盘级计算，也就是说，进行计算时，数据在磁盘上，系统进行运算需要进行磁盘io操作。

相比hadoop的批处理，storm是个实时的、分布式以及具备高容错的计算系统。同hadoop一样storm也可以处理大批量的数据，然而storm处理数据更加实时，并同样保证数据计算的高可靠性。数据的实时计算保证所有的信息都会被处理。Storm还具备容错和分布计算的特性，这让storm可以部署到不同的机器上进行大批量数据处理。

### Storm与hadoop区别

Hadoop是基于磁盘进行计算，需要进行磁盘读写；storm基于内存计算，数据通过网络导入内存。Storm的网络直传、内存计算，其时延必然比hadoop通过HDFS传输低得多，当计算任务适合用流式模型时，宜选用storm的流处理。Storm的流式处理，省去了批处理的数据收集时间，同时也省去了作业调度的时延。所以从时延上来说，storm要快于hadoop。

从原理上讲，hadoop基于HDFS，需要进行切分数据、产生中间文件、排序、数据压缩、数据复制等，效率较低。Storm基于ZeroMQ这个高性能消息通讯库，不会持久化数据。

在消耗资源相同的情况下，一般来说storm的延时低于mapreduce，但是吞吐量也低于mapreduce。Storm是典型的实时流计算系统，mapreduce是典型的批处理系统。

总的来说，hadoop使用磁盘作为数据交换的媒介，storm的数据是一直在内存中流转的。两者面向的领域也不同，hadoop是面向批量处理的，吞吐量大，基于任务调度；storm是实时处理，基于数据流。以水为例，hadoop可看作纯净水，要一桶一桶地搬；而storm是水管，只要预先接好水管，用的时候打开水龙头，不就源源不断地出来了。在这个类比中，“水”就是我们需要处理的数据，“桶装纯净水”和“水管”就是数据的来源。

Hadoop MapReduce不能满足时效性，原因包括：批数据只启动一次，数据处理完之后进程停止；启动进程本身是需要时间的，输入切分、调度、起进程都需要花费时间；共享集群任务比较杂，可能需要等待资源；所有数据都需要读写磁盘。Storm的解决方案为，进程常驻运行，数据在内存中流转，刚好适合时效性需求。

### Storm与其他大数据计算平台对比

与MapReduce对比

1. 常驻运行。
2. 流式处理：数据来一点处理一点，没有数据时，程序并不终止，进行等待。
3. 实时处理：数据保存在内在中，并不读写磁盘。

与Queue+Worker系统比较

1. 维护简单：无需维护queue，queue和worker的关系。
2. 扩展容易：可以方便地增加机器，提高并发数量。
3. 自动容错：进程、机器和网络出现异常时，消息可重发。

### Storm流处理

对于处理大批量数据的map/reduce程序，任务完成之后就停止了，而storm是用于实时计算的，当数据处理任务完成后，程序并不会停止，而是会等待任务，当有新的任务时，则再次执行，这个过程不断循环，直至手动终止程序。

Storm是针对在线业务的计算平台，如统计用户的交易量、生成某个用户的推荐列表等实时性较高的需求。因此，storm是一个“流处理”框架。什么是流处理？就是说，storm将数据以stream流的方式，按照topology顺序，依次处理并最终生成计算结果。

Storm是一个分布式、实时、可靠、具有容错性的流处理系统。什么是分布式？分布式平台具有两个特征：水平扩展和自动容错。storm把工作任务委托给各个组件，每个组件负责处理其中一项任务。Storm集群的输入流由spout组件管理。数据经spout传递给bolt，bolt将数据保存起来，或者将数据传递给其他bolt。因此，storm集群就是在一连串的bolt之间转换spout传递过来的数据。

数据处理流程大致分三个阶段：

1. 数据采集与准备。目前典型的处理策略，数据一般出自页面打点和解析db的log。流计算系统将数据采集到消息队列，批处理系统一般将数据采集进分布式文件系统（比如HDFS ），有的也使用消息队列。我们暂且将消息队列和文件系统称为预处理存储，二者在延时和吞吐上没有太大差别。接下来从这个预处理存储到数据计算阶段有很大区别，流计算实时读取消息队列，将数据放入流计算系统进行相应运算。批处理系统一般人积累一大批数据批量导入到计算系统，这里就有了延时的区别。
2. 数据计算，涉及中间计算媒介，通过中间媒介导入数据。Hadoop使用磁盘作为数据交换媒介，storm通过网络将数据直接导入内存，使用内存作为媒介。Storm里程是常驻的，有数据就可以进行实时的处理。Storm每个计算单元之间通过网络直接进行数据传输，而mapreduce进行map任务后运算的结果要写入到HDFS，相对来说多了磁盘读写，比较慢。对于复杂运算，storm的运算模型直接支持DAG（有向无环图）。
3. 数据结果展现。流计算的计算结果一般直接反馈到最终的结果集中（展示页面，数据库，搜索引擎索引），而mapreduce一般需要整次运算结束后将结果批量导入到结果集中。实际上，流计算和批处理系统并没有本质的区别，像storm的trident也有批量概念，而mapreduce可以将每次运算的数据集缩小（比如每隔几分钟启动一次），facebook的puma就是基于hadoop做的流计算系统。

### storm的特性

**易于扩展**：我们希望系统是可以扩展的，因为伴随业务发展，公司运营产生的数据量会越来越大，数据的计算量相应地也越来越大。为了处理更多的数据，需要数据处理系统可以动态扩展，方便地增加机器来应对更多的数据量。Storm使用hadoop zookeeper来进行集群协调，这样可以保证集群系统的良好运行。

**实时**：每条信息都进行处理。系统的设计也保证了消息能够得到快速处理，使用zeromq（0mq）作为底层消息队列。

**基于内存**：storm是内存级计算，数据通过网络或其他途径导入内存。计算机读写内存的速度相比磁盘快几个数量级。进行磁盘的延迟约为内存访问的75000倍。数据不写磁盘，延迟低得多。

**简单的编程模型**：类似于MapReduce降低了并行批处理的复杂性，storm降低了实时处理的复杂性。storm默认支持Clojure、Java、Ruby和python，要增加对其他语言的支持，只需实现一个简单的storm通信协议即可。

**可靠的消息处理**：storm保证每个消息至少得到一次完整处理，任务失败时，会从消息源重试消息。

快速：每秒钟每个节点可处理100万个tuple。

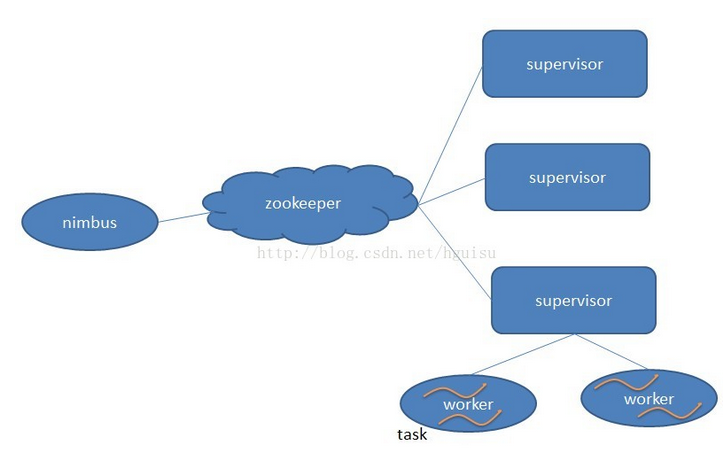
**容错机能**：storm的topology一旦进行提交，storm就会一直运行直到topology废弃或关闭。当在执行时出现错误时，storm会重新分配任务，也就是说，集群中的一个节点挂了不会影响应用进程。Storm也会管理工作进程和节点的故障。

**开源**：twitter公司将storm开源，受到开发者的欢迎。storm有非常活跃的开源社区和参与人群。

### Storm的组件

Storm集群表面上看与hadoop集群很类似，在hadoop上运行的是mapreduce任务，而storm上运行的是拓扑（topology）。这两者是很不一样的，其中一个关键区别是：mapreduce任务最终会结束，而topology会一直执行下去（除非手动停掉）。

Storm集群由一个主节点和一群工作节点组成，主节点通常为nimbus后台程序，工作节点称为supervisor，它们通过zookeeper进行集群协调。



**主节点（master node）**

主节点运行一个守护进程（daemon），Nimbus，它负责在集群中分发代码，分配工作任务并监测失败情况。Nimbus类似hadoop里的JobTracker。

**工作节点（worker node）**

工作节点同样运行一个守护进程，它运行的守护进程为Supervisor。Supervisor监听工作任务，根据要求运行工作进程。每个工作节点执行一个拓扑（topology）的子集。

**Zookeeper**

Zookeeper负责保持主节点和工作节点的协调工作。实时应用程序的逻辑则被封装进storm的拓扑（topology）。Storm中的拓扑是一组由spouts（数据源）和bolts（数据操作）通过stream groupings进行连接的图。下面对这些概念进行更加深入的分析。

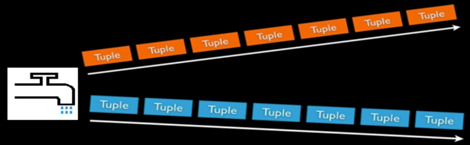
**消息流（Stream）**

消息流（stream）是storm里的关键抽象，stream是持续的tuple序列，它以分布式进行创建和处理。每个消息流在定义的时候分配一个id。在storm中，处理stream的基本原语是spout和bolt。通过实现spout和bolt提供的接口来处理业务逻辑。



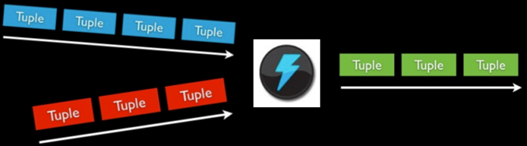
**消息源（Spout）**

简单来说，spout从数据源读取数据供storm拓扑使用，一个spout可以是可靠的，也可以是不可靠的。可靠的spout保证在数据处理失败时将元组（tuple）进行数据重发。不可靠的spout在数据发送后并不追踪元组数据。消息源可以发射多条消息流，使用OutputFieldsDeclarer的declareStream来定义多个stream，再使用SpoutOutpputCollector来发射指定的stream。在Spout中最主要的方法为nextTuple()，这个方法向拓扑发送一个新的元组数据或者在没有数据时直接返回。另外有两个比较重要的方法是ack和fail。当检测到一个tuple被整个topology成功处理的时候调用ack，否则调用fail。Storm只对可靠的spout调用ack和fail。



**Bolt**

Bolt负责storm拓扑中所有的处理过程。Bolt可以做任何事情，比如：连接过滤，数据聚合，与文件、数据库通讯，等等。Bolt从spout接收数据并处理，也可能在遇到复杂流处理时进一步将元组数据发送给另外一个Bolt。Bolt中最主要的方法是execute()，它接收一个元组（tuple）作为输入。不管是在spout还是在bolt中，将一个元组发送给多个流，都可以在declareStream()中指定这些流。



**Stream grouping**

流分组（stream grouping）定义了流在bolt任务中是如何切分的。Storm提供了内置的流分组方法，包括：随机分组（shuffle grouping）、字段分组（fields grouping）、全部分组（all grouping）、全局分组（global grouping）、无分组（none grouping）、直接分组（direct grouping）。还可以实现CustomStreamGrouping接口来定制自己需要的分组。

Storm在底层使用了zeromq（0mq），这是一种先进的、可嵌入的网络通讯库。Zeromq具有以下特性

并发架构的socket库

速度快，对于集群产品和超级计算，比TCP要快。

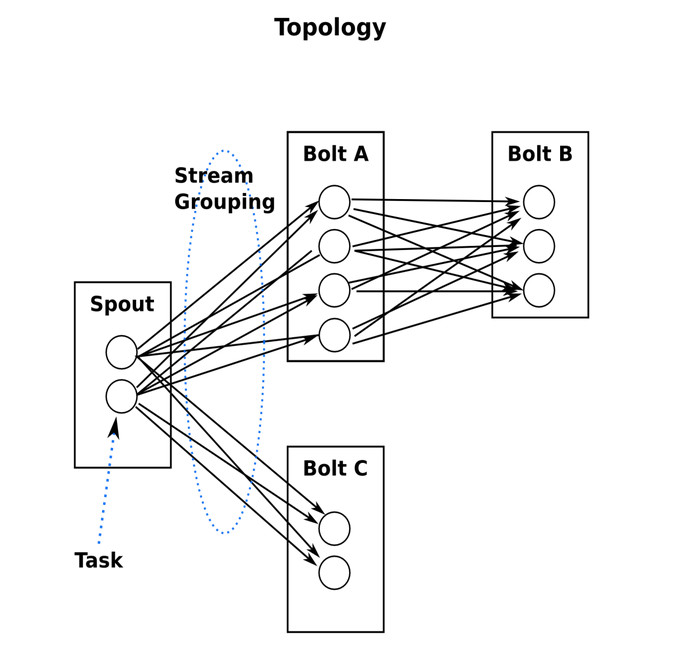
可通过进程内（inproc），进程间（IPC），TCP和多播协议（multicast）进行通信。

异步IO，可扩展，多核消息传递应用程序。

可利用扇出（fanout），发布订阅（pub-sub），管道（pipeline），请求应答（req-rep）等方式实现N-N连接。

**Topology**

Topology是由spouts和bolts组成的图，通过stream groupings将spouts和bolts连接起来，如下图所示：



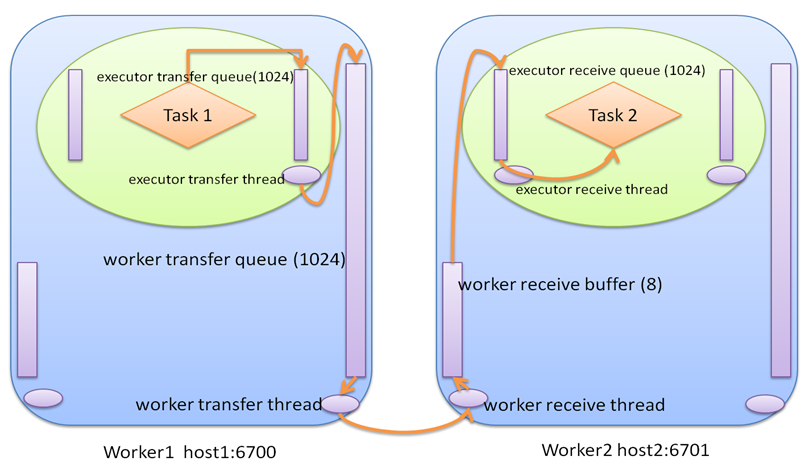
Topology一直保持运行直到手动停掉。Storm自动重新分配执行失败的任务，并且保证不会有数据丢失。如果有的机器意外停机，它所执行的任务会自动被转移到其他机器上。

如何运行一个topology呢？很简单，首先，把所有的代码及依赖打进一个jar包，然后运行类似下面的命令：

Storm jar test.jar backtype.storm.MyTopology arg1 arg2 arg3

这个命令会运行主类：backtype.storm.MyTopology，参数是arg1，arg2，arg3。这个类的main函数定义了这个topology并把它提交给nimbus。Storm jar命令连接到nimbus并且上传jar包。

Storm数据流程



Executor执行spout.nextTuple()或bolt.execute()，调用emit生成新的tuple，放到executor的transfer queue。Executor transfer thread把自己的transfer queue里面的tuple放到worker transfer queue。Worker transfer thread把transfer queue里面的tuple序列化后发送到远程的worker。Worker receive thread分别从网络接收数据，反序列化成tuple放到对应的executor的receive queue。Executor receive thread从自己的receive queue取出tuple，调用bolt.execute()。

### Storm集群的各级容错

Storm的可靠性机制，可以选择不同的可靠性级别来满足需求。storm集群中除了nimbus外，没有单点存在，任何节点都允许出故障而保证数据不丢失。nimbus被设计为无状态的，出现故障时，只要及时重启即可，不会对正在运行的任务产生影响。下面详细说明storm如何保证在各种情况下数据不被丢失。

**任务级故障**

bolt任务崩溃，消息未响应，这时，acker中所有与此bolt任务关联的消息都会因为超时而失败，相应的spout调用fail方法。

acker任务失败，在失败之前其所有消息都会因超时而失败，spout的fail方法将会被调用。

spout任务失败，在这种情况下，spout任务对接的外部设备（如MQ）负责消息的完整性，在客户端异常的情况下，kestrel队列将处理pending状态的所有消息重新放回队列。

**任务槽（slot）故障**

worker失败，每个worker包含数个bolt或spout任务，supervisor负责监控这些任务。在worker失败后，supervisor尝试重启它们。

supervisor失败，supervisor是无状态的，因此supervisor失败后不会影响当前正在执行的任务，失败后只要及时将它重启即可。supervisor失败后并不会自动发送通知，需要有外部的监控程序进行监控及时重启。

nimbus失败。nimbus也是无状态的，也不会对当前任务产生影响，但是同样需要重新启动。nimbus也是不自举的，需要外部监控来及时重启。

**集群节点故障**

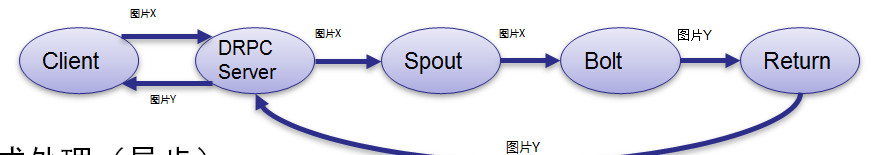
storm集群中的节点发生故障，此时nimbus会将此机器上所有正在运行的任务转移到其他可用的机器上运行。

zookeeper集群中的节点故障。zookeeper保证一半的机器宕机时仍可正常运行，当机器出现宕机时，及时修复故障机器即可。

### Storm应用场景

1. 请求应答（同步）

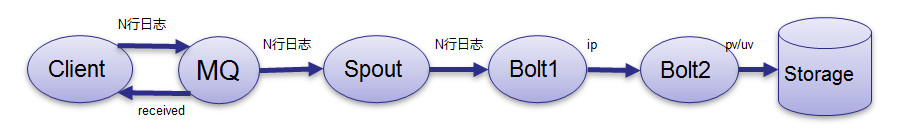
实时图片处理、实时网页分析



1. 流式处理（异步）

逐条处理：数据之间无关系：如日志按标准格式入库。

分析统计：数据之间有关系（聚合等），如日志pv/uv统计、访问热点统计。



### 实现

在本项目中，设计了一个由spout和bolt组成的拓扑，可以处理海量数据（日志文件），日志文件中包含分布式服务器集群所有进程和服务的运行状况（stat），根据这个stat日志信息，实时读取并交由storm拓扑进行分析处理，最终得到有价值的分析结果。

# 第二章 Storm集群部署

相关依赖

Java 6+

Python 2.6.6

Zookeeper 3.4.5

Apache-storm-0.9.2-incubating

集群规划

1个nimbus，3/5个zookeeper，n个supervisor

Java API

Spout

nextTuple()：回调函数，循环触发。

Ack(id)：回调函数，消息成功处理后触发。

Fail(id)：回调函数，消息超时时触发。

Bolt

Execute(Tuple input)：回调函数，数据触发。

Collector.emit(tuple)：通过collector向下游发送tuple。

Collector.ack(tuple)：通过collector确认已经成功处理输入tuple。

Storm有两种操作模式：本地模式和远程模式。在本地模式中，你完全可以在本地开发和测试拓扑。在远程模式下，你需要向集群提交并执行拓扑。远程模式不会出现调试信息，因此它也称为生产模式。在单一开发机上建立storm集群，可以在部署到生产环境之前，测试拓扑在集群环境下没有问题。

本地机器和远程集群是什么关系呢？storm集群由主节点Nimbus管理。本地机器与nimbus建立通讯提交代码（封装成jar包）和拓扑并在集群上执行。Nimbus将负责在集群中的分发代码，分配工作任务，执行拓扑。

安装storm集群

所需步骤如下

1. 搭建Zookeeper集群
2. 安装storm依赖库
3. 下载并解压storm
4. 配置storm
5. 启动storm各个后台进程

搭建Zookeeper集群

Storm使用zookeeper进行集群协调。Zookeeper并不用于消息传递，storm对zookeeper的压力相当低。在大多数情况下，单个节点的zookeeper集群可以满足要求，不过为了保证故障恢复或者部署大规模storm集群，可部署更大规模的节点（官方推荐的最小节点数为3个）。

1. 下载安装Java JDK，JDK版本为JDK 6以上。
2. 根据zookeeper集群的负载情况，合理设置java堆大小，避免发生内存置换。比如，机器内存为4GB，可分配3GB的最大堆空间。
3. 下载解压Zookeeper包。
4. 修改Zookeeper配置文件zoo.cfg

tickTime=2000

dataDir=/home/zq/data # 指定Zookeeper的数据文件路径

dataLogDir=/home/zq/zookeeper/log # 指定Zookeeper的日志文件路径

clientPort=2181 # 用于客户端进行连接的端口

initLimit=5

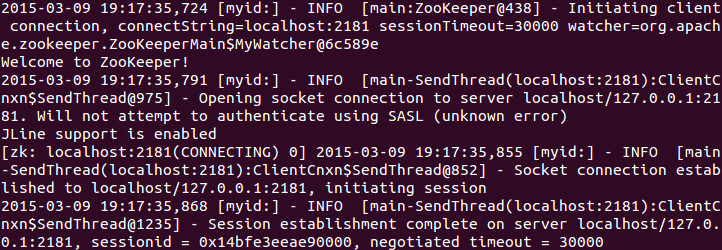
syncLimit=2

server.1=zoo1:2888:3888

server.2=zoo2.2888:3888

server.3=zoo3.2888:3888

1. 启动Zookeeper服务。执行bin/zkServer.sh start。
2. 启动Zookeeper客户端，测试服务是否可用。执行bin/zkCli.sh。Zookeeper服务成功，会输出如下信息：



安装Storm依赖库

需要在Nimbus和Supervisor机器上安装Storm的依赖库，依赖库包括：ZeroMQ、JZMQ、Java、Python、unzip。

部署zookeeper

下载解压storm发行版

storm版本更新到0.9.0.1后，安装部署也变得比较容易，与之前的版本相比，省去了zeromq和jzmq的安装，也减少了许多编译插件时出现的bug。新版本storm的一大亮点是引入了新的网络传输netty transport。当前可同时使用两种方式：原来的0mq传输和新的netty传输。0mq是一个本地库，对平台的依赖度比较高，版本差异也比较大。而netty transport是纯java方案，消除了storm本地库依赖，而且netty transport的网络传输性能比0mq快一倍以上。

修改配置文件storm.yaml，若要使用netty transport，要将相关的配置值添加到storm.yaml中。

启动storm各个后台进程

启动Nimbus后台程序：在storm主控节点上运行”bin/storm nimbus > /dev/null 2>&1 &”

启动Supervisor后台程序：在storm各个工作节点上运行”bin/storm supervisor > /dev/null 2>&1 &”

查看ui：在storm主控节点上运行”bin/storm ui > /dev/null 2>&1 &”启动UI后台程序，启动成功后可通过http://{nimbus host}:8080查看worker的资源使用情况、topologies的运行状态等信息。

Storm是快速失败的系统，storm可在任意时刻停止，当进程重启后可被正确地恢复执行。

Storm后台进程启动后，在storm安装目录下logs子目录下存放各个进程的日志文件。

要使用storm UI，UI后台程序必须和storm nimbus部署在同一台机器上，否则无法正常工作，因为UI进程会检查本地是否存在nimbus。