Alibaba Group 阿里巴巴集团 | 技术丛书

大数据丛书 | 十二五国家重点图书出版规划项目

Broadview
www.broadview.com.cn

实时

海量

高效

可靠

Storm实战

构建大数据实时计算

阿里巴巴集团数据平台事业部商家数据业务部 编著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

内 容 简 介

本书是一本系统并且具有实践指导意义的Storm工具书和参考书，对Storm整个技术体系进行了全面的讲解，不仅包括对基本概念、特性的介绍，也涵盖了一些原理说明。本书的实战性很强，各章节都提供了一些小案例，同时对于本地，以及集群环境的部署有详细介绍，易于理解，操作性强。

全书一共10章：第1章全面介绍了Storm的特性、能解决什么问题，以及和其他流计算系统的对比；第2章通过实际运行一个简单的例子，以及介绍本地环境和集群环境的搭建，让读者对Storm有了直观的认识；第3章深入讲解了Storm的基本概念，同时实现一个Topology运行；第4章和第5章阐述了Storm的并发度、可靠处理的特性；第6章~第8章详细而系统地讲解了几个高级特性：事务、DRPC和Trident；第9章以实例的方式讲解了Storm在实际业务场景中的应用；第10章总结了几个在大数据场景应用过程中遇到的经典问题，以及详细的排查过程。

本书既适合没有Storm基础的初学者系统地学习，又适合有一定Storm基础但是缺乏实践经验的读者实践和参考。

图书在版编目（CIP）数据

Storm 实战：构建大数据实时计算 / 阿里巴巴集团数据平台事业部商家数据业务部编著. —北京：电子工业出版社，2014.8
（大数据丛书. 阿里巴巴集团技术丛书）
ISBN 978-7-121-22649-6

Ⅰ. ①S… Ⅱ. ①阿… Ⅲ. ①数据处理 Ⅳ.①TP274

中国版本图书馆 CIP 数据核字(2014)第 050578 号

策划编辑：刘　皎
责任编辑：徐津平
印　　刷：三河市鑫金马印装有限公司
装　　订：三河市鑫金马印装有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路173信箱　邮编：100036
开　　本：900×1280　1/32　印张：5.75　字数：91千字
版　　次：2014年8月第1版
印　　次：2014年8月第1次印刷
印　　数：4000册　　定价：59.00元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至zlts@phei.com.cn，盗版侵权举报请发邮件至dbqq@phei.com.cn。

服务热线：（010）88258888。

第 1 章

Storm 基础

1.1 Storm 能做什么

在大数据处理方面，相信大家已经对 Hadoop 耳熟能详了，Hadoop 处理的是存放在其分布式文件系统 HDFS 上的数据，Hadoop 使用磁盘作为中间交换的介质，在对海量数据进行离线分析时得心应手，但处理实时数据流却是力有未逮。

Storm 是一个开源的分布式实时计算系统，可以简单、可靠地处理大量的数据流。Storm 有很多使用场景，如实时分析、在线机器学习、持续计算、分布式 RPC、ETL，等等。Storm 支持水平扩展，具有高容错性，保证每个消息都会得到处理，而且处理速度很快（在一个小集群中，每个节点每秒可以处理数以百万计的消息）。Storm 的部署和运维都很便捷，而且更为重要的是可以使用任意编程语言来开发应用。

1.2　Storm 特性

Storm 有如下特点。

1．编程模型简单

基于 Google Map/Reduce 来实现的 Hadoop 为开发者提供了 map、reduce 原语，使并行批处理程序变得非常简单和优美。同样，Storm 也为大数据的实时计算提供了一些简单优美的原语，这大大降低了开发并行实时处理任务的复杂性，帮助你快速、高效的开发应用。

2．可扩展

在 Storm 集群中真正运行 Topology 的主要有三个实体：工作进程、线程和任务。Storm 集群中的每台机器上都可以运行多个工作进程，每个工作进程又可创建多个线程，每个线程可以执行多个任务，任务是真正进行数据处理的实体，Spout、

Bolt 被开发出来就是作为一个或者多个任务的方式执行的。

因此，计算任务在多个线程、进程和服务器之间并行进行，支持灵活的水平扩展。

3．高可靠性

Storm 可以保证 Spout 发出的每条消息都能被“完全处理”，这也是直接区别于其他实时系统的地方，如 Yahoo！S4。

请注意，Spout 发出的消息后续可能会触发产生成千上万条消息，可以形象地理解为一棵消息树，其中 Spout 发出的消息为树根，Storm 会跟踪这棵消息树的处理情况，只有当这棵消息树中的所有消息都被处理了，Storm 才会认为 Spout 发出的这个消息已经被“完全处理”。如果这棵消息树中的任何一个消息处理失败了，或者整棵消息树在限定的时间内没有“完全处理”，那么 Spout 发出的消息就会重发。

考虑到尽可能减少对内存的消耗，Storm 并不会跟踪消息树中的每个消息，而是采用了一些特殊的策略，它把消息树当作一个整体来跟踪，对消息树中所有消息的唯一 id 进行

异或计算，通过是否为零来判定 Spout 发出的消息是否被“完全处理”，这极大地节约了内存和简化了判定逻辑，后面会对这种机制进行详细介绍。

这种模式，每发送一个消息，都会同步发送一个 ack 或 fail，对于网络的带宽会有一定的消耗，如果对于可靠性要求不高，可通过使用不同的 emit 接口关闭该模式。

上面所说的，Storm 保证了每个消息至少被处理一次，但是对于有些计算场合，会严格要求每个消息只被处理一次，幸而 Storm 的 0.7.0 版引入了事务性拓扑，解决了这个问题，本书后面会有详述。

4．高容错性

如果在消息处理过程中出现了一些异常，Storm 会重新部署这个出问题的处理单元。Storm 保证一个处理单元永远运行（除非你显式的结束这个处理单元）。

当然，如果处理单元中存储了中间状态，那么当处理单元重新被 Storm 启动时，需要应用自己处理中间状态的恢复。

5．支持多种编程语言

除了用 Java 实现 Spout 和 Bolt，你还可以使用任何你熟悉的编程语言来完成这项工作，这一切得益于 Storm 所谓的多语言协议。多语言协议是 Storm 内部的一种特殊协议，允许 Spout 或者 Bolt 使用标准输入和标准输出来进行消息传递，传递的消息为单行文本或者是 JSON 编码的多行。

Storm 支持多语言编程主要是通过 ShellBolt、ShellSpout 和 ShellProcess 这些类来实现的，这些类都实现了 IBolt 和 ISpout 接口，以及让 Shell 通过 Java 的 ProcessBuilder 类来执行脚本或者程序的协议。

可以看到，采用这种方式，每个 Tuple 在处理的时候都需要进行 JSON 的编解码，因此在吞吐量上会有较大影响。

6．支持本地模式

Storm 有一种“本地模式”,也就是在进程中模拟一个 Storm 集群的所有功能，以本地模式运行 Topology 跟在集群上运行 Topology 类似，这对于我们开发和测试来说非常有用。

7．高效

用 ZeroMQ 作为底层消息队列，保证消息能快速被处理。

8．运维和部署简单

Storm 计算任务是以“拓扑”为基本单位的，每个拓扑完成特定的业务指标，拓扑中的每个逻辑业务节点实现特定的逻辑，并通过消息相互协作。

实际部署时，仅需要根据实际情况配置逻辑节点的并发数，而不需要关心部署到集群中的哪台机器。所有部署仅需通过命令提交一个 jar 包，全自动部署。停止一个拓扑，也只需通过一个命令操作。

Storm 支持动态增加节点，新增节点自动注册到集群中，但现有运行的任务不会自动负载均衡。

9．图形化监控

图形界面，可以监控各个拓扑的信息，包括每个处理单元的状态和处理消息的数量。

1.3 其他流计算系统

这里主要将 Yahoo! S4 和 IBM InfoSphere Streams 与 Storm 进行对比。

以易用性（开发效率）、性能、通用性为基准，从系统模型、应用开发环境、系统性能、高可用性和现有代码的复用等方面进行对比。

列出对某些功能的支持与否并不表示此产品本身的优劣，首先应该考虑的是在特定场景下这些功能是否有必要。

1.3.1 Yahoo! S4

Yahoo! S4（Simple Scalable Streaming System）是一个通用的、分布式的、可扩展的、分区容错的、可插拔的流式系统，基于开源协议 Apache License 2.0。

1．系统模型

S4 系统由多个处理单元（Processing Elements，PEs）相互配合进行计算，PE 之间通过消息的形式传递，PE 消费事件，同时发出一个或多个可能被其他 PE 处理的事件，或者直接产出结果。

通过把任务分解为尽可能小的处理单元，各处理单元之间形成流水线，从而提高并发度和吞吐量。各处理单元的粒度及逻辑均由开发者自行掌握，所以并不能保证各处理单元分解得足够合理，进而影响并发性。这点与 Storm 的方式一样。

与 Storm 不同的是，S4 内置的 PE 还可以处理 count、join 和 aggregate 等常见任务需求。

2．开发

S4 系统使用 Java 开发，采用了极富层次的模块化编程，每个通用功能点都尽量抽象出来作为通用模块，而且尽可能让各模块实现可定制化。

3．通信协议

S4 节点间通信采用“Plain Old Java Objects”(POJOs) 模式，应用开发者不需要写 Schemas 或用哈希表在节点间发送消息元组 (tuples)。但底层通信协议采用 UDP，在要求一定可靠性的系统中，这点颇受诟病。

4．高可用

S4 集群中的所有处理节点都是等同的。这种架构将使得集群的扩展性很好，处理节点的总数理论上无上限；同时，S4 没有单点的问题。一旦处理单元崩溃，可以转而利用可用的其他处理单元继续处理，但和 Storm 一样，处理单元中的状态数据无法依靠系统本身恢复，需要借助外部的系统。

5．运维与部署

S4 不支持动态部署 ,也不支持动态增删节点。这点 Storm 都已支持。

1.3.2　IBM InfoSphere Streams

IBM 的商业化的实时流处理系统，有较为完善的 IDE 支持，以及自定义业务描述语言。

1．系统模型

IBM InfoSphere Streams 同样也是把任务分解为尽可能小的处理单元，各处理单元之间形成流水线，从而提高并发度和吞吐量。不同的是，各处理单元只能完成预定的操作，这些操作组合起来完成一个整体的功能。从机制上保证了处理单元的粒度，有助于系统整体性能的提升。

系统由很多 PE 节点组成，PE 仅仅实现规定的一些操作（如 join、merge 等），强制限定了每个处理单元的粒度，从而可以提高每个单元的处理速度，使得流水线更流畅。多个 PE 可能集成到一个线程中，降低了系统中线程的数量。在设计阶段无须关心 PE 与主机的对应关系，此关系在运行阶段根据集群的配置自动部署。

2．开发

定制的开发环境 Eclipse-SPL（Stream Programming Language）。为流处理定制的 SPL 语言可以简洁地描述出整个系统的业务，也可以使用其他语言（如 C++）来定制特定的流处理模块（operator）。

3．运维与部署

部署半自动化，程序设计、编译阶段无须指定主机信息。只需指定约束关系（如 PE1 与 PE2 不能再同一台主机上），在拓扑部署的时候根据集群的实际情况和预定的约束来确定 PE 在哪台主机上执行。

动态增加节点，新节点自动注册到集群中，与 Storm 不同的是，InfoSphere Streams 可以将现有业务根据负载自动均衡。而 Storm 已经运行的业务不会自动负载均衡。

4．高可用

InfoSphere Streams 与 Storm、S4 类似，可把失败节点上的

任务自动转移到其他可用节点上，但同样也不保证数据的恢复。

1.4 应用模式

1．海量数据处理

Storm 由于其高效、可靠、可扩展、易于部署、高容错及实时性高等特点，对于海量数据的实时处理非常合适。例如，对于统计网站的页面浏览量（如 Page View，简称 PV）指标，Storm 可以做到实时接收到点击数据流，并实时计算出结果。

2．中间状态存储与查询

这里的中间状态分为两种，一种指的是 Storm 处理流数据实时计算出的结果，是在实时、快速地变化着的。Storm 提供了实时处理数据流的平台，但是并未提供现成的取得实时处理结果的接口，查询这些实时处理的结果，即所谓的中间状态，就需要 Storm 与一些存储服务相结合，比如 MySQL 和 HBase。可以将 Storm 实时计算的中间结果实时地写入

MySQL 或者 HBase ,用户就能直接通过数据库的接口取得实时的结果了。

Storm 处理单元中存储的中间状态可以不单单是计算结果，还可以是计算逻辑类的快照或者“还原点”，这样有一个好处就是错误恢复。Storm 的容错机制确保了如果一个处理单元崩溃，则重启一个处理单元继续处理该数据流。这意味着 Storm 中每个处理单元的处理逻辑应该是无状态的，这样每次重启后，依然能基本正确地处理业务逻辑。但是，对于大部分计算场合，譬如 PV 这样的累计指标，如果没有中间状态，一旦重启，这个处理单元就会重新从 0 开始累计 PV，显然这样的结果是不正确的。因此，如果利用 MySQL 或者 HBase 实时存储处理的中间状态，就可以减少处理单元崩溃后的损失，从最近的中间状态恢复。

另外，Storm 错误恢复还需要数据源的配合，处理单元恢复后能够从数据源继续读取尚未处理的数据处理，或者跳过，前进一定的跨度继续处理。Storm 本身提供了与 kestrel 队列交互的 Spout 来支持这些特性。在实践中，也可以将数

据实时写入其他存储介质，如 HBase，然后由 Storm 实时读取 HBase 的数据，利用 HBase 的特性支持数据的前后定位。

3．数据增量更新

在实际业务中，PV 这种指标的计算，以 HBase 为例，最简单的想法是每次增加都实时修改 HBase。容错策略是，处理单元崩溃恢复后，继续累计 HBase 中的值，但每次计数修改 HBase 的方式，对 HBase 的压力太大。

可以考虑数据在 Storm 内计算短暂的一段时间后，增量地合并到 HBase，以牺牲一定查询结果的实时性换取 HBase 压力的减轻。容错上，每次崩溃，也只是丢失在内存中未合并到 HBase 的那部分计数，尽可能做到崩溃时数据的快速恢复和误差可控。

4．结合概率算法实时计算复杂指标

Storm 实时处理数据，相对于离线处理，需要更多的空间来存储中间状态，对于复杂的指标，可能中间状态的存储最后成为瓶颈，导致内存无法容下。业务上对于实时计算的

指标，有时并不需要完全精确，因此，可以利用一些概率算法来解决这种问题。

例如，对于网站来说访客数（Unique Visitor，UV）指标是指某一时段访问的访客数量，需要对访客的唯一标识去重并计数才能算出。这样，就需要存储从开始计数到现在所有访客的唯一标识，用以去重计算。在访问量巨大、指标交叉维度繁多时，很容易形成瓶颈。采用 Hyper LogLog 这样的概率算法，只需对 UV 指标存储一个位图信息，就能估计出 UV 值，而位图的大小取决于算法需要达到的精度，可以根据业务调节。对于所有类似去重的指标，都可以采用这样的概率算法。

第 2 章

Storm 初体验

2.1　本地环境搭建

由于 Storm 的分布式特性，用户提交的 Topology 可能会分布到多台物理机器上运行，这对 Topology 的开发、测试和调试造成了一定困难。Storm 提供本地集群机制，允许用户将 Topology 提交到本地集群，而且所有的 Bolt、Spout 都运行在一个进程内，能很方便地对 Topology 进行调试。

2.1.1　环境准备

Eclipse 的环境准备

Eclipse 需要安装 Maven 插件，即 m2e。可在 Eclipse 主界面菜单中单击 Help→About Eclipse 按钮，打开 About Eclipse 对话框，查看是否已安装 m2e 插件。

如果未安装，可通过单击 Help→Eclipse Marketplace... 按钮，找 Maven Integration for Eclipse 选项进行安装。

2.1.2 Storm jar 下载、配置

1．本地下载 jar 包

Storm 的官方主页为 http://storm-project.net/，在此处可下载最新的 jar 包。

源代码地址为 https://github.com/nathanmarz/storm，此处可获取最新代码，从源代码编译 jar 包（注：目前 Storm 已经成为 Apache 的一个孵化项目，官方的 git 仓库由 Apache 维护，github 上有一个镜像，参见链接 https://github.com/apache/incubator-storm）。

获取 jar 包后，在 Eclipse 的 Package Explorer 项目中单击鼠标右键，选择 Properties 按钮，在弹出的对话框中选择 Java Build Path 选项，再选择 Libraries 选项卡，通过 Add External JARs...选项将 Storm jar 文件加入编译路径。

2．使用 Maven

如果你的项目是 Maven 项目，可以简单地在 pom.xml

文件中添加对 Storm jar 的依赖。

推荐使用 Maven 项目进行 Storm 开发。

在你的 pom.xml 中添加以下代码：

```
<repository>
     <id>clojars.org</id>
     <url>http://clojars.org/repo</url>
</repository>
<dependency>
     <groupid>storm</groupid>
     <artifactid>storm</artifactid>
     <version>0.7.2</version>
</dependency>
```

2.1.3　运行一个简单的实例

运行一个本地集群实例很简单，只需要把 Topology 提交给 backtype.storm.LocalCluster 类的对象就可以了。用户还可以像调试普通 Java 程序一样在本地集群中调试 Blot 和 Spout。

storm-starter 项目包含一系列简单的 Storm 实例代码，下面介绍如何利用 storm-starter 来运行本地集群，下文假设读者使用 Linux、Eclipse 和 Maven。

首先，获取 storm-starter 代码，在命令行运行：

```
$ git clone https://github.com/nathanmarz/ storm-starter.git
```

在命令行运行经典的 WordCount 程序：

```
$ cd storm-starter
$ mvn -f m2-pom.xml compile exec:java -Dstorm.topology= \
> storm.starter.WordCountTopology
```

将 storm-starter 项目代码导入到 Eclipse IDE 中：

```
$ cd storm-starter
$ git checkout 0.7.0
$ mv m2-pom.xml pom.xml
```

打开 Eclipse，单击菜单中 File→Import 选项，选择获取的 storm-starter 目录。

一直单击 Next 按钮，直到最后单击 Finish 按钮，完成导入。

找到 src/jvm/storm/WordCountTopology.java，右键单击 Run As→Java Application 选项，即可以在 Eclipse 中运行 WordCount 实例。

注意，在运行 storm-starter 的过程中可能会遇到以下问题。

1）twitter4j-core 和 twitter4j-stream 这两个包无法下载。

这是由于网络原因导致 twitter Maven 仓库无法使用。

解决方法：从 Maven 中央仓库手动下载这两个包，注释掉 storm-starter/pom.xml 的<repositories>...</repositories>，然后再试。

2)storm 和 clojure 这两个包的下载速度太慢，难以等待。

这是由于 http://clojars.org/repo 这个仓库太慢，国内访问困难，导致 Maven 难以下载。

解决方案如下。

（1）从官方下载完整的 storm-0.7.0，地址 https://www.dropbox.com/s/pfz2xsy3om6g9eo/storm-0.7.0.zip。

（2）解压 storm-0.7.0.zip 到 storm-starter 目录下。

（3）将 storm-starter/pom.xml 中 clojure、clojure-contrib、storm 这 3 个包的依赖注释掉。

（4）手动将 storm-0.7.0 的所有 jar 包加入 Eclipse。在 Package Explorer 视图中找到 storm-starter 工程，右键单击 Properties→Java Build Path→Libraries→Add Library 选项，单击 User Library→Next→User Libraries→New 按钮新建一个 Library，起名为 storm070，然后选择 storm070，单击 Add External JARs... 按钮，将 storm-starter/storm-0.7.0/lib 和 storm-starter/storm-0.7.0/下的所有 jar 文件加入其中。

3）可以运行，但产生名为“when launching multilang subprocess”的异常。

这是由于 storm-starter/multilang/resuources 未导入工程，或者 Python 版本过高导致。

解决方案如下。

（1）命令行执行 python –version，查看 Python 是否为 3.x 版本，如果是，需要安装 Python 2.x 版本替换。

（2）检查 storm-starter/multilang/resuources 是否导入工程，如果没有，需要手动 import 此文件夹到工程中。

最后，如果正确运行了 storm.starter. WordCountTopology，将会有类似于如下代码的输出：

```
8370 [Thread-30] INFO backtype.storm.daemon.task - Emitting: split default ["am"]
8370 [Thread-19] INFO backtype.storm.daemon.task - Emitting: count default [i, 50]
8370 [Thread-30] INFO backtype.storm.daemon.task - Emitting: split default ["at"]
8370 [Thread-17] INFO backtype.storm.daemon.task - Emitting: count default [am, 49]
8370 [Thread-28] INFO backtype.storm.daemon.task - Emitting: split default ["at"]
8370 [Thread-17] INFO backtype.storm.daemon.task - Emitting: count default [am, 50]
8370 [Thread-30] INFO backtype.storm.daemon.task - Emitting: split default ["two"]
8370 [Thread-19] INFO backtype.storm.daemon.task - Emitting: count default [at, 49]
8370 [Thread-28] INFO backtype.storm.daemon.task - Emitting: split default ["two"]
8371 [Thread-19] INFO backtype.storm.daemon.task - Emitting: count default [at, 50]
8371 [Thread-30] INFO backtype.storm.daemon.task - Emitting: split default ["with"]
8371 [Thread-21] INFO backtype.storm.daemon.task - Emitting: count default [two, 49]
8371 [Thread-28] INFO backtype.storm.daemon.task - Emitting: split default ["with"]
```

运行大概会持续 10 秒钟，然后退出。storm-starter 可以帮助我们学习使用 Storm 和利用本地集群模式进行调试。

2.1.4　本地集群原理

本地集群模式通过多线程来运行 Topology，把 Spout 和 Blot 集中到一个进程的多个线程中执行，来模拟真实的运行情况。

在本地集群模式中，许多耗时的工作都使用 Thread.sleep 方法模拟，有时会导致运行速度过慢，这点需要注意。

2.2 Storm 集群

2.2.1 Storm 集群组件

Storm 集群中包含两类节点：主控节点（Master Node）和工作节点（Worker Node）。它们对应的角色如下。

（1）主控节点上运行一个被称为 Nimbus 的后台程序，它负责在 Storm 集群内分发代码，分配任务给工作机器，并负责监控集群运行状态。Nimbus 的作用类似于 Hadoop 中 JobTracker 的角色。

（2）每个工作节点上运行一个被称为 Supervisor 的后台程序。Supervisor 负责监听从 Nimbus 分配给它执行的任务，据此启动或停止执行任务的工作进程。每一个工作进程执行一个 Topology 的子集，一个运行中的 Topology 由分布在不同工作节点上的多个工作进程组成。

Nimbus 和 Supervisor 节点之间所有的协调工作是通过 Zookeeper 集群来实现的，如图 2-1 所示。

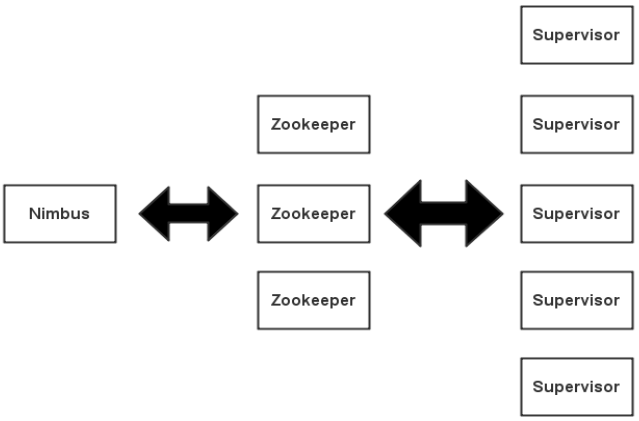

图 2-1

此外，Nimbus 和 Supervisor 进程都是快速失败(fail-fast) 和无状态（ stateless ）的；Storm 集群中所有的状态要么在 Zookeeper 集群中，要么存储在本地磁盘上。这意味着你可以用 kill -9 来结束 Nimbus 和 Supervisor 进程，它们在重启后可以继续工作。这个设计使得 Storm 集群拥有不可思议的稳定性。

2.2.2 安装 Storm 集群

本节将详细描述如何搭建一个 Storm 集群。下面是需要依次完成的安装步骤。

- 搭建 Zookeeper 集群。
- 安装 Storm 依赖库。
- 下载并解压 Storm 发布版本。
- 修改 storm.yaml 配置文件。
- 启动 Storm 的各个后台进程。

1．搭建 Zookeeper 集群

Storm 使用 Zookeeper 协调集群，由于 Zookeeper 并不用于消息传递，所以 Storm 给 Zookeeper 带来的压力相当小。大多数情况下，单个节点的 Zookeeper 集群足够胜任，不过为了确保故障恢复或者部署大规模 Storm 集群，可能需要更大规模节点的 Zookeeper 集群（对于 Zookeeper 集群，官方

推荐的最小节点数为 3 个）。在 Zookeeper 集群的每台机器上完成以下安装部署步骤。

（1）下载安装 Java JDK，官方下载链接为 http://java.sun.com/javase/downloads/index.jsp，JDK 版本为 JDK 6 或以上。

（2）根据 Zookeeper 集群的负载情况，合理设置 Java 堆的大小，尽可能避免发生 swap，导致 Zookeeper 性能下降。保守起见，4GB 内存的机器可以为 Zookeeper 分配 3GB 最大堆空间。

（3）下载后解压安装 Zookeeper 包，官方下载链接为 http://hadoop.apache.org/zookeeper/releases.html。

（4）根据 Zookeeper 集群节点的情况，在 conf 目录下创建 Zookeeper 配置文件 zoo.cfg，代码如下。

```
tickTime=2000
dataDir=/var/zookeeper/
clientPort=2181
initLimit=5
syncLimit=2
server.1=zoo1:2888:3888
server.2=zoo2:2888:3888
server.3=zoo3:2888:3888
```

其中，dataDir 指定 Zookeeper 的数据文件目录，server.id=host:port:port，id 是每个 Zookeeper 节点的编号，保存在 dataDir 目录下的 myid 文件中，zoo1~zoo3 表示各个 Zookeeper 节点的 hostname，第一个 port 是用于连接 leader 的端口，第二个 port 是用于 leader 选举的端口。

（1）在 dataDir 目录下创建 myid 文件，文件中只包含一行，且内容为该节点对应的 server.id 中的 id 编号。

（2）启动 Zookeeper 服务。

```
$ java -cp zookeeper.jar:lib/log4j-1.2.15.jar:conf \
> org.apache.zookeeper.server.quorum.QuorumPeerMain zoo.cfg
```

或者

```
$ bin/zkServer.sh start
```

（3）通过 Zookeeper 客户端测试服务是否可用。

```
$ java -cp zookeeper.jar:src/java/lib/log4j-1.2.15.jar:conf:src/ \
> java/lib/jline-0.9.94.jar \
> org.apache.zookeeper.ZooKeeperMain -server 127.0.0.1:2181
```

或者

```
$ bin/zkCli.sh -server 127.0.0.1:2181
```

注意事项如下。

（1）由于 Zookeeper 是快速失败的，且遇到任何错误情况，进程均会退出，因此，最好能通过监控程序将 Zookeeper 管理起来，保证 Zookeeper 退出后能被自动重启[1]。

（2）Zookeeper 运行过程中会在 dataDir 目录下生成很多日志文件和快照文件，而 Zookeeper 进程并不负责定期清理或合并这些文件，导致占用大量磁盘空间，因此，需要通过 cron 等方式定期清除没用的日志和快照文件[2]。具体命令格式如下。

```
$ java -cp zookeeper.jar:log4j.jar:conf org.apache.zookeeper \
> .server.PurgeTxnLog <dataDir> <snapDir> -n <count>
```

1 具体方法请参考 http://zookeeper.apache. org/doc/r3.3.3/ zookeeperAdmin. html#sc_supervision

2 具体方法请参考 http://zookeeper.apache.org/ doc/r3.3.3/zookeeperAdmin. html#sc_maintenance

2．安装 Storm 依赖库

接下来，需要在 Nimbus 和 Supervisor 机器上安装 Storm 的依赖库，具体步骤如下。

（1）安装 ZeroMQ 2.1.7。请勿使用 ZeroMQ 2.1.10 版本，因为该版本的一些严重 BUG 会导致 Storm 集群运行时出现奇怪的问题。少数用户在 ZeroMQ 2.1.7 版本会遇到"IllegalArgumentException"的异常，此时将 ZeroMQ 降为 2.1.4 版本可修复这一问题。

（2）安装 JZMQ。

（3）安装 Java 6。

（4）安装 Python 2.6.6。

（5）安装 unzip。

以上依赖库的版本是经过 Storm 测试的，Storm 并不能保证在其他版本的 Java 或 Python 库下可运行。

（1）安装 ZeroMQ 2.1.7。

下载后编译安装 ZeroMQ：

```
$ wget http://download.zeromq.org/zeromq-2.1.7.tar.gz
$ tar -xzf zeromq-2.1.7.tar.gz
$ cd zeromq-2.1.7
$ ./configure
$ make
$ sudo make install
```

注意事项　如果安装过程报错 uuid 找不到，则通过如下方法安装 uuid 库：

```
$ sudo yum install e2fsprogsl  -b current
$ sudo yum install e2fsprogs-devel  -b current
```

（2）安装 JZMQ。

下载后编译安装 JZMQ：

```
$ git clone https://github.com/nathanmarz/jzmq.git
$ cd jzmq
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
```

为了保证 JZMQ 正常工作，可能需要完成以下配置。

① 正确设置 JAVA_HOME 环境变量

② 安装 Java 开发包

③ 升级 autoconf

④ 如果你用的是 Mac OS X 系统，参考本页脚注1的链接

注意事项　如果运行./configure 命令出现问题，参考本页脚注2的链接。

（3）安装 Java 6。

① 下载并安装 JDK 6，参考本页脚注3的链接

② 配置 JAVA_HOME 环境变量

③ 运行 java、javac 命令，测试 Java 正常安装

（4）安装 Python 2.6.6。

① 下载 Python 2.6.6

1 http://cbcg.net/blog/2011/07/30/getting-zeromq-and-jzmq-running-on-mac-os-x/

2 http://stackoverflow.com/questions/3522248/how-do-i-compile-jzmq-for-zeromq-on-osx

3 http://www.oracle.com/technetwork/java/javase/index-137561.html #linux

```
$ wget http://www.python.org/ftp/python/2.6.6/Python-2.6.6.tar.bz2
```

② 编译安装 Python 2.6.6

```
$ tar -jxvf Python-2.6.6.tar.bz2
$ cd Python-2.6.6
$ ./configure
$ make
$ make install
```

③ 测试 Python 2.6.6

```
$ python -V
Python 2.6.6
```

（5）安装 unzip。

① 如果使用 RedHat 系列的 Linux 系统，执行以下命令安装 unzip：

```
$ yum install unzip
```

② 如果使用 Debian 系列的 Linux 系统，执行以下命令安装 unzip：

```
$ apt-get install unzip
```

3．下载并解压 Storm 发布版本

下一步，需要在 Nimbus 和 Supervisor 机器上安装 Storm 发行版本。

（1）下载 Storm 发行版本，推荐使用 Storm 0.8.1。

```
$ wget https://github.com/downloads/nathanmarz/storm /storm-0.8.1.zip
```

（2）解压到安装目录下。

```
$ unzip storm-0.8.1.zip
```

4．修改 storm.yaml 配置文件

Storm 发行版本解压后，目录下有一个 conf/storm.yaml 文件，用于配置 Storm。默认配置可以在 https://github.com/nathanmarz/storm/blob/master/conf/defaults.yaml 中的配置选项覆盖 defaults.yaml 中的默认配置。以下配置选项是必须在 conf/storm.yaml 中进行配置的。

（1）**storm.zookeeper.servers**：Storm 集群使用的 Zookeeper 集群地址，其格式如下。

```
storm.zookeeper.servers:
  - "111.222.333.444"
  - "555.666.777.888"
```

如果 Zookeeper 集群使用的不是默认端口，那么还需要 storm.zookeeper.port 选项。

（2）**storm.local.dir**：Nimbus 和 Supervisor 进程用于存储少量状态，如 jars、confs 等本地磁盘目录，需要提前创建该目录并给予足够的访问权限。然后在 storm.yaml 中配置该目录，如：

```
storm.local.dir: "/home/demo/storm/workdir"
```

（3）**java.library.path**：Storm 使用的本地库（ZMQ 和 JZMQ)加载路径 ,默认为"/usr/local/lib:/opt/local/lib:/usr/lib"，一般来说 ZMQ 和 JZMQ 默认安装在/usr/local/lib 下，因此不需要配置即可。

（4）**nimbus.host**：Storm 集群 Nimbus 机器地址，各个 Supervisor 工作节点需要知道哪个机器是 Nimbus，以便下载 Topologies 的 jars、confs 等文件，如：

```
nimbus.host: "111.222.333.444"
```

（5）**supervisor.slots.ports**：对于每个 Supervisor 工作节点，需要配置该工作节点可以运行的 Worker 数量。每个 Worker 占用一个单独的端口用于接收消息，该配置选项用于定义哪些端口是可被 Worker 使用的。默认情况下，每个节点上可运行 4 个 Workers，分别在 6700、6701、6702 和 6703 端口，如：

```
supervisor.slots.ports:
    - 6700
    - 6701
    - 6702
    - 6703
```

5．启动 Storm 各个后台进程

最后一步，启动 Storm 的所有后台进程。和 Zookeeper 一样，Storm 也是快速失败的系统，这样 Storm 才能在任意时刻被停止，并且当进程重启后被正确地恢复执行。这也是为什么 Storm 不在进程内保存状态的原因，即使 Nimbus 或 Supervisors 被重启，运行中的 Topologies 也不会受到影响。

以下是启动 Storm 各个后台进程的方式。

（1）**Nimbus**：在 Storm 主控节点上运行"bin/storm nimbus >/dev/null 2>&1 &"，启动 Nimbus 后台程序，并放到后台执行。

（2）**Supervisor**：在 Storm 各个工作节点上运行"bin/storm supervisor >/dev/null 2>&1 &"，启动 Supervisor 后台程序，并放到后台执行。

（3）**UI**：在 Storm 主控节点上运行"bin/storm ui >/dev/null 2>&1 &"，启动 UI 后台程序，并放到后台执行，启动后可以通过 http://{nimbus host}:8080 观察集群的 Worker 资源使用情况、Topologies 的运行状态等信息。

注意事项如下。

（1）启动 Storm 后台进程时，需要对 conf/storm.yaml 配置文件中设置的 storm.local.dir 目录具有写权限。

（2）Storm 后台进程被启动后，将在 Storm 安装部署目录下的 logs/子目录下生成各个进程的日志文件。

（3）经测试，Storm UI 必须和 Storm Nimbus 部署在同一台机器上，否则 UI 无法正常工作，因为 UI 进程会检查本机是否存在 Nimbus 链接。

（4）为了方便使用，可以将 bin/storm 加入系统环境变量中。

至此，Storm 集群已经部署、配置完毕，可以向集群提交 Topology 运行。

2.2.3 向集群提交任务

1．启动 Storm Topology

```
$ storm jar allmycode.jar org.me.MyTopology arg1 arg2 arg3
```

其中，allmycode.jar 是包含 Topology 实现代码的 jar 包，org.me.MyTopology 的 main 方法是 Topology 的入口，arg1、arg2 和 arg3 为 org.me.MyTopology 执行时需要传入的参数。

2．停止 Storm Topology

```
$ storm kill {toponame}
```

其中，{toponame}为 Topology 提交到 Storm 集群时指定的 Topology 任务名称。

2.2.4　本章参考资料

[1] https://github.com/nathanmarz/storm/wiki/Tutorial

[2] https://github.com/nathanmarz/storm/wiki/Setting-up-a-Storm-cluster