银河麒麟服务器操作系统 V4

storm 软件适配手册

天津麒麟信息技术有限公司

2019 年 5 月

目 录

1 概述

1.1 系统概述

银河麒麟服务器操作系统主要面向军队综合电子信息系统、金融系统以及电力系统等国家关键行业的服务器应用领域，突出高安全性、高可用性、高效数据处理、虚拟化等关键技术优势，针对关键业务构建的丰富高效、安全可靠的功能特性，兼容适配长城、联想、浪潮、华为、曙光等国内主流厂商的服务器整机产品，以及达梦、金仓、神通、南大通用等主要国产数据库和中创、金蝶、东方通等国产中间件，满足虚拟化、云计算和大数据时代，服务器业务对操作系统在性能、安全性及可扩展性等方面的需求，是一款具有高安全、高可用、高可靠、高性能的自主可控服务器操作系统。

1.2 环境概述

服务器型号	长城信安擎天 DF720 服务器
CPU 类型	飞腾 2000+处理器
操作系统版本	Kylin-4.0.2-server-sp2-2000-19050910.Z1
内核版本	4.4.131
storm 版本	1.0.1

1.3 Storm 软件简介

Apache Storm 是 Twitter 开源的实时数据处理框架，是一个免费开源、分布式、高容错的实时计算系统。Storm 令持续不断的流计算变得容易，弥补了 Hadoop 批处理所不能满足的实时要求。Storm 经常用于在实时分析、在线机器学习、持续计算、分布式远程调用和 ETL 等领域。Storm 的部署管理非常简单，而且在同类的流式计算工具，Storm 的性能也是非常出众的。

1.4 Storm 基本概念

storm 中服务器节点分为主节点和从节点，Nimbus 为主节点，Supervisor 为从节点，以及若干组件构成。下面为对一些术语进行简单的介绍:

- **Nimbus**：主节点，是一个调度中心，负责分发任务 ；
- **Supervisor**：从节点，任务执行的地方 ；
- **Worker**：任务工作进程，一个 Supervisor 中可以有多个 Worker。；
- **Executor**：Worker 进程在执行任务时，会启动多个 Executor 线程 ；
- **Topology**：任务的抽象概念。由于 storm 是流式计算的框架，它的数据流和拓扑图很像，所以它的任务就叫 topology；

- **Spout**：从数据源获取数据并进行分发；
- **Bolt**：得到 Spout 或者上一个 Bolt 的数据,然后进行处理后交给下一个 Bolt 处理；
- **Tuple**：在 storm 中，一条数据可以理解为是一个 Tuple。

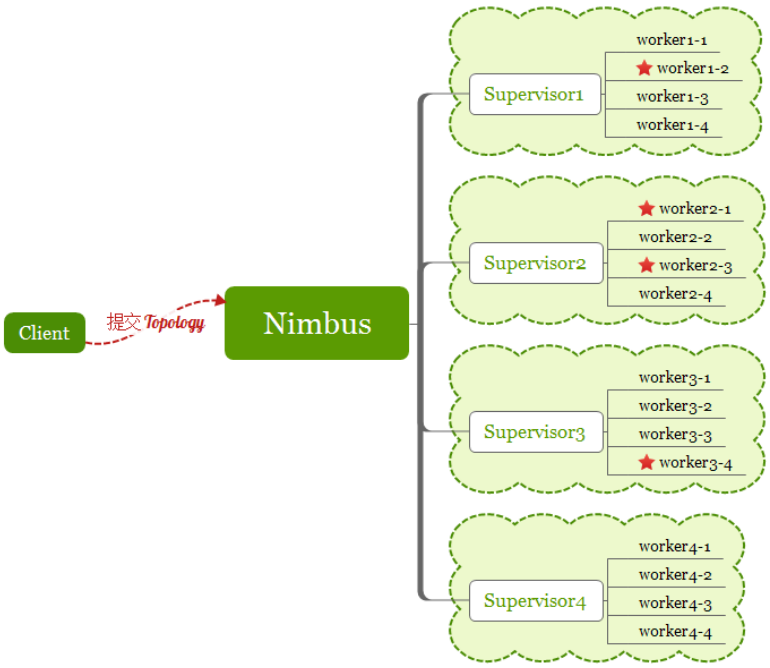
1.5 Storm 架构

1.5.1 任务提交处理流程

Nimbus 是调度中心，Supervisor 是任务执行的地方。Supervisor 上面有若干个 Worker，每个 Worker 都有自己的端口，Worker 可以理解为一个进程。另外，每个 Worker 中还可以运行若干个线程。

当客户端向 storm 集群提交一个 Topology 时，这里的提交就是在集群上通过命令 storm jar xxx 启动 topology。如果我们是在 Supervisor 节点上执行 storm jar xxx，那么 Supervisor 会将 jar 包拷贝到 Nimbus，之后 Nimbus 对 Topology 进行调度。Nimbus 会根据 Topology 所需要的 Worker 进行分配，将其分配到各个 Supervisor 的节点上执行。

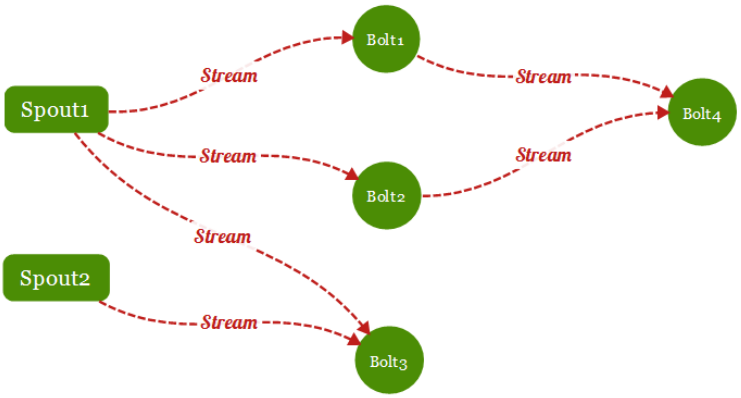
现在假设有 4 个 Supervisor 节点，每个 Supervisor 都配置 4 个 Worker。这时提交一个 Topology，需要 4 个 Worker，那可能的分配情况可能如下图所示：



1.5.2 Storm 中的数据流

启动完 Topology 后,相关组件就开始运行起来了。在 Storm 中，Spout 组件主要用来从数据源拉取数据，形成一个 Tuple 后转交给 Bolt 处理。Bolt 接收到 Tuple 处理完后，可以选择继续交给下一个 Bolt 处理，也可以选择不往下传。这样数据

以 Tuple 的形式一个接一个的往下执行，就形成了一个拓扑数据流。storm 数据在组件间的流向如下图所示：



1.5.3 消息的可靠性保证

一条数据在 spout 中形成一个 tuple，然后交给一个个 bolt 执行，那我们怎么保证这个 tuple 被完整的执行了呢？这里的完整执行说的是这个 tuple 必须在后面的每一个 bolt 都成功处理，假设在一个 bolt 中发生异常导致失败，这就不能算完整处理。

为了保证消息处理过程中的可靠性，storm 使用了 ack 机制。storm 会专门启动若干 acker 线程，来追踪 tuple 的处理过程，acker 线程数量可以设置。

每一个 tuple 在 spout 中生成的时候，都会分配到一个 64 位的 messageId。通过对 messageId 进行哈希我们可以通过向 acker 线程发送消息来通知它监听这个 tuple。

acker 线程收到消息后，会将发出消息的 spout 和 messageId 绑定起来。然后开始跟踪该 tuple 的处理流程。如果这个 tuple 全部处理完，那么 acker 线程就会调用发起这个 tuple 的那个 spout 实例的 ack()方法。如果超过一定时间这个 tuple 还没处理完，那么 acker 线程就会调用对应 spout 的 fail()方法，通知 spout 消息处理失败，spout 组件就可以重新发送这个 tuple。

从上面的介绍我们知道了，tuple 数据的流向会形成一个拓扑图，也可以理解成是一个 tuple 树。这个拓扑图的节点可能会有很多个，如果要把这些节点全部保存起来，处理大量的数据时势必会造成内存溢出。对于这个难题，storm 使用了一种非常巧妙的方法，使用 20 个字节就可以追踪一个 tuple 是否被完整的执行。这也是 storm 的一个突破性的技术。

2 Storm 软件适配

2.1 下载 storm 软件

$	wget

```
https://archive.apache.org/dist/storm/apache-storm-1.0.1/apache-storm-1.0.1.tar.gz
$ tar -zxvf apache-storm-1.0.1.tar.gz -C /usr/local/
```

2.2 修改环境变量

```
$ vim /etc/profile
```

在文件的最后添加：

```
export STORM_HOME=/usr/local/apache-storm-1.0.1
export
PATH=$JAVA_HOME/bin:$ZOOKEEPER_HOME/bin:$MAVEN_HOME/bin:$STOR
M_HOME/bin:$KAFKA_HOME/bin:$PATH
```

```
$ vim /etc/profile
```

2.3 修改配置文件

```
$ cd /usr/local/apache-storm-1.0.1/conf/
$ vim storm.yaml
```

修改 storm.zookeeper.servers 的值为如下：

```
storm.zookeeper.servers:
    - "localhost"
```

并添加如下内容：

```
nimbus.host: "localhost"
storm.zookeeper.port: 2181
storm.local.dir: "/usr/local/apache-storm-1.0.1/data"
supervisor.slots.ports:
    - 6700
    - 6701
    - 6702
    - 6703
```

2.4 启动 storm

```
$ storm nimbus &
$ storm supervisor &
$ storm ui &
```

3 Storm 测试用例及使用场景举例

3.1 运行 storm 自带测试用例

```
$ cd /usr/local/apache-storm-1.0.1
$ ./bin/storm jar examples/storm-starter/storm-starter-topologies-1.0.1.jar org.apache.storm.starter.WordCountTopology test
```

3.2 Storm 使用场景举例

我们在这里演示 kafka+storm 的常见使用场景，对单字段字符串进行匹配过滤。其中，kafka 负责生成和消费数据，storm 使用 kafkaspout 从指定的 kafka 主题中将数据取下来，对数据进行 Avro 反序列化，然后对数据进行过滤匹配(filterbolt)，最后将匹配的结果生产到 kafka 中（kafkabolt）。

3.2.1 运行方法

首先在 Kafka 的 bin 文件夹下输入命令：

```
$ ./kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 50 --topic apptest
```

创建 topic，然后在 storm 的 bin 文件夹下执行：

```
$ ./storm jar /media/testconsumer.jar com.Testconsumer.kafka.Myconsumer 1 2 4 1
```

提交 storm 前 topic 中应存有足够消费 10 分钟的数据。

3.2.2 运行结果

在 storm/bin 目录下输入：

```
$ ./storm jar /media/testconsumer.jar com.Testconsumer.kafka.Myconsumer 1 2 4 1
```