传统关系型数据库是先存后计算，而storm则是先算后存，甚至不存. 和同样是计算框架的MapReduce相比，MapReduce集群上运行的是Job，而Storm集群上运行的是Topology。但是Job在运行结束之后会自行结束，Topology却只能被手动的kill掉，否则会一直运行下去. storm使用zookeeper来协调整个集群， 但是要注意的是storm并不用zookeeper来传递消息。所以zookeeper上的负载是非常低的.

Spout:

继承了BaseRichSpout后，不用实现close、 activate、 deactivate、 ack、 fail 和 getComponentConfiguration 方法，只关心最基本核心的部分。  
通常情况下（Shell和事务型的除外），实现一个Spout，可以直接实现接口IRichSpout，如果不想写多余的代码，可以直接继承BaseRichSpout.

Bolt：

通常情况下，实现一个Bolt，可以实现IRichBolt接口或继承BaseRichBolt，如果不想自己处理结果反馈，可以实现 IBasicBolt接口或继承BaseBasicBolt，它实际上相当于自动实现了collector.emit.ack(inputTuple)

Topology运行之前，会加载本地的 storm.yaml 配置文件。

**Storm任务分配机制分析**

一、Storm的任务分配流程及算法如下：

1、 先由nimbus来计算拓扑的工作量，及计算多少个task，task的数目是指spout和bolt的并发度的分别的和，例如一个拓扑中有一个spout和一个bolt，并且spout的并发度为2，bolt的并发度为3，则task数为5；

2、 nimbus会把计算好的工作分配给supervisor去做，工作分配的单位是task，即把计算好的一堆task分配给supervisor去做，即将task-id映射到supervisor-id+port上去，具体分配算法如下：

(1)从zk上获得已有的assignment(新的toplogy当然没有了）  
(2)查找所有可用的slot，所谓slot就是可用的worker，在所有supervisor上配置的多个worker的端口。  
(3)将任务均匀地分配给可用的worker

3、 Supervisor会根据nimbus分配给他的任务信息来让自己的worker做具体的工作

4、 Worker会到zookeeper上去查找给他分配了哪些task，并且根据这些task-id来找到相应的spout/bolt，它还需要计算出这些spout/bolt会给哪些task发送消息，然后建立与这些task的连接，然后在需要发消息的时候就可以给相应的task发消息。

二、Nimbus的任务分配算法特点如下：

1、 在slot充沛的情况下，能够保证所有topology的task被均匀的分配到整个机器的所有机器上

2、 在slot不足的情况下，它会把topology的所有的task分配到仅有的slot上去，这时候其实不是理想状态，所以在nimbus发现有多余slot的时候，它会重新分配topology的task分配到空余的slot上去以达到理想状态。

3、 在没有slot的时候，它什么也不做

三、可插拔的任务分配器

Storm在0.8.0之后引入了可插拔式的任务分配器，使得Storm的任务分配更加灵活。

1、 默认调度器

Storm默认的调度实现是DefaultScheduler.clj;Storm调度可以通过storm.sheduler配置，如果不配置，则使用backtype.storm.sheduler. DefaultScheduler.clj；上述的Nimbus的任务分配算法使用的就是该默认调度器；

2、 调度隔离机制

在Storm的0.8.2之后加入了隔离调度机制，它使得一些拓扑中分享集群变得简单和安全。隔离调度程序允许指定哪些拓扑应该孤立，这意味着它们运行在集群中的一组专用的机器上，这些专用的机器上将没有其他拓扑运行。一旦所有隔离拓扑分配完，剩余的机器将在所有非隔离的机器中共享集群拓扑。

它通过backtype.storm.sheduler.IsolationScheduler来实现，即将storm.sheduler配置为backtype.storm.sheduler.IsolationScheduler，如下：

storm.sheduler：backtype.storm.sheduler.IsolationScheduler

使用isolation.scheduler.machines配置确定多少机器在一个拓扑中，该配置是一个拓扑名称映射到机器的数量，例如：

isolation.scheduler.machines:

“app1”: 5

“app2”: 3

“app3”: 2

3、 自定义调度器

Storm的这种实现方式实现了可插拔的任务分配器，我们也可以写一个自己的任务调度器来实现我们的特定的场景以及需求，例如：如果想将某个spout或者bolt分配到固定的机器上去；或者想将两个很耗CPU的拓扑分开来，即不让他们运行在同一机器上等。要实现一个我们自己的Scheduler只需要实现IScheduler接口即可

四、我们对Storm分配机制的丰富

Storm自身的分配机制会尽量保证一个Topology会被平均分配到当前集群上，但是它没有考虑整个集群的负载均衡；例如现在集群有三台机器（三台Supervisor），每个上面的可用Slot数目均为四个，那么现在提交Topology，并且Topology占用1个worker，提交多个Topology后，它会先将整个集群中的一个机器占满，然后再去给别的机器分配。这种分配方式对有些场景是不太适用的，因此我们对Storm自身的分配机制增加了额外的一个配置；

配置项如下：

default.schedule.mode: "average"

如果default.schedule.mode配置为average，则在使用默认的分配机制时会优先将任务分配给空闲Slot数目最多的机器。

**Storm的高可用：**

关于Storm的高可用，有以下几个方面：

　　(1)**数据利用阶段可以通过ACK机制保证数据被处理**；

　　(2)**在进程级别，worker失效，supervisor会自动重启worker线程**;

　　(3)**在组件级别，supervisor节点失效，会在其他节点重启该supervisor任务**；

　　但是一个很大的问题，nimbus节点失效怎么办？

Supervisor进程和Nimbus进程，需要用Daemon程序如monit来启动，失效时自动重新启动。  
因为它们在进程内都不保存状态，状态都保存在本地文件和ZooKeeper，因此进程可以随便杀。

如果Nimbus进程所在的机器都直接倒了，需要在其他机器上重新启动，Storm目前没有自建支持，需要自己写脚本实现。  
即使Nimbus进程不在了，也只是不能部署新任务，有节点失效时不能重新分配而已，不影响已有的线程。  
同样，如果Supervisor进程失效，不影响已存在的Worker进程。

Zookeeper本身已经是按至少三台部署的HA架构了。