Hadoop 手册

数据服务中心 - 大数据应用

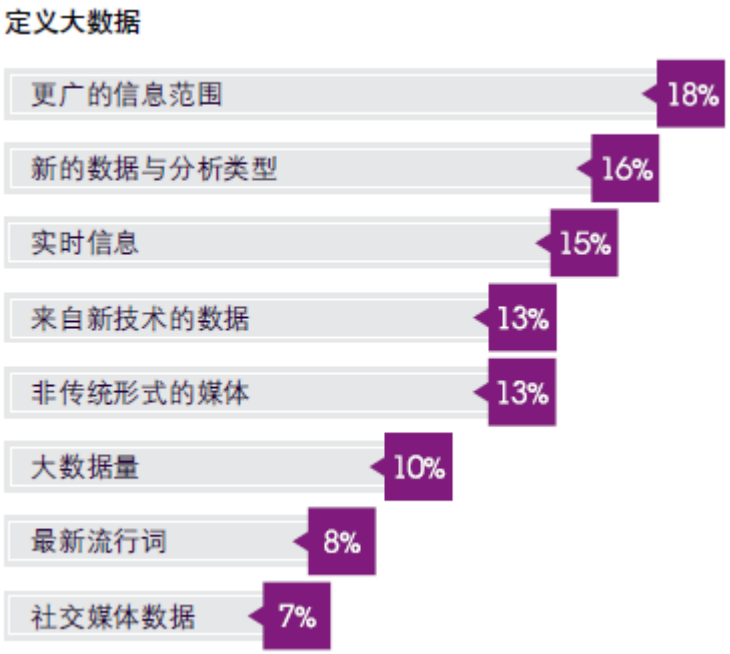
2013-3-5

海南易建科技股份有限公司

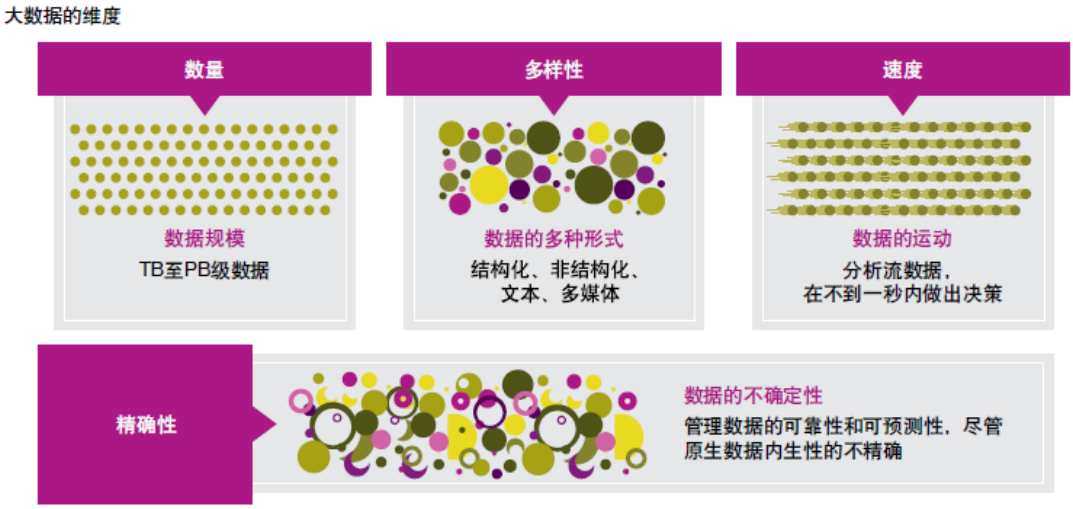
李景帆 整理

1. 大数据理论

大数据是什么？IBM 在 2012 年中期对 95 个国家中 26 个行业的 1144 名专业人员的进行的 Big Data 研究。受访者来自多个学科，包括业务专业人员(占总样本的 54%)和 IT 专业人员(占总样本的 46%)。受访者对大数据的理解如下图描述：



这些结果与确定大数据的三个维度的有用方式相一致 - 即“3V”：数量(volume)、多样性(variety)和速度(velocity)。尽管这些涵盖大数据本身的关键属性，但 IBM 认为，企业需要考虑一个重要的第四维度：精确性(veracity)。将精确性作为大数据的第四个属性凸显了应对与管理某些类型数据中固有的不确定性的重要性。



在国内业内讨论中认为，精确性和价值(value)相关，而价值(value)的产生才是大数据体系存在的意义。

涵盖这四个维度有助于定义和区分大数据：

数量：数据量。数量也许是与大数据最相关的特征，指企业为了改进企业中的决策而试图利用的大量数据。数据量持续以前所未有的速度增加。

多样性：不同类型的数据和数据源。多样性是指管理多种数据类型的复杂性，包括结构化、半结构化和非结构化数据。企业需要整合并分析来自复杂的传统和非传统信息源的数据，包括企业内部和外部的数据。随着传感器、智能设备和社会协同技术的爆炸性增长，数据的类型无以计数，包括：文本、微博、传感器数据、音频、视频、点击流、日志文件等。

速度：数据在运动中。数据创建、处理和分析的速度持续在加快。加速的原因是数据创建的实时性天性，以及需要将流数据结合到业务流程和决策过程中的要求。速度影响数据延时 – 从数据创建或获取到数据可以访问的时间差。

精确性：数据不确定性。精确性指与某些数据类型相关的可靠性。追求高数据质量是一项重要的大数据要求和挑战，高质量的数据也间接影响了数据的价值。但是，即使最优秀的数据清理方法也无法消除某些数据固有的不可预测性，例如天气、经济或者客户最终的购买决定等。

大数据是这些特征的组合，为企业在当前的数字化市场中创造竞争优势提供了机会。它使企业能够转变与客户交互并满足客户需求的方式，并且使企业，甚至整个行业能够实现自身的转型。利用新的大数据技术和分析方法改进决策和绩效的机会存在于每个行业中。

2. Hadoop 的背景及知识体系

2.1 Hadoop 介绍

Hadoop 作为 Apache 基金会资助的开源项目，由 Doug Cutting 带领的团队进行开发，基于 Lucene 和 Nutch 等开源项目，实现了 Google 的 GFS 和 Hadoop 能够稳定运行在 20 个节点的集群；2006 年 1 月，Doug Cutting 加入雅虎公司，同年 2 月 Apache Hadoop 项目正式支持 HDFS 和 MapReduce 的独立开发。同时，新兴公司 Cloudera 为 Hadoop 提供了商业支持，帮助企业实现标准化安装，并志愿贡献社区。

2008 年 2 月，雅虎宣布搭建出世界上最大的基于 Hadoop 的集群系统—Yahoo! Search Webmap，另外还被广泛应用到雅虎的日志分析、广告计算、科研实验中；Amazon 的搜索门户 A9.com 中的商品搜索的索引生成就是基于 Hadoop 完成的；互联网电台和音乐社区网站 Last.fm 使用 Hadoop 集群运行日志分析、A/B 测试评价、AdHoc 处理和图表生成等日常作业；著名 SNS 网站 Facebook 用 Hadoop 构建了整个网站的数据仓库，进行网站的日志分析和数据挖掘。UC Berkeley 等著名高校也对 Hadoop 进行应用和研究，以提高其整体性能，包括 Matei Zaharia 等人改进了 Hadoop 的推测式执行技术并发表了 Improving MapReduce Performance in Heterogeneous Environment；Tyson Condie 等人改进了 MapReduce 体系，允许数据在操作之间用管道传送，开发了 Hadoop Online Prototype（HOP）系统，并发表了 MapReduce Online。

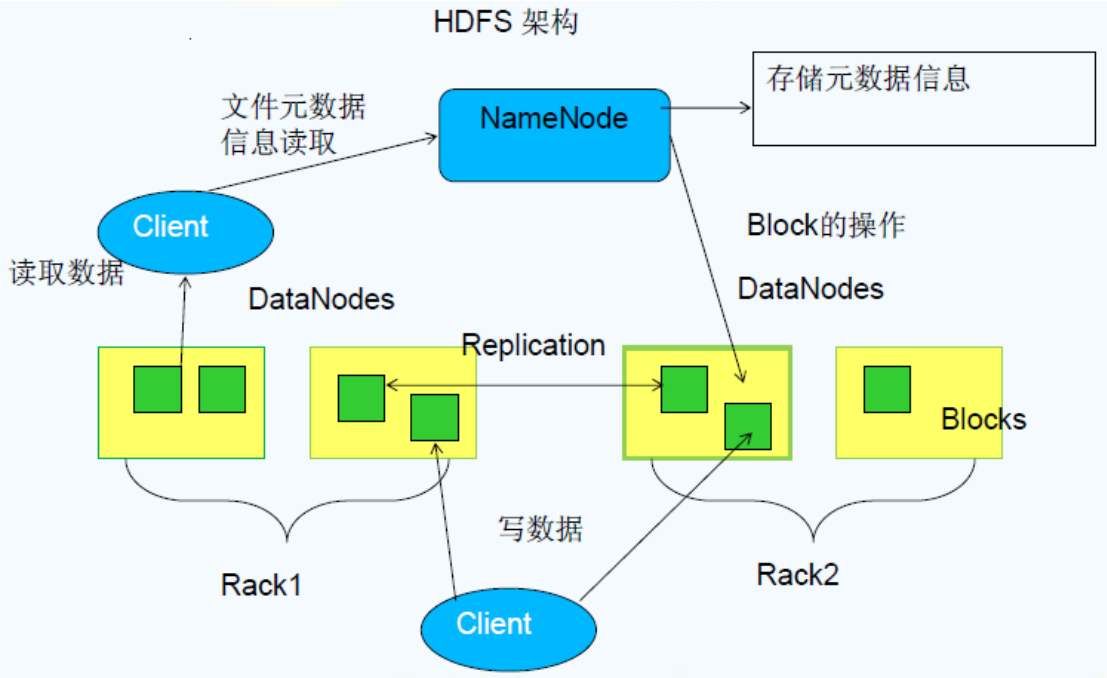
2008 年之后，国内应用和研究 Hadoop 的企业也越来越多，包括淘宝、百度、腾讯、网易、金山等。淘宝是国内最先使用 Hadoop 的公司之一；百度在 Hadoop 上进行广泛应用并对它进行改进和调整，同时赞助了 HyperTable 的开发。总之，互联网企业是 Hadoop 在国内的主要使用力量。同样的，很多科研院所也投入到 Hadoop 的应用和研究中，包括中科院、清华大学、浙江大学和华中科技大学等。

Hadoop 有存储和计算能力方面的优势，在 2012 年的 11 月份，Fackbook 在 Hadoop 的体系下存储了 100PB 的字节。并且是原 Terasort 记录的保持者。Terasort 给出 1TB 的随机数据，Hadoop 用了 1406 台主机（5624 块硬盘），用时 62 秒完成了排序。

2.2 Hadoop 的基础原理

Hadoop 由 HDFS、MapReduce、HBase、Hive 和 ZooKeeper 等成员组成。其中，HDFS 和 MapReduce 是两个最基础最重要的成员。

HDFS 是 Google GFS 的开源版本，一个高度容错的分布式文件系统，它能够提供高吞吐量的数据访问，适合存储海量（PB 级）的大文件（通常超过 64M），其原理如下图所示：



NameNode	DataNode	SecondaryNameNode
存储元数据	存储文件内容	将NameNode的fsimage与edit log从NameNode 复制到临时目录
元数据保存在内存中与磁盘上	文件内容保存在磁盘	将fsimage同edit log合并并产生新的fsimage
保存文件，block dataNode之间的映射关系	维护block id 到datanode本地文件的映射关系	将产生的新的fsimage上传给NameNode 清除NameNode中的edit log

HDFS 支持传统的层次型文件组织结构。用户或者应用程序可以创建目录，然后将文件保存在这些目录里。文件系统名字空间的层次结构和大多数现有的文件系统类似：用户可以创建、删除、移动或重命名文件。

2.2.1 Namenode 和 Datanode

HDFS 采用 master/slave 架构。一个 HDFS 集群是由一个 Namenode 和一定数目的 Datanodes 组成。Namenode 是一个中心服务器，负责管理文件系统的名字空间(namespace)以及客户端对文件的访问。集群中的 Datanode 一般是一个节点一个，负责管理它所在节点

上的存储。HDFS 暴露了文件系统的名字空间，用户能够以文件的形式在上面存储数据。从内部看，一个文件其实被分成一个或多个数据块，这些块存储在一组 Datanode 上。Namenode 执行文件系统的名字空间操作，比如打开、关闭、重命名文件或目录。它也负责确定数据块到具体 Datanode 节点的映射。Datanode 负责处理文件系统客户端的读写请求。在 Namenode 的统一调度下进行数据块的创建、删除和复制。

Namenode 和 Datanode 被设计成可以在普通的商用机器上运行。这些机器一般运行着 GNU/Linux 操作系统(OS)。HDFS 采用 Java 语言开发，因此任何支持 Java 的机器都可以部署 Namenode 或 Datanode。由于采用了可移植性极强的 Java 语言，使得 HDFS 可以部署到多种类型的机器上。一个典型的部署场景是一台机器上只运行一个 Namenode 实例，而集群中的其它机器分别运行一个 Datanode 实例。

集群中单一 Namenode 的结构大大简化了系统的架构。Namenode 是所有 HDFS 元数据的仲裁者和管理者，这样，用户数据永远不会流过 Namenode。但是现阶段，普通的 NameNode 节点，会出现单点故障的问题，可能会在下一个版本解决。或者使用 NameNode 集群的方式进行热备处理。

2.2.2 数据复制

HDFS 被设计成能够在一个大集群中跨机器可靠地存储超大文件。它将每个文件存储成一系列的数据块，除了最后一个，所有的数据块都是同样大小的。为了容错，文件的所有数据块都会有副本。每个文件的数据块大小和副本系数都是可配置的。应用程序可以指定某个文件的副本数目。副本系数可以在文件创建的时候指定，也可以在之后改变。HDFS 中的文件都是一次性写入的，并且严格要求在任何时候只能有一个写入者。

Namenode 全权管理数据块的复制，它周期性地从集群中的每个 Datanode 接收心跳信号和块状态报告(Blockreport)。接收到心跳信号意味着该 Datanode 节点工作正常。块状态报告包含了一个该 Datanode 上所有数据块的列表。

2.2.3 文件系统元数据的持久化

Namenode 上保存着 HDFS 的名字空间。对于任何对文件系统元数据产生修改的操作，Namenode 都会使用一种称为 EditLog 的事务日志记录下来。例如，在 HDFS 中创建一个文件，Namenode 就会在 Editlog 中插入一条记录来表示；同样地，修改文件的副本系数也将往 Editlog 插入一条记录。Namenode 在本地操作系统的文件系统中存储这个 Editlog。整个文件系统的名字空间，包括数据块到文件的映射、文件的属性等，都存储在一个称为 FsImage

的文件中，这个文件也是放在 Namenode 所在的本地文件系统上。

Namenode 在内存中保存着整个文件系统的名字空间和文件数据块映射(Blockmap)的映像。这个关键的元数据结构设计得很紧凑，因而一个有 4G 内存的 Namenode 足够支撑大量的文件和目录。当 Namenode 启动时，它从硬盘中读取 Editlog 和 FsImage，将所有 Editlog 中的事务作用在内存中的 FsImage 上，并将这个新版本的 FsImage 从内存中保存到本地磁盘上，然后删除旧的 Editlog，因为这个旧的 Editlog 的事务都已经作用在 FsImage 上了。这个过程称为一个检查点(checkpoint)。

Datanode 将 HDFS 数据以文件的形式存储在本地的文件系统中，它并不知道有关 HDFS 文件的信息。它把每个 HDFS 数据块存储在本地文件系统的一个单独的文件中。Datanode 并不在同一个目录创建所有的文件，实际上，它用试探的方法来确定每个目录的最佳文件数目，并且在适当的时候创建子目录。在同一个目录中创建所有的本地文件并不是最优的选择，这是因为本地文件系统可能无法高效地在单个目录中支持大量的文件。当一个 Datanode 启动时，它会扫描本地文件系统，产生一个这些本地文件对应的所有 HDFS 数据块的列表，然后作为报告发送到 Namenode，这个报告就是块状态报告。

2.2.4 通讯协议

所有的 HDFS 通讯协议都是建立在 TCP/IP 协议之上。客户端通过一个可配置的 TCP 端口连接到 Namenode，通过 ClientProtocol 协议与 Namenode 交互。而 Datanode 使用 DatanodeProtocol 协议与 Namenode 交互。一个远程过程调用(RPC)模型被抽象出来封装 ClientProtocol 和 Datanodeprotocol 协议。在设计上，Namenode 不会主动发起 RPC，而是响应来自客户端或 Datanode 的 RPC 请求。

2.2.5 健壮性

HDFS 的主要目标就是即使在出错的情况下也要保证数据存储的可靠性。常见的三种出错情况是：Namenode 出错, Datanode 出错和网络割裂(network partitions)。

每个 Datanode 节点周期性地向 Namenode 发送心跳信号。网络割裂可能导致一部分 Datanode 跟 Namenode 失去联系。Namenode 通过心跳信号的缺失来检测这一情况，并将这些近期不再发送心跳信号 Datanode 标记为宕机，不会再将新的 IO 请求发给它们。任何存储在宕机 Datanode 上的数据将不再有效。Datanode 的宕机可能会引起一些数据块的副本系数低于指定值，Namenode 不断地检测这些需要复制的数据块，一旦发现就启动复制操作。在下列情况下，可能需要重新复制：某个 Datanode 节点失效，某个副本遭到损坏，Datanode 上

的硬盘错误，或者文件的副本系数增大。

从某个 Datanode 获取的数据块有可能是损坏的，损坏可能是由 Datanode 的存储设备错误、网络错误或者软件 bug 造成的。HDFS 客户端软件实现了对 HDFS 文件内容的校验和(checksum)检查。当客户端创建一个新的 HDFS 文件，会计算这个文件每个数据块的校验和，并将校验和作为一个单独的隐藏文件保存在同一个 HDFS 名字空间下。当客户端获取文件内容后，它会检验从 Datanode 获取的数据跟相应的校验和文件中的校验和是否匹配，如果不匹配，客户端可以选择从其他 Datanode 获取该数据块的副本。

FsImage 和 Editlog 是 HDFS 的核心数据结构。如果这些文件损坏了，整个 HDFS 实例都将失效。因而，Namenode 可以配置成支持维护多个 FsImage 和 Editlog 的副本。任何对 FsImage 或者 Editlog 的修改，都将同步到它们的副本上。这种多副本的同步操作可能会降低 Namenode 每秒处理的名字空间事务数量。然而这个代价是可以接受的，因为即使 HDFS 的应用是数据密集的，它们也非元数据密集的。当 Namenode 重启的时候，它会选取最近的完整的 FsImage 和 Editlog 来使用。

Namenode 是 HDFS 集群中的单点故障(single point of failure)所在。如果 Namenode 机器故障，是需要手工干预的。目前，自动重启或在另一台机器上做 Namenode 故障转移的功能还没实现。

2.3 MapReduce 的原理和执行流程

待续

3. Hadoop 集群的安装

3.1 Hadoop 集群的系统及软件版本

操作系统：CentOS release 6.2 64 位

Hadoop 版本：hadoop-1.1.1

Eclipse 版本：eclipse-java-helios-SR2 （在应用 hadoop-eclipse-plugin-1.0.0.jar 包的情况中，建议使用这个 Eclipse 版本，否则会有许多从本地连接远程集群的问题会出现）

3.2 安装 Hadoop 集群前的准备

在 CentOS 系统中创建控制 Hadoop 集群的用户，该用户后面会有开发环境中的用户对应起来，减少 Hadoop 开发的调试复杂性。**需要在集群中每一个操作系统中都建立相同的用户。**

```
[root@Master /]# useradd hadoop
[root@Master /]# passwd hadoop
Changing password for user hadoop.
New password:
BAD PASSWORD: it is too simplistic/systematic
BAD PASSWORD: is too simple
Retype new password:
passwd: all authentication tokens updated successfully.
```

该示例中创建了一个 hadoop 的用户，后面将用它来管理 hadoop 集群。并且在本地的开发机器中的用户一致，便于后面的调试。

由于集群中的各个机器需要通信，所以关闭系统的防火墙和 SELinux 复杂的保护，如果后面 Hadoop 集群的部署在生产环境中时，再根据需要开放防火墙。

永久关闭防火墙：`#chkconfig --level 35 iptables off`

关闭 SELiunx，执行命令 `vim /etc/sysconfig/seliunx`

将 SELIUNX=enforcing 改为 SELIUNX=disabled

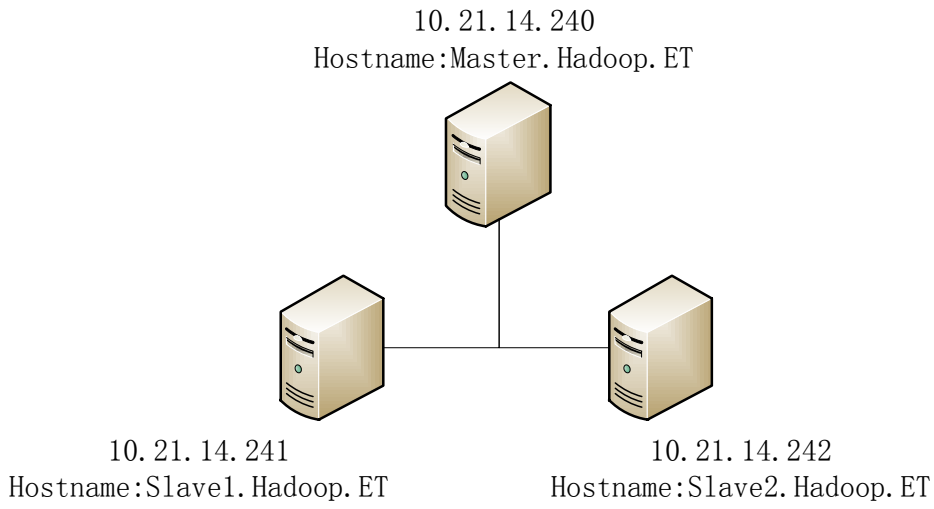
接着继续执行如下命令:

```
#setenforce permissive
```

关闭不需要的服务，可以避免 Hadoop 集群的应用麻烦。执行如下命令：

```
# for SERVICES in abrtd acpid auditd cpuspeed haldaemon mdmonitor messagebus udev-post;do chkconfig ${SERVICES} off;done
```

部署架构图



3.3 网络配置

在 Hadoop 测试集群中，一共规划了三台机器，部署和配置如下表。

Explain	hostname	IP Address	
Master	Master.Hadoop.ET	10.21.14.240	
Slave1	Slave1.Hadoop.ET	10.21.14.241	
Slave2	Slave2.Hadoop.ET	10.21.14.242	

3.3.1 配置 hosts 文件

Hadoop 和 HBase 在通信时，需要用到机器名进行寻址，所以要对/etc/hosts 文件进行编辑。将上表中所规划的三台机器名和 IP 地址在 hosts 文件中进行对应。

10.21.14.240 Master.Hadoop.ET

10.21.14.241 Slave1.Hadoop.ET

10.21.14.242 Slave2.Hadoop.ET

3.3.2 配置 SSH 无密码验证

Hadoop 运行过程中需要管理远端 Hadoop 守护进程，在 Hadoop 启动之后，NameNode 是过 SSH（Secure Shell）来启动和停止各个 DataNode 上的各种守护进程的。这就必须在节点之间执行指令时，不需要进行密码的输入，简化操作。故此需要配置 SSH 运用无密码公钥认证的形式，这样 NameNode 可以使用 SSH 无密码登陆并启动 DataName 进程，同样原理，DataNode 上也能使用 SSH 无密码登陆到 NameNode。因为在安装 CentOS 系统时，已经选择了 ssh 和 rsync 服务，所以在这儿不需要进行安装，可以通过命令来检查服务是否正常。

```
[root@Master /]# ssh 10.21.14.241
The authenticity of host '10.21.14.241 (10.21.14.241)' can't be established.
RSA key fingerprint is cb:bf:07:18:c5:e4:38:f9:85:fa:71:f7:32:21:ee:1b.
Are you sure you want to continue connecting (yes/no)? now^H
Host key verification failed.
```

3.3.2.1 SSH 无密码登陆的原理

Master（NameNode | JobTracker | HMaster）作为客户端，要实现无密码公钥认证，连接到服务器（DataNode | Tasktracker）上时，需要在 Master 上生成一个密钥对，包括一个公钥和一个私钥,而后将公钥复制到所有 Slave 上。当 Master 通过 SSH 连接 Salve 时，Salve 就会生成一个随机数并用 Master 的公钥对随机数进行加密,并发送给 Master。Master 收到加密数之后再用私钥解密，并将解密数回传给 Slave，Slave 确认解密无误之后就允许 Master 进行连接了。这个就是典型的非对称加密算法的应用。

3.3.2.2 Master 机器上生成密码对

切换至 hadoop 的用户，并且在在 Master 节点上执行以下命令：

```
ssh-keygen -t rsa -P ''
```

这条命令是生成其无密码密钥对，询问其保存路径时直接回车采用默认路径。生成的密钥对：id_rsa 和 id_rsa.pub，默认存储在/home/hadoop/.ssh 目录下。

```
[root@Master /]# su hadoop
[hadoop@Master /]$ ssh-keygen -t rsa -P ''
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hadoop/.ssh/id_rsa):
Created directory '/home/hadoop/.ssh'.
Your identification has been saved in /home/hadoop/.ssh/id_rsa.
Your public key has been saved in /home/hadoop/.ssh/id_rsa.pub.
The key fingerprint is:
28:72:c6:f3:d6:c8:57:4e:d3:b4:db:b2:ba:4d:42:82 hadoop@Master.Hadoop.ET
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|                 |
|            .    |
|   .   o   o .   |
|  . * E S = o    |
|   + = o * . o   |
|      = o o + .  |
|     . .   + o   |
|          ooo    |
+-----------------+
```

查看/home/hadoop/下是否有.ssh 文件夹，且.ssh 文件下是否有两个产生的无密码密钥对。

```
[hadoop@Master /]$ ls -ls /home/hadoop/.ssh/
total 8
4 -rw-------. 1 hadoop hadoop 1675 Mar  6 14:32 id_rsa
4 -rw-r--r--. 1 hadoop hadoop  405 Mar  6 14:32 id_rsa.pub
```

3.3.2.3 配置 Master 与 Slave 的信任关系

在 Master 节点上做如下配置，把 id_rsa.pub 追加到授权的 key 里面去。

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

```
[hadoop@Master /]$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
[hadoop@Master /]$
```

在进行信任建立前，需要做两件事儿。第一件事是修改文件 authorized_keys 权限（**权限的设置非常重要，因为不安全的设置，会让系统不能正常使用 RSA 功能，可能会让机器之间的访问依然需要密码验证。**），另一件事是用 root 用户设置**/etc/ssh/sshd_config** 的

内容。使其无密码登录有效。

1， 修改 authorized_keys 文件的权限。

执行命令：`chmod 600 ~/.ssh/authorized_keys`

2， 用 root 用户登录服务器修改 SSH 配置文件/etc/ssh/sshd_config 的下列内容。

RSAAuthentication yes # 启用 RSA 认证

PubkeyAuthentication yes # 启用公钥私钥配对认证方式

AuthorizedKeysFile .ssh/authorized_keys # 公钥文件路径（和上面生成的文件同）

设置完之后记得**重启 SSH 服务**，才能使刚才设置有效。

执行命令重启 SSH 服务：`service sshd restart`

3，需要把公钥复制所有的 Slave 机器上。切换成 hadoop 用户，执行 scp 命令

`scp ~/.ssh/id_rsa.pub hadoop@10.21.14.241:~/`

`scp ~/.ssh/id_rsa.pub hadoop@10.21.14.242:~/`

```
[hadoop@Master /]$ scp ~/.ssh/id_rsa.pub hadoop@10.21.14.241:~/
The authenticity of host '10.21.14.241 (10.21.14.241)' can't be established.
RSA key fingerprint is cb:bf:07:18:c5:e4:38:f9:85:fa:71:f7:32:21:ee:1b.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.21.14.241' (RSA) to the list of known hosts.
hadoop@10.21.14.241's password:
id_rsa.pub                                    100%  405     0.4KB/s   00:00
[hadoop@Master /]$ scp ~/.ssh/id_rsa.pub hadoop@10.21.14.242:~/
The authenticity of host '10.21.14.242 (10.21.14.242)' can't be established.
RSA key fingerprint is 32:ac:81:8d:c3:be:e6:f8:4d:2c:dc:ff:e4:71:e9:e3.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.21.14.242' (RSA) to the list of known hosts.
hadoop@10.21.14.242's password:
id_rsa.pub                                    100%  405     0.4KB/s   00:00
[hadoop@Master /]$
```

上面的命令是将 Master 上面的文件 id_rsa.pub 复制到 10.21.14.241 和 10.21.14.242 的用户为 hadoop 的/home/hadoop/目录下面。

4，在各个 Slave 机器目录/home/hadoop/下创建.ssh 文件夹

这一步并不是必须的，如果在 Slave1.Hadoop.ET 的/home/hadoop 已经存在就不需要创建了，因为之前并没有对 Slave 机器做过无密码登录配置，所以该文件是不存在的。用下面命令进行创建。（**备注：用 hadoop 登录系统，如果不涉及系统文件修改，一般情况下都是用我们之前建立的普通用户 hadoop 进行执行命令。**）

`mkdir ~/.ssh`

然后是修改文件夹.ssh 的用户权限，把他的权限修改为 700，用下面命令执行：

```
chmod 700 ~/.ssh
```

备注：如果不进行上面的操作。会因为.ssh 文件夹的权限设置不对，导致 RSA 无密码远程登录失败。

5，追加到各个 Slave 系统的授权文件 authorized_keys 中

使用下面命令进行追加并修改 authorized_keys 文件权限：

```
cat ~/id_rsa.pub >> ~/.ssh/authorized_keys
```

```
chmod 600 ~/.ssh/authorized_keys
```

6，在各个 Slave 系统中切换回 root 用户修改/etc/ssh/sshd_config 文件

RSAAuthentication yes # 启用 RSA 认证

PubkeyAuthentication yes # 启用公钥私钥配对认证方式

AuthorizedKeysFile .ssh/authorized_keys # 公钥文件路径（和上面生成的文件同）

重启 SSH 服务，才能使刚才设置有效。执行命令重启 SSH 服务：

```
service sshd restart
```

7，在 Master.Hadoop.ET 上面使用 SSH 无密码登录 Slave1.Hadoop.ET 进行测试

```
[root@Master /]# su hadoop
[hadoop@Master /]$ ssh 10.21.14.241
[hadoop@Slave1 ~]$ exit
logout
Connection to 10.21.14.241 closed.
[hadoop@Master /]$ ssh 10.21.14.242
[hadoop@Slave2 ~]$ exit
logout
Connection to 10.21.14.242 closed.
[hadoop@Master /]$
```

可以看出，已经成功的配置了从 Master 至 Slave 的信任关系

3.3.2.4 配置 Slave 与 Master 的信任关系

与 Master 无密码登录所有 Slave 原理一样，就是把 Slave 的公钥追加到 Master 的.ssh 文件夹下的 authorized_keys 中，记得是追加（>>）。

为了说明情况，现在就以 Slave1.Hadoop.ET 无密码登录 Master.Hadoop.ET 为例，进行一遍操作，巩固一下前面知识。

首先创建 Slave1.Hadoop.ET 自己的公钥和私钥，并把自己的公钥追加到

authorized_keys 文件中。用到的命令如下：

```
ssh-keygen –t rsa –P ''
```

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

接着是用命令 scp 复制 Slave1.Hadoop.ET 的公钥 id_rsa.pub 到 Master.Hadoop.ET 的/home/hadoop/目录下。

```
scp ~/.ssh/id_rsa.pub  hadoop@10.21.14.240:~/
```

在 Master 上面，登陆 Hadoop 用户追加到 Master.Hadoop 的 authorized_keys 中。

```
cat ~/id_rsa.pub >> ~/.ssh/authorized_keys
```

最后是测试从 Slave1.Hadoop.ET 到 Master.Hadoop.ET 的信任关系是否建立成功。

```
[hadoop@Slave1 /]$ ssh 10.21.14.240
Last login: Wed Mar  6 21:41:38 2013 from master.hadoop.et
[hadoop@Master ~]$
```

3.4　Java 环境安装

所有的机器上都要安装 JDK，先在 Master 服务器安装，然后其他服务器按照步骤重复进行即可。安装 JDK 以及配置环境变量，需要以 root 的角色进行。

1，创建 Java 的目录，将 jdk-6u38-linux-x64.bin 复制进去。

```
mkdir /usr/java
```

2，接着进入/usr/java 目录下通过下面命令使其 JDK 获得可执行权限，并安装 JDK。

```
chmod +x jdk-6u38-linux-x64.bin
```

```
./jdk-6u38-linux-x64.bin
```

3，配置环境变量，编辑/etc/profile 文件，在后面添加 Java 的 JAVA_HOME、CLASSPATH 以及 PATH 内容。

```
vim /etc/profile
```

在/etc/profile 文件的尾部添加以下内容：

```
export JAVA_HOME=/usr/java/jdk1.6.0_38/
export JRE_HOME=/usr/java/jdk1.6.0_38/jre
export HADOOP_HOME_WARN_SUPPRESS=1
export CLASSPATH=.:$CLASSPATH:$JAVA_HOME/lib:$JRE_HOME/lib
```

export PATH=$PATH:$JAVA_HOME/bin:$JRE_HOME/bin

4，保存并退出，执行下面命令使其配置立即生效。

```
source /etc/profile
```

5，配置完毕并生效后，用下面命令判断是否成功。

```
Java – version
```

```
[root@Master java]# source /etc/profile
[root@Master java]# java -version
java version "1.6.0_38"
Java(TM) SE Runtime Environment (build 1.6.0_38-b05)
Java HotSpot(TM) 64-Bit Server VM (build 20.13-b02, mixed mode)
[root@Master java]#
```

3.5 Hadoop 的安装

所有的机器上都要安装 hadoop，先在 Master 服务器安装，然后将配置好的 hadoop 复制到其它 Slave 机器上即可。**需要注意的是安装和配置 hadoop 要以 root 的身份进行。**

1，将 hadoop-1.1.1.tar.gz 复制到/usr/的目录下，root 用户登陆 Master.Hadoop.ET 服务器进行解压。进入/usr 的目录之后，执行如下命令：

```
tar  -zxvf hadoop-1.1.1.tar.gz
```
#解压 hadoop-1.1.1.tar.gz 安装包

2，为了方便后面的操作，将 hadoop-1.1.1 进行更名。

```
mv hadoop-1.1.1 hadoop
```
#将 hadoop-1.1.1 文件夹重命名 hadoop

3， 在/usr/hadoop 的目录中，创建一个 tmp 的文件夹，用于 hadoop 的配置。

```
mkdir /usr/hadoop/tmp
```

然后删除 hadoop 的压缩包

```
rm -rf hadoop-1.1.1.tar.gz
```

4，接着需要将 hadoop 文件夹的权限分配给 hadoop 用户，这步很重要。

```
chown  -R hadoop:hadoop hadoop
```
#将文件夹 hadoop 权限分配给 hadoop 用户与 hadoop 用户组 hadoop:hadoop

```
drwxr-xr-x. 15 hadoop hadoop  4096 Nov 19 18:50 hadoop
```

5，在系统中配置 hadoop 的环境变量。方便 hadoop 的命令执行。使用 vi 编辑/etc/profile 文件，内容如下

```
export JAVA_HOME=/usr/java/jdk1.6.0_38/
export JRE_HOME=/usr/java/jdk1.6.0_38/jre
export CLASSPATH=.:$CLASSPATH:$JAVA_HOME/lib:$JRE_HOME/lib
export PATH=$PATH:$JAVA_HOME/bin:$JRE_HOME/bin

export HADOOP_HOME=/usr/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
```

执行 source /etc/profile 命令重启/etc/profile

3.3.3 Hadoop 的基础配置

根据以下的配置，可以将 hadoop 的集群运行起来，但需要在生产环境中做更详细的配置，可以查看相关的章节。

1， 配置 hadoop-env.sh

该 hadoop-env.sh 文件位于/usr/hadoop/conf 目录下。它的作用是配置与 hadoop 运行环境相关的变量，其中有一个变量是 JAVA_HOME，将它修改为 java 的路径。

export JAVA_HOME=/usr/java/jdk1.6.0_38

```
# The java implementation to use.  Required.
export JAVA_HOME=/usr/java/jdk1.6.0_38
```

Hadoop 配置文件在 conf 目录下，之前的版本的配置文件主要是 Hadoop-default.xml 和 Hadoop-site.xml。由于 Hadoop 发展迅速，代码量急剧增加，代码开发分为了 core，hdfs 和 map/reduce 三部分，配置文件也被分成了三个 core-site.xml、hdfs-site.xml、mapred-site.xml。core-site.xml 和 hdfs-site.xml 是站在 HDFS 角度上配置文件；core-site.xml 和 mapred-site.xml 是站在 MapReduce 角度上配置文件。

2， 配置 core-site.xml 文件

修改 Hadoop 核心配置文件 core-site.xml，这里配置的是 HDFS 的地址和端口号。

<configuration>

```
  <property>
    <name>hadoop.tmp.dir</name>
    <value>/usr/hadoop/tmp</value>
  </property>
  <property>
    <name>fs.default.name</name>
    <value>hdfs://10.21.14.240:9000</value>
  </property>
</configuration>
```

备注：如没有配置 hadoop.tmp.dir 参数，此时系统默认的临时目录为：/tmp/hadoo-hadoop。而这个目录在每次重启后都会被删除，必须重新执行 hadoop namenode format 才行，否则会出错。

3，配置 hdfs-site.xml 文件

修改Hadoop中HDFS的配置,该文件配置中的备份方式默认为3。如果实际情况salve少于 3 台就会报错，而且当前阶段只需要配置基础环境，所以改成 1 即可。

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>
```

4，配置 mapred-site.xml 文件

修改 Hadoop 中 MapReduce 的配置文件，配置的是 JobTracker 的地址和端口。

```
<configuration>
      <property>
        <name>mapred.job.tracker</name>
        <value>http://10.21.14.240:9001</value>
      </property>
</configuration>
```

5，配置 masters 文件。

去掉 localhost，加入 Master 机器的 IP：10.21.14.240

6， 配置 slaves 文件（Master 主机特有）

去掉 localhost，加入集群中所有 Slave 机器的 IP，也是每行一个。

10.21.14.241

10.21.14.242

7，使用 scp 的命令，将 Master 中配置好的 hadoop 复制到集群中的所有 Slave 机器中

```
scp -r /usr/hadoop root@10.21.14.241:/usr/
```

```
scp -r /usr/hadoop root@10.21.14.242:/usr/
```

8，接着在所有的 Slave 上修改/etc/profile 文件（配置 java 环境变量的文件），将以下语句添加到末尾，并使其有效：

```
export HADOOP_HOME=/usr/hadoop
export PATH=$PATH:$HADOOP_HOME/bin
```

然后使用 `source /etc/profile` 命令重启 profile

9，给集群中的所有系统，分配 hadoop 目录的权限给 hadoop 用户。

```
chown -R hadoop:hadoop /usr/hadoop
```

3.3.4 启动及验证

1， 格式化 HDFS 文件系统

在 Master.Hadoop.ET 上使用普通用户 hadoop 进行操作。（备注：只需一次，下次启动不再需要格式化，只需 start-all.sh 即可）

```
hadoop namenode -format
```

```
[hadoop@Master /]$ hadoop namenode -format
13/03/13 22:52:44 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = Master.Hadoop.ET/10.21.14.240
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 1.1.1
STARTUP_MSG:   build = https://svn.apache.org/repos/asf/hadoop/common/branches/b
ranch-1.1 -r 1411108; compiled by 'hortonfo' on Mon Nov 19 10:48:11 UTC 2012
************************************************************/
13/03/13 22:52:44 INFO util.GSet: VM type       = 64-bit
13/03/13 22:52:44 INFO util.GSet: 2% max memory = 17.77875 MB
13/03/13 22:52:44 INFO util.GSet: capacity      = 2^21 = 2097152 entries
13/03/13 22:52:44 INFO util.GSet: recommended=2097152, actual=2097152
13/03/13 22:52:44 INFO namenode.FSNamesystem: fsOwner=hadoop
13/03/13 22:52:45 INFO namenode.FSNamesystem: supergroup=supergroup
13/03/13 22:52:45 INFO namenode.FSNamesystem: isPermissionEnabled=true
13/03/13 22:52:45 INFO namenode.FSNamesystem: dfs.block.invalidate.limit=100
13/03/13 22:52:45 INFO namenode.FSNamesystem: isAccessTokenEnabled=false accessK
eyUpdateInterval=0 min(s), accessTokenLifetime=0 min(s)
13/03/13 22:52:45 INFO namenode.NameNode: Caching file names occuring more than
10 times
13/03/13 22:52:45 INFO common.Storage: Image file of size 112 saved in 0 seconds
.
13/03/13 22:52:45 INFO namenode.FSEditLog: closing edit log: position=4, editlog
=/usr/hadoop/tmp/dfs/name/current/edits
13/03/13 22:52:45 INFO namenode.FSEditLog: close success: truncate to 4, editlog
=/usr/hadoop/tmp/dfs/name/current/edits
13/03/13 22:52:45 INFO common.Storage: Storage directory /usr/hadoop/tmp/dfs/nam
e has been successfully formatted.
13/03/13 22:52:45 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
```

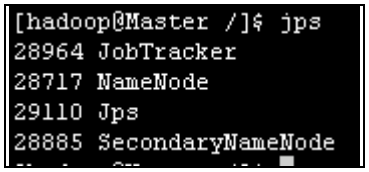
2， 启动 Hadoop 系统

执行命令 `start-all.sh`

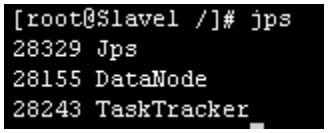
```
[hadoop@Master /]$ start-all.sh
starting namenode, logging to /usr/hadoop/libexec/../logs/hadoop-hadoop-namenode
-Master.Hadoop.ET.out
10.21.14.242: starting datanode, logging to /usr/hadoop/libexec/../logs/hadoop-h
adoop-datanode-Slave2.Hadoop.ET.out
10.21.14.241: starting datanode, logging to /usr/hadoop/libexec/../logs/hadoop-h
adoop-datanode-Slave1.Hadoop.ET.out
10.21.14.240: starting secondarynamenode, logging to /usr/hadoop/libexec/../logs
/hadoop-hadoop-secondarynamenode-Master.Hadoop.ET.out
starting jobtracker, logging to /usr/hadoop/libexec/../logs/hadoop-hadoop-jobtra
cker-Master.Hadoop.ET.out
10.21.14.241: starting tasktracker, logging to /usr/hadoop/libexec/../logs/hadoo
p-hadoop-tasktracker-Slave1.Hadoop.ET.out
10.21.14.242: starting tasktracker, logging to /usr/hadoop/libexec/../logs/hadoo
p-hadoop-tasktracker-Slave2.Hadoop.ET.out
[hadoop@Master /]$
```

3， 使用 jps 命令查看 hadoop 是否运行起来

在 Master 中 jps 的结果



在 Slave 中 jps 的结果



完成 Hadoop 集群的基本搭建

4. HBase 集群的安装

待续

5. Hive 的安装

待续

6. Eclipse 开发环境的配置

待续

7. Hadoop、MapReduce、HBase、Hive API

待续

8. Hadoop 与 ESB 的结合实例

待续