Hadoop 包含两个部分，Hadoop 文件系统和 MapReduce 编程模型，图 1 为 Hadoop 的组成部分。其中 HDFS 运行在商用硬件上，HDFS 是高度容错的，可运行在廉价硬件上；HDFS 能为应用程序提供高吞吐率的数据访问，适用于大数据集的应用中；HDFS 在 POSIX 规范进行了修改，使之能对文件系统数据进行流式访问，从而适用于批量数据的处理。HDFS 为文件采用一种"一次写多次读"的访问模型，从而简化了数据一致性问题，使高吞吐率数据访问成为可能，一些 Map/Reduce 应用和网页抓取程序在这种访问模型下表现完美。HDFS 在云计算中特别是其分布式系统布局得到了人们广泛的关注，并得到了很好的应用。

HDFS	Distributed file system Subject of this paper!
MapReduce	Distributed computation framework
HBase	Column-oriented table service
Pig	Dataflow language and parallel execution framework
Hive	Data warehouse infrastructure
ZooKeeper	Distributed coordination service
Chukwa	System for collecting management data
Avro	Data serialization system

图 1 Hadoop 组成

1．HDFS 体系结构

HDFS 的体系结构如图 2 所示，HDFS 采用 master/slave 架构，一个 HDFS 集群主要由一个 Namenode 和一定数目的 DataNode 组成，并且还有 Secondray namenode 和客户端共同构成整个完整的 HDFS 体系，HDFS 内部的所有通信都基于标准的 TCP/IP 协议。

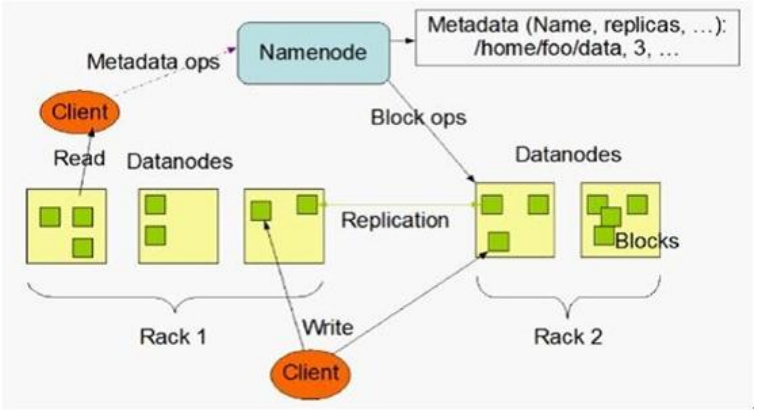

图 2 HDFS 体系结构图

2．Namenode

Namenode（图 3）是 HDFS 的中心服务器，负责管理文件系统的 namespace（命名空间）和客户端对文件的访问。Namenode 执行文件系统的 namespace 操作例如打开，关闭，重命名文件和目录，同时决定了 block 到具体的 DataNode 的映射关系，DataNode

在 Namenode 的指挥之下进行 block 的创建、删除和复制。除此之外，集群配置管理，处理事务日志：记录文件的创建、删除等。因为 Namenode 的全部元数据在内存中存储，所以内存大小决定了整个集群的存储量。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，用户数据不会和 Namenode 进行交互而是直接和 DataNode 进行传输。

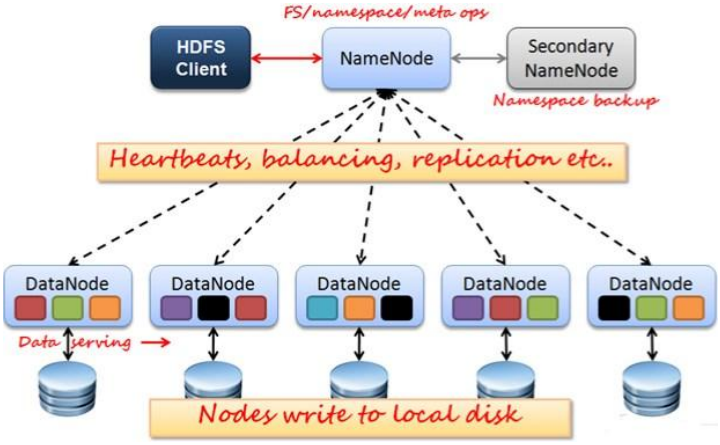

图 3 NameNode 和 DataNode

3．DataNode

DataNode 在集群中一般是一个节点一个，它可以作为块服务器，存储本地文件系统的数据和块的元数据，并且提供服务数据和元数据给客户端，负责管理节点上他们附带的存储。在内部，一个文件其实分为多个 block，这些 block 存储在 DataNode 集合里。DataNode 在 Namenode 的指挥下进行 block 的创建、删除和复制。除此之外，DataNode 进行块报道即周期性发送所有正存在的块的报告给 Namenode 进行系统检测，在 DataNode 内部也进行数据发送给其他的 DataNode 流水线。Namenode 和 DataNode 都是设计成可以跑在不同的廉价的运行 linux 及其上。一个典型的部署场景是一台机器跑一个单独的 Namenode 节点，集群中其他机器各跑一个 DataNode 实例。

4．文件系统的名字空间(namespace)

HDFS 支持传统的层次性文件组织结构，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、移动或者重命名文件。当前 HDFS 不支持用户磁盘配额和访问权限控制，也不支持硬链接和软连接。Namenode 负责维护文件系统的名字空间，任何对文件系统名字空间或者属性的更改都将被 Namenode 记录下来。应用程序可以设置 HDFS 保存文件的副本数目。文件副本的数目成为文件的副本系数，这个信息也是由 Namenode 保存。

5．Secondary Namenode

为了提高 NameNode 的可靠性，从 Hadoop 0.23 开始引入了 Secondary NameNode（图 4）。Secondary namenode 其实作为一个保存 Namenode 修改的过程，Fsimage 和 Editlog 是 HDFS 的核心结构，Fsimage 记录了名字空间 namespace，Editlog

记录了元数据的改变。每次从 Namenode 中复制 fsimage 和 log 到一个临时目录，然后在临时目录中集合 Fsimage 和 Editlog 到新的 fsimage，然后更新新的 Fsimage 给 Namenode。

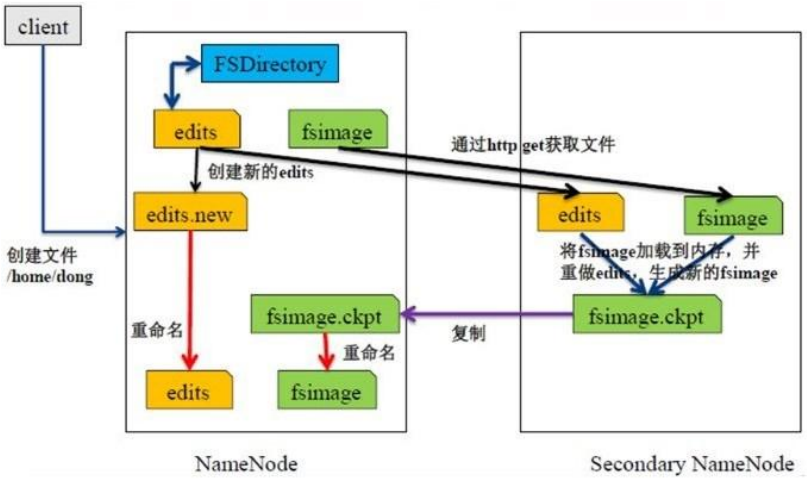

图 4 Namenode 和 Secondary Namenode 运行关系

6．HDFS 数据组织

从 HDFS 体系结构可以看到 Namenode 作为控制枢纽，控制整个系统，但真正的数据交换却不经过 Namenode 而是直接在客户端和 DataNode 之间进行。在 HDFS 中，默认的数据块的大小为 64M，文件所对应的块按照一定的部署策略存于 DataNode 中，Namenode 管理了文件到 DataNode 的映射。这个数据组织结构减少了客户端和 Namenode 交互需求，对同一块的读写只需要向 Namenode 发送一个请求即可。另外块大小可以让客户端在一个给定块上进行多次操作，降低了网络交互困难，同时也减少了 Namenode 的元数据大小。图 5 显示了数据组织结构，Namenode 中元数据存放了文件路径，副本个数，存放位置等信息，通过这种组织方式能很好地找到文件块位置加快了查找速度。

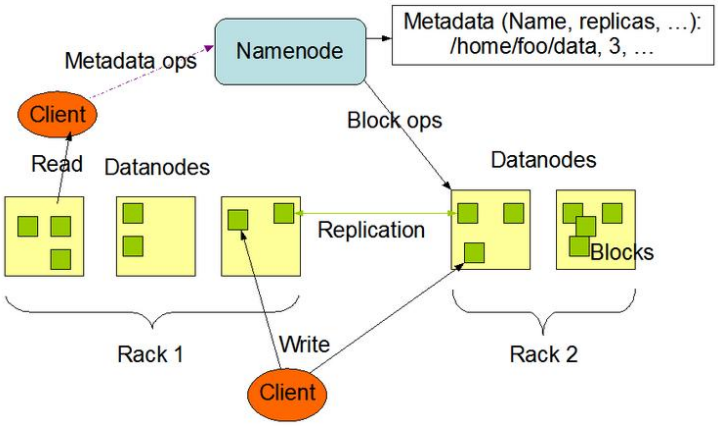

图 5 HDFS 的数据组织结构

➢ 数据块

HDFS 被设计成支持大文件，适用 HDFS 的是那些需要处理大规模的数据集的应用。这些应用都是只写入数据一次，但却读取一次或多次，并且读取速度应能满足流式读取的需要。

HDFS 支持文件的“一次写入多次读取”语义。一个典型的数据块大小是 256MB。因而，HDFS 中的文件总是按照 256M 被切分成不同的块，每个块尽可能地存储于不同的 Datanode 中。

- 分段

客户端创建文件的请求其实并没有立即发送给 Namenode，事实上，在刚开始阶段 HDFS 客户端会先将文件数据缓存到本地的一个临时文件。应用程序的写操作被透明地重定向到这个临时文件。当这个临时文件累积的数据量超过一个数据块的大小，客户端才会联系 Namenode。Namenode 将文件名插入文件系统的层次结构中，并且分配一个数据块给它。然后返回 Datanode 的标识符和目标数据块给客户端。接着客户端将这块数据从本地临时文件上传到指定的 Datanode 上。当文件关闭时，在临时文件中剩余的没有上传的数据也会传输到指定的 Datanode 上。然后客户端告诉 Namenode 文件已经关闭。此时 Namenode 才将文件创建操作提交到日志里进行存储。如果 Namenode 在文件关闭前宕机了，则该文件将丢失。

上述方法是对在 HDFS 上运行的目标应用进行认真考虑后得到的结果。这些应用需要进行文件的流式写入。如果不采用客户端缓存，由于网络速度和网络堵塞会对吞估量造成比较大的影响。这种方法并不是没有先例的，早期的文件系统，比如 AFS，就用客户端缓存来提高性能。为了达到更高的数据上传效率，已经放松了 POSIX 标准的要求。

- 管道复制

当客户端向 HDFS 文件写入数据的时候，一开始是写到本地临时文件中。假设该文件的副本系数设置为 3，当本地临时文件累积到一个数据块的大小时，客户端会从 Namenode 获取一个 Datanode 列表用于存放副本。然后客户端开始向第一个 Datanode 传输数据，第一个 Datanode 一小部分一小部分(4KB)地接收数据，将每一部分写入本地仓库，并同时传输该部分到列表中第二个 Datanode 节点。第二个 Datanode 也是这样，一小部分一小部分地接收数据，写入本地仓库，并同时传给第三个 Datanode。最后，第三个 Datanode 接收数据并存储在本地。因此，Datanode 能流水线式地从前一个节点接收数据，并在同时转发给下一个节点，数据以流水线的方式从前一个 Datanode 复制到下一个。

7．HDFS 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。每个文件数据块大小和副本系统都是可配置的，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 的文件都是一次性写入的，并且严格要求在任何时候只有一个写入者。Namenode 管理数据块的复制，周期性的从集群中的每个 DataNode 接受心跳检测和块状态报告。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，如图 6 中显示。

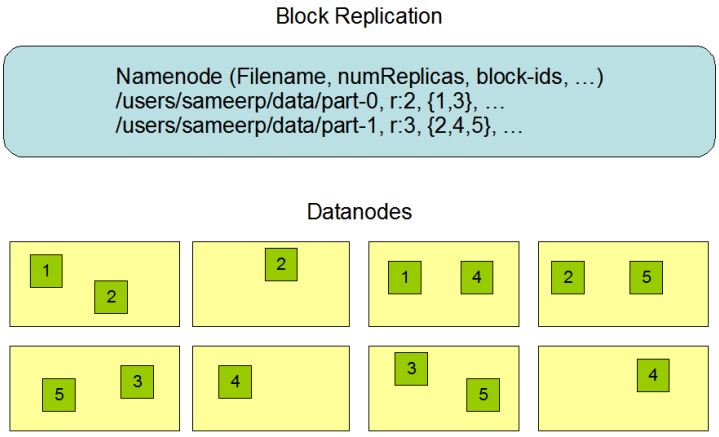

图 6 数据块的复制

8．副本存放

副本存放是 HDFS 可靠性和性能的关键，优化的副本存放策略是 HDFS 区分于其他大部分分布式文件系统的重要特性。这种特性需要做大量的调优，并需要经验的累积。HDFS 采用一种机架感知(rack-aware)的策略来改进数据的可靠性、可用性和网络带宽的利用率。通过机架感知的过程，Namenode 可以确定每个 DataNode 所属的机架 id。一个简单但没有优化的策略就是将副本存放在不同机架之上，这样可以有效防止当整个机架失效时数据的丢失，并且允许读数据的时候充分利用多个机架的带宽。这种策略设置可以将副本均匀分布在集群中，有利当组件失效情况下的负载均衡。但是，增加了写代价。

大多数情况下，副本系数为 3，。图 6-27 为 HDFS 的副本分配图，HDFS 的存放策略是一个副本存放在本地机架的节点上，一个副本存放在同一机架的另一节点上，最后一个副本存放在不同机架的节点上，一个数据节点内最多放一个此块副本。这种策略减少了机架间的数据传输提高了写操作的效率。机架的错误远远比节点的错误少，所以这个策略不会影响到数据的可靠性和可用性。与此同时，数据块只存放在两个不同机架上，减少了读取数据所需的时间。

9．副本选择

为了降低整体的带宽消耗和读取延时，HDFS 会尽量让读取程序读取离它的节点上副本。如果读取程序同一机架上有一副本，则读取该副本。如果 HDFS 集群跨越多个数据中心，那么客户端首先读取本地数据中心的副本。

10．多副本写策略

当客户端当 HDFS 文件写入数据时，一开始是写到本地临时文件中，假如副本系数设置为 3，当本地临时文件累积到一个数据块大小时，客户端会从 Namenode 获取一个 DataNode 列表用于存放副本，然后客户端开始向第一个 DataNode 传输数据，第一个 DataNode 一小部分一小部分的接收数据，将每一部分写入本地仓库，并同时传输该部分到列表中第二个 DataNode。第二个 DataNode 一样，一小部分一小部分的接收数据，写入本地仓库，并同时传送给第三个 DataNode。最后，第三个 DataNode 接收数据并存储在

本地。因此，DataNode 能流水线式地从前一个节点接收数据，并在同时转发给下一个节点，数据以流水线的方式从前一个 DataNode 复制到下一个。

11．HDFS 元数据组织

- HDFS 元数据及其修改的存放位置

HDFS 系统的元数据存放于内存之中，由 Namenode 管理。元数据记录了文件名，副本个数以及存放位置等信息，以这种存储方式将整个系统数据联系起来。元数据的修改存放在 Editlog 之中，存于本地 Namenode 之上。在 Namenode 的元数据全部存放贮存之中，且没有文件系统元数据的页请求。在元数据类型方面，有文件系列的映射，有每个文件的块系列，有每个文件的属性等类型，事务日志记录了文件的创建、文件删除等信息。

Namenode 上保存着 HDFS 的命名空间，对于文件系统元数据产生修改的操作将会以 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。整个系统的 namespace，包括数据块到文件的映射、文件的属性等。都放在 FsImage 的文件中，这个文件也是放在 Namenode 所在的本地文件系统上。

- HDFS 元数据的持久化

Namenode 在内存中保存着整个文件系统的 namespace 和文件数据块映射的映像。这个关键的元数据设计很紧凑，因而有一个 4G 内存的 Namenode 支撑足够大量的文件和目录。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 的事务已经作用于 FsImage 上了。这个过程称为一个检查点，在当前现实中，检查点只发生在 Namenode 启动时。

DataNode 将 HDFS 数据以文件的形式存储在本地文件系统中，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文件中。DataNode 并不在同一目录创建所有文件。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，然后作为报告发送给 Namenode。

12． HDFS 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。下面将分别对 HDFS 的措施进行分析。

- 磁盘数据错误、心跳检测和重新复制

NameNode 周期性地从集群中的每个 DataNode 接受心跳包和块报告，NameNode 可以根据这个报告验证映射和其他文件系统元数据。收到心跳包，说明该 DataNode 工作正常。如果 DataNode 不能发送心跳信息，NameNode 会标记最近没有心跳的 DataNode 为宕机，并且不会给他们发送任何 I/O 请求。寄存在 dead DataNode 上的任何数据将不再

有效。这种心跳检测每三分钟进行一次。DataNode 死亡有可能引起一些 block 副本数据低于指定值，Namenode 不断跟踪需要复制的 block，在任何需要的情况下启动复制。下列情况需要重新复制，当某个 DataNode 节点失效，某个副本遭到损坏，DataNode 的硬盘错误或者文件的副本系数增大。数据复制，选择最近的复制块进行相应，另外安全模式下不发生文件块的复制，但接受心跳报告。

- 数据完整性

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 文件内容的校验检查。当客户端创建一个新的 HDFS 文件时，会计算这个文件每个数据块的校验和，并将此校验和作为一个单独的隐藏文件保存在同一个 HDFS 的 namspace 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是否匹配，如果不匹配，客户端则可从其他 DataNode 中获取该数据块的副本。

- 元数据磁盘错误

FsImage 和 Editlog 是 HDFS 的核心数据结构，如果这些文件损坏了，整个 HDFS 实例都会失效。因此，Namenode 可以配置成支持维护多个 FsImage 和 Eidtlog 的副本。任何对 FsImage 或 Eidtlog 的修改都将同步到副本上。这在元数据组织中讲到了就不细述了。

- 存储空间回收

当用户或应用程序删除某个文件时，这个文件并没有立刻从 HDFS 中删除。实际上，HDFS 会将这个文件重命名转移到/trash 目录中。只要文件还在/trash 中，该文件就可以被迅速的恢复。文件在/trash 中保存的时间是可配置的，当超过这个时间 Namenode 就会将该文件从 namespace 中删除，删除文件会使得该文件相关的数据块被释放。如果用户想恢复删除的文件，可以通过浏览/trash 目录找回文件，/trash 目录仅仅保存了被删除的最后副本。

13. 负载均衡

Hadoop 的 HDFS 集群中容易出现机器与机器之间磁盘利用率不平衡的情况，例如集群中增加新的数据节点，集群中负载不均衡会产生很多问题，例如机器之间无法达到更好网络带宽，磁盘无法利用等等，因此就需要进行负载均衡策略。实际上，在 Hadoop 中包含一个 balance 程序，通过运行这个程序到达负载均衡的状态，例如命令 sh $HADOOP_HOME/bin/start-balancer.sh-t 10%，这个命令中-t 参数后面跟的是 HDFS 达到平衡状态的磁盘使用率偏差值。如果机器与机器之间磁盘使用率偏差小于 10%，那么我们就认为 HDFS 集群已经达到了平衡的状态。

开发人员在开发 Balancer 程序的时候，遵循了以下几点原则：

- 在执行数据重分布的过程中，必须保证数据不能出现丢失，不能改变数据的备份数，

不能改变每一个 rack 中所具备的 block 数量。

- 系统管理员可以通过一条命令启动数据重分布程序或者停止数据重分布程序。
- Block 在移动的过程中，不能暂用过多的资源，如网络带宽。
- 数据重分布程序在执行的过程中，不能影响 name node 的正常工作。

基于这些基本点，目前 Hadoop 数据重分布程序实现的逻辑流程如图 7 所示。

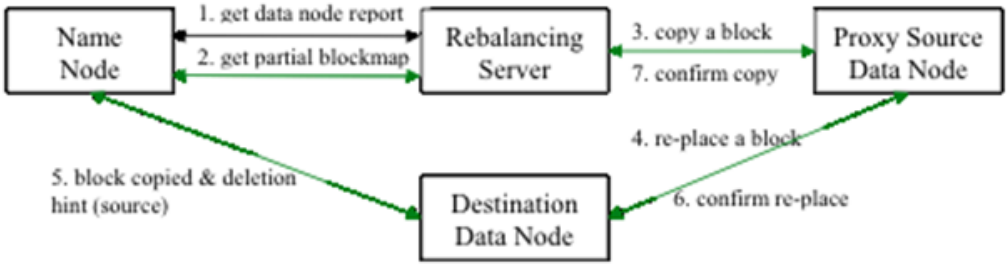

图 7 Hadoop 数据重分布逻辑流程图

Rebalance 程序作为一个独立的进程与 name node 进行分开执行。

1 Rebalance Server 从 Name Node 中获取所有的 Data Node 情况：每一个 Data Node 磁盘使用情况。

2 Rebalance Server 计算哪些机器需要将数据移动，哪些机器可以接受移动的数据。并且从 Name Node 中获取需要移动的数据分布情况。

3 Rebalance Server 计算出来可以将哪一台机器的 block 移动到另一台机器中去。

4，5，6 需要移动 block 的机器将数据移动的目的机器上去，同时删除自己机器上的 block 数据。

7 Rebalance Server 获取到本次数据移动的执行结果，并继续执行这个过程，一直没有数据可以移动或者 HDFS 集群以及达到了平衡的标准为止。

Hadoop 现有的这种 Balancer 程序工作的方式在绝大多数情况中都是非常适合的。

例 现在我们设想如下情况：

- 数据是 3 份备份。
- HDFS 由 2 个 rack 组成。
- 2 个 rack 中的机器磁盘配置不同，第一个 rack 中每一台机器的磁盘空间为 1TB，第二个 rack 中每一台机器的磁盘空间为 10TB。
- 现在大多数数据的 2 份备份都存储在第一个 rack 中。

在这样的一种情况下，HDFS 级群中的数据肯定是不平衡的。现在我们运行 Balancer 程序，但是会发现运行结束以后，整个 HDFS 集群中的数据依旧不平衡：rack1 中的磁盘剩余空间远远小于 rack2。这是因为 Balance 程序的开发原则 1 导致的。简单地说，就是在执行 Balancer 程序的时候，不会将数据中一个 rack 移动到另一个 rack 中，所以就导致了 Balancer 程序永远无法平衡 HDFS 集群的情况。

针对这种情况，可以采取两种方案：

- 继续使用现有的 Balancer 程序，但是修改 rack 中的机器分布。将磁盘空间小的机器分叉到不同的 rack 中去。
- 修改 Balancer 程序，允许改变每一个 rack 中所具备的 block 数量，将磁盘空间告急的 rack 中存放的 block 数量减少，或者将其移动到其他磁盘空间富余的 rack 中去。