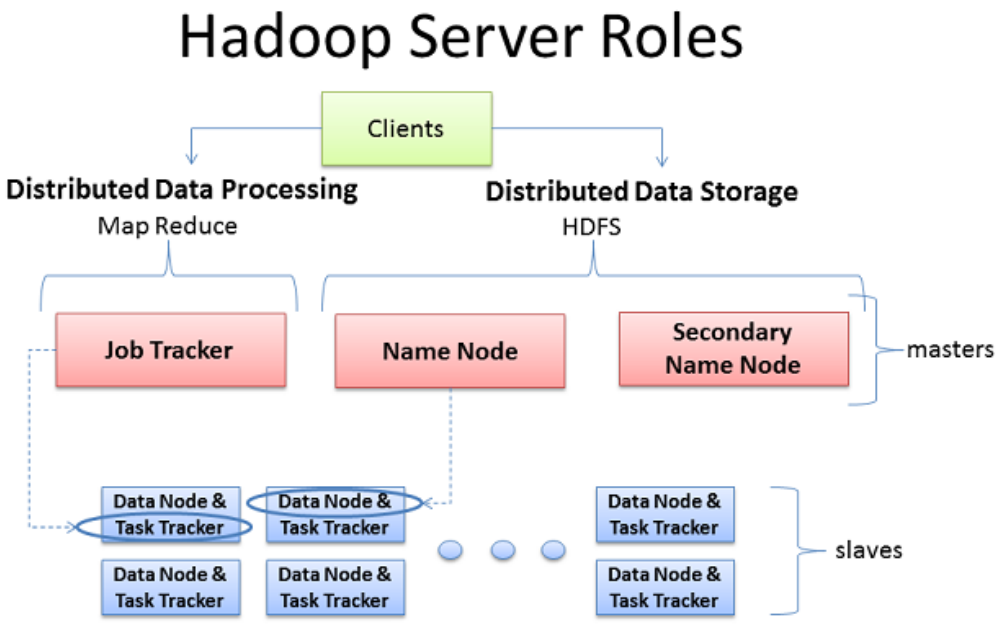
深入理解 Hadoop 集群和网络

云计算和 Hadoop 中网络是讨论得相对比较少的领域。本文原文由 Dell 企业技术专家 Brad Hedlund 撰写，他曾在思科工作多年，专长是数据中心、云网络等。文章素材基于作者自己的研究、实验和 Cloudera 的培训资料。

本文将着重于讨论 Hadoop 集群的体系结构和方法，及它如何与网络和服务器基础设施的关系。最开始我们先学习一下 Hadoop 集群运作的基础原理。



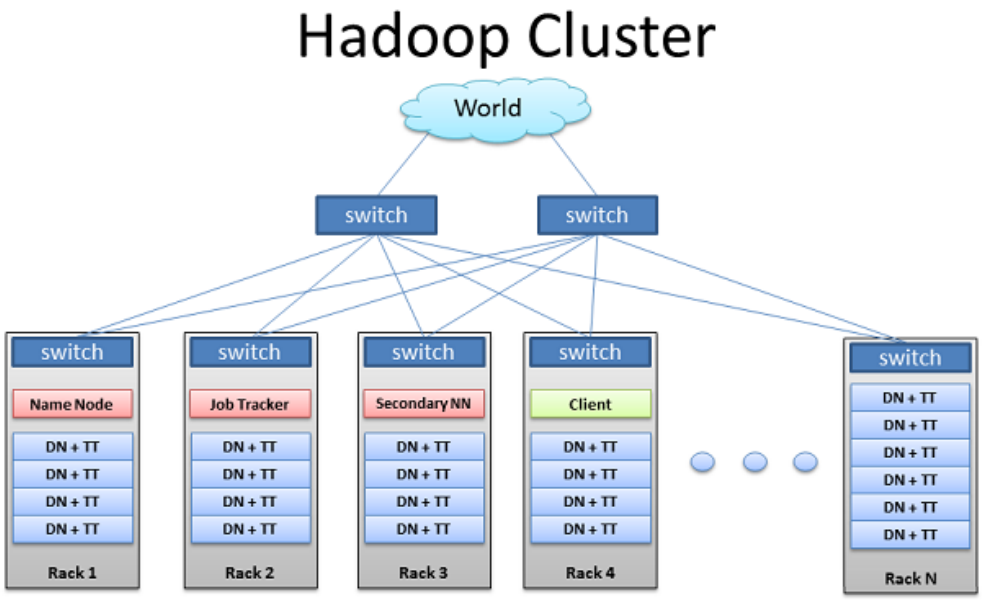
Hadoop 里的服务器角色

Hadoop 主要的任务部署分为 3 个部分，分别是：Client 机器，主节点和从节点。主节点主要负责 Hadoop 两个关键功能模块 HDFS、Map Reduce 的监督。当 Job Tracker 使用 Map Reduce 进行监控和调度数据的并行处理时，名称节点则负责 HDFS 监视和调度。从节点负责

了机器运行的绝大部分，担当所有数据储存和指令计算的苦差。每个从节点既扮演者数据节点的角色又冲当与他们主节点通信的守护进程。守护进程隶属于 Job Tracker，数据节点在归属于名称节点。

Client 机器集合了 Hadoop 上所有的集群设置，但既不包括主节点也不包括从节点。取而代之的是客户端机器的作用是把数据加载到集群中，递交给 Map Reduce 数据处理工作的描述，并在工作结束后取回或者查看结果。在小的集群中（大约 40 个节点）可能会面对单物理设备处理多任务，比如同时 Job Tracker 和名称节点。作为大集群的中间件，一般情况下都是用独立的服务器去处理单个任务。

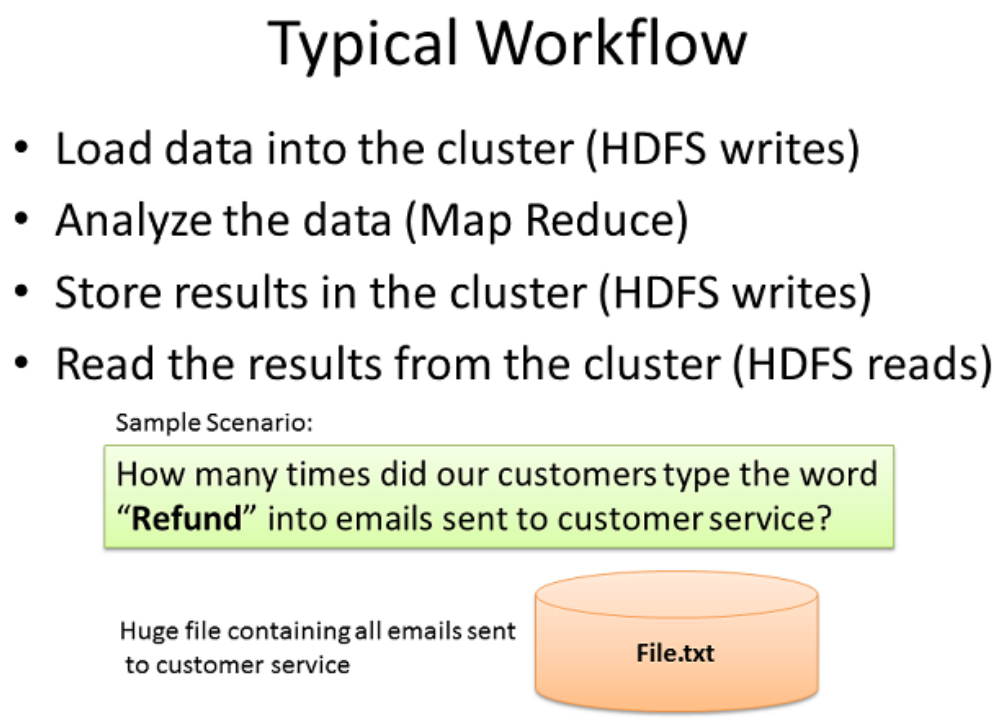
在真正的产品集群中是没有虚拟服务器和管理层的存在的，这样就没有了多余的性能损耗。Hadoop 在 Linux 系统上运行的最好，直接操作底层硬件设施。这就说明 Hadoop 实际上是直接在虚拟机上工作。这样在花费、易学性和速度上有着无与伦比的优势。



Hadoop 集群

上面是一个典型 Hadoop 集群的构造。一系列机架通过大量的机架转换与机架式服务器（不是刀片服务器）连接起来，通常会用 1GB 或者 2GB 的宽带来支撑连接。10GB 的带宽虽然不常见，但是却能显著的提高 CPU 核心和磁盘驱动器的密集性。上一层的机架转换会以相同的带宽同时连接着许多机架，形成集群。大量拥有自身磁盘储存器、CPU 及 DRAM 的服务器将成为从节点。同样有些机器将成为主节点，这些拥有少量磁盘储存器的机器却有着更快的 CPU 及更大的 DRAM。

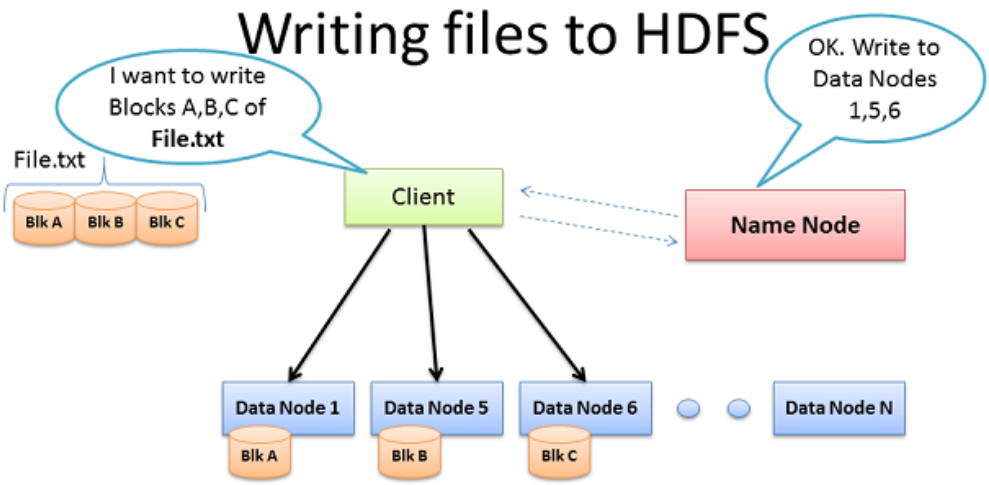
下面我们来看一下应用程序是怎样运作的吧：



adoop 的工作流程

在计算机行业竞争如此激烈的情况下，究竟什么是 Hadoop 的生存之道？它又切实的解决了什么问题？简而言之，商业及政府都存在大量的数据需要被快速的分析和处理。把这些大块的数据切开，然后分给大量的计算机，让计算机并行的处理这些数据 — 这就是 Hadoop 能做的。

下面这个简单的例子里，我们将有一个庞大的数据文件（给客服部门的电子邮件）。我想快速的截取下“Refund”在邮件中出现的次数。这是个简单的字数统计练习。Client 将把数据加载到集群中（File.txt），提交数据分析工作的描述（word cout），集群将会把结果储存到一个新的文件中（Results.txt），然后 Client 就会读结果文档。

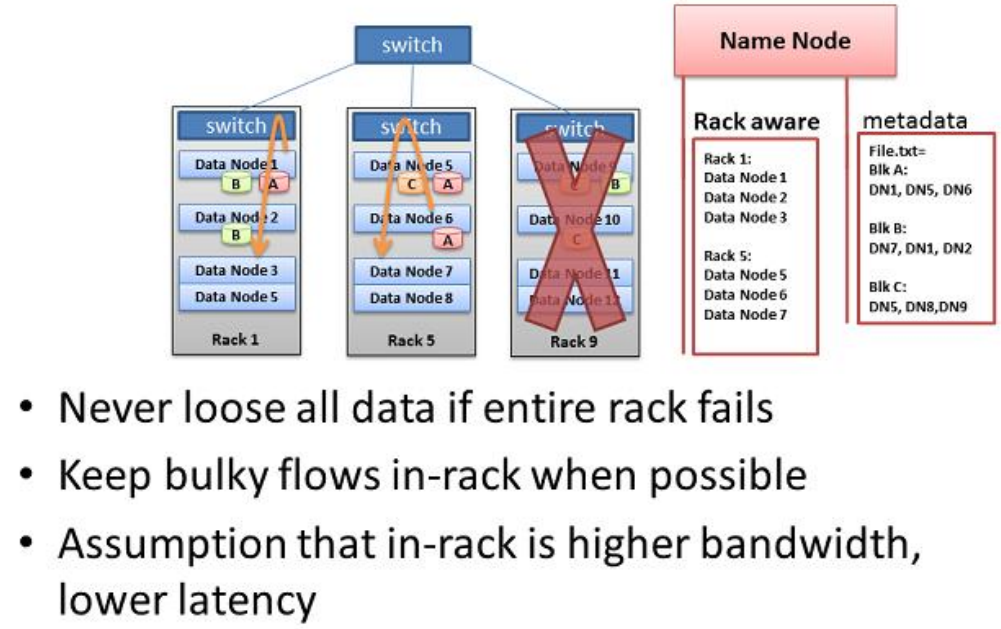


- Client consults Name Node
- Client writes block directly to one Data Node
- Data Nodes replicates block
- Cycle repeats for next block

向 HDFS 里写入 File

Hadoop 集群在没有注入数据之前是不起作用的，所以我们先从加载庞大的 File.txt 到集群中开始。首要的目标当然是数据快速的并行处理。为了实现这个目标，我们需要竟可能多的机器同时工作。最后，Client 将把数据分成更小的模块，然后分到不同的机器上贯穿整个集群。模块分的越小，做数据并行处理的机器就越多。同时这些机器机器还可能出故障，所以为了避免数据丢失就需要单个数据同时在不同的机器上处理。所以每块数据都会在集群上被重复的加载。Hadoop 的默认设置是每块数据重复加载 3 次。这个可以通过 hdfs-site.xml 文件中的 dfs.replication 参数来设置。

Client 把 File.txt 文件分成 3 块。Cient 会和名称节点达成协议（通常是 TCP 9000 协议）然后得到将要拷贝数据的 3 个数据节点列表。然后 Client 将会把每块数据直接写入数据节点中（通常是 TCP 50010 协议）。收到数据的数据节点将会把数据复制到其他数据节点中，循环只到所有数据节点都完成拷贝为止。名称节点只负责提供数据的位置和数据在族群中的去处（文件系统元数据）。



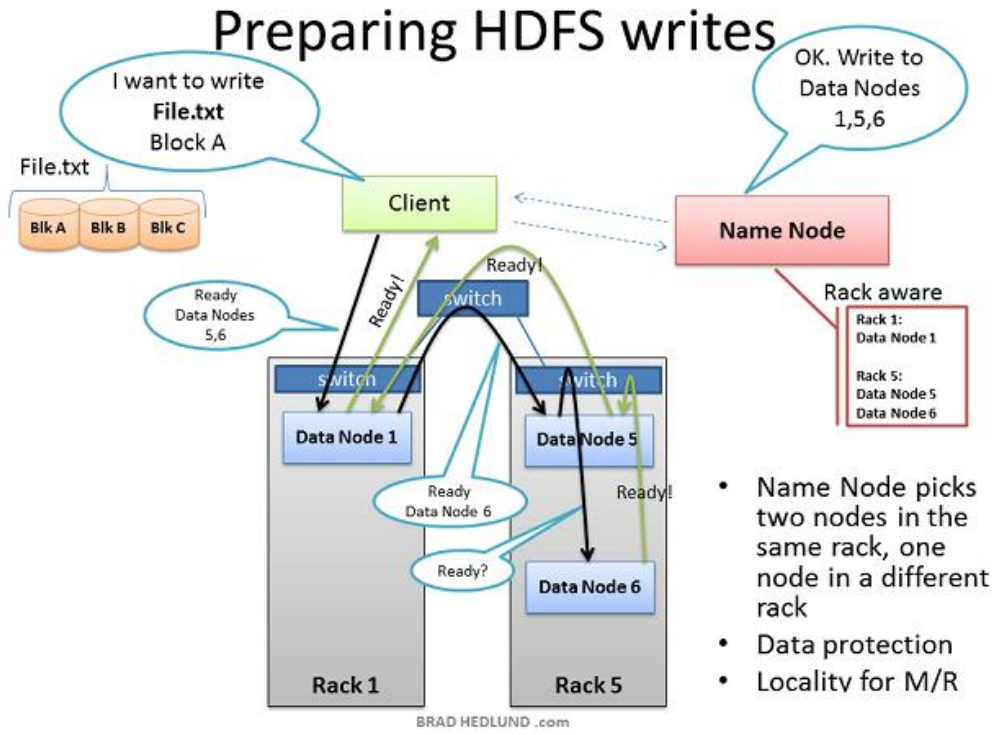
Hadoop 的 Rack Awareness

Hadoop 还拥有“Rack Awareness”的理念。作为 Hadoop 的管理员，你可以在集群中自行的定义从节点的机架数量。但是为什么这样做会给你带来麻烦呢？两个关键的原因是：数据损失预防及网络性能。别忘了，为了防止数据丢失，每块数据都会拷贝在多个机器上。假如同一块数据的多个拷贝都在同一个机架上，而恰巧的是这个机架出现了故障，那么这带来的绝对是一团糟。为了阻止这样的事情发生，则必须有人知道数据节点的位置，并根据实际情况在集群中作出明智的位置分配。这个人就是名称节点。

假使通个机架中两台机器对比不同机架的两台机器会有更多的带宽更低的延时。大部分情况下这是真实存在的。机架转换的上行带宽一般都低于其下行带宽。此外，机架内的通信的延时一般都低于跨机架的（也不是全部）。那么假如 Hadoop 能实现“Rack Awareness”的理念，那么在集群性能上无疑会有着显著的提升！是的，它真的做到了！太棒了，对不对？

但是扫兴的事情发生了，首次使用你必须手动的去定义它。不断的优化，保持信息的准确。假如机架转换能够自动的给名称节点提供它的数据节点列表，这样又完美了？或者反过来，数据节点可以自行的告知名称节点他们所连接的机架转换，这样也的话也同样完美。

在括补结构中网络中，假如能知道名称节点可以通过 OpenFlow 控制器查询到节点的位置，那无疑是更加令人兴奋的。

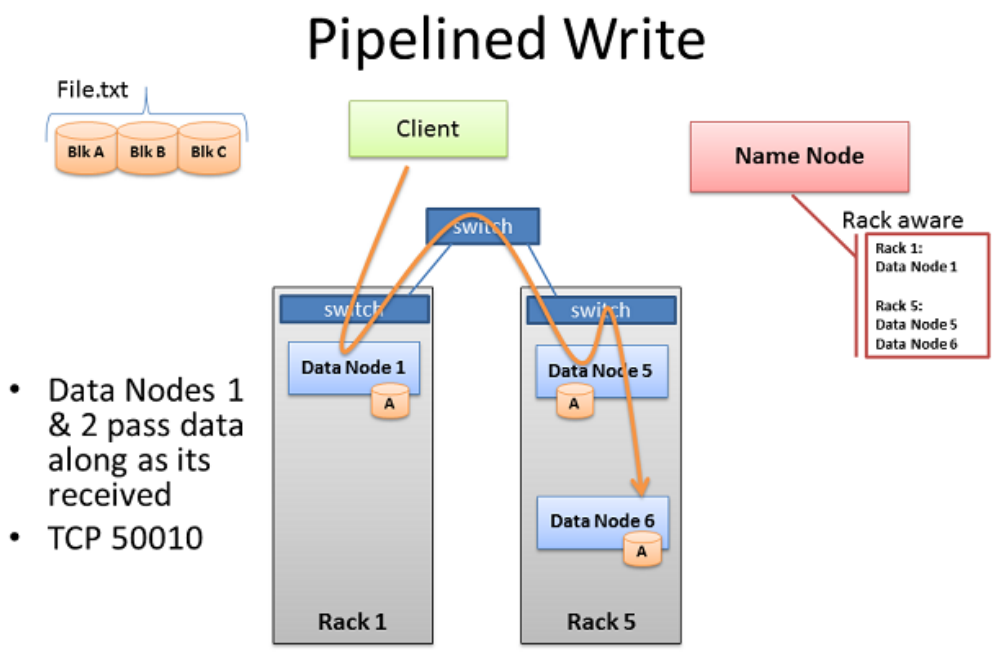


准备 HDFS 写入

现在 Client 已经把 File.txt 分块并做好了向集群中加载的准备，下面先从 Block A 开始。Client 向名称节点发出写 File.txt 的请求，从名称节点处获得通行证，然后得到每块数据目标数据节点的列表。名称节点使用自己的 Rack Awareness 数据来改变数据节点提供列表。核心规则就是对于每块数据 3 份拷贝，总有两份存在同一个机架上，另外一份则必须放到另一个机架上。所以给 Client 的列表都必须遵从这个规则。

在 Client 将 File.txt 的“Block A”部分写入集群之前，Client 还期待知道所有的目标数据节点是否已准备就绪。它将取出列表中给 Block A 准备的第一个数据节点，打开 TCP 50010 协议，并告诉数据节点，注意！准备好接收 1 块数据，这里还有一份列表包括了数据节点 5 和数据节点 6，确保他们同样已准备就绪。然后再由 1 传达到 5，接着 5 传达到 6。

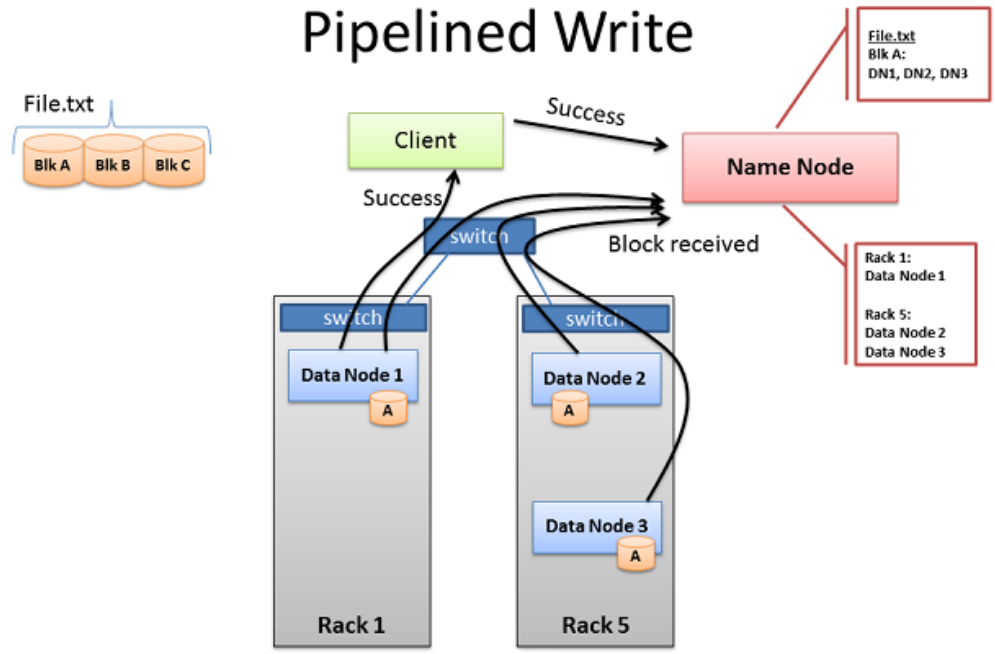
数据节点将从同样的 TCP 通道中响应上一级的命令，只到 Client 收到原始数据节点 1 发送的的“就绪”。只到此刻，Client 才真正的准备在集群中加载数据块。



HDFS 载入通道

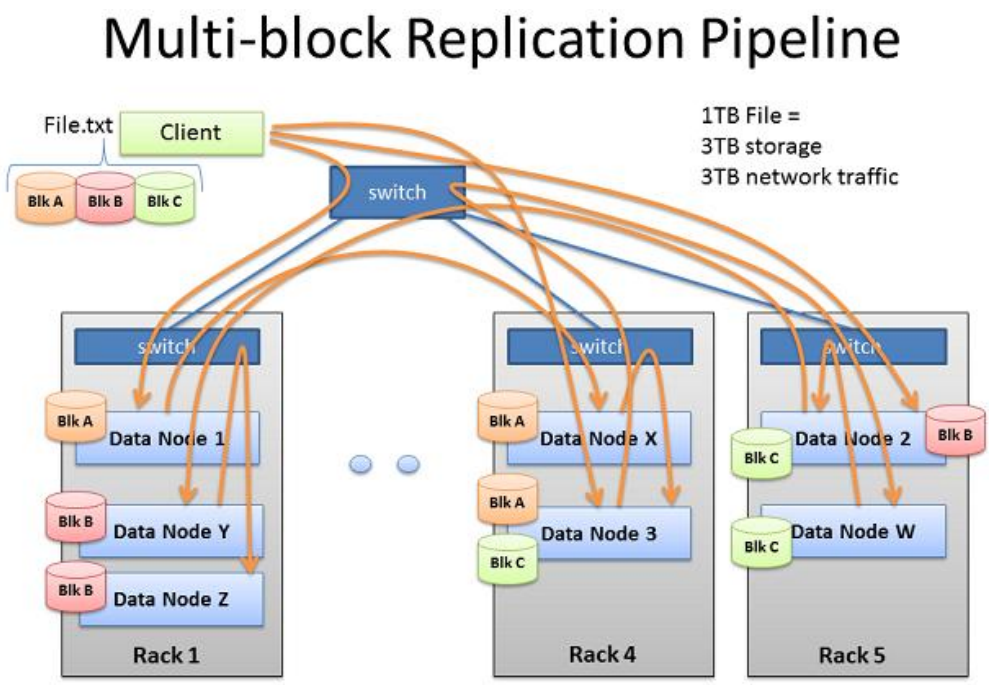
当数据块写入集群后，3 个（当然数据节点个数参照上文的设置）数据节点将打开一个同步通道。这就意味着，当一个数据节点接收到数据后，它同时将在通道中给下一个数据节点送上一份拷贝。

这里同样是一个借助 Rack Awareness 数据提升集群性能的例子。注意到没有，第二个和第三个数据节点运输在同一个机架中，这样他们之间的传输就获得了高带宽和低延时。只到这个数据块被成功的写入 3 个节点中，下一个就才会开始。

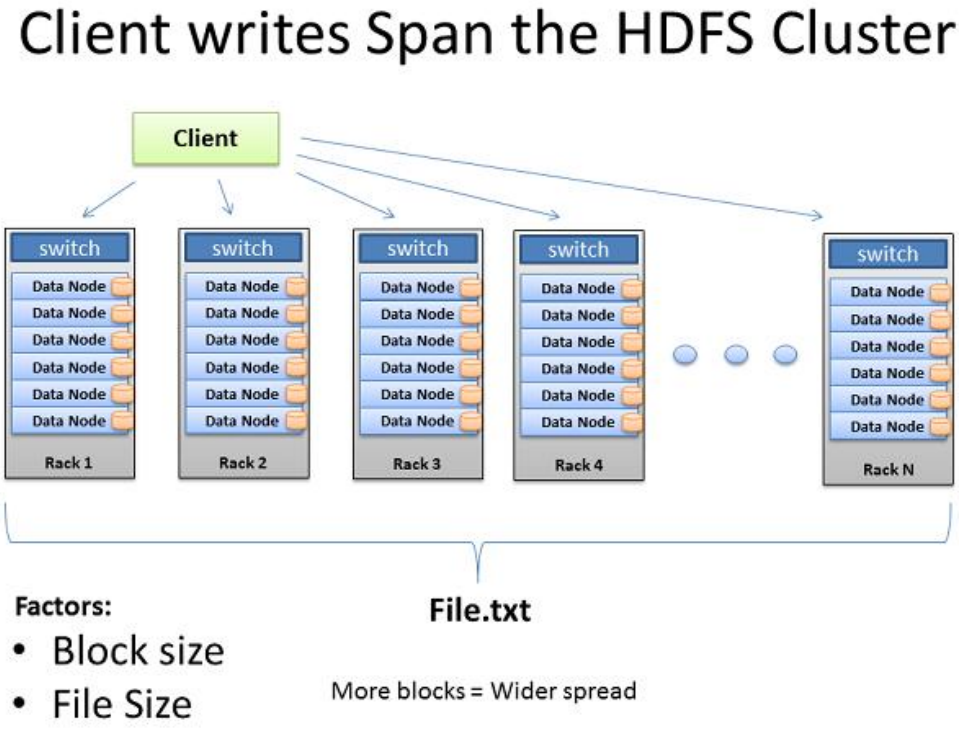


HDFS 通道载入成功

当 3 个节点都成功的接收到数据块后，他们将给名称节点发送个“Block Received”报告。并向通道返回“Success”消息，然后关闭 TCP 回话。Client 收到成功接收的消息后会报告给名称节点数据已成功接收。名称节点将会更新它元数据中的节点位置信息。Client 将会开启下一个数据块的处理通道，只到所有的数据块都写入数据节点。



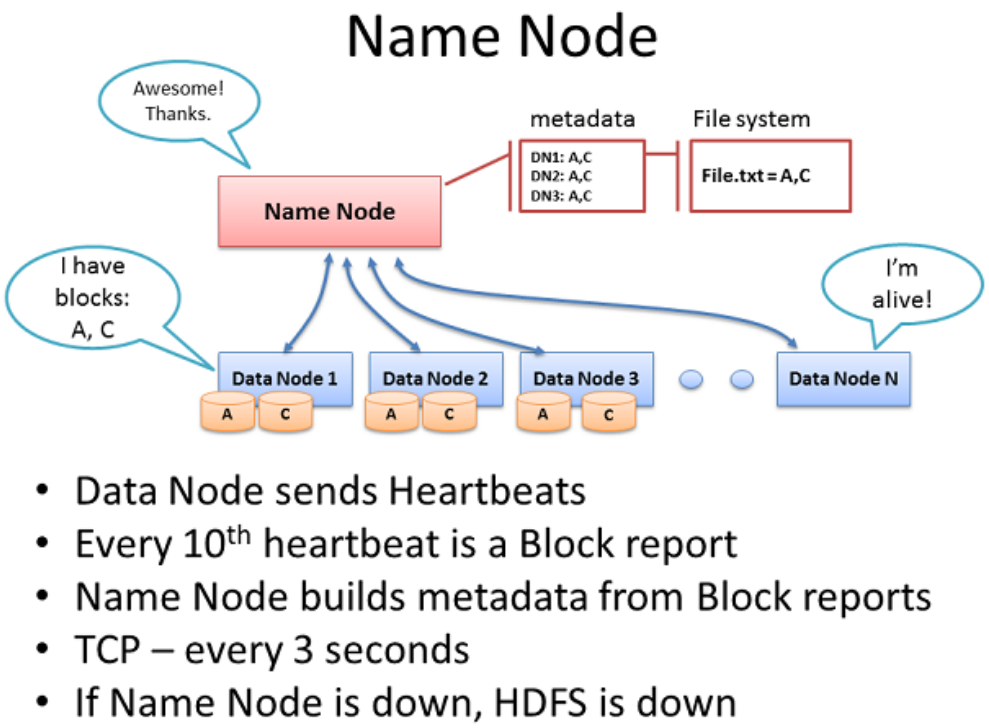
Hadoop 会使用大量的网络带宽和存储。我们将代表性的处理一些 TB 级别的文件。使用 Hadoop 的默认配置，每个文件都会被复制三份。也就是 1TB 的文件将耗费 3TB 的网络传输及 3TB 的磁盘空间。



Client 写入跨度集群

每个块的复制管道完成后的文件被成功写入到集群。如预期的文件被散布在整个集群的机器，每台机器有一个相对较小的部分数据。个文件的块数越多，更多的机器的数据有可能传播。更多的 CPU 核心和磁盘驱动器，意味着数据能得到更多的并行处理能力和更快的结果。这是建造大型的、宽的集群的背后的动机，为了数据处理更多、更快。当机器数增加和集群增宽时，我们的网络需要进行适当的扩展。

扩展集群的另一种方法是深入。就是在你的机器扩展更多个磁盘驱动器和更多的 CPU 核心，而不是增加机器的数量。在扩展深度上，你把自己的注意力集中在用较少的机器来满足更多的网络 I/O 需求上。在这个模型中，你的 Hadoop 集群如何过渡到万兆以太网节点成为一个重要的考虑因素。

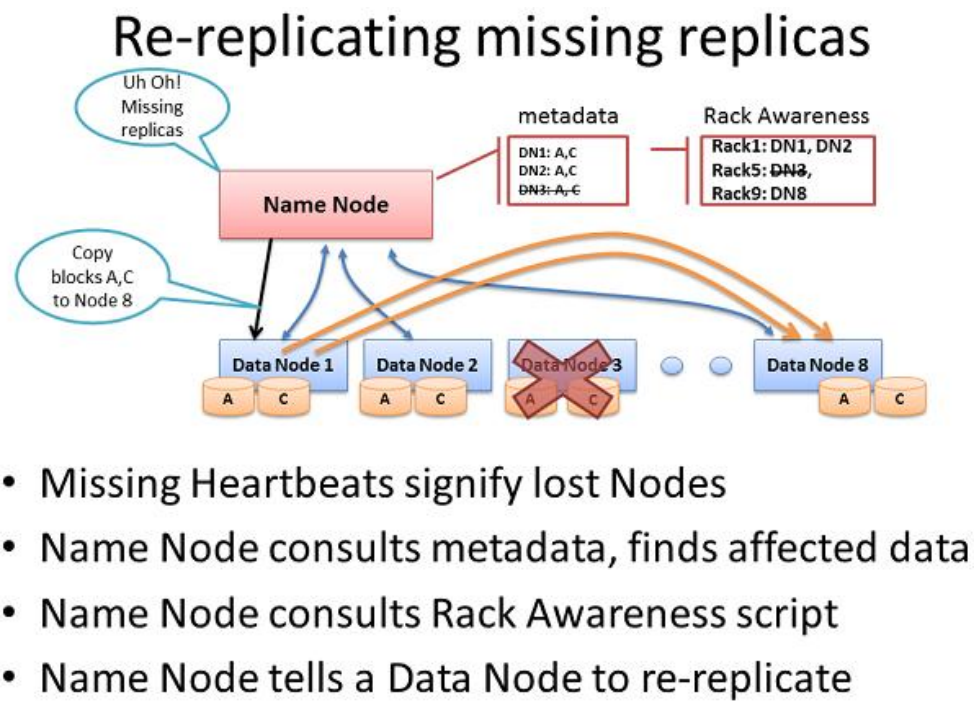


名称节点

名称节点包含所有集群的文件系统元数据和监督健康状况的数据节点以及协调对数据的访问。这个名字节点是 HDFS 的中央控制器。它本身不拥有任何集群数据。这个名称节点只知道块构成一个文件，并在这些块位于集群中。

数据节点每 3 秒通过 TCP 信号交换向名称节点发送检测信号，使用相同的端口号定义名称节点守护进程，通常 TCP 9000。每 10 个检测信号作为一个块报告，那里的数据节点告知它的所有块的名称节点。块报告允许名称节点构建它的元数据和确保第三块副本存在不同的机架上存在于不同的节点上。

名称节点是 Hadoop 分布式文件系统（HDFS）的一个关键组件。没有它，客户端将无法从 HDFS 写入或读取文件，它就不可能去调度和执行 Map Reduce 工作。正因为如此，用双电源、热插拔风扇、冗余网卡连接等等来装备名称节点和配置高度冗余的企业级服务器使一个不错的想法。

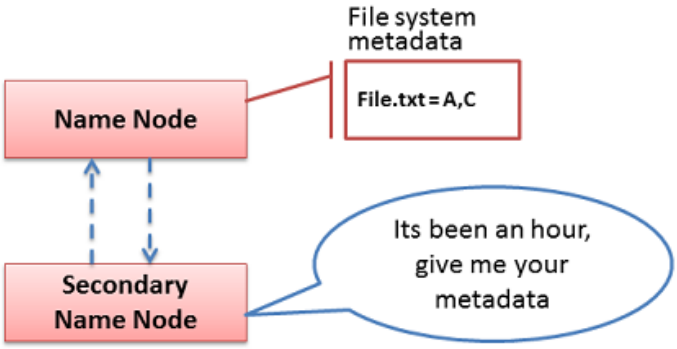


重新复制缺失副本

如果名称节点停止从一个数据节点接收检测信号，假定它已经死亡，任何数据必须也消失了。基于块从死亡节点接受到报告，这个名称节点知道哪个副本连同节点块死亡，并可决定重新复制这些块到其他数据节点。它还将参考机架感知数据，以保持在一个机架内的两个副本。

考虑一下这个场景，整个机架的服务器网络脱落，也许是因为一个机架交换机故障或电源故障。这个名称节点将开始指示集群中的其余节点重新复制该机架中丢失的所有数据块。如果在那个机架中的每个服务器有 12TB 的数据，这可能是数百个 TB 的数据需要开始穿越网络。

Secondary Name Node



- Not a hot standby for the Name Node
- Connects to Name Node every hour*
- Housekeeping, backup of Name Node metadata
- Saved metadata can rebuild a failed Name Node

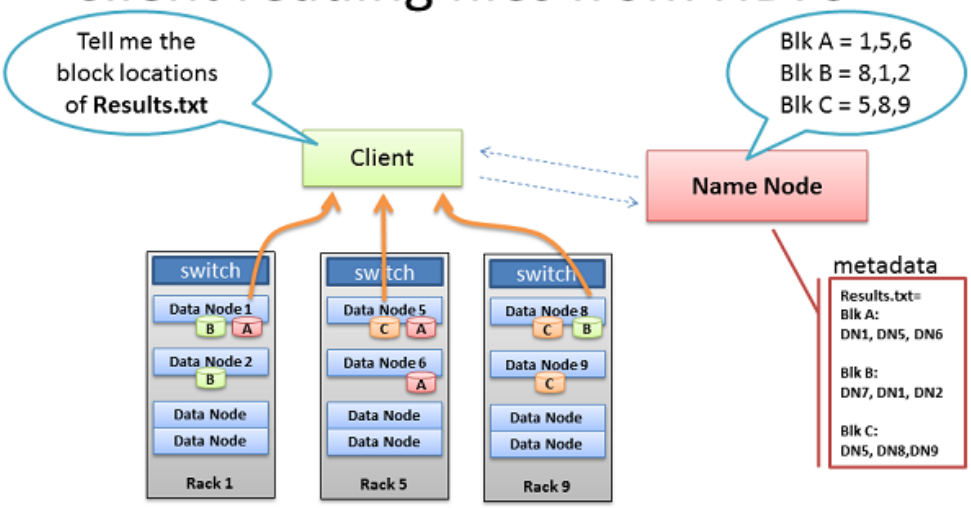
二级名称节点

Hadoop 服务器角色被称为二级名称节点。一个常见的误解是，这个角色为名称节点提供了一个高可用性的备份，这并非如此。

二级名称节点偶尔连接到名字节点，并获取一个副本的名字节点内存中的元数据和文件用于存储元数据。二级名称节点在一个新的文件集中结合这些信息，并将其递送回名称节点，同时自身保留一份复本。

如果名称节点死亡，二级名称节点保留的文件可用于恢复名称节点。

Client reading files from HDFS

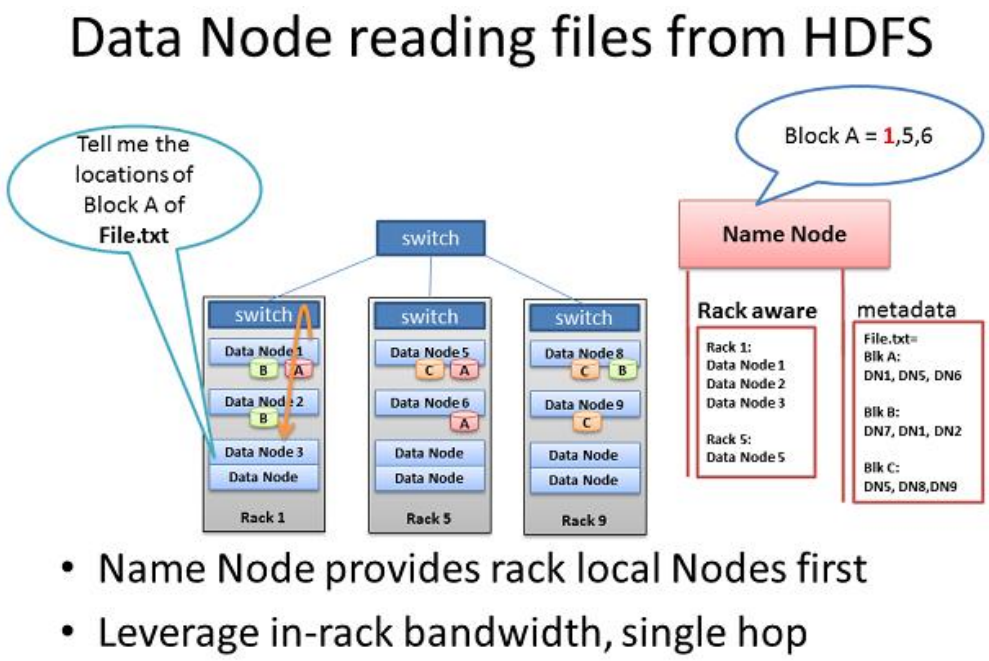


- Client receives Data Node list for each block
- Client picks first Data Node for each block
- Client reads blocks sequentially

从 HDFS 客户端读取

当客户想要从 HDFS 读取一个文件，它再一次咨询名称节点，并要求提供文件块的位置。

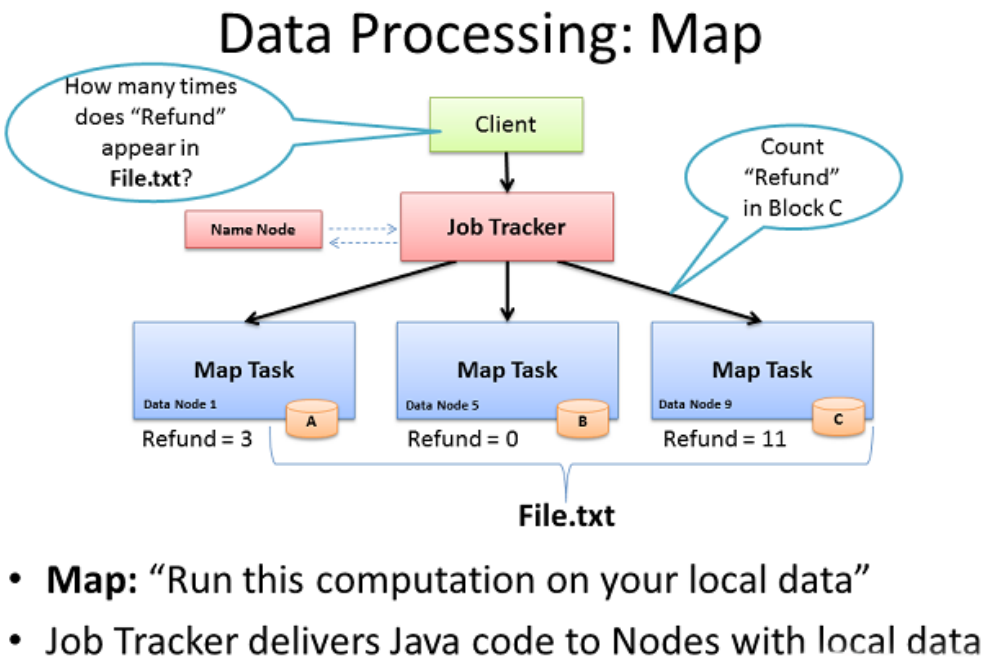
客户从每个块列表选择一个数据节点和用 TCP 的 50010 端口读取一个块。直到前块完成，它才会进入下一个块。



从 HDFS 中读取数据节点

有些情况下，一个数据节点守护进程本身需要从 HDFS 中读取数据块。一种这样的情况是数据节点被要求处理本地没有的数据，因此它必须从网络上的另一个数据节点检索数据，在它开始处理之前。

另一个重要的例子是这个名称节点的 Rack Awareness 认知提供了最佳的网络行为。当数据节点询问数据块里名称节点的位置时，名称节点将检查是否在同一机架中的另一种数据节点有数据。如果是这样，这个名称节点从检索数据里提供了机架上的位置。该流程不需要遍历两个以上的交换机和拥挤的链接找到另一个机架中的数据。在机架上检索的数据更快，数据处理就可以开始的更早，,工作完成得更快。



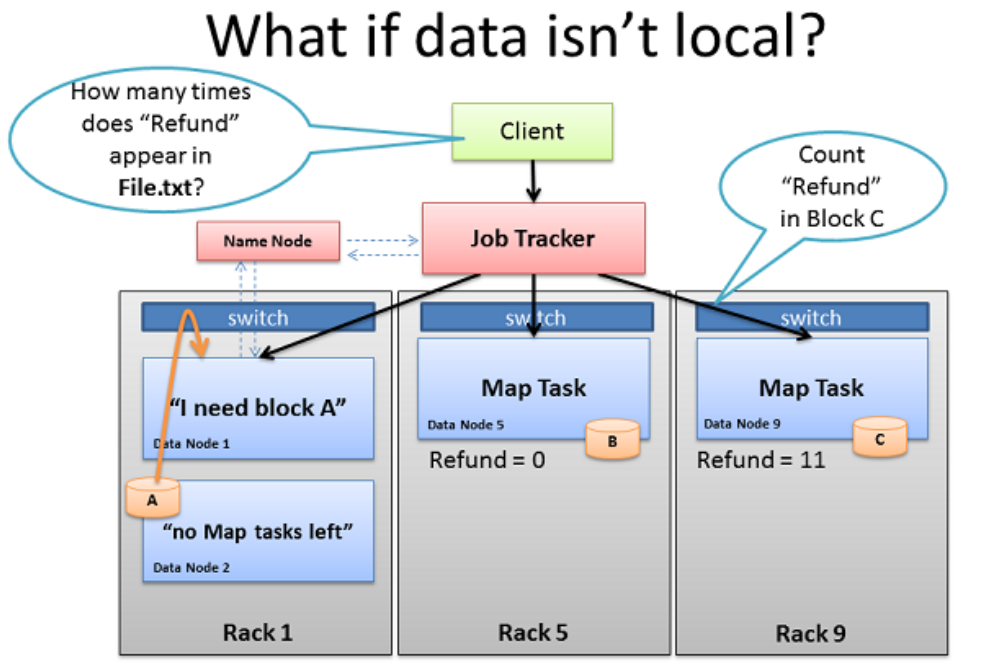
Map Task

现在 file.txt 在我的机器集群中蔓延，我有机会提供极其快速和高效的并行处理的数据。包含 Hadoop 的并行处理框架被称为 Map Reduce，模型中命名之后的两个步骤是 Map 和 Reduce。

第一步是 Map 过程。这就是我们同时要求我们的机器他们本地的数据块上来运行一个计算。在这种情况下，我们要求我们的机器对“Refund”这个词在 File.txt 的数据块中出现的次数进行计数。

开始此过程，客户端机器提交 Map Reduce 作业的 Job Tracker，询问“多少次不会在 File.txt 中出现 Refund”(意译 Java 代码)。Job Tracker 查询名称节点了解哪些数据节点有 File.txt 块。Job Tracker 提供了这些节点上运行的 Task Tracker 与 Java 代码需要在他们的本地数据上执行的 Map 计算。这个 Task Tracker 启动一个 Map 任务和监视任务进展。这 Task Tracker 提供了检测信号并向 Job Tracker 返回任务状态。

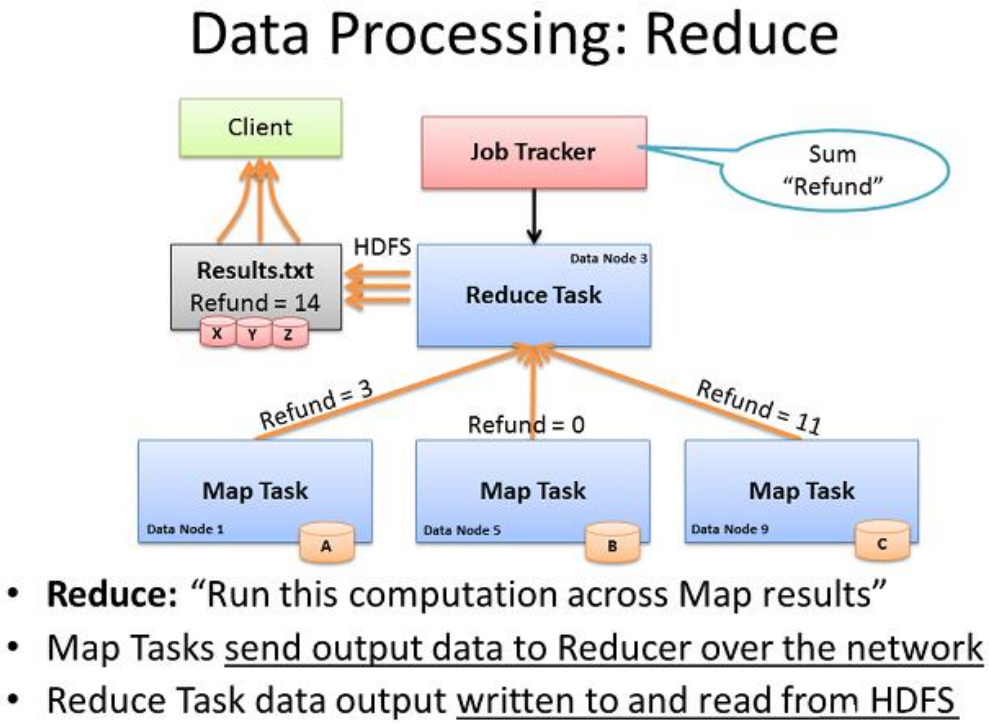
每个 Map 任务完成后，每个节点在其临时本地存储中存储其本地计算的结果。这被称作“中间数据”。 下一步将通过网络传输发送此中间数据到 Reduce 任务最终计算节点上运行。



Map Task 非本地

虽然 Job Tracker 总是试图选择与当地数据做 Map task 的节点，但它可能并不总是能够这样做。其中一个原因可能是因为所有的节点与本地数据，已经有太多的其他任务运行，并且不能接受了。

在这种情况下，Job Tracker 将查阅名称节点的 Rack Awareness 知识，可推荐同一机架中的其他节点的名称节点。作业跟踪器将把这个任务交给同一机架中的一个节点，节点去寻找的数据时，它需要的名称节点将指示其机架中的另一个节点来获取数据。



Reduce Task 从 Map Tasks 计算接收到的数据

第二阶段的 Map Reduce 框架称为 Reduce。机器上的 Map 任务已经完成了和生成它们的中间数据。现在我们需要收集所有的这些中间数据，组合并提纯以便进一步处理，这样我们会有一个最终结果。

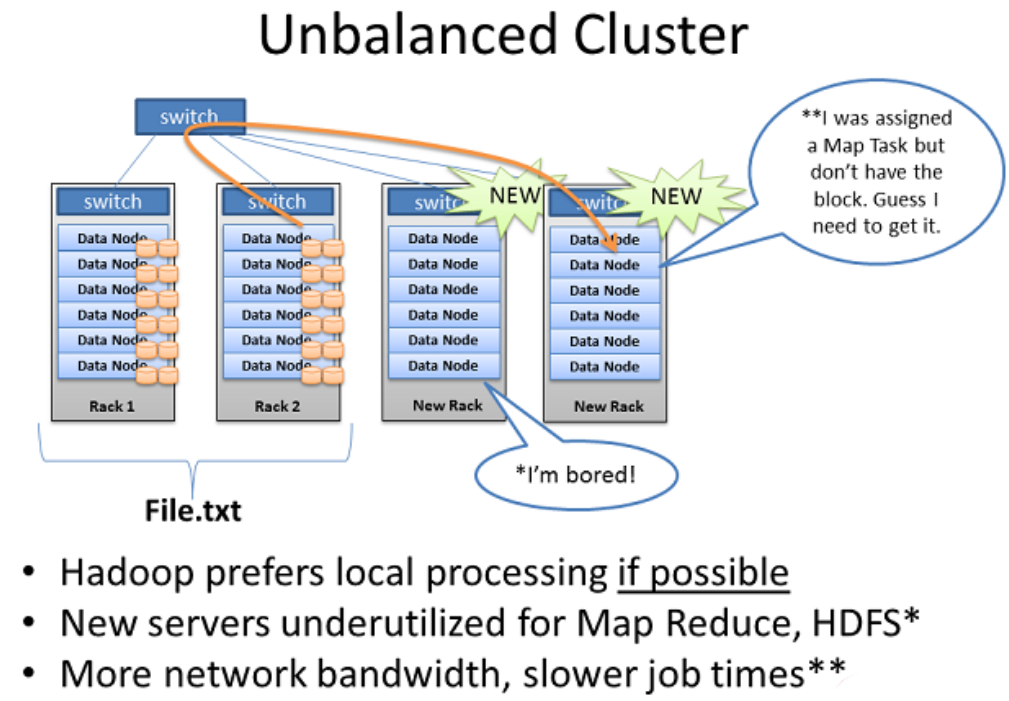
Job Tracker 在集群中的任何一个节点上开始一个 Reduce 任务，并指示 Reduce 任务从所有已完成的 Map 任务中获取中间数据。Map 任务可能几乎同时应对 Reducer，导致让你一下子有大量的节点发送 TCP 数据到一个节点。这种流量状况通常被称为“Incast”或者“fan-in”。对于网络处理大量的 incast 条件，其重要的网络交换机拥有精心设计的内部流量管理能力，以及足够的缓冲区（不太大也不能太小）。

Reducer 任务现在已经从 Map 任务里收集了所有的中间数据，可以开始最后的计算阶段。在本例中，我们只需添加出现“Refund”这个词的总数，并将结果写入到一个名为 Results 的 txt 文件里。

这个名为 Results 的 txt 文件，被写入到 HDFS 以下我们已经涵盖的进程中，把文件分成块，流水线复制这些块等。当完成时，客户机可以从 HDFS 和被认为是完整的工作里读取 Results.txt。

我们简单的字数统计工作并不会导致大量的中间数据在网络上传输。然而，其他工作可能会产生大量的中间数据，比如对 TB 级数据进行排序。

如果你是一个勤奋的网络管理员，你将了解更多关于 Map Reduce 和你的集群将运行的作业类型，以及作业类型如何影响你的网络流量。如果你是一个 Hadoop 网络明星，你甚至能够提出更好的代码来解决 Map Reduce 任务，以优化网络的性能，从而加快工作完工时间。

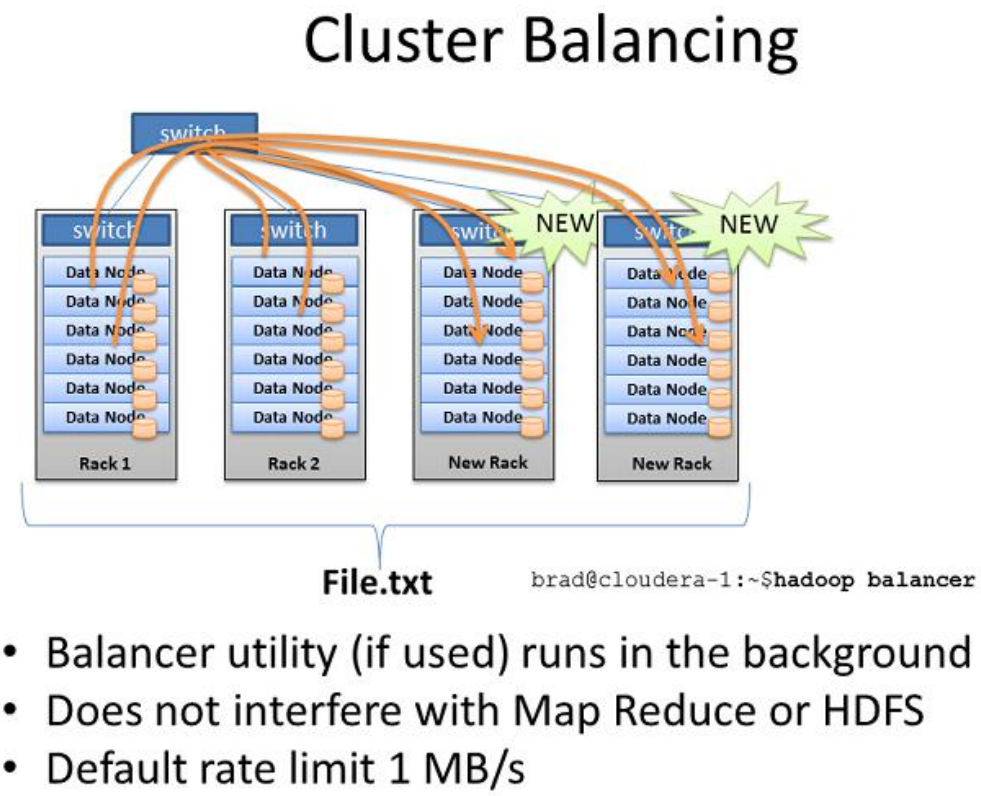


不平衡的 Hadoop 集群

Hadoop 可以为你的组织提供一个真正的成功，它让你身边的数据开发出了很多之前未发现的业务价值。当业务人员了解这一点，你可以确信，很快就会有更多的钱为你的 Hadoop 集群购买更多机架服务器和网络。

当你在现有的 Hadoop 集群里添加新的机架服务器和网络这种情况时，你的集群是不平衡的。在这种情况下，机架 1&2 是我现有的包含 File.txt 的机架和运行我的 Map Reduce 任务的数据。当我添加了两个新的架到集群，我的 File.txt 数据并不会自动开始蔓延到新的机架。

新的服务器是闲置的，直到我开始加载新数据到集群中。此外,如果机架 1&2 上服务器都非常繁忙，Job Tracker 可能没有其他选择，但会指定 File.txt 上的 Map 任务到新的没有本地数据的服务器上。新的服务器需要通过网络去获取数据。作为结果，你可能看到更多的网络流量和较长工作完成时间。



Hadoop 集群均衡器

为了弥补集群的平衡性，Hadoop 还包含了均衡器。

Balancer 目光聚焦于节点间有效储存的差异，力所能及的将平衡维持在一定的临界值上。假如发现剩余大量储存空间的节点，Balancer 将找出储存空间剩余量少的节点并把数据剪切到有大量剩余空间的节点上。只有的终端上输入指令 Balancer 才会运行，当接收到终端取消命令或者终端被关闭时，Balancer 将会关闭。

Balancer 可以调用的网络带宽很小，默认只有 1MB/s。带宽可以通过 hdfs-site.xml 文件中的 dfs.balance.bandwidthPerSec 参数来设置。

Balancer 是集群的好管家。没当有新机组添加时候就会用到它，甚至一经开启就会运行整个星期。给均衡器低带宽可以让它保持着长时间的运行。

个人认为假如均衡器能成为 Hadoop 的核心而不是只是一项功能，那样一定会比较有意思！