第一章　绪论

1.1　背景介绍

在计算机历史上，每一次技术的进化都伴随着数据的爆炸，这些海量增长的数据日益成为人们所关注的焦点。通过这些海量数据，服务提供商可以分析并提取出用户的喜好，从而更加快捷、精准的向用户推送服务，而在计算机行业内，如何高效快速的处理这些海量数据，一直是学术界所热议的话题。

大量的数据要求程序运行在性能极高的平台上运行，传统的大型机异常昂贵。在这种高成本的压力下催生了高性能集群计算的概念，如高性能计算 (High Perfermance Computing) 集群，简称 HPC 集群。集群是将被称为节点的多个计算机系统结合起来实现远远超出任何一个普通客户端 PC 或服务器所能达到的性能水平。

计算集群以并行计算的概念为基础，它将整个任务分解成多个独立的任务分发到各个节点进行处理。这样做可以获得更大的性能，因为多个系统会共同工作来处理一个单独的大型任务请求。一个典型的集群都会有一个充当分解和收集工作结果接口的“头节点”，还有多个处理各种计算的“计算节点”。

但传统的 HPC 使用的是资源共享结构，即共享存储。这种结构使得 HPC 容错性差，必须使用稳定的刀片服务器、高速网、SAN 存储等设备。因此价格贵且扩展性差。同时，使用 HPC 编程需要考虑内容和程序稳定性、分布一致性及其效率问题，编程难度极大。故 HPC 仅适合于实时、细粒度计算，如银行、政府等企业。

MapReduce 是 Google 公司在 2004 年在 OSDI 国际会议上提出的一种简单的并行计算模型，它借鉴了函数式编程语言的特点，将大规模数据的分布式处理程序抽象为一个运行在分布式集群上的两个用户自定义函数：Map（映射）函数和 Reduce（化简）函数，从而实现了分布式处理海量数据。如今，MapReduce 已经被广泛应用在大规模数据排序，日志分析，数据挖掘，机器学习等领域，它的出现极大程度简化了编写处理大规模数据的分布式程序的难度，使用户可以忽略底层处理细节的前提下快捷的开发出分布式程序。目前已有很多基于大规模集群环境的 MapReduce 框架的实现，其中使用最广泛的是 Yahoo 基于 Java 实现的 Hadoop。

Hadoop 是一个分布式系统基础架构，由 Apache 基金会开发。其主要功能是为 MapReduce 的实现提供底层支持，包括集群管理、文件存储和容错容灾。Hadoop 实现了 MapReduce 框架，使得 Hadoop 具有能够对大量数据进行分布式处理能力。同时 Hadoop 是可靠的，因为它假设计算元素和存储会失败并维护多个工作数据副本，确保能够针对失败的节点重新分布处理。Hadoop 是高效的，因为它以并行的方式工作，通过并行处理加快处理速度。Hadoop 还是可伸缩的，

能够处理 PB 级数据。此外，Hadoop 依赖于社区服务器，因此它的成本比较低，任何人都可以使用。

本文的研究就是基于 Hadoop 和 MapReduce 之上的。

1.2 国内外现状

MapReduce 编程模型的思想来源于函数式编程语言 Lisp，由 Google 公司于 2004 年提出并首先应用于大型集群。同时，Google 也发表了 GFS、BigTable 等底层系统以应用 MapReduce 模型。在 2007 年，Google’s MapReduce Programming Model-Revisted 论文发表，进一步详细介绍了 Google MapReduce 模型以及 Sazwall 并行处理海量数据分析语言。Google 公司以 MapReduce 作为基石，逐步发展成为全球互联网企业的领头羊。

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 MapReduce 思想。在 2004 年，Doug Cutting 和 Mike Cafarella 实现了 Hadoop 分布式文件系统和 MapReduce 并发布了最初版；2005 年 12 月，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。2011 年 12 月 27 日，Apache Hadoop 团队表示，经历了六年的风雨，Hadoop 已经可以应用于正式生产中，且已经被（包括很多大公司）广泛应用，为了终结关于它是否成熟的争论（有些客户希望在应用前看到版本号是 1.0），因此团队决定直接从 0.20 版跳至 1.0 版。

目前，在企业界和学术界对 Hadoop 的关注度都非常高，全球很多公司和机构都已将 Hadoop 投入生产环境中使用。

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，使用 Hadoop 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。Hadoop 已经成为大公司做分布式集群运行 MapReduce 程序的首选软件。

1.3　本文工作和章节介绍

本文以 Web 访问日志处理程序的设计为主线，介绍并分析了 Hadoop 和 MapReduce 的运作原理。

第一章、绪论。介绍研究背景及国内外现状。

第二章、背景介绍。介绍 MapReduce 和 Hadoop 的原理和发展过程以及 Web 访问日志的格式和作用。

第三章、基于 Hadoop 的 MapReduce 程序设计。介绍 Web 访问日志处理程序的程序功能、程序运行流程、适用于两种 Hadoop 运行方式的程序的开发和运行及其性能测试。

第四章、基于 Hadoop 的 MapReduce 实现的性能分析。通过对第三章中所开发的程序的运行结果分析，找出 Hadoop 及 MapReduce 框架的性能瓶颈。

第五章、全文总结

第二章　开发背景介绍

2.1　MapReduce 简介

2.1.1　什么是 MapReduce

Google 的很多程序员为了处理海量的原始数据，已经实现了数以百计的、专用的计算方法。这些计算方法用来处理大量的原始数据，比如，文档抓取（类似网络爬虫的程序）、Web 请求日志等等；也为了计算处理各种类型的衍生数据，比如倒排索引、Web 文档的图结构的各种表示形势、每台主机上网络爬虫抓取的页面数量的汇总、每天被请求的最多的查询的集合等等。大多数这样的数据处理运算在概念上很容易理解。然而由于输入的数据量巨大，因此要想在可接受的时间内完成运算，只有将这些计算分布在成百上千的主机上。如何处理并行计算、如何分发数据、如何处理错误？所有这些问题综合在一起，需要大量的代码处理，因此也使得原本简单的运算变得难以处理。

为了解决上述复杂的问题，Google 设计了一个新的抽象模型，使用这个抽象模型，我们只要表述我们想要执行的简单运算即可，而不必关心并行计算、容错、数据分布、负载均衡等复杂的细节，这些问题都被封装在了一个库里面。设计这个抽象模型的灵感来自 Lisp 和许多其他函数式语言的 Map 和 Reduce 的原语。我们意识到我们大多数的运算都包含这样的操作：在输入数据的“逻辑”记录上应用 Map 操作得出一个中间 key/value pair 集合，然后在所有具有相同 key 值的 value 值上应用 Reduce 操作，从而达到合并中间的数据，得到一个想要的结果的目的。使用 MapReduce 模型，再结合用户实现的 Map 和 Reduce 函数，用户可以非常容易的实现大规模并行化计算；通过 MapReduce 模型自带的“再次执行”（re-execution）功能，也提供了初级的容灾实现方案。Google 将这种抽象模型做以总结，并于 2004 在 OSDI 国际会议上提出，最终风靡全球。

以 MapReduce 为模型编写的程序能自动地在大规模的普通机器上实现并行化处理。这个系统在运行时需要关注的细节有：分割输入数据，在机群上的调度，机器的错误处理，管理机器之间必要的通信。这样就可以让那些没有并行分布式处理系统经验的程序员使用大量分布式系统的资源。MapReduce 可以灵活调整，使其在由普通机器组成的机群上实现运行：一个典型的 MapReduce 可以计算处理几千台机器上的以 TB 计算的数据。程序员发现这个系统非常好用：已经实现了数以百计的 MapReduce 程序，每天在 Google 的机群上都有 1000 多个 MapReduce 程序在执行。

2.1.2　MapReduce 的实现过程

MapReduce 的实现过程并不复杂，图2.1向我们展示了 MapReduce 操作的全部流程。

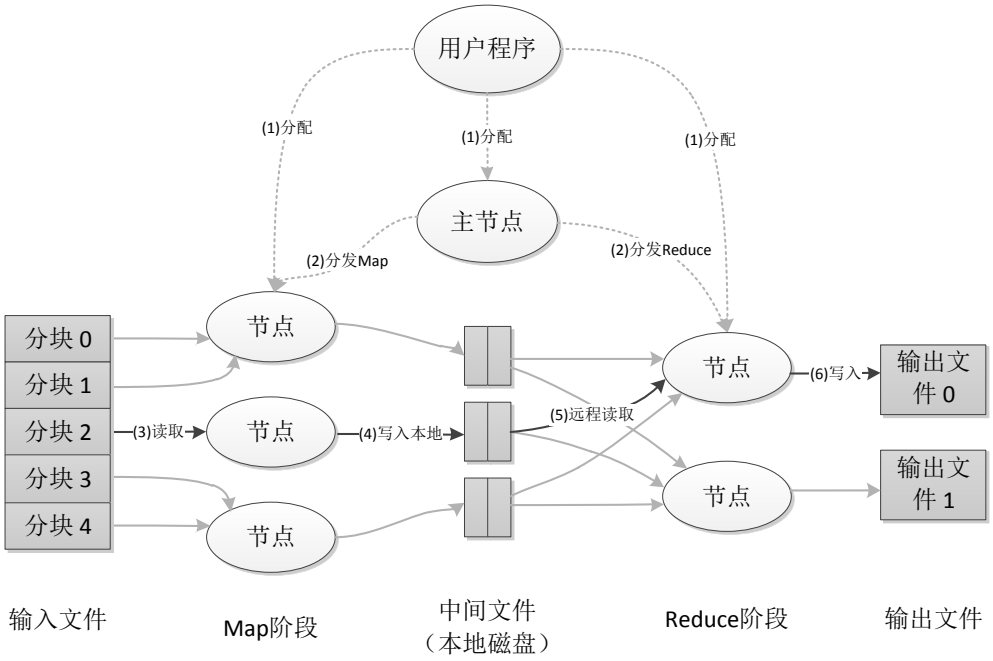

图 2.1 MapReduce 实现过程

当用户的程序调用 MapReduce 的函数的时候，将发生下面的一系列动作 (下面的数字和图2.1中的数字标签相对应)：

1. 在用户程序里的 MapReduce 库首先分割输入文件成 M 个片，每个片的大小一般从 16 到 64MB(用户可以通过可选的参数来控制)。然后在机群中开始大量的拷贝程序。

2. 这些程序拷贝中的一个是 master，其他的都是由 master 分配任务的 worker。有 M 个 map 任务和 R 个 reduce 任务将被分配。管理者分配一个 map 任务或 reduce 任务给一个空闲的 worker。

3. 一个被分配了 map 任务的 worker 读取相关输入 split 的内容。它从输入数据中分析出 key/value 对，然后把 key/value 对传递给用户自定义的 map 函数。由 map 函数产生的中间 key/value 对被缓存在内存中。

4. 缓存在内存中的 key/value 对被周期性的写入到本地磁盘上，通过分割函数把它们写入 R 个区域。在本地磁盘上的缓存对的位置被传送给 master，master 负责把这些位置传送给 reduce worker。

5. 当一个 reduce worker 得到 master 的位置通知的时候，它使用远程过程调用来从 map worker 的磁盘上读取缓存的数据。当 reduce worker 读取了所有的中间数据后，它通过排序使具有相同 key 的内容聚合在一起。因为许多不同的 key 映射到相同的 reduce 任务，所以排序是必须的。如果中间数据比内存还大，那么还需要一个外部排序。

6. reduce worker 迭代排过序的中间数据，对于遇到的每一个唯一的中间 key，它把 key 和相关的中间 value 集传递给用户自定义的 reduce 函数。reduce 函数的输出被添加到这个 reduce 分割的最终的输出文件中。

7. 当所有的 map 和 reduce 任务都完成了，管理者唤醒用户程序。在这个时候，在用户程序里的 MapReduce 调用返回到用户代码。

在成功完成之后，MapReduce 执行的输出存放在 R 个输出文件中 (每一个 reduce 任务产生一个由用户指定名字的文件)。一般，用户不需要合并这 R 个输出文件成一个文件 –他们经常把这些文件当作一个输入传递给其他的 MapReduce 调用，或者在可以处理多个分割文件的分布式应用中使用他们。

2.1.3 MapReduce 的优点

相比于常规分布式程序，MapReduce 架构中的分布式程序具有以下几个优点：

1. MapReduce 将并行计算分离成两个独立函数，Map 函数和 Reduce 函数。这种抽象的计算模型将分布式计算从复杂的可模式化的细节管理中剥离出来，使得开发分布式程序变得更加简洁。

2. MapReduce 思想来自于函数式编程，架构清晰易懂，入门门槛低，适合于大幅度推广。

3. MapReduce 框架轻量，高效，运行过程透明，便与底层的定制和二次开发。

2.2 Hadoop 简介

2.2.1 什么是 Hadoop

Hadoop 是 Apache 下的一个开源软件，它作为一个开源的软件平台使编写和运行用于处理海量数据的应用程序更加容易。Hadoop 框架的核心思想是 MapReduce。MapReduce 是一个用于进行大数据量计算的编程模型，同时也是一种高效的任务调度模型，它将一个任务分成很多更细粒度的子任务，这些子任务能够在空闲的处理节点之间调度，使处理速度越快的节点处理越多的任务，从而避免处理速度慢的节点延长整个任务的完成时间。

Hadoop 是一个能够对大量数据进行分布式处理的软件框架。但是 Hadoop 是以一种可靠、高效、可伸缩的方式进行处理的。Hadoop 是可靠的，因为它假设计算元素和存储会失败，因此它维护多个工作数据副本，确保能够针对失败的节点重新分布处理。Hadoop 是高效的，因为它以并行的方式工作，通过并行处理加快处理速度。Hadoop 还是可伸缩的，能够处理 PB 级数据。此外，Hadoop 依赖于社区服务器，因此它的成本比较低，任何人都可以使用。

2.2.2 Hadoop 的组成

Hadoop 整体功能由五个子进程实现，他们分别为 NameNode、SecondaryNameNode、Jobtracker、DataNode 和 TaskTracker，其具体功能如下：

NameNode，NameNode 是分布式文件系统的管理者，主要负责文件系统的命名空间，集群的配置信息和数据块的复制信息等，并将文件系统的元数据存储在内存中。

SecondaryNameNode，SecondaryNameNode 有两个作用，一是镜像备份，二是日志与镜像的定期合并。它会周期性的将 EditLog 中记录的对 HDFS 的操作合并到一个 checkpoint 中，然后清空 EditLog。如果没有 SecondaryNameNode 的这个周期性的合并过程，那么当每次重启 NameNode 的时候，就会花费很长的时间。而这样周期性的合并就能减少重启的时间。同时也能保证 HDFS 系统的完整性。

JobTracker，负责任务的接受初始化，调度以及对 TaskTrackee 的监控。JobTracker 作为一个单独的 JVM 运行，对整个 MapReduce 程序的运行起着至关重要的作用。

DataNode，负责为响应来自 HDFS 客户机的读写请求。它们还响应创建、删除和复制来自 NameNode 的块的命令。并与 NameNode 通过心跳包来维持联系。DataNode，负责为响应来自是文件实际存储的位置，它将块 (Block) 的元数据信息存储在本地文件系统中，周期性地将所有的 Block 信息发给 NameNode。

TaskTrackee，运行作业划分后的任务。

其中 NameNode、SecondaryNameNode 和 Jobtracker 运行在 Master 节点上，DataNode 和 TaskTracker 运行在 Slave 节点上。

本文中，因为实验环境限制，Hadoop 运行模式为伪分布式，即五个子进程均处于同一宿主机上。如图2.2

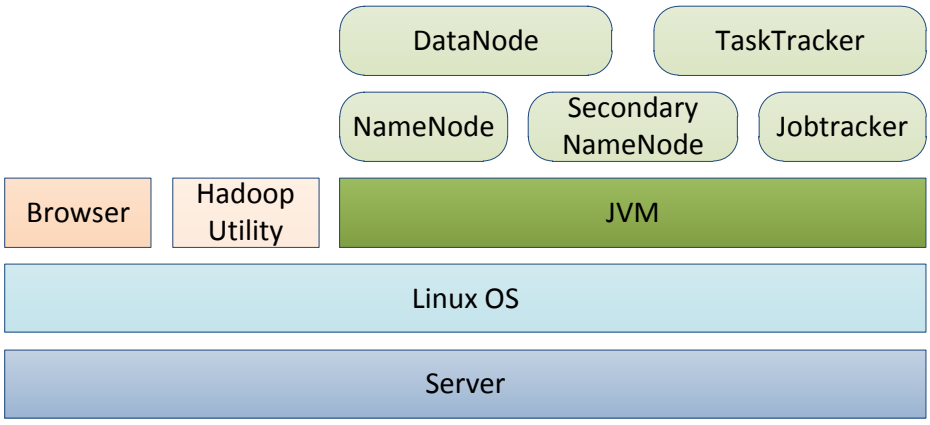

图 2.2 五个子进程

2.2.3 HDFS 的实现

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 是分布式计算的存储基石，Hadoop 的分布式文件系统和其他分布式文件系统有很多类似的特质。分布式文件系统基本的几个特点：

1. 对于整个集群有单一的命名空间。
2. 数据一致性。适合一次写入多次读取的模型，客户端在文件没有被成功创建之前无法看到文件存在。
3. 文件会被分割成多个文件块，每个文件块被分配存储到数据节点上，而且根据配置会由复制文件块来保证数据的安全性。

图**??**：HDFS 结构示意图上图中展现了整个 HDFS 三个重要角色：NameNode、DataNode 和 Client。NameNode 可以看作是分布式文件系统中的管理者，主要负责管理文件系统的命名空间、集群配置信息和存储块的复制等。NameNode 会将文件系统的 Meta-data 存储在内存中，这些信息主要包括了文件信息、每一个文件对应的文件块的信息和每一个文件块在 DataNode 的信息等。DataNode 是文件存储的基本单元，它将 Block 存储在本地文件系统中，保存了 Block 的 Meta-data，同时周期性地将所有存在的 Block 信息发送给 NameNode。Client 就是需要获取分布式文件系统文件的应用程序。这里通过三个操作来说明他们之间的交互关系。

文件写入：

1. Client 向 NameNode 发起文件写入的请求。
2. NameNode 根据文件大小和文件块配置情况，返回给 Client 它所管理部分 DataNode 的信息。
3. Client 将文件划分为多个 Block，根据 DataNode 的地址信息，按顺序写入到每一个 DataNode 块中。

文件读取：

1. Client 向 NameNode 发起文件读取的请求。
2. NameNode 返回文件存储的 DataNode 的信息。
3. Client 读取文件信息。

文件 Block 复制：

1. NameNode 发现部分文件的 Block 不符合最小复制数或者部分 DataNode 失效。

2. 通知 DataNode 相互复制 Block。

3. DataNode 开始直接相互复制。

因此，HDFS 具有以下的几个设计特点：

1. Block 的放置：默认不配置。一个 Block 会有三份备份，一份放在 NameNode 指定的 DataNode，另一份放在与指定 DataNode 非同一 Rack 上的 DataNode，最后一份放在与指定 DataNode 同一 Rack 上的 DataNode 上。备份无非就是为了数据安全，考虑同一 Rack 的失败情况以及不同 Rack 之间数据拷贝性能问题就采用这种配置方式。

2. 心跳检测 DataNode 的健康状况，如果发现问题就采取数据备份的方式来保证数据的安全性。

3. 数据复制（场景为 DataNode 失败、需要平衡 DataNode 的存储利用率和需要平衡 DataNode 数据交互压力等情况）：这里先说一下，使用 HDFS 的 balancer 命令，可以配置一个 Threshold 来平衡每一个 DataNode 磁盘利用率。例如设置了 Threshold 为 10

4. 数据交验：采用 CRC32 作数据交验。在文件 Block 写入的时候除了写入数据还会写入交验信息，在读取的时候需要交验后再读入。

5. NameNode 是单点：如果失败的话，任务处理信息将会纪录在本地文件系统和远端的文件系统中。

6. 数据管道性的写入：当客户端要写入文件到 DataNode 上，首先客户端读取一个 Block 然后写到第一个 DataNode 上，然后由第一个 DataNode 传递到备份的 DataNode 上，一直到所有需要写入这个 Block 的 NataNode 都成功写入，客户端才会继续开始写下一个 Block。

7. 安全模式：在分布式文件系统启动的时候，开始的时候会有安全模式，当分布式文件系统处于安全模式的情况下，文件系统中的内容不允许修改也不允许删除，直到安全模式结束。安全模式主要是为了系统启动的时候检查各个 DataNode 上数据块的有效性，同时根据策略必要的复制或者删除部分数据块。运行期通过命令也可以进入安全模式。在实践过程中，系统启动的时候去修改和删除文件也会有安全模式不允许修改的出错提示，只需要等待一会儿即可。

HDFS 是 MapReduce 的读写的底层实现，对 MapReduce 进化起着至关重要的作用。

2.2.4 Hadoop 的运行过程

在 Hadoop 的系统中，会有一台 Master，主要负责 NameNode 的工作以及 JobTracker 的工作。JobTracker 的主要职责就是启动、跟踪和调度各个 Slave 的任务执行。还会有多台 Slave，每一台 Slave 通常具有 DataNode 的功能并负责 TaskTracker 的工作。TaskTracker 根据应用要求来结合本地数据执行 Map 任务以及 Reduce 任务。

具体过程如下：

1. 在分布式环境中客户端创建任务并提交。

2. InputFormat 做 Map 前的预处理，主要负责以下工作：

   1) 验证输入的格式是否符合 JobConfig 的输入定义，这个在实现 Map 和构建 Conf 的时候就会知道，不定义可以是 Writable 的任意子类。

   2) 将 input 的文件切分为逻辑上的输入 InputSplit，其实这就是在上面提到的在分布式文件系统中 blocksize 是有大小限制的，因此大文件会被划分为多个 block。

   3) 通过 RecordReader 来再次处理 inputsplit 为一组 records，输出给 Map。（inputsplit 只是逻辑切分的第一步，但是如何根据文件中的信息来切分还需要 RecordReader 来实现，例如最简单的默认方式就是回车换行的切分）

3. RecordReader 处理后的结果作为 Map 的输入，Map 执行定义的 Map 逻辑，输出处理后的 key 和 value 对应到临时中间文件。

4. Combiner 可选择配置，主要作用是在每一个 Map 执行完分析以后，在本地优先作 Reduce 的工作，减少在 Reduce 过程中的数据传输量。

5. Partitioner 可选择配置，主要作用是在多个 Reduce 的情况下，指定 Map 的结果由某一个 Reduce 处理，每一个 Reduce 都会有单独的输出文件。（后面的代码实例中有介绍使用场景）

6. Reduce 执行具体的业务逻辑，并且将处理结果输出给 OutputFormat。

7. OutputFormat 的职责是，验证输出目录是否已经存在，同时验证输出结果类型是否如 Config 中配置，最后输出 Reduce 汇总后的结果。

2.2.5 Hadoop 的特点

Hadoop 是面向大型集群实现 MapReduce 方法而设计的编程框架，其实现具有以下几个特点。

1. 可扩展：不论是存储的可扩展还是计算的可扩展都是 Hadoop 的设计根本。Hadoop 中许多组件都是可插拔的，如调度器

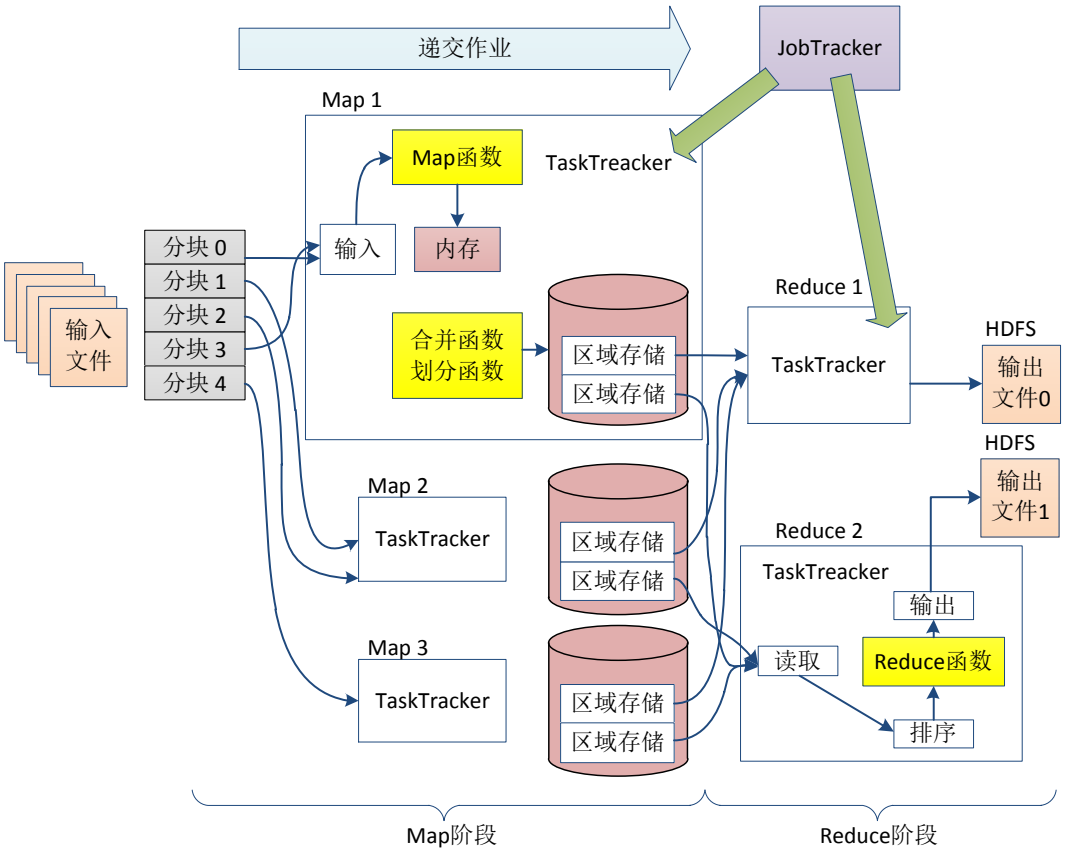

图 2.3 Hadoop 运行流程

2. 经济：框架可以运行在任何普通的 PC 上，使用者只需使用普通的 PC 机即可搭建性能强大的集群。

3. 可靠：分布式文件系统的备份恢复机制以及 MapReduce 的任务监控保证了分布式处理的可靠性。

4. 高效：分布式文件系统的高效数据交互实现以及 MapReduce 结合 Local Data 处理的模式，为高效处理海量的信息作了基础准备。

5. 使用方便：Hadoop 使用 Java 语言实现，利用 ssh 协议与各节点之间通信，底层依赖小，可移植性强，适应于各种环境。

2.2.6 MapReduce 在 Hadoop 中的三种实现方式及特点

1. 原生 Java

2. Streaming

   Hadoop 提供了 MapReduce 的 API，并允许你使用非 Java 的其他语言来写自己的 map 和 reduce 函数。Hadoop 的 Streaming 使用 Unix 标准流作为

Hadoop 和应用程序之间的接口，所以我们可以使用任何编程语言通过标准输入/输出来写 MapReduce 程序。

Streaming 天生适合用于文本处理 (到 0.21.0 版本时，Streaming 也可以处理二进制流)，在文本模式下使用时，它有一个数据的行视图。map 的输入数据通过标准输入流传递给 map 函数，并且是一行一行地传输，最后将结果行写到标准输出。map 输出的键/值对是以一个制表符分隔的行，它以这样的形式写到标准输出。reduce 函数的输入格式相同——通过制表符来分隔的键/值对——并通过标准输入流进行传输。reduce 函数从标准输入流中读取输入行，该输入已由 Hadoop 框架根据键排过序，最后将结果写入标准输出。

值得一提的是 Streaming 和 Java MapReduce API 之间的设计差异。Java API 控制的 map 函数一次只能处理一条记录。针对输入数据中的每一条记录，该框架均需调用 Mapper 的 map() 方法来处理，然而在 Streaming 中，map 程序可以自己决定如何处理输入数据，例如，它可以轻松读取并同时处理若干行，因为它受读操作的控制。用户的 Java map 实现的是“推”记录方式，但它依旧可以同时处理多行，具体做法是通过 mapper 中实例变量将之前读取的多行汇聚在一起。在这种情况下，需要实现 close() 方法，以便知道何时读到最后一条记录，进而完成对最后一组记录行的处理。

3. Pipes

Hadoop 的 Pipes 是 Hadoop MapReduce 的 C++ 接口代称。不同于使用标准输入和输出来实现 map 代码和 reduce 代码之间的 Streaming，Pipes 使用套接字作为 tasktracker 与 C++ 版本 map 函数或 reduce 函数的进程之间的通道。

应用程序对 Hadoop C++ 库链接提供了一个与 tasktracker 子进程进行通信的简单封装。通过扩展 HadoopPipes 命名空间中定义的 mapper 和 reducer 两个类，我们定义了 map() 和 reduce() 方法，同时我们提供各种情况下 map() 和 reduce() 方法的实现。这些方法采用了上下文对象 (MapContext 类型或 ReduceContext 类型)，进而提供了读取输入数据和写入输出数据，以及通过 JobConf 类来访问作业配置信息的功能。

与 Java 接口不同，C++ 接口中的键和值按字节缓冲，用标准模板库 (Standard Template Library，STL) 中的字符串表示。这样做简化了接口，但把更重的负担留给了应用程序开发人员，因为开发人员必须来回封送 (marshall) 字符串与特定应用领域内使用的具体类型。

因 Pipes 通过 trasktracker 子进程进行通信，但由于 Hadoop 提供的 C++ 接口功能过于简单，开发负担大，故本文中仅对原生 Java 和 Streaming 的实现做性能对比。

2.2.7 Hadooop 计数器及其含义

Hadoop 里有一个很常用的工具叫计数器（Counter）, 主要用来记录 Hadoop job 的运行状态。通过计数器，用户可以观察 MapReduce job 运行期间的各个细节数据，大多数优化都是基于计数器的数值表现。本文中后期的性能分析主要依靠 Hadoop 计数器提供的数据完成。

Hadoop job 提供的默认计数器分为五个组分别是 Job Counters、File Input Format Counters、File Output Format Counters、FileSystem Counters 和 MapReduce Framework 其包含的数据和含义如下：

1. Job Counters

   这个计数器描述与 job 调度相关的统计

   Data-local map tasks：Job 在被调度时，如果启动了一个 data-local(源文件的幅本在执行 map task 的 taskTracker 本地)

   FALLOW_SLOTS_MILLIS_MAPS：当前 job 为某些 map task 的执行保留了 slot，总共保留的时间是多少

   FALLOW_SLOTS_MILLIS_REDUCES：与上面类似

   SLOTS_MILLIS_MAPS：所有 map task 占用 slot 的总时间，包含执行时间和创建/销毁子 JVM 的时间

   SLOTS_MILLIS_REDUCES: 与上面类似

   Launched map tasks：此 job 启动了多少个 map task

   Launched reduce tasks：此 job 启动了多少个 reduce task

2. File Inut Format Counters

   这个计数器表示 map task 读取文件内容 (总输入数据) 的统计

   BYTES_READ：Map task 的所有输入数据 (字节)，等于各个 map task 的 map 方法传入的所有 value 值字节之和。

3. File Output Format Counters

   这个计数器表示 map task 写入文件内容 (总输入数据) 的统计

   BYTES_READ：Map task 的所有输出数据 (字节)，等于各个 map task 的 map 方法传出的所有 value 值字节之和。

4. FileSystem Counters

   MapReduce job 执行所依赖的数据来自于不同的文件系统，这个计数器表示 job 与文件系统交互的读写统计

   FILE_BYTES_READ：job 读取本地文件系统的文件字节数。假定我们当前 map 的输入数据都来自于 HDFS，那么在 map 阶段，这个数据应该是 0。

但 reduce 在执行前，它的输入数据是经过 shuffle 的 merge 后存储在 reduce 端本地磁盘中，所以这个数据就是所有 reduce 的总输入字节数。

FILE_BYTES_WRITTEN：map 的中间结果都会 spill 到本地磁盘中，在 map 执行完后，形成最终的 spill 文件。所以 map 端这里的数据就表示 map task 往本地磁盘中总共写了多少字节。与 map 端相对应的是，reduce 端在 shuffle 时，会不断地拉取 map 端的中间结果，然后做 merge 并不断 spill 到自己的本地磁盘中。最终形成一个单独文件，这个文件就是 reduce 的输入文件。

HDFS_BYTES_READ：整个 job 执行过程中，只有 map 端运行时，才从 HDFS 读取数据，这些数据不限于源文件内容，还包括所有 map 的 split 元数据。所以这个值应该比 FileInputFormatCounters.BYTES_READ 要略大些。

HDFS_BYTES_WRITTEN：Reduce 的最终结果都会写入 HDFS，就是一个 job 执行结果的总量。

5. Map-Reduce Framework

这个计数器包含了相当多地 job 执行细节数据。这里需要有个概念认识是：一般情况下，record 就表示一行数据，而相对地 byte 表示这行数据的大小是多少，这里的计数器表示经过 reduce merge 后像这样的输入形式“aaa”, [5, 8, 2, ⋯]。

Combine input records：Combiner 是为了减少尽量减少需要拉取和移动的数据，所以 combine 输入条数与 map 的输出条数是一致的。

Combine output records：经过 Combiner 后，相同 key 的数据经过压缩，在 map 端自己解决了很多重复数据，表示最终在 map 端中间文件中的所有条目数

Failed Shuffles：copy 线程在抓取 map 端中间数据时，如果因为网络连接异常或是 IO 异常，所引起的 shuffle 错误次数

GC time elapsed(ms)：通过 JMX 获取到执行 map 与 reduce 的子 JVM 总共的 GC 时间消耗

Map input records：所有 map task 从 HDFS 读取的文件总行数

Map output records：map task 的直接输出 record 是多少，就是在 map 方法中调用 context.write 的次数，也就是未经过 Combine 时的原生输出条数

Map output bytes：Map 的输出结果 key/value 都会被序列化到内存缓冲区中，所以这里的 bytes 指序列化后的最终字节之和

Merged Map outputs：记录着 shuffle 过程中总共经历了多少次 merge 动作

Reduce input groups：Reduce 总共读取了多少个这样的计数器

Reduce input records：如果有 Combiner 的话，那么这里的数值就等于 map 端 Combiner 运算后的最后条数，如果没有，那么就应该等于 map 的输出条数

Reduce output records：所有 reduce 执行后输出的总条目数

Reduce shuffle bytes：Reduce 端的 copy 线程总共从 map 端抓取了多少的中间数据，表示各个 map task 最终的中间文件总和

Shuffled Maps：每个 reduce 几乎都得从所有 map 端拉取数据，每个 copy 线程拉取成功一个 map 的数据，那么增 1，所以它的总数基本等于 reduce number * map number

Spilled Records：spill 过程在 map 和 reduce 端都会发生，这里统计在总共从内存往磁盘中 spill 了多少条数据

SPLIT_RAW_BYTES 与 map task 的 split 相关的数据都会保存于 HDFS 中，而在保存时元数据也相应地存储着数据是以怎样的压缩方式放入的，它的具体类型是什么，这些额外的数据是 MapReduce 框架加入的，与 job 无关，这里记录的大小就是表示额外信息的字节大小

2.3 Web 访问日志简介

2.3.1 什么是 Web 访问日志

Web 访问日志是 Web 服务器的运行记录文档，当用户访问一个网站时，服务器上的 Web 服务会接受该用户的请求，并将用户的请求信息记录并保存下来。目前常见的 Web 访问日志格式主要由两类，一类是 Apache 的 NCSA 日志格式，另一类是 IIS 的 W3C 日志格式。NCSA 格式又分为 NCSA 普通日志格式（CLF）和 NCSA 扩展日志格式（ECLF）两类，目前最常用的是 NCSA 扩展日志格式（ECLF）及基于自定义类型的 Apache 日志格式；而 W3C 扩展日志格式（ExLF）具备了更为丰富的输出信息，但目前的应用并不广泛。目前，大多数网站使用 NCSA 日志格式来存储日志。

Nginx 是一款基于 Epoll 模型的 web 服务程序，近年来发展迅猛，截止 2012 年 5 月，Nginx 已经占据 Web 软件 10.32% 的份额[?]，Nginx 的默认日志为 Apache 的 NCSA 扩展日志格式（ECLF），本实验中所用的日志为 Nginx 默认生成的日志。

2.3.2 NCSA 扩展日志格式的组成

这是一个最常见的基于 NCSA 扩展日志格式（ECLF）的 Nginx 日志样例：

```
222.25.151.30 - - [28/May/2012:19:12:03 +0800] "GET /bbs/uc_server/ava
tar.php?uid=46369&size=small HTTP/1.1" 301 0 "http://rs.xidian.edu.cn/
bbs/index.php" "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Tri
```

```
dent/4.0; GTB7.2; .NET CLR 1.1.4322; .NET CLR 2.0.50727; CIBA; TheWorl
d)" -
```

可以很清楚的看到，这个日志主要由以下几个部分组成：

1. 访问主机（remotehost）

   显示主机的 IP 地址或者已解析的域名。

2. 标识符（Ident）

   由 identd 或直接由浏览器返回浏览者的 EMAIL 或其他唯一标示，因为涉及用户邮箱等隐私信息，目前几乎所有的浏览器就取消了这项功能。

3. 授权用户（authuser）

   用于记录浏览者进行身份验证时提供的名字，如果需要身份验证或者访问密码保护的信息则这项不为空，但目前大多数网站的日志这项也都是为空的。

4. 日期时间（date）

   一般的格式形如 [22/Feb/2010:09:51:46 +0800]，即 [日期/月份/年份: 小时: 分钟: 秒钟时区]，占用的的字符位数也基本固定。

5. 请求（request）

   即在网站上通过何种方式获取了哪些信息，也是日志中较为重要的一项，主要包括以下三个部分：

6. 请求类型（METHOD）

   常见的请求类型主要包括 GET/POST/HEAD 这三种；

7. 请求资源（RESOURCE）

   显示的是相应资源的 URL，可以是某个网页的地址，也可以是网页上调用的图片、动画、CSS 等资源，是 Web 访问日志挖掘中的一个重点；

8. 协议版本号（PROTOCOL）

   显示协议及版本信息，通常是 HTTP/1.1 或 HTTP/1.0。

9. 状态码（status）

   用于表示服务器的响应状态，通常 1xx 的状态码表示继续消息；2xx 表示请求成功；3xx 表示请求的重定向；4xx 表示客户端错误；5xx 表示服务器错误。

10. 传输字节数（bytes）

    即该次请求中一共传输的字节数。

11. 来源页面（referrer）

    用于表示浏览者在访问该页面之前所浏览的页面，只有从上一页面链接过来的请求才会有该项输出，如果是新开的页面则该项为空。上例中来源页面是google，即用户从 google 搜索的结果中点击进入。

12. 用户代理（agent）

    用于显示用户的详细信息，包括 IP、OS、Bowser 等。

2.3.3 Web 访问日志的作用及价值

Web 访问日志记录了用户访问一个网站的所有过程，是个非常好的数据挖掘原料。通过对 Web 访问日志的分析，网站服务提供商可以获得某一指定用户的点击流（Clickstream），网站的流量及访问时段，网站的内部漏洞及可能的注入点。

有效的利用 Web 访问日志可以高效精准的分析出用户喜好，帮助网站运营商提升网站质量。

2.4 本章小结

本章主要介绍本文的研究背景，详述了 MapReduce、Hadoop 的起源、发展及其运行机制，并详述了 Hadoop Counter 和 Web 访问日志的内容及其格式。为进一步的开发和研究做了铺垫。

第三章　基于 Hadoop 的 MapReduce 程序设计

3.1　Web 访问日志分析程序设计

3.1.1　程序应用场景

随着互联网的高速发展，各种 Web2.0 网站、电子商务网站创造了前所未有的访问记录，各种大型网络游戏不断刷新着在线用户数峰值，与此同时这些大型系统都记录下了海量的运行日志。挖掘出日志中蕴藏的信息来改进用户体验提升服务质量是非常有价值的。本程序利用 MapReduce 框架，利用分布式集群处理访问日志计算 PV，可以解决传统生产环境中单机处理 Web 访问日志速度过慢的问题。

3.1.2　程序实现过程

本程序基于 Hadoop 1.0.1，利用 MapReduce 思想进行程序开发。MapReduce 思想十分简单，首先将所有的数据分发到各个集群上，并在集群执行 Map 函数，之后将 Map 结果合并并交给 reduce 做进一步的计算，详见本文第二章。

根据 MapReduce 思想，根据实际需求，本程序运行流程图如图3.1所示：

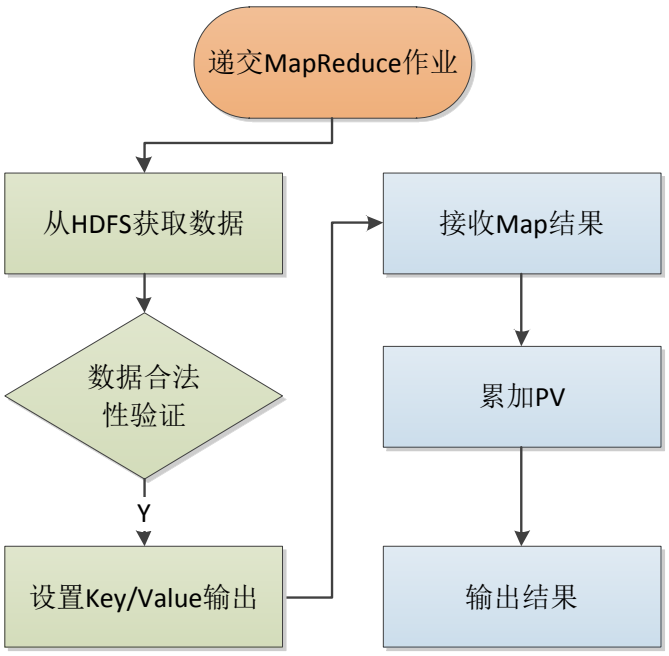

图 3.1 程序流程图

Map 函数的功能为：从 HDFS 上获取日志文件，文件每一行作为一个输入，Mapper 在读取行之后，验证其访问的有效性，之后设置“pv”作为 Mapper 的 key，1 作为值，将 Map 结果输出。Reduce 函数的功能为：从 Map 端接收合并好的 Key/Value 对，将 key 所对应的 Value 的值相加，得到 PV 值，将新的 Key/Value 对输出。

3.1.3 程序运行环境

Hadoop 依赖 SSH 和 JDK，具有很高的通用性，但一般情况下，为了整体集群的效率，操作系统一般选用 Linux，使用 Sun JDK，本程序的开发运行环境具体如下：

实验机器配置：

CPU：Pentium(R) Dual-Core CPU E5800 @ 3.20GHz

内存：2G DDR2 800

硬盘：Seagate 320G 7200RPM

操作系统：CentOS_6.2_x64

CentOS 是 RedHat 的开源版本，具有了 RedHat 的一切特性并有着稳定的社区支持，以其高效和稳定著称，是理想的 Hadoop 宿主操作系统。6.2 是其最新稳定版。本文中所使用的 CentOS 为 netinstall 版，这种方式安装出来的 CentOS 体积最小，软件包最新，服务数量最少，运行速度最快，且不损失任何系统稳定性。

JDK：Sun JDK 1.6.0_31

Java Development Kit (JDK) 是 Sun 公司针对 Java 开发人员发布的免费软件开发工具包 (SDK，Software development kit)。自从 Java 推出以来，Sun JDK 已经成为使用最广泛的 Java SDK。1.6.0_31 是其最新稳定版。

Hadoop 1.0.1

Hadoop 是 Apache 社区下的开源软件，1.0.1 是其最新稳定版。

编程语言：Java、Python 2.6

编程语言的差异会对程序的运行效率产生至关重要的影响，本实验选用 Java、Python 两种最常见的解释性编程语言，便于后期性能对比。

3.2 程序开发与实现

3.2.1 原生 Java 的实现

Hadoop 使用 Java 语言实现，其自身提供了一套完整的 Java 类库，使用起来十分方便。

首先，创建一个 Java 文件，在文件头部定义 package 名称，引入相关类库之后就可以开始编写 Mapreduce 程序了，程序中需要在 Java 包内定一个一个 Public 类 pv，运行时，将类名作为参数传入 Hadoop。pv 类里定义 Map 和

Reduce 类和一个 main 函数：TokenizerMapper 继承 Mapper 类，IntSumReducer 继承 Reducer 类，main 函数里指定运行的 Map 和 Reduce 类名、和数据类型，其各部分具体功能如下：

TokenizerMapper 定义两个 private 变量，用于存储 Key/Value 的值。之后定义一个 map 函数，判断读入的数据的合法性，若合法，则给 value 赋值 1，反之赋值 0。

IntSumReducer 定义一个变量，用于存储最终 pv 的值。之后定一个一个函数 reducer，将 Map 传入的结果累加。最后输出 Key 和 pv 值。

main 函数中实例化 Hadoop 的配置类、和 Job 类，并给 job 类的各项属性赋值。

编写完毕之后，应对程序进行调试，Hadoop 为 Eclipse 提供了完整的调试插件，但由于其配置复杂且超出了本文讨论范围，在此不详细给出。

其在 Hadoop 中的运行流程如下：

至此，Web 日志处理的原生 Java 程序就实现了。

3.2.2 基于 Streaming 的 Python 实现

Hadoop 在支持原生 Java 的同时也对外提供了 Streaming 方法，该方法把 HDFS 中的文件转化成文件流，传入外部程序处理，使得 Hadoop 具有处理非 Java 编写的程序的能力。本文选用了和 Java 及其类似的解释性编程语言 Python，便于后期的性能测试对比。

在 Hadoop 的 Streaming 中，所编程序在开发过程中无需引用任何 Hadoop 类库，Hadoop 会将文件转换成文件流输入，程序处理之后，再将 Key/Value 对以 std 方式输出，Hadoop 会自动捕获输出并进行下一步处理。实现是，Mapper 和 Reducer 需要单独书写。

新建文件 Mapper.py，文件首部向系统声明文件执行类型为 python，又因程序使用 std I/O 并分割字符串，所以在头部引用 sys 和 itemgetter。之后使用 for 循环语句从 sys.stdin 读取数据，并判断其合法性，若合法，则给 value 赋值 1，反之赋值 0，最后使用 print 输出 Key/Value 对，并以一个缩进符号分割开来。

新建文件 Reducer.py，文件首部和引用于 Mapper.py 相同，定义存储 pv 值的变量 sum，使用 for 循环从 sys.stdin 读取 Mapper 中生成的文件流，使用 itemgetter 分割 Key/Value 对到对应的变量，将 pv 值累加。最后使用 print 输出 Key 和 pv 值。

由于 Streaming 使用 std I/O，使得其调试非常简单，用户可以直接在本机使用 cat 命令通过 Unix 管道即可进行简单的的调试。运行命令 cat 10.log | ./mapper.py 即可查看 Map 的结果。运行 cat 10.log | ./mapper.py | ./reducer.py 即可查看 MapReducer 结果。

运行时向 hadoop 指定 map 文件和 reduce 文件，并使用 file 命令将 map/reduce 程序分发到各节点即可。

其在 Hadoop 中的运行流程如下：

至此，Web 日志处理的 Python 程序就实现了。

3.3 实现效果及性能测试

3.3.1 测试数据及方法

为了验证本程序的正确性和运行效率，笔者使用了 9 组数据，测试数据为基数为 10 到 100,000,000，增量为十倍，共 8 组清洗过的有效合法日志，日志取自实际生产环境服务器，具有一定的参考价值。所有测试数据均存放在采用 Hadoop 默认配置的 HDFS 上。

测试环境为伪分布式的单机 Hadoop 节点，Java 和 Python 程序的运行均使用 Hadoop 默认参数执行，测试过程中，使用 sar 对系统资源进行监测，测试结束后，收集运行日志进行更进一步的分析。

3.3.2 基于原生 Java 的实现及性能测试

本实验中 Java 程序均使用 Sun_JDK_1.6.0_31 编译并打包，Java 程序上传到服务器上之后，执行命令

```
hadoop jar pv.jar test.pv input/100.log output/java/100
```

其中 pv.jar 为打包后的程序，test.pv 为程序中所对应的包及类，input/100.log 为 HDFS 上的日志文件，output/java/100 为输出目录。

命令执行后，终端会实时跟踪程序运行情况，用户也可以通过网页终端查看当前程序的运行状况。程序执行完毕之后，Hadoop 会将程序运行信息输出至终端，程序运行结果保存在指定的 HDFS 目录下用户可以运行一下命令查看运行结果。

```
hadoop dfs -cat output/java/100/part-r-00000
```

在本实验中，所用测试数据是提前使用工具处理好的数据，目的在于验证程序的准确性并分析对比其运行时间。具体运行结果和运行时间如下表：

3.3.3 基于 Streaming 的 Python 实现及其性能测试

本实验中的 Python 程序均使用 CentOS 自带的 Python 2.6.6 解析，Python 程序上传到服务器上之后，执行命令

```
hadoop jar $HADOOP_HOME/contrib/streaming/hadoop-streaming-1.0.1.jar \
-file mapper.py -mapper mapper.py \
-file reducer.py -reducer reducer.py \
-input input/100.log -output output/python/100
```

其中 $HADOOP_HOME/contrib/streaming/hadoopstreaming1.0.1.jar 为 Hadoop 内置的 Streaming 类，file 命令将程序分发到各节点，mapper 和reducer 命令分别制定 mapper.py 和 reducer.py 为 Mapper 和 Reducer，input/100.log 为 HDFS 上的日志文件，output/python/100 为输出目录。

命令执行后，同原生 Java 一样，终端会实时跟踪程序运行情况，用户也可以通过网页终端查看当前程序的运行状况。程序执行完毕之后，Hadoop 会将程序运行信息输出至终端，程序运行结果保存在指定的 HDFS 目录下用户可以运行一下命令查看运行结果。

```
hadoop dfs -cat output/python/100/part-r-00000
```

同上一小节，在本实验中，所用测试数据是提前使用工具处理好的数据，目的在于验证程序的准确性并分析对比其运行时间。具体运行结果和运行时间如下表：

3.4　本章小结

本章以 Web 访问日志程序开发为例，分别介绍了在 Hadoop 下的两种程序开发模式：原生 Java 和 Streaming，并对其性能做了测试，下一章会根据本章第三节的测试结果做以分析和说明。

第四章　基于 Hadoop 的 MapReduce 实现的性能分析

4.1　Hadoop 调度对性能的影响

Hadoop 程序运行时，会有进程监控节点资源，并将系统中的空闲资源按一定的策略分配给作业。

4.1.1　Hadoop 默认调度机制 FIFO

Hadoop 默认的调度器是 JobQueueTaskScheduler，使用规则为 FIFO (First In First Out，即先入先出) 原则，即按照作业的优先级高低，再按照到达时间的先后选择被执行的作业。

其具体工作流程如下图：

对于 JobQueueTaskScheduler 的任务调度原则可总结如下：

1. 1. 任务先进先出；
2. 2. 尽量使集群每一个 TaskTracker 达到负载均衡 (这个均衡是 task 数量上的而不是实际的工作强度)；
3. 3. 尽量分配作业的本地任务给 TaskTracker，但不是尽快分配作业的本地任务给 TaskTracker，最多分配一个非本地任务给 TaskTracker(一是保证任务的并发性，二是避免有些 TaskTracker 的本地任务被偷走)，最多分配一个 reduce 任务；
4. 4. 为紧急的 Task 预留一定的 slot；

这种调度方式简单明了，JobTracker 负载较小，适用于大多数的应用场景。

它的主要缺点就是对于在集群比较繁忙的情况，低优先级的作业将很难分配到集群的计算资源，这样对于那些低优先级同时又要求一定的响应时间的短作业来说是非常不利的。

为了解决这种应用差异性带来的性能损失，Hadoop 允许用户自定义调度器。Hadoop 的调度器被设计为一个可插拔的模块，用户可以根据自己实际应用要求设计调度器。

但由于调度机制在单机为分布式下的执行过于简单，其调度结果不具有参考价值，故在本文中不予以进一步的讨论。

4.2 Hadoop 运行机制对 MapReduce 性能的影响

4.2.1 原生 Java

通过第三章的测试结果可以看出，原生 Java 程序运行时，Map 在经过一次大量的读操作之后，其写操作和 Reduce 的读写操作均经过合并优化。Map 的结果被很好的合并，并交给了 Reducer 进行处理。程序运行效率非常高。

由于 Hadoop 提供了类库，所有操作均可通过接口实现，且对 Hadoop 内部透明，方便数据合并及优化，降低了 I/O 压力，性能损失很小。

4.2.2 基于 Streaming 的实现

通过第三章的的测试结果可以看出，基于 Streaming 的实现，Python 的运行时间随着文件的增大逐渐比 Java 长，性能变差的主要是因为 Map 的结果没有进行 combine。从表可以看出，Streaming 中的 Map 和 Reduce 结果没有经过任何 Combine，直接存入本地磁盘，从而造成了大量的空间浪费，增大了 I/O 负担，进入 Reduce 进一步处理的数据没有经过合并，造成性能损失。

以处理 1 亿条日志为例，从表可以得出，Streaming 的实现与 Java 相比，写入量增大约 100 倍，读入量增大约 10 万倍，内存使用量增大约 3 倍，Reduce 函数输入量增大了约 15 万倍，造成了性能损失。

由此可见，MapReduce 框架中的输入输出优化、缓存和合并机制是 MapReduce 程序运行效率的关键。

4.3 节点计算能力对 MapReduce 性能的影响

Hadoop 中，作业在 TaskTracker 上执行，TaskTracker 的计算能力直接影响着 Hadoop 的整体性能。MapReduce 是分布式计算框架，节点之间的通信通过网络传输。在 Map 和 Reduce 过程中，Hadoop 产生了大量的网络和磁盘 I/O，尤其是在 Mapper 和 Reducer 结束时，会引起数据量极大的“网络风暴”，这在实际生产环境中会给网卡和交换机带来很大的压力，若配置不当，将会成为 MapReduce 的瓶颈。

同时，在 Hadoop 运行过程中，Reduce 在 shuffle 阶段对下载来的 Map 数据，并不是立刻就写入磁盘的，而是会先缓存在内存中，然后当使用内存达到一定量的时候才刷入磁盘，这就要求节点机有足够大的内存去缓存数据，默认情况下，reduce 会使用其 heapsize 的 70% 来在内存中缓存数据。内存的大小决定了缓存数量的多少，减少不必要的 I/O。

磁盘读写速度也是 MapReduce 中影响性能的一个重要指标，在 Map 阶段，Hadoop 会将 Map 的中间结果保存在节点本地磁盘，因为中间结果是临时的，不需要写多份。本地磁盘的读写 I/O 速度在某些应用中也会成为 MapReduce 程序的瓶颈。

另外，在实际生产环境中，节点计算之间的计算能力会有略微的不同，默认的 Hadoop 调度器会使作业运行时出现性能短板，这种短板在异构环境中尤为明显。

4.4　实例程序编写方法对 MapReduce 性能的影响

通过 Java 和 Streaming 两种实现方式的比较，可以看出影响 MapReduce 较大的部分为 Suffle 部分，从 MapReduce 运行原理可以看出，Suffle 部分的合并是根据 Key 值合并的，如何合理的选择 Key 值，降低 Reducer 的处理时间，对 MapReduce 程序的运行起着至关重要的作用。

但实际开发中，Key 的分布并不一定是均匀的，极端情况下，所有数据共享一个 Key，后果是 reduce 任务只会在集群其中一个节点上运行，不仅没有利用好集群处理能力，反倒因为大量数据集中而导致计算效率低下，降低程序整体运行效率。

作为了解决这个问题，程序员需要在编程时尽量选择分布均匀的参数作为 Key，或者使用其他方法将 Key 打散。下文以 Web 日志处理中的处理 UV 为例，描述了一种打散 Key 的方法。

UV 是独立访客 (unique visitor) 的缩写，对于网站的用户粘度和回头率统计有着至关重要的作用。

在网站设计中，通常会给 Cookie 一个 Key 为 uid，Value 为用户 id 的一组数据。在 Web 日志中，该 uid 即可用来统计 UV。

在未打散 Key 的情况下，uid 的值会比较分散，但对于游客用户来说，uid 值为空，即 0，这就会造成 Map 的结果中，uid 分布不均匀。这时，将 uid 的值设为用户 ip 的后三位，从而使游客的统计均匀的分发到不同的 Key 中。但仅仅这样做是不够的，每个 uid 都不同，会导致 Map 结果中 Key 的数量过多，不便于数据合并和传输。这时，可以使用取模函数对 uid 取模，将 Key 的数量控制在合适的范围之内。进而分发到不同的 Reduce 节点上。

在 Reduce 阶段，Reducer 接收到 Key/Value 对时，可以对 Cookie 值进行判断，若 uid 为空，则访客数 +1，反之则该 uid 的统计数 +1。

具体实现流程如下图：

实际生产环境中，Key 的打散和控制还有其他多种方法，如将字符串转换为整形数字提高处理速度、或者将一个 MapReduce 操作划分成两份 MapReduce 程序去执行。具体使用方法应该根据应用程序的要求来确定，在此不一一赘述。

4.5　本章小结

第五章　总结与展望

5.1　本文总结

本文通过对基于 Hadoop 实现的 MapReduce 编程框架的介绍，详述了 MapReduce 编程思想及 Hadoop 的实现过程，并在此基础上使用两种实现方式实现了 Web 访问日志程序的编写与其性能分析。

根据第三章的测试和第四章的分析，可以得出以下四个结论：

1. MapReduce 运行效率高，分布式程序编写简单，是个优秀的编程框架。
2. I/O 可能会造成 MapReduce 性能瓶颈
3. 现阶段，原生 Java 比 Streaming 效率高
4. MapReduce 程序编写中，Key 的选择对数据合并、Reduce 分发有着至关重要的作用。

5.2　进一步的工作

1. 使用不同的 Hadoop 调度机制，测试其对集群的影响，找出并总结其调度方式所对应的应用场景。
2. 找出 Streaming 中负责数据合并的部分，理解其处理过程并进行适当优化。