6.1.1 分布式并行编程方式

在摩尔定律的作用下，以前程序员根本不用考虑计算机的性能会跟不上软件的发展，因为约每18个月，CPU的主频就会增加一倍，性能也将提升一倍，软件根本不用做任何改变，就可以享受免费的性能提升。然而，由于晶体管电路已经逐渐接近其物理上的性能极限，摩尔定律在2005年左右开始逐渐失效了，人类再也不能期待单个 CPU的速度每隔18个月就翻一倍，为我们提供越来越快的计算性能。Intel、AMD、IBM等芯片厂商开始从多核这个角度来挖掘CPU的性能潜力，多核时代以及互联网时代的到来，将使软件编程方式发生重大变革，基于多核的多线程并发编程以及基于大规模计算机集群的分布式并行编程是将来软件性能提升的主要途径。

许多人认为这种编程方式的重大变化将带来一次软件的并发危机，因为我们传统的软件方式基本上是单指令单数据流的顺序执行，这种顺序执行十分符合人类的思考习惯，却与并发并行编程格格不入。基于集群的分布式并行编程能够让软件与数据同时运行在连成一个网络的许多台计算机上，这里的每一台计算机均可以是一台普通的 PC 机。这样的分布式并行环境的最大优点是可以很容易的通过增加计算机来扩充新的计算结点，并由此获得不可思议的海量计算能力，同时又具有相当强的容错能力，一批计算结点失效也不会影响计算的正常进行以及结果的正确性。Google就是这么做的，他们使用了叫做MapReduce的并行编程模型进行分布式并行编程，运行在叫做GFS (Google File System)的分布式文件系统上，为全球亿万用户提供搜索服务。

我们知道，传统的程序设计基本上是单指令单数据流的顺序执行。

而基于集群的分布式并行编程能够可以并行执行大规模数据处理任务。

这里，我们对几个术语进行简要辨析：

并发是两个任务可以在重叠的时间段内启动，运行和完成。

并行是任务在同一时间运行，例如，在多核处理器上。

也就是说，并发是独立执行过程的组合，而并行是同时执行（可能相关的）计算。

并行编程和分布式编程是达到软件并发的两种基本途径。它们是两种不同的、有时又相互交叉的编程范例。

并行编程技术将程序必须处理的作业分配给一个物理或虚拟计算机内的两个或者多个处理器执行。

分布式编程技术则一般是将作业分配给集群内的不同计算机的一个或者多个处理器。

并行程序可以分成进程或者线程实现。而分布式程序仅能分成进程实现。

在技术上，并行程序有时候是分布式的，例如PVM编程。

分布式编程有时用于实现并行性，例如MPI编程。

但并非所有的分布式程序都包括并行性，分布式程序的各部分可以在不同时间间隔内的不同时刻执行。

Google公司发明MapReduce的初衷主要是为了解决其搜索引擎中大规模网页数据的并行化处理。Google公司在发明了MapReduce之后首先用其重新改写了其搜索引擎中的Web文档索引处理系统。由于MapReduce可以普遍应用于很多大规模数据的计算问题，此后，Google公司内部进一步将其广泛应用于很多大规模数据处理问题。到目前为止，Google公司内有上万个各种不同的算法问题和程序都使用MapReduce进行处理。

那么，在MapReduce出现之前，已经有那些较为成熟的并行计算框架？

传统并行计算框架包括MPI、PVM和CORBA等：

MPI(Message Passing Interface,消息传递接口）：MPI是一种基于消息传递的并行编程技术。并行机不一定在各处理器之间共享存储，当面向非共享存储系统开发并行程序时，程序的各部分之间通过来回传递消息的方式通信。要使得消息传递方式可移植，就需要采用标准的消息传递库，这就促成MPI的面世。MPICH是影响最大、用户最多的MPI实现。

PVM（Parallel Virtual Machine，虚拟并行计算机）：是集群编程的一种标准，是一个软件包，它允许一个异构计算机集通过网络连接在一起，使用起来如同一个单独的大规模并行计算机。

CORBA（Common Object Request Broker Architecture，公共对象请求代理体系结构）：是分布式跨平台面向对象编程的标准。

MapReduce与这些传统并行计算框架相比有哪些优势？

如表5.1所示，我们从集群架构/容错性/硬件/价格/扩展性/编程难易程度/适用场景等方面加以比对，MapReduce集群的构建完全选用价格便宜、易于扩展的低端商用服务器，而非价格昂贵、不易扩展的高端服务器。正是由于MapReduce集群中使用大量的低端服务器，因此，节点硬件失效和软件出错被认为是常态，MapReduce具有良好的容错性和可扩展性。此外，MapReduce提供了一种抽象机制将程序员与系统层细节隔离开来，程序员仅需描述需要计算什么（What to compute），而具体怎么去计算（How to compute）就交由系统的执行框架处理，这样程序员可从系统层细节中解放出来，而致力于其应用本身计算问题的算法设计。从适用场景来看，MapReduce适宜数据密集型，是大数据时代的海量数据处理利器。

综上，可以看出，MapReduce与这些传统并行计算框架相比具有显著的优势。

MapReduce的推出给大数据并行处理带来了巨大的革命性影响，使其已经成为事实上的大数据处理的工业标准。尽管MapReduce还有很多局限性，但人们普遍公认，MapReduce是到目前为止最为成功、最广为接受和最易于使用的大数据并行处理技术。

6.1.2 MapReduce模型简介

2003年和2004年，Google公司在国际会议上分别发表了两篇关于Google分布式文件系统和MapReduce的论文，公布了Google的GFS和MapReduce的基本原理和主要设计思想。

Hadoop实现了Google 的MapReduce编程模型，提供了简单易用的编程接口，也提供了它自己的分布式文件系统 HDFS,与 Google 不同的是，Hadoop是开源的，任何人都可以使用这个框架来进行并行编程。如果说分布式并行编程的难度足以让普通程序员望而生畏的话，开源的Hadoop的出现极大的降低了它的门槛，基于 Hadoop 编程非常简单，无须任何并行开发经验，可以轻松的开发出分布式的并行程序，并让其令人难以置信地同时运行在数百台机器上，然后在短时间内完成海量数据的计算。

MapReduce是面向大数据并行处理的计算模型、框架和平台，它隐含了以下三层含义：

1）MapReduce是一个基于集群的高性能并行计算平台（Cluster Infrastructure）。它允许用市场上普通的商用服务器构成一个包含数十、数百至数千个节点的分布和并行计算集群。

2）MapReduce是一个并行计算与运行软件框架（Software Framework）。它提供了一个庞大但设计精良的并行计算软件框架，能自动完成计算任务的并行化处理，自动划分计算数据和计算任务，在集群节点上自动分配和执行任务以及收集计算结果，将数据分布存储、数据通信、容错处理等并行计算涉及到的很多系统底层的复杂细节交由系统负责处理，大大减少了软件开发人员的负担。

3）MapReduce是一个并行程序设计模型与方法（Programming Model & Methodology）。它借助于函数式程序设计语言Lisp的设计思想，提供了一种简便的并行程序设计方法，用Map和Reduce两个函数编程实现基本的并行计算任务，提供了抽象的操作和并行编程接口，以简单方便地完成大规模数据的编程和计算处理

MapReduce采用“分而治之”策略，一个存储在分布式文件系统中的大规模数据集，会被切分成许多独立的小数据集，称之为分片（split），这些分片可以被多个Map任务并行处理。

MapReduce设计的一个理念就是“计算向数据靠拢”，而不是“数据向计算靠拢”，因为，移动数据需要大量的网络传输开销。通常，MapReduce框架的计算机节点和存储节点是运行在同一组节点上的，即MapReduce和HDFS运行的节点通常是在一起的，这样集群的网络带宽利用更加高效。

需要指出的实施，Hadoop框架是用Java实现的，但是，MapReduce应用程序则不一定要用Java来写。

MapReduce计算模型的核心是Map（映射）和Reduce（归约）两个函数。

Google的那篇MapReduce论文里的原文是：Our abstraction is inspired by the map and reduce primitives present in Lisp and many other functional languages。也即MapReduce的灵感来源于函数式语言（比如Lisp）中的内置函数map和reduce。

基于 MapReduce 计算模型编写分布式并行程序非常简单，程序员的主要编码工作就是实现 Map 和 Reduce 函数，功能是按一定的映射规则将输入的 <key, value> 对转换成另一个或一批 <key, value> 对输出。

这里，我们以统计一个计算文本文件中每个单词出现的次数的程序为例，<k1,v1> 可以是 <行在文件中的偏移位置, 文件中的一行>，经 Map 函数映射之后，形成一批中间结果 <单词，出现次数>, 而 Reduce 函数则可以对中间结果进行处理，将相同单词的出现次数进行累加，得到每个单词的总的出现次数。