基于 Hadoop 平台的云计算节能研究①

吴 岳

(国家林业局 林产工业规划设计院, 北京 100010)

摘 要: 云计算的广泛应用导致数据中心的产生. 数据中心的能效的高低不仅涉及到电费, 还关系到否符合环境法规. 作者通过修改 Hadoop YARN 编程模型, 使用 RAPL 的能耗限制功能来降低应用程序中计算失衡时的能耗. 目的是测试在不会明显地降低性能的条件下, 通过 RAPL 接口控制 CPU 的能耗是否有效. 通过实验表明, 在同样的负载下, Phadoop 架构在分块矩阵乘法上相对于原来的 Hadoop 架构的能耗降低了 34%.

关键词: 云计算; 数据中心; 能耗; Hadoop; YARN

Power-Saving of Cloud Computing Based on Hadoop

WU Yue

(Planning and Design Institute, Forest Products Industry State Forestry Administation, Beijing 100010, China)

Abstract: An increased adoption of cloud computing has lead to a greater concentration of hardware in massive named datacenters.It is essential for these datacenters to be energy efficient not only to cut down on electricity costs but also to be in compliance with environmental regulations.The author implemented an enhanced version of Hadoop YARN framework that utilizes RAPL's power capping feature to mitigate computational imbalances in an application and to reduce CPU power consumption, named Phadoop. The purpose of the experiment is investgating whether it is beneficial to use RAPL interfaces to conserve the energy consumption of a CPU in a cloud-based workload without significant loss of performance. Experimental results indicate a reduction in energy consumption of Phadoop up to 34% compared to Hadoop.

Key words: cloud computing; datacenter; power consumption; Hadoop; YARN

云计算技术因为其特有的灵活性、容错性与安全性, 在过去的几年里得到了广泛的关注. 当今的云计算服务提供商中, 亚马逊、微软、谷歌等大型企业提供了从应用软件、开发平台, 到基础架构等一系列产品. 云计算服务不仅降低了用户维护设备的开销, 还可以根据用户需求随时在线调整服务配置, 从而节约了时间.

在基于云的服务模式中, 服务提供商的硬件设备需要集中放置, 称为数据中心. 对数据中心有效的管理是为用户提供经济与高效服务的关键. 虽然硬件的集中放置可以提供快速的更新和恢复, 但是也带来了对供电与恒温的要求. 由于对计算资源需求的增加, 数据中心的耗电量不断上升, 耗电量已经成为数据中心运营的一项重要预算. 随着来自地方政府对环境监管压力的增大, 能效已经成为数据中心成功运行的关键挑战之一.

近年来, 云服务提供商们不断探索降低数据中心设备能耗的技术. 典型的节能技术和机制包括下面几种:

(1) 动态调节机制. 由于处理器能耗在服务器能耗中占了很大一部分, 处理器节能很早就受到重视并取得了不少进展. 处理器能耗由静态能耗和动态能耗两部分组成, 其中动态能耗与时钟频率和供电电压均成正比关系. 因此, 可通过改变时钟频率和供电电压来动态调节处理器能耗. 处理器动态调节技术在个人电脑和服务器领域已经有了比较成熟的产品, 比如 Intel 的 Enhanced SpeedStep 和 AMD 的 Power Now!等, 部分操作系统也添加了支持模块, 如 Linux 的 CPU

① 收稿时间:2015-02-28;收到修改稿时间:2015-04-02

freq 内核子系统等. 另外, 内存可以通过调节时钟频率、供电电压; 硬盘也可以通过调节转速实现节能, 即多转速磁盘技术, 不过与处理器调压调频技术相比这些技术使用的很少.

(2) 服务器休眠机制. 即使采用了动态调节机制, 空转的服务器仍有较高的能耗. 如果能将设备在完全不需要提供服务的特定时段内置于休眠状态, 将显著减少总体能耗. 由于休眠/唤醒机制在节能以及其他方面的广泛应用, Intel、Microsoft 等公司共同制定了 ACPI 规范以支持这一机制并形成了工业标准. 通过定义服务器的全局状态、设备电源状态和处理器电源状态, 服务器可以动态地管理电源.

(3) 负载调度优化. 负载调度优化主要是通过用虚拟化技术或服务调度迁移技术(将低负载服务器上的服务整合迁移到部分服务器上, 然后让无负载的进入休眠状态或者关机), 提高正常运转服务器的使用率, 达到数据中心整体节能的效果. 虚拟化技术目前已有很大的发展, 并且有很多成熟的企业级产品, 如开源的 Xen、VMware 公司的 VMware 系列、Microsoft 公司 Hyper-v 等. 借助虚拟机实时迁移技术和 VMware DRS 等虚拟机调度产品, 可以按照负载的变化进行虚拟机动态调度, 进一步提高了服务器使用率, 降低了数据中心整体能耗.

(4) 供电、制冷系统优化. 常见的如功耗封顶(Power Capping)技术, 通过对服务器功耗进行动态设置或者封顶, 来帮助用户动态分配数据中心里的电力和制冷资源. 用户可以根据服务器使用中的最大实际功耗来限定其功率上界, 减少不必要的过度供给, 而节省下来的电力可重分派给新的系统. 另外还有 UPS(不间断电源)改进(比如使用飞轮 UPS 取代铅蓄电池 UPS)、海水冷却、数据中心废热供暖等方法. 采用配有可调节电源接口的硬件设备或者可与电源管理工具协同工作的硬件设备都可以降低能耗. Intel 公司的 RAPL (Running Average Power Limit)技术就是这样的成果之一. RAPL 是通过硬件实现的机制, 最早用于 Sandy-bridge 处理器家族, 可以测量和限制 CPU 和内存的运行功率. 通常, 在一个数据中心的能耗峰值中, CPU 和内存的能耗分别约占了 33%和 30%, 降低 CPU 和内存的能耗能够提高数据中心的整体能效.

本文分析了在执行基于云的负载运算时, 使用 RAPL 接口减少 CPU 能耗的优势. 实验中使用的 Hadoop YARN 架构, 适合在大量数据聚合时执行 map-reduce 操作, 它简单的编程模型, 使用户能够轻松实现分布式应用程序.

作者首先对 Hadoop YARN 架构进行了修改, 减少运行应用程序的 I/O 开销与能耗, 修改后的 Hadoop YARN 框架称作 Phadoop 架构. Phadoop 架构使用一个进程池来处理 map-reduce 任务, 它支持缓存初始任务运行时从文件中读取的数据, 并在后续任务中重复使用. 这个修改优势体现在迭代的 map-reduce 任务中, 新的 map-reduce 任务从底层文件系统冗余地读取相同的输入数据. Phadoop 架构还包括一个实时系统用来调用 RAPL 服务, 用来动态地限制节点运行 map-reduce 任务时的能耗. 最后通过实验表明, Phadoop 架构在执行稀疏矩阵乘法上相对于原来的 Hadoop 架构的能耗降低了 34%.

1 Hadoop平台

Hadoop 是一个由 Apache 基金会所开发的分布式系统基础平台. 用户可以在不了解分布式底层细节的情况下, 开发分布式应用程序, 充分发挥集群的优势进行高速运算和存储. Hadoop 平台最核心的组成是 Hadoop 分布式文件系统(HDFS)和 Map/Reduce 编程模型. HDFS 为海量的数据提供了存储, 则 Map-Reduce 编程模型为处理海量数据提供了程序框架.

Hadoop 平台的优势主要体现在以下几个方面:
可靠性: Hadoop 是按位存储和处理数据的, 可靠性高;
扩展性: Hadoop 是在可用的服务器集群间分配数据并完成计算任务的, 这些集群可以方便地扩展到数以千计的节点中.

Hadoop 在很多大型网站上都已经得到了应用, 可以说是目前最为广泛应用的开源云计算软件平台.

2 Map-Reduce与HadoopYarn

Map-Reduce 是一种编程模型, 用于大规模数据集的并行运算. 它极大地方便了编程人员在不会分布式并行编程的情况下, 将自己的程序运行在分布式系统上.

如图 1 所示, 首先用户程序 (ClientNode) 提交了一个 job, job 的信息会发送到 Job Tracker 中. Job Tracker 是 Map-Reduce 框架的中心, 它与集群中的机器定时通信 (heartbeat), 需要管理所有 job 的失败、

重启等操作[1]. Map-Reduce 集群中每台机器都有 TaskTracker, 它主要是监视自己所在主机的资源情况. TaskTracker 需要把这些信息通过 heart beat 发送给 JobTracker, JobTracker 会搜集这些信息, 用来分配新提交的 job 运行在哪些主机上.

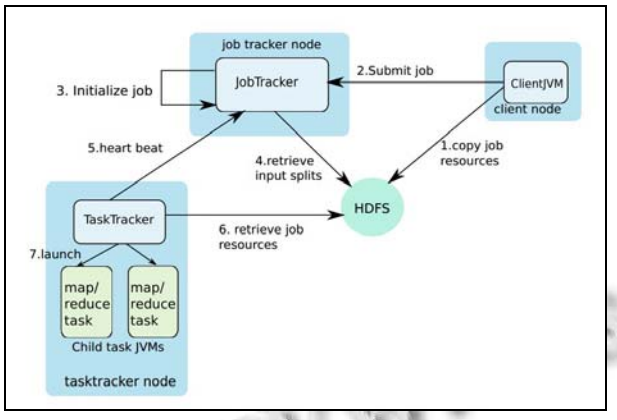


图 1 Hadoop Map-Reduce 编程模型

JobTracker 是 Map-Reduce 的集中处理点, 存在单点故障. JobTracker 完成了太多的任务, 造成了过多的资源消耗[2]. 在 TaskTracker 端, 以 Map-Reduce task 的数目作为资源的表示过于简单, 没有考虑到 cpu 和内存的占用情况, 如果两个大内存消耗的 task 被调度到了一块, 很容易出现内存溢出.

由于 Hadoop 的 Map-Reduce 框架存在着上述问题, 从 0.23.0 版本开始, 它被完全重构. 新的 Hadoop Map-Reduce 框架命名为 Map-ReduceV2 或者 Yarn, 其架构如图 2 所示.

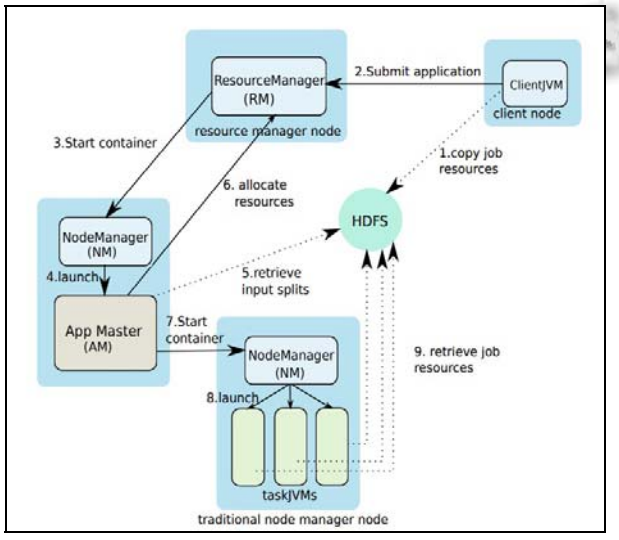


图 2 Hadoop YARN (Map-Reduce V2)编程模型

重构的思想是将 JobTracker 两个主要的功能分离成单独的组件, 这两个功能是资源管理和任务调度/监控[3]. 新的资源管理器全局管理所有应用程序计算资源的分配, 每一个应用的 ApplicationMaster 负责相应的调度和协调. ResourceManager 和每一台主机的节点管理服务器能够管理用户在那台主机上的进程. 每一个应用的 ApplicationMaster 是一个详细的框架库, 它结合从 ResourceManager 获得的资源和 NodeManager 协同工作来运行和监控任务.

HadoopYarn 框架相比较老版本的 Map-Reduce 框架有一些明显优势:

(1) 极大地减少了 JobTracker(ResourceManager)的资源消耗, 并且使监测每一个 Job 子任务(tasks)状态的程序分布式化, 从而更安全.

(2) HadoopYarn 框架, ApplicationMaster 是一个可变更的部分, 用户可以对不同的编程模型写自己的 AppMst, 让更多类型的编程模型能够运行在 Hadoop 集群中, 可以参考 HadoopYarn 框架官方配置模板中的 mapred-site.xml 配置.

(3) 对于资源的表示以内存为单位, 比以前用剩余 slot 数目为单位更合理.

(4) 老版本的 Map-Reduce 框架中, JobTracker 的一个主要负荷就是监控 Job 下的 tasks 的运行状况, 现在这个部分就交给 ApplicationMaster 完成, 而 Resource Manager 中有一个模块叫做 Applications Masters, 它负责监测 ApplicationMaster 的运行状况.

3 修改后的Hadoop YARN模型

作者首先对 Hadoop YARN 架构进行了修改, 减少运行应用程序的 I/O 开销与能耗, 修改后的 Hadoop YARN 框架称作 Phadoop 架构.

在 Hadoop YARN 模型中, 迭代的操作需要从文件系统冗余的读取输入数据, 带来了很多 I/O 开销. 例如, 如果在稀疏矩阵的乘法操作上执行 map-reduce, 迭代操作中的每个任务从输入矩阵 B 中读取相同数据块, 这种操作的读取范围包括多个磁盘, 磁盘的位置可能是本地的, 也可能是服务器集群中的[4].

为了降低 I/O 开销, 需要重新设计 Hadoop YARN 编程模型. 它的设计思路是: 将第一次 map-reduce 任务读取的数据保存在内存中, 并在相同的作业中重复调用.

Phadoop 架构使用进程池来执行 map-reduce 程序. 图 3 对比展示了在 Hadoop 架构和 Phadoop 架构中一次 map-reduce 任务的生命周期. 在原始的 Hadoop 架构中, 每次重复执行 map-reduce 任务都要重新创建进程; 在修改后的 Phadoop 架构中, 每次重复执行 map-reduce 任务时重新使用最初创建进程的数据, 而不再重新创建进程. Phadoop 架构提供的接口允许程序利用输入缓存来降低整体 I/O 开销.

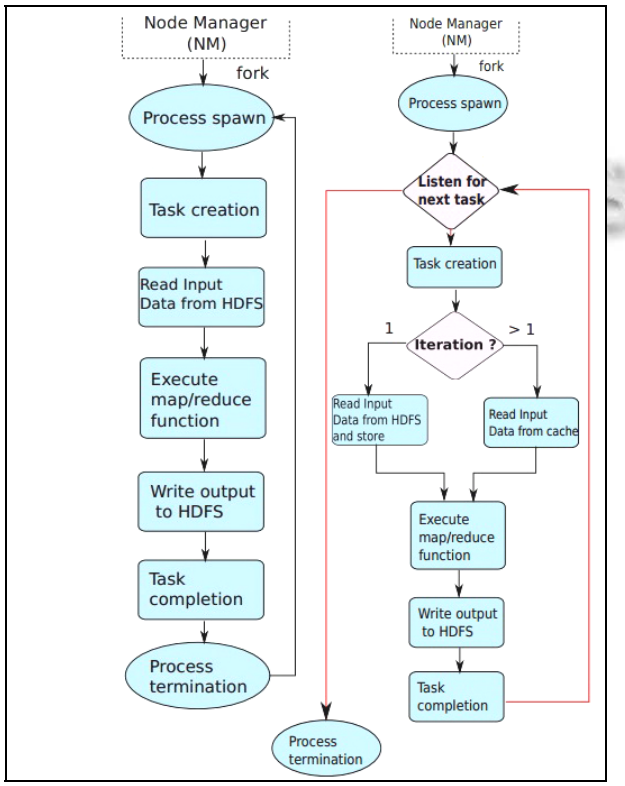


图 3　Hadoop 平台(左)和 Phadoop 平台(右)下 map/reduce 任务的生命周期

Phadoop 架构还包括一个实时系统调用 RAPL 服务, 可以封顶节点执行 map-reduce 任务时的能耗. 这个服务通过 RAPL 接口, 通过收集 map-reduce 任务运行数据来判断能耗峰值, 并动态地限制节点执行子任务时的能耗上限.

在 Hadoop YARN 架构中, ApplicationMaster 主要负责与 ResourceManager 协调任务执行时的资源分配, 并监测由 NodeManager 响应的 map-reduce 任务进展. 在 Phadoop 架构中, RAPL 服务不再作为独立的后台进程, 而被整合入 Application Master.

为了实现任务中输入数据的重复使用功能, 需要对在 Hadoop YARN 架构中的 NodeManager 和客户端 map-reduce 应用程序模块进行改进. 执行任务前需要启动 Container, NodeManager 控制着 Container 的运行. 客户端 map-reduce 应用程序按照 NodeManager 提供的参数执行子任务. 在迭代开始时, ContainerManager 派生出子进程所需的数量来执行 map-reduce 任务. 在节点的 loop-back 接口上, 每个进程根据所处理任务的识别码计算出一个端口号, 并在这个端口上进行监听.

例如, 16000 是某个节点产生的所有进程的基本端口号, 为了执行 ID 为 2 的任务而产生的子进程, 那么它将监听 16002 号端口. 当这些进程启动后, NodeManager 通过 TCP 套接字连接到一个子进程, 并发送任务执行参数(ApplicationMaster 的主机地址、端口号与环境变量). 当接收到这些参数后, 子进程使用它们设置或者重置执行环境, 包括当前任务的数据存储区路径与安全令牌的位置. 当子进程执行任务完成, 会向 ApplicationMaster 报告状态, 并监听在初始化时的端口上等待下一个任务. 在迭代开始时, 每个任务读取输入数据块, 并通过 ContextObject 将它们存储在内存中. 迭代中的后续任务被相应的进程重复使用这些数据, 从而明显降低了 I/O 的开销.

ApplicationMaster 使用 Umbilical 协议与它的子任务进行通讯. 实施 RAPL 服务需要修改 ApplicationMaster、子任务模块与 Umbilical 协议. 如图 4 所示, map-reduce 子任务向 ApplicationMaster 报告的信息, 包括主机名、运行数据包与执行时间. ApplicationMaster 中的 RAPL 决策引擎(RDE)使用这些信息结合节能策略来确定每个任务的执行时间. RDE 还会从每个数据包中选择一个单独的任务作为强制能耗上限(PLE), 用作该数据包的能耗极大值. 在下次迭代操作开始时, 这个对象被相应的子任务通过 Umbilical 协议传回 ApplicationMaster. PLE 使用对象执行时间以及任务的上一次执行时间来确定一个能耗上限, 并通过 RAPL 接口设置在数据包上. 能耗上限是基于每个数据包运行过程中收集的数据确定的. 为了便于收集信息, 比如在不同能耗限制下的执行时间, 应用程序的检查可以脱机执行[5].

4　实验分析

4.1　实验环境

实验中所使用的服务器配置为 8 核 Intel Xeon

E5-2667 处理器, 16GB 内存. 该 CPU 支持 RAPL 技术, 工作主频为 3.2GHz. 在该服务器中安装了 Phadoop 与 Hadoop 程序架构, 使用稀疏矩阵乘法作为负载, 迭代执行 map-reduce 任务, 监测 CPU 与内存工作时的性能与能耗.

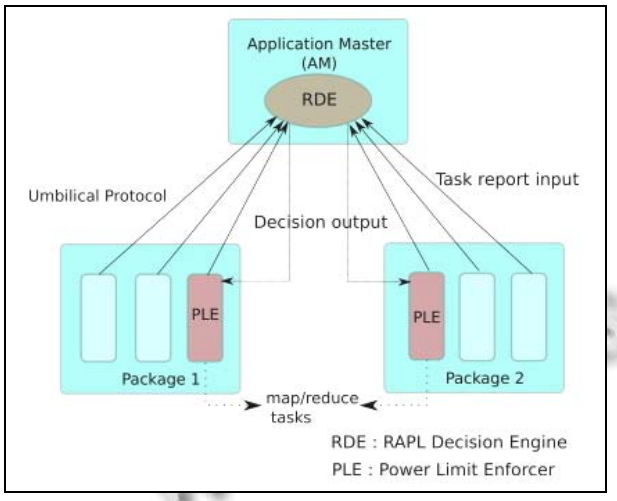


图 4 Phadoop 平台下 RAPL 实时系统运行交互示意图

4.2 实验方法

实验中的每次 map-reduce 任务都并行处理两种类型的输入, 分别是规则的数据块与不规则的数据块. 不规则数据块容易导致并发任务中的计算失衡. Phadoop 程序架构被运行在两种不同模式下, 分别是节能模式与缓存模式. 在节能模式下, Phadoop 架构运行实时 RAPL 服务并允许缓存. 在缓存模式下, Phadoop 架构不能运行实时 RAPL 服务.

在第一组实验中, 所有负载使用规则的数据块作为输入数据, 可以比 Phadoop 架构和 Hadoop 架构下较独立的 map-reduce 任务的执行时间. 这时 Phadoop 架构工作在缓存模式下, 可以显示出输入缓存的优势.

在第二组实验中, 所有的负载使用不规则的数据块作为输入数据. 因此, Phadoop 架构和 Hadoop 架构下的并发的执行 map-reduce 任务会产生不平衡, 从而触发实时 RAPL 服务.

通过统计 RAPL 服务监控的能耗与应用程序的总共执行时间, 可以计算出平均能耗, 即 Phadoop 架构和 Hadoop 架构中 CPU 执行每次操作时的能耗指数.

4.3 实验参数

矩阵乘法是线性代数中非常重要的一个操作原语. 许多图算法使用稀疏矩阵乘法作为基本构建单元, 比如图聚类、环检测、多源广度优先搜索、所有点对的最短路径等. 实验中选择稀疏矩阵乘法作为计算负载[6].

对稀疏矩阵乘法迭代地执行 map-reduce 操作. 每个并行进程更新矩阵乘积的数据块, 于此相反, 每个并行进程负责所有矩阵 B 中的单个块相乘.

以下公式用来描述稀疏矩阵乘法的实施与实验性评估. A、B 和 C 都是稀疏矩阵. $A \in S^{m*l}$、 $B \in S^{l*n}$、C=A*B. S 的上标表示一个稀疏矩阵的规模. A_{ij} 代表矩阵 A 的第 i 行第 j 列数据块. A(i;)与 A(;j)分别表示矩阵 A 第 i 行与第 j 列的所有数据块. 输入矩阵的块因数记作 b. nnz(A)表示矩阵 A 中非零元素的个数. P 表示处理器上运行的进程的集合, P_{ij} 表示运行在处理器第 i 行第 j 列的进程. P(i;)与 P(;j)分别表示运行在处理器上第 i 行和第 j 列的所有进程.

稀疏矩阵乘法中执行 map-reduce 任务分为两个阶段. 第一阶段包括对矩阵乘积进行部分求和的迭代运算; 第二阶段是对这些部分求和结果进行汇聚来计算出结果矩阵. 第一阶段中的每次迭代都是执行一次 map-reduce 任务, 将矩阵 A 中的一行数据块与矩阵 B 相乘. map 操作从输入矩阵中分开数据块, reduce 任务分配从矩阵 B 中分配一个单独数据块, 然后矩阵 A 中相应的数据块用来做矩阵块的部分乘积[7].

如图 5, 在单次迭代操作涉及到矩阵 A 的一个数据块与矩阵 B 的所有数据块. 当第 i 次迭代时, A(i;)被分配给 B_{kj}(其中, k =[0, l/b]并且 j=[0, n/b]), 并得到 A_{ik}. 当接收到矩阵 A 和矩阵 B 被分配的数据块时, 每个 reduce 任务都执行稀疏矩阵乘法操作并将片段的结果写入到 HDFS 中. 在第二阶段中, 所有迭代操作的片段结果汇聚为最终结果——矩阵 C.

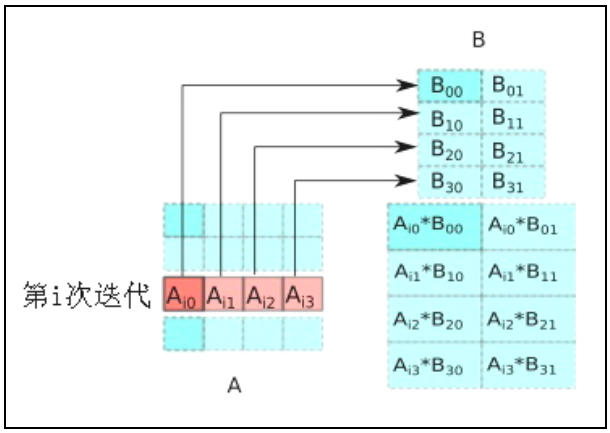
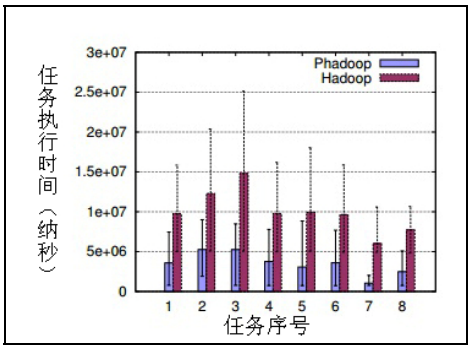
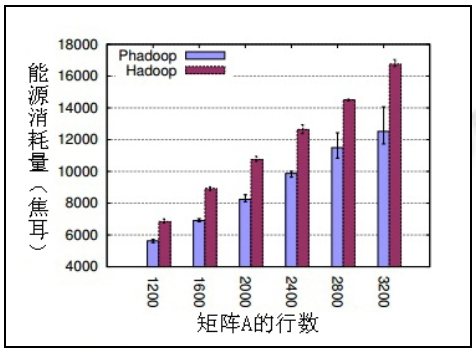


图 5 矩阵乘法中第 i 次迭代操作时的处理过程

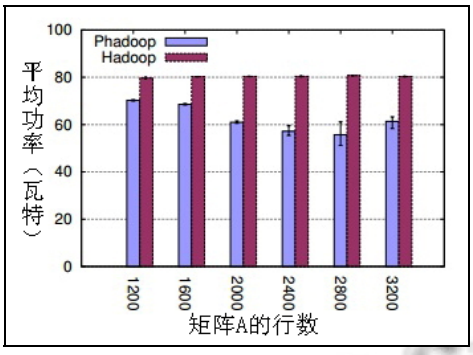
4.4 实验结果



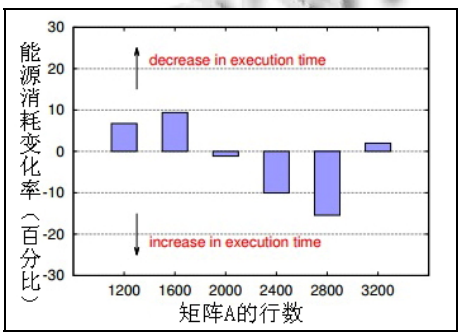
(a) Phadoop(缓存模式)与 Hadoop 对比图



(b) Phadoop(节能模式)与 Hadoop 对比图



(c) Phadoop(节能模式)与 Hadoop 对比图



(d) Phadoop(节能模式)与 Hadoop 对比图

图 6 实验结果对比图

4.5 结果分析

图 6 所示, 试验使用稀疏矩阵乘法作为负载. 应用程序中相乘的两个输入稀疏矩阵分别是 A 和 B, 分别使用随机数进行了初始化. 矩阵 B 的规模为 2000*2000, 矩阵 A 的规模为 n*2000, n 的范围是 1200 到 3200, 在程序每次运行时以 400 为增量增加. n 的变化导致了迭代次数的增加. 矩阵 A 的每个数据块尺寸为 400*500, 矩阵 B 的每个数据块尺寸为 500*1000. 生成矩阵的密度, 即非零元素的数量是均匀分布或二次分布的.

每次迭代 map-reduce 任务中的 map 操作负责从输入矩阵 A 和矩阵 B 中分配数据块给 reduce 操作, 实际上由 reduce 操作并行执行数据块乘法. reduce 操作将矩阵 A 的数据块与矩阵 B 的数据块相乘时, 待处理数据块的密度决定了计算负载. 当一个矩阵的数据块密度是变化的, 另一个矩阵的数据块密度是固定的, reduce 操作就会出现不平衡.

图 6(a)比较了 Phadoop 架构在缓存模式下与 Hadoop 架构执行稀疏矩阵乘法的时间. 图像中的 y 轴表示执行时间, x 轴表示任务索引. 在实验中, n 的值为 2000, 矩阵 B 的数据块规模 500*1000, nnz(Bij)为一个常数. 图中的柱状表示一次程序运行时所有迭代的平均任务执行时间. 柱状图上方的误差条表示相关任务执行时间的数值范围. 结果表明, 在执行效率上 Phadoop 架构平均比 Hadoop 架构高出 60%. 主要因为在每次迭代 reduce 操作中, Phadoop 架构缓存矩阵 B 数据, 而 Hadoop 架构需要重复从分布式文件系统中读取数据. 通过所有的迭代 reduce 操作, 矩阵 B 中的数据块 Bij 与矩阵 A 中的一列数据块 A(;i)相乘. 虽然矩阵 A 的密度是相同的, 但是每个数据块中非零元素的分布式不同的. 因此, reduce 任务执行整数乘法的次数在每次迭代中是变化的, 从而导致任务执行时间的变化.

在图 6(b)中, y 轴表示平均能耗, x 轴表示矩阵 A 的总行数. 在这些实验中, $nnz(B_{ij})=x2$, $x=20+(i+2*j)*30$. 根据这个等式, 矩阵 B 的数据块密度是二次变化的. 但是矩阵 A 的所有数据块都是均匀的密度, 即它们的非零元素个数相同. 矩阵 A 的数据块尺寸为 400*500, 当行数以 400 为增量增加是, 迭代的次数也由 1 开始增加. 矩阵 A 行数的增加引起迭代次数的增加, 无论 Phadoop 架构还是 Hadoop 架构中, 总执行时间都会增加. 通过图中可以看出, 在所以输入的条件下,

Phadoop 架构的能耗都低于 Hadoop 的架构, 大概降低了 17%到 25%.

图 6(c)展示了 Phadoop 架构和 Hadoop 架构下平均功耗的降低, 范围在 11%到 30%. 当矩阵 B 的所有数据块的密度呈二次曲线变化时, 每次并发执行 reduce 任务时矩阵 B 的数据块都是不同的密度. reduce 任务中出现了明显的计算不平衡引发了实时 RAPL 功能. 实时 RAPL 系统应用节能策略来降低平均功耗.

图 6(d)中, y 轴表示全部工作执行时间降低的百分比, x 轴表示矩阵 A 的行数. TPhadoop 表示 Phadoop 架构中全部工作执行时间, THadoop 表示 Hadoop 架构中全部工作执行时间, 那么下降百分比为 (THadoop-TPhadoop)/THadoop*100%. 正值表示减少, 负值表示增加. Phadoop 架构中总执行时间的减少因为缓存了输入值. 每次并发的 reduce 任务在迭代开始时, 存储矩阵 B 的数据块在内存中, 并在后续迭代中使用. 对于某些输入下百分比低于, 是因为 RAPL 服务的运行开销超过了缓存输入值节省的开销.

5 结语

在过去的几年中, 越来越多的企业采用了云计算. 这个趋势将会持续下去, 必然导致大量硬件集中放置运行, 数据中心将成为电力消耗的“大户”. 数据中心的能效的高低不仅涉及到电费和硬件维护成本, 还关系到否符合环境法规[8].

本实验的目的是测试在云计算的负载下, 如果不明显地降低性能, 通过 RAPL 接口控制 CPU 的能耗是否有效. 修改后的 Hadoop YARN 编程模型, 使用 RAPL 的能耗限制功能来降低应用程序中计算失衡时的能耗. 通过实验表明, 在同样的负载下, Phadoop 架构在稀疏矩阵乘法上相对于原来的 Hadoop 架构的能耗降低了 34%.

参考文献

1 金伟健,王春枝.适于进化算法的迭代式 MapReduce 框架.计算机应用,2013,12:3591–3595.

2 董新华,李瑞轩,周湾湾,王聪,薛正元,廖东杰.Hadoop 系统性能优化与功能增强综述.计算机研究与发展,2013,增刊 2:1–15.

3 许丞,刘洪,谭良.Hadoop 云平台的一种新的任务调度和监控机制.计算机科学,2013,01:112–117.

4 袁玉,崔超远,乌云,陈祝红.单机下 Hadoop 小文件处理性能分析.计算机工程与应用,2013,03:57–60.

5 王峰,雷葆华.Hadoop 分布式文件系统的模型分析.电信科学,2010,12:95–99.

6 刘正伟,文中领,张海涛.云计算和云数据管理技术.计算机研究与发展,2012, 增刊 1:26–31.

7 江务学,张璟,王志明.MapReduce 并行编程架构模型研究.微电子学与计算机,2011,6:168–170.

8 钱育蓉,于炯,王卫源,孙华,廖彬,杨兴耀. 云计算环境下软硬件节能和负载均衡策略.计算机应用,2013,12: 3326–3330.