异构大数据编程环境 Hadoop+

何文婷[1,2] 崔慧敏[1] 冯晓兵[1]

[1](中国科学院计算技术研究所 北京 100080)
[2](中国科学院大学 北京 100049)

摘 要 互联网和物联网技术的飞速发展开启了“大数据”时代。目前，硬件的高速发展催生了许多异构芯片，它们越来越多地出现在大规模数据中心里，支持不同的应用程序，在提升性能的同时降低整体功耗。文章重点介绍了基于 MapReduce 编程模型的 Hadoop+ 框架的设计与实现，它允许用户在单个任务中调用 CUDA/OpenCL 的并行实现，并能通过异构任务模型帮助用户。在我们的实验平台上，五种常见机器学习算法使用 Hadoop+ 框架相对于 Hadoop 能达到 1.4×～16.1×的加速比，在 Hadoop+ 框架中使用异构任务模型指导其资源分配策略，对单个应用负载上最高达到 36.0% 的性能提升；对多应用的混合负载，最多能减少 36.9%，平均 17.6% 的应用执行时间。

关键词 异构；数据中心；Hadoop+；MapReduce
中图分类号 TP 316.4 **文献标志码** A

Hadoop+: A Big-data Programming Framework for Heterogeneous Computing Environments

HE Wenting[1,2] CUI Huimin[1] FENG Xiaobing[1]

[1](*Institute of Computing Technology*, *Chinese Academy of Sciences*, *Beijing* 100080, *China*)
[2](*University of Chinese Academy of Sciences*, *Beijing* 100049, *China*)

Abstract The rapid development of Internet and Internet of Things opens the era of big data. Currently, heterogeneous architectures are being widely adopted in large-scale datacenters, for the sake of performance improvement and reduction of energy consumption. This paper presents the design and implementation of Hadoop+, a programming framework that implements MapReduce and enables invocation of parallelized CUDA/OpenCL within a map/reduce task, and helps the user by taking advantage of a heterogeneous task model. Experimental result shows that Hadoop+ attains 1.4× to 16.1× speedups over Hadoop for five commonly used machine learning algorithms. Coupled with a heterogeneous task model that helps allocate computing resouces, Hadoop+ brings a 36.0% improvement in data processing speed for single-application workloads, and for mixed workloads of multiple applications, the execution time is reduced by up to 36.9% with an average 17.6%.

收稿日期：2015-12-27 **修回日期**：2016-03-02
基金项目：国家重点基础研究发展计划(973)(2011CB302504)；国家高技术研究发展计划(863)(2012AA010902、2015AA011505)；国家自然科学基金(61202055、61221062、61303053、61432016、61402445)
作者简介：何文婷，博士研究生，研究方向为并行计算和机器学习算法；崔慧敏(通讯作者)，副研究员，研究方向为并行编程和编译，E-mail：cuihm@ict.ac.cn；冯晓兵，博士生导师，研究方向为编程模型和编译优化。

Keywords　heterogeneous; datacenter; Hadoop+; MapReduce

1 引　言

由于通信、网络、存储、传感器、各种便携设备的飞速发展，它们每天为我们收集到的数据越来越多，信息量以每五年10 倍的增速迅速膨胀[1]。2008 年，Google 每天处理 2 万 TB 的数据[2]；Facebook 的数据仓库中存储着多达 300 PB 的 Hive 数据，并以每天 600 TB 的速率增长[3]；2013 年后，大数据市场日趋成熟，其增长率由 2013 年的 60% 降至 2014 年的 40%，Wikibon 预测到 2020 年，大数据市场将达到 610 亿美元[4]。2013 年，我们所要处理的数据规模达到 4.4 ZB，到 2020 年，这一数字预计将达到 44 ZB。2013 年，全球数据中心总投资 1 513 亿美元，比 2012 年增加 8%，且仍处在快速增长中[5]。这些数据被称为所谓的“大数据”[6]，它具有海量的数据规模(Volume)、多样化的数据类型(Variety)、极快的数据增长速度(Velocity)、极大但是密度极低的价值(Value)等特点[7,8]。由于大数据潜在的巨大商用价值，相关的大数据处理技术被多个领域，如社交、医疗、国家安全、电子商务、金融服务、实时搜索等深入探索。

从应用的角度而言，大数据时代的来临不仅为商业模式带来了重要的变革，也对计算模式提出了巨大的挑战，它改变了服务器的使用模式。因此，云服务平台的提供商在提供功能强大的服务器的同时，也需要提供一个便捷友好的开发环境。MapReduce[9]以其易于开发及在大规模集群中易于布署的特性正在成为数据中心中一个日益重要的大规模并行编程模型。它在非计算密集型应用领域(如日志分析、用户画像等)及计算密集型领域(如深度学习等)都得到了广泛应用。

从硬件的角度而言，一方面芯片的并行度在急剧提高，计算机芯片被迅速推进到多核/众核时代；另一方面，芯片越来越多元化。在这两个因素的推动下，数据中心也呈现出异构的特征，即多种多核/众核芯片同时共存于同一个数据中心，以有效支持各种不同的应用程序，在提升性能的同时降低数据中心的功耗。

但如何使异构加速器强大的计算能力能够在大数据处理中发挥作用，仍然是一个巨大的挑战。研究人员针对图形处理单元(Graphics Processing Unit，GPU)和 MapReduce 的结合开展了诸多的研究工作。但是这些工作或集中在一个 GPU 内部，如 MARS[10]、MapCG[11]，它们利用 GPU 的多线程技术，给每个 GPU 线程分配一部分<key, value>对进行处理；或对 MapReduce 的模型进行了修改，需要程序员重新适应新的编程模型，如 Phoenix[12]、Mesos[13]、Orchestra[14]、Dyrad[15]等。

近年来，众多的研究人员开始研究在云计算的环境下，如何在 MapReduce 中对 GPU 集群进行功能和性能上的支持，如 HadoopCL[16]、MRCL[17]、HAPI[18]等，其思想在于将 OpenCL 与 MapReduce 相结合，通过源源变换自动从 MapReduce 代码生成 GPU 上所要执行的代码，以更好地利用异构集群中的加速部件，但需要程序员静态指定所需要使用的加速器类型。HadoopCL2[19] 在 HadoopCL 的基础上增加了运行时动态的内存分配，加速器调度模块。Glasswing[20]通过 OpenCL 调用多核中央处理单元(Central Processing Unit，CPU)和加速器协作完成应用，并引入管道机制，让计算时间和数据交换时间重叠，更进一步提高并行的程度。Hadoop+[21]框架扩展了 Hadoop Yarn 的资源管理模块，允许用户为单个 Map/Reduce 任

务提供 CUDA/OpenCL 的显式并行实现，并在此基础上构建异构任务模型预测任务在共享资源竞争下的数据处理速度，帮助用户选择最佳的任务配置。

本文的后续章节安排如下：第 2 节介绍 Hadoop+ 框架的实现，包括 Hadoop+ 框架的编程接口和总体工作流程；第 3 节介绍 Hadoop+ 框架中的资源管理，重点介绍 Hadoop 框架扩展的 GPU、内存分配、IO 资源的管理；第 4 节评测了在本文的实验平台上，5 种常见的机器学习算法在 Hadoop+ 中的实现相对于 Hadoop 所能取得的性能，并测试了混合负载在异构任务模型指导的资源分配策略下相对于先来先服务策略分配 GPU 能取得的性能提升；第 5 节给出了异构大数据编程环境的相关工作；第 6 节总结全文。

2 Hadoop+ 框架实现

本节介绍 Hadoop+ 框架，它扩展了 Hadoop 框架，允许用户在 Map/Reduce 任务中显式调用 CUDA/OpenCL 并行加速的任务实现，能够利用在异构集群中不同的加速器，提升原来 MapReduce 程序的并行度。

图 1[21]是 Hadoop+ 框架的流程图。除了向下兼容 Hadoop 框架中对 MapReduce 中的 Map 和 Reduce 函数接口的支持，Hadoop+ 框架还为编程者提供了 PMap 和 PReduce 函数接口，显式地允许编程者指定使用 CUDA/OpenCL 等并行技术加速程序的执行。由于框架以相同的方式提供对 PMap 和 PReduce 的支持，简单起见，下文以 PMap 为例阐释 Hadoop+ 框架的编程接口和整体工作流程。

2.1 Hadoop+ 框架的编程接口

Hadoop 编程框架中，一个 Map 函数的输入是一个<key, value>对，而在 Hadoop+ 中，一个 PMap 函数的输入则是一系列<key, value>对组成的数据集，但 Reduce 函数和 PReduce 函数的输入相同，都是所有 Map 任务的输出中具有相同 key 值的数据。因此，PMap 和 PReduce 函数可以被定义为如下形式：

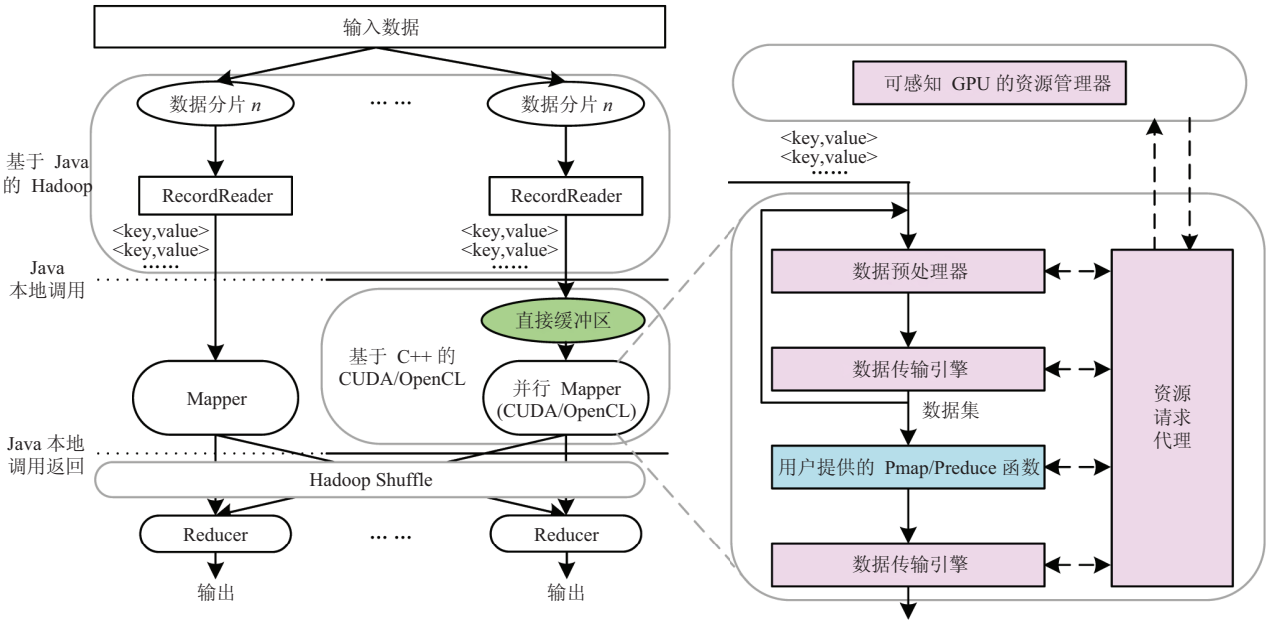

图 1 Hadoop+ 框架流程图

Fig. 1 Overview of Hadoop+ framework

PMap: (k1, v1) * —>(k2, v2) *

PReduce: (k2, (v2)*) —>(k2, v3) *

其中 * 表示前面一个数据/数据对的列表。以 K-近邻算法(K-Nearest Neighbor，KNN)应用为例，其在 Hadoop 和 Hadoop+ 中的实现伪代码分别如**算法 1** 和**算法 2** 所示。

算法 1　KNN 在 Hadoop 中的实现

```
function Map(train, TEST)
    Compute All Distances(train, TEST)
end function
function Reduce(test, Distance of (test, TRAIN))
    Select TopK(Distance(test, TRAIN))
end function
```

算法 2　KNN 在 Hadoop+ 中的实现

```
function PMap(train dataset, TEST)
    for all train in train dataset do
        Compute Distances CUDA(train, TEST)
        Insert Distance CUDA(train, TEST) to TopK(TEST)
    end for
end function
function Reduce(test, Distance of (test, TRAIN))
    Select TopK(Distance(test, TRAIN))
end function
```

在**算法 2** 中，PMap 函数以一个数据集而不是**算法 1** 中的单个数据为输入，并在计算时，显式地调用 CUDA Kernel 加速的 KNN 算法的实现。

2.2　Hadoop+ 框架的总体工作流程

Hadoop+ 扩展了 Hadoop 中的资源管理器，以非抢占的方式管理集群中的 GPU 资源，如图 1 的“可感知 GPU 的资源管理器”模块所示。运行时，每个 Map 任务通过“资源请求代理”申请相应的计算资源，若能成功申请到 GPU，则使用 PMap 的实现调用 CUDA Kernel 完成计算，否则任务用原来的 Hadoop 实现。

和 Hadoop 一样，Hadoop+ 也把输入数据划分成多个分片(Split)，每个并行的 Mapper 处理一个分片的数据。如果一个应用的 Map 任务没有 PMap 实现，则运行时采用原有的 Hadoop 中的实现，串行地处理每个<key, value>对的数据。

在 PMap 实现中，函数使用了 Java 的直接缓冲区(Direct Buffer)机制，每个输入数据分片中的数据由 Record Reader 解析成一系列<key, value>对，并攒成一个<key, dataset>存入直接缓冲区，作为 PMap 函数的输入。直接缓冲区为本地内存，Java 端和本地代码均可访问，避免了缓冲区复制，它不是分配在堆上的，因此不被 Java 虚拟机的垃圾回收机制直接管理，只有直接缓冲区相应的 Java 对象被垃圾回收时，操作系统才会释放直接缓冲区所申请的空间。

数据存入直接缓冲区后，框架通过 Java 本地调用(Java Native Interface)使用本地的函数(如 CUDA/OpenCL 等)进行相应的数据处理和计算，具体分为 4 步：

(1)数据预处理器：调用框架内置或用户自定义的函数进行数据解析，包括文本解析、Java 和本地平台间数据的大小端转换等，将 Java 端传来的数据转换成本地代码计算所需的格式；

(2)数据传输引擎：将经过预处理的数据积累到由“资源请求代理”请求到的计算部件(如 CPU/GPU 等)上对应的存储空间(主机内存/GPU 的全局内存等)；

(3)用户提供的 PMap/PReduce 函数被调用，计算出中间结果<key, value>对，通过数据传输引擎传回 Hadoop+ 框架，进行洗牌等后续计算；

(4)数据传输引擎将第(3)步中的计算结果拷贝到另一个直接缓冲区，结束 Java 本地调用，由 Mapper 的后续代码将计算结果根据作业的划分(partition)函数写到本地磁盘的不同分区，等待洗牌等后续计算。

上述的几个模块详见图 1 的右侧对图 1 左侧并行 Mapper(CUDA/OpenCL)模块的扩展说明。框架对 PReduce 也以类似的方式支持并行，如果用户没有指定 PMap/PReduce 函数，则可用 Hadoop 版的 Map/Reduce 函数完成计算。

3 Hadoop+ 框架中的资源管理

在 Hadoop 分布式文件系统(Hadoop Distributed File System，HDFS)中，数据以 Block 为单位组织。考虑到负载均衡、数据的局部性等因素，传统的 Hadoop 框架中，一个任务所要处理的数据分片的大小通常与 HDFS 的 Block 大小相当。而 Hadoop+ 框架中，为了高效利用 GPU，一个任务处理的输入数据分片大于传统的 Hadoop 框架中的任务。与在 CPU 上运行的 Hadoop 任务不同，运行在 GPU 上的任务在任务启动后，先从 HDFS 上读取数据，攒到系统内存中，在数据读取阶段结束后将整个数据分片的数据拷到 GPU 的全局内存，减少系统调用 CPU 与 GPU 间的数据拷贝函数的开销，提高 GPU 资源的使用效率。因此，与 Hadoop 框架中的 CPU 任务相比，运行在 GPU 上的任务需要给每个任务分配更大的内存。此外，Hadoop+ 框架需要对异构计算资源(以 GPU 为例)和系统 IO 资源的管理，提高系统资源的使用率，并减小同时运行的 GPU 任务由于系统 IO 资源竞争导致单个任务性能下降幅度，限制整体性能的提升。

3.1 计算资源管理

Hadoop+ 框架扩展了 Hadoop 的资源管理模块，集群中，每个从结点在每个心跳时向资源管理器更新当前本结点上可用的 GPU 和已被占用的 GPU 信息。图 1 所示的可感知 GPU 的资源管理器模块将集群中所有可用的 GPU 资源以 GPU 资源池的形式进行维护，集群中的所有 GPU 资源被表示成＜主机编号，GPU 编号＞的形式，并被标记成不可抢占的。可感知 GPU 的资源管理器用先来先服务的方式处理所有 GPU 资源请求，保证一个 GPU 同时只会被一个 PMap/PReduce 任务占用。此外，可感知 GPU 的资源管理器维护一个异构任务模型[21]，控制系统中各应用的任务在 CPU 和 GPU 上的执行比例，保证系统资源的利用率较高，并且任务间由于共享资源竞争造成的性能下降最少。

Map/Reduce 任务运行时，资源请求代理检测系统中是否存在由用户提供的 CUDA/OpenCL 并行加速的任务实现，若有，则向可感知 GPU 的资源管理器请求 GPU 资源；若成功申请到 GPU，则任务执行 PMap 实现的并行版本。否则，使用任务的 Hadoop 实现完成计算。

3.2 存储资源管理

图 2 是 Hadoop+ 中执行在 GPU 上的任务需要分配在 CPU 上的缓冲区(如蓝色模块所示)，和需要分配在 GPU 上的缓冲区(如紫色模块所示)。

为了减少系统调用 CUDA 内存拷贝函数的次数及相应的系统调用的开销，框架首先为任务在内存中分配一个直接缓冲区(Direct Buffer)，Java 代码和本地代码均能访问该块内存中的数据。任务开始执行后，首先由用户指定的数据预处理函数把由 RecordReader 从 HDFS 文件中读到的<key, value>对数据逐一存放到直接缓冲区中，待直接缓冲区存满后，调用用户提供的预处理函数，把缓冲区中的数据经过预处理后传输到 CPU 上由用户的本地代码在内存上分配的预处理数据缓冲区(Pre-Processed Data Buffer)。本地代码在 CPU 主存中分配的工作缓冲区(Working Buffer on CPU)由多块连续的预处理数据缓冲区及一个元数据缓冲区(Meta Data Buffer)组成。其中，元数据缓冲区中存储描述各预处理缓冲区的首地址相对于工作缓冲区首地址的偏移量，及要传输到 GPU Kernel 中的相关参数。

整个输入数据分片的数据经过预处理，传输到 CPU 上的“工作缓冲区”后，调用 CUDA 的内存拷贝函数，把数据一次性拷到 GPU 的工作缓冲区，减少该系统调用的次数和开销，之后调用用户提供的 GPU Kernel 函数，完成相应的计算。GPU Kernel 的数据将以与上文相应的方式传到输出直接缓冲区(Output Direct Buffer)中，由

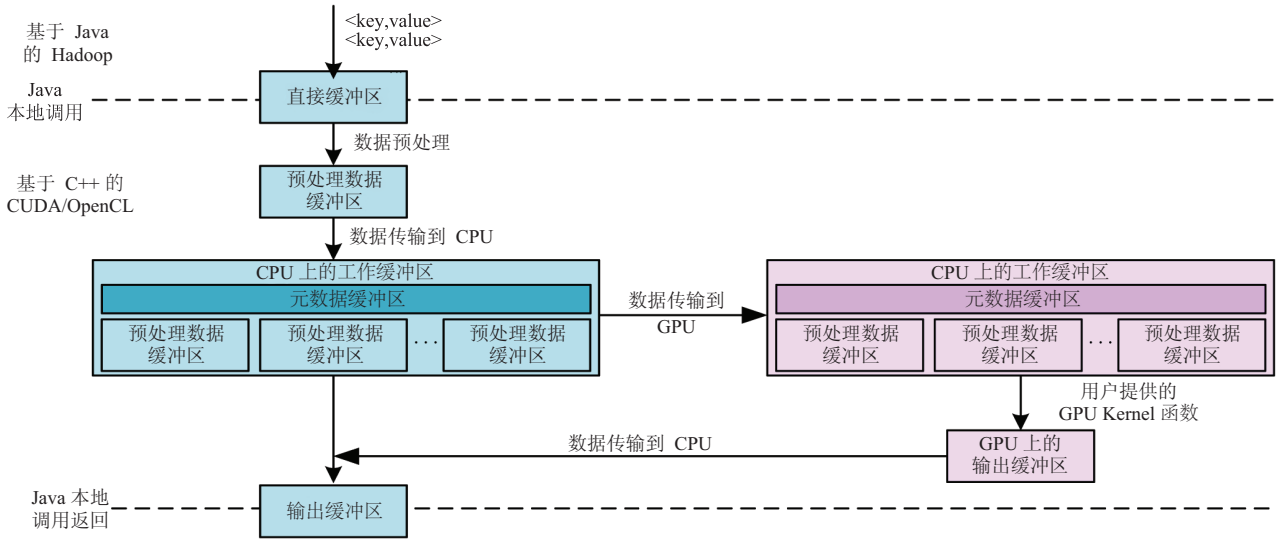

图 2　Hadoop+ 中的存储管理

Fig. 2　The memory management in Hadoop+

Java 代码输出，继续执行后续的混洗流程。

3.3　I/O 资源管理

由 3.2 节的叙述可知，Hadoop+ 中，执行在 GPU 上的代码会在任务开始执行时，从 HDFS 上读取数据，经过数据预处理和相关的格式转换存放到主存中。因此，多个同时启动的 GPU 任务会在任务的开始执行阶段竞争系统的 IO 资源，并由于共享资源的竞争导致单个任务的性能下降。

He 等[21]提出的异构任务模型可以根据当前系统中发出 IO 请求的任务数和系统中的总体 IO 请求大小预测任务在系统中运行时的执行时间，从而帮助用户根据不同的目标选择合适的任务配置。根据该模型，混合负载中单个任务发出的 IO 请求较大时，Hadoop+ 可以根据该模型保持同一结点上的 IO 带宽利用率较高，而单个运行在 GPU 上的任务执行时间延迟较小，从而提高系统的整体吞吐率。

3.4　Hadoop+ 的优势

Hadoop+ 提供了新的函数接口 PMap/PReduce，允许用户显式地调用 CUDA/OpenCL 的并行实现，相对于传统的 Hadoop 和其他相关工作，具有以下优势：

（1）自动协同 GPU 与 CPU 工作：现有的 Hadoop 框架在设计时未曾考虑利用集群中除 CPU 以外的计算资源，因此无法直接调用现有的 CUDA/OpenCL 程序。在 Hadoop+ 中，每个 Map 任务在检测到 PMap 的 CUDA/OpenCL 实现并成功分配到 GPU 资源后，会调用相关的 CUDA Kernel，执行相应的计算；若用户没有为应用的 Map 任务提供 CUDA/OpenCL 实现或者该任务没有申请到 GPU 资源，则任务按照原来的 Hadoop 实现完成数据处理。因此，Hadoop+ 能为用户提供统一的编程接口，充分利用异构集群中的计算资源完成大规模数据处理。

（2）整合强大的库函数：高性能计算时代，领域专家们为各领域中的常用函数编写了很多经过深度手工优化的库函数，相比于 HadoopCL[16]、HadoopCL2[19]等相关工作中直接把用户的 Java 字节码源源变换成 OpenCL Kernel 加速计算，Hadoop+ 允许 CUDA/OpenCL 函数的显式调用可以在大规模数据处理中充分发挥这些经过深度优化的库函数性能，避免了源源变换对程序中所使用的数据结构等的限制，并且，这些库函数的调用减少了很多相关 MapReduce

应用的代码编写，提高了生产效率。与传统的GPGPU 集群编程相比，在 Hadoop+ 框架中调用这些函数实现可以省去程序员手动管理程序的数据切分，主存中的内存分配与管理，结点间通信的负担，将用户在单机上的程序轻松扩展到集群上工作。

(3)控制任务配置模型：Hadoop+[21]提出了异构任务模型，通过系统的实时负载预测任务在共享资源竞争下的执行时间，从而帮助各应用选择不同用户目标下的最佳任务配置。

4 实验结果与分析

4.1 实验环境

本文实验所用的异构集群由 1 个主结点和 6 个从结点组成。每个从结点的处理器为 Intel 2.00 GHz Xeon E5-2620 CPU，它有 6 个物理核，支持超线程技术，最高可同时支持 12 个硬件线程。该 CPU 支持 4 个内存通道，并配置了 4 条 1 333 MHz 的 8G 内存，最高内存带宽为 42.6 Gb/s；每个结点的硬盘为 1 000 GB SATA 机械硬盘，最高读/写速率为 128 MB/s。每个结点有 2 个 NVIDIA Tesla C2050 GPU，每个 GPU 有 3 GB 的全局内存、14 个流式多处理器(Streaming Multi-processors)，每个流式多处理器上有 32 个流处理器(Streaming Processors)、32 768 个寄存器和 48 KB 共享内存。

Hadoop+ 框架是基于 Hadoop-2.1.0-beta 版的实现进行扩展的。程序的 Hadoop 实现使用 MultithreadedMapper，在程序启动前根据异构任务模型确定每个多线程任务的线程数，每次程序运行时设置启动 1 个 Reducer，每个 Mapper/Reducer 的 Container 分配的物理内存是 2.5 GB，可以使用的最大/最小堆内存为 256 MB，Hadoop+ 框架中分给每个并行的 Mapper Container 的数据分片攒成一个 数据集作为 PMap 函数的输入。由于 Hadoop+ 框架允许异构平台上同时运行不同计算资源执行的 Container，它们的运行速度相差甚多，本实验关闭任务的推测执行，以防运行在 CPU 上的 Container 总是被杀死，造成不必要的资源浪费。

4.2 测试程序

我们在 Hadoop+ 中实现了 5 种常用的机器学习算法，具体如下：

(1)K-近邻(KNN)：机器学习领域常用的分类算法，该算法对给定的训练数据集和测试数据集，在训练集中找出每个测试样本最近的 K 个邻居，以这 K 个邻居样本所属分类为测试样本进行分类。所用的 Hadoop 参考实现来自 Cover 等[22]文献。为公平起见，参考实现中也在 Map 阶段加入了最大堆，保留当前数据分片中所有测试样本的 K 近邻，以减小混洗阶段的数据量。测试数据为 128 维向量，训练样本集大小 628 978 770，测试样本集大小为 448，K=4，每个任务的数据分片大小为 1 024 MB。

(2)K 均值(Kmeans)：机器学习领域常用的聚类算法，该算法将数据集中的样本划分为 K 个集合，每个集合内的样本属于同一类别。所用的 Hadoop 参考实现来自 Mahout[23]，所用测试数据为 19 489 756 692 个 4 维向量，类别 K=8 192，每个任务的数据分片大小为 1 024 MB。

(3)向量相似(RowSimilarity，RS)：该算法是推荐系统中的常用算法，它计算输入向量间的相似度，此处我们用两个输入向量的 cosin 值表示其相似度，参考实现来自 Mahout[23]，输入数据是 5 062 656 个 16 777 216 维稀疏向量，稀疏度为 0.062 5，每个任务的数据分片大小为 256 MB。

(4)朴素贝叶斯(Naive Bayes，NB)：该算法是文本分类领域常用算法，它用文本中的词频作为特征，判断文本所属分类。参考实现来自 Mahout[23]，所用文本总大小为 618 GB，类别数为 20，单个任务输入数据大小为 1 024 MB。

(5) 反向传播(Back Propagation，BP)：人工神经网络领域常用的训练算法，它计算网络中所有权重值的损失函数的梯度，迭代更新网络权重值，以最小化损失函数值。所用参考实现来自 Liu 等[24]文献，输入数据是 6 099 586 个 48 维向量，隐藏层和输出层结点数分别为 65 536 和 48，单个任务的输入数据分片大小为 4 MB。

由于所选的 5 种机器学习应用都是 Map 阶段的执行占整个应用的运行时间的大部分，我们为其 Map 任务的实现提供了 CUDA 并行的实现，以利用异构集群中的 GPU 资源对 Map 阶段进行加速，而 Reduce 任务仍用原来的 Hadoop 实现，只使用系统中的 CPU 资源完成任务。在下文中，我们对 Hadoop+ 框架分别在单个应用和多应用的混合负载中进行了测试，结果如 4.3～4.5 节所示。

4.3　Hadoop+ 框架中单个应用的性能评估

本节中，我们测试了 4.2 中 5 种应用在 Hadoop+ 中的实现相对于 Hadoop 实现的性能提升，如图 3[21]所示。为公平起见，我们为应用的 Hadoop 实现取了其在平台上的最优配置——每个从结点同时执行 6 个 CPU 任务。

在 5 个测试程序上，Hadoop+ 中的实现相对于它们各自的 Hadoop 实现取得了 1.4×～16.1× 的加速比，如图 3 所示。应用程序在 Hadoop+ 上所能取得的性能提升取决于应用本身的特征：计算通信比和计算部分 GPU Kernel 相对于 CPU 实现的加速比。计算通信比越高，计算部分使用 GPU 能获得性能提升越多的应用在实验中所获得的加速比越大。

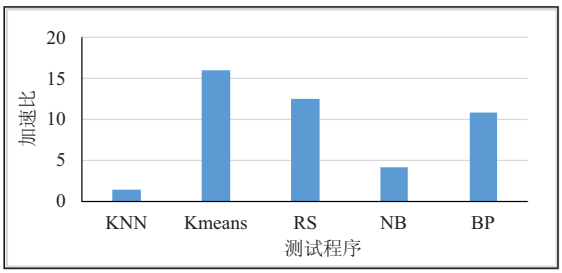

图 3　测试程序在 Hadoop+ 上的实现相对于 Hadoop 的加速比

Fig. 3　Speedup of Hadoop+ over Hadoop

4.4　Hadoop+ 的资源管理策略评估

图 4[21]是 5 种机器学习应用使用 Hadoop+ 的异构任务模型启发的资源管理策略相对于集群上的默认配置(每个结点同时启动 2 个 GPU 任务和 4 个单线程的 CPU 任务时的数据处理速度，如每簇中左边的深色柱子所示)所取得的性能提升。图中 *g*G-*c*C-*t*T 表示每个结点同时运行 *g* 个 GPU 任务和 *c* 个 *t* 线程的 CPU 任务。每簇中右边的浅色柱子表示应用在异构任务模型启发的资源管理中所选的最优配置能取得的归一化的数据处理速度。RS 应用的最优配置就是其默认配置，Kmeans 和 BP 应用在由异构任务模型的指导选择的最优配置下所取得的性能提升分别只有 1.1% 和 0.7%。但对于使用 GPU 加速不显著的应用 KNN 和 NB，使用异构任务模型指导的资源管理策略能分别取得 36.0% 和 8.3% 的性能提升。这主要是由于 KNN 应用的单线程 CPU 任务对系统的 IO 资源需求较大，与 GPU 任务同时运行会延长 GPU 任务的数据读取时间，影响平台的整体数据处理速度；而 NB 应用的 CPU 任务在多核 CPU 上可扩展性好，使用 3 个 2 线程的 CPU 任务与 2 个 GPU 任务同时运行可以最大程度地使

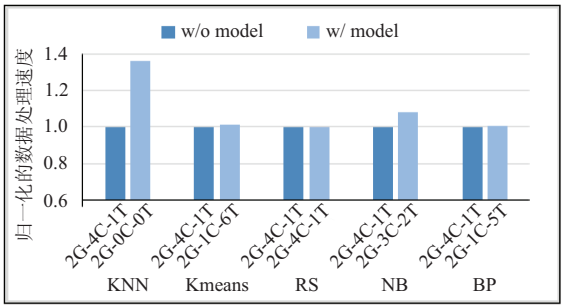

图 4　Hadoop+ 的资源管理策略取得的性能提升

Fig. 4　Performance improvement by leveraging the resource management strategy in Hadoop+

用平台上的 IO 资源和多核 CPU 计算资源。

4.5 Hadoop+ 中多应用的混合负载的性能评估

我们取 4.2 节中的程序两两组合，产生了由 C_5^2=10 种测试程序的不同组合构成的混合负载，并评测这 10 种负载中使用 Hadoop+ 框架的性能。我们分别使用先来先服务策略和 He 等[21]提出的异构任务模型指导的资源分配策略在异构集群中把 GPU 资源分配给混合负载中的不同应用。前者把 GPU 资源依次分配给按时间顺序提交到系统中的应用程序，后者则把 GPU 资源优先分配给在异构集群中相对于 Hadoop 实现能取得较高加速比的应用，而把 CPU 资源全部分配给混合负载中的另一个应用。两种策略下混合负载的执行时间归一化到各负载使用先来先服务策略的执行时间，如图 5[21]所示。

各簇中，左侧深色的柱子是混合负载在先来先服务调度策略下执行的时间，右侧浅色的柱子是混合负载在异构任务的资源分配策略下的执行时间。使用异构任务模型后，10 种负载的执行时间最高可减少 36.9%，平均减少 17.6%。

对于每个负载，如果负载中的两个应用都能在 GPU 上取得显著的性能提升，则使用异构任务模型指导的资源分配策略不能获得额外的加速比。如混合负载 Kmeans＋RS、Kmeans＋BP、RS＋BP，负载中的两个应用的任务基本都在系统中的 GPU 上完成。

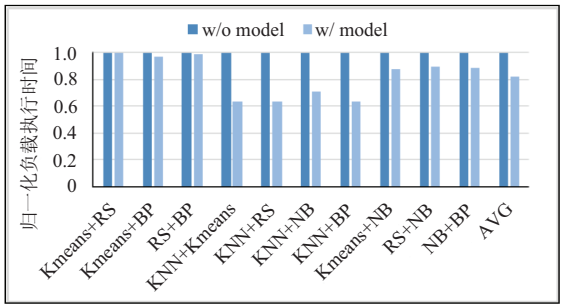

图 5 混合负载在 Hadoop+ 上使用异构任务模型的性能提升

Fig. 5 Performance Improvement by leveraging the heterogeneous task model in hybrid workload with Hadoop+

相对地，如果负载中的两个应用在 GPU 上能取得的性能提升差异明显，则使用异构任务模型指导 GPU 资源的分配能获得不同程度的性能提升。混合负载中，可以通过 GPU 获得较大加速比的应用优先获得系统中的 GPU 资源，而另一个应用也能在 CPU 上有较好的性能表现，因此负载的整体执行时间减小。图 5 中的另外 7 个负载属于这种情况。

5 相关工作

5.1 单结点的异构多核 MapReduce 编程框架

在单结点的 GPGPU 平台上，Catanzaro 等[25]基于 CUDA 实现了 MapReduce 编程模型，支持编程者只提供 Mapper 和 Reducer 的 CUDA Kernel，由框架完成程序的整体并行；Mars[10]利用了 GPU 硬件的高并行度，编程者只需定义一些类似于基于 CPU 的传统 MapReduce 框架的应用程序接口(API)，其他的如 GPU 运行时的实现细节由框架完成，对编程者透明。但它只是把 GPU 看作一般的众核处理器，没有充分利用 GPU 本身的特性。MapCG[11]提供了基于 GPU 的哈希表，省去了 Reducer 阶段开始时将中间结果进行排序的阶段，可以将用户提供的 MapReduce 程序高效地移植到 CPU 或 GPU 上。Chen 等[26]和 Ji 等[27]引入了 GPU 上共享内存的使用来优化基于 GPU 的 MapReduce 编程框架，但以上的工作都是基于单个 GPGPU 的结点，所处理的数据规模受限于 GPU 的全局内存。

5.2 异构多核集群中的 MapReduce 编程框架

GPMR[28]是一个独立的框架，它修改了 MapReduce 框架，把大量的 Map 和 Reduce 操作项合并在一起，传送给 GPU，以提高 GPU 的利用率；通过增加 Map 阶段的数据积累预操作和 Reduce 阶段的本地部分归约，处理硬盘上的数据集和 GPU 之间的数据传送；并通过集成

MapReduce 中的管道机制让数据传送和计算的时间部分交叠，有效减少程序运行的整体时间。但其实现目前只适合于计算密集型程序，对于通讯较多的程序或大规模的集群，该框架尚不能很好地解决结点间的通讯开销较大的问题。MRCL[17]将 CPU 用作管理和任务分配，而 GPU 用来进行主要的计算，基于 Hadoop 和 OpenCL 实现了一个与平台无关的框架，可以无缝地集成到现有的 MapReduce 框架中。但该框架目前仅适用于数据依赖较为简单，没有复杂的分支程序使用在一些基本的数值计算、文本和图形处理上，还不足以用到实际应用。

在 CPU 和 GPGPU 结点混杂的集群中，HAPI[18]基于 Hadoop 和 Aparapi[29]，在 Map 阶段首先对输入的<key, value>对进行预处理，再把由一组<key, value>对产生的数据传送到 GPU 上进行计算，然后把计算结果从 GPU 上传给 Reducer。但 Map 阶段进行预处理及数据传输的过程需要编程者人工地维护一个数据结构，存储将要传送到 GPU 上的这一组数据，增加了编程的复杂度。HadoopCL[16]基于 Hadoop 和 Aparapi，支持用户直接写 Java 程序，由框架自动根据 Java 字节码生成 OpenCL 程序，并通过异步传输和专门的 I/O 进程优化了数据通讯，从而提高了整个模型的可编程性和效率。但要有专门的进程维护异步传输和 I/O 通讯。Glasswing[20]是另一个用 OpenCL 来实现利用多核 CPU 和加速器的 MapReduce 编程框架，更进一步地，它将不同任务的数据传输阶段和计算阶段交迭，实现了更细粒度的并行，以此提高性能。Axel[30]通过不同类型的处理单元和数据通信方式利用了时空的局部性，在混合了 FPGA、GPU 和 CPU 的 MapReduce 集群中成功地运行了 N-body 程序。

5.3　异构数据中心中的负载均衡

LATE[31]是第一个指出 MapReduce 编程框架在异构环境中的问题的工作。他们指出，在异构环境下，MapReduce 编程框架中识别，管理掉队任务的机制不再适用，并提出了在异构环境中更为适用的投机执行机制，优先完成新任务，优先挑选计算能力强的结点，并禁止过多投机任务的执行，减小作业的最终完成时间。Shirahata 等[32]针对 Hadoop 平台，提出了一种基于 GPU 异构计算集群的混合 Map 任务调度策略。该研究对 Map 任务整个运行过程进行监测和性能测试，利用性能测试的信息来决定调度策略，使得 MapReduce 作业的执行时间最短。Ahmad 等[33]指出异构集群中，现有的 MapReduce 框架的负载均衡策略会导致 Map 阶段暴发大量且集中的网络通信，结点间的异构性会放大 Reduce 阶段的负载不均衡问题，由此，他们提出了 Tarazu，以实现感知通信的 Map 计算负载均衡及任务调度，避免集中发生网络通信，并可预测 Reduce 阶段的负载均衡。Gandhi 等[34]提出 Tarazu 过早地估计了 MapReduce 中的目标负载分布，并对 Reduce 任务进行负载重分布，相对于 Tarazu 实现了 23% 的性能提升。MATE-CG[35]处理结点内的异构情况，即集群中每个结点是相同的，但是结点内包含 CPU 和 GPU。其运行时根据任务的特点对数据进行多层划分，动态分配给集群中的 CPU 或 GPU，并考虑数据的局部性等因素，实现负载平衡。

6　结束语

大数据时代，为有效支持多种应用，提升性能节约能耗，异构集群也正越来越多地被大规模集群所采纳。本文回顾了异构大数据编程的一系列相关工作，介绍了 Hadoop+ 框架的总体流程和资源管理。在本文所述的异构集群中，5 种常见机器演习应用使用 Hadoop+ 框架能相对于其各自的 Hadoop 实现取得 1.4×～16.1×的加速比。

参 考 文 献

[1] Hendrickson S. Getting Started with Hadoop with Amazon's Elastic MapReduce [M]. EMR, 2010.

[2] Buxton B, Hayward V, Pearson I, et al. Big data: the next Google, interview by Duncan Graham-Rowe [J]. Nature, 2008, 455(7209): 8-9.

[3] Vagata P, Wilfong K. Scaling the Facebook data warehouse to 300 PB. [EB/OL]. [2014-04-10]. https://code.facebook.com/posts/229861827208629/scaling-the-facebook-data-warehouse-to-300-pb/.

[4] Wikibon. Wikibon big data vendor revenue and market forecast, 2020 [EB/OL]. [2015-04-15]. http://www.kdnuggets.com/2015/04/wikibon-big-data-market-forecast-2020.html.

[5] IDC. The digital universe of opportunities [EB/OL]. [2014-04-01]. https://www.emc.com/collateral/analyst-reports/idc-digital-universe-2014.

[6] Che DR, Safran M, Peng ZY. From big data to big data mining: challenges, issues, and opportunities [M] // Database Systems for Advanced Applications. Springer Berlin Heidelberg, 2013: 1-15.

[7] Kaisler S, Armour F, Espinosa JA, et al. Big data: issues and challenges moving forward [C] // Proceedings of the 46th Hawaii International Conference on System Sciences (HICSS'13), 2013: 995-1004.

[8] Sagiroglu S, Sinanc D. Big data: a review [C] // Proceedings of the International Conference on Collaboration Technologies and Systems, 2013: 42-47.

[9] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters [C] // Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation, 2004: 107-113.

[10] He BS, Fang WB, Luo Q, et al. Mars: a MapReduce framework on graphics processors [C] // Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, 2008: 260-269.

[11] Hong CT, Chen DH, Chen WG, et al. MapCG: writing parallel program portable between CPU and GPU [C] // Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques, 2010: 217-226.

[12] Ranger C, Raghuraman R, Penmetsa A, et al. Evaluating MapReduce for multi-core and multiprocessor systems [C] // Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture (HPCA'07), 2007: 13-24.

[13] Hindman B, Konwinski A, Zaharia M, et al. Mesos: a platform for fine-grained resource sharing in the data center [C] // Proceedings of the 8th USENIX conference on Networked Systems Design and Implementation (NSDI'11), 2011: 295-308.

[14] Chowdhury M, Zaharia M, Ma J, et al. Managing data transfers in computer clusters with orchestra [C] // ACM SIGCOMM Computer Communication Review, 2011: 98-109.

[15] Isard M, Budiu M, Yu Y, et al. Dryad: distributed data-parallel programs from sequential building blocks [C] // Proceedings of the 2th ACM SIGOPS/European Conference on Computer Systems, 2007: 59-72.

[16] Grossman M, Breternitz M, Sarkar V. HadoopCL: MapReduce on distributed heterogeneous platforms through seamless integration of Hadoop and OpenCL [C] // IEEE 27th International Parallel and Distributed Processing Symposium Workshop, 2013: 1918-1927.

[17] Xin M, Li H. An implementation of GPU accelerated MapReduce: using Hadoop with OpenCL for data- and compute-intensive jobs [C] // Proceedings of the 2012 International Joint Conference on Service Sciences (IJCSS'12), 2012: 6-11.

[18] Lin Y, Okur S, Radio C. Hadoop+Aparapi: Making Heterogeneous Mapreduce Programming Easier [M]. Hgpu Org, 2012.

[19] Grossman M, Breternitz M, Sarkar V. HadoopCL2: motivating the design of a distributed, heterogeneous programming system with machine-learning spplications [J]. IEEE Transactions on Parallel & Distributed Systems, 2016, 27(3): 762-775.

[20] El-Helw I, Hofman R, Bal HE. Scaling MapReduce vertically and horizontally [C] // Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2014: 525-535.

[21] He WT, Cui HM, Lu BB, et al. Hadoop+: modeling and evaluating the heterogeneity for MapReduce applications in heterogeneous clusters [C] // Proceedings of the 29th ACM on International Conference on Supercomputing, 2015: 143-153.

[22] Cover T, Hart P. Nearest neighbor pattern classification [J]. IEEE Transactions on Information Theory, 2006, 13(1): 21-27.

[23] Apache. Apache mahout [EB/OL]. [2014-01-01]. http://mahout.apache.org/.

[24] Liu ZQ, Li HY, Miao GS. MapReduce-based back-propagation neural network over large scale mobile data [C] // The sixth International Conference on Natural Computation, 2010: 1726-1730.

[25] Catanzaro BC, Sundaram N, Kurt K. A map reduce framework for programming graphics processors [C] // Workshop on Software Tools for Multicore Systems, 2008.

[26] Chen LC, Agrawal G. Optimizing MapReduce for GPUs with effective shared memory usage [C] // Proceedings of the 21st International Symposium on High-Performance Parallel and Distributed Computing, 2012: 199-210.

[27] Ji F, Ma XS. Using shared memory to accelerate MapReduce on graphics processing units [C] // Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium, 2011: 805-816.

[28] Stuart JA, Owens JD. Multi-GPU MapReduce on GPU clusters [C] // Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium, 2011: 1068-1079.

[29] Frost G. Aparapi in amd developer websize [EB/OL]. [2011-09-14]. http://developer.amd.com/tools/heterogeneous-computing/aparapi/.

[30] Tsoi KH, Luk W. Axel: a heterogeneous cluster with FPGAs and GPUs [C] // Proceedings of the 18th annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays, 2010: 115-124.

[31] Zaharia M, Konwinski A, Joseph AD, et al. Improving MapReduce performance in heterogeneous environments [C] // The Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, 2008: 29-42.

[32] Schirahata K, Sato H, Matsuoka S. Hybrid map task scheduling for GPU-based heterogeneous clusters [C] // Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science, 2010: 733-740.

[33] Ahmad F, Chakradhar ST, Raghunathan A, et al. Tarazu: optimizing MapReduce on heterogeneous clusters [C] // Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems, 2012: 61-74.

[34] Gandhi R, Xie D, Hu YC. PIKACHU: how to rebalance load in optimizing MapReduce on heterogeneous clusters [C] // Proceedings of the 2013 USENIX Conference on Annual Technical Conference, 2013: 61-66.

[35] Jiang W, Agrawal G. MATE-CG: A MapReduce-like framework for accelerating data-intensive computations on heterogeneous clusters [C] // Proceedings of the 2012 IEEE International Parallel & Distributed Processing Symposium, 2012: 644-655.