DOI:10.7544/issn1000-1239.2016.20148351
53(6):1263-1270,2016

基于 YARN 集群的计算加速部件扩展支持

李 钦 朱延超 刘 轶 钱德沛
(北京航空航天大学计算机学院 北京 100191)
(网络技术北京市重点实验室(北京航空航天大学) 北京 100191)
(liqin0728@gmail.com)

Accelerator Support in YARN Cluster

Li Qin, Zhu Yanchao, Liu Yi, and Qian Depei
(*School of Computer Science and Engineering, Beihang University, Beijing* 100191)
(*Beijing Key Laboratory of Network Technology (Beihang University), Beijing* 100191)

Abstract Accelerators, such as GPU and Intel MIC, are widely used in scientific computing and image processing, and have strong potentials in big data processing and HPC based on cloud platform. YARN is a new generation of Hadoop distributed computing framework. Its adoption of computing resources is only limited to CPU, lacking of support for accelerators. This paper adds the support to nodes with accelerators to YARN to solve the problem. By analyzing the problem of supporting heterogeneous node, there are three identified difficulties which should be solved to introduce hybrid/heterogeneous to YARN. The first one is how to manage and schedule the added accelerator resources in the cluster; the second one is how to collect the status of accelerators to the master node for management; the third one is how to address the contention issue among multiple accelerator tasks concurrently running on the same node. In order to solve the above problems, the following design tasks have been carried out. Resource encapsulation which bundles neighbor nodes into one resource encapsulation is designed to solve the first problem. Management functions which collect the real-time accelerators status from working nodes are designed on the master node to solve the second problem. Accelerator task pipeline which splits accelerator tasks into three parts and executes them in parallel is designed on the nodes with accelerators to solve the third problem. Our scheme is verified with a cluster consisting of 4 nodes with GPU, and the workload testing the cluster includes LU, QR and Cholesky decomposition from the third party benchmark MAGMA, and the program performes feature extraction and clustering upon 50000 images. The results prove the effectiveness of the scheme presented.

Key words distributed system; yet another resource negotiator (YARN); accelerator; heterogeneous nodes; graphics processing unit (GPU); node binding; task scheduling

摘 要 以 GPU 和 Intel MIC 为代表的计算加速部件已在科学计算、图形图像处理等领域得到了广泛的应用,其在基于云平台的高性能计算及大数据处理等方向也具有广泛的应用前景. YARN 是新一代 Hadoop 分布式计算框架,其对计算资源的分配调度主要针对 CPU,缺少对计算加速部件的支持. 在 YARN 中添加计算加速部件需要解决多个难点,分别是计算加速部件资源如何调度以及异构节点间如

收稿日期:2014-12-10;**修回日期**:2015-09-22
基金项目:国家"八六三"高技术研究发展计划基金项目(2012AA01A302);北京市科学研究与研究生培养共建项目
This work was supported by the National High Technology Research and Development Program of China (863 Program) (2012AA01A302) and the Scientific Research and Graduate Student Education Project of Beijing.

何共享问题、多个任务同时调用计算加速部件而引起的资源争用问题和集群中对计算加速部件的状态监控与管理问题.为了解决这些问题,提出了动态节点捆绑策略、流水线式的计算加速部件任务调度等,实现了 YARN 对计算加速部件的支持,并通过实验验证了其有效性.

关键词 分布式系统;YARN;计算加速部件;混合异构节点;图形图像处理器;节点捆绑;任务调度

中图法分类号 TP316.4

2013 年,Apache 推出了其 Hadoop 分布式计算框架的第 2 代,命名为 yet another resource negotiator (YARN)[1],其采用新的计算框架,使用专门的 Resource Manager 对整个集群资源进行管理,单个节点的资源被划分成多个 Container(Ct),由 Node Manager 对其进行管理,而具体的集群应用则通过 Application Master 完成所有的调度和管理.这使得它以后将不仅仅支持 MapReduce 计算模型[2],更可以拓展至对内存计算(例如 Spark[3])、实时计算(例如 Storm[4])等的支持,并使得多种计算模型可以同时使用在同一个集群中.

近年来,伴随着多核/众核处理器的发展,产生了以 GPU 和 Intel MIC 为代表的异构处理器和加速部件,并在科学计算、图形图像处理等领域得到了广泛的应用.随着云计算技术的快速发展,基于云计算的高性能计算(HPC in cloud)快速兴起,同时,大数据应用需求对分布式系统的计算性能也提出了更高的要求,这些都为加速部件在分布式系统中的应用提供了很好的应用场景.然而,在第 1 代 Hadoop 以及新的 YARN 框架中,都只以单节点上的通用 CPU 作为计算资源在集群中进行分配和调度,并没有将计算加速部件作为另一种计算资源加以考虑.

近些年,如何在包括 Hadoop 等分布式平台下添加对于计算加速部件的支持成为分布式集群系统的一个研究方向.该主要研究方向有 3 个:1)将对于如 GPU 这样的计算加速部件的支持加入到 MapReduce 分布式计算模型中,通过对 MapReduce 模型的某个模块或者逻辑的创新型改进,实现包括 Hadoop 分布式系统在内的所有支持 MapReduce 模型的分布式集群系统对于计算加速部件的支持[5-10];2)在参考 Hadoop 调度系统的基础上,改进其调度策略或者设计一个新的分布式集群系统,使其原生态地支持计算加速部件资源,其中多以计算加速部件作为加速模块或者加速节点使用[11-16];3)不对 Hadoop 的架构和任务调度方法进行修改,而是在某一种或者某几种的特定应用上进行针对性的优化,以达到提高集群性能的目的[17-18].

本文提出并实现了基于 YARN 架构的计算加速部件(包括 GPU 和 Intel MIC)扩展支持.通过使用动态节点捆绑以及流水线式计算加速部件任务调度策略,对 YARN 架构进行了扩展,使得在全部或部分节点配置计算加速部件的分布式系统中,上层应用可以使用计算加速部件.

1 YARN 基本架构

YARN 是 Hadoop2.0 的资源管理系统,其基本的设计思想是将 MRv1 中的 JobTracker 拆分成 2 个独立的服务:一个全局的资源管理器(resource manager, RM)和每个应用程序独有的应用控制器(application master, AM).其中,RM 负责整个系统的资源管理和分配,而 AM 负责单个应用程序的管理[19].

YARN 整体上采用 Mater/Slave 结构,在整个资源框架中,RM 为 Master,Node Manager(NM)为 Slave,RM 负责对各个 NM 上的资源进行统一的管理和调度.当用户提交一个应用程序时,需要提供 1 个用于跟踪和管理这个程序的 AM,它负责向 RM 申请资源,并要求 NM 启动可以占用一定资源的任务.由于不同的 AM 被分布在不同的节点上,因此它们之间不会相互造成影响.图 1 描述了 YARN 的基本组成架构.

1) RM

RM 是一个全局的资源管理器,负责整个系统的资源管理和分配.它由调度器(scheduler)和应用程序管理器(applications manager)这 2 个组件构成.调度器根据容量、队列等限制条件将系统中的资源分配给各个正在运行的应用程序.它不从事任何与具体应用程序有关的工作,而是根据各个应用程序的资源需求进行分配.应用程序管理器负责管理整个系统中的所有应用程序,包括应用程序的提交、与调度器协商已启动 AM、监控 AM 的运行状态并在失败时重新启动它等.

2) AM

用户提交的每个应用程序都包含一个 AM,它

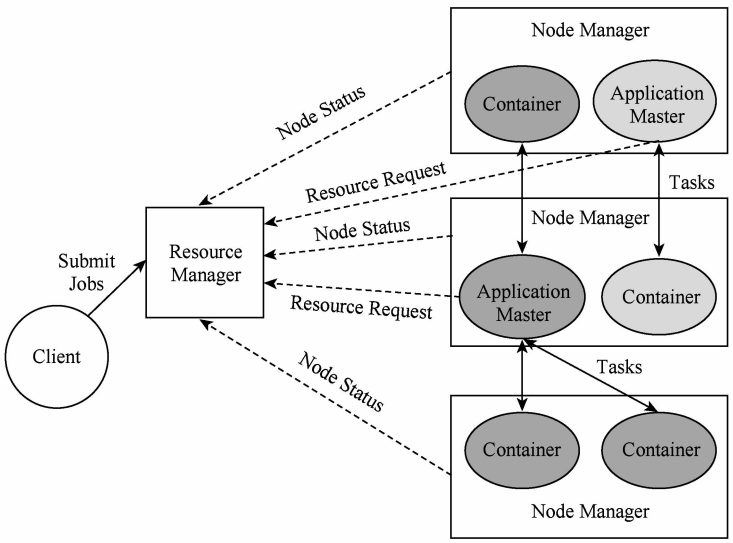

Fig. 1　YARN architecture.

图 1　YARN 架构示意图

将与 RM 调度器协商获取资源,再将得到的资源进一步分配给内部的任务,与 NM 通信以启动或停止任务,监控任务的运行状态,并在任务运行失败时重新申请资源再次启动任务.

3) NM

NM 是每个节点上的资源和任务管理器,它会定时向 RM 汇报本节点的使用情况和各个 Ct 的运行状态,以及接受并处理来自 AM 的 Ct 启动或停止等请求.

4) Ct

Ct 是 YARN 中的资源抽象,它封装了某个节点的资源,主要包括 CPU 和内存.当 AM 向 RM 申请资源时,RM 为 AM 返回的资源就是使用 Ct 表示.YARN 会为每一个任务分配一个 Ct,且该任务只能使用该 Ct 中的资源.

2　计算加速部件的支持扩展

2.1　设计思路

在现有的 YARN 框架中添加例如 GPU 或者 Intel MIC 这样的计算加速部件,并使之能够满足上层用户对资源的需求,为此需要解决 3 个问题:

1) 集群中对计算加速部件的状态监控与管理问题.现有的 YARN 中不支持除 CPU 之外的计算资源,因此需要新添加对计算加速部件的管理.本文采用由 NM 负责获取当前节点的计算加速部件的硬件信息,并在 RM 与 NM 之间增加实时计算加速部件的状态反馈,使 RM 获知计算加速部件的资源状态,整个集群的计算加速部件状态由 RM 进行监控与管理.

2) 计算加速部件资源如何调度以及异构节点间如何共享问题.现有 YARN 框架将每个节点 CPU 计算资源分割成相同粒度的容器 Ct,然后根据一定的调度策略对任务进行调度,其中各个 Ct 中的计算资源都是相互独立的,不存在 Ct 到 Ct 之间的资源共享机制.但是与传统的 CPU 资源不同,计算加速部件并不一定存在于每个节点上,并且在实际的应用场景中也不能要求每个节点都拥有计算加速部件.因此,计算加速部件的资源调度重点在于如何解决将集群中的计算加速部件资源让每个节点都能够获取的问题.本文将集群中无计算加速部件的节点(NNode)与有计算加速部件的节点(ANode)建立动态捆绑关系,NNode 中的需要计算加速部件执行的任务将会提交给对应的 ANode 上,并异步等待计算结果的返回.详细的节点捆绑策略将在 2.3 节中进行讲述.

3) 多个任务同时调用计算加速部件而引起的资源争用问题.单机环境下,多个进程如果同时使用例如 GPU 这样的计算加速部件时,会造成计算、通信等资源的争用,严重影响性能.在集群环境下同样有这样的问题.本文采用在 ANode 上建立一个特殊的 Ct,称为 AcCt (accelerator container),用来管理和调度获取到的计算加速部件任务.详细的 AcCt 任务调度策略将在 2.4 节讲述.

2.2　系统结构

在分布式集群系统 YARN 上支持带有计算加速部件节点的设计方案的系统架构图如图 2 所示:

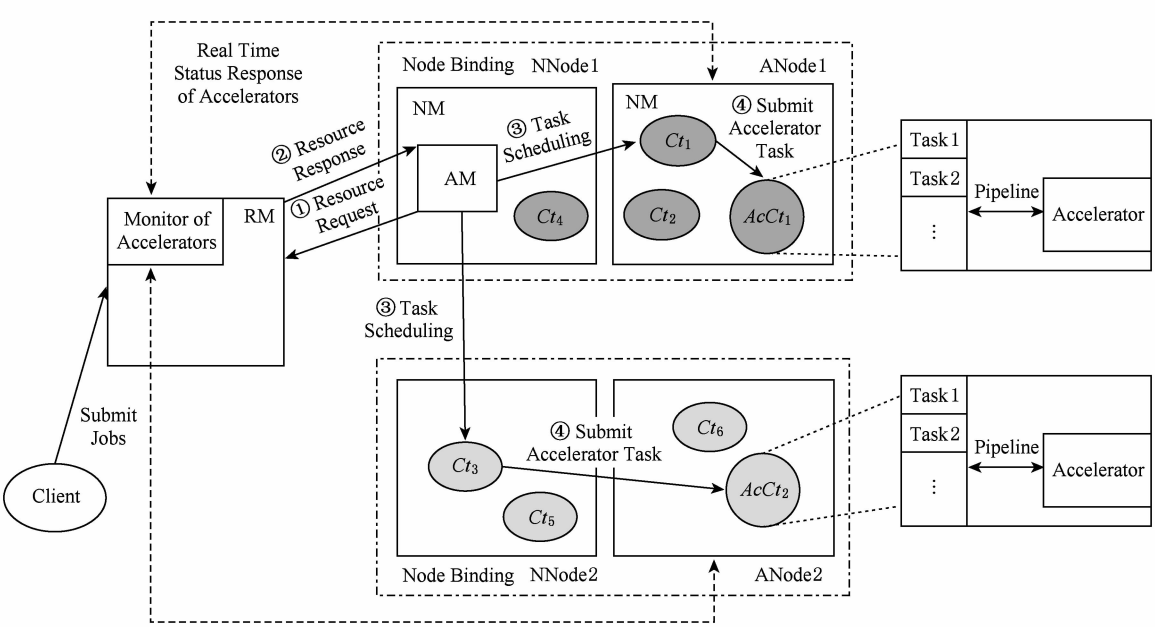

Fig. 2 Extended architecture of YARN.

图 2 扩展后 YARN 架构示意图

根据图 2 的架构图,应用调度流程总结如下:

1) 集群启动时,运行在各个节点的 NM 会向 RM 提交当前节点资源信息,这其中包括计算加速部件的资源信息,RM 会根据计算加速部件的资源信息按照节点捆绑策略建立 NNode 映射到 ANode 的捆绑列表,并返回给各个 NM,这时,ANode 就会将本地的 AcCt 启动起来,并进行监听.

2) 用户向 RM 提交作业后,会产生一个新的 AM,这时,AM 会向 RM 申请作业所需的资源(包括计算加速部件任务的资源). RM 会按照之前建立好的捆绑映射列表返回给 AM 所需的 Ct 信息,但具体的捆绑映射列表对 AM 透明,不需 AM 获知.

3) 作为 Master 的 AM 就会向分配给它的 Ct 发送任务并调度,若 Ct 遇到的是需求计算加速部件计算资源的任务,则会异步地向对应的 ANode 发送任务信息,这时,接收到任务的 AcCt 会按照流水线调度策略对任务进行处理,并在处理完成后将结果返回.

2.3 节点捆绑策略

节点捆绑策略的基本原则是节点间距离最近原则,即同一机架内视为距离最近可捆绑,机架间节点相互隔离、不可捆绑. 策略的核心是在实现动态捆绑的前提下尽量使用"平衡且代价小"的调度算法,具体来讲就是在动态变换时尽量保持原有 NNode 与 ANode 之间的对应关系.

RM 会在各个 NM 启动时接收到当前节点的计算加速部件信息,同时 NM 与 RM 之间的心跳线也会时刻传递当前节点的辅助存储器信息. RM 会根据这些接收到的资源信息进行 NNode 与 ANode 之间的捆绑. RM 会为每个 ANode 建立 1 个以 IP 地址为键的栈结构,用来存储其捆绑的 NNode. 同时,RM 会为每个机架建立 1 个对应的排序队列,用来放置之前建立的各个栈,排序队列以栈的存储节点个数为序.

1) 节点添加机制

① 接收 NNode 并添加时,将当前机架的排序队列中最小的栈取出,将当前 NNode 放入,再将栈插入回队列;

② 接收 ANode 并添加时,首先为其建立栈结构,然后判断所在机架的排序队列是否平衡(当前栈大小是否与队列中最小的栈大小相同),若已经平衡,则将当前栈加入到排序队列,就完成了策略;若不平衡,则从队列中取出最大栈,从栈头取出一个 NNode 放入到当前栈中,再将取出的栈插入回队列中. 这时,再次判断是否平衡,如不平衡则再次进行以上操作直到队列平衡,如图 3 所示.

2) 死亡退出机制

① NNode 节点超时断开时,将其从对应的栈中删除,再对排序队列进行刷新;

② ANode 节点断开时,将其从排序队列取出,在栈中的各个 NNode 都取出来,最后按照 NNode 的节点添加机制进行再次添加.

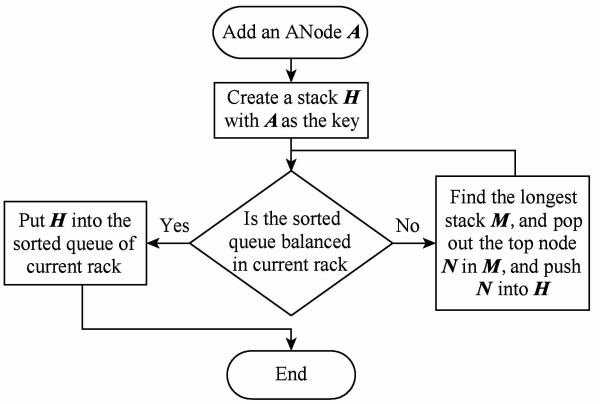

Fig. 3　Strategy of adding ANode.

图 3　添加 ANode 节点策略

2.4　计算加速部件任务调度策略

在 ANode 节点上,当 NM 开始运行时会同时启动 AcCt 来完成对所在节点计算加速部件资源的管理和任务调度.它使用远程读、处理、远程写的 3 级流水方式,将计算加速部件的计算资源高效地利用.具体的实现方式如图 4 所示:

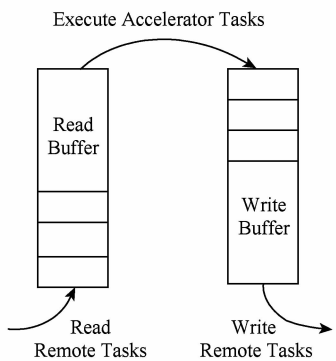

Fig. 4　Accelerator task scheduling in single node.

图 4　单节点内计算加速部件任务流水线调度策略

任务调度由 3 个独立线程和 2 个阻塞队列构成.远程读线程不断监听其他节点的连接,在获取连接后将远程需要处理的数据信息写入到本地的处理缓冲区中;处理缓冲区的数据结构是阻塞队列,处理线程会不断地从中取出需要处理的数据并执行;当处理线程完成处理操作后,会将处理好的结果写入到写缓冲区中;写缓冲区也由 1 个阻塞队列构成,它为远程写线程提供数据;远程写线程将从写缓冲区中的数据远程写入到与之建立连接的节点上.这样就完成了 3 级流水线的调度策略.

3　实验环境及结果

3.1　实验环境

本文的实验部署在以 4 台服务器建立起来的集群中,以 GPU 作为计算加速部件进行测试.详细的软硬件参数如表 1 所示:

Table 1　Test Environment

表 1　测试环境

Item	Parameters
Machine	Inspur NP3560
CPU	Intel® Xeon® X5650 2.67 GHz, 2 six-core Processor
Memory/GB	16
GPU	NVidia Tesla C2075(448 cores)
Bandwidth/Mbps	1 000
YARN	Hadoop-2.2.0
CUDA	Release 4.0, V0.2.1221
Java	Java™ SE Runtime Environment (build 1.6.0_31-b04)

3.2　实验测试用例

MAGMA(matrix algebra on GPU and multicore architectures)是由田纳西大学 ICL 实验室创建并维护的一组 GPU 加速的线性代数(LA)集,它为基于 GPU 的异构体系结构提供了基于 GPU 的 LA 测试包,以及例如 LAPACK 和 BLAS 这样的 LA 依赖包来进行 CPU 测试[20].

本文选取了 MAGMA 矩阵计算中最为常用的 LU 分解、Cholesky 分解和 QR 分解算法作为集群中单次任务的基准程序,矩阵大小为 10 000×10 000.表 2 给出了在本文实验环境下单个节点使用 GPU 计算和 CPU 计算的时间和它们之间的加速比.

Table 2　CPU/GPU Execution Time and Speedup of MAGMA Benchmark in Single Node

表 2　单机环境下 MAGMA 基准程序的 CPU/GPU 执行时间和加速比

Decomposition Algorithms	Running Time/s		Speedup
	CPU	GPU	
LU Decomposition	55.0	6.2	8.9
QR Decomposition	384.1	23.2	16.6
Cholesky Decomposition	60.5	7.6	8.0

每个测试应用中有 100 个相互独立的任务组成,其中单个任务分别是 LU 分解、Cholesky 分解和 QR 分解的基准程序.通过改变单次任务的强度(1,2,4,8 倍强度)和运行任务时进行的数据传输大小(1 MB,5 MB,10 MB,20 MB,50 MB)来探索本文的适用范围和性能.选择最高 8 倍的单次任务强度系数,是因为其所对应的时间已接近集群的单任务

时间上限,高时间消费的单次任务在集群容错机制下,性能会下降得更加严重.选择最大 50 MB 的数据传输大小是因为在实际的应用中单次的任务数据量一般不会超过 50 MB.而对于超过的任务,应该首先考虑是否应该将任务分拆成多个进行.

本文还选取了一个对 5 万张图片进行多种特征提取和聚类索引的 MapReduce 操作的应用实例.该应用使用了 GPU 加速的图片颜色和纹理 LBP 特征提取的 Map 操作,并在 Reduce 操作使用 K-means 算法对图片特征进行聚类[21].

由于实验环境的限制,本文使用了 Cluster4-2(4 个节点中 2 个节点拥有 GPU),Cluster4-3(4 个节点中 3 个节点拥有 GPU)和 Cluster4-4(4 个节点中 4 个节点拥有 GPU)这 3 种集群环境进行测试,并与没有使用本文方法(Cluster4-0,即 4 个节点都是用 CPU 进行计算)的环境进行对比.

3.3 实验数据及分析

图 5 给出了由 MAGMA 矩阵计算中 LU 分解、Cholesky 分解和 QR 分解算法 Benchmark 集群应用在使用不同的单次任务强度(1,2,4,8 倍强度)和任务数据传输大小(1 MB,5 MB,10 MB,20 MB,50 MB)条件下,在 Cluster4-4,Cluster4-3,Cluster4-2 环境运行的时间与未使用本文方案的 Cluster4-0 环境运行时间的加速比折线图.

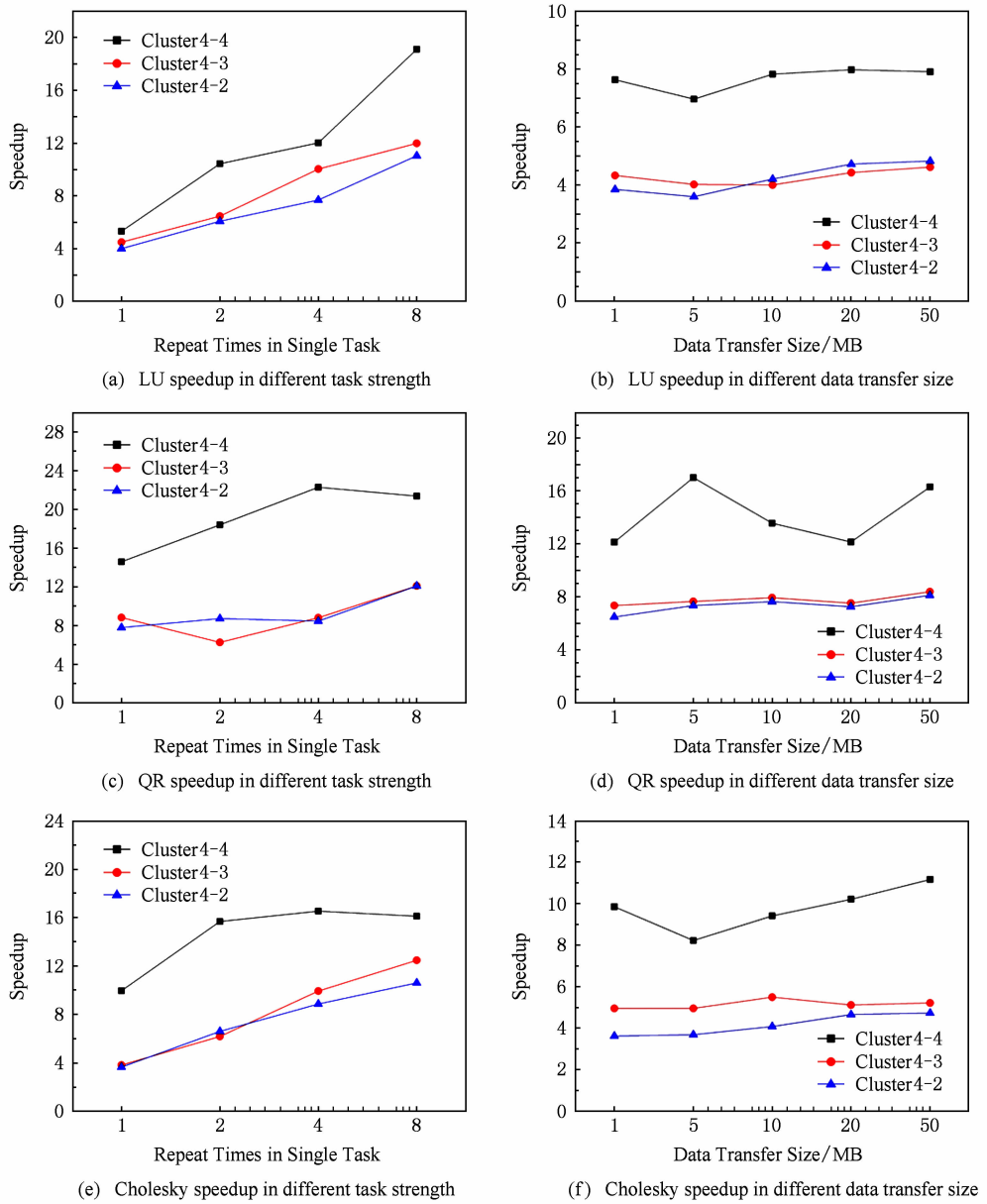

Fig. 5 Test results of benchmark programs of LU, QR, Cholesky algorithms in different clusters.

图 5 LU,QR,Cholesky 算法基准程序在不同集群环境下的实验结果

从不同的集群环境上看,Cluster4-2 和 Cluster4-4 都出现了与 GPU 个数相匹配的加速比,并且 LU 和 Cholesky 算法在单次任务强度增大到 8 倍时出现了高于单机环境下的加速比情况.这主要是因为随着任务强度的增大,CPU 算法耗费的时间过长,会更多地触发集群容错中的机制保证集群的正常运转,例如推测式执行等.但是本文方案中的任务异步执行和 GPU 加速的方法防止了这样的情况发生.

Cluster4-3 也出现了一定的加速比,但是因为其在进行动态节点捆绑策略时,无法均匀地将有 GPU 节点和没有 GPU 节点绑定,造成了 2 个节点各独立使用 1 个 GPU 而另外 2 个节点共用 1 个 GPU 的不平衡.同时,Cluster4-2 在节点绑定后,出现 2 个节点共用 1 个 GPU、另 2 个节点也共用 1 个 GPU 的拓扑结构,与 Cluster4-3 不平衡的拓扑同构,因而其加速比曲线与 Cluster4-2 基本重合.这种 GPU 分配不均匀的问题,在集群节点个数增多的情况下会有一定程度的改善.

从单次任务强度变化上看,LU,Cholesky,QR 这 3 个算法在不同集群环境下,加速比除了在强度为 1 的情况外,都会随着单次任务强度的升高而变大,单次任务强度与加速比成正相关.而强度为 1 时出现的个别数据不符合正相关曲线情况,是因为整个集群启动时间在运行时间较短的应用中影响更大.

从数据传输大小变化上看,LU,Cholesky,QR 这 3 个算法在不同集群环境下,其加速比都没有表现出与数据传输大小之间的正反相关性,因此可以认为单次任务的数据传输大小在对整个应用的运行时间并没有明确的正反相关性.

图 6 给出了 5 万张图片进行特征提取和聚类的 MapReduce 操作的应用实例在不同集群环境下的运行时间柱状图.从实验数据可以看出,应用实例的运行时间基本能够随着集群节点中增加 GPU 计算资源而显著地减少运行时间.虽然在 Cluster4-3 环境下出现了与 Benchmark 应用程序相同的情况导致与 Cluster4-2 运行时间相同,但这种情况在集群节点个数增多时能够得到改善.

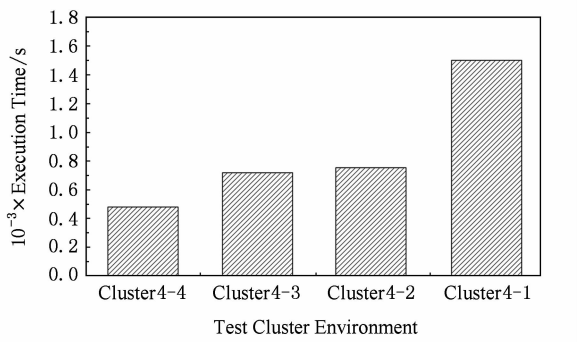

Fig. 6　Test result of the application for picture extraction and clustering.

图 6　图片特征提取聚类应用程序的实验结果

4　结　　论

本文方案使用了集群计算加速部件统一管理、动态节点捆绑和流水线式计算加速部件任务调度策略,对原有的 YARN 架构进行了改进,增加了对包括 GPU 和 Intel MIC 在内的计算加速部件资源的调度和管理,使得在全部或部分节点配置计算加速部件的分布式系统中,能够较好地对集群中的计算加速部件进行利用.本文通过在 4 台节点组成的带有 GPU 计算加速部件的集群环境中,使用 LU,QR,Cholesky 分解算法以及 5 万张图片特征提取及聚类索引的应用实例,对本文方案进行了实验测试,实验结果验证了本文方案的有效性.

5　未来工作展望

针对实验结果中 Cluster4-3 出现的性能问题,可以将现有方案中对节点的捆绑更加细化为基于 Container 的绑定,即在 RM 中维护的不再是节点间的捆绑关系,而是每个节点上的 Container 之间的捆绑关系.这样,就会更为细粒度地对集群资源进行分配和共享.但是也会遇到一些困难,例如 Container 并非真正的物理模块,而是由 YARN 对节点内资源进行隔离产生的逻辑模块,它的产生和销毁会更加灵活和频繁,这对整个管理模块的调度策略和性能是个挑战.

再者,可以对计算加速部件的性能建立权值表,并按照节点的权值信息优化动态捆绑策略.因为在本文当前的版本中,节点上的计算加速部件只是无与有之分,但是在实际环境中,不同型号和性能的计算加速部件可能同时存在于一个集群中.因此,如果建立一个性能的权值表,动态生成或者让用户自己修改和维护这个表格,并根据权值信息优化动态捆绑策略,这样就可以更好地利用不同的计算加速部件,从而实现效率的提升.

参　考　文　献

[1]　Vinod V, Arun M, Chris D. Apache Hadoop YARN: Yet another resource negotiator [C] //Proc of SoCC'13. New York: ACM, 2013: 5

[2] Jeffrey D, Sanjay G. MapReduce: Simplified data processing on large clusters [C] //Proc of OSDI'04. Berkeley, CA: USENIX Association, 2004: 135-150

[3] Apache Spark: Lightning-fast cluster computing [EB/OL]. [2014-07-31]. https://spark.apache.org

[4] Apache Storm: Distributed and fault-tolerant realtime computation [EB/OL]. 2014 [2015-07-31]. https://storm.apache.org

[5] Jeff S, John O. Multi-GPU MapReduce on GPU clusters [C] //Proc of IPDPS'2011. Piscataway, NJ: IEEE, 2011: 1068-1079

[6] Liu Mingliang, Jin Ye, Zhai Jidong. ACIC: Automatic cloud I/O configurator for HPC applications [C] //Proc of SC'13. Piscataway, NJ: IEEE, 2013: 1-12

[7] Li Xiaobing, Wang Yandong, Jiao Yizheng, et al. CooMR: Cross-task coordination for efficient data management in MapReduce programs [C] //Proc of SC'13. Piscataway, NJ: IEEE, 2013: 1-11

[8] Max G, Mauricio B, Vivek S. HadoopCL: MapReduce on distributed heterogeneous platforms through seamless integration of Hadoop and OpenCL [C] //Proc of IPDPSW'13. Piscataway, NJ: IEEE, 2013: 1918-1927

[9] Gunho L, Byung-Gon C, Randy H. Heterogeneity-aware resource allocation and scheduling in the cloud [C/OL] //Proc of HotCloud'11. Berkeley, CA: USENIX Association, 2011 [2014-07-31]. http://static.usenix.org/event/hotcloud11/tech

[10] Faraz A, Srimat C, Anand R, et al. Tarazu: Optimizing MapReduce on heterogeneous clusters [C] //Proc of ASPLOS'12. New York: ACM, 2011: 61-74

[11] Fengguang S, Jack D. A scalable framework for heterogeneous GPU-based clusters [C] //Proc of SPAA'12. New York: ACM, 2012: 91-100

[12] Kuen T, Wayne L. Axel: A heterogeneous cluster with FPGAs and GPUs [C] //Proc of FPGA'10. New York: ACM, 2010: 115-124

[13] George B, Aurelien B, Anthony D, et al. DAGuE: A generic distributed DAG engine for high performance computing [J]. Parallel Computing, 2012, 38(1): 37-51

[14] Cédric A, Jérôme C, Samuel T, et al. Data-aware task scheduling on multi-accelerator based platforms [C] //Proc of the 16th IEEE Int Conf on Parallel and Distributed Systems. Piscataway, NJ: IEEE, 2010: 291-298

[15] Vignesh T R, Michela B, Gagan A, et al. ValuePack: Value-based scheduling framework for CPU-GPU clusters [C] //Proc of SC'12. Piscataway, NJ: IEEE, 2012: 1-12

[16] Jungwon K, Sangmin S, Jun L, et al. SnuCL: An OpenCL framework for heterogeneous CPU/GPU clusters [C] //Proc of ICS'12. New York: ACM, 2012: 341-352

[17] Wittek P, Darányi S. Accelerating text mining workloads in a MapReduce-based distributed GPU environment [J]. Journal of Parallel and Distributed Computing, 2013, 73(2): 198-206

[18] Zhai Yanlong, Luo Zhuang, Yang Kai, et al. High performance massive data computing framework based on Hadoop cluster [J]. Computer Science, 2013, 40(3): 100-103 (in Chinese)
(翟岩龙, 罗壮, 杨凯, 等. 基于 Hadoop 的高性能海量数据处理平台研究[J]. 计算机科学, 2013, 40(3): 100-103)

[19] Apache Hadoop. Apache Hadoop: Apache Hadoop nextgen MapReduce [EB/OL]. [2014-07-31]. http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

[20] Innovative Computing Laboratory (ICL) of University of Tennessee. MAGMA: Matrix algebra on GPU and multicore architectures [EB/OL]. [2014-07-31]. http://icl.cs.utk.edu/magma/news/index.html

[21] Wang Xiangrong, Gao Fei, Li Qin, et al. GPU-based acceleration of local binary pattern algorithm for Internet applications [J]. Computer Engineering and Science, 2013, 35(11): 153-159 (in Chinese)
(王香荣, 高飞, 李钦, 等. 面向互联网应用的图像 LBP 算法 GPU 并行加速[J]. 计算机工程与科学, 2013, 35(11): 153-159)

Li Qin, born in 1989. Master. His main research interests include distributed system and high performance computing.

Zhu Yanchao, born in 1989. PhD candidate. His research interests include high performance computer architecture and distributed systems (zyc0627cool@163.com).

Liu Yi, born in 1968. Professor of Beihang University since 2006, and associate professor of Xi'an Jiaotong University from 2001 to 2006. Received his PhD degree in computer science from Xi'an Jiaotong University. His main research interests include high performance computing and computer architecture (yi.liu@jsi.buaa.edu.cn).

Qian Depei, born in 1952. Master in computer science, professor of Beihang University, director of the Sino-German Joint Software Institute. Fellow member of China Computer Federation. His current research interests include high performance computer architecture and implementation technologies, multicore/many-core programming, distributed systems, overlay networks, and network measurement (depeiq@buaa.edu.cn).