数据流计算环境下的集群资源管理技术

汤小春，符莹，丁朝，毛安琪，李战怀
西北工业大学计算机学院，陕西 西安 710129

摘要
以集群为基础的高性能计算的发展经历了3个阶段的演化，即计算子系统与存储子系统的分离、计算子系统与存储子系统的融合以及以数据并行为基础的dataflow编程模型。随着Spark、Flink等数据流编程模型在大数据计算领域的广泛使用，计算作业类型千变万化，如何保证各种数据流计算作业对集群资源的共享使用是集群资源管理的核心，也是降低基础设施成本的主要手段。分析集群资源管理的历史变化，从数据流编程模型的角度出发，对HoD、集中式、双层调度、分布式以及混合式管理展开了深入的探索，介绍了其各自的优缺点以及应用现状，为数据流计算环境下的集群资源管理和调度的使用或者研发提供一定的参考和借鉴。

关键词
数据流模型；集群资源；调度框架；大数据

中图分类号：TP31 文献标识码：A doi: 10.11959/j.issn.2096-0271.2020026

State-of-art research of cluster resource management in dataflow computing model

TANG Xiaochun, FU Ying, DING Zhao, MAO Anqi, LI Zhanhuai
School of Computer Science, Northwestern Polytechnical University, Xi'an 710129, China

Abstract
The development of cluster-based high-performance computing has undergone three stages of evolution. With the widespread use of dataflow programming models such as Spark and Flink in the field of big data computing, how to ensure the fair share with the cluster resources by various dataflow computing applications is extremely important. It is also a main means to reduce the cost of infrastructures. As the drawbacks of traditional cluster resource management have becoming increasingly apparent in dataflow computing model, many alternative cluster resource management, including HoD, centralized scheduling, two-level scheduling, distributed scheduling, and hybrid scheduling management, have been proposed in recent years. Their respective advantages and disadvantages were introduced, and a certain reference for the uses or researches in development of cluster resource management and scheduling in a dataflow computing environment was provided.

Key words
dataflow model, cluster resource, schedule framework, big data

1 数据流计算环境下集群资源管理的发展历史

高性能计算机的动力源自各类科学计算对计算性能的迫切需求和解决生产问题的能力。20世纪80年代，计算机性能的纵向扩展，即计算机处理器性能的提高，受到了成本的挑战，每提高性能一次，其代价呈指数增加。故计算机性能的横向扩展得到了人们的重视，即通过网络将廉价、功能强大的个人计算机（personal computer，PC）或者工作站组织在一起，形成通用计算集群。在通信顺畅的情况下，一个集群计算系统能承担一个大型计算机系统所负担的任务。

但是一个简单的通用集群并不能很好地分工协作，若要一个通用集群计算系统真正担负起一个大型机的作业，那么就必须使用一套资源管理工具来管理和分配这些汇聚在一起的资源，使每个任务均能透明、合理和公平地使用它们。

20世纪90年代初，集群系统无论是在硬件还是在资源管理系软件方面都经历了长足的发展[1]。从最初的高性能集群资源管理到服务于网格计算的集群资源管理[2]，再到云计算环境的集群资源管理，不管是资源管理框架还是任务调度策略都取得了明显的进步，亦产生了大量的研究成果。这些成果也是目前数据流（dataflow）环境下集群资源管理研究的基础[3-7]。

传统的集群计算机系统将计算系统和存储系统分开，其资源管理系统和任务调度系统各自独立。计算系统只管理可用的计算资源、内存以及计算节点的负载信息，存储系统只管理数据的存储空间、可靠性以及I/O性能等指标。存储系统和计算系统之间通过高速网络进行连接，目的在于满足不同的计算节点对数据的快速访问需求。数据的存储采用3级层次结构，即计算节点用来存储临时数据，共享文件系统用来满足用户对数据的共享访问，后端归档存储系统用于备份历史数据。这样的3层体系结构在数据密集型计算作业上取得了巨大的成功。其最大的优点在于存储系统和计算系统的分离，使得二者可以被单独升级和更新。然而，优势往往会转化为劣势，分离的结构势必造成网络带宽的瓶颈。随着计算节点的扩大，计算系统和存储系统之间的网络带宽变得越来越小，无法满足以数据流为基础的大规模的数据处理和分析系统。

以数据流为基础的资源管理要求计算和数据之间具有亲和性，当亲和性较高时，集群系统的计算能力会显著提高。数据并行系统（例如Hadoop分布式文件系统（Hadoop distributed file system，HDFS））的出现使得计算和数据的亲和性变得至关重要。集群系统中的计算系统和存储系统进行了无缝融合，即集群中的节点既承担计算系统的任务，也承担存储系统的任务。数据并行系统使得任务调度方式变得非常灵活。例如，将数据移动到计算节点后，在计算节点运行任务；按照计算代价最小原则找到一个折中的节点，在该节点运行任务；或者将计算移动到数据节点，在数据节点运行任务。

现有的大数据分析中，Spark[8-9]、Flink[10]等采用数据流编程模型。数据流计算模型与控制流计算模型有明显区别，它通过数据驱动的方式执行，计算任务向数据移动。在数据流计算环境下，集群资源管理和调度必须考虑数据的本地化。随着数据形式的变化和数据规模的增大，集群的规模也逐渐由小到大、计算模型由批流独立到合二为一、计算资源由CPU到CPU-GPU等快速变化。正因为这些变化，集群资源框架的研究内容以及最后的部署实现都出现了大量的变化。目前，广泛使用的有

集中管理框架[11-12]和双层调度框架[13]，处于研究阶段的有分布式管理框架[14]、混合管理框架[15]，以及一些其他的优化策略。

本文对目前数据流计算环境下不同集群资源框架与任务调度的国内外现状进行了介绍与分析，并对未来数据流计算环境下集群管理面临的问题提出了一些看法。接下来，首先对数据流计算环境下的集群资源管理面临的主要问题进行探索，然后分析现有的集群资源管理框架，最后对数据流计算环境下集群资源管理框架进行展望。

2 数据流计算环境下集群资源管理的主要研究问题

数据流计算[16]是针对当前大规模密集型计算在多核处理器以及集群系统上的应用特点而设计的一种新的计算模型。将数据计算任务与通信任务分割，通过任务调度与分配，利用数据自身的天然并行的特点来提高应用程序的并行性，使各个计算资源之间负载均衡。对于传统控制流程序的调度，只有在数据计算任务满足控制条件时才可以被执行。而在数据流模型中，任务与任务之间通过数据来控制，任务由数据驱动执行。任务计算完成后输出数据，只要数据到达，后续任务就立即被调度并执行。

随着数据流编程模型的出现，集群资源管理和调度研究主要面临3个方面的矛盾。

- 数据注入与任务调度延迟的矛盾。随着数据规模的扩大，以数据驱动为主的计算并行度增加，集群计算资源的获得成为瓶颈。依据数据的注入速度以及数据并行度，尽可能地为每个并行数据分配到计算资源是资源管理的目标。
- 数据处理方式和需求日益多样、复杂，批流合一的处理模式中存在高吞吐量和低延迟的矛盾。计算任务数量的上升，一方面增加了调度时需要面对的性能瓶颈；另一方面加大了资源分配过程的开销和数据传输的代价，使整体负载管理变得困难。另外，流处理任务对延迟的敏感性要求调度开销最小。
- 资源分配本地化与任务调度延迟的矛盾。随着数据规模的扩大，集群数量不断增大，集群计算节点之间的网络带宽成为瓶颈。依据数据的存储位置特性，将任务尽可能地调度到数据资源所在节点是资源管理的目标。但是这导致一些任务在不满足本地化的情况下无法快速得到计算资源，使得本地化调度和调度延迟成为一对矛盾。

针对以上3个挑战，研究者从数据流计算环境入手，对照资源管理和调度方法，得到集群资源管理和调度框架的变化。本文从现有的研究成果中，总结出以下几个问题。

- 计算系统与数据存储系统之间网络设备的显著差异使得各个节点取得集群节点中相同数据分块或其副本的时间代价出现差异，因此任务调度的数据本地化是优化的关键。根据集群系统中的机架内外的逻辑关系，形成了树形网络拓扑结构，其存在5个级别的数据本地化约束，即同一个执行器、同集群节点、共享服务、机架内和机架之间。不同的数据本地化约束会导致不同的任务执行代价。
- 当存在大量的计算任务需要调度时，为满足任务调度中的需求，需要确保作业的资源需求。而现有的资源管理基本上采用公平分配、静态容量分配以及共享分配的方式，无法依据数据的注入速度动态分配资源。
- 高通量和低延迟问题。在诸多数据处理方式中，业务分析的数据范围横跨流式数据和历史数据，既需要低延迟的流式

数据分析，也需要对PB级的历史数据进行探索性的数据分析。流式数据作业处理是比较特殊的一类，因为用户对任务的有效时间有刚性的要求，所以资源管理和任务调度策略能够保证最低的资源要求。与批处理任务相比，流处理对延迟更敏感，调度器必须提供足够的计算资源，才能保证在任务的截止时间内完成计算。

3 数据流计算环境下的资源管理的研究现状

依照调度策略，现有的数据流计算环境下的资源管理系统主要的资源管理方式分为按需调度、集中式调度、双层调度、共享状态调度、全分布式调度、混合式调度以及CPU-GPU混合的异构集群资源调度。

3.1 按需调度

支持大数据处理的初期的集群系统，由于大数据自身的编程模型千变万化，使用传统的高性能集群是当时的首选，如HoD（Hadoop on demand）。在这种集群系统中，资源调度采用PBS方案或者Torque资源管理工具，但是涉及任务调度的部分则使用由Maui或Maob提供的作业调度工具。这种解决方案的优点在于集群资源管理和任务调度模型已经成熟，大数据程序开发者只关心应用程序的业务逻辑。然而，这些资源调度框架和策略并不符合以数据流编程模型为主的大数据应用需求，其关键的问题是无法在数据的本地化和资源公平分配之间协调。

3.2 集中式调度

集中式调度指的是使用主从结构的模型，主节点管理整个集群的资源和调度任务。也就是说，整个集群系统只有一个作业调度器，所有计算任务的资源请求和资源分配都通过高度集中的管理节点来实施。

3.2.1 典型系统

典型的集中式资源管理为YARN以及Borg。

（1）YARN

YARN资源管理框架如**图1**所示。资源管理器作为后台服务驻留在一个专用的计算节点上，为各种应用框架分配计算资源。依据应用程序的需求、调度优先级别以及可用计算资源，资源管理器动态地分配容器到某个计算节点，以便运行任务。节点管理器（PN）运行在各个集群节点上，其定期向资源管理器汇报资源状态，同时对分配后的资源进行限制，资源限制的单位是容器（也是一种逻辑的资源单元）。

作业通过一定的提交协议被提交到资源管理器，安全以及通过许可证的作业被发送到调度器上运行。一旦调度器有足够的资源，作业的状态就变为运行状态。这个过程包括为作业的主应用程序在某个计算节点上分配一个容器并启动。接收作业的记录会被持久化，以便失败后进行恢复。

主应用程序是一个作业的核心，管理动态的资源消耗以及作业的执行流程。为了得到一个容器，主应用程序向资源管理器请求资源，请求信息包括本地化偏好说明以及容器的属性。资源管理器将根据集群系统的可用资源，依据一定的调度策略尽量满足主应用程序的请求。主应用程序一旦收到资源，将把一个任务发送到容器中运行。任务运行过程中，容器与主应用程序直接通信，以便汇报任务的状态。

（2）Borg

Borg资源管理框架由管理器和代理组成，如**图2**所示。

数据中心的多个集群计算节点被组织成一个集群，资源的分配和管理采用集中式框架，其主节点是管理器，从节点是代理。管理器统一维护整个集群的资源状态，代理采用心跳的方式向管理器汇报任务的状态变化和自身节点的健康状况。主节点使用热机备份的模式，也就是说同时运行多个管理器副本，当处于活动状态的主节点出现宕机时，系统就立即从多个副本节点中选择新的主节点。为了体现计算任务调度的灵活性，调度器从资源管理中被剥离出来，作为单独的服务。早期的Borg是集中式的，最新的系统在集中式的基础上进行了改良，允许副本中的调度器参与调度，并共享集群资源状态信息，因而其调度策略与共享状态调度器的策略相似。

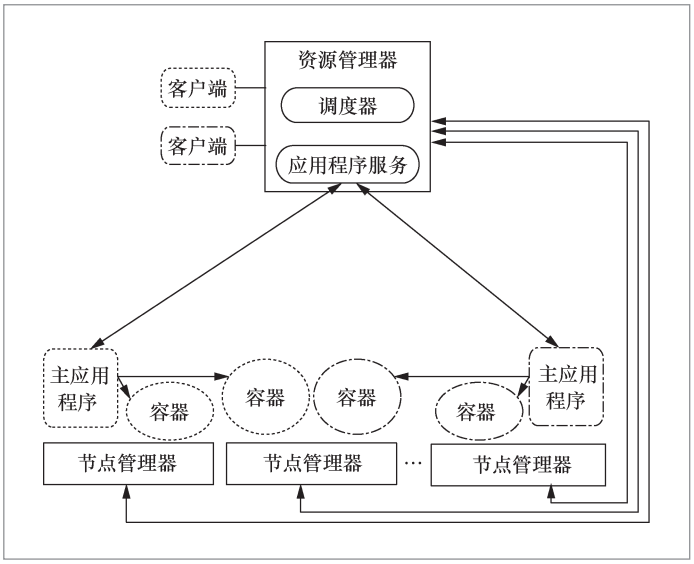

图 1 YARN 资源管理框架

3.2.2 优缺点

集中式资源管理的优点包括：任务调度和资源调度采用紧耦合模式，任务调度的效率较高；全局状态的一致性容易保证，调度算法实现简单。其缺点包括：仅适合批处理计算任务，并且任务的运行时间超过1 s；主节点是瓶颈，容易引起单点故障，热机备份是主要的高可用性手段；可扩展性较差，当集群规模扩大，存在大量的任务请求以及心跳信息时，主节点无法及时处理各种请求，造成任务的延迟以及状态信息陈旧问题。

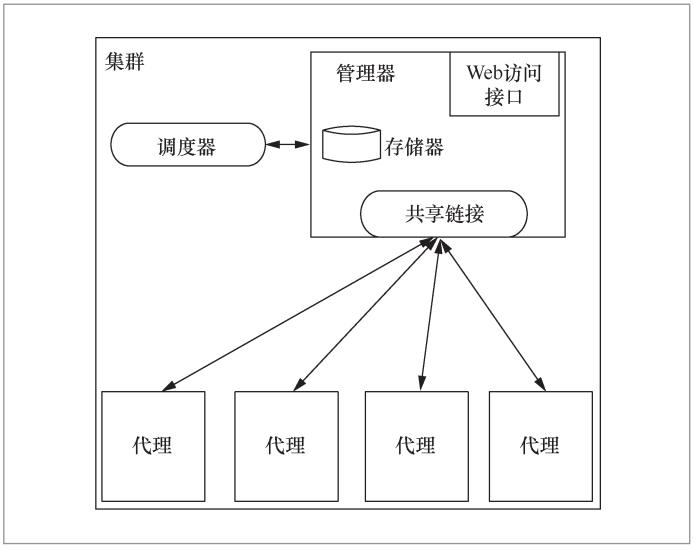

图 2 Borg 资源管理框架

3.3 双层调度

在集中式资源管理框架中，资源调度和任务调度是紧耦合的，任务调度不灵活。双层调度将任务调度和资源调度完全分割，采用松散耦合的方式，主节点只负责各个应用框架的资源分割与分配，应用框架自己根据逻辑，在获得的计算资源上调度任务。

与集中式资源管理框架不同，双层资源管理结构把任务调度从资源管理中分离出去，由应用程序自身负责，因此大大提高了任务调度的灵活性。但是这也使得各个应用程序之间无法得到全局的资源状态。

3.3.1 典型系统Mesos

Mesos调度资源管理框架如图3所示。系统由一个Mesos主服务和运行在不同计算节点上的从服务程序组成。运行在资源管理上的编程框架由两部分组成：计算框架（FW）和执行器。计算框架调度器负责向主服务注册，并获得计算资源，执行器被发送到各个计算节点，用来运行编程框架的任务。Mesos主服务的主要任务是向各个计算框架提供资源清单。资源清单的内容是多个从服务节点上的可用资源的列表。

图3中的第①步，集群节点从服务1向主服务提供4个CPU以及4 GB的主机内存。主服务上的调度器向计算框架1提供这些空闲资源。第②步，主服务向计算框架1发送可以使用的资源。第③步，计算框架1向主服务回复哪些任务使用了这些资源，例如图3中任务1和任务2分别得到2个CPU和2 GB的内存。第④步，主服务向从服务1发送提交任务的命令。

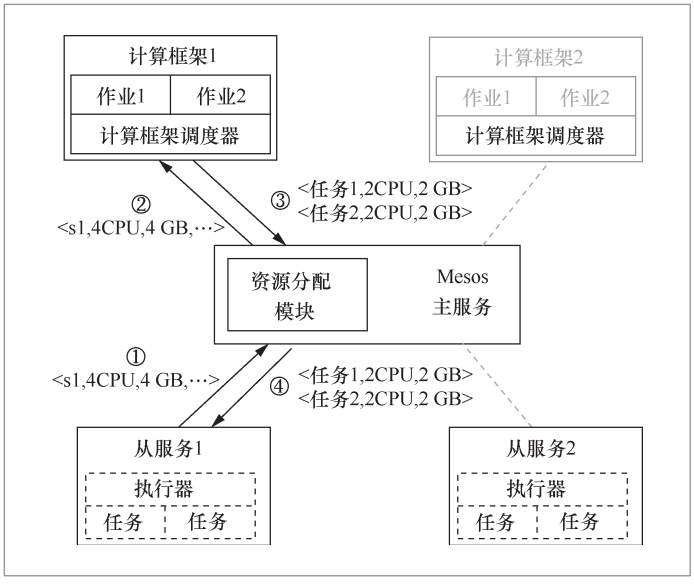

图 3 Mesos 调度资源管理框架

3.3.2 优缺点

双层调度的优点包括：通过增加计算框架调度器，在一定程度上降低了资源管理器的负担；提高了任务调度的灵活性。其缺点包括：可扩展性弱，由于资源管理是集中式的，没有从根本上解决可扩展性问题；调度延迟扩大，加锁机制导致欠缺任务调度并行性。

3.4 共享状态调度

双层资源管理框架解决了任务调度的灵活性，但是为了保证资源状态的一致性而采用加锁机制，导致各个任务调度器只能串行执行。为了解决这些问题，共享状态资源管理框架被提出。该框架中，资源管理仍然采用主从结构，各个任务调度器可获得全部的集群资源，并进行并发任务调度。共享状态资源管理框架的核心是全局集群资源的共享。

3.4.1 典型系统Omega

Omega为每个应用程序的调度器提供全局计算资源，通过乐观封锁协议来维护全局资源的一致状态。主节点维护一个全部集群资源状态（full states）。每个应用程序调度器可以得到一个全部集群资源状态的副本，并依据副本做出调度决策。一旦应用程序调度器给出调度决策，就需要提交调度结果到全部集群资源状态，提交过程遵守事务原则。不管提交是否成功，调度器都需要把结果同步到Omega的全局资源状态，如果有必要，则再次进行调度。

Omega的资源管理框架如图4所示。图4中的第①步，从服务计算节点向资源管理器汇报最新状态。主服务资源分配器形成整个集群的资源状态，分别向计算框架1

以及计算框架2提供全部空闲资源。第②步和第③步，主服务向计算框架1以及计算框架2发送可以使用的全部资源。第④步和第⑤步，计算框架1以及计算框架2向主服务提交全局资源状态变化值。主服务检查计算框架1以及计算框架2提交的内容是否存在冲突，如不存在冲突，执行第⑥步，主服务向从服务发送提交任务的命令。

如果两个不同的作业分配到相同的资源，那么Omega会使用乐观控制协议来处理资源竞争问题。Omega将集群中的资源日志记录以及任务的日志记录当作全局数据，在一个作业的任务执行中，作业的调度器可以请求多种不同的计算资源，当所有计算资源被全部获得且任务执行结束时，这次资源分配才能算成功。当缺乏可用的集群计算资源或任务执行失败时，作业的执行状态会返回执行前的状态。Omega采用传统数据库的死锁检测方法，一旦检测到死锁存在，就撤销一个任务或者一个作业，由于采用乐观封锁协议，所以只有在真正分配资源时才会检查资源的封锁状态。

3.4.2 优缺点

共享状态调度器的优点包括：集群资源的使用达到最优化；不同作业的调度器并行工作，提升了调度效率。其缺点包括：不同计算框架调度策略之间的交互是无法预测的；解决冲突增加了系统的开销。

3.5 全分布式调度

大规模数据分析框架正朝着两个趋势发展：更短的任务执行时间和更大的并行度。调度100 ms内完成的高并发的作业对于调度器来说是一个巨大的挑战，与此同时还要保证高吞吐率和高可用性。

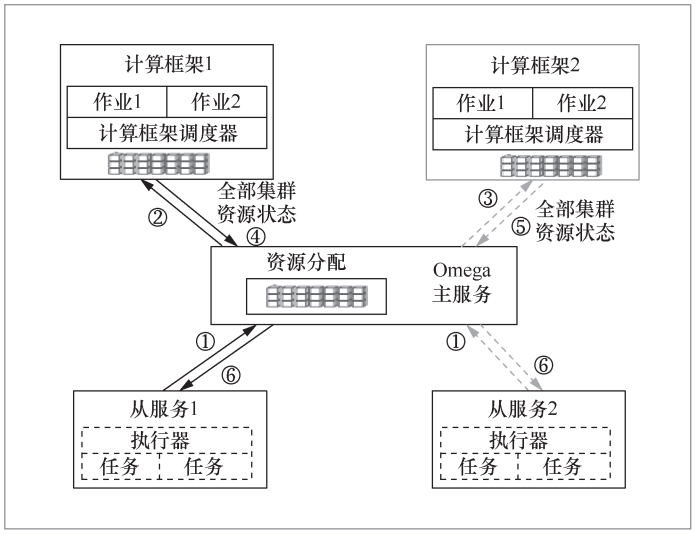

图 4 Omega 资源管理框架

当前分布式处理框架主要采用分层调度的方法。例如Mesos和YARN，由一个中心节点同时维护作业和资源2类信息，调度效率较低，不能很好地胜任短作业的调度。

全分布式调度框架采用无中心节点方式。各个作业的调度器之间不进行任何协调处理，也不进行任何通信和同步。每个作业的调度器只知道自己使用的集群节点的负载状态信息。在分布式调度框架下，作业采用采样的方式在随机选择的节点上提交任务，通常会选择负载较小的节点作为最后的执行节点。作业调度器与双层调度资源管理中的计算框架调度器看起来相似，但分布式调度中不存在集中资源信息，其全局资源状态和资源分配都是根据统计知识和随机采样进行的。

3.5.1 典型系统Apollo

Apollo资源管理的框架如**图5**所示，计算框架用于管理一个作业的生命周期。资源监控器获得整个集群节点的资源信息，并提供给FW使用。每个计算节点上有一个

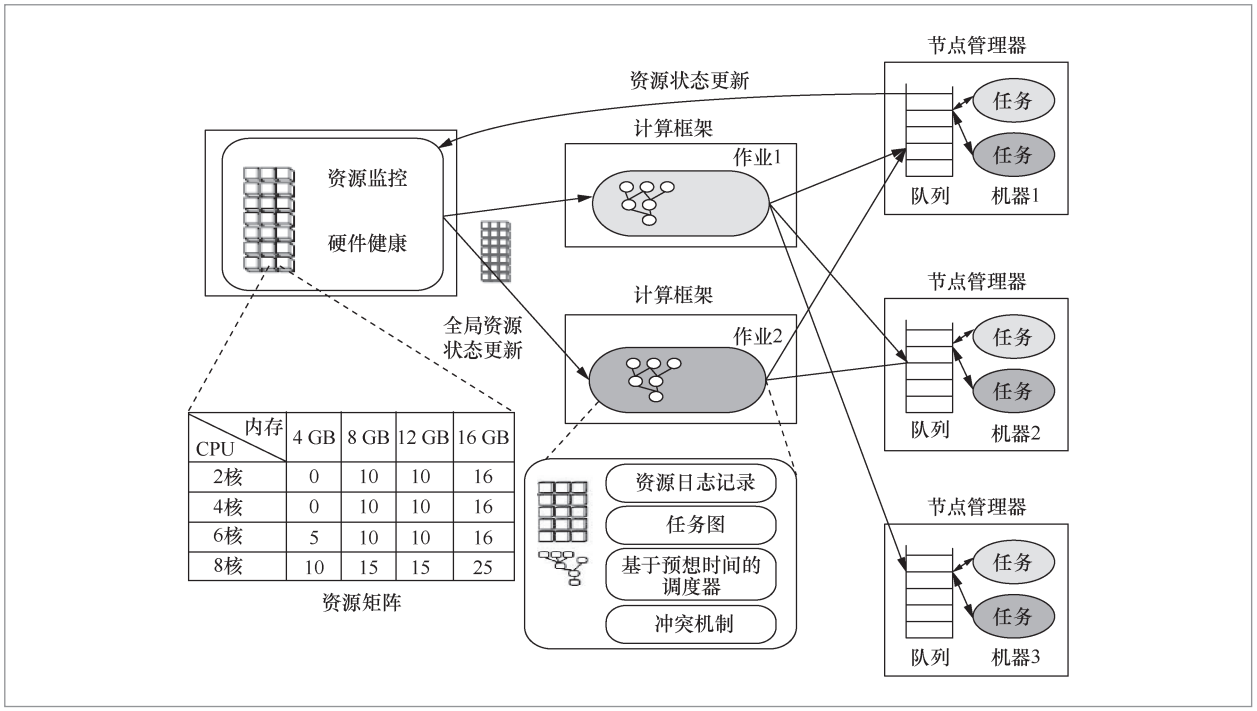

图 5 Apollo 资源管理框架

服务程序，称为节点管理器。PN运行在各个节点上，负责管理本地节点的资源状态，同时也实时调度本地资源。资源监控器动态地获得各个集群节点的资源状态变化，形成一个全局的资源状态视图，为各个FW提供调度依据。

资源监控器被看作一个逻辑实体，在物理实现上存在多种形式。由于资源监控器主要用于动态地收集整个集群中各个计算节点的资源状态，因此，可以使用层次结构或者目录服务的方式来持久化集群资源的状态信息。在分布式集群资源管理中，资源监控器不是性能的关键路径，在资源监控器不可用的场合，可以通过FW持续地进行资源分配。另外，一旦一个任务被分配到某个PN上，资源监控器就可以从PN中得到最新的资源状态变化。

为了预测资源的使用情况以及优化调度策略，每一个PN都维护着一个任务执行队列，并通过任务等待状态预测等待时间矩阵，FW则使用这些信息进行优化调度。但是等待时间矩阵也存在缺陷，如信息太陈旧，使得调度不是最优。Apollo也保留了可靠的实现策略。最后，在为FW提供有保证的资源（如确保服务级别协议（service level agreement，SLA））与实现较高的集群利用率之间存在固有的冲突，因为集群上的负载和作业的资源需求都在不断波动。Apollo通过机会调度解决了这种紧张关系，机会调度创建了第二类任务来使用空闲资源。

3.5.2 优缺点

全分布式调度的优点是分布式调度框架基于简单的同时运行数，将每台机器的同时运行数设为大于1，可以并行运行多个任务，这种资源管理方式简单。其缺点包

括：作业的任务基本相似，因为任务差距较大会导致资源分配不公平以及调度不准确；由于不存在全局资源管理，作业的资源分配无法实现严格公平；调度结果是次优的，因为作业调度器只依据部分知识进行资源分配，无法得到全局的最优决策。

3.6 混合式调度

混合式调度框架是为了解决大量短时间任务和长时间任务混合时，任务的吞吐量和延迟这一矛盾而进行的学术研究，它一方面借鉴了全分布式资源管理框架的优点，另一方面又利用了集中式资源管理框架的优点。目前学术研究中有Tarcil、Mercury和Hawk框架。通常采用两种不同的管理框架，一种是为短时间任务设计的分布式管理框架，另外一种是集中式调度框架，主要服务长时间运行的任务。混合调度框架中的2个部分分别对应集中式和分布式。从现有的文献看，目前尚未出现开源的系统。

3.6.1 典型系统Mercury

如图6所示，Mercury由以下几个部分组成。

（1）节点管理器

节点管理器是一个服务程序，运行在每个集群的计算节点上，它用于与应用程序进行交互，加强每个节点上任务的运行。

（2）资源协调和监控管理框架

资源协调和监控管理框架是一个子系统，包含运行在某个节点上的集中式调度器以及运行在各个不同的集群节点上的分布式调度器，这些分布式调度器是松散耦合关系，通过一个资源协调器进行协调。这个混合的调度器实现了整个集群计算资源的分配。资源分配的单位是容器，其资源类型包含CPU、内存等。

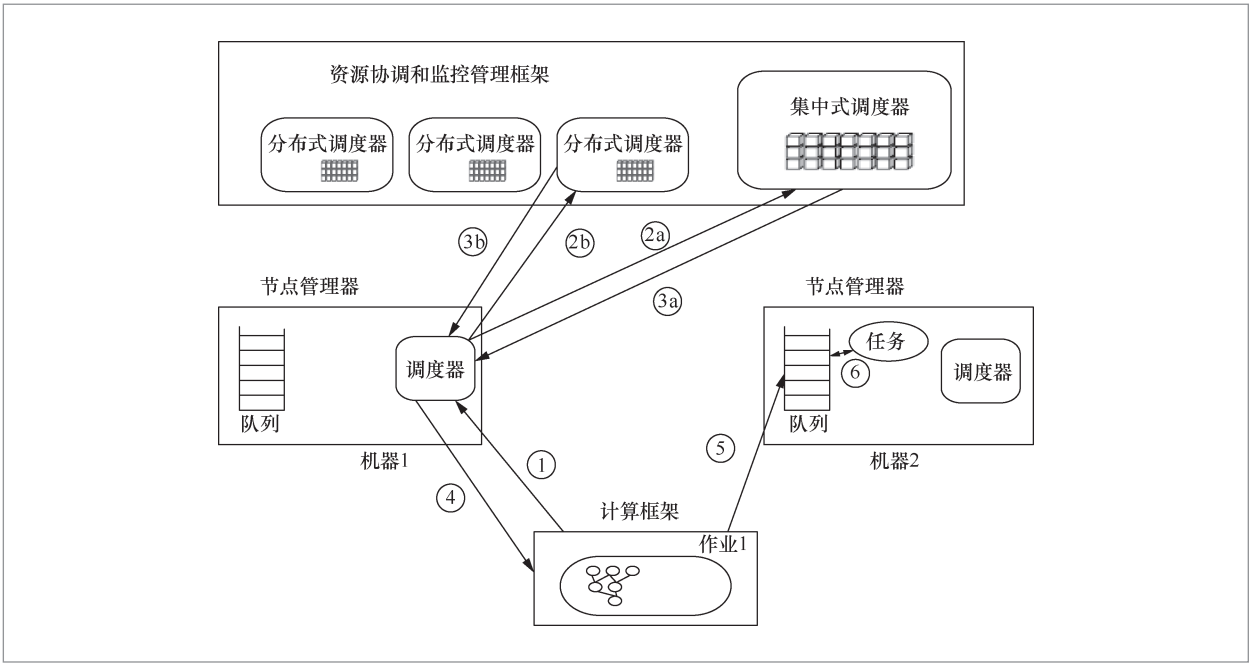

图 6 Mercury 资源管理框架

由于没有将某些特有的计算资源划分给每一个调度器，所以资源分配的过程是动态的。资源分配中，如果不同的任务分配到相同的资源池，那么冲突解决是在PN上实现的。

如果有一个计算框架请求计算资源（如图6中的①），PN收到信息后，会确认资源分配的类型，如果是确保资源需求，那么资源请求就会被传递到分布式调度器（如图6中的②b），如果是排队资源需求，那么资源请求就会被发送到集中式调度器（如图6中的②a）；集中式调度器或者分布式调度器对资源请求进行处理，分配相应的资源到任务，并将调度结果发送到PN的调度器（如图6中的③a或者③b）；PN的调度器得到分配结果后，直接将分配决策转发到不同的FW（如图6中的④）；FW收到分配结果后，启动任务执行（如图6中的⑤）。

3.6.2 优缺点

混合式调度架构的优点是对长短任务的调度采用不同策略后，保证了SLA。其缺点包括：集中式调度器和分布式调度器之间存在资源分配冲突；各个分布式调度器的调度结果可能是最优结果；分布式调度器存在将同一个计算资源分配给不同任务导致的冲突。

3.7 CPU–GPU混合的异构集群资源调度

GPU逐渐成为许多数据流应用程序的关键工具，例如在深度学习/机器学习、数据分析、基因组测序等领域，存在大量的应用程序，这些程序依靠GPU进行加速。在大多数情况下，GPU能够使程序的速度提高10倍；而在一些特定的场合，GPU可以使程序的速度提高300倍。另外，许多现代的深度学习程序直接建立在GPU深度学习库上，如统一计算设备架构（compute unified device architecture，CUDA）深度神经网络库，即cuDNN。可以说，深度学习等许多应用程序如果没有GPU支持就无法生存。

但是，CPU-GPU混合的异构集群资源管理系统在资源调度方面对GPU的支持不尽人意。最早支持GPU的资源管理系统是Mesos，它能够像CPU、内存等资源一样支持GPU。但是，它的支持是粗粒度的，无法实现GPU资源的共享，亦缺乏对GPU的显存、GPU核的管理。虽然它在MPI程序中被很好地使用，但是在数据流计算环境下的应用程序中较少被使用。最近，YARN也开始支持GPU资源，亦将GPU作为一种类型的资源来管理和调度，目前可以有效地支持英伟达设备。它通过资源配置文件设置GPU资源类型，在每个节点上进行调度，隔离的粒度是每个GPU设备，支持将N个GPU设备分配给每个任务（N小于每个节点上可用的GPU设备数）。但是，它也将一个或所有GPU设备分配给任务，是一种粗粒度分配模式，通过控制组群（control groups，CGroups）对GPU进行限制，也支持通过Docker进行资源限制。由于CUDA在Java语言方面的不足，YARN通过第三方的监视框架来监视GPU的使用状况。Kubernetes是借鉴Borg系统的容器管理系统，它通过容器来隔离GPU计算资源，实现GPU资源的共享访问。但是，它是一个应用级别的资源管理系统，如果直接应用于数据流计算环境下的资源管理，还存在一定的困难和不足。

在现有的分布式以及混合式集群资源管理系统中，尚未见到涉及CPU-GPU混合的集群资源管理系统。参考文献[12]对YARN进行了扩展，支持GPU计算资源。该文献只研究了如何根据设置的GPU资源特性防止任务数量超过可用GPU数量限制的问题，

对GPU的资源共享以及多个任务之间分配GPU计算资源的研究相对缺乏。

现有的GPU集群资源管理系统只实现了粗粒度的资源调度，缺乏细粒度的资源调度系统。针对GPU资源的细粒度管理和分配，学术界也存在一些研究：一种是按照时间分片来调度GPU资源，即将多个Kernel函数采用并发的方式，依次共享计算资源；另一种是空间分片，将GPU按照流多处理器（streaming multiprocessors，SM）粒度进行调度，即多个Kernel函数并行提交到不同的SM上。按照空间共享可以真正实现Kernel函数的并行，但是在缺乏CUDA相关调度指令的前提下，研究比较困难，故在时间分片方面的研究较多。但是这些细粒度的资源管理很少涉及GPU集群的管理，亦未涉及CPU-GPU混合的集群资源管理。

4 数据流计算环境下的集群资源管理展望

4.1 存在的问题

（1）未体现真正的资源管理

现有的资源管理（如Mesos、YARN等）只能达到资源协商，不能很好地体现资源管理。Mesos、YARN等资源调度框架只能给出什么时间哪个任务能够使用资源，不能快速、准确地让某个任务得到一个具有优先使用权的资源，并且资源大小保证符合SLA。

（2）支持的作业类型有限

现有的资源管理（如Mesos、YARN等）可以很好地对MapReduce、Spark等类型的作业进行资源分配，但是不能很好地支持其他类型的作业。例如，在设置资源限制时，往往需要指定容器需要的最小资源以及最大资源，而且这些设置对于全体集群有效，如果同时有一些特殊的作业，如内存数据存储系统（Redis）等也需要限制，因其vCore需求很小，而内存需求较大，如何设置容器大小就成为难题。

（3）不被资源管理的作业与被资源管理的作业之间存在冲突

不被资源管理的作业有时会使用大量的资源，此时其就会与被资源管理的作业竞争计算资源，导致资源过度使用，进而使系统性能下降。

（4）不支持有关联关系的作业的资源同步协调

当两个不同的作业之间存在关联关系时，例如一个是生产数据，另外一个是消费数据，它们的资源使用很难协调。生产者拥有大量资源而产生大量数据，消费者却因资源限制而无法消费大量数据。

（5）GPU管理调度算法缺乏，容易引起资源过载

现有的Mesos资源分配按照任务的需求分别对全局的CPU核、内存等进行分配，可能导致一些不需要GPU的任务将节点上的CPU核以及内存全部占用，一旦有新的GPU任务请求资源时，就会面临CPU以及内存资源过载的情况。

（6）不支持GPU的拓扑结构

由于现有的GPU资源管理不考虑GPU之间的拓扑结构，当多个任务之间存在GPU与GPU之间的数据传输时，资源分配系统无法将任务分配到带宽较高的多个GPU上。

4.2 数据流环境下的集群资源管理解决方向

在数据流计算环境下，批处理和流处理可实现无缝融合。而现有的集群资源管

理在批处理方面应用较好，面对批处理与流处理混合的环境，需要考虑批处理和流处理的特点，综合考虑集中和分布调度。集群管理既可以支持流数据的计算，也支持批数据的计算，实现分布式与集中式资源管理的综合统一。

数据流计算环境下，许多高通量视频流数据的处理离不开CPU加速，即GPU，CPU-GPU混合的资源管理系统是另一个研究重点。区别于现有的GPU集群资源管理，多个Kernel函数的并行运行是其目的，因此GPU资源共享是一个关键点。

数据流计算环境下，数据的注入和处理需要同步。生产数据的应用程序和处理数据的应用程序之间存在生产-消费关系，为了防止数据处理速度过慢导致的阻塞或者数据处理过快导致的“饥饿”现象，既需要按照数据注入的速度或者数据的大小来动态地调整计算资源的分配，也要考虑各个计算节点的硬件性能的差异或者任务差异带来的处理不匹配问题，例如CPU性能差异、任务需求差异、计算节点网络带宽限制等。

在数据流计算环境下，对于多个应用之间存在相互依赖的应用程序，为满足任务在各个GPU之间的流水调度，集群资源管理需要维护全局的GPU设备的拓扑结构信息，其目的在于减少GPU之间的通信开销。

5 结束语

数据流计算环境下的集群资源管理和任务调度一直是研究的热点。国际上有众多的研究人员从事资源管理和任务调度的工作，不仅包括谷歌、微软、脸书等大型的互联网企业，还包括一些大学的研究小组。他们都有自己的研究特色，并且在某些方面取得了较好的效果，为集群资源管理的研究注入了活力。正是在他们的推动下，集群资源管理框架和任务调度策略产生了不同的系统结构。在中国，腾讯、阿里巴巴、百度等一些大型的互联网公司已经取得了不小的收获，他们大多数采用改造开源的集群管理来适应自己的业务需求，很少提出一种全新的架构。相对于数据流计算应用在国内的整体形势来说，关于集群资源管理和任务调度的研究成果还是较少。希望本文能为从事数据流计算的开发者提供资源管理和任务调度方面的帮助。

参考文献:

[1] HOVESTADT M, KAO O, KELLER A, et al. Scheduling in HPC resource management systems: queuing vs planning[J]. Genetica, 2003, 112-113(1): 445-461.

[2] MISHRA M K, PATEL Y S, ROUT Y, et al. A survey on scheduling heuristics in grid computing environment[J]. International Journal of Modern Education and Computer Science, 2014, 6(10): 57-77.

[3] 杜小勇, 陈跃国, 范举, 等. 数据整理——大数据治理的关键技术[J]. 大数据, 2019, 5(3): 13-22.
DU X Y, CHEN Y G, FAN J, et al. Data wrangling: a key technique of data governance[J]. Big Data Research, 2019, 5(3): 13-22.

[4] 陈康, 郑纬民. 云计算: 系统实例与研究现状[J]. 软件学报, 2009, 20(5): 1337-1348.
CHEN K, ZHENG W M. Cloud computing: system instances and current research[J]. Journal of Software, 2009, 20(5): 1337-1348.

[5] KARANASOS K, RAO S, CURINO C, et al. Mercury: hybrid centralized and distributed scheduling in large shared clusters[C]// 2015 USENIX Annual

Technical Conference. Berkeley: USENIX Association, 2015: 485-497.

[6] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

[7] PARK J J K, PARK Y, MAHLKE S. Dynamic resource management for efficient utilization of multitasking GPUs[C]//The 22nd International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM Press, 2017: 527-540.

[8] ZAHARIA M, CHOWDHURY M, DAS T, et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing[C]// The 9th USENIX Networked Systems Design and Implementation. Berkeley: USENIX Association, 2012: 2-14.

[9] ARMBRUST M, XIN R S, LIAN C, et al. Spark SQL: relational data processing in Spark[C]// The 2015 ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2015: 1383-1394.

[10] CARBONE P, KATSIFODIMOS A, EWEN S, et al. Apache Flink: stream and batch processing in a single engine[J]. IEEE Data Engineering Bulletin, 2015, 38(4): 28-38.

[11] FUKUTOMI D, IIDA Y, AZUMI T, et al. GPUhd: augmenting YARN with GPU resource management[C]// International Conference on High Performance Computing in Asia-Pacific Region. New York: ACM Press, 2018: 127-136.

[12] VERMA, A, PEDROSA, L, KORUPOLU M, et al. Large-scale cluster management at Google with Borg[C]// The 10th European Conference on Computer Systems. New York: ACM Press, 2015: 1-17.

[13] HINDMAN B, KONWINSKI A, ZAHARIA M, et al. Mesos: a platform for fine-grained resource sharing in the data center[C]// The 8th USENIX Conference on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2011: 295-308.

[14] BOUTIN E, EKANAYAKE J, LIN W, et al. Apollo: scalable and coordinated scheduling for cloud-scale computing[C]// The 11th USENIX Conference on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2014: 285-300.

[15] KONSTANTINOS K, SRIRAM R, CARLO C, et al. Mercury: hybrid centralized and distributed scheduling in large shared clusters[C]// 2015 USENIX Annual Technical Conference. Berkeley: USENIX Association, 2015: 485-497.

[16] AKIDAU T, BRADSHAW R, CHAMBERS C, et al. The dataflow model: a practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing[J]. Proceedings of the VLDB Endowment, 2015, 8(12): 1792-1803.

作者简介

汤小春（1969- ），男，博士，西北工业大学计算机学院副教授，主要研究方向为大数据计算、大图数据挖掘、集群资源管理等。

符莹（1996- ），女，西北工业大学计算机学院硕士生，主要研究方向为大数据计算、集群资源管理等。

丁朝（1995- ），男，西北工业大学计算机学院硕士生，主要研究方向为大数据计算、集群资源管理等。

毛安琪（1996- ），女，西北工业大学计算机学院硕士生，主要研究方向为大数据计算、集群资源管理等。

李战怀（1961- ），男，博士，西北工业大学计算机学院教授，大数据存储与管理工业和信息化部重点实验室主任，主要研究方向为数据库理论与技术、数据流、数据密集型计算、内存计算、数据挖掘等。

收稿日期： 2020-01-19

基金项目： 国家重点研发计划基金资助项目（No.2018YFB1003400）

Foundation Item: The National Key Research and Development Program of China(No.2018YFB1003400)