分　数：	
评卷人：	

華中科技大學

研究生（数据中心技术）课程论文（报告）

题 目：大数据平台的任务调度

学　　号 M202173810

姓　　名 舒欢

专　　业 电子信息

课程指导教师 施展 童薇

院（系、所） 计算机科学与技术学院

2022 年 1 月 5 日

大数据平台的任务调度

舒欢[1)]

[1)](华中科技大学 计算机科学与技术学院 武汉 430074)

摘 要 当前数据中心的一个趋势是部署异构存储设备，以满足各种大数据工作负载的不同存储需求。例如，许多节点同时配置 SSD 和 HDD。HDFS 还引入了异构存储感知特性，以适应这种混合存储集群。而目前大数据处理平台(Hadoop、Spark 等)上的任务调度器仅利用数据局部性原理考虑网络数据传输的开销。在异构存储集群中，任务完成时间也会受到数据所在存储设备(ssd 和 hdd)的速度的影响。忽略存储设备的不同速度，会导致 SSD 等高速设备利用率不高。为解决这个问题，近些年来，研究员研究实现了多种大数据平台中的任务调度算法，以确保大数据平台能够充分利用异构的存储系统提高集群的整体性能，本文对这些算法进行研究，分析其实现的原理、流程等。

关键词 数据中心、大数据平台、存储异构、任务调度

Task Scheduling for Big Data Platforms

Huan Shu[1)]

[1)](College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074)

Abstract A current trend in the data center is to deploy heterogeneous storage devices to meet the different storage requirements of various big data workloads. For example, many nodes are configured with both SSDs and HDDs. HDFS has also introduced heterogeneous storage-aware features to accommodate such hybrid storage clusters. In contrast, the task scheduler on current Big Data processing platforms (Hadoop, Spark, etc.) only considers the overhead of network data transfer using the data locality principle. In heterogeneous storage clusters, task completion time is also affected by the speed of the storage devices (ssd and hdd) where the data is located. Ignoring the different speeds of storage devices can lead to underutilization of high-speed devices such as SSDs. To solve this problem, in recent years, researchers have studied and implemented a variety of task scheduling algorithms in big data platforms to ensure that big data platforms can make full use of heterogeneous storage systems to improve the overall performance of the cluster. In this paper, we study these algorithms and analyze the principles and processes of their implementation

Key words Data centers; big data platforms; storage heterogeneity; task scheduling

1 引言

近年来，Hadoop、Spark 等大数据处理平台应运而生，成为广泛部署的高效框架，支持各种大数据工作负载，如 KMeans、PageRank 等迭代应用、类 sql 查询等。不同类型的负载对存储业务的性能有不同的要求。例如，Grep 是一个 I/O 密集型应用程序，I/O 开销占了执行时间的大部分，而 KMeans 是一个 cpu 密集型应用程序，它花费了大部分时间执行计算。随着存储技术的快速发展，不同类型的存储设备的出现能够满足各种 I/O 需求。

因此，研究人员和 IT 从业人员面临的主要挑战是，如何发展底层存储和 I/O 基础设施，以应对指数增长的数据量和存储服务的不同性能需求，并以经济可行的方式做到这一点。异构存储系统的出现是存储技术的一个有前景的趋势，它采用了不同类型的存储设备，如 HDD、SSD，特别是在大数据工作负载中广泛使用的 HDFS，从 2.3.0 版本开始支持异构存储。也就是说，HDFS 可以在每个 DataNode 上使用 SSD 和 HDD。通过充分利用存储设备不同速度的优点，这提供了高成本效益的潜力。例如，HDD 可以存储访问频率较低的数据，为大数据量提供廉价的大容量存储;而 SSD 可以存储访问频率较高的数据，性能较好

数据处理工作，称为作业，由一系列相关阶段组成。每个阶段由许多并行任务组成。常见的阶段包括扩张和减少。Map 任务从 HDFS 读取数据，通过用户定义的 Map 函数处理数据，并将输出的数据发送给后续阶段。由于数据分布在不同的机器上，网络容量是有限的，所以希望将映射任务放置在存储输入数据的机器上或附近，即数据局部性。将映射任务分配给机器的算法称为任务调度算法。众所周知，改进局部性可以通过减少网络流量来减少地图任务的处理时间，因为很少有 map 任务需要远程获取数据。通过增加数据局部性来提高系统效率的尝试已经被提出。但在异构存储集群中，由于数据存储在不同类型的存储设备上，且存储设备的读写性能不同，导致本地任务执行时间不同。当前的任务调度策略只考虑数据局部性，没有区分存储类型，不能充分利用异构存储的优势。此外，当今数据中心的互联网络正从 10GE 向 25GE 或 40GE，甚至 IB 过渡。数据局部性对任务执行时间的影响随着网络速度的提高逐渐减小。相反，I/O 性能对任务执行时间的影响越来越大。

本文针对异构环境下的任务调度算法进行了一些研究，分析各个任务调度算法的实现原理、流程等，并且分析各个算法的实现的角度，以及各自的区别和联系，从而更好的理解异构环境下大数据平台但是任务调度算法。

2 数据中心概述

2.1 HDFS异构存储

自 Hadoop-2.3.0 以来，HDFS 在其设计中引入了异构存储特性，使得应用能够充分利用不同类型存储设备的优势。存储类型标识底层存储介质，HDFS 支持以下存储类型：

- **RAM_DISK**:这种特殊的内存存储类型用于加速低耐久性、单副本的写操作；
- **SSD**:固态硬盘，用于存储热数据和 I/ O 密集型应用；
- **DISK**:硬盘驱动器相对便宜，并提供顺序的 I/O 性能。这是默认的存储类型；
- **ARCHIVE**:归档存储用于非常密集的存储，对于很少访问的数据非常有用。这种存储类型每 TB 通常比普通硬盘便宜

当你在数据节点上配置数据目录时，你可以通过在路径前加上括号内的存储类型来指定它使用的存储类型。如果你没有指定存储类型，则假定它是 DISK。此外，HDFS 有六种预先配置的存储策略，如表 1 所示。当创建一个 HDFS 文件或目录时，应用程序应该为这个文件或目录指定一个存储策略。否则，就会使用默认的策略，即 "Hot"。

表 1 HDFS 存储策略

Storage Policy	Description
Hot (default)	All replicas are stored on DISK
Cold	All replicas are stored on ARCHIVE
Warm	One replica is stored on DISK and the others are stored on ARCHIVE
ALL_SSD	All replicas are stored on SSD
ONE_SSD	One replica is stored on SSD and the others are stored on DISK
LAZY_PERSIST	The replica is written to RAM_DISK and then lazily persisted to DISK

2.2 HDFS异构存储与当前任务调度策略不匹配

如上所述，目前 HDFS 已经考虑了异构存储，并支持多种存储策略来区分存储设备的类型。当前的任务调度策略由于考虑了较高的网络开销，只将数据局部性作为调度任务的唯一因素，并且忽略了

异构存储导致的数据存储类型的不同。在异构存储环境中，由于数据分布在不同类型的存储设备上，对于 HDFS I/ O 密集型任务，即使它们都在本地执行，由于不同类型的存储设备(即从 SSD 读取数据 vs HDD 读取数据)的性能，它们的执行时间仍然存在很大差异，最后，导致不同的作业执行时间。此外，当前 HDFS 支持异构存储，如 ONESSD 存储策略，充分利用 SSD 的优势。但当前的任务调度策略不支持不同类型的存储设备。因此，当前的任务调度策略并没有充分利用 HDFS 异构存储特性的优势。

2.3 高速网络对数据局部性的影响

随着网络性能(10GigE、40GigE 等)的快速提高和现代互联技术在大数据处理集群中的广泛应用，数据局部性对任务执行时间的影响越来越小，通过网络获取数据的速度越来越快。随着互联网带框的不断增加，任务的执行时间瓶颈正在向存储设备的 I/O 性能转移。

3 研究进展

3.1 RUPAM算法

RUPAM 算法是一个分布式数据处理框架的异构感知任务调度程序，其考虑了底层资源的异构性以及应用程序每个阶段中每个任务的各种资源使用特征。RUPAM 管理一个作业中所有任务的生命周期。为此，其使用了一种不牺牲数据局部性的自适应启发式算法，以一种动态的方式将机器与正确的任务进行有效匹配，从而最大限度地从机器的资源中获益。启发式与实时资源利用相结合，使得 RUPAM 避免了不同资源需求的资源争和任务重叠，从而获得较高的整体性能。

RUPAM 主要有三个组件构成，分别是资源监视器（RM）、任务管理器（TM）以及调度器（Dispatcher）。RM 负责系统资源的实时监控，它有一个运行在 Spark master 上的中央 Monitor 和一个运行在每个 Spark worker 节点上的分布式采集器，采集器报告资源的使用情况，如每个节点上的 CPU、内存、网络、I/O 和 GPU，监视器手机并记录各个来自各个节点的信息，同时 RM 还可以根据资源能力进行扩展，从而收集到跟多的信息，如 NVM 设备，TM 负责跟踪任务的资源使用情况，以确定每个任务的任何资源瓶颈。调度器组件负责实现 RUPAM 的主要逻辑流程，如确定每个节点启动的执行器的大小，在特定节点上启动的任务数量，将任务与适合的节点匹配，并根据多种因素调度任务。

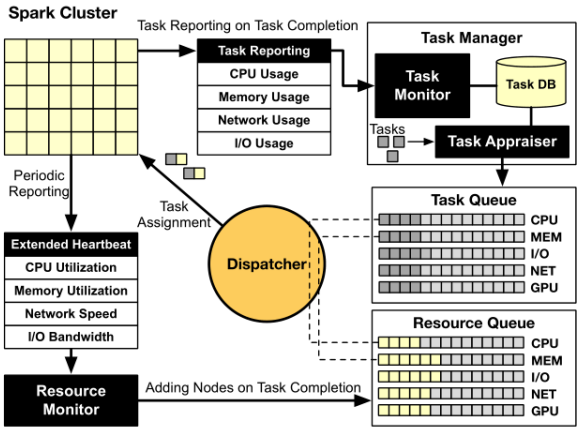

图 1 RUPAM 系统结构图

RUPAM 的动态调度机制依赖于实时的资源监控和任务描述，即，当一个节点可以运行一个任务时，RUPAM 应该将一个具有匹配性能特征的任务分配给该节点的资源特征，如若该节点的内存容量较大，则需要分配一个需要较大内存容量的任务给该节点。然而，一个节点的资源利用率并不是一成不变的。可用的资源会随着任务的启动和完成而改变。保持每个节点的最新资源利用率是至关重要的。此外，同一个任务的资源瓶颈可能会随着它所执行的节点的状态而改变。例如，一个任务在 CPU 时钟速度较差的节点上执行时，可能有 CPU 的瓶颈，但同一个任务在下一次迭代中，在一个强大的节点上执行时，可能会花费大量的时间在网络上洗数据（以弥补 CPU 的瓶颈）。一个节点上的任务之间的资源争夺也会影响这种特性。因此，在整个生命周期中跟踪任务特征，以决定给定任务的最佳节点的权衡也很重要。为此，RM 实时监控每个节点的资源利用率，以提供最新的利用率指标，而 TM 在应用程序执行时跟踪每个任务，并在任务完成时收集统计数据。

资源监控：为了调度员确定每个资源类别的最佳节点，RUPAM 为每个资源类型使用一个优先级队列（图 1 中的“资源队列”），即 CPU、GPU、网络、存储和内存，每个队列都按容量降序排序（最强大/能力/容量第一），相关利用率升序排序（使用最少的第一），RUPAM 只需要在单轮中整理出小的节点子集，所有的队列都可以在下一轮报价前清空，通过这种方式，保持了资源队列的大小和相关

排序时间的低复杂性，这使 RUPAM 的开销最小化。

任务监控：为了给具有一定能力的特定节点选择一个合适的任务，RUPAM 还需要确定任务的特性。为此，TM 监控每个应用程序的任务资源使用情况，并记录这些信息。RUPAM 使用任务特征数据库来存储任务指标。一旦任务被提交给 TM，RUPAM 首先在任务特征数据库中搜索任务并检索其特征。为了将具有不同资源需求的任务在一个阶段中分开，TM 也为每种资源类型保留一个队列（图 1 中的“任务队列”）。如前所述，这些队列将在当前任务完成后和新任务波到达前被重置。

任务调度：于是我们可以将分布式异构存储上的任务调度问题建模成 n 个任务在 m 个机器上的调度问题，我们的目标是最小化所有机器上所有任务的总处理时间，易知获取最优解是 NP 难问题，因此 RUPAM 采用了一种基于贪心算法的启发式算法，该算法一种列表调度算法。即 TM 为每个任务确定资源瓶颈，并将其传递给调度器进行调度。调度器等待底层节点可用。结合来自 TM 的任务指标和来自 RM 的节点信息，调度器能够将任务匹配到节点。

在 RM 填充“资源队列”之后，RUPAM 以轮询的方式每次从每个资源队列中退出一个节点，以确保没有一个单一资源类型的任务被耗尽。由于从一个特定的优先级队列中脱队的第一个节点将具有最高的容量/能力和最低的可用资源类型的利用率，调度器会检查该资源类型队列中的任务，确保该节点有足够的可用内存来启动任务,最后，按照 PROCESS_LOCAL(数据在 java 进程内)、NODE_LOCAL(数据在节点上)、RACK_LOCAL(数据在同一机架上的节点上)和 ANY(数据在不同机架上的节点上)的顺序找到该节点的最佳局部性的任务。这个启发式方法贪婪地找到一个对资源 R 具有最佳能力和最低竞争的节点 N，然后将一个任务 T 调度到 N 上，这个任务在之前的运行中以 R 为瓶颈，现在 N 为 T 提供了最佳的位置。RUPAM 并不试图为每个任务找到最佳调度策略，因为这可能导致巨大的调度延迟，而且会产生反效果。相反，RUPAM 为一个任务尝试不同的节点分配，例如，具有良好的 CPU 或更好的 I/O 吞吐量，并记录该任务取得最佳性能的节点。这个节点被用来安排任务，即使该任务在该节点上有一些瓶颈，例如 CPU。这种将任务 "锁定 "在它能提供最佳观察性能的节点上的做法，也能防止由于任务特性的临时波动而在节点之间来回移动任务。

3.2 H-Scheduler算法

为了充分利用数据局部性和异构存储来加速任务的执行，H-Scheduler 同时考虑了数据局部性和存储类型作为任务调度决策的因素，它有两个步骤。第一步是任务分类，按照存储类型对一个作业中的任务进行分类。第二步是任务调度，根据优先级设置，将分类后的任务分配到合适的节点上。只有 HDFS 的 I/O 密集型作业可以从高性能的存储设备中受益，因为高性能的存储设备可以加速 map 任务的完成。H-Scheduler 的目标是通过利用存储设备的异质性来减少 HDFS I/O 密集型 map 任务的执行时间。

（1）任务分类

对于每个任务，如果与该任务关联的数据块存储在该节点上的某个设备(HDD 或 SSD)上，我们称该任务为该任务的本地节点，我们称该任务为该节点上的本地任务;否则，该节点被称为该任务的远程节点，相应地，该任务被称为该节点上的远程任务。在调度任务之前，H-Scheduler 应该知道每个任务处理的数据的存储类型。因此，我们将存储信息从 HDFS 传递到任务调度层，如图 2 所示。然后，H-Scheduler 可以根据任务的位置和与任务关联的数据块的存储类型对任务进行分类。任务分为四种类型。它们是本地 SSD 任务、远程 SSD 任务、本地 HDD 任务和远程 HDD 任务。

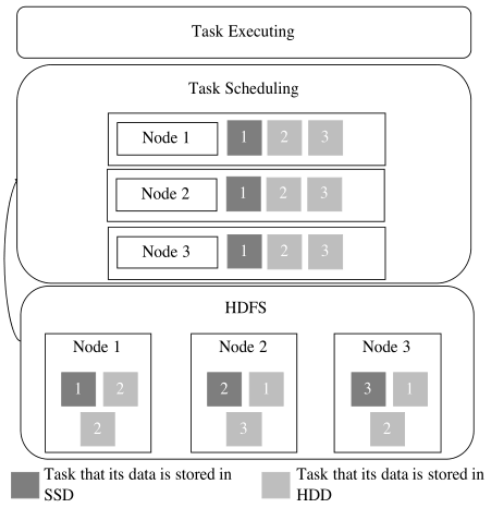

图 2 节点存储信息图

（2）任务优先级

如上所述，任务有四种类型，即 SSD 本地任务、SSD 远程任务、HDD 本地任务、HDD 远程任务。我们需要通过为这四种类型的任务设置优先级顺

序来确定它们的执行顺序，以便优化作业的执行时间。令“lSSD”、“rSSD”、“lHDD”、“rHDD”分别表示本地 SSD 任务、远端 SSD 任务、本地 HDD 任务和远端 HDD 任务，则易知上述任务的执行时间大小如公式（1）所示。

$$t_{lSSD} < t_{rSSD} < t_{lHDD} \approx t_{rHDD} \tag{1}$$

所以作业的执行时间 T 可以用公式（2）表示，其中 N_{CPU} 表示 CPU 的个数。

$$T = \frac{N_{lSSD} \cdot t_{lSSD} + N_{rSSD} \cdot t_{rSSD}}{N_{CPU}} + \frac{N_{lHDD} \cdot t_{lHDD} + N_{rHDD} \cdot t_{rHDD}}{N_{CPU}} \tag{2}$$

由于 $t_{lHDD} \approx t_{rHDD}$，所以我们简单地使用 t_{HDD} 来代替它们。因此,我们公式（3），同时我们可以用 Δt 来表示一个远程 SSD 任务的执行时间比一个本地 SSD 任务的执行时间差，即公式（4）。

$$N_{lHDD} \cdot t_{lHDD} + N_{rHDD} \cdot t_{rHDD} = N_{HDD} \cdot t_{HDD} \tag{3}$$

$$t_{rSSD} - t_{lSSD} = \Delta t \tag{4}$$

将公式（3）（4）带入而可以得到公式（5），而在公式（5）中，N_{hdd}, t_{HDD}, N_{SSD}, t_{lSSD}，Δt 是常量，因此作业执行时间仅由 N_{lSSD} 决定。也就是说，为了使作业的执行时间最小化，本地 SSD 任务(即 N_{lSSD})的数量应该最大化。

$$T = \frac{N_{lSSD} \cdot t_{lSSD} + (N_{SSD} - N_{lSSD}) \cdot (t_{lSSD} + \Delta t)}{N_{CPU}} + \frac{N_{HDD} \cdot t_{HDD}}{N_{CPU}} \tag{5}$$

由于 H-Scheduler 的原理是让更多的 SSD 任务在本地执行，所以本地 SSD 任务应该具有最高的优先级。虽然本地 HDD 任务和远程 HDD 任务的执行时间相同，但为了避免不必要的网络开销，本地 HDD 任务的优先级应该高于远程 HDD 任务。

当某个节点的本地任务用完后，它会从其他节点读取数据，执行远程任务。如果选择先执行一个远程 SSD 任务，则任务执行时间比本地 SSD 任务增加 Δt。但如果首先选择执行远程 HDD 任务，则由于 $t_{lHDD} \approx t_{rHDD}$，作业执行时间增加不大。在远程 HDD 任务执行过程中，可能会释放部分计算资源，用于执行部分本地 SSD 任务。因此，任务的执行时间可能会比远程 SSD 任务立即执行的时间短。因此，与远程立即执行 SSD 任务相比，优先调度远程 HDD 任务，可以增加 SSD 任务在本地执行的概率，减少任务执行时间。因此，我们将远程 SSD 任务的优先级设置为最低。

基于以上讨论，HScheduler 的优先级顺序被设计为:本地 SSD 任务>本地 HDD 任务>远端 HDD 任务>远端 SSD 任务。

（3）节点选择

节点选择指的是每个远程任务从其中读取数据，因为每个数据块有三个副本，它们分别驻留在三个节点上。根据上面讨论的 H-Scheduler 的优先级顺序，当一个节点没有任何本地任务时，它需要从其他节点读取数据，执行远程任务。这样既可以充分利用其计算资源，又可以增加作业执行过程的并行度，从而减少作业执行时间。当执行远程任务时，我们应该确定远程任务从哪个节点读取数据，有三种可能的节点选择策略：

- 随机选择；
- 最多选择：选择拥有最多本地任务的节点；
- 最少选择：选择本地任务数量最少的节点；

假设所有节点的计算资源相等，但它们的本地任务数量不同，因此每个节点完成其本地任务的时间也不同。当一个节点完成其本地任务，其计算资源变得空闲时，它将从其他节点读取数据以执行远程任务。假设 X 是分配给工作 J 的任何计算节点，X 有工作 J 的 k 个本地任务。假设 Y 是工作 J 的一个计算节点，Y 有工作 J 的最多的本地任务。m-k 越大，节点 X 在完成本地任务后有更多时间执行远程任务。这意味着，节点 X 将执行更多的远程任务。所以缩小节点间 m 和 k 的差距可以减少任务执行时间。随机选择使差距保持稳定，最小选择减少了 k，从而扩大了差距，而最大选择减少了 m，从而缩小了差距。

然而，在实际环境中，如果太多的任务同时从同一节点读取数据，特别是数据存储在磁盘上，“最多选择”可能会增加作业执行时间。因此，从同一节点上同时读取数据的任务数量应该设置一个阈值。

3.3 Trident算法

Trident 算法是一种由原则的任务调度算法，可以利用底层分层存储系统提供的存储类型信息，以局部感知和存储层感知的方式做出最佳调度决策。具体地说，调度问题被编码到一个任务和可用集群资源的二部图中，其中的边权表示基于位置和存储层信息读取数据的成本。然后将该问题表述为最小代价最大匹配优化问题，并使用标准求解器求解最优任务分配。

数据处理平台，如 Hadoop 和 Spark，负责为应用程序分配资源和调度任务的执行。在图 3 的例子中，假设任务Ti想要处理相应的数据块Bi，调度器可以实现两个数据本地任务（$T1$ 和$T3$）以及一个机架本地任务（$T2$）。当考虑到存储层时，Trident 算法进一步将任务$T1$ 和$T3$ 分类为内存本地和固态硬盘本地。虽然目前的调度器只考虑了数据的局部性，但考虑存储层是充分利用分层存储所带来的好处的关键。

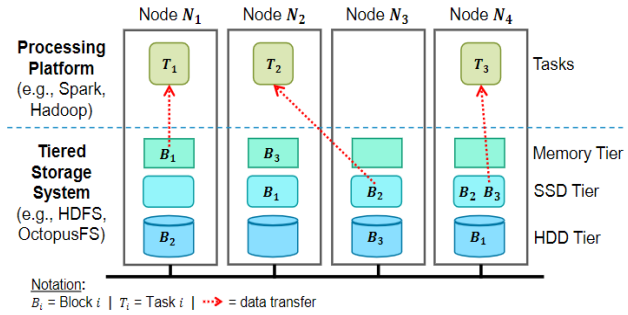

图 3 节点架构图

（1）成本定义

一个典型的计算集群由一组节点N={N_1, ..., N_r}组成，以机架网络拓扑结构排列。集群提供一组资源R={R_1, ..., R_m}用于执行任务。资源代表物理资源的逻辑捆绑（例如，⟨1 个 CPU，2GB RAM⟩），例如 Hadoop 中的容器或 Spark 中的执行器插槽，它被绑定到一个特定的节点。对于每个资源Rj，我们将其位置定义为$L(R_j)$=N_k，其中N_k∈N。最后，一组任务T={T_1, ..., T_n}需要在集群上执行的资源。

在传统的大数据环境中（即在没有分层存储的情况下），每个任务都包含一个基于它要处理的数据的位置的首选节点位置列表。例如，如果一个任务要处理一个复制在节点N_1、N_2和N_4上的数据块，它的首选位置列表就包含这三个节点。然而，当数据存储在一个分层的存储系统中，每个块的存储层也是可用的。 因此，我们将任务的首选位置$P(T_i)$=[⟨N_k,p_k^i⟩]定义为一对列表，其中一对中的第一个条目代表节点N_k∈N，第二个条目p_k^i∈R 代表存储层。我们把p_k^i定义为一个数字分数，代表从该存储层读取数据的成本。因此，分数越低，性能就越好。可以使用各种指标来设置分数，但其绝对值并不像代表各层的相对性能那么重要。例如，如果内存、SSD 和 HDD 介质的 I/O 带宽分别为 3200、400 和 160MB/s，那么分数 1、8 和 20 将反映从这三个层级读取数据的相对成本。

在资源Rj上调度一个任务T_i，将产生一个基于以下成本函数的分配成本C，其具体的公式如下公式（6）所示：

$$C(T_i,R_j)=\begin{cases} p_k^i & \text{if } L(R_j)=N_k \text{ in } P(T_i) \\ c_1+p_k^i & \text{if } L(R_j) \text{ in same rack as some } N_k \text{ in } P(T_i) \\ c_2 & \text{otherwise (with } c_2 \gg c_1\text{)} \end{cases} \quad (6)$$

根据公式 6，如果资源R_j的位置是N_k中的一个节点T_i的首选位置，成本将等于相应的层级偏好分数p_k^i。或者，如果R_j与N_k中的一个节点T_i在同一个机架上，成本将等于相应的偏好分数p_k^i加上一个常数c_1，这代表了一个机架内的网络传输成本。否则，成本将等于一个常数c_2，代表跨机架的网络传输成本。机架间的网络成本通常比机架内的成本高得多，并且使本地阅读 I/O 成本相形见绌；因此，我们不给c_2添加偏好分数。

（2）最小成本最大匹配问题

根据上面定义，任务调度问题可以被编码为一个二方图G=（T,R,E）。顶点集T和R分别对应于任务和资源，并共同构成图G的顶点。边集E中的每条边（T_i,R_j）都将顶点T_i∈T连接到顶点R_j∈R，其成本如公式 1 所规定。寻找最优任务调度即相当于在二叉图T中找到一个最大匹配M={(T_i,R_j)}，其中(Ti,Rj)∈T ×R，使总成本函数最小化。

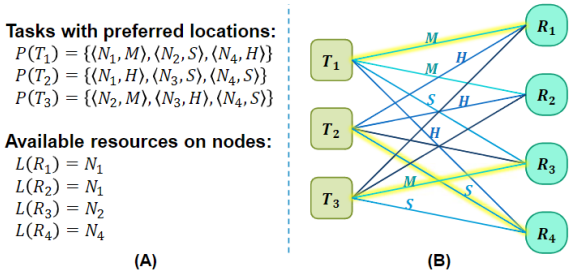

图 4 二分图

图 4(A)展示了一个有三个任务和位于三个不同节点上的四个可用资源的例子。每个任务Ti根据图 3 所示的相应数据块Bi的存储位置，有 3 个首选位置（即⟨节点、层⟩对）的列表。为了便于参考，每个层级的偏好得分用常数M、S和D表示，对应于底层存储系统的内存、SSD 和 HDD 层级，其中M<S<D。所创建的具有 7 个顶点（3 个代表任务，4 个代表资源）的二方图在图 4（B）中显示。每条边都对应于一个任务对一个资源的潜在分配，并标明用公式 1 计算的成本。本例中任务调度的目标是选择形成最大匹配的三条边，并使总成本最小。最

佳的任务分配（见图 4（B）中突出显示的边）包括两个内存本地任务（T_1 在R_1 和T_3 在R_3）和一个 SSD 本地任务（T_2 在R_4）。

有几个标准的求解器可以用来解决最小成本的最大匹配问题，并找到最佳的任务分配，包括 Simplex 算法、Ford-Fulkersonmethod 和 Hungarian Algorithm。虽然 Simplex 算法对小问题有很好的表现，但对大问题来说效率不高，因为它的旋转操作变得很昂贵。它的平均运行时间复杂度为$O(\max(n,m)^3)$，其中n为任务数，m为资源数，但有一个组合的较差情况复杂度。Ford-Fulkerson 方法需要将问题转换为最小成本的最大流量问题，在这种情况下，复杂度为$O((n+m)nm)$。Trident 选择使用匈牙利算法，因为它的运行时间是强多项式的，隐含的常数因子很低，这使得它在实践中更有效率。具体来说，其复杂度为 $O(nmx+x^2\lg(x))$，其中 $x=\min(n,m)$。

4 总结与展望

随着存储技术的发展，像 Spark 和 Hadoop 等大数据平台的现代计算机集群越来越异构，当今应用程序的资源需求也变得异构和动态，在机器学习、数据分析、图分析等方面的现代应用程序，在应用程序的生命周期中可能有不同的资源需求。然而，大数据平台通常忽略应用程序的动态资源需求和底层的异构性，这种高级软件平台和底层引见之间的根本不匹配导致性能下降，以及由于无法有效利用不同的资源而造成的资源浪费。

存储异构的出现为集群计算中的任务调度问题引入了一个新的维度。具体来说，对于任务调度程序来说，在做决策时同时考虑访问数据的位置和存储层是很重要的，以便提高应用程序性能和集群利用率。本文研究了近些年的几种任务调度算法，对每个算法的实现原理，实现方法进行的详细的分析，对分级存储中任务调度问题有了更深入的了解。

通过对算法的研究，发现上述算法，通过充分考虑和利用异构存储的优势，使得大数据平台在处理作业任务调度时的效率有了明显的提升，然而通过文献搜索，发现这方面的研究还比较少，仅有一些少数的算法，所以后续研究可以沿着这个方向发展，希望可以研究出一个性能更好的算法，提高平台的任务调度，作业处理效率。

5 参考文献

[1] Xu L, Butt A R, Lim S H, et al. A heterogeneity-aware task scheduler for spark[C]//2018 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, 2018: 245-256.

[2] Pan F, Xiong J, Shen Y, et al. H-scheduler: Storage-aware task scheduling for heterogeneous-storage spark clusters[C]//2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 2018: 1-9.

[3] Herodotou H, Kakoulli E. Trident: task scheduling over tiered storage systems in big data platforms[J]. Proceedings of the VLDB Endowment, 2021.