基于 ISE 算法的分布式 ETL 任务调度策略研究

王卓昊[1] 杨冬菊[2,3] 徐晨阳[1]
(中国科学技术信息研究所 北京 100038)[1]
(大规模流数据集成与分析技术北京市重点实验室 北京 100144)[2]
(北方工业大学数据工程研究院 北京 100144)[3]

摘 要 随着数据仓库的规模不断扩大,数据集成下的 ETL(Extraction-Transformation-Loading)任务也随之增多,单机调度显然已经不能满足当下繁多复杂的 ETL 任务调度。针对 ETL 任务调度如何提高效率、缩短关键任务等待时间、提升资源利用率等问题,构建了一套分布式 ETL 任务调度框架,该框架由调度器和若干执行器组成,通过任务预处理、任务调度分配、任务执行 3 个阶段来完成 ETL 任务调度。在任务预处理阶段,对 ETL 任务建立权重模型,并根据权重确定调度优先级。在任务调度分配阶段,调度器根据各个执行器节点的性能及负载情况来约束执行器节点的选择,并设计贪心平衡(Greedy Balance,GB)算法来进行 ETL 任务执行请求的分发,使执行器节点的负载相对均衡。在任务执行阶段,通过高响应比优先(Highest Response Ratio Next,HRRN)算法确定执行器节点队列下任务的执行优先级。实验结果表明,分布式 ETL 任务调度框架及相应的一体化调度执行(Integrated Scheduling Execution,ISE)算法能够有效提高集群资源的利用率,缩短任务调度的执行时间。

关键词 任务调度,负载均衡,动态分配,分布式集群,ETL,数据集成
中图法分类号 TP301 **文献标识码** A **DOI** 10.11896/jsjkx.190100023

Research on Distributed ETL Tasks Scheduling Strategy Based on ISE Algorithm

WANG Zhuo-hao[1] YANG Dong-ju[2,3] XU Chen-yang[1]
(Institute of Scientific and Technical Information of China, Beijing 100038, China)[1]
(Beijing Key Laboratory on Integration and Analysis of Large-scale Stream Data, Beijing 100144, China)[2]
(Data Engineering Institute, North China University of Technology, Beijing 100144, China)[3]

Abstract With the expansion of the data warehouse, ETL tasks have also increased under data integration. Stand-alone scheduling obviously cannot meet the needs of many complex ETL tasks. Aiming at how to improve the efficiency of ETL task scheduling, reduce the critical task waiting time, and improve the resource utilization and so on, this paper constructed a distributed ETL task scheduling framework consisting of a scheduler and several actuators and completing the ETL task scheduling through the task preprocessing, task scheduling and task execution. In the task preprocessing stage, a weight model is established for the ETL task, and the scheduling priority is determined according to the weight. In the task scheduling stage, the scheduler constrains the choice of actuator nodes according to the performance and load conditions of each actuator node, and a greedy balance (GB) algorithm is designed to distribute the ETL task execution requests, so that the load of the actuator nodes is relatively balanced. In the task execution phase, the execution priority of tasks under the actuator node queue is determined by the high response ratio first (Highest Response Ratio Next, HRRN) algorithm. Experiment results show that the distributed ETL task scheduling framework and the corresponding integrated scheduling execution (ISE) algorithm can effectively improve the utilization of cluster resources and shorten the task scheduling execution time.

Keywords Task scheduling, Load balancing, Dynamic allocation, Distributed clustering, Extraction-Transformation-Loading, Data integration

1 前言

ETL 技术是大数据环境下构建数据仓库的核心技术之一,是将分散、异构的数据经过抽取、转换再加载到统一标准库的过程。抽取、转换、加载可以组合成一个可调度的 ETL 作业。如何高效率地调度这些任务作业也是构建数

到稿日期:2019-01-24 返修日期:2019-04-21 本文受国家自然科学基金重点项目(61832004)资助。
王卓昊(1977-),男,博士,副研究员,主要研究方向为计算机应用、云计算与大数据应用等,E-mail:wangzh@most.cn;**杨冬菊**(1975-),女,博士,副研究员,主要研究方向为服务计算、数据集成与分析等,E-mail:yangdongju@ncut.edu.cn(通信作者);**徐晨阳**(1992-),男,硕士,主要研究方向为数据集成、数据分析等。

据仓库的重要组成部分。

以科技资源管理数据集成业务为例，业务包括预申报、申报、任务书业务下的项目、课题、单位、人员等异构数据向统一标准库的集成，并采用了开源工具 Kettle 进行数据集成。ETL 作业任务的表现形式是类型为 ktr，kjb 的脚本文件，任务以元数据[1]的形式存储在 ETL 任务资源库。针对该集成业务下数据量大、执行频率高、业务复杂等特点，已经采用了 ETL 集群调度方案来进行 ETL 作业调度，但目前仍存在以下问题：1)业务复杂，待集成数据的重要性不同，调度时没有考虑 ETL 任务的等级划分，导致与核心业务相关的 ETL 任务的等待时间过长；2)在为执行节点分配任务执行请求时，采用了轮询算法，依照执行节点的顺序依次进行任务调度分配，没有考虑由于任务执行时间不同带来的执行节点负载不同的问题，从而造成集群资源不均衡；3)在执行器节点的执行队列上，使用先来先服务算法(First Come First Served，FCFS)执行 ETL 任务，这种静态、不可调整的执行顺序造成了 ETL 任务的等待时间和执行时间的不平衡，延长了整个任务的执行周期，因此需要对 ETL 任务的执行优先级进行动态、可调整的划分。

针对上述问题，本文在集群调度的基础上提出了一个分布式 ETL 任务调度框架。该框架分为任务预处理、任务调度分配、任务执行 3 个阶段，分别从任务权重、节点负载以及 ETL 任务执行优先级调整方面对 ETL 任务进行一体化调度，以提升 ETL 任务的调度和执行效率。

2 相关工作

任务调度包括单机调度、分布式任务调度等，同时有多种调度算法。随着大数据时代的到来，Saleh 等[2-3]研究了云计算和雾计算环境下大规模数据的任务调度。在分布式调度环境[4]的基础上，如何提升环境资源利用率[5]和降低资源消耗[6]也成为了当下的研究热点。Kokilavani 等[7]提出了基于 Min-Min 算法的任务调度方案，通过算法的优化来提高分布式平台的调度服务质量。其研究重点是如何合理、有效地将任务分配到执行节点，对节点的性能差异没有太多的考虑，因此常常造成单节点负载过高，而有的节点空闲的情况。面对业务数据加载快、更新及时性高的特点，Mallet 等[8]研究了从逻辑时间安排到实时调度的工作，保障了任务被调度执行的及时性。Zhang 等[9]和 Ge 等[10]研究了基于节点负载均衡的任务调度方案，但上述研究未考虑任务等级与任务执行阶段的问题。此外，为了适应日益多样化和复杂的任务调度要求，以及尽可能缩短任务调度的执行周期，Yu 等[11]和 Sundar 等[12]提出了任务调度机制，研究了任务调度中完成时间的最小化，其具有较好的扩展易用性和负载均衡保障。因此，在分布式环境下，受节点性能差异的约束，如何动态地分配任务和利用有效的算法来缩短任务调度的执行周期、优化调度策略是分布式调度的难点。

3 分布式 ETL 任务调度框架

在大数据的发展中，核心业务数据集成的效率变得越来越重要，数据集成下 ETL 任务调度是决定集成效率的重要因素之一。针对当前 ETL 任务调度机制和算法存在的问题，本文提出了一个分布式 ETL 任务调度框架和一体化调度执行算法(Itegrated Scheduling Execution，ISE)，框架如图 1 所示。该框架分为 ETL 任务预处理、ETL 任务调度分配、ETL 任务执行 3 个阶段。其中，调度器负责 ETL 任务预处理和 ETL 任务调度分配，执行器负责 ETL 任务执行，调度器和执行器相互分离。由于业务数据繁多和数据加载及时性的特点，我们采取一种频率为每小时一次的定期获取 ETL 任务的策略，调度器会定时整点从 ETL 任务资源库读取任务。

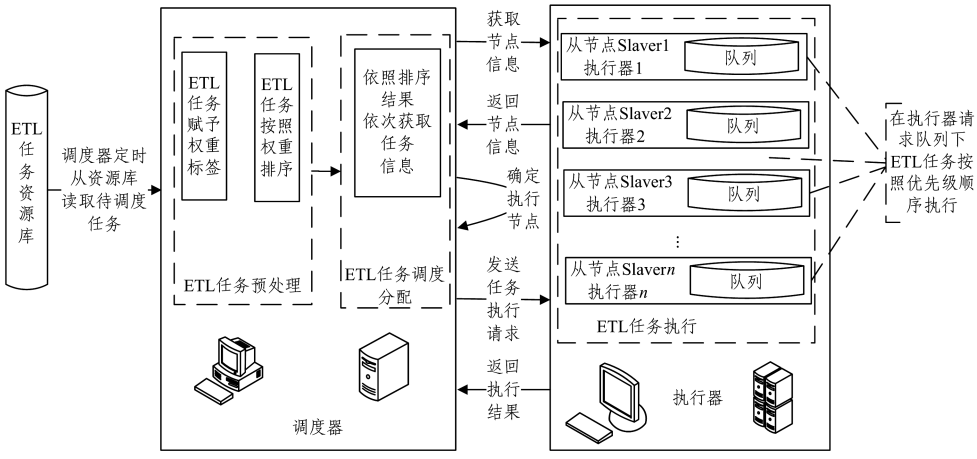

图 1 分布式 ETL 任务的调度框架

Fig. 1 Distributed ETL task scheduling framework

在 ETL 任务预处理阶段，调度器获取任务后，首先设计 ETL 任务的权重模型，在构建 ETL 任务时，根据数据要加载的业务表权重等级赋予任务权重标签，根据权重将 ETL 任务进行排序以供调度，以此来满足与核心业务数据相关的 ETL 任务能够先得到调度执行。

在 ETL 任务调度分配阶段，新一批任务经过 ETL 任务预处理以后，调度器首先获取各执行器的 CPU、内存使用率来判断各个节点执行器的当前性能，计算各个节点的活跃性，以确定可继续分配任务的执行器节点，从而实现一种动态的、各节点之间负载均衡的任务分配策略。确定可分配任务执行请求的执行节点以后，按照贪心平衡算法分发 ETL 任务执行请求。

在 ETL 任务执行阶段，任务执行请求分发到执行器节点以后，在每个执行器的任务请求执行队列下，采用高响应比优先算法来动态调整 ETL 任务的执行优先级，并按照队列内的

执行优先级顺序进行执行。

4　一体化调度执行算法的设计

本文在分布式 ETL 任务调度框架的基础上提出了一体化调度执行算法，该算法是 3 个阶段下 ETL 预处理算法、ETL 任务调度分配算法、ETL 任务执行算法的总体概括。ISE 算法的工作流程如图 2 所示，ISE 算法的工作顺序如图 3 所示。ISE 算法的具体步骤如算法 1 所示。

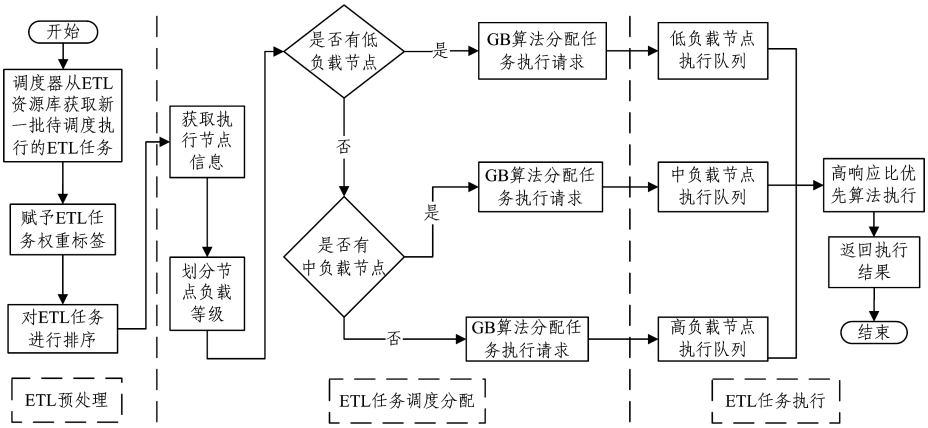

图 2　一体化调度执行算法的工作流程图

Fig. 2　Flowchart of ISE algorithm work flow

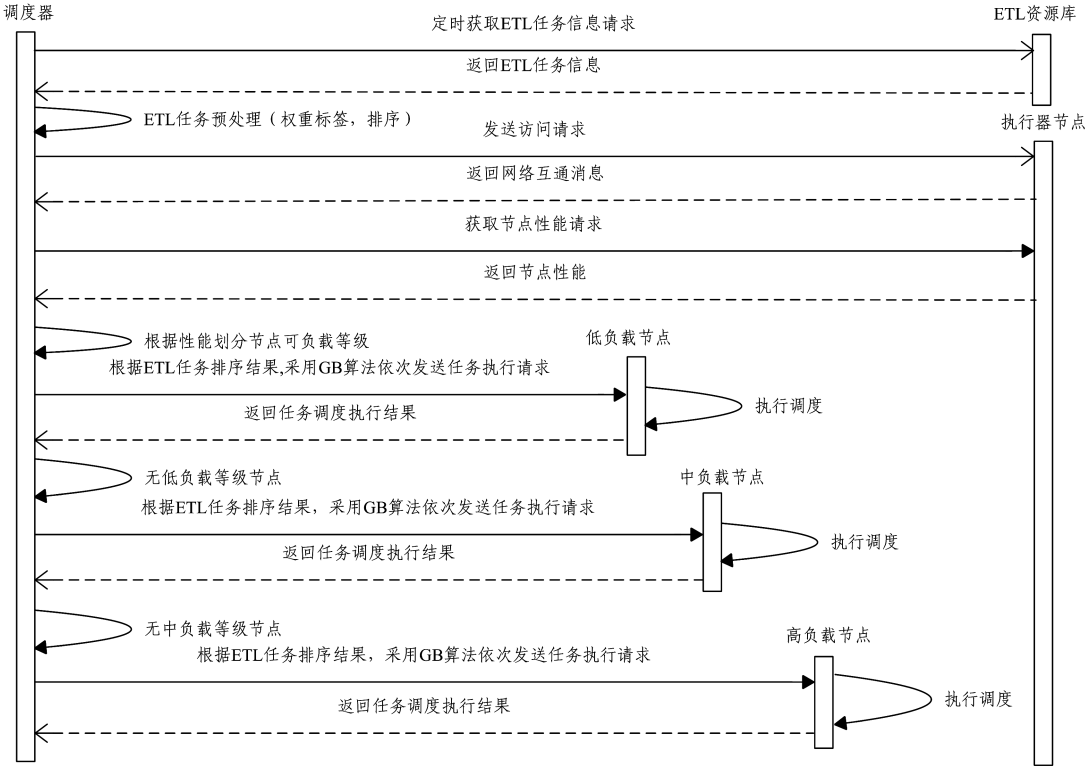

图 3　一体化调度执行算法的工作顺序图

Fig. 3　Flowchart of ISE algorithm work sequence

算法 1　ISE 算法

输入：ETL 资源库下新一批待调度执行的任务

输出：ETL 任务的调度执行结果

1. 初始化新一批任务信息。
2. 调度器获取 ETL 资源库下新一批任务调度请求。
3. 确定 ETL 任务权重等级，依照权重进行排序。
4. 调度器获取当前执行节点的性能信息，由节点活跃性划分节点负载等级。
5. 若低负载节点不为空，则依次将请求根据 GB 算法分配到低负载节点的执行器节点上；若低负载节点为空，则将请求根据 GB 算法分配到中负载节点的执行器节点上；若中负载节点为空，则将请求根据 GB 算法分配到高负载节点的执行器节点上。
6. 对于执行器节点的任务执行队列来说，根据高响应比优先算法来确定任务执行的优先级，并进行执行。
7. 将执行结果返回到调度器。

由于该算法为 3 个阶段算法的总体概括，因此该算法的时间复杂度取 3 个阶段算法的时间复杂度的最大值，即该算法的时间复杂度为 $O(N^2 \log N)$。

4.1　ETL 任务预处理算法

数据仓库下的数据集成包含很多 ETL 任务，如异构数据

的抽取、转换、加载,以及日志信息的抽取整合等。当前大数据环境下,由于不少企业对核心业务数据加载的效率要求非常高,因此需要在任务调度初期对 ETL 任务进行调度优先级的划分。因此,本文设计了权重模型来支撑统一标准库下数据的存储,在完善数据存储的同时对业务实体进行权重等级的划分;其次通过定义业务数据要加载的目标表的权重,对与目标表相关的 ETL 任务赋予权重标签,构建 ETL 任务的权重模型,从而通过权重对 ETL 任务的调度等级进行划分。本文以科技资源管理数据集成业务为例来对权重进行相关定义的描述:对于统一标准库中的业务表数据模型(见图 4),通过数学模型的描述方式对相关业务表进行权重划分,给出项目、课题、单位、人员 4 个主体,项目和课题之间是一对多的关系。实体和附属表之间的联系权重用 Wl_1 表示,实体和维度之间的联系权重用 Wl_2 表示,实体与实体之间的联系权重用 Wl_3 表示。

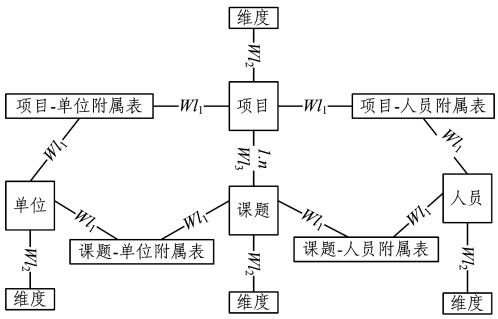

图 4 目标库带权数据存储模型

Fig. 4 Target library weighted data storage model

根据上述带权存储模型,相关业务实体的权重形式化表示为:

$$Weight(W)=\frac{n_1*Wl_1+n_2*Wl_2+n_3*Wl_3}{n_1+n_2+n_3} \tag{1}$$

其中,n_i 为 Wl_i 的个数。

对于联系权值,根据不同业务需求可以进行适当的调整。根据目标主体权重对相关 ETL 任务赋予权重标签后,需要对批量 ETL 进行权重等级的划分,本文在快速排序[13]的基础上采用稳定快速排序算法来对 ETL 任务进行排序,以达到任务序列依据权重排序的最优值。批量 ETL 任务的稳定快速排序算法如算法 2 所示。

算法 2 稳定快速排序算法

输入:ETL 任务,ETL 任务权重

输出:依照权重排序的 ETL 任务序列

1. 获取批量 ETL 任务的权重,将其作为要进行排序的数据。
2. 将要排序的数据划分成独立的两部分。
3. 将每部分数据各自分成 2 个小块,使 1 个小块中的所有数据都比另 1 个小块中的所有数据小(或大)。
4. 把 4 块中 2 个小(或大)块连接起来,放在所有数据的前部,把另外 2 块也连接起来,放在所有数据的后部。
5. 通过步骤 3)、步骤 4)对形成的两部分数据分别进行稳定快速排序,直到所有数据都是有序的。

整个排序过程可以递归进行。根据算法 2,如果对 n 个数进行排序,最坏的情况需要移动 n 个数,假设花费的时间为 $F(n)$,对 n 个数排序进行一次划分的时间为 $T(n)$,则该算法的时间复杂度为:

$$\begin{aligned}C(n)&=F(n)+T(n)=F(n)+n+2T(n/2)=F(n)+n+\\&2[F(n/2)+n/2+2T(n/4)]=2F(n)+2n+\\&4T(n/4)=\cdots=kF(n)+kn+2kC(n/2k)=\cdots=\\&\log(n)*[F(n)+n]+nT(1)=log(n)*[F(n)+\\&n]=O(n\log(n))\end{aligned}$$

4.2 ETL 任务调度分配算法

本文中的任务调度框架是基于分布式环境的 ETL 任务调度框架,调度器和执行器处于分离状态,且执行器节点处于分布式的状态,当分布式节点运行以后,由于不同执行器节点运行的任务数和任务所含的数据量不同,因此同一时刻各个执行器节点的性能和当前负载是不同的,调度器需要根据执行器节点的性能来合理地分配任务数量,从而保证各个节点乃至整个集群环境的负载均衡。在任务预处理阶段,对于已经排序好的 ETL 任务,调度器依照排序的结果依次获取 ETL 任务的执行请求,其次向执行器节点发送访问请求。如果网络互通,执行器节点会返回网络互通消息,通过获取各个执行器节点的 CPU、内存(RAM)使用率来查看各个节点的资源使用情况,执行器节点返回各个节点的性能信息,调度器根据性能划分节点负载等级。该流程能够根据各节点当前的性能来对任务进行合理、动态的分配,以达到集群环境的负载均衡。本文通过定义各个执行器节点的活跃性来衡量节点当前是否能够再接受任务调度执行需求。各节点活跃性的公式如下:

$$Activity(A)=\frac{(1-C)+(1-R)}{2},0<C\leqslant 1,0<R\leqslant 1 \tag{2}$$

其中,C 为 CPU 使用率,R 为内存使用率。

根据各执行器节点活跃性 A 对其当前负载进行等级划分,活跃性低代表节点当前负载等级较高,以此类推,将执行器节点划分为高、中、低 3 个等级。节点当前的负载等级为:

$$L=A=\begin{cases}0\sim 35\% & \text{高}\\ 36\%\sim 75\% & \text{中}\\ 76\%\sim 100\% & \text{低}\end{cases} \tag{3}$$

本文根据活跃性对执行器节点进行负载等级的分类,划分出高负载节点、中负载节点、低负载节点,即将集群下的节点环境分割成 3 个小组,每个组由零至多个节点组成,同组的节点成员的负载量类似。低负载节点中的执行器节点的负载量最低,当前可再接受任务的执行能力最强,因此应当优先考虑将任务调度请求分配到低负载执行器节点上。若低负载节点为空,则分配任务到中负载节点,以此类推。若低、中负载节点都为空,即整个执行器节点当前的负载量都很高,则说明当前任务调度执行的效率会减弱。一旦整体的执行器节点被长时间划分到高负载节点等级下,则该集群环境的性能需要进行提升或增加相应执行器节点的数量,以此来改善整个分布式集群的性能。贪心算法[14-15]是指所求问题的整体最优解可以通过一系列局部最优的选择,即贪心选择来达到。这

是贪心算法可行的第一个基本要素。贪心选择自顶向下,以迭代的方法相继做出选择,每进行一次贪心选择就将所求问题简化为一个规模更小的子问题,即从局部最优达到整体最优。本文在贪心算法的基础上,引入任务数据量,增设最优期待分配任务数据量,提出了贪心平衡算法,通过数学模型的方法来对 GB 算法的分配任务过程进行形式化描述。首先假设分布式集群下各个节点的初始处理能力相同,且每个节点都可以独立工作,即不需要其他节点的辅助,并屏蔽外在环境对服务器节点的影响(如温度等)。相关定义如下:设 $E=\{e_1,e_2,e_3,\cdots,e_n\}$ 表示新一批 n 个相互独立参与调度的 ETL 任务集合,其中 e_i 为任务 i;$D=\{d_1,d_2,d_3,\cdots,d_n\}$ 表示 n 个 ETL 任务所含数据量的集合,其中 d_i 为任务 i 所含的数据量;$N=\{n_1,n_2,n_3,\cdots,n_j\}$ 表示分布式集群执行器节点的集合,共 j 个节点,其中 n_i 为执行器节点 i,d_{n_i} 表示在执行器节点 i 上分配的 ETL 任务所含的数据量。当任务分配完毕时,参与执行的执行器节点上的 ETL 任务所含的数据总量为:$Data=\sum_{i=0}^{j}dn_i$。每个执行器节点的最优期待分配任务数据量 Opt 为:

$$Optimum(Opt)=\frac{Data}{j}=\frac{\sum_{i=0}^{j}d_{n_i}}{j} \tag{4}$$

用任务所含数据量的方差表示执行器节点之间的负载均衡指数 μ:

$$\mu=\frac{\sum_{i=0}^{j}(d_{n_i}-Opt)^2}{j} \tag{5}$$

ETL 任务在调度执行时的目标函数 $TotalTime$ 表示为:

$$TotalTime=\sum_{i=0}^{j}T_i+\sum_{i=0}^{j}Te_i+\sum_{i=0}^{j}Tw_i \tag{6}$$

其中,T_i 表示主节点调度器映射执行器节点 i 的时间(包含网络互通请求和获取性能);Te_i 表示 ETL 任务 i 的执行时间(根据所含数据量进行评估);Tw_i 表示 ETL 任务 i 等待执行的时间。

在获取 ETL 任务调度请求到将任务分配到执行器节点进行调度执行的过程中,应尽可能保证整个调度流程所耗费的时间最短(即 $TotalTime$ 最小)、集群资源负载均衡。首先调度器与执行器节点之间的映射步骤必不可少,即 T_i 的时间相对固定,重点是考虑如何将任务有效均衡地分配到执行器节点,以及 Te_i 和 Tw_i 的时间尽可能达到相对平衡。GB 算法分配 ETL 任务的具体步骤如下:

(1)首先初始化 ETL 任务集合 $E=\{e_1,e_2,e_3,\cdots,e_n\}$、ETL 任务所含数据量集合 $D=\{d_1,d_2,d_3,\cdots,d_n\}$、分布式集群某负载等级执行器节点集合 $N=\{n_1,n_2,n_3,\cdots,n_j\}$。

(2)对 ETL 任务集合 E 按照数据量从大到小进行排序,并将结果存入到队列 Q 中。$Q=\{q(e_i,d_i),\cdots,q(e_k,d_k)\}$,其中 $d_i\geqslant d_k(1\leqslant i,k\leqslant n)$。

(3)遍历队列 Q,若不为空,则按照排序取出数据量最大的任务 $q(e_i,d_i)$,同时计算各个执行器节点所含任务的数据量,得到含任务数据量最小的执行器节点 i,$1\leqslant i\leqslant n$。

(4)分配 $q(e_i,d_i)$ 到节点 i 进行执行,直到所有可分配任务的节点都经历首次任务分配。

(5)遍历队列 Q,若不为空,则结合当前执行器节点所分配数据量排列节点集合 N,找到第一个满足分配的任务数据量小于每个执行器节点最优期待的任务所含数据量 $\frac{Data}{j}$ 的执行器节点,将 $q(e_k,d_k)$ 加入到该执行器节点的等待队列下,如果不存在这样的执行器节点,则返回执行步骤(3)、步骤(4),直至队列 Q 为空。

根据算法描述,如果将 n 个任务分配到 j 个节点上($n>j$),首先需要对 n 个任务进行遍历,取最大数据量的任务;其次对 j 个节点进行遍历,取所含数据量最小的节点进行任务分发;然后再次遍历节点,取所分配数据量小于最优期待分配数据量的节点,进行剩余任务的分发。那么该算法可描述为一个外层循环中嵌套 3 个并行的循环,该算法的时间复杂度为 $O(N^2)$。

4.3　ETL 任务执行算法

待任务分配到执行器节点后,每个执行器节点都有一个执行队列负责存储任务,每个任务占用队列上的一个线程资源。对于任务执行的优先级来说,先前的研究已经有短作业优先、先来先服务等算法来约束;对于任务量(执行时间)大小不一的 ETL 任务来说,这两种算法都存在着资源利用率不高的情况,因此需要对每个执行器节点上的任务执行的优先级进行动态调整[16]。在研究任务执行优先级算法的基础上,本文采用了高响应比优先算法(HRRN)来规划任务执行的优先级,该算法能够有效地结合上述两种算法的优势,使每个任务都能够较为公平地获取线程资源并执行调度。设任务 i 的优先级为 EP_i,任务 i 的执行时间为 Te_i,任务 i 的等待时间为 Tw_i,任务 i 的触发时间为 Tt_i,任务 i 的开始执行时间为 Ts_i,任务 i 的执行结束时间为 Tf_i,则有:

$$EP_i=\frac{Te_i+Tw_i}{Te_i}=1+\frac{Tw_i}{Te_i} \tag{7}$$

其中,$Te_i=Tf_i-Ts_i$,$Tw_i=Ts_i-Tt_i$。

由式(7)可以看出:EP 一定是大于 1 的,当 Tw_i 一定时,Te_i 越小,优先级 EP 越高,类似短作业优先算法;当 Te_i 一定时,Tw_i 越大,优先级 EP 越高,类似先来先服务算法;当 Tw_i 和 Te_i 都处于不可定的状态时,该算法也能够有效地动态调整任务优先级,使得 ETL 任务的执行时间和等待时间整体上相对平衡。根据算法描述,首先遍历当前作业列表,在一个作业运行期间内,如果有其他作业到达,将它们按照优先级降序排列,则该算法可描述为一个外层循环中嵌套一个循环和一次排序,则该算法的时间复杂度为 $O(N^2\log N)$。

HRRN 算法动态调整执行器节点队列中任务执行优先级的流程图如图 5 所示。将批量 ETL 任务调度请求缓存到执行器节点缓存队列,如果线程资源大于 ETL 任务数,则所有 ETL 任务均可同时执行,如果线程资源无法满足调度请求时,则启动 HRRN 算法对 ETL 任务执行优先级划分,确立执行任务以后获取线程资源,分发任务调度请求去执行,任务执行完成后返回信息并同时释放线程资源。

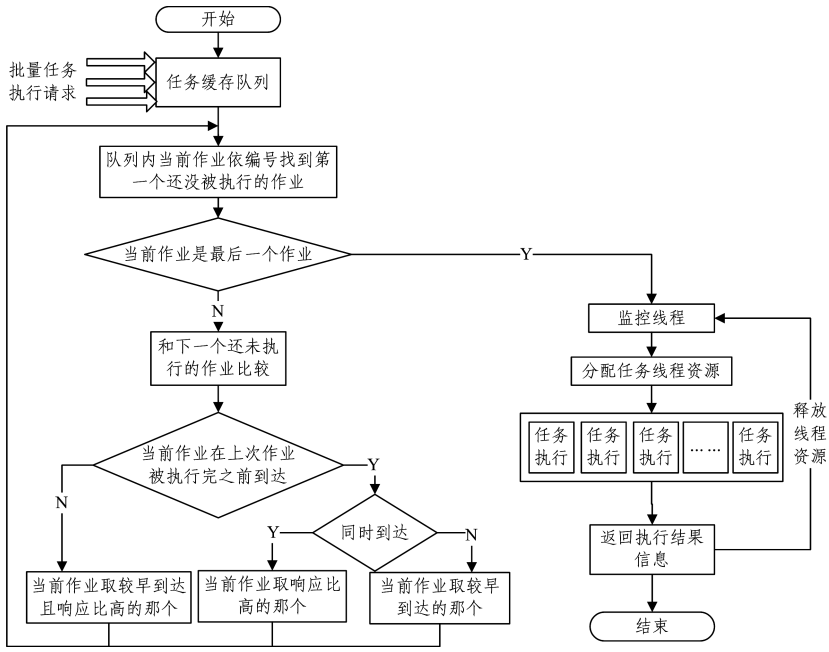

图 5 HRRN 算法辅助的队列下任务执行流程图

Fig. 5 Task execution flow chart under queue assisted by HRRN algorithm

5 实例验证分析

5.1 实验环境

为了测试本文调度策略的性能,我们构建了一套分布式集群环境,包括 1 台主节点(调度器)和 3 台从节点(执行器),其中主节点负责任务的分发,执行节点上分别部署搭建执行引擎来负责执行任务。

上述分布式系统的具体配置如表 1 所列。为了验证本文 ISE 算法调度策略的性能,将本文的调度执行策略与近几年的 ETL 任务调度方案进行对比。在相同的条件下,将所提策略的调度执行结果与在 ETL 任务调度分配阶段分别采用 Min-Min 算法、轮询算法以及在 ETL 任务执行阶段采用先来先服务(FCFS)算法进行的调度执行结果进行对比实验分析。

表 1 分布式环境配置信息

Table 1 Information of distributed environment

节点类型	CPU/GHz	内存 RAM/GB	操作系统	网络传输速率/(M/s)
主节点(调度器)	8 核 Intel3.40	8	Windows	500
从节点 1(执行器 1)	4 核 Intel2.60	4	Windows	100
从节点 2(执行器 2)	4 核 Intel2.60	4	Windows	100
从节点 3(执行器 3)	4 核 Intel2.60	4	Windows	100

5.2 实验结果分析

本文主要从 ETL 任务调度的执行时间上对 3 种调度执行策略进行对比分析,并从分布式系统资源平均利用率和执行器节点之间的负载是否均衡这两个方面进行验证分析。本次实验中,定义 ETL 任务调度的执行时间为:在多个任务的情况下,调度器从 ETL 资源库获取待调度任务到所有任务执行完成的整个持续时间。ETL 任务调度执行时间的结果对比如图 6 所示。本文在相同测试环境下,针对不同任务数量(10,50,100,300)对 3 种调度执行策略分别进行测试,具体实验结果如表 2 所列。

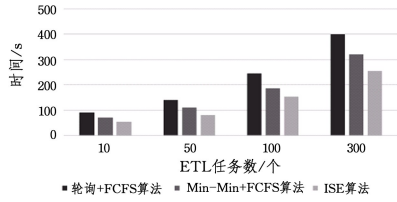

图 6 任务调度执行时间对比图

Fig. 6 Comparison chart of task scheduling execution time

表 2 ETL 任务调度执行时间的结果数据

Table 2 Results of ETL task scheduling execution time

(单位:s)

任务数	调度方案		
	轮询+FCFS	Min-Min+FCFS	ISE
10	90	70	55
50	140	110	80
100	245	185	155
300	400	320	255

由图 6 和表 2 可以得出,在不同 ETL 任务数量的情况下,本文提出的 ISE 算法调度策略虽然增加了 ETL 预处理阶段,但耗费的时间均比轮询+FCFS 算法调度执行和 Min-Min+FCFS 算法调度执行的时间短,且随着任务数量的增加,本文提出的 ISE 算法调度执行所消耗时间的增长速度较为缓慢。

当 ETL 任务数为 100 时,采用轮询+FCFS 算法和 Min-Min+FCFS 算法以及一体化算法的 ETL 任务调度执行时间分别为 245s,185s,155s。实验中统计了 3 种调度执行算法策略在时间节点 45s,85s,110s,150s 上的资源利用率,进而计算出分布式系统资源的平均利用率,如图 7 所示。轮询+FCFS 调度执行方案的 CPU、RAM 平均利用率为 59% 和 71%;Min-Min+FCFS 调度执行方案的 CPU、RAM 平均利用率为 49% 和 59%;ISE 算法调度执行方案的 CPU、RAM 平

均利用率为 41%和 52%。

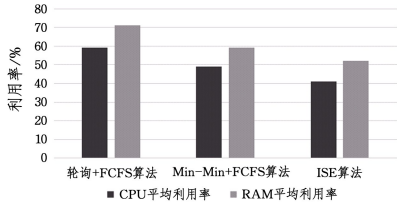

图 7　各节点 CPU、内存利用率对比图

Fig. 7　Comparison of CPU and memory usage of each node

由图 7 可以得出，一体化算法调度执行策略在各个节点上的 CPU、内存利用率均比其他两种调度执行策略小，也就是说一体化算法调度执行策略能够充分利用节点资源。

在负载均衡方面，当 ETL 任务数为 300 时，采用一体化算法的 ETL 任务调度执行时间为 255 s。3 个节点在不同时间节点(1 min，2 min，3 min，4 min)上的 CPU、内存使用率的对比信息如图 8、图 9 所示。可以看出，3 个执行节点的 CPU 利用率在 51%～56%之间波动，内存利用率在 62%～68%之间波动，执行节点之间的负载相对均衡，从而有效地验证了本文 ISE 算法调度执行策略的优势。

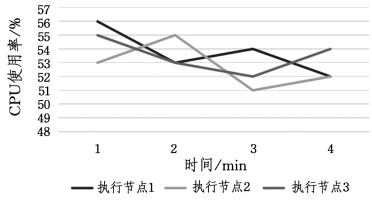

图 8　各执行节点的 CPU 负载情况

Fig. 8　CPU load of each execution node

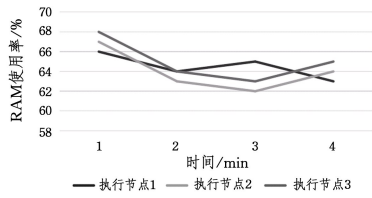

图 9　各执行节点的 RAM 负载情况

Fig. 9　RAM load of each execution node

结束语　为了解决核心业务数据集成效率缓慢以及当前分布式环境下任务调度节点资源浪费、负载不均衡的问题，本文构建了一个分布式 ETL 任务调度框架，并提出了一体化调度执行算法。该框架和算法能够有效提高分布式环境资源利用率，提高任务调度执行效率，为分布式任务调度方案提供了参考价值。同时本文研究仍有不足之处：1)在节点性能的衡量指标上只考虑了 CPU 和内存的使用率，还有更多的性能判断指标；2)ETL 执行涉及到数据量、任务复杂度、网络等多个不确定因素，本文只通过任务所含数据量来判断任务执行时间，并仅以此作为衡量负载均衡的指标。因此，考虑更多的衡量指标来完善分布式多集群 ETL 任务的调度策略是下一步研究的重点。

参 考 文 献

[1] ZHANG L. Integration and collection of heterogeneous data based on metedata[C]// 2013 6th International Conference on Information Management, Innovation Management and Industrial Engineering. Xi'an, 2013: 205-208.

[2] SALEH H, NASHAAT H, SABER W, et al. IPSO Task Scheduling Algorithm for Large Scale Data in Cloud Computing Environment[J]. IEEE Access, 2019, 7(1): 5412-5420.

[3] ISLAM T, HASHEM M M A. Task Scheduling for Big Data Management in Fog Infrastructure[C]// 2018 21st International Conference of Computer and Information Technology (ICCIT). IEEE, 2018: 1-6.

[4] SAHAR M, VAHID R. A hybrid heuristic workflow scheduling algorithm for cloud computing environments[J]. Journal of Experimental & Theoretical Artificial Intelligence, 2015, 27(6): 1-15.

[5] YAO Y, GAO H, WANG J, et al. New Scheduling Algorithms for Improving Performance and Resource Utilization in Hadoop YARN Clusters[J]. IEEE Transactions on Cloud Computing, 2019, 7(1): 1-1.

[6] SUN J, CHO H, EASWARAN A, et al. Flow Network-Based Real-Time Scheduling for Reducing Static Energy Consumption on Multiprocessors[J]. IEEE Access, 2019, 7(1): 1330-1344.

[7] KOKILAVANI T, GEORGE D I, THINAM A. Load Balanced MinMin Algorithm for Static MetaTask Scheduling in Grid Computing[J]. International Journal of Computer Applications, 2011, 20(2): 43-49.

[8] MALLET F, ZHANG M. Work-in-Progress: From Logical Time Scheduling to Real-Time Scheduling[C]// 2018 IEEE Real-Time Systems Symposium (RTSS). IEEE, 2018: 143-146.

[9] ZHANG L, LIU S F, HAN L. Task scheduling algorithm based on load balancing [J]. Journal of Jilin University (Science Edition), 2014(4): 769-772.

[10] GE W C, YE B. Improved priority table scheduling algorithm based on load balancing priority[J]. Journal of Shenyang University of Technology, 2017, 39(3): 241-247.

[11] YU W, LIU F, XIONG Z, et al. A Task Scheduling Mechanism Based on Quartz of Power Consumption Information Acquisition System[C]// 2018 5th International Conference on Information Science and Control Engineering (ICISCE). IEEE, 2018: 98-101.

[12] SUNDAR S, CHAMPATI J P, LIANG B. Completion Time Minimization in Multi-user Task Scheduling with Heterogeneous Processors and Budget Constraints[C]// 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS). IEEE, 2018: 1-6.

[13] MAHMOUD R, UWE R. The Quicksort process[J]. Stochastic Processes and Their Applications: An Official Journal of the Bernoulli Society for Mathematical Statistics and Probability, 2014, 124(2): 1036-1054.

[14] XIA H. Load balancing greedy algorithm for reduce on Hadoop platform[C]// 2018 IEEE 3rd International Conference on Big Data Analysis (ICBDA). IEEE, 2018: 212-216.

[15] WANG C, MAO Y, HU B, et al. Ship Block Transportation Scheduling Problem Based on Greedy Algorithm[J]. Journal of Engineering Science & Technology Review, 2016, 9(2): 93-98.

[16] LI J M, WANG X, WU Y X. An Improved Priority List Task Scheduling Algorithm[J]. Computer Science, 2014, 4(5): 20-23, 36.