第41卷 第6A期
2014年6月

计 算 机 科 学
Computer Science

Vol. 41 No. 6A
Jun 2014

一种基于DAG的MapReduce任务调度算法

唐一韬[1] 黄 晶[2] 肖 球[2]

(湖南长沙民政职业技术学院 长沙 410004)[1] (湖南大学信息科学与工程学院 长沙 410082)[2]

摘 要 Hadoop已成为研究云计算的基础平台,MapReduce是其大数据分布式处理的计算模型。针对异构集群下MapReduce数据分布、数据本地性、作业执行流程等问题,提出一种基于DAG的MapReduce调度算法。把集群中的节点按计算能力进行划分,将MapReduce作业转换成DAG模型,改进向上排序值计算方法,使其在异构集群中计算更精准、任务的优先级排序更合理。综合节点的计算能力与数据本地性及集群利用情况,选择合理的数据节点分配和执行任务,减少当前任务完成时间。实验表明,该算法能合理分布数据,有效提高数据本地性,减少通信开销,缩短整个作业集的调度长度,从而提高集群的利用率。

关键词 DAG,调度算法,MapReduce,Hadoop,异构环境,大数据

中图法分类号 TP302 **文献标识码** A

Task Scheduling Algorithm for MapReduce Based on DAG

TANG Yi-tao[1] HUANG Jing[2] XIAO Qiu[2]

(Software Engineering, Changsha Social Work College, Changsha 410004, China)[1]

(School of Information Sciences and Engineering, Hunan University, Changsha 410082, China)[2]

Abstract Hadoop has been the basic platform of cloud computing research, and MapReduce is the computing mode for distributed processing of big data. For heterogeneous cluster, considering MapReduce's defects in data distribution, data locality and process of the job execution, we proposed a DAG based MapReduce scheduling algorithm. The algorithm groups the nodes based on their computing ability, transforms MapReduce job execution to DAG model and improves upward ranking to achieve better accuracy and a more reasonable sequencing of task priority. By combining the computing ability of nodes, data locality and cluster utilization, choosing the proper data nodes for task distribution and execution, our algorithm shortens task completion time. The experimental result shows that the proposed algorithm can distribute data reasonably, improve data locality effectively, reduce communication overhead, shorten schedule length of set of job, thus improving utilization of cluster.

Keywords DAG, Scheduling algorithm, MapReduce, Hadoop, Heterogeneous environment, Big data

1 引言

随着互联网技术的不断发展,越来越多的用户融入其中,互联网已经成为人们日常生活生产不可缺少的部分,如搜索引擎、社交网络、智能化社会等。这些应用的一个最大共同点就是面向大规模的数据处理。互联网数据爆炸式增长,使得需要处理的数据由GB级别不断增长,甚至达到PB级别。大数据处理是当今社会面临的一项重大难题。云计算作为一种大规模数据分析和处理的概念被提出。MapReduce[1]作为云计算的一种实例受到了广泛关注,同时也得到了广泛应用。MapReduce是一种编程模式,它能够监护运行在集群中的并行数据处理程序。MapReduce能抽象出并行处理的一般特征,即将数据处理的过程分为映射(Map)和规约(Reduce)两个过程。利用MapReduce不仅能处理大规模数据,而且能将很多繁琐的细节隐藏起来,这将极大简化程序员的开发工作。

Hadoop[2]是MapReduce一个开源实现版本。作为云计算的重要组成部分,它已经开始提供弹性的MapReduce作业集群服务。此外,它还实现了一个分布式的文件系统HDFS[3],具有很好的容错性。但是,Hadoop作为一个数据处理的基础平台,虽然被广泛应用但是还有很多不足的地方。近些年来,在Hadoop的性能方面,研发人员采取了很多措施来对其进行改进,包括磁盘I/O、网络带宽、数据延迟等。同时通过研究Hadoop自身的性能影响因素,如MapReduce作业执行流程的合理性、数据分布、数据本地性及任务调度算法等对其进行改进。然而,Hadoop仍然面临着很多性能挑战。其中很重要的一方面是Hadoop任务调度算法。Hadoop任务调度算法的主要功能是负责调度主节点将任务分派给计算节点,而每当计算节点空闲时便会向主节点请求任务。Ha-

本文受2011年湖南省"十二五"规划课题:基于立体化教学环境的创新型IT人才模式研究与实践,2012年国家教育部资助国内青年骨干教师访问学者科研业务费资助。

唐一韬(1977—),男,副教授,主要研究方向为构件技术,E-mail:20707085@qq.com;黄 晶(1986—),男,博士生,主要研究方向为分布式计算、并行计算;肖 球(1987—),男,博士生,主要研究方向为并行计算。

doop 任务调度算法的研究主要内容包括 5 个方面：1) Hadoop 集群的多用户共享；2) 性能改进，如数据本地性、网络 I/O、数据分布；3) 调度模型的理论化；4) 任务的预测执行和容错调度等；5) 实时调度，Hadoop 原来作为离线数据处理平台的实时性是比较差的，关于其任务调度实时性的研究也由于 Hadoop 的广泛应用而逐渐兴起。

2 相关研究

目前关于 MapReduce 任务调度算法的关注点多体现在数据本地性[4]、调度公平性、容错性等方面。数据本地性是指任务计算所需要的数据与负责计算该任务的处理器在同一节点上，它是 Hadoop 中任务能否在有限时间内完成的重要影响因子。文献[5]提出在延迟调度算法中强调数据的本地性，但该方法在公平性方面明显不足，不适合执行大作业或每个节点只有较少 slot 的任务。文献[6]中作者提出在应用与节点间建立关系来有效地放置数据从而提高数据的本地性。有学者提出 NKS 算法[7]来提高 map 任务的数据本地性，但此算法更适合针对同构的集群环境。

文献[8]在已知数据节点分布的情况下提出基于节点与任务优先级的调度算法。考虑集群的异构性，文献[9]提出了 LATE(Longest Approximate Time to End)调度算法，该算法通过预算待执行与正在执行任务的剩余时间并将执行最慢的任务作为备份任务，来减少 MapReduce 作业在集群中的执行时间。文献[10]中提出预测任务的剩余时间，以根据剩余时间动态预测任务执行所需资源。文献[11-13]提出基于截止期限的 MapReduce 调度算法。文献[11]中，作者根据任务的数据处理单元时间与数据传输单元时间建立一个任务执行模型，利用该模型来计算满足任务截止期限所需要的 Map 任务与 Reduce 任务数量，但该算法只适用于同构集群。文献[12]提出的算法可动态预测当前 MapReduce 任务的性能并根据其性能来分配任务所需的资源。文献[13]考虑数据的本地性与集群异构的特点提出了基于截止期限的 MTSD 调度算法，该算法将节点按计算能力划分成不同级别，计算能力越强的节点存储的数据越多，从而提高 Map 任务的数据本地性，并基于节点分类算法提出一个新的任务剩余执行时间估算模型。文献[14]将一个 MapReduce 任务看成一个 DAG(有向无环图)，把数据处理任务当工作流处理，该文献利用模糊时间戳来探测哪个数据集需被更新，再计算出一个任务序列来决定采用哪个任务来完成数据的处理。

DAG 任务调度模型已被广泛用于各类调度系统如汽车电子系统中网络离线任务调度。较有名的 DAG 调度算法有 CPOP[15]和 HEFT[15]等，HEHT 提出了向上排序值的概念，其计算公式为：

$$rank_u(n_i)=\overline{w_i}+\max_{n_j\in succ(n_i)}\{c_{i,j}+rank_u(n_j)\} \quad (1)$$

式中，$\overline{w_i}$为任务 n_i 在各个处理器上的平均执行时间，$C_{i,j}$ 为任务 n_i 到任务 n_j 处理器间的通信开销。向上排序值已成为事实的任务优先级排序标准而被多种 DAG 任务调度算法所采用。但采用平均值作为计算开销的向上排序值对于异构集群而言其计算结果还不够精准。

本文针对异构集群，提出一种基于 DAG 的 MapReduce 任务调度算法。将集群中的节点按计算能力进行划分，改进向上排序值的计算方法，使其在异构集群中的计算更精准，任务的优先级排序更为合理。综合节点的计算能力与数据本地性，选择合理的数据节点进行计算，有效减少通信开销与作业调度长度。

3 基于 DAG 的 MapReduce 调度算法

3.1 MapReduce 计算框架与 DAG 转换

MapReduce 是一个分布式计算框架，其特点是使用抽象的编程模型，能有效隐藏分布式系统底层实现细节，用户只需提供 Map 函数与 Reduce 函数就可写出分布式程序。最基本的处理步骤是将处理过程分为 Map 和 Reduce 两个阶段。Map 函数根据输入数据产生若干个〈key，value〉的键值对，而 Reduce 函数则把若干个〈key，value〉键值对其键值相同的进行合并，形成最终结果并以〈key，value〉的形式输出，如图 1 所示。

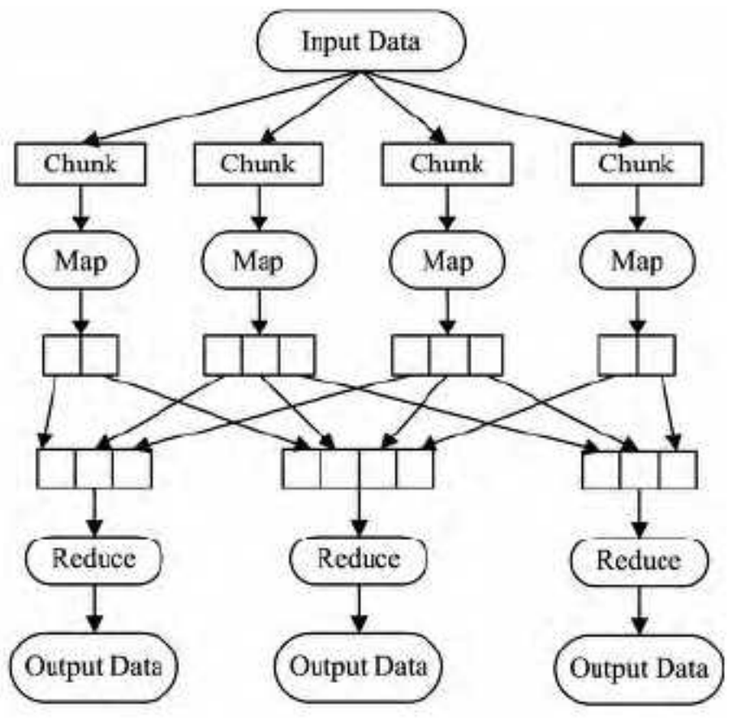

图 1 MapReduce 数据处理过程

图 1 展示了基本的 MapReduce 处理过程，但很多情况下 Reduce 阶段产生的结果并不是最终结果而是作为另外一个计算任务的输入，并开始另外一个计算任务。因此结构上一个 MapReduce 应用可由一个 DAG 来表示，$G=(V,E)$，其中 V 是任务 v 的集合，E 代表任务之间边 e 的集合。Map 程序与 Reduce 程序需要在集群数据节点上运行，每个数据节点可包含若干个 slot。本文将这些数据节点统称为节点，并用集合 $D=\{d_1,d_2,d_3\cdots,d_k\}$ 来表示。

云计算环境中可能存在多个用户同时提交作业或单个用户同时提交多个作业，每个作业都可表示为一个 DAG。为形式化地描述这些作业，本文用 $G=\{V,W,E,C\}$ 来表示一个 DAG。其中 $V=\{v_1,v_2,\cdots,v_k\}$ 表示任务的集合。由于集群的异构性，不同处理器具有不同的处理能力，W 是一个 $V*D$ 的矩阵，$w_{i,j}$ 表示任务 v_i 在处理器 d_j 上的计算开销。$E=\{e_{1,2},e_{2,3},\cdots,e_{i,j}\}$ 表示任务间的依赖关系。$C=\{c_{1,2},c_{2,3},\cdots,c_{i,j}\}$ 表示任务之间的通信开销，如果任务 v_i,v_j 分配在同一数据节点上处理，则 $c_{i,j}$ 为 0。一般 Map 任务是 Reduce 任务的直接前驱，相对 Reduce 任务是 Map 任务的后继，如果 Reduce 任务后还有 MapReduce 任务，则该 Reducer 任务是下一 Map 任务的前驱，迭代下去直到最后一个 Reduce 任务结束。没有前驱的任务叫做入口，记为 V_{entry}，没有后继的任务叫做出口，记为 V_{exit}。在 Map 任务开始之前是没有前驱的，而一个任务可能有多个 Map，为保证只有唯一入口，设 V_{entry} 在第一轮 Map 前执行，计算开销为 0。同理为保证只有唯一出口，设 V_{exit} 在最后一轮 Reduce 任务后执行，计算开销为 0，执行 V_{exit}

代表该 DAG 所有任务执行结束。转化结果如图 2 所示。

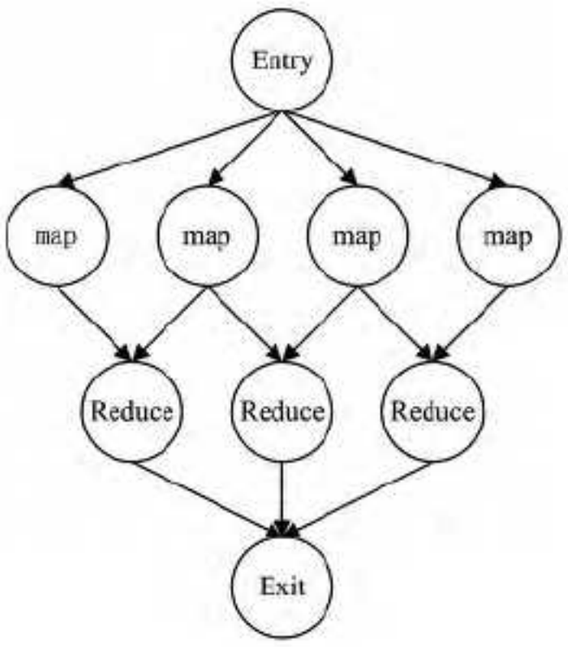

图 2 MapReduce 程序转化为 DAG

3.2 节点分类算法

通常在异构集群中包含不同计算能力与存储能力的节点。本文设节点的计算能力与存储能力成正比，即计算能力强的节点其存储空间大，用数据处理速度来衡量节点的计算能力。将节点按计算能力划分的目的是：(1)可优化数据存储的分布，以提高数据本地性；(2)可更准确地预算任务在节点上的运行时间。

设集群节点的计算能力可分为 K 个不同层次，设 L_P 代表第 P 层节点的计算能力级别，其中 $1<P<K$，设：

$$l_p=\begin{cases} l_{p-1}*\rho, & 1<P<k \\ 1, & P=1 \end{cases} \tag{2}$$

其中，P 为大于 1 的常量因子，且为提高分类的精确度，规定：

$$1<|\rho\approx l_p/l_{p-1}|<1.3 \tag{3}$$

设 P_i 为集群中某一节点计算能力，P_i 可表示为：

$$P_i=\frac{T_i}{B},1<i<N \tag{4}$$

其中，B 为在节点 i 上所处理数据的大小，N 为节点个数，T_i 为节点处理数据 B 所消耗的时间。

为计算节点的计算能力，我们将在集群上运行一组规定的作业，设定集群中的每个节点为一个 Map 槽与一个 Reduce 槽，并保证相同的输入，即尽量保证对每个节点的测试环境是公平的。记录每节点完成任务所需要的时间 $T=\{T_1,T_2,\cdots,T_N\}$。节点的分类算法如算法 1 所示。

算法 1 节点分类算法

```
N={1,2,3,…,n};//集群节点
Level=1;
CLASS[]=NULL;
LeveNum[]=NULL; //记录每层节点的数量
for i from 1 to n
  node=N[i];
  runTaskOn(node);
  T[i]=getExcuteTime(node);
end for
Sort T[] in descending order;
dividend=T[0];
for i=2 to n
  if dividend/T[i] <
LeveNum[Level].value=LeveNum[Level].value+1;
Then Class []=Class[] U T[i].node;
  else
    Level++;
    dividend=T[i]
  LeveNum[Level].value=LeveNum[Level].value+1;
    Then Class []=Class[] U T[i].node;
  end if
end for
```

节点分类算法可根据节点的计算能力将节点分类，并得到在某个计算能力层次上所拥有的节点个数。在同一层次上的节点其计算能力认为是相近甚至相等的，若设 $Class[i]$ 为处理能力为第 i 层的节点集合，$LevelNum[i].value$ 为对应集合元素的个数，则处于第 i 层的节点其计算能力 P_i 可表示为：

$$p_i=\sum_{P_j\in Class[i]} p_j/LevelNum[i].value \tag{5}$$

数据有规律地存储可有效提高数据的本地性，减少任务间的通信开销。本文根据节点的计算能力来进行数据的存储。假定通过节点分类算法已知集群中包含 K 类不同的节点，P_i 为第 i 类计算能力，D 为集群所需要存储的数据总量，设 D_i 为第 i 类节点应该分配的数据量，为了保证各节点处理数据的平衡，可设：

$$\begin{aligned} &\frac{D_1}{P_1}=\frac{D_2}{P_2}=\cdots\frac{D_i}{P_i}=\cdots=\frac{D_k}{P_k} \\ &D_1+D_2+\cdots+D_i+\cdots+D_k=D \end{aligned} \tag{6}$$

通过解式(6)中的联立方程，即可求得 i 类节点应存储的数据量 D_i。而在同一类节点中需要存储数据时，因计算能力相当，故可随机选择节点进行数据存储，但需要进行负载平衡时，仍可以迭代运行式(6)来决定数据的分布。

3.3 任务优先级排序

向上排序 $rank_u(n_i)$ 已成为经典的 DAG 任务调度排序，在向上排序中采用平均值 $\overline{w_i}$ 作为计算开销，其中平均值的计算方式为：

$$\overline{w_i}=\sum_{j=1}^{q} w_{i,j}/q \tag{7}$$

$w_{i,j}$ 代表任务 n_i 在处理器 P_j 上的预计执行时间。采用平均值作为计算开销，能近似地估算出任务在处理器上的执行时间，但对于异构的集群其得出的计算结果并不精确。考虑 Hadoop 集群异构的特性，本文认为在平均值的基础上每个任务在不同处理器上应有对应的排序值。设集群中包含若干类计算能力的节点，P_k 代表第 K 类计算节点，则任务 n_i 在第 K 类处理器上向上排序值为：

$$rank_u(n_i,p_k)=\overline{w_k}+\max_{n_j\in succ(n_i)}\{rank_u(n_j,p_k)+c_{i,j}\} \tag{8}$$

其中：

$$\overline{w_k}=\sum_{k\in class[k]} w_{i,k}/LevelNum[i].value \tag{9}$$

且 $$rank_u(n_{exit},p_k)=\overline{w_{exit,K}} \tag{10}$$

求取每个任务在每类处理器上的向上排序值后，便可求出每个任务在集群节点上的向上排序值，其计算方式为：

$$rank_u(n_i)=\frac{1}{|k|}*\sum_{j=1}^{k} rank_u(n_i,P_j) \tag{11}$$

这样得出来的向上排序值更符合异构集群的特性。

对计算任务 n_i 在节点 P_j 上的预算时间 $w_{i,j}$ 采取平均估算法，其估算方法与文献[13]类似。设函数 $CM(J,P)$ 代表已完成的任务 J 在计算能力为 P 级的节点上的 Map 任务集合。TM 代表 Map 任务 $M_m(M_mCM(J,p))$ 的完成时间，则平均完成时间 $AvgM$ 为：

$$AvgM=\frac{\sum_{M_m\in CM(J,p)} TM_m}{|CM(J,p)|} \tag{12}$$

当某一任务分配在处理器 P_j 上时，其执行时间近似相等于所有分配在 P_j 上已完成任务的执行时间的平均值。同理设 $CR(J,P)$ 代表已完成的任务 J 在计算能力为 P 级的节点上的 Reduce 任务集合，TR 代表已完成的 reduce 任务 R_r $(R_r CR(J,R))$ 的完成时间，该节点上 Reduce 任务的平均完成时间 $AvgR$ 为：

$$AvgR=\frac{\sum_{R_r\in CR(J,p)} TR_r}{|CR(J,p)|} \tag{13}$$

$AvgR$ 作为一般情况下 Reduce 任务分配在节点 P_j 上的预算执行时间。分别计算 Map 任务与 Reduce 任务的目的是考虑 Map 任务与 Reduce 任务的区别，提高预算的精确度。计算还有前驱节点任务没完成的任务时，可用该方法估算出分配在某处理器上该任务的执行时间，如果该任务没有前驱结点任务或前驱节点任务已完成，则可知待执行任务的数据输入规模，利用已知的数据规模能更精确地提高估算时间的精确度。设 $S(J,S,P)$ 为已在计算能力为 P 级的节点上完成的任务的数据规模集合，任务 n_i 为当前需要在处理器 P_j 计算能力为 P 级的节点上执行的 Map 任务，其数据输入规模为 Sc，则其预执行时间 $w_{i,j}$ 计算方法为：

$$w_{i,j}=\frac{s_c}{|S(J,S,P)|*1/|CM(J,P)|}*AvgM \tag{14}$$

同理若 n_i 为 Reduce 任务，其它参数同上，则：

$$w_{i,j}=\frac{s_c}{|S(J,S,P)|*1/|CR(J,P)|}*AvgR \tag{15}$$

Hadoop 中，在多个用户同时提交作业或一个用户同时提交多个作业等情况下，合理地选择其作业优先执行，既能提高集群的利用率还能保证用户作业执行的公平性。为保证较好的公平性，本文在调度过程中选用轮转调度[16,17]。优先任务的选取具体策略如下：

1）按 3.1 节中方法将用户提交的作业转换成 DAG；

2）计算各 DAG 中每个任务的向上排序值 $rank_u$；

3）将每个 DAG 中向上排序值最大的任务放入待执行队列；

4）将执行队列中的任务按向上排序值降序排序；

5）轮转调度待执行队列中的任务，直到队列为空，重复步骤 2）。

3.4 节点选择

由于集群异构的特点，对节点的选择比优先级的选择更为复杂。如何选取合适的节点来执行任务是影响集群性能的重要因子。选择不合理的节点来执行任务无疑会加大各 DAG 的调度长度。影响 Hadoop 中集群执行任务性能的主要因素包括数据本地性、节点计算能力、网络带宽。本文假设集群中各节点间的网络带宽是一样的，不考虑节点间的网络带宽的差异性。

任务在集群中执行，影响其通信开销的是数据本地性，而影响其运算时间的是节点的计算能力，如何综合两者之间的平衡进行调度是接下来考虑的重点。设 $CT(n_i,P_j)$ 为分配待执行任务 n_i 在节点 P_j 上运行直到任务完成所需要的时间，则：

$$CT(n_i,P_j)=w_{i,j}+ts(i,j)*x_{i,j}+\sum_{t=1}^{M}CT(n_t,P_j) \tag{16}$$

其中，$w_{i,j}$ 为任务 n_i 在节点 P_j 上的执行时间，定义如式(14)、式(15)所示，$ts(i,j)$ 为任务 n_i 在节点 P_j 上执行准备数据需要的时间，若任务 n_i 需要的数据已经存储在节点 P_j 上，则 $x_{i,j}$ 为 0，否则 $x_{i,j}$ 的值为 1。M 为已分配在 P_j 上等待执行的任务数。数据准备时间一般为数据从一个节点传输到另一节点所消耗的时间，设集群数据的传输速率为 V_{st}，任务 n_i 运行所需要的数据大小为 B，则：

$$ts(i,j)=B/Vst \tag{17}$$

综上所述可知，将任务 n_i 分配在集群中的最小完成时间 LCT(Least completion time) 可以表示为：

$$LCT=\min\{CT(n_i,P_1),CT(n_i,P_2),\cdots,CT(n_i,P_k)\} \tag{18}$$

通过式(18)的计算，能准确地算出将任务 n_i 分配在集群中的哪个节点到任务的完成所需要的时间最少，但是从实际的应用范围考虑，如果集群中含有上万个数据节点，对每个任务在每个节点上进行时间的预算将消耗很大的计算资源，这样调度机制本身消耗的资源过大，因此需要对策略进行改进。

在 3.2 节中已将节点按计算能力进行分类，注意到同一类中的节点其计算能力是相当的，因此计算同一类中的节点的 $CT(n_i,P_j)$ 时，由于任务 n_i 在处理器上的运行时间是相等的，因此只需要考虑其通信开销及节点上等待的任务个数，即可求出该类中哪个节点的 $CT(n_i,P_j)$ 值最小。在同一类节点中，$CT(n_i,P_j)$ 值最小的节点只可能出现在以下两种节点上。

1）节点上等待任务数最小的节点；

2）任务所需要的数据在该节点上，其通信开销为 0。

因此用 C_l 代表计算能力为 l 级的计算节点集合，用 $Y=1$ 表示待执行任务所需数据在对应节点上，否则为 $Y=0$，则最小 $CT(n_i,P_j)$ 值 $\min C_l\text{-}CT(n_i,P_j)$ 可表示为：

$$\min C_l\text{-}CT(n_i,p_j)=\min\{\sum_{t=1,Y=1}^{M}CT(n_t,P_j),ts(i,j)+\sum_{\substack{t=1\\Y=0}}^{M}CT(n_t,P_j)\} \tag{19}$$

其中，$ts(i,j)+\sum_{\substack{t=1\\Y=0}}^{M}CT(n_t,P_j)$ 只需要求其中待执行任务数最小的节点即可，如果同时存在待执行任务数最小的情况，则随机选取。通过节点的分类按节点计算能力和待执行任务数来计算 $CT(n_i,P_j)$ 值，将大幅度降低调度机制本身的开销。通过计算出每一类中预算时间最小的节点，可以得出节点在集群中的最小 $CT(n_i,P_j)$ 值，即：

$$LCT=\min\{\min C_1\text{-}CT(n_i,P_j),\cdots,\min C_K\text{-}CT(n_i,P_j)\} \tag{20}$$

在节点的选择中，本文将综合考虑数据的本地性与节点的计算能力，充分保障集群的利用率，并降低 DAG 的调度长度，对每个 DAG 提供公平调度与相对完成时间更短的调度长度，则把所有用户提交的全部作业转化成多个 DAG 也能保证多个 DAG 的完成时间更短。本文对节点的选择策略描述如下：

1）当集群中有空闲节点时，申请任务；

2）根据节点的计算能力分层，计算每层中的最小 $CT(n_i,P_j)$ 值；

3）将各层的最小 $CT(n_i,P_j)$ 值进行比较，选取具有最小 $CT(n_i,P_j)$ 值的节点作为当前任务分配的节点；

4）重复步骤 1）。

3.5 算法描述

基于 DAG 的 MapReduce 调度算法策略，首先须将用户提交的作业转换成 DAG，针对集群异构的特点将集群节点按计算能力进行分层，并根据异构性改进向上排序值的计算方式，较准确地算出每个 DAG 每个任务的向上排序值，采用公平性较好的轮转调度策略按向上排序值轮转调度每个 DAG。在节点的选择方面，不采用有空闲节点即把任务分配给该节点的分配方式，而是采用有空闲节点即可申请任务，但不一定能分配任务的方式，即按照当前任务根据其输入数据的规模、所需数据的本地性及节点的计算能力来决定当前任务被分配的节点。通过预算出任务被分配到哪个节点能被最快地完成，可有效缩短 DAG 的调度长度，从而提高整个集群的利用率。

算法 2　基于 DAG 的 MapReduce 调度算法

for(提交作业的个数)

　按 3.1 节中方法将每个作业转换为 DAG，并计算每个 DAG 任务的向上排序值

end for

for(DAG 个数)

　从每个 DAG 中选取向上排序值最大的且未被调度的任务放入就绪队列

　for(就绪队列长度)

　　将就绪队列按向上排序值降序排序，并轮转调度任务

While(有空闲节点)

　　计算任务在不同计算能力层次集群上的最小完成时间

　　从个层中选取具有最小完成时间的节点来完成分配的任务

　end while

　end for

end for

4　实验

本小节将我们的算法与 Hadoop 经典算法 FIFO 及基于时间约束的 MTSD 相比较。为了评估算法的性能，我们运行了一系列的作业。从数据本地性、总作业完成时间来评价算法的性能。同文献[13]一致，选择典型的 MapReduce 程序运行：Wordcount、Join、Girdmix。集群的硬件配置如表 1 所列。

表 1　Hadoop 集群信息表

Level	Amount	CPU	Memory	HardDisk
1	4	3.20Hz	2GB	500GB
2	4	2.7GHz	1G	320GB
3	4	1.6GHz	512M	80GB
4	4	800Hz	256M	40GB

通过提交多个不同的任务来测试算法的性能。

4.1　数据本地性

将节点进行分类，按节点的计算能力合理地将数据分布在节点上能有效提高数据本地性，从而减少任务间的通信开销。将任务按数据本地性与运算时间综合考虑既能充分利用数据的本地性，又不会使所有任务集中分布在计算能力强的节点上而导致任务分布不均匀从而降低整个集群的利用率。实验分 3 组进行，第一组运行 FIFO 默认调度算法；第二组运行基于时间约束的 MTSD 调度算法，该算法选用类似的节点分类方法将节点分类并按比例存放数据，但并未优化节点选择方案；第三组运行本文的调度算法。其数据本地性的分布情况如图 3—图 5 所示。

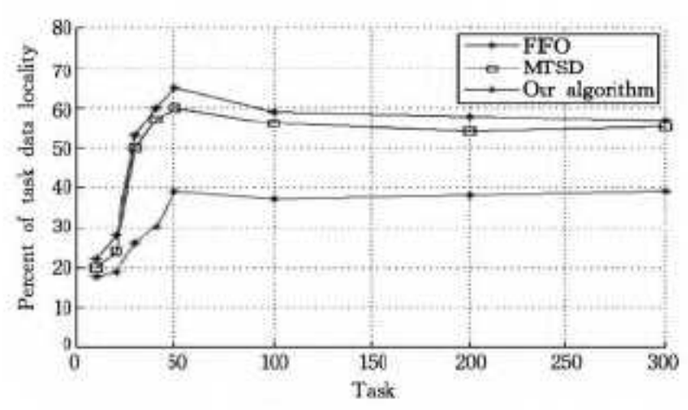

图 3　WordCount 作业

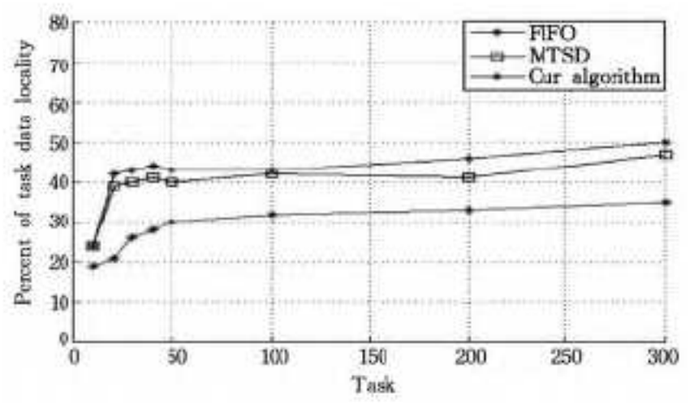

图 4　Join 作业

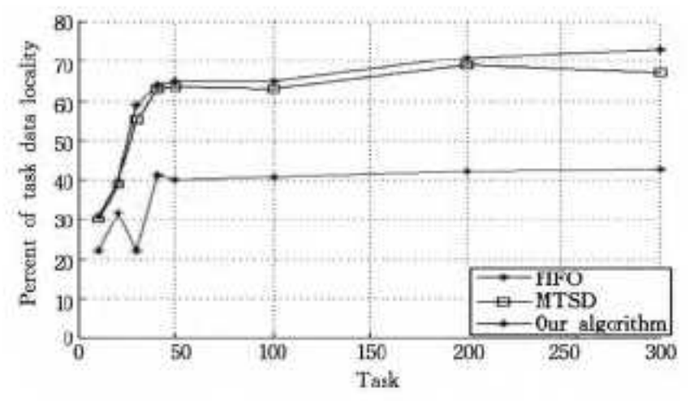

图 5　Gridmix 作业

从图 3—图 5 中可知，合理运地将节点进行分类对数据进行不同比例的存储可以将数据本地性提高约至 57%，而在任务分配时根据数据本地性及节点的计算能力选择适当的节点处理任务更能提高数据的本地性(约至 63%)，从而减少通信开销。

4.2　作业完成时间

集群的计算能力以集群在规定时间内完成任务的数量作为衡量标准。本实验将测试 FIFO 调度算法及本文提出的算法在多个用户同时提交多个不同任务的情况下不同调度算法完成任务的时间。其测试结果如图 6 所示。

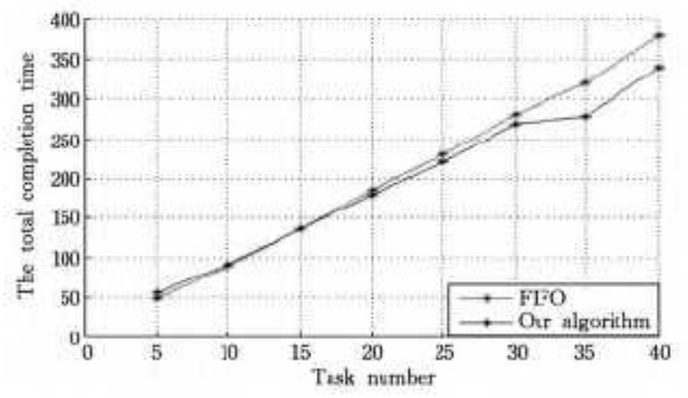

图 6　作业完成时间对比

从图 6 中可看出，默认的 FIFO 机制在任务数较小的情况下，总体任务完成时间会比本文提出的算法要短，其原因是我们的算法需要计算任务分配在哪个节点更合适，而计算本身需要消耗资源，当任务数少时反而需要更多的时间，但随着任务数的增加，对每个任务选择合适的节点执行就能逐步减少总任务的完成时间。

本文在任务执行时的优势在于，当有空闲节点申请任务时，系统既有未执行的任务又有空闲的计算资源，算法会根据系统当前运行状况将任务分配到预期执行时间最少的节点上

(下转第 51 页)

[10] Hamerly G, Perelman E, Lau J, et al. Using machine learning to guide architecture simulation[J]. Journal of Machine Learning Research, 2006, 7(2): 343-378

[11] Ipek E, McKee S A, Caruana R. Efficiently exploring architectural design spaces via predictive modeling[C]// Proceedings of ASPLOS. 2006: 195-206

[12] Jung H, Ju M, Che H. A Theoretical Framework for Design Space Exploration of Many Core Processors[C]// Proceedings of MASCOTS. 2011: 117-125

[13] 喻之斌, 金海, 邹南海. 计算机体系结构软件模拟技术[J]. 软件学报, 2008, 19(4): 1051-1068

[14] Guo Q, Chen T Y, Zhou Z, et al. Effective and efficient microprocessor design space exploration using unlabeled design configurations[C]// Proceedings of IJCAI. 2011: 1671-1677

[15] Tesauro G. Online resource allocation using decompositional reinforcement learning[C]// Proceedings of AAAI. 2005: 886-891

[16] Ganapathi A, Kuno H, Dayal U. Predicting multiple performance metrics for queries: better decisions enabled by machine learning [C]// Proceedings of ICDE. 2009: 592-603

[17] Cho C-B, Poe J, Li Tao, et al. Accurate, scalable and informative design space exploration for large and sophisticated multi-core oriented architectures[C]// Proceedings of MASCOTS. 2009: 16-25

[18] 李胜梅, 程步奇, 高兴誉, 等. 主成分线性回归模型分析应用程序性能[J]. 计算机研究与发展, 2009, 46(11): 1949-1955

[19] Huffmire T, Sherwood T. Wavelet-based phase classification [C]// Proceedings of PACT. 2006: 95-104

[20] Yuan Jing-ling, Jiang Tao, Zhong Luo. Grey neural network based predictive model for multi-core architecture 2D spatial characteristics[C]// Proceedings of ISNN. LNCS551, 2009: 889-892

[21] Martinez J F, Ipek E. Dynamic multi-core resource management: a machine learning approach[J]. IEEE MICRO, 2009, 29(5): 8-17

[22] 余凯, 贾磊, 陈雨强, 等. 深度学习的昨天, 今天和明天[J]. 计算机研究与发展, 2013, 50(9): 1799-1804

[23] Pang Jiu-feng, Li Xian-feng, Xie Jin-song. Microarchitectural Design Space Exploration via Support Vector Machine[J]. Acta Scientiarum Naturalium Universitatis Pekinensis, 2010, 46(1): 55-63

(上接第 46 页)

直到系统计算资源被完全利用, 不仅可缩短整体作业运行时间, 还可提高系统资源利用率。

结束语 本文针对异构的 Hadoop 集群提出基于 DAG 的 MapReduce 调度算法。该算法把每个提交给 Hadoop 的作业当成一个 DAG, 因此需要对每个作业进行 DAG 转化。针对集群的异构性将节点进行分类, 其目的是: (1)合理分布数据; (2)选择适当的节点运行任务。采用改进向上排序值使其在异构集群中更准确地算出每个任务的优先级, 再根据集群节点的计算能力及当前使用状况合理地选择节点执行任务。通过实验证明, 本文提出的方法能有效提高数据本地性并减少整个作业集的完成时间。

参 考 文 献

[1] Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters[J]. Communications of the ACM, 2008, 51(1): 107-113

[2] Apache Hadoop. Hadoop [EB/OL]. http://hadoop.apache.org/, 2009-03-06

[3] Vaquero L M, Rodero-Merino L, Caceres J, et al. A Break In the cloud: Towards a Ckoud Definition[J]. ACM SIGCOMM Computer Communication Review, 2009, 39(1): 50-55

[4] 陆嘉恒. Hadoop 实战(第 3 版)[M]. 北京: 机械工业出版社, 2011

[5] Zaharia M, Borthakur D, Sarma J S, et al. Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling[C]// Proceedings of the 5th European Conference on Computer Systems. 2010: 265-278

[6] Xie J, Yin S, Ruan X J, et al. Improving MapReduce Performance through Data Placement in Heterogeneous Hadoop Clusters[C]// IEEE International Symposium on Parallel & Distributed Processing, Workshops and PhdForum. 2010: 1-9

[7] Zhang X H, Zhong Z Y, Feng S Z, et al. Improving Data Locality of MapReduce by Scheduling in Homogeneous Computing Environments[C]// IEEE 9th International Symposium on Parallel and Distributed Processing with Applications. 2011: 120-126

[8] Guo Lei-tao, Sun Hong-wei, et al. A data distribution aware task scheduling strategy for mapreduce system[C]// First International Conference on Cloud Computing. 2009

[9] Verma A, Cherkasova L, Campbell R. Resource Provisioning Framework for MapReduce Jobs withPerformance Goals[J]. Lecture Notes in Computer Science, 2011, 7049: 165-186

[10] Polo J, Carrera D, et al. Performance-driven task co-scheduling for mapreduce environments[C]// Proc of IEEE/IFIP Network Operations and Management Symposium. 2010

[11] Kc K, Anyanwu K. Scheduling Hadoop Jobs to Meet Deadlines [C]// IEEE Second International Conference on Cloud Computing Technology and Science. 2010: 388-392

[12] Polo J, Carrera D, Becerra Y, et al. Performance-Driven Task Co-Scheduling for MapReduce Environments[C]// IEEE proceedings of Network Operations and Management Symposium. 2010: 373-380

[13] Tang Zhuo, Zhou Jun-qing, Li Ken-li, et al. MTSD: A task scheduling algorithm for MapReduce base on deadline constraints[C]// IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum. 2012

[14] Zaliva V, Orlov V. Hamake: A Data Flow Approach to Data Processing in Hadoop[C]// CLOSER. 2012: 457-461

[15] Furst S. Challenges in the design of automotive software[C]// Proceedings of the Conference on Design, Automation and Test in Europe. European Design and Automation Association, 2010: 256-258

[16] Arabnejad H, Barbosa J. Fairness resource sharing for dynamic workflow scheduling on Heterogeneous Systems[C]// Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on. IEEE, 2012: 633-639

[17] Klobedanz K, Koenig A, Mueller W. A reconfiguration approach for fault-tolerant flexray networks[C]// Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011. IEEE, 2011: 1-6