Journal of Computer Applications
计算机应用,2013,33(8):2158 -2162
ISSN 1001-9081
CODEN JYIIDU
2013-08-01
http://www.joca.cn

文章编号:1001-9081(2013)08-2158-05
doi:10.11772/j.issn.1001-9081.2013.08.2158

Hadoop 任务分配策略的改进

黄承真,王　雷*,刘小龙,况亚萍
(中国科学技术大学 自动化系,合肥 230027)
(*通信作者电子邮箱 wangl@ustc.edu.cn)

摘　要:Hadoop 广泛应用于大数据的并行处理,其现有的任务分配策略多面向同构环境,或者没有充分利用集群的全局信息,或者在异构环境下无法兼顾执行效率与算法复杂度。针对这些问题,提出异构环境下的任务分配算法 λ-Flow 算法,将原先一次完成的任务分配过程划分成多轮,每轮基于当前集群状态,以及上轮任务的执行情况,动态进行任务分配,直至全部任务分配结束,以期达到最优执行效率。通过与其他算法对比实验表明,λ-Flow 算法能够更好地适应集群的动态变化,有效减少作业执行时间。

关键词:Hadoop;MapReduce;任务分配;异构环境;最小费用最大流
中图分类号:TP301.6;TP393.027.2　　**文献标志码**:A

Tasks assignment optimization in Hadoop

HUANG Chengzhen, WANG Lei*, LIU Xiaolong, KUANG Yaping
(*Department of Automation, University of Science and Technology of China, Hefei Anhui 230027, China*)

Abstract: Hadoop has been widely used in large data parallel processing. The existing tasks assignment strategies are almost oriented to a homogenous environment, but ignore the global cluster state, or not take into account the efficiency of the implementation and the complexity of the algorithm in a heterogeneous environment. To solve these problems, a new tasks assignment algorithm named λ-Flow which was oriented to a heterogeneous environment was proposed. In λ-Flow, the tasks assignment was divided into several rounds. In each round, λ-Flow collected the cluster states and the execution result of the last round dynamically, and assigned tasks in accordance with these states and the result. The comparative experimental result shows that the λ-Flow algorithm performs better in a dynamic changing cluster than the existing algorithms, and reduces the execution time of a job effectively.

Key words: Hadoop; MapReduce; tasks assignment; heterogeneous environment; min cost max flow

0　引言

近年来,随着全球数据的爆炸式增长,海量数据的处理成为了一个新的研究热点。云计算[1]应运而生,类如 MapReduce[2]、Dryad[3]等编程模型的出现,使得大规模集群计算变得更加简单。Hadoop[4]是 Google MapReduce 的开源实现,已经被广泛地应用于各种大数据的并行处理问题,在搜索引擎、文本处理、机器翻译等领域都得到了大量应用与研究。Hadoop 具有良好的扩展性,能够处理分布在数以万计的低成本计算节点中的海量数据。例如:Facebook 的 Hadoop 数据仓库中存储了超过 2 PB 的数据,每天有超过 7 500 个应用在 Hadoop 集群上运行,处理超过 60 TB 的数据[5]。

Hadoop 中的文件在存储时被切分成若干个相同大小的数据块,每个数据块备份后分散存储在集群的数据节点上。在进行数据处理时,Hadoop 首先将一个作业切分成若干个 Map 任务并分配到各个节点上并行执行,每个 Map 任务处理一个数据块,Reduce 任务从各个节点上获取执行结果并产生最终的输出。

在执行 Map 任务时,如果数据恰好在 Map 任务执行的节点上,则称该 Map 任务是本地任务,否则称为远程任务。执行一个本地的 Map 任务比执行一个远程的 Map 任务代价更小,在 Hadoop 这种网络带宽比较稀缺的环境中体现得更为明显。如何调度任务使得作业在执行过程中拥有更多的本地化任务,从而减小作业执行代价已经成为了一个研究热点。

当前主要的调度算法可以分为静态调度和动态调度两类。静态调度的调度策略事先确定,不考虑集群运行时的动态变化;而动态调度是调度器动态地获取集群的状态信息,并根据这些信息动态调整调度策略。Hadoop 默认的任务分配算法[6]使用贪心算法,当一个节点出现空闲的资源时,尽可能地分配本地任务,如果没有本地任务,则分配远程任务。这种任务分配策略的优点是简单、易于实现,同时也减轻了 Jobtracker 的负担,但也存在着不利于任务执行的本地化的显著缺点。Facebook 提出了公平调度器[7],使用最大最小公平共享算法在作业间实现公平调度,该算法为新作业分配资源时,必须强制结束已有作业的部分任务以释放资源,从而导致资源浪费;而且公平调度没有区分对待长作业和短作业,无法保证系统整体性能。Zaharia 等[8]提出了延迟调度算法,即当一个作业中没有本地化的任务时,调度器先调度其他作业,而让该作业等待一小段时间。该调度策略在集群中存在大量短任务时能够非常有效地提高任务的本地性,但当集群中的任

收稿日期:2013-03-05;**修回日期**:2013-04-22。　**基金项目**:中央高校基本科研业务费专项资金资助项目(WK2100100012)。
作者简介:黄承真(1987 -),男,重庆人,硕士研究生,主要研究方向:云计算、大数据、Hadoop;　王雷(1972 -),男,安徽宿州人,副教授,博士,主要研究方向:计算机网络、媒体处理、云计算与云存储;　刘小龙(1989 -),男,重庆人,硕士研究生,主要研究方向:流媒体、网络传播与控制;　况亚萍(1991 -),女,安徽淮北人,硕士研究生,主要研究方向:云计算、虚拟化、网络传播与控制。

务多为长任务时表现不佳。Yahoo! 提出了计算能力调度器[9],将作业以队列为单位进行划分,适合于多用户共享集群的情况。当节点出现空闲时,调度器会根据计算能力调度算法依次选择队列、作业和任务,但没有考虑异构环境下的系统行为。Zaharia 等[10]提出了一种异构环境下的调度策略,通过 LATE(Longest Approximate Time to End)算法,预测任务执行的进度,并在其他节点重新执行那些进度缓慢的任务。该策略考虑了节点之间的异构性,并具有良好的鲁棒性,但对任务本地化考虑不够。以上调度器都属于动态调度,其共同的缺点是在节点资源出现空闲时才对进行任务的分配,没有利用 JobTracker 空闲时将分配策略预先制定好。为了更有效地进行任务分配,Fischer 等[11]对 Hadoop 任务分配问题进行了建模,提出了 Hadoop 任务分配(Hadoop Task Assignment, HTA)问题,即给定一个作业的数据在节点的分布情况,如何进行分配使得任务执行代价最小。HTA 问题已经被证明是一个 NP 完全问题。Fischer 等提出了基于网络最大流的 MaxCover-BalAssign 算法解决 HTA 问题,Sastry[12]实现了该算法。该算法被认为是接近最优的,但复杂度太高,并且属于静态调度,没有考虑到实际集中状态的变化。Jin 等[13]同样利用最大流算法提出了同构环境下的 Balance-Reduce 算法,综合考虑了任务本地化、网络状态以及集群负载,但是该算法复杂度仍较高,也属于静态调度。这两种算法还忽略了任务失败需要重新调度的问题。

综上所述,现有的任务分配策略或者面向同构环境,或者没有充分利用集群的全局信息,或者在异构环境下无法兼顾执行效率与算法复杂度。本文针对异构环境提出动态算法 λ-Flow,该算法基于当前集群状态,以及上轮任务的执行情况,动态进行任务分配,以期达到最优执行效率。实验证明该算法能够有效地减小任务执行的代价,提高任务的本地性。

1　问题建模

参照文献[11,13]提出的模型,做出如下定义:

定义 1　Hadoop 任务分配(HTA)问题。T 是作业中所有任务的集合,S 是所有节点的集合,Hadoop 任务分配问题指的就是如何将 T 中的每个任务分配到 S 上,使得任务执行的花费最小。

定义 2　数据放置图(Data Placement)。数据放置图是一个二分图 $G=(T\cup S,E)$,其中:T 是所有任务 t 的集合;S 是所有节点 s 的集合;$E\subseteq T\times S$ 是连接 T 与 S 之间边的集合,如果节点 s 上拥有任务 t 的数据的一个备份,那么 t 与 s 之间就存在一条边 $e(t,s)$。

定义 3　节点能力(Node Capability)。一个节点 i 的能力 NC_i 是其 CPU 能力、内存大小以及网络带宽的综合。定义当前节点可使用的资源总和为当前节点能力 CNC_i。在本文中认为同一个作业下的任务的执行效率跟当前节点能力 CNC_i 成正比。

定义 4　网络传输代价(Network Transfer Cost)。当一个任务 i 被分配到节点 j 上执行,而其任务数据却在节点 n 上时,就需要将数据从节点 n 通过网络传输到节点 j 上,定义该传输代价为 NTC_{ij}。如果 $n==j$,则 $NTC_{ij}=0$。

定义 5　任务代价(Task Cost)。定义任务代价表 $C=\{C_{ij},i\in T,j\in S\}$,其中:$C_{ij}$ 表示任务 i 在节点 j 上执行的总代价,包括传输任务数据的代价 NTC_{ij} 和真正执行的代价 RC_{ij},即 $C_{ij}=NTC_{ij}+RC_{ij}$。假定任务数据在节点 n 上,当 $n==j$ 时,$C_{ij}=RC_{ij}$。在 Hadoop 中,集群环境不断变化,因而网络传输代价 NTC 和执行代价 RC 也在变化,导致任务代价不断变化。

定义 6　任务分配策略(Task Assignment Strategy)。任务分配策略是一个函数 $\alpha:T\to S$,函数 $\alpha(t)=s$ 表示将任务 t 分配到节点 s 上执行。分配策略 α 是完全的当且仅当对于任意的 $t\in T$,$\alpha(t)$ 都被定义且 $\alpha(t)\in S$。定义 $C_i^\alpha(i\in S)$ 为分配策略 α 下分配到节点 i 上的任务执行的总代价,定义 $C^\alpha=\max\{C_i^\alpha, i\in S\}$ 为分配策略 α 的代价。

基于以上的定义,给出 Hadoop 任务分配问题形式化的定义:给定 Hadoop 中作业的数据放置图 $G=(T\cup S,E)$,网络传输代价 NTC,每个任务在不同节点的执行代价 RC,找到一个任务分配策略 α 使得作业的执行代价最小,即 $C=\min\limits_{\alpha}C^\alpha$。

例 1　如图 1 所示,图中有 5 个任务和 3 个节点,每个任务到节点之间的连线表示该任务数据的一个备份放置在这个节点上。可以看到,每个任务的数据的备份数是可以不一样的,每个节点的能力也是不一样的,每个任务在不同的节点上执行的代价也是不一样的。假定某一时刻任务代价表 C 和集群节点能力 CNC 如表 1 和表 2 所示,分配策略 α 如表 3 所示。

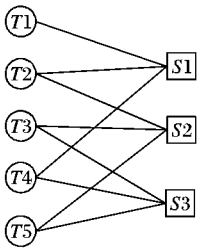

图 1　数据放置图

表 1　任务执行代价表 C

节点	任务				
	T1	T2	T3	T4	T5
S1	(0,5)	(0,5)	(1.5,5)	(0,5)	(1.5,5)
S2	(1,2.5)	(0,2.5)	(0,2.5)	(2,2,5)	(0,2,5)
S3	(1,2)	(1,2)	(0,2.0)	(0,2,0)	(0,2)

表 2　节点当前能力表 CNC

节点	CNC
S1	1.0
S2	2.0
S3	2.5

表 3　分配策略 α

任务	分配到的节点	任务	分配到的节点
T1	S1	T4	S3
T2	S2	T5	S3
T3	S2		

根据任务代价表 C,当前分配策略 α 下每个节点的中代价 $C_1^\alpha=5.0, C_2^\alpha=5.0, C_3^\alpha=4.0$,该分配策略 α 的代价为 $C^\alpha=\max\{5.0,5.0,4.0\}=5.0$。

2　λ-Flow 算法描述

本章中将详细介绍 λ-Flow 算法的原理与实现。其中 λ 是

步长,除了第一轮任务分配时为每个节点分配 $\lambda+1$ 个任务外,其余每轮任务的分配都将至多分配 λ 个任务。上一轮任务的执行情况将为下一轮任务的分配提供依据。算法统计上一轮任务执行的网络代价和实际执行时间,来预测当前集群的情况。在每一轮任务分配时,将任务分配问题转化为一个最小花费最大流问题[14]。

假定 Hadoop 集群中存在两个虚拟的节点 u 和 v, u 与各个任务均相连,网络带宽为 1,传输代价为 0,而每个机器节点与 v 相连,网络带宽为 λ,传输代价也为 0,于是得到以下关于 Hadoop 任务流网络的定义。

定义 7 Hadoop 任务流网络。给定一个数据放置图 $G'=(T\cup S,E')$ 和定义任务代价表 C,任务流网络 $G=(V,E)$ 是一个有向图,其中:$V=\{u\}\cup T\cup S\cup\{v\}$,设 $E=\{(u,t),t\in T\}\cup E'\cup\{(s,v),s\in S\}$。$u$ 为源点,v 为汇点。每条边 $e(x,y)\in E$ 均有一非负容量 $c(x,y)\geqslant 0$ 和一非负单位流量费用 $cost(x,y)\geqslant 0$,如式(1)与式(2)所示。

$$c(x,y)=\begin{cases}1, & x=u,y\in T\\ 1, & x\in T,y\in S\text{ 且 }e(x,y)\in E'\\ \lambda, & x\in S,y=v\end{cases}\tag{1}$$

$$cost(x,y)=\begin{cases}0, & x=u,y\in T\\ C_{xy}, & x\in T,y\in S\\ 0, & x\in S,y=v\end{cases}\tag{2}$$

这样便将某一时刻 t 的 Hadoop 任务分配问题转化成了最小费用最大流问题,其中每个流 f 对应于一个 Hadoop 任务分配策略 α。

例 2 数据放置图如图 1 所示,增加源点 u 与汇点 v,并给每条边赋予容量和单位代价,如图 2 所示,其中每条边的容量和单位代价可以分别由式(1)和式(2)计算得到。

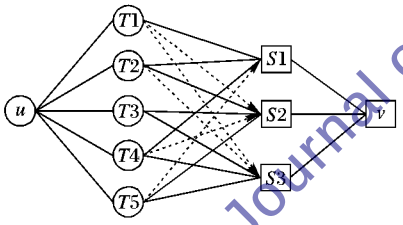

图 2 任务流网络

下面给出任务流网络最小费用最大流算法:

```
TaskMinCostMaxFlow(G,u,v,λ)
1)  getNetworkCapacity(G,c,λ)
2)  getNetworkCost(G,cost)
3)  for each edge(i,j) in E(G)
4)    do f[i,j] = 0
5)       f[j,i] = 0
6)  G_f = getGf()                          //得到残余网络
7)  while TRUE                             //循环查找增广路
8)    do p = existPath(G_f,u,v)
9)  if p == NULL
10)   return f
11) c_f(p) = min{c_f(i,j):(i,j) in p}
12) for each edge(i,j) in p                //更新增广路径 p 上的流量
13)   do f[i,j] = f[i,j] + c_f(p)
14)      f[j,i] = -f[i,j]
```

算法中,G_f 指的是残留网络,λ 指的是步长,首先通过上一轮任务的执行情况计算当前集群的容量和花费,然后得到流网络的残余网络,循环寻找最小费用可增广路,直到没有增广路为止,此时的流已经是最小费用最大流。

寻找最小费用增广路的算法 existPath 如下:

```
existPath(G_f,s,t)
1)  initialize-single-source(G,s);
2)  initialize-queue(Q);
3)  enqueue(Q,s);
4)  while not empty(Q)
5)    do u = dequeue(Q);
6)    for each v ∈ adj[u]
7)      do tmp = d[v];
8)         relax(u,v);          //对 u 的每个邻接点 v 松弛操作
9)         if (tmp > d[v]) and (not v in Q) then
10)          do enqueue(Q,v);
11) if (dist[t] == INF) then
12)   do return false;
13) return true;
```

算法使用了 SPFA(Shortest Path Faster Algorithm)[15]来计算残留网络 G_f 的最小费用增广路。SPFA 设立一个先进先出的队列用来保存待优化的节点,优化时每次取出队首节点 u,并且用 u 点当前的最短路径估计值对 u 点所指向的节点 v 进行松弛操作,如果 v 点的最短路径估计值有所调整,且 v 点不在当前的队列中,就将 v 点放入队尾。这样不断从队列中取出节点来进行松弛操作,直至队列空为止。

```
λ-Flow()
1)  initCost()
2)  while there exsits unassignment task in taskList
3)    do f = TaskMinCostMaxFlow(G,u,v,λ)
4)       TaskAssignment(f)
5)       time = waitForRunningComplete(λ-1)
6)       if there exsits failed task t then
7)         do addTask(taskList,t)
8)       collect the number of the left tasks in each datanode
9)       updateCost(time,t)
10)      updateCapacity(time,t)
```

算法开始时,假定所用节点的计算能力是一致的,节点之间的网络传输代价也是一致的,TaskMinCostMaxFlow 方法求得当前的最小费用最大流 f,根据 f 进行分配,并统计节点之间远程任务的数量,当有某个节点上已经运行完成 $\lambda-1$ 个任务时,统计出各个节点已执行完和未执行完的任务,以此预测节点的计算能力,计算下一轮每个节点应该分配的任务数量,从而更新任务流网络中边的容量和代价,经行下一轮的分配,直到所有的任务都分配执行结束。

算法的复杂度主要在于最小费用最大流算法的执行,假定集群节点数量为 $|S|$,每个节点每轮平均分配 x 个任务,作业的总长度为 L,那么整个算法将执行 $L/(x*N)$ 轮;existPath 算法的复杂度为 $O(k|E|)$,其中 $k\ll|E|$;TaskMinCostMaxFlow 算法复杂度为 $O(k|E|*\lambda|S|)$;所以整个算法的复杂度为 $O(L/(x*|S|)*k|E|*\lambda|S|)=O(L/x*k|E|*\lambda)$。$\lambda$ 跟集群状态变化的速率有关,其值越小时越易感知集群状态的变化,因此可将 λ 取为一个很小的常量,此时算法复杂度可以简化为 $O(L/x*k|E|)$。而除了第一轮分配外,其余每轮分配都是在最快的那个节点仅剩下最后一个任务未执行的时候分配,所以任务分配跟任务的执行并不存在时间上的关联,完全不影响任务的执行,所以实际中的花费更小。

3　实验及性能分析

在 Hadoop 中对 λ-Flow 算法进行了实现,FlowScheduler 实现了 TaskSchedur 抽象类,是 λ-Flow 算法的核心类,采用最小费用最大流算法进行多轮的任务分配;实现了配置管理类 FlowSchedulerConf,对算法配置参数进行读取;以及 FlowScheduleListener 类,将自己加入到系统的 Listener 队列中。实验是在一个具有 20 台计算机的集群上进行的,每个数据块的备份数都设置成了 3,将 λ-flow 算法的实验结果分别与 Hadoop Default Scheduler(HDS)算法、MaxCover-BalAssign(MC-BA)算法进行了对比。这里先简要介绍 HDS 与 MC-BA 算法。

HDS 算法:Hadoop 默认的任务分配策略,当某一节点有空闲的资源时,调度器使用贪心算法选取一个花费最小的任务,并将该任务分配给此节点。也就是说,当存在本地任务时,分配本地任务;当不存在本地任务时,随机选择一个任务分配给该节点。

MC-BA 算法:该算法由 Fischer 等[11]提出,并被 Sastry[12]实现。该算法迭代地产生一系列分配策略,选择出其中最优的分配策略。每次迭代的过程都包含两个步骤:MaxCover 和 BalAssign。在 MaxCover 中使用最大流算法,尽可能多地分配本地任务;在 BalAssign 中使用贪心算法,选择最小负载的节点,将剩下的未分配的任务进行分配。

3.1　λ-Flow 算法运行时间

在算法中,默认将步长 λ 设置成 5,作业的任务数从 100 到 800,进行了 8 组实验。分别对三个算法的运行时间进行测量,结果如图 3 所示。可以看到,随着任务数的增长,MC-BA 算法的执行时间急速增长,而 λ-Flow 算法所用时间仅比 HDS 算法略高。而由于 λ-Flow 算法从第一轮以后,每轮任务的分配与任务的执行同时进行,所以实际占用整个作业执行时间的比例更小。

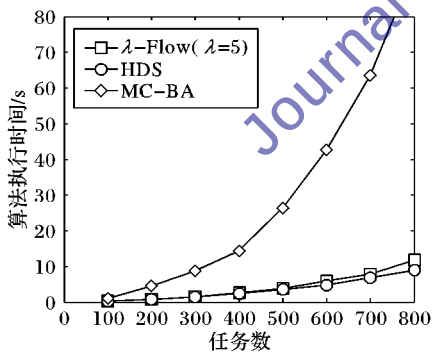

图 3　算法执行时间对比

3.2　作业完成时间

对同一个作业在不同的分配算法下作业执行的总时间经行了测量。考虑到集群状态不断发生着变化,集群负载也不断发生着变化。定义集群负载的平均变化率 r 为 10 s 内各个节点资源的变化情况,r 越大,表明集群环境越不稳定。分别在不同的集群负载变化率下对 λ-Flow、HDS 和 MC-BA 算法进行了对比测试,结果如图 4 所示。从图中可以看出,当集群比较稳定,即集群变化率很小时,MC-BA 算法和 λ-Flow 算法都比 HDS 算法表现要好;随着集群变化率的增大,λ-Flow 算法下的作业完成时间基本保持不变,而 MC-BA 算法下的作业完成时间急剧增加;当 $r \geqslant 0.3$ 时,MC-BA 算法下的作业完成时间已经比 HDS 算法下的作业完成时间更多了。这是因为 MC-BA 算法属于静态分配算法,其任务分配策略是根据作业执行前的集群环境制定的,当集群环境如节点负载、网络环境等发生变化时,MC-BA 算法可能会造成某一高负载的节点迟迟不能完成分配的任务,从而形成明显的长尾效应,进而影响到作业的整体执行时间。这也是集群负载变化率 $r \geqslant 0.3$ 时 MC-BA 算法甚至比 HDS 算法表现更差的原因所在。而 λ-Flow 算法属于动态算法,能够根据集群状态动态的调整分配策略,每轮分配都基于当前集群环境和上轮任务执行结果,在尽可能保证任务本地化的前提下,按照当前节点能力 *CNC* 成比例地进行分配,与 HDS 算法相比,λ-Flow 算法具有更好的本地性,而与 MC-BA 算法相比,λ-Flow 算法有更好的自适应性。

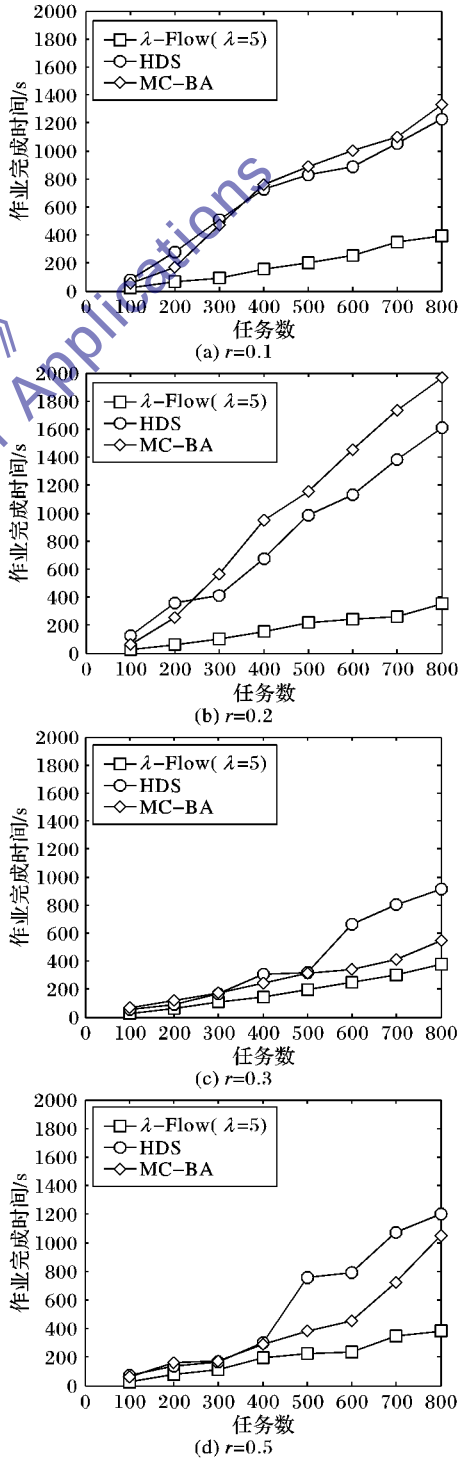

图 4　不同负载变化率 r 下作业的完成时间

3.3 作业完成时间与步长 λ 的关系

作业的完成时间跟每一轮分配任务的个数 λ 也有关系，调整 λ，在不同的集群变化率下对一个具有 800 个任务的作业的完成时间进行了测量，如图 5 所示。

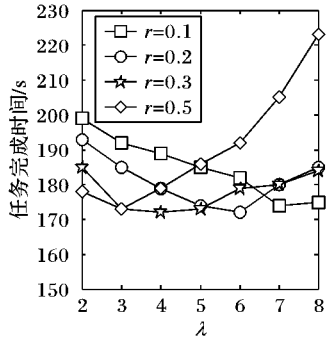

图 5 步长 λ 与作业完成时间的关系

从图 5 中可以看到，当集群变化率很小时，随着步长的增加，作业完成时间微微减少；当集群变化率 $r = 0.5$ 时，随着步长的增加，作业完成时间迅速增加。这是因为当集群变化率很小时，增加步长可以减少 λ-Flow 算法迭代的次数，能够少量地减少作业完成时间；而当集群变化率较大时，增加步长将无法及时感知集群状态的变化，从而增加了作业完成时间。在实际应用中，需要根据集群变化率设置 λ，一般情况下，当集群变化率很小且每个任务执行时间较短时，适当地增加 λ 能够减少作业的完成时间；而当集群变化率较大且每个任务执行时间较长时，应当适当地减小 λ 的值，这样算法才能通过上一轮任务的执行情况更快感知到集群的变化，更有利于整个作业的执行效率。由于 Hadoop 集群除了执行用户任务外，还需要通过定期的检查进行负载均衡、垃圾回收等维持自身的运转，所以集群环境在不断变化，并且在 λ-Flow 算法中，JobTracker 计算下轮任务分配策略的时机是在最快的那个节点只剩下一个任务还未执行的时候，所以 λ 的最小值为 2。从图 5 可以看出，当集群变化率为 0.1 时，$\lambda = 7$ 和 $\lambda = 8$ 条件下作业的完成时间已经几乎相同，所以设定 λ 的最大值为 8。事实上，λ-Flow 算法根据当前节点能力 *CNC* 平衡了各个节点之间的负载，也有利于减缓集群环境变化。λ 的设置可自定义，从图 5 中可以看出，当步长 $\lambda = 5$ 时，无论集群变化率为 0.1 还是 0.5，作业执行时间都相对比较良好，所以算法中默认 $\lambda = 5$。如果用户能够清楚地知道集群的变化率，可以对 λ 自行进行设置。

综合上述实验结果，λ-Flow 算法通过对上轮所分配任务的执行情况，动态地调整任务的分配策略，能够及时感知集群状态的变化，有效地减少作业的完成时间，从而提高作业执行的效率。

4 结语

随着数据密集型应用越来越多，Hadoop 也被应用得越来越广泛。当前 Hadoop 的任务分配策略多适用于同构环境中或者没有考虑到集群环境的变化。本文提出 λ-Flow 算法，通过动态地获取集群状态，根据上一轮分配任务的执行情况，动态地调整任务分配的数量。实验结果表明 λ-Flow 算法能够有效地减少作业执行时间，尤其能够适应于集群的动态变化，在异构环境下表现良好。

不足之处包括，λ-Flow 算法在更大规模集群中的表现还有待进一步验证，算法没有考虑作业间的公平性、优先级等问题，这将是下一步工作的目标。

参考文献：

[1] ARMBRUST M, FOX A, GRIFFITH R, *et al*. Above the clouds: a Berkeley view of cloud computing [J]. Communications of the ACM, 2010, 53(4): 50 – 58.

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

[3] ISARD M, BUDIU M, YU Y, *et al*. Dryad: distributed data-parallel programs from sequential building blocks[C]// EuroSys '07: Proceedings of the 2007 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems. New York: ACM, 2007: 59 – 72.

[4] Hadoop[EB/OL]. [2012 – 12 – 27]. http://hadoop.apache.org/.

[5] THUSOO A, SHAO Z, ANTHONY S, *et al*. Data warehousing and analytics in-frastructure at Facebook [C]// SIGMOD '10: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2010: 1013 – 1020.

[6] WHITE T. Hadoop: the definitive guide[M]. Sebastopol, CA, USA: O'Reilly Media, 2009.

[7] Fair Scheduler for Hadoop [EB/OL]. [2012 – 12 – 10]. http://Hadoop.apache.org/common/docs/current/Fair_scheduler.html.

[8] ZAHARIA M, BORTHAKUR D, SARMA J S, *et al*. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling [C]// EuroSys '10: Proceedings of the 5th European Conference on Computer Systems. New York: ACM, 2010: 265 – 278.

[9] Capacity scheduler for Hadoop[EB/OL]. [2012 – 12 – 13]. http://Hadoop.apache.org/common/docs/current/Capacity_scheduler.html.

[10] ZAHARIA M, KONWINSKI A, JOSEPH A D, *et al*. Improving MapReduce performance in heterogeneous environments [C]// OSDI'08: Proceedings of The 8th USENIX Conference on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2008: 29 – 42.

[11] FISCHER M J, SU X Y, YIN Y T. Assigning tasks for efficiency in Hadoop: extended abstract [C]// SPAA '10: Proceedings of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures. New York: ACM, 2010: 30 – 39.

[12] SASTRY G. FlowScheduler: towards efficient task assignment for Hadoop [EB/OL]. (2011 – 05 – 02) [2012 – 12 – 08]. http://girishsastry.com/papers/writeup.pdf.

[13] JIN J H, LUO J Z, SONG A B, *et al*. BAR: an efficient data locality driven task scheduling algorithm for cloud computing [C]// CCGRID '11: Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. Washington, DC: IEEE Computer Society, 2011: 295 – 304.

[14] ORMEN T H, LEISERSON C E, RIVEST R L, *et al*. 算法导论[M]. 殷建平, 徐云, 王刚, 等译. 北京: 机械工业出版社, 2006: 396 – 408.

[15] 段凡丁. 关于最短路径的 SPFA 快速算法[J]. 西南交通大学学报, 1994, 29(2): 207 – 212.