基于动态等待时间阈值的延迟调度算法*

邹伟明，于　炯，英昌甜，胡　丹
（新疆大学 信息科学与工程学院，乌鲁木齐 830046）

摘　要：针对已有的延迟调度算法存在的两个问题，即建立在节点会很快空闲的理论假设下有一定限制，当节点不会很快空闲时算法性能严重下降和基于静态的等待时间阈值不能适应云计算数据中心动态的负载变化及不同用户作业的需求，提出了一种基于动态等待时间阈值的延迟调度算法（dynamic waiting time delay scheduling，DWTDS）。该算法通过给无本地数据节点设置节点最大等待时间，以适应节点不会很快空闲的情况；通过分析数据中心各动态参数，根据概率模型调整作业的等待时间阈值。实验验证该算法在响应时间及负载均衡性方面优于已有的延迟调度算法。

关键词：云计算；延迟调度算法；数据本地性；Hadoop；MapReduce
中图分类号：TP393；TP301.6　　**文献标志码**：A　　**文章编号**：1001-3695（2012）11-4073-06
doi：10.3969/j.issn.1001-3695.2012.11.018

Dynamic waiting time delay scheduling algorithm in cloud computing

ZOU Wei-ming，YU Jiong，YING Chang-tian，HU Dan
（*College of Information Science & Engineering*，*Xinjiang University*，*Urumqi* 830046，*China*）

Abstract：There are two deficiencies in the current delay scheduling algorithms. Firstly，a limitation of these policies is that servers are not always become idle quickly as assumed，the performance of the algorithms declined serious when servers are not become idle quickly. Secondly，delay scheduling algorithms based on static waiting time threshold，cannot adapt to dynamic load of a data center and the different user needs . To address this issue，this paper proposed a dynamic waiting time delay scheduling algorithm（DWTDS）. the algorithm according to setting servers' biggest waiting time to adapt to the servers were not idle quickly，DWTDS adjusted jobs' waiting time threshold dynamically according to the information of variables factor in dada center. It shows that DWTDS outperforms previous delay scheduling algorithms in term of the job response time and load balance of the node.

Key words：cloud computing；delay scheduling algorithm；data-locality；Hadoop；MapReduce

0　引言

近年来，互联网应用平台（如 Facebook、Google、淘宝等）的数据量日趋庞大，为了解决对海量数据的存储与处理，并利用有效的资源管理与调度策略提高处理效率，云计算海量数据处理平台（如 MapReduce[1]、Dryad[2]、Hadoop[3]等）应运而生，在这些平台上，需要并发执行多个并行作业。对于数据密集型作业，网络带宽是计算集群中急缺的资源[4]。为了减少任务执行过程中的网络传输开销，可以将任务调度到输入数据所在的计算节点。因此，需要研究面向数据本地性（data-locality）[1,5]的任务调度算法。而系统中既包括子任务少、执行时间短、对响应时间敏感的即时作业（如数据查询作业），也包括子任务多、执行时间长的长期作业（如数据分析作业），研究公平调度算法可以及时地为不同的作业分配资源，使其快速响应。所以，为了提高系统性能和作业吞吐率，作业调度算法需要在保证作业之间公平分享数据中心资源的同时加强作业的数据本地性。

为了权衡用户公平性与数据本地性，加州大学伯克利分校的 Zaharia 等人[6]研究了云计算平台上的延迟调度（delay scheduling）算法。该算法采用延迟等待策略，在基本保证作业公平分享数据中心资源的同时，大幅提高数据本地性。

1　研究背景

1.1　云计算海量数据处理平台

在云计算海量数据处理平台中，数据文件由云计算分布式文件系统如 GFS[7]、HDFS[8]等管理。每一个文件（File）被划分成若干个大小相等的数据块（block），数据块分布在数据中心计算节点的本地磁盘；为了保证文件的可靠性，每一个数据块都存在多个数据块副本。当处理一个文件时，管理节点将一个作业（job）划分成若干个相互独立的小任务（task），每一个任务处理一个数据块，多个任务可以并行执行，这样可以加速一个作业的完成。当一个作业的所有任务完成后，该作业才能完成。如图 1 所示，一个 1 024 MB 的文件被分成八个大小相等（128 MB）的数据块，每个数据块有两个副本，分别存储在不同的计算节点

收稿日期：2012-04-18；**修回日期**：2012-05-21　　**基金项目**：新疆维吾尔自治区自然科学基金资助项目（2011211A011）；国家自然科学基金资助项目（60863003，61063042）

作者简介：邹伟明（1987-），男，湖南长沙人，硕士研究生，主要研究方向为网格、云计算（yxzwm3_8@126.com）；于炯（1964-），男，北京人，教授，博士，主要研究方向为网络安全、网格、分布式计算；英昌甜（1989-），女，新疆乌鲁木齐人，硕士研究生，主要研究方向为分布式计算、云计算；胡丹（1986-），男，新疆乌鲁木齐人，硕士研究生，主要研究方向为分布式计算、云计算.

上。为处理该文件,管理节点将该作业分成八个小任务,每个任务处理一个数据块。当某计算节点空闲时,该节点向管理节点请求任务,管理节点根据优先级策略,对队列中的作业排序,选取队首作业中的一个任务,并将其指派到该计算节点。从作业的第一个任务开始执行到最后一个任务结束的时间称为作业的完成时间。当作业中的所有任务都完成后,该作业才能完成。其中从本地磁盘读取输入数据的任务称为数据本地(data-locality)任务,通过网络从其他节点读取输入数据的任务称为数据远程(data-remote)任务。如图1所示,计算节点1的数据本地任务为1、2、4、5、7,数据远程任务为3、6、8。

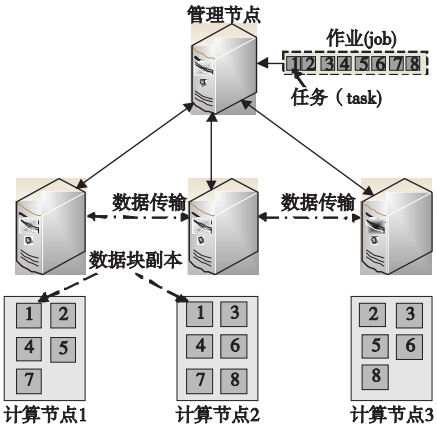

图1 云计算系统体系构架

1.2 相关任务调度算法

Hadoop 平台默认的调度算法[8],以"尽力而为"的策略保证数据本地性,即优先分配本地任务。虽然该算法易于实现,但不能保证较高的数据本地性。为此,Fischer 等人[5]提出了 MaxCover-BalAssign 算法。虽然该算法的理论上限接近最优解,但是时间复杂度过高,难以应用在大规模环境中。为了减少算法的时间复杂度,Jin 等人[9]设计了 BAR 调度算法。基于先均匀分配再均衡负载的思想,BAR 算法得到优于 MaxCover-BalAssign 算法的调度结果。为了同时保证数据本地性和作业公平性,Zaharia 等人[6]提出了基于等待策略的延迟调度算法。由于该算法采用静态的等待时间阈值不能适应数据中心负载的动态变化,文献[10]对其进行了改进。虽然该算法能动态调整等待时间阈值,但对所有作业统一一个动态的等待时间阈值,忽略了不同用户作业的需求,没能达到全局最优。

1.3 延迟调度算法

延迟调度算法的基本思想是:公平性方面,利用 Max-Min 公平调度算法[11]达到统计复用;数据本地性方面,若当前空闲的计算节点本地磁盘中没有队首作业所需的数据,则跳过该作业先调度其他作业而让队首作业等待;若队首作业的等待时间超过阈值,则立刻调度队首作业而不再等待。延迟调度的关键思想在于:当前空闲的节点不一定存在队首作业的本地任务,而云计算系统下任务完成得很快,很快就会有含本地任务的计算节点空闲,而这个等待时间开销远小于调度远程任务的开销。在 Yahoo 云计算中心(3 100 个计算节点的集群)采用 Hadoop 对 Facebook 中进行数据分析[6],平均每个 map 任务的处理时间为 19 s,节点的空闲速率为每秒 27.1 个计算节点,即多等待 1 s 就会有 27.1 个计算节点空闲,这 27.1 个计算节点中很可能就有队首任务的本地数据节点,从而可以把任务调度到本地数据节点,不需要网络传输开销,而直接调度远程任务的网络开销远大于这个等待时间。其中 83% 的作业都小于 271 个 map 任务,基于这个空闲速率,10 s 内集群中 83% 的作业都可以分配到计算节点。延迟调度算法的执行过程如算法 1 所示。目前,延迟调度算法的思想已在 Hadoop 云计算系统中得到实现。

算法 1 延迟调度算法(DS)

输入:当前到达的空闲计算节点 n。

```
将作业按照正在运行的任务数从小到大排序,并将结果保存在 jobs 中
for j in jobs do
  if j has unlaunched task t with data on n then
    launch t on n
  //j 中存在输入数据在节点 n 上且未被分配的任务则分配给 n
  set j. wait = 0
  /* j. wait 是作业 j 已经等待的时间,当 j 中有任务被分配后,将其清零 */
  else if j has unlaunched task t then
    if j. wait > = T_WAIT then
  /* T_WAIT 是等待时间阈值,若作业 j 等待的时间超过阈值且 j 中存在未被分配的任务,则将其分配给 n */
      launch t on n
  else
    j. wait + +
  //作业 j 不超时则继续等待
    end if
  end if
end for
```

1.4 问题的提出

1.4.1 无本地数据节点闲置问题

随着任务的调度,某些节点已无本地数据,当其中一些节点空闲时,依照延迟调度算法只能等待超时的任务进行调度,而延迟调度算法的数据本地任务所占比例很高(97% 以上),因超时而需要远程调度的任务数很少,因此这些无本地数据节点会因得不到超时任务而一直等待,造成资源的浪费,特别是在节点负载比较大不会很快空闲的情况下,会严重影响作业的完成时间。

1.4.2 设置静态等待时间阈值的问题

1)网络传输开销会随远程任务数量的变化而变化,作业等待的时间也应作相应的变化。例如,当网络上要传输的数据量增加时,网络的传输速度会有所下降,作业等待的时间应相对增加。

2)作业的大小不同,空闲计算节点所含本地任务的概率也不一样,而且随着任务的执行,总任务数量减小,空闲计算节点所含本地任务的概率也会减小,延迟算法中对所有作业统一一个静态的等待时间阈值会影响短任务的完成时间,也会影响后期任务的执行。

针对以上问题,有必要对延迟算法进行一些研究改进。

2 改进的延迟调度算法

本文提出的 DWTDS 算法主要对 DS 算法作了以下两方面的改进:a)给每个作业 J_i 分配一个等待时间阈值 T_i,而不是对所有作业设定一个固定的等待时间阈值,T_i 随着作业的执行进度及数据中心负载的变化自动调整;b)给无本地数据节点增设一个节点最大等待时间 T_n,当某个空闲节点已无本地任务又无超时任务可调度、且下一个计算节点不会很快空闲时,该节点等待一定的时间 T_n 后随机选择一个队首未分配的任务执行。

2.1　问题定义

定义 1　节点空闲速率。单位时间内到达的空闲计算节点的数目称为计算节点的空闲速率,记做 λ。λ 可由一段时间内系统平均空闲计算节点的到达速率确定。

定义 2　任务的执行时间。任务在计算节点开始执行到任务完成的时间,记做 T_l。

定义 3　任务的网络传输开销。当空闲节点执行远程任务时,读取远程任务的数据所需的时间称为任务的网络传输开销,记做 T_r。

对于数据本地任务,由于其从本地磁盘中读取数据,所以传输开销 $T_r=0$。

对于数据远程任务,其传输开销 T_r 表示为

$$T_r = q \times r \tag{1}$$

网络带宽为单位时间内传输的数据量。当远程任务数量增加时,数据的传输开销会线性增大[9],可以表示为 $T_r = q \times r$。其中:r 为远程任务的数量;q 为网络因素,q 值越大表示网络越拥塞,$1/q$ 表示网络的数据传输速率。

定义 4　作业的等待时间阈值。作业 J_i 期望的等待时间称为等待时间阈值,记做 T_iw。

$$T_i^w = E(T_{\text{wait}}) \tag{2}$$

其中:T_{wait} 为作业的等待时间,作业在 T_i^w 时间内等待可满足本地性的空闲计算节点到达,若 T_i^w 时间内没等到合适的计算节点,则不再等待。

定义 5　节点的最大等待时间。当空闲计算节点向管理节点请求分配任务而无任务分配时该节点等待执行任务的时间。该算法只对无本地数据节点设置节点的最大等待时间,记做 T_n。

定义 6　作业的数据本地概率。对于 T_i^w 时间内将要到达的空闲计算节点来说,其能满足某个作业 J_i 数据本地性的概率称为该作业的数据本地概率, 记做 P_i。

$$P_i = 1 - (1 - \frac{N}{M})^{\lambda T_i^w} \tag{3}$$

其中:N 为作业 J_i 的数据块副本分布在计算节点的数目;M 为计算集群中计算节点总数;λT_i^w 表示 T_i^w 时间内到达的空闲计算节点数目。

定义 7　任务的总开销。任务的总开销为从作业得到执行时间片到任务执行完成的时间,包括等待开销、网络传输开销和任务的执行时间三部分,记做 C_{tast}。

$$C_{\text{tast}} = T_{\text{wait}} + T_r + T_l \tag{4}$$

其中:T_{wait} 为任务的等待开销;T_r 为任务的网络传输开销;T_l 为任务在计算节点上的执行时间。对于数据本地任务,由于其从本地磁盘中读取数据的时间相比任务的执行时间可以忽略[9],故 $C_{\text{localtast}} = T_{\text{wait}} + T_l$。

定义 8　作业的完成时间。从作业得到执行时间片到作业完成的时间,记做 C_{job}。

$$C_{\text{job}} = \sum_{i=1}^{t} C_{\text{tast}} \tag{5}$$

2.2　概率模型求解

DWTDS 算法的目的是同时保证公平性与本地性,既要尽量减小作业的总完成时间,也要减小每个作业的完成时间。通过式(4)和(5)可知,可以减小每个任务的开销。设作业 J_i 的等待时间阈值为 T_i^w,即 $T_i^w = E(T_{\text{wait}})$(见定义 4)。若在 T_i^w 时间内有满足数据本地性的空闲计算节点到达,则任务的总开销 $C_{\text{tast}} = T_{\text{wait1}} + T_l (T_{\text{wait1}} \leqslant T_i^w)$;若在 T_i^w 时间内无满足数据本地性的空闲计算节点到达,则任务的总开销 $C_{\text{tast}} = T_{\text{wait2}} + T_r + T_l$ $(T_{\text{wait2}} \geqslant T_i^w)$。故等待 T_i^w 时间任务的总开销的数学期望为

$$\begin{aligned} E(C_{\text{tast}}) &= (T_{\text{wait1}} + T_l) \times P_i + (T_{\text{wait2}} + T_l + T_r) \times (1 - P_i) = \\ &[T_{\text{wait1}} \times P_i + T_{\text{wait2}} \times (1 - P_i)] + T_l + T_r(1 - P_i) = \\ &E(T_{\text{wait}}) + T_l + T_r(1 - P_i) = \\ &T_i^w + T_l + T_r(1 - P_i) \end{aligned} \tag{6}$$

将式(1)和(3)代入式(5)求导,可得任务总开销的数学期望最小值及其所对应的 T_i^w。

$$T_i^w = -\frac{1}{\lambda} \times \log_{(1-P_i)} \ln (1 - P_i) \tag{7}$$

其中:$P_i = 1 - (1 - \frac{N}{M})^{\lambda T_i^w}$。该模型比文献[10]中的模型更简单,但由于考虑了不同用户作业的需求,给每个作业都要分配一个等待时间阈值,增加了系统的计算开销。

2.3　节点闲置问题

考虑无本地数据节点闲置问题,当某个空闲节点已无本地任务又无超时任务可调度、且下一个计算节点不会很快空闲时,该节点等待一定的时间 T_n 后随机选择一个队首未分配的远程任务执行,该节点等待的时间应满足 $T_n \leqslant T_r$。

证明　设某个作业由 t 个任务组成,该作业的完成时间为 C_{job},在此不妨假设该作业的前 $t-1$ 个任务已经完成,记完成时间为 T,根据算法 1,此时该作业的等待时间置 0,即 $T_{\text{wait}}=0$,现有一个无本地数据节点空闲,则该节点的等待时间 $T_n = T_{\text{wait}}=0$。由于作业不超时且无本地任务分配,故该节点继续等待,之后有两种情况出现:a)若在 T_i^w 时间内有本地数据节点到达,则把该任务分配给数据本地节点(此时 $T_r=0$);b)若等待 T_{wait} 后该任务因超时被远程调度。

由式(4)可知,当情况 a)时:

$$\begin{aligned} C_{\text{job}} = \sum_{i=1}^{t} C_{\text{tast}} = \sum_{i=1}^{t} (T_{\text{wait}} + T_r + T_l) = \\ T + T_{\text{wait}} + T_r + T_l = \\ T + T_n + T_l \end{aligned}$$

当情况 b)时:

$$\begin{aligned} C_{\text{job}} = \sum_{i=1}^{t} C_{\text{tast}} = \sum_{i=1}^{t} (T_{\text{wait}} + T_r + T_l) = \\ T + T_{\text{wait}} + T_r + T_l = \\ T + T_n + T_r + T_l \end{aligned}$$

若不采取等待策略($T_{\text{wait}}=0$),直接把该任务远程分配给该空闲等待的无本地数据节点,则

$$\begin{aligned} C'_{\text{job}} = \sum_{i=1}^{t} C_{\text{tast}} = \sum_{i=1}^{t} (T_{\text{wait}} + T_r + T_l) = \\ T + T_{\text{wait}} + T_r + T_l = \\ T + T_r + T_l \end{aligned}$$

显然,当 $T_n \geqslant T_r$ 时,无论情况 a)或 b)都有 $C_{\text{job}} \geqslant C_{\text{job}}'$,原因很简单,若等待时间 T_n 超过 T_r,则直接分配远程任务的开销比继续等待的开销小,故节点的等待时间应满足 $T_n \leqslant T_r$(注意:T_r 是变量,会随着网络的负载变化而变化,如式(1)),证毕。

现举例说明,如图 2 所示,设一个集群有三个计算节点,初始负载分别为 10.2、8.0、0.2,向该集群提交一个任务数为 8 的作业,采用两个副本策略,其分布如图 1 所示,设每个任务在节点上的执行时间 $T_r = 1$ s,网络因素 $q = 0.1$,则 $T_r = q \times r$(r 为远

程任务的数量),等待时间阈值为 3 s。按照 DS 算法的执行过程如图 2 所示。当第 8 个任务调度完成后,节点 3 已无数据本地任务,此时负载为 5.2,作业开始等待其他节点的空闲,等待 2.8 s 后节点 2 空闲,马上调度任务 1,接着依次调度任务 4 和 7,最后节点负载分别为 10.2、11、5.2,作业的完成时间为 11 s。

若在调度完第 8 个任务后,由于节点 3 已无数据本地任务,为解决该无数据本地任务节点闲置问题,在 DWTDS 算法中对这样的节点设置一个节点等待时间 $T_n = T_r$。若节点等待的时间超过 T_r,则选择一个队首未完成的远程任务执行,如图 3 所示,此时 $T_r = q \times r = 0.1 \times 1 = 0.1$,0.1 s 后在节点 3 上执行任务 1,此后等待 $T_r = q \times r = 0.1 \times 2 = 0.2$ 后调度任务 4,节点 3 的实际负载变为 7.5(除去等待时间 0.3 s),等待 0.2 s 后节点 2 空闲,在节点 2 上调度执行任务 7,最后节点的负载分别为 10.2、9.0、7.5,作业的完成时间为 9.0 s。相比 DS 算法作业的完成时间减少 2 s,且节点的负载更加均衡,没有造成资源的闲置。

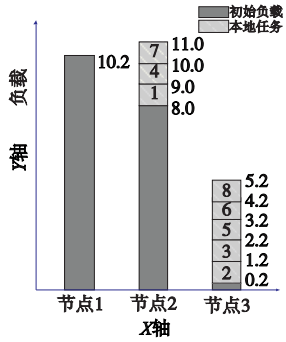

图2 DS调度过程

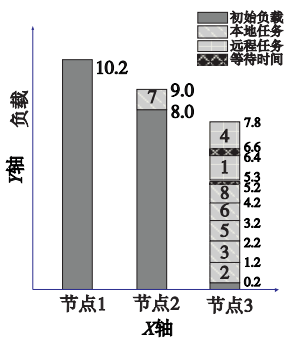

图3 DWTDS调度过程

2.4 DWTDS 算法描述

基于动态等待时间阈值的延迟调度(DWTDS)算法的基本步骤如下:

a)调用 Max-Min 公平调度算法确定作业优先级。

b)通过节点初始负载确定节点空闲速率并计算出网络传输开销。

c)由式(7)计算得到等待时间阈值。

d)通过比较已经等待的时间和等待时间阈值,确定作业是否继续等待。

e)对无本地数据节点设置节点最大等待时间,确定节点是否继续等待。

算法 2 基于动态等待时间阈值的延迟调度算法(DWTDS)

输入:当前到达的空闲计算节点 n。

```
将作业按照正在运行的任务数从小到大排序,并将结果保存在 jobs 中
for j in jobs do
  if j has unlaunched task t with data on n then
    launch t on n
    // j 中存在输入数据在节点 n 上且未被分配的任务则分配给 n
    set T_wait = 0
  /* T_wait 是作业 j 已经等待的时间,当 j 中有任务被分配后将其清零 */
    λ = λ0; //计算节点空闲速率
    T_r = q * r; //计算网络开销
    T_j^w = T_j0^w; //计算每个作业 j 的等待时间阈值
  else if j has unlaunched task t then
    if T_wait > = T_jw then
  /* 若作业 j 等待的时间超过阈值且 j 中存在未被分配的任务,则将其分配给 n */
      launch t on n
      set T_wait = 0
  else if n∈N 且 T_n > = T_r then
  /* N 为无本地数据节点集合,T_n 是节点的等待时间,若无本地任务且无超时任务而节点的等待时间超过此时的网络传输开销,则分配远程任务 t 给 n */
      launch t on n
      set T_wait = 0
    else
    T_wait ++
  //否则作业 j 继续等待
    end if
    end if
end for
```

3 实验及结果分析

为了分析本文中所提出的 DWTDS 算法的性能,下面将改进的延迟调度(DWTDS)算法与原始的延迟调度(DS)算法、无延迟的 FIFO 算法进行比较。通过 MATLAB 仿真实验,分析了算法的性能。

3.1 实验设置

采用 MATLAB 仿真 Hadoop 环境,设置 50 个计算节点,每个数据块大小为 128 M(Facebook 中数据分析设置的默认值),采用 Hadoop 默认的三个副本策略,副本分配矩阵随机给出,因为在 Facebook 中平均每个数据块的处理时间为 19 s[6],在这里设置 $T_l = 20$ s,在 10 Gbit 的网络环境中传输 128 M 的数据需要 1 s,1 Gbit 的网络环境中需要 10 s,在此仿真实验中可设置 $q \in [1,10]$[9]。为了分析算法在不同的集群工作负载(包括 IO 负载及 CPU 负载)环境中的效率,实验通过调节 λ 值反映集群工作负载大小。λ 值越小,表示集群工作负载越大,节点的空闲速度越慢。

为了更好地分析算法在实际环境中的性能,实验原始数据将仿照 Facebook 中在一周内所提交的作业进行设置[6],Facebook 中大部分都是小作业,其中 1 ~ 2 个任务的作业占总作业数的 55% 以上,中小作业(100 个任务以下的作业)占总任务数的 80% 以上,作业依照任务数目分成五组,并根据实际环境中的作业负载来确定每组作业的数目。具体设置如表 1 所示。

表 1 作业参数设置

组号	任务数	平均任务数	作业数
1	1	1	20
2	2	2	11
3	3 ~ 20	10	7
4	21 ~ 60	50	5
5	61 ~ 150	100	3

通过同时提交这五组作业,分析比较不同的集群工作负载及不同的网络负载环境下各算法对数据本地性、作业完成时间、节点负载的影响。

3.2 集群工作负载对算法的影响分析

为了比较算法在不同集群工作负载下的性能,实验将计算节点的空闲速率分别设置为 $\lambda = 0.05$,$\lambda = 0.5$,$\lambda = 5$,统一在 10 Gbit/s 的网络环境下分析不同负载对各算法作业完成时间的影响。

如图 4 所示,当 $\lambda = 5$ 时,集群工作负载比较小,节点的空闲速率快,在这种情况下,改进的 DWTDS 算法与 DS 算法的性能差不多,作业的总完成时间分别为 298.7 s 和 299.0 s。图 5 为 $\lambda = 0.5$ 时集群工作负载下作业完成时间的比较。如图 6 所

示,当 λ =0.05 时,集群工作负载很大,节点的空闲速率慢,采用 DS 算法静态的等待时间阈值继续等待务必会影响小任务的完成。从图 6 中可以看出,第 1、2 组作业的完成时间 DS 算法最差,而 DWTDS 算法通过分析集群工作负载动态调整等待时间,能很好地适应节点不会很快空闲的情况,各组作业的完成时间都比 DS 算法少,其中第 5 组作业的完成时间比 DS 算法少 174.9 s。

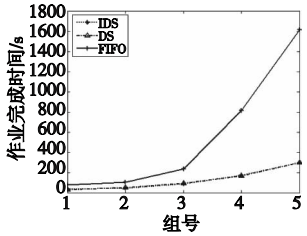

图4　λ=5集群工作负载下作业完成时间的比较

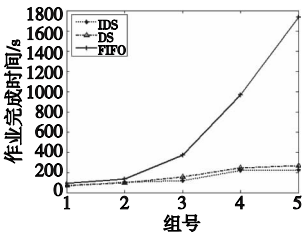

图5　λ=0.5集群工作负载下作业完成时间的比较

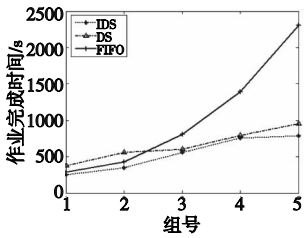

图6　λ=0.05集群工作负载下作业完成时间的比较

3.3　网络负载对数据本地性与公平性的影响分析

为了解决数据本地性与公平性相互冲突的矛盾,延迟调度算法通过延迟策略以牺牲一定的公平性为代价提高数据本地性。数据本地性的提高可以减少网络传输开销,缩短作业的完成时间。实验通过比较数据本地任务占总任务的比例,分析各调度算法的性能。实验表明,DWTDS 算法可以根据集群状态及任务的执行进度作出快速反应,动态调整数据本地性。

如图 7 所示,在 10 Gbit/s 网络中,开始时网络传输开销很小,应该减少等待时间,基于 DWTDS 算法,组 1 的 50% 任务都被远程调度,而随着远程任务数量的增加,网络传输开销会增大,应该增大等待时间,保证任务的本地性。从图 7 中可以看出,DWTDS 算法中组 2 ~ 组 5 的大部分任务都可以被指派到其输入数据所在的计算节点中,本地概率都在 98% 以上,说明算法能根据网络负载的变化作出及时的调整,而 DS 算法中组 2 的数据本地任务所占比例仍不到 50%,对网络负载变化反应比较慢。

如图 8 所示,在 1 Gbit/s 网络中,网络传输开销比较大,应该尽量把任务调度到数据所在节点,以减小网络传输开销,从而应该增加等待时间。若采用静态的等待时间阈值,从图 8 中可以看出,DS 算法数据本地性会大大降低,而 DWTDS 算法作业的等待时间阈值会随网络负载的变换作出相应的调整,还可以看出,组 1 的数据本地任务所占比例为 75%,比 10 Gbit/s 网络环境中提高了 25%,而其他几组均接近 100%,明显优于 DS 算法与无延迟的 FIFO 算法。

3.4　网络负载对作业完成时间的影响分析

该算法的主要目标是通过减少每个作业的完成时间,使任务的总完成时间缩短,从而提高算法的性能。图 9 ~ 12 分别是在 10 Gbit 和 1 Gbit 的网络环境下各算法作业完成时间的比较。

如图 9、10 所示,组 2 的完成时间 DS 算法比 DWTDS 算法少,因为开始时网络传输开销小,为保证数据本地性(由图 7 可知)等待的时间越长,会增加作业的完成时间。随着任务的执行,网络传输开销也增大,数据本地性越高作业的开销越小,因此组 3 ~ 组 5 DWTDS 算法的完成时间要少于 DS 算法,以组 5 为例,DWTDS 算法的作业完成时间比 DS 算法少 39.7 s,比 FIFO 算法少 1390.1 s。

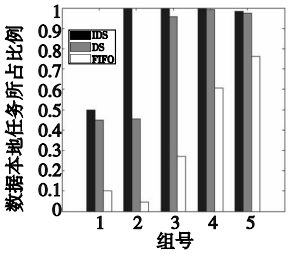

图7　10 G bit(q=1,λ=0.5)网络环境下数据本地性的比较

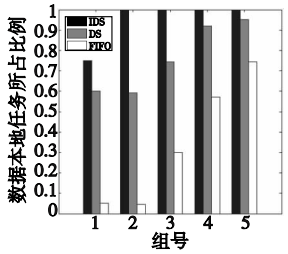

图8　1 G bit(q=10,λ=0.5)网络环境下数据本地性的比较

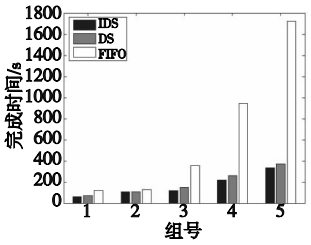

图9　10 G bit(q=1,λ=0.5)环境下作业完成时间的比较(三种算法)

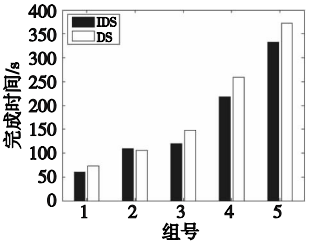

图10　10 G bit(q=1,λ=0.5)环境下作业完成时间的比较(两种算法)

在 1 Gbit 的网络环境下,网络传输开销增大,作业的完成时间也明显增加。DWTDS 算法:a)通过更高的数据本地性(由图 8 可知),以保证作业的快速完成;b)由于数据中心负载加重,会出现节点不会很快空闲的情况,DWTDS 算法能有效地解决节点闲置问题,加快作业的完成。如图 11、12 所示,DWTDS 算法要明显优于 DS 算法以及 FIFO 算法。

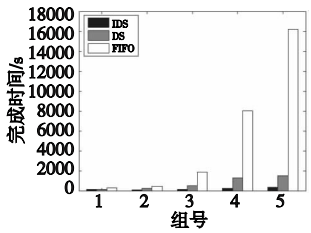

图11　1 G bit(q=10,λ=0.5)环境下作业完成时间的比较(三种算法)

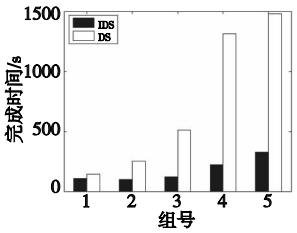

图12　1 G bit(q=10,λ=0.5)环境下作业完成时间的比较(两种算法)

3.5　计算节点负载分析

为了更合理地利用资源,防止出现某个资源负载过重而无法继续提交任务或某些资源负载过轻而浪费资源的情况,均衡节点负载有重要意义。图 13 ~ 16 分别是在 10 Gbit 和 1 Gbit 的网络环境下各算法节点负载的比较。如图 13、15 所示,有延迟的 DWTDS 算法与 DS 算法都体现了比无延迟的 FIFO 算法更好的负载均衡性,这也是延迟调度算法公平性的体现。

如图 14、16 所示,DWTDS 算法节点负载更加均衡,在 1 Gbit 网络环境下,网络开销比较大,使得节点负载也加重,DS 算法中存在的无本地数据节点闲置问题凸显出来。如图 16 所示,节点 1、2、3、7、8……的负载在 1000 s 左右,而节点 5、12、13……的负载在1600 s 左右,因为节点 1 在完成最后一个任务时的节点负载为 996 s,此时节点 1 已无数据本地任务,只能等待超时任务,造成该节点一直闲置。而 DWTDS 算法采用给无数据本地任务节点设置最大等待时间的方法,在节点无数据本地任务且无超时任务分配时,等待一定的时间后选择队首的一

个未分配的远程任务分配执行,解决了 DS 算法中存在的无本地数据节点闲置问题。

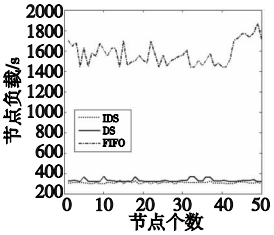

图13 10 G bit环境下节点负载的比较(三种算法)

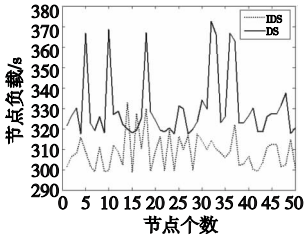

图14 10 G bit环境下节点负载的比较(两种算法)

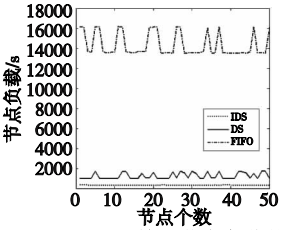

图15 1 G bit环境下节点负载的比较(三种算法)

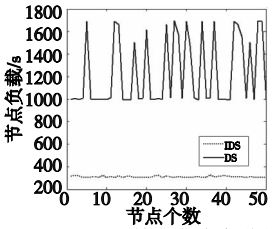

图16 1 G bit环境下节点负载的比较(两种算法)

4 结束语

本文提出了一种基于动态等待时间阈值的延迟调度算法。该算法在等待时间阈值及节点闲置问题上对原有的算法进行改进,对每个作业设置一个动态的等待时间阈值,以调整作业的数据本地性与公平性;对无本地数据节点增设一个最大等待时间,较好地解决了节点闲置问题,使其能够适应节点不会很快空闲的情况。仿真实验结果表明,改进的延迟调度算法通过以上两方面的改进,在作业完成时间及负载均衡方面优于已有延迟调度算法的性能。

参考文献:

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

[2] ISARD M, BUDIU M, YU Yuan, *et al*. Dryad: distributed data-parallel programs from sequential building blocks[C]//Proc of ACM SIGOPS/EuroSys European Conference on Computer Systems. New York: ACM Press, 2007: 59-72.

[3] Hadoop [EB/OL]. (2011-12-18) [2012-03-12]. http://hadoop.apache.org.

[4] WANG Guo-hui, NG T S E. The impact of virtualization on network performance of amazon EC2 data center[C]//Proc of the 29th Conference on Information Communications. Piscataway, NJ: IEEE Press, 2010: 1163-1171.

[5] FISCHER M J, SU Xue-yuan, YIN Yi-tong. Assigning tasks for efficiency in Hadoop: extended abstract[C]//Proc of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures. New York: ACM Press, 2010: 30-39.

[6] ZAHARIA M, BORTHAKUR D, SARMA J S, *et al*. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling[C]//Proc of European Conference on Computer Systems. Paris: [s. n.], 2010: 265-278.

[7] GHEMAWAT S, GOBIOFF H, LEUNG S T. The Google file system [C]//Proc of the 19th ACM Symposium on Operating Systems Principles. New York: ACM Press, 2003: 29-43.

[8] WHITE T. Hadoop: the definitive guide[M]. [S. l.]: O'Reilly Media, Inc, 2009.

[9] JIN Jia-hui, LUO Jun-zhou, SONG Ai-bo, *et al*. BAR: an efficient data locality driven task scheduling algorithm for cloud computing [C]//Proc of the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. Washington DC: IEEE Computer Society, 2011: 295-304.

[10] 金嘉晖,罗军舟,宋爱波,等. 基于数据中心负载分析的自适应延迟调度算法[J]. 通信学报, 2011, 32(7): 47-56.

[11] Max-min fairness[EB/OL]. (2011-11-16)[2012-03-12]. http://en.wikipedia.org/wiki/Max-min_fairness.

[12] CHANG F, DEAN J, GHEMAWAT S, *et al*. Bigtable: a distributed storage system for structured data[J]. ACM Trans on Comput Systems, 2008, 26(2): 1-26.

(上接第 4063 页)率,并对本文算法的有效性进行了分析。但是本文仍有不足之处,如针对该方法只在一个数据集上进行了实验验证。在多个数据集上验证本文的算法和将该方法拓展到动态的 PPI 网络进行链接预测是笔者进一步研究的任务。

参考文献:

[1] KEMPE D, KLEINBERG J M, TARDOS E. Maximizing the spread of influence through a social network[C]//Proc of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2003: 137-146.

[2] ZHOU Tao, LV Lin-yuan, ZHANG Yi-cheng. Predicting missing links via local information[J]. The European Physical Journal B, 2009, 71(4): 623-630.

[3] 吕琳媛. 复杂网络链接预测[J]. 电子科技大学学报, 2010, 39(5): 651-661.

[4] 王琳,商超. 无标度网络中的链路预测问题研究[J]. 计算机工程, 2012, 38(3): 67-71.

[5] LIBEN-NOWELL D, KLEINBERG J. The link prediction problem for social networks[C]//Proc of International Conference on Information and Knowledge Management. 2003: 556-559.

[6] CHEN Ji-lin, GEYER W, DUGAN C, *et al*. Make new friends, but keep the old recommending people on social networking sites[C]//Proc of the 27th International Conference on Human Factors in Computing Systems. 2009: 201-210.

[7] MAXWELL J C. A treatise on electricity and magnetism[M]. 3rd ed. Oxford: Clarendon, 1892: 68-73.

[8] PAN Jia-yu, YANG H I, FALOUTSOS C, *et al*. Automatic multimedia cross-modal correlation discovery[C]//Proc of the 10th ACM-SIGKDD International Conference on Knowledge Discovery and Data Mining. 2004: 653-658.

[9] IAKOVIDOU N, SYMEONIDIS P, MANOLOPOULOS Y. Multiway spectral clustering link prediction in protein-protein interaction networks[C]//Proc of the 10th IEEE International Conference on Information Technology and Applications in Biomedicine. 2010: 1-4.

[10] POPESCUL A, UNGAR L H. Statistical relational learning for link prediction[C]//Proc of at IJCAI Workshop on Learning Statistical Models from Relational Data. 2003.

[11] RUAN Quan-song, DUTTA D, SCHWALBACH M S, *et al*. Local similarity analysis reveals unique associations among marine bacterioplankton species and environmental factors[J]. Bioinformatics, 2006, 22(20): 2532-2538.

[12] RUAN Quan-song, STEELE J A, SCHWALBACH M S, *et al*. A dynamic programming algorithm for binning microbial community profiles [J]. Bioinformatics, 2006, 22(12): 1508-1514.

14 检验 Q 中的除点 P 之外的所有点 S,如果 $S_p == S_s$,从 V 中删除 S 点
15 endif
16 endwhile
17 V′ = ∅, Q = ∅;
18 endfor
19 找出 E′中所有重复的边,只保留重复项中相似度最大的一条边
20 printf(E′);
end

因为本文方法是基于网络拓扑结构进行预测的,所以考虑孤立点和邻居节点很少的点没有意义,为了尽快有效地预测潜在的信息,本文不考虑网络中度小于 3 的节点。当且仅当顶点 p 的相关节点集 $S_p > 3$ 时,将 S_p 中的所有节点加入到 V'中(第 3 行),搜索全图中的除点 P 之外的所有节点。如果有点与 S_p 中的所有节点都相连,将该点加入到队列 Q 中,作为 S_p 的候选项(第 4、5、6 行)。计算候选项中任意两点之间的相似度,如果大于相似度阈值,就预测这两点之间存在链接关系(第 11、12 行)。检验所有候选项的节点相关集是否与 S_p 相同,如果相同,从图 V 中删除该点(第 14 行)。将 V'、Q 清空,按照上述方法再考虑图中其他节点,直到图中所有的节点都被考虑过为止。检查输出结果集,删除重复边,只保留两点中权重最大的边(第 16 行)。

根据算法,先找出 PPI 网络中的每个种子节点相关的节点集,然后找到设置此相关节点的所有候选项,最后通过上述定义的相似性,预测这些候选项之间的链接关系。显然,本文算法的复杂度为 $O(n^2)$。

3　实验

3.1　数据集描述和分析

实验以酵母蛋白质相互作用网络作为研究对象,因为酵母是所有物种中蛋白质相互作用数据最为完备的,以 1998 年 Cho 等人发布的酵母基因表达数据作为微阵列数据。这些数据均可以从斯坦福基因数据库和其他生物数据库中得到。它们是通过观察酵母细胞在 α 条件下从分裂前期的 G_1 期到分裂后期的 M 期表达收集的。

3.2　结果对比

3.2.1　相似度阈值变化对实验结果的影响

实验 1　以 MIPS 数据库中的酵母蛋白质交互网络作为标准(http://mips.helmholtz-muenchen.de/),随机从网络中剔除 500 条边,构成一个不完整的网络。在这个不完整网络中按照本文的算法进行预测,再用预测后的结果与原有的标准网络进行比较。考察在阈值 ε 变化的情况下,实验结果在精确度上的变化。其中,准确率 P 按式(4)计算可得

$$p = \frac{n}{m} \tag{4}$$

其中:n 为在预测结果在标准网络中存在的链接数,m 为预测的总链接数。图 2 为实验结果精确度随 ε 的变化。

从图 2 中可以看出,随着相似度阈值 ε 的增大,预测结果的准确率也在不断增大。当 $\varepsilon = 0.9$ 时,预测结果的最高准确率可达 45%;当 $\varepsilon = 0.4$ 时,预测结果的准确率也能达到 27%,平均准确率为 32%。显然,本文的算法在链接预测中有相对较好的准确率。

3.2.2　网络不同完整性水平对实验结果的影响

实验 2　通过不同完整性水平上的网络来观察实验结果准确率的变化。首先从原酵母蛋白质交互网络中随机剔除一些边,分别保留 40%、50%、60%、70%、80%、90% 条边,形成不同的不完整网络。为了观察网络的整体水平,分别取 $\varepsilon = 0.9$, 0.8, 0.7, 0.6, 0.5, 0.4 在每个网络上进行预测,得到的结果按照式(4)分别计算出不同相似度阈值上的准确率,然后对每个网络的 6 个不同相似度 ε 上的准确率进行平均,得到每个网络上的平均准确率,再对每个网络上的平均准确率进行比较分析。图 3 为实验结果随观察点的变化。

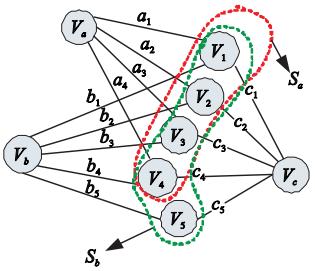

图1　例子

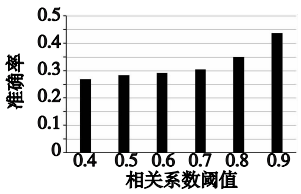

图2　实验结果精确度随ε的变化

从图 3 中可以看出,观察点越多实验结果的准确率就越大。当网络中有 90% 的边是已知时,预测结果的最高平均准确率可达 47%;当只有 40% 的边是已知时,预测结果的平均准确率也能达到 32%。从上述的分析结果可以看出,本文的算法能够进行有效地预测。

3.2.3　权重信息对实验结果的影响

实验 3　通过不同权重信息构成的权值网络来观察不同完整性水平下网络预测结果准确率的变化。首先运用只有拓扑权重信息构成一个蛋白质交互网络,利用本文的相似度预测方法进行链接预测并对准确率进行评价;然后运用只有基因表达数据权重信息构成一个蛋白质交互网络,利用本文的相似度预测方法进行链接预测并对准确率进行评价;最后将两种权重信息融合在一起构成一个蛋白质交互网络。从上述三种原始酵母蛋白质交互网络中随机地剔除一些边,分别保留 40%、50%、60%、70%、80%、90% 条边,形成不同的不完整网络。运用本文基于相似度的方法分别在相关系数大于 0.4 的情况下观察不同权重下预测结果的准确率。图 4 为实验结果比对。

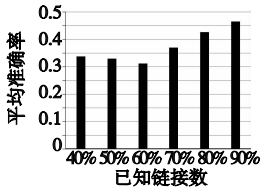

图3　实验结果随观察点的变化

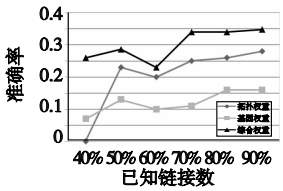

图4　实验结果比对

从图 4 中可以看出,综合权重网络的整体准确率优于单个权重网络的准确率。当网络中有 90% 的边是已知时,综合权重网络准确率可达 35%,而基因表达数据构成的权重网络的准确率只能达到 16%,拓扑权重网络的准确率只能达到 27%。从上述的分析结果可以看出,单一信息在链接预测的局限性和将两种权重信息融合起来进行链接预测可以达到很好的效果。

4　结束语

本文提出了一种基于蛋白质之间相似度的方法来预测 PPI 网络中潜在的、隐藏的链接。本文将 PPI 网络看做是一个有权图,根据网络中两节点的拓扑结构和权重信息,通过计算它们的相似度来预测它们是否存在链接关系,而且通过真实的酵母蛋白质交互网络观察预测结果的准确

(下转第 4078 页)