

一种异构环境下的基于 MapReduce 任务调度改进机制*

何翔^a, 李仁发^{a,b}, 唐卓^{a,b}

(湖南大学 a. 嵌入式系统及网络实验室; b. 网络与信息安全湖南省重点实验室, 长沙 410082)

摘要: 针对在异构环境下采用现有 MapReduce 任务调度机制可能出现各计算节点间数据迁移和系统资源分配难以管理的问题, 提出一种动态的任务调度机制来改善这些问题。该机制先根据节点的计算能力按比例放置数据, 然后通过资源预测方法估计异构环境下 MapReduce 任务的完成时间, 并根据完成时间计算任务所需的资源。实验结果表明, 该机制提高了异构环境下任务的数据本地性比例, 且能动态地调整资源分配, 以保证任务在规定时间内完成, 是一种有效可行的任务调度机制。

关键词: MapReduce; 调度算法; 资源预测; 数据放置; 异构环境

中图分类号: TP302 **文献标志码:** A **文章编号:** 1001-3695(2013)11-3370-04

doi:10.3969/j.issn.1001-3695.2013.11.042

Improved task scheduling mechanism for MapReduce in heterogeneous environment

HE Xiang^a, LI Ren-fa^{a,b}, TANG Zhuo^{a,b}

(a. Embedded System & Network Laboratory, b. Hunan Province Key Laboratory of Network & Information Security, Hunan University, Changsha 410082, China)

Abstract: The existing MapReduce scheduling mechanism that used in heterogeneous environment may lead to the migration of data between compute nodes, and manage system resource allocation difficulty. This paper proposed a dynamic task scheduling mechanism to improve these problems. First, this mechanism distributed data in proportion according to the computing capacity of each node. Then it estimated MapReduce job completion time in heterogeneous environment by using resource prediction model, and calculated task slot requirements based on its completion time. The experiment results show that this mechanism can improve the proportion of task data locality, and dynamically allocate resources in order to ensure the job is completed in the specified time. It is demonstrated that this task scheduling mechanism is effective and feasible.

Key words: MapReduce; scheduling algorithm; resource prediction; data placement; heterogeneous environment

0 引言

随着互联网和分布式计算技术的发展,越来越多的公司使用新兴的 MapReduce 编程模式及它的开源实现 Hadoop 来处理大规模数据密集型计算^[1]和数据挖掘^[2]等方面的应用。MapReduce^[3]编程模式是由 Google 公司提出,它是目前用来解决分布式集群中数据密集型处理问题最流行的模型。Hadoop^[4]是一个能够对大量数据进行分布式处理的软件框架,实现了 Google 的 MapReduce 编程模型和框架,它能够把应用程序分割成许多小的工作单元,并把这些单元放到集群中的任何节点上执行。

在异构的 Hadoop 集群中每个机器的软硬件配置是不同的,其计算能力也会有明显的不同,高速节点会比低速节点更快地处理完存储在本地磁盘的数据。为了尽快地完成数据,快速节点在处理完本地输入数据任务时,会耗费有限的网络带宽来请求处理附近慢速节点未处理的数据。而由慢速节点向快速节点移动未处理的数据所产生的时间开销是无法预测的,这

会使集群的资源分配难以预测。针对这一问题,目前国内外已经有一些研究^[5,6]。文献[7]提出 SAMR (self-adaptive MapReduce) 的任务调度策略,它通过每个节点的任务历史执行信息, SAMR 能动态地把执行最慢的任务进行备份操作缩短执行时间,但是 SAMR 在执行备份任务时,并没有考虑异构环境下数据的分布问题。文献[8]提出的 LATE (longest approximate time to end) 调度算法考虑了系统的异构性,其通过估计所有任务的剩余时间并把执行最慢的任务作为备份任务,这样可以有效缩短 MapReduce 作业在异构环境中执行的相应时间,但是它也没有考虑数据放置的问题。文献[9]提出了基于已知节点数据分布的 MapReduce 任务调度策略,该调度策略基于节点和任务的优先级,在充分考虑系统中数据的分布情况下进行任务调度,但是它没有考虑异构环境下的资源分配情况。文献[10]提出一种预测任务执行的剩余时间,并根据任务剩余时间动态预测 MapReduce 任务的性能和为该任务调整资源分配的方法,但该方法是在同构环境下实现的,也没有考虑数据放置问题。与文献[10]相近的研究还有文献[11~13],它们都

收稿日期: 2013-01-31; **修回日期:** 2013-03-14 **基金项目:** 国家自然科学基金资助项目(60673061, 60873074)

作者简介: 何翔(1988-),男,湖南祁阳人,硕士研究生,主要研究方向为云计算(354679194@qq.com);李仁发(1956-),男,教授,博士,主要研究方向为嵌入式系统、信息网络;唐卓(1981-),男,讲师,博士,主要研究方向为云计算和信息安全。

没有充分考虑异构环境下数据放置和资源分配的问题,因此不能根据数据划分粒度和任务运行情况自动选择合适的任务调度策略。

基于上述问题的研究,本文在考虑到节点异构性基础上提出基于数据放置位置和资源预测方法的动态任务调度机制。

1 MapReduce 数据处理机制

1.1 MapReduce 介绍

MapReduce^[3]是 Google 提出用来简化分布式系统的编程模型,它的优势在于提出了一个抽象的计算模型,并隐藏系统框架的实现细节。MapReduce 通过用户定义的 map 和 reduce 两个函数对数据进行处理。Map 函数主要把输入数据映射成一组新的中间键值对 $\langle \text{key}, \text{value} \rangle$,通过 reduce 函数处理中间键值和得到最终输出结果 $\langle K', V' \rangle$ 。

MapReduce 作业的执行过程^[14]是在 map 阶段读取用户输入数据,并为其创建相应的键值对,然后执行用户定义的 map 函数,并将其产生的中间结果存放在所在节点的缓冲区中,当缓冲区快要溢出时则将缓冲结果写入本地磁盘。在写入磁盘之前,还需根据 reduce 任务的数目将数据划分为相同数目的分区。在 shuffle&sort 阶段,reduce 任务将会等待获取通过 HTTP 协议传递来的 map 阶段所产生的相关中间值。最后把多个 map 任务的输入中间数据进行合并操作,并将其合并的数据作为一个 reduce 任务的输入数据。在 reduce 阶段执行用户自定义的 reduce 函数,并且产生用户所需的最终结果。上述过程如图 1 所示。

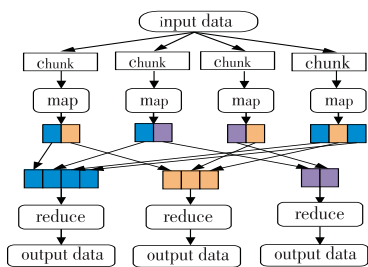


图1 MapReduce数据处理过程

1.2 Hadoop 和 Hadoop 数据放置策略

Hadoop^[4]是基于 MapReduce 的开源实现版本。Hadoop 框架包括两个部分:由运算节点组成的 MapReduce 模块和由存储节点组成的分布式文件系统 HDFS 模块,它们均采用主从式的管理方式。

MapReduce 模块主要由两类节点构成,一个主节点 jobTracker 和多个工作节点 taskTracker。JobTracker 主要负责接收客户端提交的作业,并把作业划分成两种任务类型,即 map 任务和 reduce 任务,每种任务的个数是由作业的配置文件和输入文件决定,然后将任务分配给工作节点 taskTracker。TaskTracker 负责管理和执行 map 及 reduce 任务,taskTracker 的任务执行单元(slot)只能执行一个 map 或 reduce 任务。TaskTracker 会定时向 jobTracker 汇报任务执行情况。如果 taskTracker 的任务执行单元没有任务执行,则 jobTracker 会根据调度算法给它分配一个新的任务。

在 HDFS^[15]中,负责数据块放置管理节点为 nameNode,而负责实际数据存放的节点称为 dataNode。当用户向 nameNode 发起存储请求时,nameNode 按照指定的数据放置策略将数据

存储在集群内的多个 dataNode 上。HDFS 在选择存放数据块的节点时采用的策略是将数据块的多个副本同时存放在本地机架与一个随机的远端机架的节点上。它可以确保节点的 map 任务能够从本地读取数据,而当本地节点因故障失效时,系统则通过移动远端节点的副本恢复数据,但该策略在放置多个数据副本时采用的远端节点随机选择的方法可能会导致数据迁移时不必要的性能损失。当随机选择远端机架的节点距离本地节点太远时,两者之间的数据移动会增加不必要的时间开销,同时不能保证节点之间数据存储的平衡。

2 方案及算法设计

2.1 异构集群中数据放置方法

当前 Hadoop 的数据放置方法是把用户的输入数据划分为同样大小的数据块分散在 dataNode 上。在同构的 Hadoop 集群中,这种方法是公平的,但是它并不适合异构集群,因为异构集群中各个节点的计算能力是不同的。因此本文提出基于节点计算能力的放置方法。该方法使高速节点比低速节点能够分配更多的数据块,如果存储在每个节点的数据块数量与它们的数据处理速度成正比,则数据移动量就可以极大地降低。这样就可以忽略各个节点计算能力的差异性,使各个节点几乎在同样的时间内完成本地数据的处理。

在用户输入的数据进行放置之前,需要量化异构集群中各个节点的数据处理速度。首先通过定义节点的计算能力来衡量在异构集群中每个节点的数据处理速度。计算能力表示节点处理单元数据所用的时间。

定义 1 节点 i 的计算能力 P_i 可表示为

$$P_i = \frac{T}{B} \quad 1 \leq i \leq n \quad (1)$$

其中: B 表示节点 i 处理的数据大小; T 表示节点 i 处理 B 所用时间; n 为集群中节点的个数。

每个节点计算能力的确定是由下面的步骤来完成。首先假如异构集群中有 N 个节点,在每个节点中分别单独运行相同的 MapReduce 作业 J ,并保证每个节点处理相同数量的数据;然后记录每个节点运行作业的完成时间为 T_i ;最后由于节点处理数据大小相同,通过式(1)就可以粗略地计算得到每个节点的计算能力 P_i 。

下面将说明如何通过节点的计算能力来进行数据放置。假设用户输入数据为 D ,在集群中有三个异构节点 A 、 B 、 C 。在集群每个节点上分别运行作业后,可以得到关于节点 A 、 B 、 C 的计算能力值分别为 P_A 、 P_B 和 P_C 。 D_A 为节点 A 上的数据块, D_B 为节点 B 上的数据块, D_C 为节点 C 上的数据块。为了使各个节点数据处理时间达到均衡,可以得到以下等式:

$$P_A \times D_A = P_B \times D_B = P_C \times D_C \quad (2)$$

$$\text{其中:} \quad D_A + D_B + D_C = D \quad (3)$$

解上述两个等式即可得到放置在各个节点上的数据块 D_A 、 D_B 和 D_C 。

2.2 资源预测方法

在 2.1 节中提出的基于节点计算能力的放置方法可以使各个节点处理本地数据的时间达到均衡,这样就可以正确地预测异构环境下作业的完成时间。因此在数据合理放置的基础上,本节提出资源预测方法以用于预测 MapReduce 作业的完成时间以及指导资源的合理分配。

假如 MapReduce 作业 J 初始化为 N^M 个 map 任务和 N^R 个 reduce 任务,并在有 S^M 个 map 任务执行单元(map slot)和 S^R 个 reduce 任务执行单元(reduce slot)的集群中运行。下面通过基于任务采样方式来分析作业 J 的完成时间。基于任务采样的基本步骤如下:

a) 从 N^M 个 map 任务中任意选择 N_s^M 个采样任务。若 $N^M < 2 \times S^M$ 那么 $N_s^M \leftarrow \alpha \times N^M$ ($0.1 \leq \alpha \leq 1.0$), 否则 $N_s^M \leftarrow S^M$ 。

b) 设定 reduce 任务的采样个数为 N_s^R 。若 $N^R < S^R$, 那么 $N_s^R \leftarrow N^R$, 否则 $N_s^R \leftarrow S^R$ 。

c) 在集群中运行作业 J 的采样任务并分别记录每个 map 任务 i 的完成时间 $T^M(i)$ 、输出数据大小 $D^M(i)$ 和每个 reduce 任务 j 在 shuffle&sort 阶段的完成时间 $T^S(j)$ 以及在 reduce 阶段的完成时间 $T^R(j)$ 和输入数据大小 $D^R(j)$ 。

d) 利用记录的数据建立 reduce 任务各阶段的输入数据大小和完成时间的函数关系,如下所示:

$$T^S(j) \leftarrow f(D^R(j)), T^R(j) \leftarrow g(D^R(j))$$

通过记录的数据,同样可以得到 reduce 任务的平均输入数据大小 D_E^R 为

$$D_E^R \leftarrow \frac{N^M \times \sum_{j=1}^{N_s^R} D^R(j)}{N_s^M \times N^R}$$

e) 预测采样任务在各阶段的平均完成时间: $T_E^M \leftarrow E(T^M(i))$ (T_E^M 表示所有 $T^M(i)$ 的平均时间), $T_E^S \leftarrow f(D_E^R)$, $T_E^R \leftarrow g(D_E^R)$ 。

由于作业 J 运行 shuffle&sort 的第一次操作与 map 阶段操作有重叠部分,本文定义 reduce 任务的 shuffle&sort 第一次操作的完成时间为 T_1^S 。

若 $N^M \leq S^M$, 那么 $T_1^S \leftarrow T_E^S$; 否则,若 $T_E^M \geq f\left(\frac{S^M \times D_E^R}{N^M}\right)$, 那么 $T_1^S \leftarrow f\left(\frac{S^M \times D_E^R}{N^M}\right)$; 否则 $T_1^S \leftarrow \left\lfloor \frac{N^M}{S^M} \right\rfloor f\left(\frac{S^M \times D_E^R}{N^M}\right) + T_E^M - T_{avg}^M$ 。

f) 预测作业 J 在 map 和 reduce 阶段的平均完成时间。

$$T_{avg}^M \approx \left\lfloor \frac{N^M}{S^M} \right\rfloor T_E^M, T_{avg}^R \approx \left\lfloor \frac{N^R}{S^R} - 1 \right\rfloor T_E^S + \left\lfloor \frac{N^R}{S^R} \right\rfloor T_E^R$$

通过上述基于任务采样操作,可以得到作业 J 在 map 阶段、shuffle&sort 阶段的第一次操作和 reduce 阶段的平均完成时间分别为 T_{avg}^M 、 T_1^S 和 T_{avg}^R 。那么作业 J 总的完成时间可表示为

$$T_{avg} \approx T_{avg}^M + T_{avg}^R + T_1^S \quad (4)$$

如果用 map/reduce 任务 (N^M, N^R) 和集群分配的资源 (S^M, S^R) 来表示作业 J 的完成时间,那么式(4)可表示为

$$T_{avg} \approx \left\lfloor \frac{N^M}{S^M} \right\rfloor T_E^M + \left\lfloor \frac{N^R}{S^R} \right\rfloor (T_E^R + T_E^S) + T_1^S - T_E^S \quad (5)$$

若用户作业有需要在时间期限内完成的服务要求,则必须解决以下性能问题:给定 MapReduce 作业 J 和输入数据集 D ,则需要分配多少个 map/reduce 任务执行单元给该作业才能满足其在时间期限 T 内完成的性能需求。首先通过简化式(5)得到

$$\frac{A_J}{S^M} + \frac{B_J}{S^R} = C_J \quad (6)$$

其中: $A_J = N^M \times T_E^M$, $B_J = N^R \times (T_E^R + T_E^S)$ 和 $C_J = T_{avg} - (T_1^S - T_E^S)$ 。

为了满足用户服务需求,式(6)会得到 map 和 reduce 任务执行单元不同解的集合。而只有当 S^M 和 S^R 之和取最小值时,

集群配置的任务执行单元的资源利用率最高。上述可建立函数关系如下所示:

$$f(S^M, S^R) = S^M + S^R \quad \text{其中: } \frac{A_J}{S^M} + \frac{B_J}{S^R} = C_J$$

利用拉格朗日乘数方法求解函数 $f(x, y)$ 的最小值为

$$S^M = \frac{\sqrt{A_J}(\sqrt{A_J} + \sqrt{B_J})}{C_J}, S^R = \frac{\sqrt{B_J}(\sqrt{A_J} + \sqrt{B_J})}{C_J} \quad (7)$$

2.3 基于 deadline 的动态调度算法

通过对异构集群中基于节点计算能力的资源放置方法和资源预测方法分析,本节提出基于 deadline 的动态调度算法。当作业 J 添加到队列时,根据 2.2 节中的式(7)来计算作业 J 在基于 deadline 为 T 内完成所需的执行资源数量的最小值 $\min M$ 和 $\min R$ 。若所需 $\min M$ 和 $\min R$ 较大时,则表示作业 J 越紧迫,作业 J 的优先级越高,排在作业队列的最前面。当系统有空闲的资源时,运行作业队列中第一个作业,如果作业中已完成的 map 任务小于应分配给作业的资源 $\min M$,则启动一个新的 map 任务。当至少有一个 map 任务执行完成时,才能启动 reduce 任务。同样地,当系统有空闲的资源时,作业中已完成的 reduce 任务小于应分配给作业的资源 $\min R$,则启动一个新的 reduce 任务。最后返回将要启动的任务队列。基于 deadline 的动态调度算法的伪代码描述如下:

```

1  M ← freeMapSlots
2  R ← freeReduceSlots
3  JobQueue 是当前作业队列
4  TaskQueue.initialize()
  //TaskQueue 是分配任务的队列集合
5  当基于 deadline 为 T 的作业 j 提交时
6  根据式(7)计算作业 j 的 map 和 reduce 任务执行单元最小值 minM 和 minR
7  在 JobQueue 中根据 (minM, minR) 的大小重新排序
8  repeat
9  fetch first Job j from JobQueue
10 if RunningMapTasksj < minM and M.size > 0
11 Task t ← job.obtainNewMapTask()
12 TaskQueue.add(t)
13 M.size --
14 end if
15 if FinishedMapTasksj > 0 and R.size > 0 and RunningReducesj < minR then
16 Task t ← job.obtainNewReduceTask()
17 TaskQueue.add(t)
18 R.size --
19 end if
20 until endOfQueue() or (M.size == 0 or R.size == 0)
21 return TaskQueue

```

3 实验结果与性能分析

本章通过实验来验证本文提出的基于 deadline 的动态调度算法,并和 Hadoop 现有的算法进行性能对比。本文通过两组实验来评价本文算法的有效性和性能。在实验 1 中,重点关注各实验用例的本地化计算任务比例,而在实验 2 中则更多关注作业的执行和资源分配情况。实验环境是由五个异构节点组成的 Hadoop 集群。在集群中,Hadoop 版本为 0.21,使用的操作系统为 Ubuntu 系统。集群中主要有三类节点:A 类节点是四核 Intel i3 CPU,主频 3.07 GHz,内存 4 GB,机器数 2 个;B 类节点是四核 Intel i3 CPU,主频 2.7 GHz,内存 4 GB,机器数 2 个;C 类节点是双核 Intel i3 CPU,主频 2 GHz,内存 2 GB,机器数 1 个。硬件配置情况如表 1 所示。

表 1 Hadoop 集群硬件配置表

节点类型	CPU	内存	机器数
A	4-core 3.07 GHz	4 GB	2
B	4-core 2.7 GHz	4 GB	2
C	2-core 2 GHz	2 GB	1

3.1 数据本地性

基于节点计算能力的数据放置策略可以减少集群中高速节点和低速节点之间的数据迁移,从而提高节点数据本地性。因此实验 1 通过运行 WordCount 和 Sort 应用程序来验证数据的本地性。实验分为两组,一组使用基于节点计算能力的数据放置方法(deadline),另一组使用默认的数据放置方法(FIFO)。在运行 WordCount 和 Sort 实验时,首先把 HDFS 中的数据块大小分别设置为 32 MB、64 MB、128 MB 和 256 MB。然后在不同数据块大小的情况下,分别运行两个应用程序 10 次,并分别计算 map 任务本地数据所占比例的平均值。实验结果如图 2 和 3 所示。

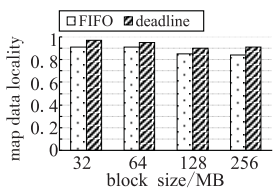


图 2 WordCount 应用中 map 任务的数据本地性比例

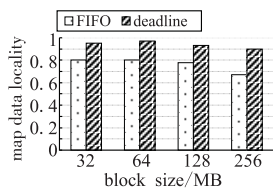


图 3 Sort 应用中 map 任务的数据本地性比例

在图 2 中,当 WordCount 应用中数据块大小为 256 MB 时,Deadline 比 FIFO 的 map 任务数据本地性比例提高大约为 23.5%。其他数据块大小为 32 MB、64 MB、128 MB 的情况下分别提高 14.3%、17.5% 和 15.1%。在图 3 中,Sort 应用中的数据块大小为 32 MB、64 MB、128 MB、256 MB 时,deadline 比 FIFO 的 map 任务数据本地性比例提高分别为 6.6%、4.8%、5.6% 和 7.6%。

3.2 MapReduce 作业执行情况

在实验 2 中,首先提交 WordCount 应用的两个实验用例,两个用例分别有不同的任务完成时间要求,实验中有足够的资源来满足作业的时间期限。连续监视作业执行进程,每 4 s 记录作业的完成情况,结果如图 4 所示。同时每 4 s 记录作业执行期间系统动态地为每个作业分配任务执行单元的数量,实验结果如图 5 和 6 所示。

图 4 表示两个 WordCount 作业的完成进程。作业 job1 在 0 s 时提交,且完成时间期限为 240 s,因此 job1 需在 D1(240 s)内完成。同样作业 job2 是在 40 s 时提交,且完成时间期限为 120 s,因此 job2 需在 D2(160 s)内完成。在作业提交后,由于数据块的初始化和分配会导致作业在执行前有 30 s 的延迟。作业 job1 在 0 s 提交后,系统会自动执行其任务。在 40 s 时,作业 job2 提交,系统并不会立即执行 job2,因为它必须等到 job1 至少有一个任务空闲时才执行。当 job2 开始执行时,系统会收集要处理数据的信息来估计 job2 的完成时间,然后系统会根据其预测值来分配可用资源给 job2。在图 5 中可以看到,系统为了保证预测值的准确性,它会一直调整所有作业的资源分配,直到作业完成。最后 job1 和 job2 都能在规定的时间期限内完成。

图 5 表示每个作业的任务执行单元的分配情况。在 job1 提交时,调度器会把所有的资源分配给 job1,用于执行其任务。

当 job2 提交后,由于 job2 的时间期限比 job1 的时间期限小,说明 job2 比 job1 拥有更高的权限,job2 将会得到比 job1 更多的资源来执行任务。系统在作业执行过程中会根据作业的执行情况进行合理的资源分配,在 job2 完成后,job1 只得到部分的资源执行而不是全部的资源。这说明本文提出的调度方法能动态地为每个作业分配合适的任务执行单元。

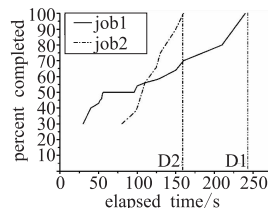


图 4 作业执行进程

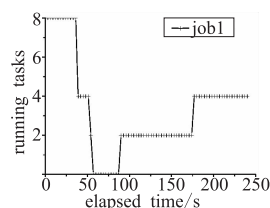


图 5 作业 job1 执行中资源分配情况

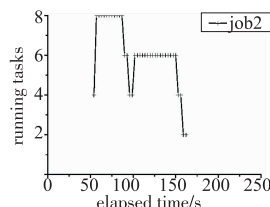


图 6 作业 job2 执行中资源分配情况

4 结束语

本文讨论了异构环境下 MapReduce 调度机制关于数据放置和资源预测的相关问题,分析了目前对这些问题的研究方法,并针对已有方法中出现的不足,本文创新地提出一种充分考虑数据本地性的动态调度机制。该机制的创新性在于充分考虑了异构环境下数据放置和资源自动分配的问题,根据数据划分粒度和任务运行情况,自动选择合适的任务调度策略。通过实验可以看出,在异构环境下基于 deadline 的动态调度算法能够有效地提高 map 任务的数据本地性比例,同时能够根据用户的性能要求动态地分配资源并在时间期限内完成任务。

参考文献:

- [1] BRYANT R E. Data intensive supercomputing: the case for DISC, CMU technical report CMU-CS-07-128 [R]. Pittsburgh: Department of Computer Science, Carnegie Mellon University, 2007.
- [2] PAVLO A, PAULSON E, RASIN A, et al. A comparison of approaches to large-scale data analysis [C]//Proc of SIGMOD International Conference on Management of Data. New York: ACM Press, 2009: 165-178.
- [3] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large clusters [C]//Proc of the 6th Conference on Operating Systems Design & Implementation. Berkeley: USENIX Association, 2004: 137-150.
- [4] Apache Hadoop [EB/OL]. [2009-03-06]. <http://hadoop.apache.org/>.
- [5] 李建江, 崔健, 王聘, 等. MapReduce 并行编程模型研究综述 [J]. 电子学报, 2011, 39(11): 2635-2642.
- [6] RAO B T, SRIDEVEI N V, REDDY V K, et al. Performance issues of heterogeneous Hadoop clusters in cloud computing [J]. Global Journal Computer Science & Technology, 2011, 11(8): 81-87.
- [7] 陈全, 邓倩妮. 异构环境下自适应的 MapReduce 调度 [J]. 计算机工程与科学, 2009, 31(A1): 168-171, 175.
- [8] ZAHARIA M, KONWINSKI A, JOSEPH A D, et al. Improving MapReduce performance in heterogeneous environments [C]//Proc of the 8th USENIX Conference on Operating Systems Design and Implementation. Berkeley: USENIX Association, 2008: 29-42.

小为 69 378 979 个条目), 路径树方法在其他所有数据集都无法运行。可以看出, 在所有的情况下, RIABG 处理的效果比深度搜索方法要好。它的查询时间比深度搜索快 3 ~ 15 倍。事实上, 在平均密集度为 10 的 rand10m10x 数据集上, 深度搜索不能在 20M ms 的限制时间内完成任务。于是得到结论: RIABG 对于大型图来说是具有扩展可达性的索引方法, 尤其在密集度增加的过程中体现得更明显。

表 13 大型合成图的伸缩性测试数据

size	deg	构建时间/ms	查询时间/ms		索引大小
		RIABG	RIABG	DFS	RIABG
rand10m	2	128796	187.2	577.6	100 M
	5	226671	5823.9	90505	100 M
	10	407158	1415296.1	— (t)	100 M
rand100m	2	1169601	258.2	762.7	800 M
	5	1084848	20467	131306	400 M

4 结束语

本文提出一种简单的大图索引方案 RIABG, 基于随机多区间标记理论, 可以快速解决大型图系统中的可达到性问题。RIABG 拥有线性建立时间复杂度和线性索引大小的空间复杂度, 同时它的查询时间可以从连续的变成针对每个查询的线性时间。实验证明, 对于大型图系统, 这些已存在的索引方法都不具有很好的扩展性, RIABG 比已存在的所有方法的性能都好, 下一步工作将把 RIABG 推广到动态图中。

参考文献:

- [1] AGRAWAL R, BORGIDA A, JAGADISH H V. Efficient management of transitive relationships in large data and knowledge bases[J]. *ACM SIGMOD Record*, 1989, 18(2): 253-262.
- [2] JAGADISH H V. A compression technique to materialize transitive closure[J]. *ACM Trans on Database Systems*, 1990, 15(4): 558-598.
- [3] CHEN Yang-jun, CHEN Yi-bin. An efficient algorithm for answering graph reachability queries [C]//Proc of the 24th International Conference on Data Engineering. 2008: 893-902.
- [4] JIN Ruo-ming, XIANG Yang, RUAN Ning, et al. Efficient answering reachability queries on very large directed graphs[C]//Proc of ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2008: 595-608.
- [5] SCHENKEL R, THEOBALD A, WEIKUM G. HOPI: an efficient connection index for complex XML document collections[C]//Proc of International Conference on Extending Database Technology. 2004: 237-255.
- [6] CHENG Jie-feng, YU J X, LIN Xue-min, et al. Fast computing reachability labelings for large graphs with high compression rate [C]//Proc of the 11th International Conference on Extending Database Technology: Advances in Database Technology. New York: ACM Press, 2008: 193-204.
- [7] KROMMIDAS I, ZAROLIAGIS C. An experimental study of algorithms for fully dynamic transitive closure[J]. *Journal of Experimental Algorithmics*, 2008, 12(16): 544-555.
- [8] WANG Hai-xun, HE Hao, YANG Jun, et al. Dual labeling: answering graph reachability queries in constant time[C]//Proc of the 22nd International Conference on Data Engineering. 2006: 75.
- [9] CHEN Yang-jun. General spanning trees and reachability query evaluation [C]//Proc of the 2nd Canadian Conference on Computer Science and Software Engineering. New York: ACM Press, 2009: 243-252.
- [10] BOUROS P, SKIADOPOULOS S, DALAMAGAS T, et al. Evaluating reachability queries over path collections[C]//Proc of SSDBM. 2009: 4-6.
- [11] JIN Ruo-ming, XIANG Yang, RUAN Ning, et al. 3-HOP: a high-compression indexing scheme for reachability query [C]//Proc of ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2009: 813-826.
- [12] COHEN E, HALPERIN E, KAPLAN H, et al. Reachability and distance queries via 2-HOP labels[J]. *SIAM Journal of Computing*, 2003, 32(5): 1335-1355.
- [13] SCHENKEL R, THEOBALD A, WEIKUM G. Efficient creation and incremental maintenance of the HOPI index for complex XML document collections [C]//Proc of the 21st International Conference on Data Engineering. Washington DC: IEEE Computer Society, 2005: 360-371.
- [14] RODITY L, ZWICK U. A fully dynamic reachability algorithm for directed graphs with an almost linear update time[C]//Proc of the 36th Annual ACM Symposium on Theory of Computing. 2004: 184-191.
- [15] DEMETRESCU C, ITALIANO G. Dynamic shortest paths and transitive closure: algorithmic techniques and data structures [J]. *Journal of Discrete Algorithms*, 2006, 4(3): 353-383.
- [16] BRAMANDIA R, CHOI B, NG W K. On incremental maintenance of 2-HOP labeling of graphs[C]//Proc of the 17th International Conference on World Wide Web. 2008: 845-854.
- [17] HE Hao, WANG Hai-xun, YANG Jun, et al. Compact reachability labeling for graph-structured data[C]//Proc of the 14th ACM International Conference on Information and Knowledge Management. New York: ACM Press, 2005: 594-601.
- [18] TRISSL S, LESER U. Fast and practical indexing and querying of very large graphs[C]//Proc of ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2007: 594-601.
- [9] GUO Lei-tao, SUN Hong-wei, LUO Zhi-guo. A data distribution aware task scheduling strategy for MapReduce system [C]//Proc of the 1st International Conference on Cloud Computing. 2009: 694-699.
- [10] POLO J, CARRERA D, BECERRA Y, et al. Performance-driven task co-scheduling for MapReduce environments [C]//Proc of the 12th IEEE/IFIP Network Operations and Management Symposium. Piscataway: IEEE Press, 2010: 373-380.
- [11] KC K, ANYANWU K. Scheduling Hadoop jobs to meet deadlines [C]//Proc of the 2nd IEEE International Conference on Cloud Computing Technology and Science. 2010: 388-392.
- [12] VERMA A, CHERKASOVA L, CAMPBELL R. Resource provisioning framework for MapReduce jobs with performance goals [C]//Lecture Notes in Computer Science, vol 7049. Berlin: Springer-Verlag, 2011: 165-186.
- [13] WOLF J, RAJAN D, HILDRUM K, et al. FLFX: a slot allocation scheduling optimizer for MapReduce workloads [C]//Proc of the 11th ACM/IFIP/USENIX Conference on Middleware. Berlin: Springer-Verlag, 2010: 1-20.
- [14] 刘鹏. 实战 Hadoop——开启通向有云计算的捷径 [M]. 北京: 电子工业出版社, 2011.
- [15] BORTHAKUR D. The Hadoop distributed file system: architecture and design [EB/OL]. (2011). http://hadoop.apache.org/hdfs/docs/current/hdfs_design.html.

(上接第 3373 页)