

用于多个 MapReduce 作业的任务调度算法

刘 涛, 武淑红, 强 彦

(太原理工大学 计算机科学与技术学院, 山西 太原 030024)

摘 要: 提出了一个可用于 workflow 服务的, 包含多个 MapReduce 作业的调度机制. workflow 应用程序具有优先级依赖约束(可以用有向无环图 DAG 表示). 同时, 对于带宽有限的低数据传输成本, 数据依赖限制应该在调度系统中. 该调度机制能够提供: 调度带有优先级约束的 MapReduce 任务; 在网络上保存数据时, 通过考虑了数据依赖限制的数据预放置策略来减少传输成本.

关键词: 云计算; 数据密集型计算; MapReduce; 任务调度

中图分类号: TP311

文献标识码: A

文章编号: 1000-7180(2013)12-0156-04

Task Scheduling Algorithm Used for Multiple MapReduce Tasks

LIU Tao, WU Shu-hong, QIANG Yan

(School of Computer Science and Technology, Taiyuan University of Technology, Taiyuan 030024, China)

Abstract: MapReduce has been widely used in data-intensive applications; it has several advantages, such as good programming interface, high availability and fault tolerance. It is suitable for distributed processing on large-scale data, for instance, network analysis, bioinformatics and high-performance computing. In shared cluster, there are many researches on scheduling issue. However, in multi-processors nodes, the scheduling method based on priority constrain which conclude multi MapReduce tasks is uncommon. This paper provides a scheduling method based on priority constrain which conclude multi MapReduce tasks. Workload applications have a priority dependency constraint (which can be represented by DAG). Meanwhile, for the limited bandwidth of the low cost of data transmission, data dependence limit should be concluded in the scheduling system. The proposed algorithm can provide: MapReduce task scheduling with precedence constraints; by considering the data dependencies restrict the data pre-placement strategy to reduce transmission costs.

Key words: cloud computing; data-intensive computing; MapReduce; task scheduling

1 引言

MapReduce 框架为数据密集型应用提供了功能上的编程模型. 并行编程中的种种复杂问题, 均由该框架负责处理. 该模型的体系结构遵从无共享结构, 每个节点上每个任务之间无关联. 简化了数据管理和容错.

如何高效地把 map-reduce 作业分配给多个共享集群, 以获取更好性能的研究有很多. 而很少有关由多个 map-reduce 任务组成的 workflow 调度机制优

化的研究.

本文提出了适用于带有优先级依赖的多个 MapReduce 作业的调度方法. 在该机制中, 考虑了两个约束: (1) 优先级约束; (2) 数据依赖. 对于多个 map-reduce 作业之间的优先级依赖, 在该方法中使用了 List 调度算法. 至于数据依赖标准, 数据预放置策略能够减少节点之间的网络成本.

2 相关工作

MapReduce 框架出现后, 研究人员对其上的作

业调度做了很多研究.一部分调度算法关注本地化问题.另一部分调度方法针对 map 和 reduce 过程中的同步问题. Hadoop 开发人员通过重新启动延迟的 map 任务,来防止一个延迟的作业影响了整体性能^[1].在共享集群中,调度的一大问题是:如何保证运行中的 map-reduce 作业之间的公平性.在本地化和公平性约束之间需要做些取舍.关于如何同时保证本地化和公平性的问题, Hadoop 中的公平调度器^[2]支持公平性约束.在该调度器中,通过牺牲小程度的公平性来获得本地性约束.该方案称为延迟调度^[3].在这种方案中,调度器并不是严格按照公平性约束来过早地把槽分配给任务,而是等待一段时间,直到满足本地化约束. Dryad^[4]属于分布式计算应用框架,它支持完全的 DAG 操作,该调度过程被看作工作流图中成本的优化问题.通过使用 Quincy 调度器, Dryad 也能够支持公平性约束,它使用基于流的调度机制.虽然 Quincy 调度器能够保证带有公平性约束的本地化,但是它把调度问题当成流网络的优化问题,较为复杂,而公平调度器中的 Delay 调度则使用了相对简单的启发式模型.

3 工作流调度约束

3.1 优先级依赖约束

有向无环图 DAG 是工作流应用程序的一个通用表示形式,这些工作流应用程序包含一系列互相之间带有优先级依赖的任务.因此, DAG 可以用来表示包含 map-reduce 任务和普通任务的工作流服务.我们可以用图 $G = (V, E)$ 来表示,其中, V 表示一组必须在共享集群中执行的 v 个任务, E 表示一组带有优先级约束的 d 有向边(节点 v_i 的权值为 w_i 表示计算成本,边的权值代表节点间的通信成本(比如数据传输成本)).在工作流图中,没有双亲的节点根节点称为进入作业,没有子节点的称为退出作业.假设在工作流图中只存在一个进入节点和一个退出节点,并且假设集群中每个节点有相同的计算能力(即同构集群).

图 1 中,圆圈 t_i 表示工作流任务,箭头表示的是优先流.

3.2 本地化和数据依赖约束

单个 map-reduce 任务中一个复杂问题是在调度中考虑本地化(带有输入数据的节点和任务分配节点之间的距离).当输入数据距离计算节点较近的时候,传输成本会比较低.在共享集群环境下,由于带宽受限,本地化是影响性能的关键问题.事实上,

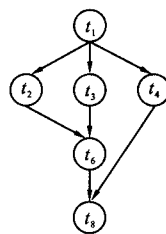


图 1 工作流应用程序的图格式

高本地化保证了任务的吞吐量.保持本地化最有效的方法是在持有输入数据的节点上处理任务,叫做节点本地化.当节点本地化不能实现的时候,在同一机架上执行任务也是可取的,这个叫做机架本地化.当本地化实现不了的时候,数据传输引起的 I/O 成本会在很大程度上影响性能.大部分 mapreduce 作业调度算法都有这样的策略,即就近分配任务,减少输入数据和目标节点之间的传输成本.本地化被看作是作业调度的基本准则.

在工作流应用程序中,多个 mapreduce 任务上的调度有更加复杂的限制.例如,一些任务可能依赖于相同的数据集.或者,一部分数据集可能被工作流中大部分 mapreduce 任务所引用,而剩下的数据集可能和 map-reduce 任务之间不存在任何引用.从数据传输成本的影响到整体的性能,在工作流应用程序中,高效的数据放置策略在数据传输过程中必不可少.因此,应该对工作流中多个任务之间的数据依赖准则加以考虑.高效的数据放置策略能够最小化机架之间或者数据中心之间的数据传输成本,通过这样的策略,本地化也可以实现.

数据依赖限制可以表示成图的格式如图 2 所示.工作流中的许多任务会把某些特定的数据集作为输入文件.这种类型的数据应该尽可能多地放置在多个机架中,以在工作流执行之前最小化数据传输成本.通过数据集的受欢迎程度,就可以决定把共享集群中的数据集放置在哪里.此外,任务执行之前,在哪里放置数据集也是一个关键问题,它和本地化约束紧密相关.在工作流程序执行过程中,离输入数据集近的任务性能更好.为了尽量减少机架或者数据中心之间传输数据集对性能的影响,就必须考虑属于预放置策略,准确的数据集放置策略能够减少数据传输成本.关于数据的预放置策略将在下一部分讨论.

4 List 调度方法

本文采用 List 调度方法. List 调度可以被分为两个阶段:任务排序阶段和节点选择阶段.

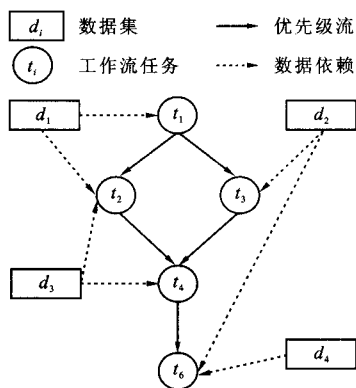


图2 任务图的数据依赖关系

4.1 任务排序阶段

在该阶段,每个任务的优先级由以下的递归等级方程决定,

$$\text{rank}(t_i) = 1 + \max_{t_j \in \text{succ}(t_i)} (\text{rank}(t_j)) \quad (1)$$

其中, $\text{succ}(t_i)$ 表示任务 t_i 的后继(子任务)集合. 具有较高等级评分的任务执行的优先级也较高. 一个工作流的退出等级评分为 0, 定义如下:

$$\text{rank}(t_{\text{exit}}) = 0 \quad (2)$$

等级函数反应了一个任务工作流图中的优先级约束. 如果任务的等级评分相同, 执行的优先权将给那些拥有更多子任务的任务, 来增加任务运行的并发数. 如果拥有相同等级评分的任务, 它们的子任务数也相同, 就随机给出优先级. 即使任务具有相等距离的入口任务, 它们的等级评分也可能不同. 到出口任务的距离决定了任务的等级评分. 例如, 图 1 中 t_6 是退出任务, 其等级评分为 0. 因此, t_5, t_4, t_3, t_2, t_1 的等级评分分别为 1, 1, 2, 2, 3.

4.2 节点选择过程

在给工作流应用程序的一系列任务评完分后, 每次调度器都将选择最高等级评分的任务, 然后为之分配资源. 由于等级函数反应了工作流应用程序中的优先级约束, 调度器才得以保证优先级依赖约束.

调度器的分配顺序与 Hadoop 的 JobTracker 机制基本相似. 处理节点由本地化约束决定. 如果节点本身拥有任务需要的数据, 调度器就会把任务分配给该节点, 这个叫做节点本地化. 如果不满足上述假设, 调度器就会把任务分配给同一机架的节点, 这个叫做机架本地化.

如果节点没有任务需要的数据, 调度器将通过网络把任务分配给带有数据集副本的节点. 因此, 节点选择过程最优选节点本地化, 其次选机架本地化.

5 数据预放置

由于数据密集型应用程序中的待处理的数据规模很大, 所以在带宽有限的情况下, 数据的移动成本会对 map-reduce 作业的整体性能造成很大的影响. 为了在工作程序执行过程中最小化数据的移动, 就必须提出一个有效的数据放置策略. 数据预分配上的创新是在不同机架或者数据中心一组节点之间最小化传输成本. 因此, 对于数据放置的问题, 有两个问题需要解决: 一是为一组数据集提供多少份冗余, 二是决定在哪儿存放数据集.

对于第一个问题, Hadoop 在分布式文件系统中使用三份备份, 其中两份在同一机架, 另外一份在另外机架的节点上. 在上述安排中, 在不同的机架上放置额外备份的目的是容错. 在本文提出的方法中, 在分布式文件系统中为不是热点的数据集设置了有四份备份, 其中两份在同一组的节点上(在同一机架), 另外两份放置在其他组的节点上. 为那些不是全局或当期使用的数据集准备四分备份, 能够在增加吞吐量的同时减少容错可能引起的问题. 而热点数据则被全局地放置在分布式文件系统中.

第二个问题, 如何对数据集分组、在哪儿存放数据集是个很重要的问题. 许多任务共享相同的数据集, 这些数据集又能在同一集合中进行分组. 因此, 数据存放依赖于任务集使用的数据集. 同时, 也要考虑工作流中执行任务的并发性. 比如, 如果许多在工作流应用程序中并发执行的任务均参照同样的输入数据集, 那么每一组节点中都应该防止该数据集. 我们提出了以下的数据凝聚力得分概念和数据预防治策略.

5.1 数据凝聚力得分

数据凝聚力得分代表在被任务 T_i 使用的数据集(D_{Ti})和驻留在每组节点(G_j)上的数据集(D_j)之间, 存在多少公共的数据集. 即数据凝聚力得分是 D_{Ti} 和 D_j 之间公共的数据集. 其关系如下:

$$\text{Cohesion}(T_i, D_j) = \text{count}(D_{Ti} \cap D_j) \quad (3)$$

如果该值比其他组节点上的数据集的得分高, 就把数据集 D_{Ti} 放置到 D_j 上, 这样更能减少数据传输成本. 因此, 任务 T_i 更适宜在数据凝聚力得分最高的那组节点上执行, 以节省数据传输成本.

在放置数据集时, 并发性问题也需要考虑. 如果不同任务的等级得分一致, 具有相同等级得分的不同任务并发的可能性就会非常高. 本文提出的数据放置算法、凝聚力得分和任务集的并发性将在以下

算法中加以考虑。

5.2 数据预放置算法

在以下算法中,数据集分布式地分配到组 G_j 的节点中,具有相同等级的任务的数据集被分布式地存放在不同组的节点中(在相同等级得分允许的情况下,同一组的 T_i 和 T_j 之间没有数据集上的重叠),这么做增加了相同等级任务并发的可能性。数据集有两份冗余可供每个任务使用,两份冗余的位置由以下算法决定。每个数据集第一份冗余的位置由凝聚力得分函数决定。如果拥有相同等级评分得分任务的数据集之间不存在重叠,数据集就会被分配到凝聚力得分最高的位置上。在任务的所有数据集的第一个位置都决定好之后,第二份冗余会被分配到凝聚力得分第二高的位置上。维护第二个位置的数据集是为了通过允许更多的节点执行任务,来提高并发性和容错性。通过使用凝聚力得分函数来考虑任务共同的数据集,就能够分配数据、决定数据集的放置位置。

输入: R_k : 带有等级 k 的一组任务

输出: D_j : 每组节点(G_j) 划分好的数据集

begin

$k = \text{root rank}$

// 第一步: 为每个数据集分配第一份冗余

For (each set of tasks with rank k , R_k) // 降序排列

root rank

For (each task $T_i \in R_k$)

For (each data set D_j residing in G_j)

$C_j = \text{Cohesion}(T_i, D_j)$

$C_f(T_i) = \text{the highest cohesion score from } D$

($f = \text{location to allocate the first replica, no}$

overlapping with other tasks in R_k)

// 把由任务 T_i 使用的第一份冗余分配给每组节点 G_j

$D_f \leftarrow DT_i$ (the data set used by the task T_i)

// 第二步: 为每个数据集分配第二份冗余

For (each set of tasks with rank k , R_k) // 降序排列

root rank

For (each task $T_i \in R_k$)

For (each data set D_j residing in G_j)

$C_j = \text{Cohesion}(T_i, D_j)$

$C_s(t_i) = \text{the second highest cohesion score from } D$

($s = \text{location to allocate the second replica, no}$

overlapping with other tasks in R_k)

and $f \neq s$ (first allocated group $\neq$ second group)

// 把由任务 T_i 使用的第二份冗余分配给每组节点 G_j

$DS \leftarrow DT_i$ (the data set used by the task T_i)

6 结束语

本文提出了适用于带有优先级依赖的多个 MapReduce 作业的调度方法。该方法采用 List 调度方法以有向无环图(DAG)的形式来解决优先级约束。同时该机制能够处理任务集之间的数据依赖约束。通过使用数据放置策略,该机制能够减少共享集群中数据传输的成本。在这个数据放置策略中,数据集会通过使用凝聚力得分来对其进行分类,以此来提高并发性和容错能力。

参考文献:

- [1] Zaharia M, Kowinski A, Joseph A, et al. Improving mapreduce performance in heterogeneous environments[C]// Proceedings of the 12th IEEE International Symposium on cluster, cloud and Grid Computing ottauxa, ON; IEEE, 2008;714-716.
- [2] Apache Software Foundation. FairScheduler[EB/OL]. [2012 -04-20]. http://hadoop.apache.org/mapreduce/docs/r0.21.0/fair_scheduler.html
- [3] Zaharia M, Borthankur D, Sarma J, et al. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling[C]// Proceedings of the EuroSys. ACM, New York; ACM, 2010;265-278.
- [4] Isard M, Prabhakaran V, Currey J, et al. Quincy: fair scheduling for distributed computing clusters[C]// Proceedings of the 22nd ACM, Symposium on Operating systems principles. [s. l.]: ACM, 2009.
- [5] Tomazs Nykiel. MRShare: Sharing Across Multiple Queries in MapReduce[J]. Proceedings of the VLDB Endowment. 2010;3(1-2):494-505.
- [6] Dawei Jiang, Beng Chin Ooi, Lei Shi, et al. The Performance of MapReduce: An In-depth Study[J]. Proceedings of the VLDB Endowment, 2010,3(1-2):472-483.
- [7] Xiaohong Zhang, Zhiyong Zhong, Shengzhong Feng, et al. Improving data locality of mapReduce by scheduling in homogeneous computing environments [J]. Ninth IEEE International Symposium on Parallel and Distributed Processing with Applications, 2011(14): 120-126.

作者简介:

刘涛 男,(1988-),硕士.研究方向为云计算。

武淑红 女,(1969-),副教授.研究方向为智能信息处理。

强彦 男,(1969-),副教授.研究方向为云计算。