CN 43-1258/TP
ISSN 1007-130X

计算机工程与科学
Computer Engineering & Science

第 36 卷第 12 期 2014 年 12 月
Vol. 36, No. 12, Dec. 2014

文章编号:1007-130X(2014)12-2286-10

基于排队网络的多优先级 MapReduce 作业调度算法*

万　聪,王翠荣,王　聪,吕艳霞,贾　朔
(东北大学信息科学与工程学院,辽宁 沈阳 110819)

摘　要:MapReduce 是一个能够对大规模数据进行分布式处理的框架,目前被各个领域广泛应用。在提供 MapReduce 服务的集群中,如何保证不同优先级用户的截止时间限定是 MapReduce 作业调度问题的一个挑战。针对这一问题,提出了一个基于排队网络的多优先级作业调度算法(MPSA)。首先分析和归纳了基于 MapReduce 模型的算法,提出了三种常见模式,采用 Jackson 排队网络对基于 MapReduce 模型的算法建立了数学模型,应用该网络模型可以求出不同优先级队列对资源的需求;随后使用 AR(1)模型进行预测,使算法可以动态地适应不同的用户访问量;利用二分查找算法,分步计算出不同优先级在 map 阶段和 reduce 阶段分配的槽位数;最后实现了在 MapReduce 模型中应用的实时调度算法。实验结果表明,与传统的 FIFO 和公平调度算法相比,本文提出的算法在用户到达率和任务规模变化的情况下,可以更加有效地满足不同优先级用户的截止时间限定。

关键词:云计算;排队网络;MapReduce;调度算法
中图分类号:TP312　　**文献标志码**:A
doi:10.3969/j.issn.1007-130X.2014.12.008

A MapReduce scheduling algorithm supporting multiple priorities based on queuing network

WAN Cong, WANG Cui-rong, WANG Cong, LÜ Yan-xia, JIA Shuo
(College of Information Science and Engineering, Northeastern University, Shenyang 110819, China)

Abstract: MapReduce is a distributed computing framework for big data processing, which has been widely used in various fields. It's a challenge to ensure the deadline of different priority users in the cluster providing MapReduce services. To solve this problem, a queuing network based multi-priority scheduling algorithm (MPSA) is proposed. Firstly, the MapReduce based algorithms are summarized and analyzed, three common patterns are proposed, and the Jackson queuing network is used to build a mathematic model of the MapReduce based algorithms. The mathematic model can be used to find the resource demands of different priority queues. Secondly, the AR(1) model is used to predict the numbers of accessing users, and the binary search algorithm is used to calculate the assigned slot numbers of different priority users in map phase and reduce phase. Finally, a real time scheduling algorithm running in the MapReduce framework is implemented. Experimental results show that, compared with the traditional FIFO and fair scheduling algorithm, the proposed scheduling algorithm can ensure the defined deadlines of different priority users more effectively when the user arrival rates and the task scales change.

Key words: cloud computing; queuing network; MapReduce; scheduling algorithm

* 收稿日期:2014-08-29;修回日期:2014-11-06
基金项目:国家自然科学基金资助项目(61300195);辽宁省教育厅科学研究一般资助项目(L2013099);中央高校基本科研业务费资助项目(N110323009)
通信地址:066000 河北省秦皇岛市东北大学秦皇岛分校计算机与通信工程学院
Address: College of Computer and Communication Engineering, Northeastern University at Qinhuangdao, Qinhuangdao 066000, Hebei, P. R. China

1　引言

MapReduce[1]是 Google 公司提出的一种用于处理大规模数据集并行计算的技术框架,被广泛应用于数据挖掘、机器学习、日志分析以及搜索引擎等领域。目前有多个系统实现了 MapReduce 模型,包括斯坦福大学的 Phoenix[2]和 Google 自己的系统等,其中最有影响力的是由 Apache 基金会开发的 Hadoop[3]。不同于传统的并行计算模型,如 MPI[4],MapReduce 主要应用于数据密集型计算,它的设计目标是服务于那些能在几分钟或者几小时内完成的作业。MapReduce 一般被部署在用高速局域网络连接的单一数据中心内,为研究机构或者企业提供数据处理服务。随着越来越多的传统算法被 MapReduce 化,MapReduce 服务的使用频率也在提高,如何为用户提供一个满意度较高的服务是 MapReduce 作业调度研究中的热点问题。

目前,很多科研机构和公司提出了一些调度算法并应用在 Hadoop 系统中,Apache 组织为 Hadoop 提供了一个遵守先进先出 FIFO(First In First Out)规则的调度算法,该算法首先按照作业的优先级高低,再按照提交时间的先后选择一个作业。Facebook 公司提出了 Fair 调度算法[5],用户从多个资源池中获得资源,可以为小规模的作业提供相对公平的资源竞争机制。Yahoo 公司提出了 Capacity[6]调度器,可以支持多个用户组,为每个用户组保证最低资源量,所有用户组共享其他的资源。伯克利大学提出了 LATE 算法,该算法可以检测出落后任务,当一个任务的进度落后于同批次任务进度的 20%时,则把该任务当做落后任务,并认为该任务会拖延作业的完成时间,从而为它启动一个备份任务。LATE 算法[7]在异构环境下可以有效地提高 MapReduce 的工作效率,但是会占用很多额外的资源来启动备份任务,而且在任务的本地化和任务进度推测方面也表现一般。在文献[8]提出的调度算法中,按照用户出价的高低来决定分配该用户的作业资源的比例,愿意多付费的用户可以得到更多的资源。文献[9]将 MapReduce 中的作业按照用户的需求分为实时和非实时两类,算法会首先努力满足实时作业的需求。另外,文献[10,11]将作业运行时间和花费作为优化目标来进行作业调度,应用智能算法和博弈论来获得一个优化的调度方案。

另外,对 MapReduce 的应用研究也是一个热点,*k*-means、isodata 和 pagerank 等常用算法已经被移植到 MapReduce 平台[12~14];在网络安全[15]、图像处理[16]等领域也都有广泛的应用。

在实际应用过程中,控制不同优先级用户的 QoS 已经成为迫切的需求。而用户到达速率的变化、不同优先级用户的访问量、单个作业的平均处理时间等等因素都会对服务质量造成影响。目前已有的调度算法或者不对用户进行区分,或者难以根据动态的用户访问情况划分资源。针对目前已有方法的局限性,本文提出了一个多优先级的调度算法 MPSA(Multi-Priority Scheduling Algorithm),保证不同优先级的用户在系统中有不同的逗留时间。本文分析了基于 MapReduce 的算法,总结出三种常见模式;将 map 阶段和 reduce 阶段分别抽象成一个服务站,用 Jackson 排队网络进行建模,求出不同优先级用户对资源的需求;使用 AR(1)模型预测了用户访问量和平均工作量;使用二分查找算法计算出分配给不同优先级用户的槽位数;提出了一个实时调度算法。最后,通过模拟环境和实际环境的实验验证了算法的性能。

2　MapReduce 模型

2.1　工作流程

使用 MapReduce 模型处理数据的过程如下:

(1)首先将需要处理的数据集分割成分片(split),每个分片的大小可以通过程序进行调整,默认为 64 MB。同时,这些分片会被上传到部署了 MapReduce 框架的集群,分布存储在各个 slave 节点中。

(2)对每个分片启动一个 map 任务,这些 map 任务都是并发执行的,一次启动的最大任务数量可以通过配置的参数决定。在 map 任务中运行用户自定义的 map 函数来处理分片。map 函数的输出被称为中间数据,中间数据以〈*key*, *value*〉对的形式存在。

(3)MapReduce 使用分区函数将 *key* 值分配给不同的 reduce 任务,拥有相同 *key* 值的〈*key*, *value*〉对会被发送给同一个 reduce 任务。map 任务产生的〈*key*, *value*〉对会被保存到节点本地的硬盘中,等待任务结束后按照 *key* 值分别发送给 reduce 任务。这个过程在 MapReduce 中被称为混洗(Shuffle),混洗过程相当于一次全局的同步工作,也是算法执行过程中,不同任务进程之间唯一的通信。

(4)当所有 map 任务都结束之后,reduce 任务开始执行,reduce 任务的数量由用户在程序中设置。Reduce 任务对所有〈*key*, *value*〉对聚合在一起,按照 *key* 值进行排序,作为用户定义的 reduce 函数的输入。

(5)每个 reduce 任务完成之后都会将执行结果写回到分布式文件系统中,当所有 reduce 任务都完成之后,整个作业执行完毕。最后,用户可以从分布式文件系统中得到作业输出的结果。

2.2 分布式文件系统

分布式文件系统是分布式计算中数据存储管理的基础,也是 MapReduce 框架的一部分,例如,在 Google 的 MapReduce 实现中包括了 GFS (Google File System),在 Hadoop 中包括了 HDFS (Hadoop Distributed File System)。这些文件系统有以下特点:首先,它们认为故障的发生是一种常态,而不是异常。文件系统由大量的服务器组成,每个服务器都随时可能发生故障,因此,容错和自动恢复都是必备的功能。第二,文件系统存储的数据集一般都数据量庞大,所以需要支持大文件并提供较高的 I/O 吞吐率。第三,假定不存在对于某个文件的随机写操作,这个假设对运行在文件系统上的应用程序类型有所限制。

2.3 MapReduce 程序模式

MapReduce 模型将程序处理的过程分为 map 和 reduce 两个阶段,但是在实际应用中存在多种使用方法,图 1 描述了基于 MapReduce 算法的三种常见模式:简单模式、迭代模式和链模式。

2.3.1 简单模式

在 MapReduce 框架中,map 任务之间或者 reduce 任务之间没有消息传递。只在两个阶段之间有一次全局性的同步。如图 1a 所示,简单模式分为两个阶段,首先执行一次 map 函数,然后再执行一次 reduce 函数就完成作业。在执行作业过程中最多需要一次数据同步的算法,都可以归为简单模式。一些简单的算法,例如 wordcount 等,就属于简单模式。

2.3.2 迭代模式

如图 1b 所示,迭代模式是简单模式的多次重复执行。每次迭代过程中 map 函数与 reduce 函数的算法都是相同的,上次迭代的输出数据作为下次迭代的输入数据。在数据挖掘和机器学习领域,很多算法都应用到了迭代模式,例如 *k*-means、pagerank 算法等。

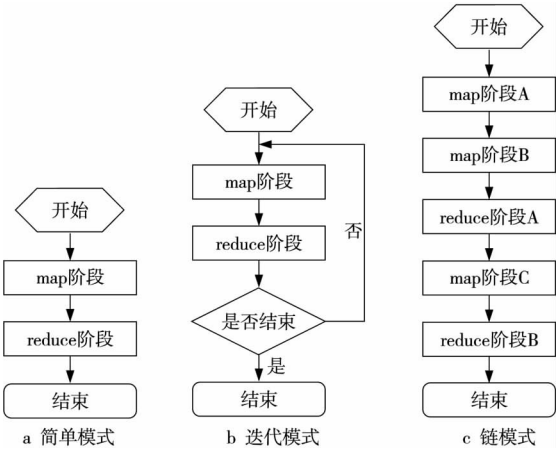

Figure 1 Three patterns of MapReduce

图 1 基于 MapReduce 算法的三种模式图

2.3.3 链模式

在 MapReduce 中,用户以 job 的形式提交作业。一般来说,每个 job 由一个 map 函数和一个 reduce 函数组成。但是,在实际应用中,往往算法的逻辑比较复杂,因此 MapReduce 还支持在一个 job 中增加多个 map 函数,这些 map 函数可以在 reduce 函数之前执行,也可以在 reduce 函数之后执行。另外,在面对更加复杂的算法时,一次数据同步难以满足需求,一个算法需要用多个 job 来实现。因此,如图 1c 所示,链模式就是顺序地执行多个 map 函数和 reduce 函数的过程。在这里,链模式包括两个范畴:第一,在一个 job 中包含 $N(N\geqslant 2)$个 map 函数和一个 reduce 函数的算法;第二,由多个 job 顺序组成的程序。

由多个 job 组成的工作流也是常见的情况,组成工作流的 job 相互之间具有依赖关系。如图 2a 所示是一个有向无环图(DAG),表示一个由四个 job 组成的工作流,job1 的输出作为 job2 和 job3 的输入,job2 和 job3 的输出作为 job4 的输入。我们可以对这个 DAG 进行拓扑排序,把它转化成一个链式的结构,如图 2b 所示。通过拓扑排序,具有依赖关系的工作流也可以用链模式来表示。

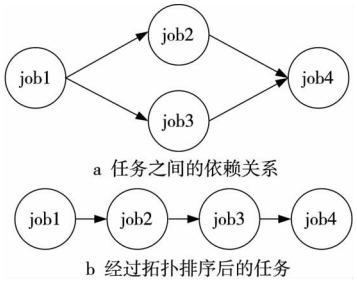

Figure 2 An example of workflow

图 2 工作流处理示例图

3　MapReduce 模型

3.1　工作流程

根据前文提出三个常见模式,我们使用 Jackson 排队网络模型对 MapReduce 模型进行建模。在建模前,本文定义了如下规则:

(1)所有的用户服从先来先服务(FCFS)原则,前一个用户离开服务站,下一个用户才能进入。

(2)所有的 map 槽位(slot)被合并作为 map 服务站,所有的 reduce 槽位(slot)合并为 reduce 服务站,统一考虑。在每个服务站内部,对用户的作业进行并行处理。

(3)map 阶段和 reduce 阶段是 MapReduce 框架中最核心的两个部分,在本文中只考虑这两个阶段的工作。

(4)每个用户提交一个作业,如果提交多个作业视为多个用户,则用户的数量是无限的,用户的到达服从泊松分布。

(5)队列中等待的用户数量没有上限,不存在被丢弃的用户。

网络模型如图 3 所示,图中圆圈代表服务站,长方形代表队列。用户从外部到达的速率为 λ,用户离开 map 服务站之后以概率 q_{12} 进入 reduce 服务站;另外还有可能完成服务,离开系统,这种可能性的概率为 q_{10};还有可能再次进入 map 服务站,进行下一轮 map 任务,这种可能的概率为 q_{11}。用户离开 reduce 服务站之后以概率 q_{20} 直接离开系统结束服务;还有可能返回 map 服务站,这种情况的概率为 q_{21}。综合以上情况,有三类用户会进入 map 队列:(1)外来用户,速率为 λ;(2)从 map 服务站出站的用户,速率为 λ_2;(3)从 reduce 服务站出站的用户,速率为 λ_3。进入 reduce 队列的用户只有从 map 服务站出站的部分用户,速率为 λ_4。

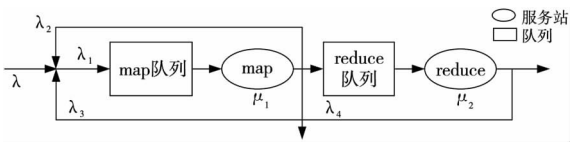

Figure 3　Queuing network model

图 3　排队网络模型图

3.2　到达率计算

在稳定状态下,用户从两个服务站离开的速率应该等于到达的速率。用 λ_1 来表示用户进入 map 服务站的速率,那么离开 map 服务站的速率也是 λ_1。而进入 reduce 服务站的速率为 λ_4。

根据泊松过程的合成定理[17]:

定理 1　泊松过程的合成。设 $n1(t)$ 与 $n2(t)$ 分别为具有参数为 λ_1、λ_2 且相互独立的两个泊松过程,则 $n1(t)+n2(t)$ 是参数为 $\lambda_1+\lambda_2$ 的泊松过程。

按照合成定理,本文给出如下推论:

推论 1　map 队列的用户到达过程是由三个泊松过程合成的泊松过程,参数为 $\lambda+\lambda_2+\lambda_3$。

因此,我们可以列出方程组,如公式(1)所示,来计算 map 队列和 reduce 队列的用户到达速率。

$$\begin{cases}\lambda_1 = \lambda+\lambda_2+\lambda_3\\ \lambda_2 = \lambda_1 q_{11}\\ \lambda_4 = \lambda_1 q_{12}\\ \lambda_3 = \lambda_4 q_{22} = \lambda_1 q_{12} q_{22}\end{cases} \tag{1}$$

计算方程组,可以得到:

$$\begin{cases}\lambda_1 = \dfrac{\lambda}{1-q_{11}-q_{12}q_{21}}\\ \lambda_2 = \dfrac{q_{11}\lambda}{1-q_{11}-q_{12}q_{21}}\\ \lambda_3 = \dfrac{q_{12}q_{21}\lambda}{1-q_{11}-q_{12}q_{21}}\\ \lambda_4 = \dfrac{q_{12}\lambda}{1-q_{11}-q_{12}q_{21}}\end{cases} \tag{2}$$

3.3　模型求解

Jackson 网络拥有 $M(M\geqslant 1)$ 个节点,每个节点都具有相互独立的负指数分布的服务时间,从外部到达系统中任何节点的输入都是独立的泊松到达过程。$m/m/1$ 模型是一种用户按照速率为 λ 的泊松过程到达,服务时间为独立分布随机变量的排队系统。

按照 Jackson 定理[18],可以将排队网络的每个节点都进行独立分析。

Jackson 定理　对于一个 Jackson 网络,稳定状态下到达节点 i 的速率为 λ_i,则:

(1)在任何节点的用户数量与其他任何节点的用户数量无关;

(2)节点 i 所表现的随机行为就好像它接受的是速率为 λ_i 的泊松到达。

通过 Jackson 定理本文可以得到以下引理:

引理 1　map 节点和 reduce 节点都可以用独立的 $m/m/1$ 模型来分析。

引理 2　满足所有 map 节点和 reduce 节点的局部平衡方程的解满足全局平衡方程。

根据排队网络模型,每个节点都可以得到如图 4 所示的状态转移图,其中状态 n 代表服务站 i 中

有 n 个顾客。当 n 大于 0 的时候,服务站繁忙;当 n 等于 0 时,服务站空闲。

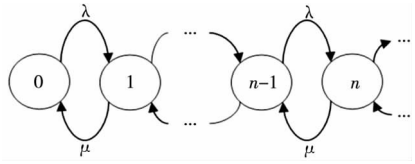

Figure 4 State transition diagram

图 4 状态转移图

设 η_{ik} 为服务站 i 处于状态 k 的稳定状态概率,那么平衡方程如式(3)和式(4)所示:

$$\lambda_i\eta_{ik-1}+\mu_i\eta_{ik+1}=\lambda_i\eta_{ik}+\mu_i\eta_{ik} \tag{3}$$

$$\lambda_i\eta_{i0}-\mu_i\eta_{i1}=0 \tag{4}$$

设 ρ_i 为服务站 i 的利用率,根据排队论 $\rho_i=\lambda_i/\mu_i$,求解平衡方程可得式(5)和式(6):

$$\eta_{i0}=1-\rho_i \tag{5}$$

$$\eta_{ik}=(1-\rho_i)\rho^k \tag{6}$$

设在服务站 i 中用户的平均数量为 L_{si},那么可以通过式(7)来计算 L_{si}:

$$L_{si}=\sum_{k=0}^{\infty}k\eta_{ik}=\frac{\rho_i}{1-\rho_i} \tag{7}$$

那么根据排队论,用户在服务站 i 中的逗留时间 T_{qi} 可以通过式(8)来计算:

$$T_{qi}=\frac{L_{si}}{\lambda_i}=\frac{1}{\mu_i-\lambda_i} \tag{8}$$

按照排队网络的理论,一个用户到达节点 i 和离开网络期间的平均时间间隔 T_i 可以用式(9)来计算,其中 $j=1,2,3,\cdots,M$,代表所有的节点。

$$T_i=T_{qi}+\sum_{j=1}^{M}q_{ij}T_j \tag{9}$$

用 T_1 表示用户从到达 map 节点到离开网络的平均时间间隔,用 T_2 表示用户从到达 reduce 节点到离开网络的平均时间间隔。在本文建立的网络系统中,用户从外部到达必须首先进入 map 节点,所以用户在整个系统中的逗留时间可以用 T_1 表示,按照式(9)列出方程组:

$$\begin{cases}T_1=T_{q1}+q_{11}T_1+q_{12}T_2\\T_2=T_{q2}+q_{21}T_1+q_{22}T_2\end{cases} \tag{10}$$

其中 $q_{22}=0$。

求解后得到:

$$T_1=T_{q1}\frac{1}{1-q_{11}-q_{12}q_{21}}+T_{q2}\frac{q_{12}}{1-q_{11}-q_{12}q_{21}} \tag{11}$$

设:

$$a=\frac{1}{1-q_{11}-q_{12}q_{21}},b=\frac{q_{12}}{1-q_{11}-q_{12}q_{21}} \tag{12}$$

再代入式(8),可以得到:

$$T_1=\frac{a}{\mu_{\text{map}}-\lambda_{\text{map}}}+\frac{b}{\mu_{\text{reduce}}-\lambda_{\text{reduce}}} \tag{13}$$

于是,用户在系统中的逗留时间就可以表示为用户到达速率、用户工作量、map 和 reduce 服务站处理速率的函数。

4 算法

4.1 工作流程

提供 MapReduce 服务的集群由多台主机组成,其中一台作为 master,进行任务调度和资源分配,其他作为 slave,提供大量 map 任务和 reduce 任务的槽位,这些槽位由 master 中的调度器统一调度。根据用户要求的截止时间,调度器将所有用户分成 i 类,分别属于不同的优先级。用户提交的作业首先进入一个等待队列,调度器根据 master 中预先存储的用户分类表判断作业的优先级,并把作业送入不同优先级的队列中。在每个优先级的队列中,作业都按照 FCFS 的规则接受服务。调度器给每个优先级别都分配了一定数量的应得槽位数,剩余的槽位作为共享使用。在作业执行过程中,调度器会从所有的槽位中选出适合的一部分分配给作业。统计模块会记录下作业的到达率、每个作业的执行时间等数据,每隔一段时间,调度器就会执行预测算法重新计算各个优先级的应得槽位数。

4.2 工作流程

通过时间序列模型,可以近似准确地预测到某个时间段的用户到达速率。文献[19,20]都成功地在实际应用中运用 AR(1)模型对用户的访问率进行了预测。用一个时间序列 $\{a_i\}$ 记录各个时间片到达用户的数量,模型可以描述为:

$$\hat{a}_{i+1}=\bar{a}+\rho(1)\cdot(a_i-\bar{a})+e \tag{14}$$

其中,$\bar{a}$ 表示时间序列 $\{a_i\}$ 的平均值;e 是白噪声,这里设置为 0;$\hat{a}_{i+1}$ 表示被预测的值;$\rho(1)$ 代表自相关函数,定义如下所示:

$$\rho(1)=\frac{E[(a_i-\bar{a})\cdot(a_{i+1}-\bar{a})]}{\sigma^2} \tag{15}$$

其中 σ^2 代表时间序列 $\{a_i\}$ 的标准差。用时间序列 $\{\lambda\}$ 和 $\{w\}$ 分别代表在一串连续的时间片中,平均用户到达速率和平均工作量,利用公式(14)就可以计算出在下一个时间片中的平均用户到达速率的平均工作量。

4.3　工作流程

优先级 i 的 map 和 reduce 服务站处理速率 μ_{i1} 和 μ_{i2} 与作业运行中 map 阶段和 reduce 阶段被分配的槽位数量 s_{i1} 和 s_{i2} 之间的关系可以用函数 $\mu_{i1}=f_1(s_{i1})$ 和 $\mu_{i2}=f_2(s_{i2})$ 来表示。通过算法 1 可以求出下一个时间间隔的用户到达速率和平均工作量,那么就可以将优先级 i 的逗留时间表示成:

$$T_i=\frac{a}{f_1(s_{i1})-\lambda_{i1}}+\frac{b}{f_2(s_{i2})-\lambda_{i2}} \tag{16}$$

需要保证每个优先级的逗留时间都低于给定的截止时间且槽位总数受到限制,所以问题可以描述为:

$$\begin{cases}\dfrac{a}{f_1(s_{11})-\lambda_{11}}+\dfrac{b}{f_2(s_{12})-\lambda_{12}}<\overline{T}_1\\ \dfrac{a}{f_1(s_{21})-\lambda_{21}}+\dfrac{b}{f_2(s_{22})-\lambda_{22}}<\overline{T}_2\\ \vdots\\ \dfrac{a}{f_1(s_{i1})-\lambda_{i1}}+\dfrac{b}{f_2(s_{i2})-\lambda_{i2}}<\overline{T}_i\end{cases}$$
$$\text{subject to } \sum s_{i1}\leqslant S_{\text{map}};\ \sum s_{i2}\leqslant S_{\text{reduce}} \tag{17}$$

其中,$\overline{T}_i$ 表示优先级 i 的预期截止时间。

但是,如式(17)所示的方程组求解比较困难。用 S_{reduce} 和 S_{map} 分别代表 reduce 阶段和 map 阶段的槽位总数,参数 k 代表他们的比值。在系统中,不同优先级之间的差别在于要求不同的截止时间,不同优先级作业在 map 阶段和 reduce 阶段花费的逗留时间比值是相同的,所以可以按照 map 阶段槽位分配的比例来分配 reduce 阶段的槽位数量,即 $s_{i1}=k\star s_{i2}$。

$$\begin{cases}\dfrac{a}{f_1(s_{11})-\lambda_{11}}+\dfrac{b}{f_2(ks_{11})-\lambda_{12}}<\overline{T}_1\\ \dfrac{a}{f_1(s_{21})-\lambda_{21}}+\dfrac{b}{f_2(ks_{21})-\lambda_{22}}<\overline{T}_2\\ \cdots\\ \dfrac{a}{f_1(s_{i1})-\lambda_{i1}}+\dfrac{b}{f_2(ks_{i1})-\lambda_{i2}}<\overline{T}_i\end{cases} \tag{18}$$

函数 f_1 和 f_2 都是单调递增的,显而易见,$\overline{T}_i$ 关于 s_{i1} 的函数是单调递减的。因此,我们可以用二分查找法从解空间中找到 s_{i1} 的解。文献[19]成功地使用二分查找算法求解方程组。

算法 1　客户资源分配计算方法

输入:I, S_{map}, k, $Tcha$, $\overline{T}_i$;

输出:s_{i1}, s_{i2}。

$mapIdle=S_{\text{map}}$;

For each $i\in I$

　$Tcha=Float.Maxvalue$;

　$mapMax=mapIdle$;

　$mapMin=1$;

　While($Tcha>Yu$)

　　$s_{i1}=(mapMax+mapMin)/2$;

　　$s_{i2}=k\star s_{i1}$;

　　按照公式(16)计算 T_i;

　　If($T_i>\overline{T}_i$)

　　　$mapMin=s_{i1}$;

　　else

　　　$mapMax=s_{i1}$;

　　　$Tcha=\overline{T}_i-T_i$;

　　End if

　End While

　$mapIdle=mapIdle-s_{i1}$;

End For

输入的参数包括代表所有优先级的集合 I、map 的总槽位数量 S_{map}、map 槽位与 reduce 槽位的比值 k、阈值 Yu 和各个优先级的截止时间 $\overline{T}_i$。在搜索过程中 s_{i1} 的检索范围是从 1 到空闲的槽位总数 $mapIdle$。s_{i2} 的值等于 s_{i1} 乘以 k。算法开始的时候 $mapIdle$ 等于 map 槽位的总数,每次完成一个优先级的计算就从自身减去已经分配的槽位数。最后,如果 $mapIdle$ 的数量小于 0,说明资源数量不足,需要管理员对集群进行扩展。如果 $mapIdle$ 的数量大于 0,那么剩余的槽位由各个优先级共享。

4.4　工作流程

按照之前的资源分配算法,每个优先级队列都获得了 s_{i1} 个 map 槽位和 s_{i2} 个 reduce 槽位,另外剩余的 $mapIdle$ 和 $reduceIdle$ 个槽位就作为共享资源来使用。在 MapReduce 框架中,当空闲槽位产生时才发生调度。实时调度中,我们引入了 Capacity 调度算法[6]中饥饿度的概念。

定义 1(饥饿度)　饥饿度是应该分配而实际未分配的资源数量与应该分配的资源数量的比值。

每个优先级队列都设置了饥饿度,当空闲槽位到达时,将会分配给饥饿度最高的优先级队列。饥饿度分为 map 饥饿度 $mHungryDegree$ 和 $reduce$ 饥饿度 $rHungryDegree$,计算方法如式(19)和式(20)所示,其中 $mAssigned$ 和 $rAssigned$ 分别表示已经分配的槽位数量。

$$mHungryDegree_i=1-mAssigned_i/s_{i1} \tag{19}$$
$$rHungryDegree_i=1-rAssigned_i/s_{i2} \tag{20}$$

为了保证每个优先级队列随时都有足够的槽位数量可用,系统需要预留一定数量的槽位。在特

定时刻,可能出现其他队列 $mAssigned_i$ 大于 s_{i1} 的情况,超出的部分将要占用共享资源的名额。当共享资源用尽时,新的空闲槽位将只分配给 $mAssigned_i$ 小于 s_{i1} 的队列。如果所有 $mAssigned_i$ 小于 s_{i1} 的队列都不需要新的槽位,那么新的空闲槽位将被保留。

另外,本地化(Locality)是在调度中需要考虑的一个重要问题。在 MapReduce 框架中,需要被处理的数据分布存储在 slave 上。当 map 任务和该任务需要处理的数据文件分别处于不同的 slave 中时,需要耗费时间和网络资源来传输数据,因此需要尽量将 map 任务调度到它的数据文件所在的 slave 上。本文所采取的策略是,首先在满足本地化条件的作业中,选取饥饿度最高的作业;其次,如果没有满足本地化条件的作业,则从全部作业中选取饥饿度最高的作业。map 阶段具体算法如下:

算法 2 map 阶段实时调度算法

输入:I, $newSlot$;

```
For each i∈I
    mHungryDegree_i = 1 − mAssigned_i / s_i1 ;/* 计算所有优先级的饥饿度 */
End For
Queue sortedQueue;//保存排序后的优先级队列
If(Mapshared==0)/* 如果共享槽位为 0,将不再为已经拥有多于应分配槽位的优先级分配更多槽位 */
    For each i∈I
        If(mAssigned_i > s_i1)
            I.remove(i);
        End If
    End For
End If
While(I≠∅)//对优先级队列按照饥饿度排序
    找到 I 中饥饿度最低的 i;
    sortedQueue.offer(i);
    I.remove(i);
End While
Queue sortedcopy= sortedQueue.copy();
While(!sortedcopy.isEmpty())/* 首先寻找满足本地化条件的优先级队列 */
    i=sortedQueue.poll();
    If(i 满足本地化条件)
        分配 newSlot 给 i;
        Return;
    End If
End While
i=sortedQueue.poll();/* 如果没有满足本地化条件的优先级队列,分配给饥饿度最高的队列 */
分配 newSlot 给 i;
Return;
```

算法以优先级队列集合 I 和新的空闲槽位 $newSlot$ 为输入,首先计算所有优先级队列的饥饿度。$Mapshared$ 表示剩余的共享槽位,如果 $Mapshared$ 等于 0,那么将所有已分配槽位数大于应分配槽位数的优先级删除。然后将所有优先级按照饥饿度排序,存入一个队列 $sortedQueue$ 中。最后从 $sortedQueue$ 队列中选择一个饥饿度最高且满足本地化条件的优先级 i,并将槽位分配给 i。如果没有满足本地化条件的队列,那么将槽位分配给饥饿度最高的优先级。reduce 阶段的调度算法与 map 阶段类似。

5 实验

实验平台是一个由 26 台物理机组成的集群,其中一台作为 master,另外 25 台作为 slave。机器配置了主频为 3.3 GHz 的 CPU、2 GB 内存和千兆网卡;操作系统为 ubuntu11,Hadoop 的版本为 1.0.3。其中每个 slave 节点配置了 4 个 map 槽位和 2 个 reduce 槽位。

实验使用了一个服务请求发生器不断向集群提交作业,作业由 Hadoop 自带的 SleepJob 程序组成,SleepJob 可以由用户设定每个任务的工作时间。实验设计了三种应用:应用一属于简单模式,作业只包括一次 map 阶段和一次 reduce 阶段;应用二属于迭代模式,作业会把 map 阶段→reduce 阶段的过程重复三次;应用三属于链模式,作业执行的过程为 map 阶段 1→map 阶段 2→reduce 阶段→map 阶段 3。

每种应用的数量都是作业总数的 1/3,由此可以计算出状态转移概率,如表 1 所示。

Table 1 State transition probability

表 1 状态转移概率表

符号	概率	描述
q_{12}	5/7	从 map 节点离开后进入 reduce 节点的概率
q_{11}	1/7	从 map 节点离开后再一次进入 map 节点的概率
q_{10}	1/7	从 map 节点离开后离开网络的概率
q_{21}	3/5	从 reduce 节点离开后进入 map 节点的概率
q_{20}	2/5	从 reduce 节点离开后离开网络的概率

实验中用到的各个参数如表 2 所示。

在 MapReduce 框架中,map 阶段的工作量是由 map task 的数量来决定的,map task 的数量固定,每个 map task 都负责处理一部分数据,执行时间大致相同。reduce 阶段工作量由中间数据量和

Table 2　Parameter list

表 2　参数列表

参数	值	描述
S_{reduce}	50 个	reduce 槽位总数
S_{map}	100 个	map 槽位总数
T_1	10 min	优先级队列 1 限定截止时间
T_2	20 min	优先级队列 2 限定截止时间
T_3	40 min	优先级队列 3 限定截止时间
map 平均工作量	100 个 map task	所有用户在 map 阶段的平均工作量
reduce 平均工作量	50 min	所有用户在 reduce 阶段的平均工作量

算法复杂度决定,reduce task 的数量不固定。因此,可以用 map task 的数量来衡量 map 阶段的工作量,用总的执行时间来衡量 reduce 阶段工作量。在实验中,每个 map 任务的执行时间设置为 1 min,每个 reduce 任务的执行时间等于作业在 reduce 阶段的实际工作量除以 reduce 的槽位数。图 5 给出了在不同用户到达率的情况下各个优先级分配的槽位数量,其中 T_1、T_2 和 T_3 标志着三个优先级队列,*share* 表示共享资源。可以看出 T_1、T_2 和 T_3 队列的 map 和 reduce 的槽位数都随着到达率的降低而减少,而共享资源增加。

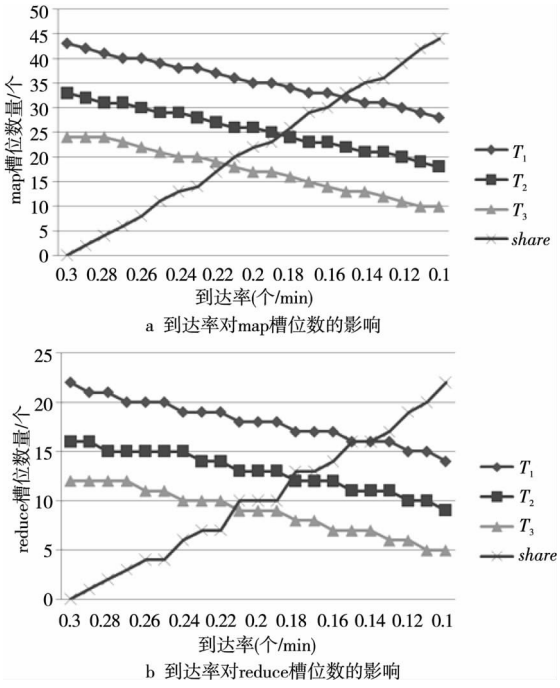

Figure 5　Impact of arrival rate on resource allocation

图 5　到达率对资源分配影响图

图 6 给出了在用户到达率为 0.3 个/min 的情况下,不同平均工作量对各个优先级分配的槽位数量的影响,分配给各个优先级的槽位数都随着工作量的增加而增加。

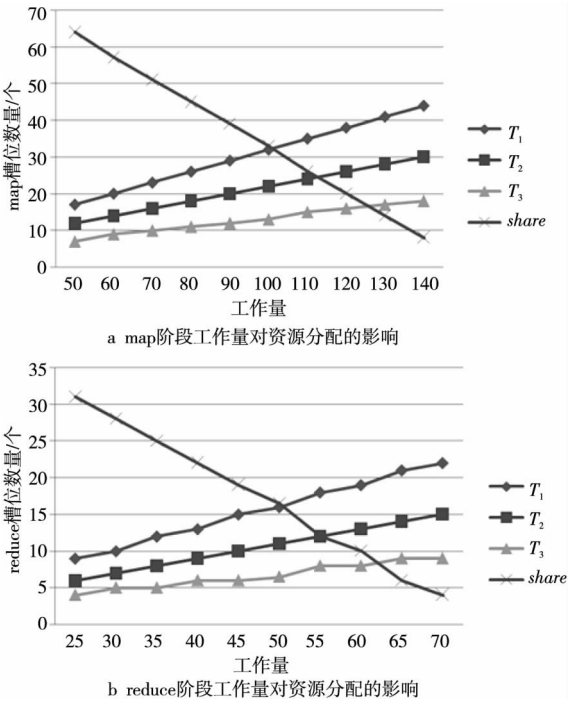

Figure 6　Impact of workload on resource allocation

图 6　工作量对资源分配影响图

图 7 给出了随着到达率的增加,各个优先级作业的平均逗留时间。实验中提交了 150 个作业,每个优先级有 50 个。从图 7 中可以看出,当用户到达率较低的时候不同优先级的平均逗留时间相差不大,这是因为还有大量的共享槽位存在,可以补充给低优先级用户。随着用户到达率的提高,不同优先级的平均逗留时间差距开始增加,越来越趋近于预期的目标。

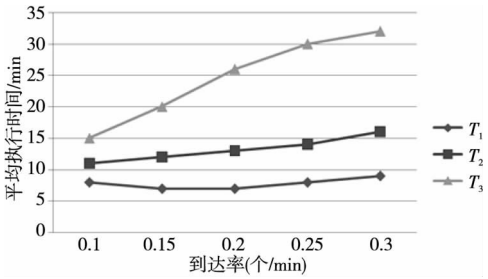

Figure 7　Average delay

图 7　平均延时图

实验使用 Fair 调度算法和 Capacity 调度算法与本文提出的 MPSA 算法进行了对比。在实验中按照不同到达率提交了 150 个作业。Fair 调度算法将所有资源平均分为三个资源池,资源池各自分配给一个优先级的用户使用,每个资源池内部都采用 FIFO 的调度算法。Capacity 调度算法设置了 T_1、T_2 和 T_3 三个队列,每个队列占有 33% 的资源。图 8 给出了三种调度算法在不同到达率情况下作业失败的比较情况。可以看出,在用户到达率

比较低的时候,三种算法的表现都很好,很少出现作业失败的情况。随着到达率的提高,MPSA 算法的表现越来越好,这是因为它会随着到达率情况动态地调整资源分配情况,给比较急迫的用户提供更多的资源,表现出了更好的自适应性。图 9 给出了三种调度算法中作业的平均逗留时间,经过对比可以发现,在到达率较低的时候,Capacity 调度算法和 Fair 调度算法的平均逗留时间相对较短;当到达率升高时,与 MPSA 算法中 T_2 队列相差不大。这是因为在到达率较低的时候,Capacity 调度算法的三个队列和 Fair 调度算法的三个资源池都资源充足。当到达率提高后 MPSA 算法动态调节各个队列的资源,区分开了不同的平均逗留时间,而 capacity 调度算法和 Fair 调度算法仍然保持资源分配方案不变。

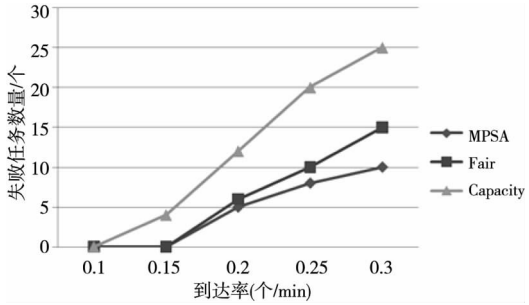

Figure 8 Number of fail jobs

图 8 失败作业数量比较图

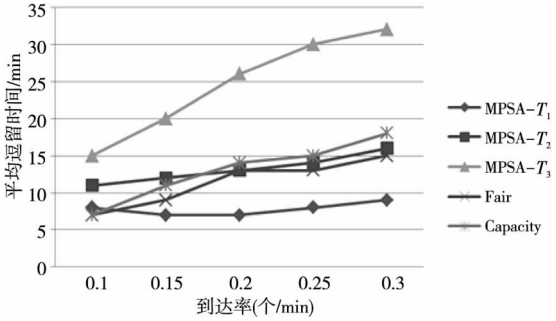

Figure 9 Average sojourn time of three scheduling algorithms

图 9 三种调度算法平均逗留时间图

6 结束语

本文总结了基于 MapReduce 编程模型算法的三种常见模式,并根据常见模式使用 Jackson 排队网络对 MapReduce 算法的运行过程进行了建模。提出了一个多优先级的调度算法,保证不同优先级的用户满足不同的截止时间限定。实验结果表明,根据用户到达率以及平均工作量的变化,算法可以有效地动态分配资源给不同优先级的用户。在实际运行过程中,各个优先级用户的平均逗留时间接近于预期数值,达成了控制访问时间的目标。与传统的调度算法相比,MPSA 调度算法失败的任务数量更少,更加适合在多优先级的条件下保证用户的服务质量。在下一步的工作中,在调度算法中将对应用的类型加以区分。

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] Ranger C,Raghuraman R,Penmetsa A,et al. Evaluating MapReduce for multi-core and multiprocessor systems [C]// Proc of High Performance Computer Architecture,2007:13-24.

[3] Hadoop [EB/OL]. [2014-08-01]. http:// hadoop. apache. org/.

[4] Lin C,Snyder L. Principles of parallel programming[M]. Lu Xin-da, Lin Xin-hua, translation. Beijing: China Machine Press, 2009. (in Chinese)

[5] Job scheduling for multi-user MapReduce clusters[R]. Techical Report UCB/EECS-2009-55, Berkeley: University of California, 2009.

[6] Capacity scheduler guide [EB/OL]. [2010-01-01]. http://hadoop. apache. org/docs/r1. 2. 1/capacity_scheduler. html.

[7] 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,2008:8-10.

[8] Sandholm T,Lai K. Dynamic proportional share scheduling in Hadoop [C]// Proc of the 15th Inernational Workshop on Job Scheduling Strategies for Parallel Processing, 2010:110-131.

[9] Dong X,Wang Y,Liao H. Scheduling mixed real-time and non-real-time applications in MapReduce environment[C]// Proc of Parallel and Distributed Systems(ICPADS),2011:9-16.

[10] Wan Cong,Wang Cui-rong, Yuan Ying,et al. Game-based scheduling algorithm to achieve optimize profit in MapReduce environment[C]// Proc of the 9th International Conference on Intelligent Computing Theories, 2013:234-240.

[11] Wan Cong,Wang Cui-rong, Wang Hai-ming, et al. Utility-driven share scheduling algorithm in Hadoop [C]// Proc of the 10th International Symposium on Neural Networks, 2013:560-568.

[12] Zhao Wei-zhong, Ma Hui-fang, Fu Yan-xiang, et al. Research on parallel k-means algorithm design based on Hadoop platform[J]. Computer Science, 2011, 38(10): 166-176. (in Chinese)

[13] Wan Cong, Wang Cui-rong, Song Xin. A MapReduce based ISODATA algorithm [C]// Proc of Intelligent Control and Information Processing(ICICIP), 2012:765-768.

[14] Zhang Yan-feng, Gao Qi-xin, Gao Li-xin. iMapReduce: A distributed computing framework for iterative computation [J]. Journal of Grid Computing, 2012, 10(1):47-68.

[15] Jiang Hong-ling, Shao Xiu-li, Li Yao-fang. Online botnet detection algorithm using MapReduce[J]. Journal of Electronics & Information Technology, 2013, 35(7):1732-1738. (in Chinese)

[16] Alham N K, Li M, Liu Y, et al. A MapReduce-based distributed SVM ensemble for scalable image classification and annotation [J]. Computers & Mathematics with Applications, 2013, 66(10):1920-1934.

[17] Zhang Ying, Zhao Li-ru, Gu Xin-liang. Resource analysis method for multi-service network based on queuing network [J]. Journal of Computer Applications, 2009, 29(9):2424-2431. (in Chinese)

[18] Lin Chuang. Performance evaluation of computer network and computer system [M]. Beijing: Tsinghua University Press, 2001. (in Chinese)

[19] Zhang Yong-zhong, Zhao Yin-liang, Li Zeng-zhi. Delay-bounded dispatching mechanism for supporting multiple priorities [J]. Journal of System Simulation, 2007, 19(5): 978-982. (in Chinese)

[20] Chandra A, Gong W, Shenoy P. Dynamic resource allocation for shared data centers using online measurements [C]// Proc of Quality of Service—IWQoS 2003, 2003:381-400.

附中文参考文献:

[4] Calvin Lin, Lawrence Snyder. 并行程序设计原理[M]. 陆鑫达, 林新华, 译. 北京:机械工业出版社, 2009.

[12] 赵卫中, 马慧芳, 傅燕翔, 等. 基于云计算平台 Hadoop 的并行 k-means 聚类算法设计研究[J]. 计算机科学, 2011, 38(10):166-176.

[15] 蒋鸿玲, 邵秀丽, 李耀芳. 基于 MapReduce 的僵尸网络在线检测算法[J]. 电子与信息学报, 2013, 35(7):1732-1738.

[17] 张英, 赵莉茹, 谷新亮. 基于排队网络的多业务网络资源分析方法[J]. 计算机应用, 2009, 29(9):2424-2431.

[18] 林闯. 计算机网络和计算机系统的性能评价[M]. 北京:清华大学出版社, 2001.

[19] 张永忠, 赵银亮, 李增智. 一个支持多优先级延迟限定的调度机制[J]. 系统仿真学报, 2007, 19(5):978-982.

作者简介:

万聪(1983-),男,河北昌黎人,博士生,讲师,研究方向为云计算和基于 MapReduce 的分布式算法研究。**E-mail**: wancong@neuq.edu.cn

WAN Cong, born in 1983, PhD candidate, lecturer, his research interests include cloud computing, and distribtued algorithm based on MapReduce.

王翠荣(1963-),女,河北唐山人,博士,教授,CCF 会员(E200005489S),研究方向为大数据处理和网络虚拟化。**E-mail**: wangcr@mail.neuq.edu.cn

WANG Cui-rong, born in 1963, PhD, professor, CCF member(E200005489S), her research interests include big data, and network virtualization.

王聪(1981-),男,河北秦皇岛人,博士,讲师,CCF 会员(E200028845M),研究方向为虚拟网络映射和数据中心资源分配。**E-mail**: congw1981@gmail.com

WANG Cong, born in 1981, PhD, lecturer, CCF member(E200028845M), his research interests include virtual network mapping, and resource allocation in data center.

吕艳霞(1982-),女,河北黄骅人,博士生,讲师,研究方向为计算机网络、分布式计算和大数据计算。**E-mail**: shaoqilyx@163.com

LÜ Yan-xia, born in 1982, PhD candidate, lecturer, her research interests include computer network, distributed computing, and big data.

贾朔(1985-),男,河北南皮人,博士生,讲师,研究方向为计算机网络、分布式计算和大数据计算。**E-mail**: jiashuo@mail.neuq.edu.com

JIA Shuo, born in 1985, PhD candidate, lecturer, his research interests include computer network, distributed computing, and big data.