MapReduce 集群中最大收益问题的研究

王习特, 申德荣, 于戈, 白梅, 聂铁铮, 寇月
(东北大学信息科学与工程学院，沈阳，110004)

摘 要 MapReduce 是目前最为流行的用于大数据分析的并行系统之一。许多企业已经搭建了自己的 MapReduce 集群，为广大用户提供计算服务。用户可以向集群提交具有完成时限要求的 MapReduce 作业，若作业被按时完成，则企业可以获得一定的收益。针对这种应用场景，本文首次提出了 MapReduce 集群中的最大收益问题。为有效地解决该问题，首先提出了一种基于序列的任务调度策略(简称为 SEQ 策略)，并证明了在处理具有完成时限约束的作业时 SEQ 策略的优势。基于 SEQ 策略，本文提出了最大收益的调度算法(scheduling algorithm for maximum benefit, 简称 AMB 算法)，该算法可以快速地确定可接收作业，并给出有效的执行方案，以达到最大化收益的目的。另外，针对在实际应用中的某些异常情况(如节点宕机)，本文也设计了有效的超时处理策略，进一步增加了算法的实用性。最后，通过大量的实验验证了本文所提出算法的有效性。

关键词 大数据；MapReduce 集群；完成时限；最大收益问题
中图法分类号 TP391

Research on Maximum Benefit Problem in a MapReduce Cluster

WANG Xi-Te, SHEN De-Rong, YU Ge, BAI Mei, NIE Tie-Zheng, KOU Yue
(College of Information Science & Engineering, Northeastern University, Shenyang, 110004)

Abstract MapReduce is one of the most popular parallel systems for big-data analysis. Many companies have built their MapReduce clusters to provide computing services to users. Users can submit their deadline-constraint MapReduce jobs to the cluster. If the jobs are finished before their deadlines, the company can get some benefits. For this application scenario, the maximum benefit problem in a MapReduce cluster is firstly presented in this paper. To solve this problem effectively, a sequence-based task scheduling strategy (SEQ strategy for short) is proposed, and we prove the advantages of SEQ strategy for the deadline-constraint job processing. Based on SEQ strategy, a novel Algorithm for Maximum Benefit, AMB, is proposed. AMB can efficiently determine the acceptable jobs and provide the effective execution strategy which can maximize the benefit. Besides, for the exceptions (e.g. node failure) in practical applications, a timeout-handling method is proposed, which can further improve the practicality of the algorithm. At last, the effectiveness of the proposed algorithm is verified through plenty of experiments.

Key words big-data；MapReduce cluster；deadline；maximum benefit problem

---

本课题得到国家“九七三”重点基础研究计划基金项目(No.2012CB316201)；国家自然科学基金面上项目(61033007)；教育部博士点基金(20120042110028)；教育部-英特尔信息技术专项科研基金(MOE-INTEL-2012-06)资助.**王习特**，男，1987年生，博士研究生，计算机学会(CCF)会员（E200027506G），主要研究领域为大数据管理，E-mail: wangxite@research.neu.edu.cn.**申德荣**，女，1964年生，博士，教授，博士生导师，主要研究领域为分布式数据管理、数据集成，E-mail: shenderong@ise.neu.edu.cn. **于戈**，男，1962年生，博士，教授，博士生导师，主要研究领域为数据库、大数据管理，E-mail: yuge@ise.neu.edu.cn. **白梅**，女，1986年生，博士研究生，主要研究领域为传感器数据管理和不确定数据数据管理，E-mail: baimei861221@163.com. **聂铁铮**，男，1980年生，博士，副教授，主要研究领域为数据质量、数据集成，E-mail: nietiezheng@ise.neu.edu.cn. **寇月**，女，1980年生，博士，副教授，主要研究领域为实体识别，E-mail: kouyue@ise.neu.edu.cn.

1 引言

如今，随着信息技术的不断发展，数据呈现出爆炸式的增长趋势，而对于海量数据的分析与处理也成为了一个热点课题[1]。MapReduce[2]作为一款流行的并行计算框架，已被公认为是用于分析和处理海量数据的最有效途径之一。该框架提供了一套简洁高效的 API，用户只需要编写相应的 Map 和 Reduce 函数，就可以轻松的处理 TB 甚至 PB 级别的海量数据，而不用关心具体的处理细节(如数据分布，容错处理等)。

目前许多大型 IT 公司都已经组建了自己的 MapReduce 集群，为广大互联网用户提供方便快捷的云计算服务。用户只需要支付一定的费用，就可以在这些集群上运行自定义的 MapReduce 作业。在这个过程中， IT 公司与用户之间经常以服务层协议[3,4](service level agreement，简称 SLA)的方式来规定服务的具体细节。常见的 SLA 主要包括如下 2 类：一种是按租用量付费，即所支付的费用与用户所申请的集群规模和使用时间成正比；另一种是按完成效果付费(或称按完成时限付费)，即用户在提交作业的同时也给出完成时间的要求，服务提供者只有按时完成该作业，才能获得相应的报酬。如情感分析、垃圾邮件检测等常见的 MapReduce 应用，都需要相应的完成时间保证，是典型的按完成时限付费的应用实例。

本文所提出的最大收益问题主要针对按完成时限付费的 MapReduce 作业。具体地，每个用户提交的作业可以抽象为如下 4 部分：a) 对作业内容的具体描述，即用户自定义的 Map 和 Reduce 函数；b) 完成时限，即用户对作业最终完成时间的要求；c) 收益，如果作业被接收并且按时完成企业可以得到的经济收益；d) 赔偿，如果作业被接收但未按时完成需要赔付给用户的费用。对于这些服务提供商而言，在一段时间内可能有大量的用户申请运行作业。考虑到有限的集群规模和计算能力，服务提供者需要选择性地接收合适的作业，并使用有效的调度算法保证所有接收的作业能够按时完成，以达到最大经济收益的目的。

针对上述情况，本文首次提出了 MapReduce 集群中的最大收益问题。为有效解决该问题，需要从大量申请中快速找到可接收的作业集合，并根据 MapReduce 作业的执行特点，给出合理的调度方案，具体的难点如下：

(1) 快速选择合理的可接收的作业集：明显地，对于 N 个作业申请，可以有 2^N 种接收作业集。服务提供者需要快速地从中确定一个合理的接收作业集。一方面，他们希望接收尽量多的且具有较高收益的作业，以最大化总收益。另一方面，如果接收的作业过多超过了系统的计算能力，会导致许多作业无法按时完成，反而会降低总收益。因此，如何选择合理的可接收作业集是难点之一。

(2) 合理的调度算法：当确定了可接收作业集合之后，一旦出现某个接收的作业无法按时完成，就会带来收益损失，因此合理的调度算法是非常必要的。但是由于MapReduce作业自身的执行特点 _，导致无法对Reduce任务进行很好的预先安排。因此，如何设计一个合理的调度算法是难点之二。

(3) 良好的超时处理策略：由于异常情况(网络阻塞、节点宕机等)在大规模集群中是无法避免的。因此即便采用优秀的调度算法，也无法完全避免作业超时。此时，系统必须合理的放弃个别作业以保证其他作业不会受到影响。因此，如何设计一个良好的超时处理策略是难点之三。

为有效解决 MapReduce 集群中的最大收益问题，本文设计了作业收益的评价标准，并以最大收益为目标设计了相关的算法。具体地，本文的贡献点总结如下：

(1)本文首次提出了在 MapReduce 环境下的最大收益问题，给出了收益的评价标准。

(2)提出了一种基于序列的任务调度策略(简称 SEQ 策略)，并证明了 SEQ 策略在处理具有完成时限约束的作业时具有明显的优势。

(3)基于 SEQ 策略，本文提出了以最大化收益为目的的 AMB 算法。对于静态的申请作业集合，AMB 算法可以快速地选择可接收的作业集合，并给出合理的调度方案，保证接收的作业按时完成；当有新作业提交时(即针对动态的作业集合)，AMB 算法采用增量的方法进行快速地接收判定，并及时地更新调度方案。

(4)在实际生产中，异常情况(如节点宕机等)无法完全避免，某些被接收的作业可能无法按时完成。针对此种情况，本文设计一种超时处理策略，

_ 对于任意一个MapReduce作业，系统都会将其切分成多个Map和Reduce任务，而所有的Reduce任务只有在该作业的全部Map任务完成之后才能够开始执行。

可以有效地提高算法的实用性。

(5)最后，本文通过大量实验验证了所提出算法的有效性。

本文第 2 节概述了 MapReduce 的背景知识和相关工作；第 3 节陈述了 MapReduce 中的最大收益问题；第 4 节详细描述了本文提出的相关算法；第 5 节给出了实验结果与分析；第 6 节对全文进行总结。

2 背景知识

本小节首先对 MapReduce 框架进行了概述，然后总结了 MapReduce 环境中的相关工作。

2.1 MapReduce概述

MapReduce 是一款由 Google 公司提出的并行计算框架，主要适用于大规模数据的分析与处理。该框架依托于一个底层的分布式文件系统[5](DFS)存储输入输出文件。典型地，数据文件会被切分成多个等大的分片，分布式地存储在 DFS 当中。该框架采用主从(master-slave)架构，主节点主要负责集群管理和任务调度，对于任何一个 MapReduce 作业，主节点都会将其分割成多个 Map 和 Reduce 任务，并将它们分给空闲的从节点进行处理。每个从节点上都具有一定数目的 Map(Reduce)任务槽，用于处理主节点所指派的 Map(Reduce)任务。

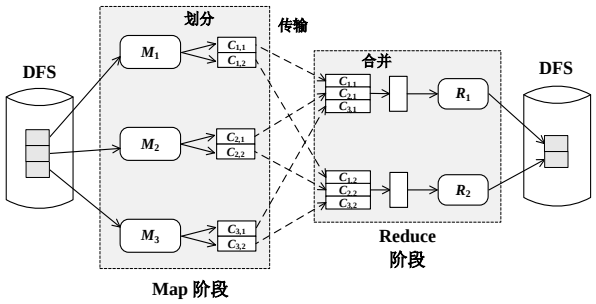


图 1 MapReduce 概况

图 1 描述了一个 MapReduce 作业执行的大体流程。当一个作业 j 被提交时，系统的主节点会将其切分成 m 个 Map 任务和 r 个 Reduce 任务。通常 m 是 j 的输入文件所包含的分片数目，而 r 由用户指定或为默认值。在 Map 阶段，首先，一个分片中的数据会被分配给一个特定的 Map 任务槽(如 M_1)，并被解析成一系列(k_1, v_1)格式的键-值对(key-value pair)。然后，每个 Map 任务槽调用用户提交的 Map 函数，输入一系列(k_2, v_2)格式的键-值对作为中间结果。中间结果会被划分成 r 个块，并被发送给相应的 Reduce 任务槽。划分函数保证具有相同键(key)的键-值对被划分到同一个块中。在 Reduce 阶段，首先，每个 Reduce 任务槽(如 R_1)将收到的块合并到一起，形成一系列(k_2，$list(v_2)$)格式的键-值对。接下来调用用户提交的 Reduce 函数，输出最终结果。最终结果将被写回到 DFS 当中。

2.2 相关工作

目前已经出现许多适用于MapReduce的任务调度算法[6-9]，如FIFO调度器 [6]、Capacity调度器 [7]、Fair调度器 [8]等等。也出现许多针对特定应用场景的调度算法：Thomas Sandholm[10]等提出一种调度算法，允许用户根据作业的重要程度，动态调整所申请的计算资源量。Matei Zaharia[11]等提出了LATE调度算法，主要适用于异构的集群环境。YongChul Kwon[12]等针对MapReduce处理过程中出现的数据倾斜问题，提出了Skewtune算法。

此外，还出现了一些针对具有完成时限要求作业的调度算法。Jorda Polo[13]等提出了 PD 调度算法，该算法可以估计作业的完成时间，如果一个作业无法按时完成，则将计算资源优先地分配给这个作业。但该算法并没有设计接收判定模块，会接收用户提交的所有作业，明显无法应用于本文提出的应用场景。DC 调度算法[14]和 MinEDF-WC 算法[15,16]则设计了接收判定模块，若算法认为提交的作业无法按时完成，则会驳回该作业。具体地，DC 调度算法根据作业的大小，尝试分配固定个数 Map 任务槽给该作业，并假设每个作业在 Reduce 阶段都可以使用全部的 Reduce 任务槽。而 MinEDF-WC 算法则根据作业的大小，尝试分配固定数目的 Map 和 Reduce 任务槽给该作业，并使每个作业所获得了任务槽总数最小。可见，DC 和 MinEDF-WC 算法所采用的估计模型都是静态的。但在 MapReduce 集群中，为达到高度的可用性，普遍采用动态的任务分配方式。因此这种静态估计模型会导致系统可用性大大降低，同时也无法充分利用系统的计算资源。另外，这些算法只考虑到了作业的完成时限要求，但并未进一步考虑收益问题，因此与本文的设计目标也不相同。

MapReduce 下的最大收益问题由本文首次提

[6] Hadoop, http://hadoop.apache.org/ 2013,10,16

[7] Capacity Scheduler, http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html 2013,10,7

[8] Fair Scheduler, http://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html 2013,8,4

出的，与现有研究存在很大不同。已有算法不能很好地解决最大收益问题，因此设计一种可以有效解决该问题的算法是十分有意义的。

3 问题描述

本文主要针对 MapReduce 环境下的最大收益问题，其他的性能指标，如吞吐量等，并不是本文的首要目标。另外，本文提出的 AMB 算法的具体任务分配方法遵循了 MapReduce 的动态分配原则，因此并不会影响 MapReduce 的其他特性，如容错处理、负载均衡等。

本文主要考虑常见的同构 MapReduce 集群，即认为每个从节点的处理能力大致相同。在一个具有 M 个 Map 任务槽和 R 个 Reduce 任务槽的 MapReduce 集群中，对于任意一个用户提交的作业 j，如下参数为已知：a) j 具有的 Map 任务个数：$j.N_m$；b) j 具有的 Reduce 任务个数：$j.N_r$ (为达到较高处理效率，$j.N_m$($j.N_r$)常为 M(R)的整数倍)；c) j 的完成时限要求：$j.deadline$；d) 当作业 j 被接收并在 $j.deadline$ 之前完成时，企业可以获得的收益：$j.benefit$。此外，企业可为所有作业设定统一的赔付比α，如果 j 被接收但未能按时完成，则需要赔付给用户的费用为 $j.benefit\cdot\alpha$。不失一般性地，若企业不提供赔付服务，则可将赔付比α设为 0。

设计目标：在一段时间内有大量用户向服务提供商(即集群的拥有者)提出作业申请，这些作业形成了申请候选集 $J=\{j_1, j_2, ..., j_{|J|}\}$。首先，服务提供商需要从 J 中选择可接收作业集 $A=\{j_1', j_2', ..., j_{|A|}'\}$，并根据合适的调度算法尝试去完成 A 中的所有作业。最终，对于 A 中的任意作业 j_i'，若 j_i'在 $j_i'.deadline$ 之前完成，那么称 j_i'是有效的，可以为提供商获得收益 $j_i'.benefit$，否则，称 j_i'是无效的，需要赔付 $j_i'.benefit\cdot\alpha$。那么，可获得的总收益可以表示为：

$$P=\sum_{j\in A, j\text{是有效的}} j.benefit-\sum_{j'\in A, j\text{是无效的}} j'.benefit\cdot\alpha \qquad (1)$$

MapReduce 下的最大收益问题，即设计有效的算法以最大化总收益 P。

4 算法描述

本小节首先提出了一种基于序列的任务调度策略，接下来根据这种策略提出了 AMB 算法，最后给出了超时处理的方法。

4.1 基于序列的任务调度策略

对于任意作业 j，首先需要估计如下 2 个基本参数：a) j 的平均每个 Map 任务的处理时间：$j.T_m$；b) j 的平均每个 Reduce 任务的处理时间：$j.T_r$(这些参数可以通过简单的抽样得出，文献[11,13]中均采用了类似的方法，具体细节详见本文附录 A)。若集群中的全部任务槽都用于处理该作业，那么可以估计 j 的 Map 阶段的耗时为 $TC_m(j)=\lceil j.N_m/M\rceil\times j.T_m$，Reduce 阶段的耗时为 $TC_r(j)=\lceil j.N_r/R\rceil\times j.T_r$。$TC_m(j)$和 $TC_r(j)$是本文调度策略中主要使用的参数。

定义 1(序列)：对于一个作业集合 JS(所包含的作业个数记作$|JS|$)，序列 S 是 JS 中所有作业的一种排列，它规定了 JS 中的所有作业的 Map 阶段的完成顺序。具体地，若记 j 的 Map 阶段的完成时间为 $COT_m(j)$，对于给定的序列 $S=\{j_1, j_2, ..., j_{|JS|}\}$，那么 $\forall j_i\in S$ $(0<i<|JS|)$，S 规定 $COT_m(j_i)<COT_m(j_{i+1})$。

基于一个给定的序列 S，本文提出了一种全新的任务调度策略(SEQ 策略)，具体过程如下：

(1) Map 部分，当一个空闲的 Map 任务槽请求任务时，主节点选择一个属于序列 S 中首作业的 Map 任务分配给该任务槽。当首作业的 Map 任务全部分配完毕之后，将该作业从序列首部移除。例如在图 2 中，给定序列 $S=\{j_1, j_2, j_4, j_3\}$，当 Map 任务槽请求任务时，SEQ 策略选择首作业 j_1 的 Map 任务分配给该槽。

(2) Reduce 部分，将集合 JS 中的作业按照完成时限由小到大的顺序进行排列，得到一个有序队列 $L_d=\{j_1', j_2', ..., j_{|JS|}'\}$。当一个空闲的 Reduce 任务槽请求任务时，主节点顺序查找 L_d 中的作业，找到第一个已经完成了 Map 阶段的作业，并选择一个属于该作业的 Reduce 任务分配给空闲的任务槽。例如在图 2 中，$L_d=\{j_1, j_2, j_3, j_4\}$，若在时间点 330 时有 Reduce 任务槽请求任务，此时作业 j_1, j_2 完成了 Map 阶段，j_1 的完成时限较小，所以选择一个属于 j_1 的 Reduce 任务分配给该槽。

作业	$TC_m(j)$ (秒)	$TC_r(j)$ (秒)	完成时限(秒)
j_1	140	220	400
j_2	190	120	600
j_3	190	180	880
j_4	160	140	900

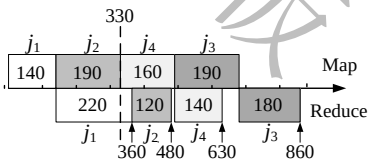


图 2 SEQ 策略举例

根据上述的作业执行策略，对于一个给定的序列 S，可以计算出任意作业 j 的 Map 阶段的完成时间：$COT_m(j)$以及 Reduce 阶段的完成时间：$COT_r(j)$。具体方法如下：

(1) 给定序列 $S=\{j_1, j_2, ..., j_{|JS|}\}$，$\forall j_i \in S$，其 Map 阶段的完成时间为 $COT_m(j_i)= COT_m(j_{i-1})+ TC_m(j)= \sum_{k\in[1,i]} TC_m(j_k)$。例如在图 2 中，给定序列 $S=\{j_1, j_2, j_4, j_3\}$，作业 j_2 的 Map 阶段完成时间为 $COT_m(j_2)= COT_m(j_1)+ TC_m(j_2)$=140+190=330。

(2) 对于作业集合 *JS*，按照作业完成时限由小到大排序，可以容易地得到队列 $L_d=\{j_1', j_2', ..., j_{|JS|}'\}$。那么对于 L_d 中的第一个作业 j_1'，根据给定的序列 *S*，可以计算得出其 Map 阶段的完成时间 $COT_m(j_1')$，那么其 Reduce 阶段的完成时间：$COT_r(j_1')= COT_m(j_1')+ TC_r(j_1')$。我们将时间段$[COT_m(j_1'), COT_r(j_1')]$标记为“占用”。那么对于 L_d 中的第 *i* 个作业 j_i'，我们首先计算得出 $COT_m(j_i')$，接下来从时间点 $COT_m(j_i')$开始，找到一系列未被“占用”的时间段，使其总长度为 $TC_r(j_i')$，标记这些时间段为“占用”，最后一个时间段的结束时间即作业 j_i'的 Reduce 阶段完成时间 $COT_r(j_i')$。例如在图 2 中，给定 $S=\{j_1, j_2, j_4, j_3\}$，$L_d=\{j_1, j_2, j_3, j_4\}$，作业 j_1 的 Map 阶段完成时间 $COT_m(j_1)$=140，Reduce 完成时间 $COT_r(j_1)$ =140+220=360，时间段[140,360]被 j_1 占用。j_2 的 Map 完成时间 $COT_m(j_2)$=330，但时间段[330,360]被 j_1 占用，因此 j_2 占用的时间段为[360,480]。j_2 的 Reduce 阶段完成时间 $COT_r(j_2)$=480。

基于上述的作业执行策略以及完成时间计算方法，下面给出有效序列的定义。

定义 2(有效序列)：给定作业集合 *JS*，对于一个序列 *S*，若使用 SEQ 策略 *S* 可以使得$\forall j \in JS$，$COT_r(j) \leqslant j.deadline$，那么称 *S* 为一个有效序列。

下面，我们说明本文提出的 SEQ 策略在处理具有完成时间约束作业时的优势。

定理 1(Map 最优性)：对于作业集合 *JS*，给定序列 *S*(即给定了作业 Map 阶段的完成顺序)，那么 SEQ 策略可以保证$\forall j_i \in JS$，j_i 在满足 *S* 约束的前提下，可以在最短时间内完成它的 Map 阶段。

证明：给定序列 $S=\{j_1, j_2, ..., j_{|JS|}\}$和 *S* 中的第 *i* 个作业 j_i，*S* 规定 j_i 的 Map 阶段必须在所有 $j_k(1\leqslant k<i)$ 的 Map 阶段结束之后才能完成，即 j_i 的 Map 阶段的最早完成时间为 $\sum_{k\in[1,i]} TC_m(j_k)$。同时，若采用 SEQ 策略进行任务分配，$j_i$ 的 Map 阶段完成时间 $COT_m(j_i)$ 亦为该值。对于任意 $j_i(i\in[1,|JS|])$，上述结论均成立，定理得证。　　证毕

定理 2(Reduce 最优性)：对于作业集合 *JS*，给定序列 *S*，如果使用 SEQ 策略进行任务分配会出现作业超时，那么无论何种调度策略都无法保证 *JS* 中所有作业都按时完成，*S* 一定不是有效序列。

证明：定理 1 已经证明了 SEQ 策略在 Map 阶段的最优性，因此这里只需要考虑 Reduce 阶段。假设当使用 SEQ 策略时，作业 *j* 是超时作业，此时有 2 种情况，a)　若 $COT_m(j)+ TC_r(j)>j.deadline$，即作业 *j* 在 Map 阶段完成之后就马上运行他的 Reduce 任务，也无法按时完成。那么显然无论使用何种调度策略，都无法使 *j* 按时完成。

b) 若 $COT_m(j)+ TC_r(j)\leqslant j.deadline$，而 *j* 的 Reduce 完成时间 $COT_r(j)>j.deadline$。那么根据 SEQ 策略，在时间段$[COT_m(j), COT_r(j)]$内，必有部分时间被一些完成时间时限小于 $j.deadline$ 的作业的 Reduce 任务占用。我们从这些作业当中，选择 Map 阶段完成时间最小的作业，记为 *j'*。在时间段$[COT_m(j'), COT_r(j)]$内运行 Reduce 任务的所有作业中，判断是否仍然存在完成时间时限小于 $j'.deadline$ 的作业，若存在，则重复上述过程，直到我们找到最终作业 j_f，使得在时间段$[COT_m(j_f), COT_r(j)]$内运行 Reduce 任务的所有作业的完成时限都不小于 $j_f.deadline$。显然地，在时间段$[COT_m(j_f), COT_r(j)]$内，不存在空闲的时间段。那么如果使用其他调度策略，可以使 *j* 按时完成，则一定会有其他作业变得超时。　　证毕

根据上述 2 个定理，在给定序列 *S* 时，本文提出的 SEQ 策略是最优的，也就是说如果使用 SEQ 策略会出现作业超时，那么任何策略都无法保证所有作业都按时完成。

4.2　AMB算法

具体地，AMB 算法由 2 个部分构成：首先，针对一般情况，即申请候选集为静态时，算法利用一种新型的评分策略确定接收优先级，并采用了有效的剪枝策略以快速地确定可以接收作业集合，并找到相应的有效序列；此外，在实际应用中，作业集合往往是动态更新的，即允许在运行过程中有新的作业提交。针对这种情况，AMB 算法采用一种增量的方法实现快速地接收判定，并在必要时更新有效序列。下面就对这 2 部分进行详细的阐述。

静态部分

对于具有大量作业的申请集合 $J=\{j_1, j_2, ..., j_{|J|}\}$，我们需要根据 SEQ 策略快速地确定可接收作业集合 $A=\{j_1', j_2', ..., j_{|A|}'\}$，并确定适合集合 *A* 的有效序列，以达到最大收益的目的。但是，对于申请集合 *J*，就存在 $2^{|J|}$种不同的接收作业集合。对于任何一种接收集合 $A=\{j_1', j_2', ..., j_{|A|}'\}$，仍然存在|*A*|!种

不同的序列。为快速找到可接收集合和对应的有效序列，本文提出了 AMB 算法，具体步骤如下。

为提高接收判定的效率，本文首先对申请集合 J 中的作业进行合理排序，以确定接收的优先次序。通过分析 MapReduce 作业的特点，排序主要考虑如下 2 个方面：

(1)直观地，由于集群的总计算能力有限，为实现收益最大，应考虑优先接收“收益比”大的作业。其中$\forall j\in J$，j 的系统耗时可以量化为 $STC(j)= TC_m(j)\cdot M/(M+R)+ TC_r(j)\cdot R/(M+R)$，则 j 的收益比：$Br(j)=j.benefit/STC(j)$，即运行 j 时每秒可以获得的收益。例如在表 1 中，作业 j_1 的收益比：$Br(j_1)=300/(150\cdot 30/50+170\cdot 20/50)\approx 1.90$。

(2)另外，考虑到 MapReduce 集群的特点，如果某个作业 j 过长，那么当运行 j 的 Map(Reduce)任务时，大部分 Reduce(Map)任务槽可能由于得不到任务而被闲置，导致系统资源的浪费，进而影响其他作业的正常接收。因此，应考虑对过长作业引入收益惩罚机制。例如在表 1 中，作业 j_5 的收益比很高，但相比于其他作业，它的 $TC_m(j_5)$和 $TC_r(j_5)$过长，若接收 j_5，那么会影响其他作业的接收，使总体收益变低，因此需要对 j_5 加入收益惩罚。

表 1 排序得分举例(Map 槽数 M=30，Reduce 槽数 R=20)

作业	$TC_m(j)$ (s)	$TC_r(j)$ (s)	完成时间 (s)	收益	收益比	排序得分
j_1	150	170	500	300	1.90	1.90
j_2	160	140	650	340	2.24	2.24
j_3	190	180	980	400	2.15	2.15
j_4	140	200	400	380	2.32	2.32
j_5	400	500	1000	1360	3.09	1.91
j_6	190	120	600	350	2.16	2.16

综合考虑上述 2 方面因素，本文提出了一种以最大收益为目标的排序得分函数，得分较高的作业将被优先接收。

$$Score(j)=\frac{j.benefit}{STC(j)\cdot Ad(j)} \tag{2}$$

其中是 $Ad(j)$为 j 的调整系数，$STC(j)\cdot Ad(j)$为 j 的调整时间。具体地，记集合 J 中所有作业 Map 阶段耗时之和 $Total\_TC_m=\sum_{j\in J} TC_m(j)$。对于作业 j，记录 J 中除 j 以外其他作业的 Map 阶段平均耗时 $\overline{TC}_m(j)=(Total\_TC_m-TC_m(j))/(|J|-1)$。给定一个惩罚阈值$\beta(\beta>1)$，若 $TC_m(j)>\overline{TC}_m(j)\cdot\beta$，那么认为 j 的 Map 阶段过长；同理，若 $TC_r(j)>\overline{TC}_r(j)\cdot\beta$，那么认为 j 的 Reduce 阶段过长，则调整系数 $Ad(j)$为：

$$Ad(j)=\begin{cases}\frac{TC_m(j)-\overline{TC}_m(j)}{TC_m(j)}\times\frac{M}{M+R}+\frac{TC_r(j)-\overline{TC}_r(j)}{TC_r(j)}\times\frac{R}{M+R}+1; \\ \quad //当TC_m(j)>\overline{TC}_m(j)\times\beta且TC_r(j)>\overline{TC}_r(j)\times\beta时 \\ \frac{TC_m(j)-\overline{TC}_m(j)}{TC_m(j)}\times\frac{M}{M+R}+1; \\ \quad //当TC_m(j)>\overline{TC}_m(j)\times\beta且TC_r(j)\le\overline{TC}_r(j)\times\beta时 \\ \frac{TC_r(j)-\overline{TC}_r(j)}{TC_r(j)}\times\frac{R}{M+R}+1; \\ \quad //当TC_m(j)\le\overline{TC}_m(j)\times\beta且TC_r(j)>\overline{TC}_r(j)\times\beta时 \\ 1; \\ \quad //当TC_m(j)\le\overline{TC}_m(j)\times\beta且TC_r(j)\le\overline{TC}_r(j)\times\beta时\end{cases} \tag{3}$$

下面，我们通过一个例子来具体说明文中排序得分和调整系数的含义。

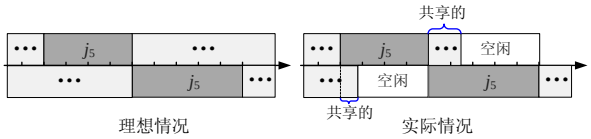


图 3 调整系数举例

例 1 如图 3 中所示，j_5是表 1 中的作业。j_5的系统耗时为 $STC(j_5)=400\times3/5+500\times2/5=440$，则 j_5的收益比为 $Br(j_5)=1360/440\approx3.09$。假设在理想情况下，$j_5$可以与其他作业共享系统资源，系统资源可以充分利用，此时 j_5的排序得分等于收益比。但在实际情况下，根据表 1 中的其它作业的运行时长可以看出，j_5的 Map 和 Reduce 阶段的耗时都远超过其他作业。在 j_5的 Map 任务运行期间，其它作业的 Reduce 任务很快地执行完毕，在剩余的时间里，许多 Reduce 任务槽由于无法获得任务而长期处于空闲状态，导致了计算资源的浪费，而实际上这种资源浪费是由 j_5引起的。同理，当 j_5的 Reduce 任务运行期间，也会有大量闲置的 Map 任务槽。具体地，j_5的 Map 和 Reduce 阶段与同其它作业共享的时间长度的期望值分别是 $\overline{TC}_m(j_5)$=166 和 $\overline{TC}_r(j_5)$=162，因此我们可以将 j_5 占用的系统时间调整为 $STC(j)\cdot Ad(j)=440\times[(400-166)/400\times3/5+(500-162)/500\times2/5+1]\approx713$，即 j_5的调整时间为 713，则排序得分为 $Score(j_5)=1360/713\approx1.91$。

为方便描述，假设申请集合 $J=\{j_1, j_2, ..., j_{|J|}\}$中的作业已经按照公式 2 的得分降序排列完毕，$\forall j\in J$，$Score(j_i)\geqslant Score(j_{i+1})$。接下来，一种基本的接收判定方法是，首先初始化接收集合 A 为空；然后，逐个审查 J 中的作业，对于$\forall j_i\in J$，依据 SEQ 策略，

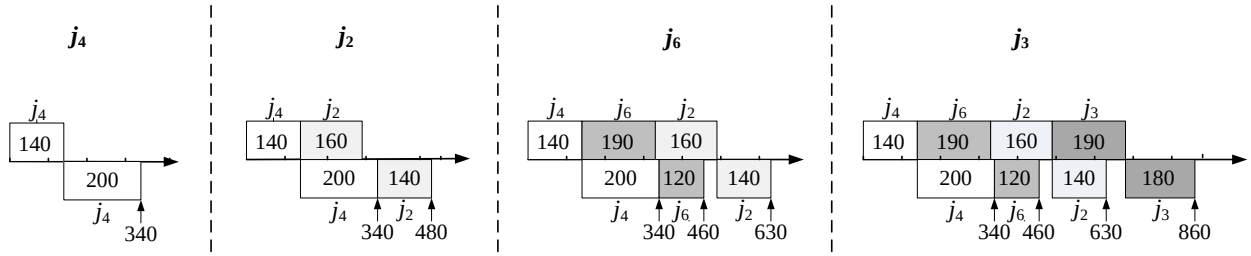


图 4 AMB 算法-静态部分

判定是否存在对于 $A\cup\{j_i\}$的有效序列。若存在，将 j_i 加入到 A 中，否则，放弃作业 j_i。但实际上，对于任意 $A\cup\{j_i\}$，都存在(|A|+1)!种序列。下面，我们就介绍 2 种剪枝策略以快速地找到有效序列。

定理 3: 给定作业集合 A 的一个有效序列 $S=\{j_1, j_2, ..., j_n\}$，对于一个新作业 j_{new}，有 n+1 个位置可供 j_{new} 插入(第 i 个位置介于 j_{i-1} 与 j_i 之间)。如果 $TC_m(j_{new})+COT_m(j_i)+TC_r(j_i)>j_i.deadline$，那么 j_{new} 无法插入到位置[1,i]中。

证明：明显地，j_{new} 插入到[1,i]中的任何位置，j_i 都将超时。　　证毕

定理 4: 给定作业集合 A 的一个有效序列 $S=\{j_1, j_2, ..., j_n\}$，对于一个新作业 j_{new}，若 $COT'_m(i)+TC_m(j_{new})+TC_r(j_{new})>j_{new}.deadline$，那么 j_{new} 无法插入到位置[i+1,n+1]中。

证明：假设 j_{new} 可以插入到[i+1,n+1]中的某个位置，根据 SEQ 策略，j_{new} 的最早完成时间≥ $COT_m(j_i)+TC_m(j_{new})+TC_r(j_{new})>j_{new}.deadline$。因此，$j_{new}$ 必然超时　　证毕

利用上述定理，本文提出的 AMB 算法(静态部分)可以快速地得到可接收的作业集合以及相应的有效序列，以达到最大收益。算法 1 给出了算法的具体过程。

算法 1.　AMB 算法(静态部分)

输入：用户提交的作业申请集合 $J=\{j_1, j_2, ..., j_{|J|}\}$

输出：接收作业集合 A 以及相应的有效序列

01. 根据公式 2 计算 J 中各作业的排序得分；
02. 将 J 中作业按照得分降序排序；
03. J 中首作业 j_1 加入 A，$\{j_1\}$加入有效序列集合φ_1；
04. FOR each $j_i\in J$ (i >1) DO
　　初始化有效序列集合φ_i 为空；
05.　FOR each $S\in\varphi_{i-1}$ DO
06.　　根据定理 3 判断 j_i 可插入的最小位置 a；
07.　　根据定理 4 判断 j_i 可插入的最大位置 b；
08.　　IF $a\leqslant b$ THEN
09.　　FOR each $p\in[a,b]$ DO
10.　　将 j_i 插入 S 的 p 位置，得到新序列 S'；
11.　　根据 SEQ 策略计算 S'中各作业完成时间；
12.　　IF　S'是有效序列 THEN
13.　　将 S'插入有效序列集合φ_i 中；
14.　　ENDIF
15.　ENDFOR
16.　ENDIF
17. ENDFOR
18. IF φ_i 不为空 THEN
19.　将 j_i 加入 A 中；
20. ELSE
21.　放弃接收 j_i，$\varphi_i=\varphi_{i-1}$；
22. ENDIF
23. ENDFOR
24. 返回集合 A 和$\varphi_{|J|}$中的任一有效序列；

例 2(AMB 算法-静态部分)**:** 继续表 1 中给出了的例子，根据得分排序得到的有序作业集合：$J=\{j_4, j_2, j_6, j_3, j_5, j_1\}$。接下来如图 4 所示，在 AMB 算法中，首先将 j_4 加入到接收集合 A 中。下面考虑 j_2，由于 $TC_m(j_2)+COT_r(j_4)+TC_r(j_4)=160+140+200=500>j_4.deadline$，根据定理 3，$j_2$ 无法插入到 j_4 之前。因此，只需要判断$\{j_4, j_2\}$是否为有效序列。根据 SEQ 策略中作业完成时间的计算方法，得知$\{j_4, j_2\}$为有效序列。接下来考虑 j_6，此时对于 A 的有效序列$\{j_4, j_2\}$，$COT_m(j_2)+TC_m(j_6)+TC_r(j_6)=300+190+120=610>j_6.deadline$，根据定理 4，$j_6$ 无法插入到 j_2 之后。同时，根据定理 3，j_6 无法插入到 j_4 之前，因此，只需要判断$\{j_4, j_6, j_2\}$是否为有效序列。经进一步计算，得知$\{j_4, j_6, j_2\}$确实为有效序列。重复上述过程直至判断完毕 J 中的所有作业。最终，通过 AMB 算法，我们得到了接收作业集合 A 以及 A 的有效序列$\{j_4, j_2, j_6, j_3\}$。其总收益为 1470。

动态部分

在实际应用中，申请作业集合往往是动态更新的，即允许在运行过程中提交新的作业申请。在这种动态的情况下，对于每个新提交的作业，系统首

先需要根据目前的剩余计算资源，判断该作业是否可以被接收；此外，若成功接收新的作业，那么还需要及时地生成新的有效序列。在本文的 AMB 算法中，主要采用了一种简单的增量方法来有效地解决上述问题。具体处理过程描述如下。

设在运行过程中的某一时间点 t，系统检测到若干个新的作业申请，记作集合 J'。首先，对 J' 中的作业按照排序得分(见公式 2)进行降序排序。明显地，得分较高的作业具有较高的接收优先级(具体原因已在静态部分阐述)。接下来，依次遍历 J'中的作业，对于任意作业 j'，可以根据定理 3、4 计算出 j'可插入原有效序列 S 中的位置。若存在有效位置，则接收 j'并生成新的有效序列。否则，放弃 j'。

例 3 (AMB 算法-动态部分)：继续表 1 中给出的例子，原有效序列已计算完毕，$S=\{j_4, j_2, j_6, j_3\}$。接下来，假设在时间点 t=300s 时，系统接收到 2 个新的作业请求，j_7、j_8，具体参数详见图 5。那么，首先计算 j_7、j_8 的排序得分，$Score(j_7)$=2.13，$Score(j_8)$=1.92，则优先考虑 j_7。接下来，根据定理 3、4，发现 j_7 可以插入到 j_6、j_3 之间，因此接收 j_7，并将有效序列更新为 $S=\{j_4, j_2, j_6, j_7, j_3\}$。类似地，判断 j_8，发现 j_8 无法插入到 S 当中，因此拒绝 j_8。图 5 描述了更新后的最终结果。

新作业集合

作业	$TC_m(j)$ (秒)	$TC_r(j)$(秒)	完成时限(秒)	收益	收益比	排序得分
j_7	100	120	920	230	2.13	2.13
j_8	150	100	950	250	1.92	1.92

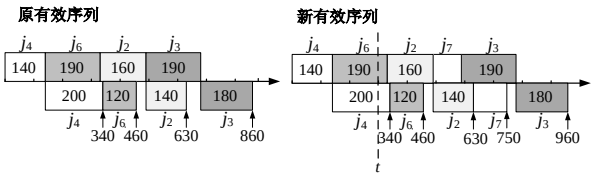


图 5 AMB 算法-动态部分

4.3 超时处理

本文的 AMB 算法主要针对同构集群设计，绝大部分情况下可以保证估计值 $j.T_m$、$j.T_r$ 贴近真实值。但在实际应用中，某些异常情况(如网络阻塞、节点宕机等)是无法完全避免的，会导致某些被接收的作业无法按时完成。在这种情况下，必须对运行中的作业进行合理的调整，才能够保证总体收益最大化。下面，我们给出一个作业超时的具体实例。

例 4 继续表 1 中所示的例子，经过 AMB 算法计算，我们已经得到了接收作业集合以及相应的有效序列$\{j_4, j_2, j_6, j_3\}$ (具体细节如图 4 所示)。但在系统执行 j_4 的 Map 任务时，出现部分节点宕机。在重启宕机节点之后，重新估计 j_4 的完成时间，发现 j_4 的 Map 阶段在时间点 170 才能完成。按照原有效序列计算各作业的完成时间，结果如图 6 所示。此时，作业 j_2 无法按时完成。

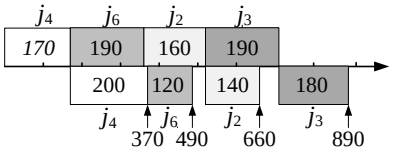


图 6 作业超时举例

当预计某些已被接收的作业无法按时完成时，就需要及时地放弃某些作业，以保证整体收益。一种最为基本的策略就是放弃正在运行的作业。因为按照原有效序列，所有未执行作业的参数均未发生变化，而某个作业无法按时完成的根源只是因为当前运行的作业无法按照原计划完成。例如在图 6 中，j_2 无法按时完成只是因为 j_4 的 Map 阶段完成时间变长。而放弃 j_4 就可以确保其他作业都按时完成。

算法 2. 超时处理算法

输入：接收作业集合 A，原有效序列 S，正在运行的作业 $j_{running}$

输出：需要放弃的作业集合 C

01. 将 A 中的作业按照收益由小到大排序；
02. IF $j_{running}$ 是收益最小的作业 THEN
03. $C=\{j_{running}\}$；
04. ELSE
05. FOR each $j_i \in A$ 且 $j_i.benefit < j_{running}.benefit$ DO
06. IF 在 S 中移除 j_i，S 变为有效序列 THEN
07. $C=\{j_i\}$，BREAK；
08. ELSE IF $j_i.benefit+\sum_{j\in C} j.benefit < j_{running}.benefit$ THEN
09. 将 j_i 加入 C 中；
10. IF 在 S 中移除 C 中的所有作业，S 变为有效序列 THEN
11. BREAK；
12. ENDIF
13. ELSE
14. $C=\{j_{running}\}$；
15. ENDIF
16. ENDFOR
17. ENDIF

但根据公式 1，为保证总收益 P 最大，应在保证剩余作业都能够按时完成的前提下，放弃收益最低的作业。基于这种思想，本文提出了一种新型的超时处理算法。算法 2 给出了该算法的具体过程。

例 5 在图 6 所示的例子中，按照作业收益进行排序，收益低于 j_4 的作业有 j_2、 j_6。首先尝试 j_2，发现放弃 j_2，序列{j_4, j_6, j_3}变为有效序列。则超时处理完成。

5 实验分析

在本小节中，我们首先基于 Hadoop 系统实现了 AMB 算法(详见附录 B)，并分别使用静态作业集和动态作业集测试了算法的性能。实验中所使用的 MapReduce 集群包括 1 个主节点和 40 个子节点。每个子节点上配置 2 个 Map 任务槽和 2 个 Reduce 任务槽，文件系统分片大小为 64MB。全部 41 个节点的配置均为：Intel Core i3 2100 3.1GHz CPU，8G 内存，500G 硬盘，操作系统为 Red Hat Linux 6.1。

5.1 静态作业集的实验结果与分析

在本部分实验中，所使用的作业主要包括如下 3 种常见的 MapReduce 作业：词频统计、倒排索引和分布式 Grep。输入的文件为维基百科提供的数据转储文件(主要内容是条目、模板、图片描述及基本的元页面等信息，链接：http://download.wikipedia.com/enwiki/)。对于一个申请作业集 J，我们主要考量如下 3 个主要参数对算法性能的影响：a) J 的规模 N：即 J 包含的作业个数；b) J 中作业的平均大小 L：我们用作业输入文件的分片数目来度量一个作业的大小，通常情况下，较大的作业需要更多的计算资源。在实验中，对于给定的 L，限定$\forall j \in J$，j 的大小在区间[0.5L,1.5L]内；c) J 中作业的平均完成时限 D：限定$\forall j \in J$，$j.deadline$ 为区间[0.5D,1.5D]内的一个随机值。此外$\forall j \in J$，j 的收益 $j.benefit$ 为[300,700]内的一个随机值，赔付比α=0.3，惩罚阈值β=2。表 2 给出了主要参数的默认值和变化范围。

表 2　实验默认参数

参数	默认值	变化范围
平均作业大小(分片个数)	150	100-200
申请集规模(作业个数)	30	20-40
平均完成时限(s)	1000	800-1600

本文以总收益 P(见公式 1)为首要性能指标。此外，本文还使用接收率和完成率以更全面地衡量算法的性能，具体地：

(1)接收率=接收作业集规模/申请作业集规模。

(2)完成率=按时完成的作业数目/接收集规模。

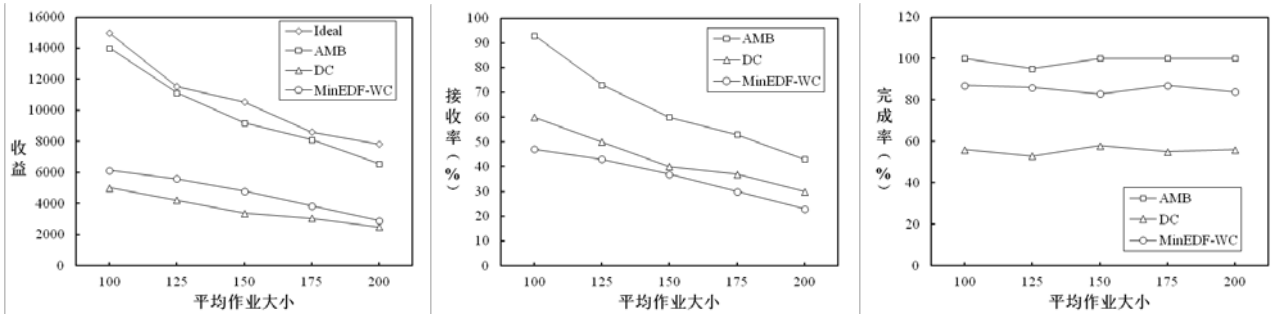
本文使用 DC 和 MinEDF-WC 作为对比算法。注意这 2 种算法仅针对具有完成时限约束的作业而设计，因此在确定接收作业集时，无法考虑收益。此外，在考量总收益时，我们引入了理想值的概念，即在最理想的状态下，假设系统的计算资源可以被收益比高的作业全部占用，且不必考虑赔偿时，企业可获得的最大收益。具体地，对于申请作业集合 J，找到具有最大完成时限的作业 j'，记 $Max\_time=j'.deadline$。接下来，将 J 中的作业按照收益比降序排序，得到有序集合 $J=\{j_1, j_2, ..., j_{|J|}\}$。确定 k 值，使得前 k 个作业的总系统耗时 $TC^k = \sum_{i=1}^{k} STC(j_i) \le Max\_time$，同时使得前 k+1 个作业的总耗时 $TC^{k+1} = \sum_{i=1}^{k+1} STC(j_i) > Max\_time$，则总收益的理想值为：

$$Ideal = \sum_{i=1}^{k} j.benefit + \frac{Max\_time - TC^k}{STC(j_{k+1})} \times j_{k+1}.benefit \quad (4)$$

可以看出，$Ideal$ 值是总收益的上限，在绝大多数情况下是无法达到的。例如在表 1 中，将作业按照收益比进行排序得到有序集合 $J=\{ j_5, j_4, j_2, j_6, j_3, j_1\}$，根据上述公式计算，$Max\_time$=1000，$k$=4，$TC^k$=440+164+152+162=918。也就是在理想状态下，系统可以接收的完整作业为 j_5, j_4, j_2, j_6，还可以接收一部分 j_3，那么理想值 $Ideal$=1360+380+340+350+(1000-918)/186≈2606。

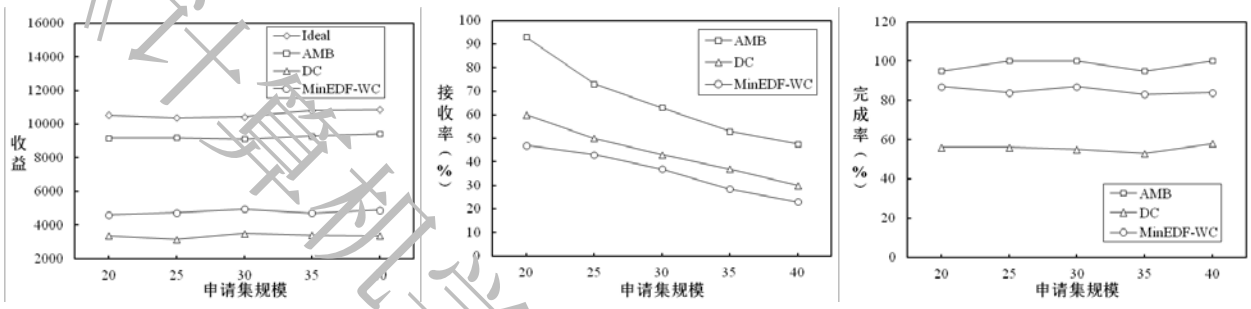
作业大小的影响

如图 7 所示，本文首先测试了不同作业大小对算法性能的影响。在图 7(a)中，我们测试作业大小对收益的影响，其中 Ideal 为公式 4 所示的理想值。可以看出，DC 和 MinEDF-WC 算法可以提供的收益较低。相比之下，AMB 算法可以带来非常高的收益，仅仅略低于理想值。随着作业逐渐增大，3 种算法的收益都会有所降低。在图 7(b)中，测试了作业大小对接收率的影响。在 3 种算法中，基于 SEQ 调度策略的 AMB 算法可以充分利用计算资源，提供了最高的接收率。而 DC 和 MinEDF-WC 算法使用较为简单的调度策略，无法充分发挥系统的计算能力，导致它们接收的作业数量较少。此外，由于完成时限不变，系统在固定时间段内的计算能力一定。因此，随着作业增大，3 种算法接收的作业数目都有所减少。在图 7(c)中，测试了作业长度对完成率的影响。容易看出，在 3 种算法中，完成率对作业长度的变化不敏感，并没有发生明显变化。其



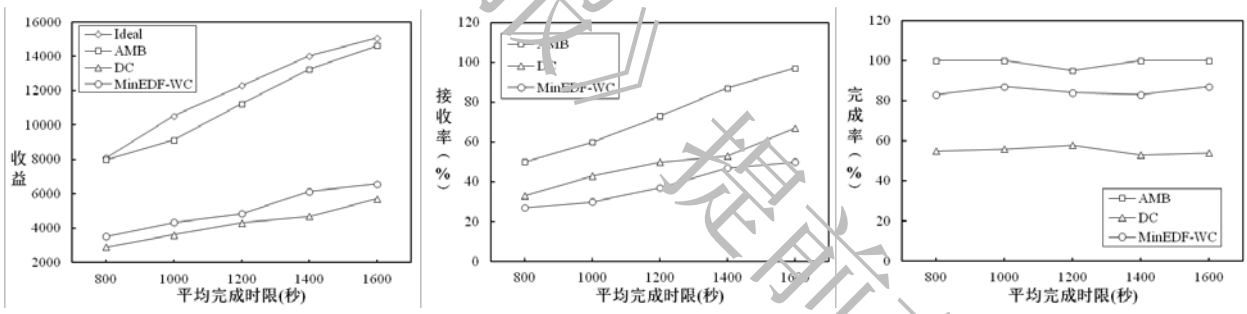
(a) 收益与平均作业大小的关系 (b) 接收率与平均作业大小的关系 (c) 完成率与平均作业大小的关系

图 7 作业大小的影响



(a) 收益与申请集规模的关系 (b) 接收率与申请集规模的关系 (c) 完成率与申请集规模的关系

图 8 申请作业集规模的影响



(a) 收益与平均完成时限的关系 (b) 接收率与平均完成时限的关系 (c) 完成率与平均完成时限的关系

图 9 完成时限的影响

中，AMB 算法使用 SEQ 策略进行任务调度，可以保证几乎全部被接收的作业都能够按时完成，超时处理算法几乎不会被调用。MinEDF-WC 算法也可以保持较高的完成率，而 DC 算法在 Reduce 任务调度策略上存在不足，导致一些作业进入 Reduce 阶段后被抛弃，因此利用率最低。另外，综合考虑图 7(b)和 7(c)的结果，我们可以发现 AMB 算法在接收率与完成率方面都占有绝对的优势，这也是该算法可以保持较高总收益的主要原因。

申请作业集规模的影响

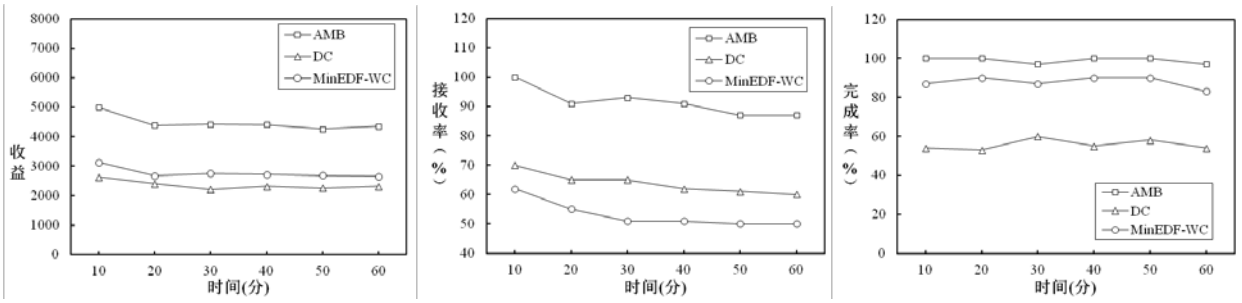
在图 8 中，我们对申请作业集规模的影响进行了测试。如图 8(a)所示，对于不同的申请集规模，总收益并没有明显变化。在图 8(b)中，随着申请集规模的增大，3 种算法的接收率都有所下降。这主要是由于完成时限没有变化，系统在一定时间段内所能接收的作业数目也不会变化，而申请总数增加，因此接收率降低。在图 8(c)中，测试了申请集规模对完成率的影响，与图 7(c)类似，完成率对作业集规模也不敏感，并未发生明显变化。

完成时限的影响

图 9 测试了完成时限对算法性能的影响。如图 9(a)所示，随着完成时限的增加，系统能够接收更多的作业，所以 3 种算法提供的收益都随之增加。类似地，在图 9(b)中，随着完成时限的增加，3 种算法的接受率也随之增加。图 9(c)反映了与图 7(c)和 8(c)相似的结果，本文不作赘述。

5.2 动态作业集的实验结果与分析

为了更贴近真实应用场景，本部分采用了动态



(a) 收益　　(b)接收率　　(c)完成率

图 10 监听结果(动态作业集)

的申请作业集合。具体地，实验中共生成了 60 个作业申请，其中作业的平均大小(分片数目)为 150，每个作业的完成时限设定为提交时间点之后的 400-600s 内的随机值，收益为[300,700]内的一个随机值，其他参数设定与 5.1 小节相同。在实验中，平均每隔 1 分钟从这 60 个作业中选取一个向集群提交，共持续 1 个小时。我们每隔 10 分钟对系统进行一次监测，得到这 10 分钟内集群的收益、作业的接收率和完成率。

图 10 描述了使用动态申请集合时的实验结果。不难发现，对于动态的作业申请，本文提出的 AMB 算法依然可以保持明显的优势，在收益、接收率和完成率方面，都明显优于 DC 和 MinEDF-WC 算法。特别地，观察图 10(a)和 10(b)发现，第一个 10 分钟观测期内的收益和接收率比其他观测期略高，这是因为在开始阶段，集群处于完全空闲状态，所有的 slave 节点都可以直接使用。而对于其他观测期，集群的一部分计算资源会被用于完成上个观测期遗留下的任务。

通过上述大量实验验证，我们进一步证明了本文提出的 SEQ 执行策略和 AMB 算法的有效性。

6 总结

本文首次提出了 MapReduce 集群中的最大收益问题，并给出了作业收益的评价标准。通过分析 MapReduce 作业的特点，本文提出了一种新型的任务调度策略，SEQ 策略，并证明了该策略在处理具有完成时限约束的作业时的良好性质。基于 SEQ 策略，本文了提出以最大收益为目标的 AMB 算法，该算法可以快速地确定可接收作业集合并给出有效的作业执行方案，以保证所有接收的作业都可以按时完成，可以最大化收益。此外，针对实际应用中无法避免的某些异常情况(如节点宕机等)，本文设计了有效的超时处理策略。最后，通过大量实验验证了本文提出算法的有效性和实用性。

参 考 文 献

[1] Wang Shan, Wang Hui-Ju, Qin Xiong-Pai, Zhou Xuan. Architecting big data: challenges, studies and forecasts. Chinese Journal of Computers, 2011, 34(10): 1741-1752(in Chinese)
(王珊, 王会举, 覃雄派, 周烜. 架构大数据: 挑战、现状与展望. 计算机学报, 2011, 34(10): 1741-1752)

[2] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters//Proceedings of the 6th Symposium on Operating Systems Design and Implementation. San Francisco, USA, 2004: 1-13

[3] Peha J-M, Tobagi F-A. A cost-based scheduling algorithm to support integrated services//Proceedings of the 10th IEEE International Conference on Computer Communications. Bal Harbour, Florida, USA, 1991: 741-753

[4] Chi Y, Moser H-J, Hacigümüş H. ICBS: incremental cost-based scheduling under piecewise linear SLAs//Proceedings of the 37th International Conference on Very Large Data Bases. Washington, USA, 2011: 563-574

[5] Ghemawat S, Gobioff H, Leung S-T. The google file system// Proceedings of the 19th ACM Symposium on Operating Systems Principles. New York, USA, 2003, 37(5): 29-43

[6] Schwarzkopf M, Konwinski A, Abd-El-Malek M, et al. Omega: flexible, scalable schedulers for large compute clusters//Proceedings of the 8th ACM European Conference on Computer Systems. Prague, Czech Republic, 2013: 351-364

[7] Wolf J, Balmin A, Rajan D, et al. CIRCUMFLEX: a scheduling optimizer for MapReduce workloads with shared scans. ACM SIGOPS Operating Systems Review, 2012, 46(1): 26-32

[8] Condie T, Conway N, Alvaro P, et al. MapReduce online//Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation. San Jose, California, USA, 2010: 313-328

[9] Zaharia M, Borthakur D, Sen Sarma J, et al. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling//Proceedings of the 5th ACM European Conference on

Computer Systems. Paris, France, 2010: 265-278.

[10] Sandholm T, Lai K. Dynamic proportional share scheduling in hadoop//Proceedings of the 15th Job Scheduling Strategies for Parallel Processing. Atlanta, Georgia, USA, 2010: 110-131

[11] Zaharia M, Konwinski A, Joseph A D, et al. Improving MapReduce performance in heterogeneous environments//Proceedings of the 10th Symposium on Operating Systems Design and Implementation, San Diego, California, USA, 2008: 29-42

[12] Kwon Y-C, Balazinska M, Howe B, et al. Skewtune: mitigating skew in MapReduce applications//Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. Scottsdale, Arizona, USA, 2012: 25-36

[13] Polo J, Carrera D, Becerra Y, et al. Performance-driven task co-scheduling for MapReduce environments//Proceedings of the 2010 IEEE Network Operations and Management Symposium. Osaka, Japan, 2010: 373-380

[14] Kc K, Anyanwu K. Scheduling hadoop jobs to meet deadlines//Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science. Indianapolis, Indiana, USA, 2010: 388-392

[15] Verma A, Cherkasova L, Campbell R-H. ARIA: automatic resource inference and allocation for MapReduce environments//Proceedings of the 8th ACM International Conference on Autonomic Computing. Karlsruhe, Germany, 2011: 235-244

[16] Verma A, Cherkasova L, Kumar V-S, et al. Deadline-based workload management for mapreduce environments: pieces of the performance puzzle//Proceedings of the 2012 IEEE Network Operations and Management Symposium. Maui, Hawaii, USA, 2012: 900-905

附录

A 参数估计模块：

对于任意作业 j，需要估计的参数包括：a) j 的平均每个 Map 任务的处理时间 $j.T_m$；b) j 的平均每个 Reduce 任务的处理时间 $j.T_r$。本文采用一种较为通用的抽样估计方法来计算上述参数，类似的计算方法已在相关研究(如文献[11,13]等)中普遍采用。具体的细节描述如下。

对于作业 j，首先随机的抽取其一小部分输入数据，在一台计算机上独立运行该作业(只包括一个 Map 任务和一个 Reduce 任务，该过程可在客户端或集群主控节点实现)。在抽样过程中，若记文件分片大小为 s，抽取的数据量为 f，Map 阶段的运行时间为 t_m，Reduce 阶段的运行时间为 t_r，那么：

$$\begin{cases} j.T_m = \dfrac{t_m \times s}{f} \\ j.T_r = \dfrac{t_r \times s \times j.N_m}{f \times j.N_r} \end{cases} \tag{5}$$

根据上述参数，还可以容易地计算出 j 的 Map 阶段耗时 $TC_m(j)$和 Reduce 阶段耗时 $TC_r(j)$，相关计算方法已在 4.1 小节中给出，这里不再赘述。

例如：假设一个 MapReduce 集群具有 20 个 Map 任务槽和 20 个 Reduce 任务槽，输入文件分片大小为 64MB。系统接收到一个新作业 j，$j.N_m$=40，$j.N_r$=20。那么，参数估计模块首先抽取一小部分输入数据(假设为 8MB)，并在一台独立的计算机上运行作业 j，得到 Map 阶段耗时为 5s，Reduce 阶段耗时为 4s。那么根据公式 5，可以得到 j 的平均每个 Map 任务的处理时间 $j.T_m$=5×64/8=40s，平均每个 Reduce 任务的处理时间 $j.T_r$=4×64×40/(8×20)=64s。进一步地，可以估计得到 Map 阶段耗时 $TC_m(j)$=⌈40×40/20⌉=80s，Reduce 阶段耗时 $TC_r(j)$=⌈64×20/20⌉=64s。

不可否认，采用抽样方法得到的参数值不是十分精确，但以目前的技术手段也无法从根本上的解决上述问题。为进一步降低参数值不准确对本文算法性能的影响，可以考虑加入松弛因子。具体地，可以根据集群运行的历史情况设定一个松弛因子ρ(ρ>1，集群的处理能力越不稳定，ρ值越大)。当一个新作业 j 被提交时， j 的 Map 阶段耗时的计算公式被调整为 $TC_m(j)$=⌈$j.N_m/M$⌉×$j.T_m$×ρ，Reduce 阶段的耗时调整为 $TC_r(j)$=⌈$j.N_r/R$⌉×$j.T_r$×ρ。通过使用松弛因子ρ来适当放大估计值，可以降低接收作业超时的风险。当某个作业处理完毕时，我们对有效序列进行矫正，降低松弛因子带来的资源浪费。

B 基于 Hadoop 的 AMB 算法实现：

在 Hadoop 中，任务调度器是一个独立的可插拔模块，系统允许用户根据个人需求自定义调度算法。具体地，用户只需要继承抽象类 TaskScheduler 并在配置文件中进行设定，就可以轻松的将自定义调度器加载到系统中使用。而本文的 AMB 算法就是使用上述方法实现的。注意 AMB 算法仅需要在生成有效序列时进行一定量的计算。一旦生成有效序列之后，调度器只需要按照序列的顺序进行任务分配即可(具体过程与默认的 FIFO 调度器相似)。因此，任务调度过程不会对系统调度逻辑等产生影响。

WANG Xi-Te, born in 1987, Ph. D. student. His research area includes big-data management.

SHEN De-Rong, born in 1964, Ph. D., professor, Ph. D. supervisor. Her research area includes distributed data management and data integration.

YU Ge, born in 1962, Ph. D., professor, Ph. D. supervisor.

His research area includes database and big-data management.

BAI Mei, born in 1986. Ph. D. student. Her research area includes sensor data management and uncertain data management.

Background

Along with the development of information technology, the amount of data generated by every walk of life becomes extremely huge, and the big-data management becomes a very hot topic. As a popular parallel data processing framework, MapReduce is a highly desirable technique for the big-data analysis. With the help of MapReduce, users can easily process terabyte-level data without considering the execution details (e.g. data distribution, fault-tolerance).

Many companies have built their MapReduce clusters to provide computing services for users. Users can submit their deadline-constraint MapReduce jobs to the cluster. If the job is accepted and finished before its deadline, the company can get some benefits from the user. If the job is accepted but not finished on time, user will receive some compensations paid by the company. In order to achieve maximum benefit, the company needs to accept the MapReduce jobs discretely and adopts suitable scheduling algorithm to guarantee that all the accepted jobs can be finished on time.

In response to the application scenario above, in this paper, we first present the maximum benefit problem in MapReduce. To solve this problem effectively, we propose SEQ strategy and a novel Algorithm for Maximum Benefit (AMB for short). AMB can efficiently determine the acceptable job set and provide the effective execution strategy which can guarantee that all the accepted jobs are finished on time. We also propose a timeout-handling method which can further improve the practicality of the algorithm. At last, the effectiveness of the proposed algorithm is verified through plenty of experiments.

There have been several scheduling algorithms which are designed for the deadline-constraint jobs in MapReduce. However, none of them has considered the maximum benefit problem, and them are not suitable for the application scenario in this paper.

This work is supported by the National Basic Research 973 Program of China under Grant No.2012CB316201, the National Natural Science Foundation of China under Grant Nos. 61033007, the National Research Foundation for the Doctoral Program of Higher Education of China Grant No. 20120042110028 and the MOE-Intel Special Fund of Information Technology under Grant No. MOE-INTEL-2012-06.

NIE Tie-Zheng, born in 1980, Ph. D., associate professor. His research area includes data quality and data integration.

KOU Yue, born in 1980, Ph. D., associate professor. Her research area includes entity resolution.