计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(2):219−225 [doi: 10.15888/j.cnki.csa.007792]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

# 用于 Hadoop2.x 的 MapReduce 性能评估模型①

吴　岳

(国家林业和草原局 林产工业规划设计院, 北京 100010)
通讯作者: 吴　岳, E-mail: wuyue98@126.com

**摘　要**: 基于 MapReduce 的程序被越来越多地应用于大型数据分析的应用中. Apache Hadoop 是最常用的开源 MapReduce 模型之一. 程序运行时间的缩短对于 MapReduce 程序以及所有数据处理应用而言至关重要, 而能够准确估算 MapReduce 程序的执行时间是优化程序的重要环节. 本文定义了一个在 Hadoop2.x 版本中能够准确估算 MapReduce 作业负载执行时间的性能模型. 该模型包括一个优先级树模型与一个排队网络模型, 分别用于展示一个 MapReduce 作业中不同任务之间的依赖关系及 MapReduce 作业内的同步约束. 最后, 实验证明了该模型的可用性.

**关键词**: MapReduce 性能模型; Hadoop2.x; 队列模型; 均值算法

引用格式: 吴岳.用于 Hadoop2.x 的 MapReduce 性能评估模型.计算机系统应用,2021,30(2):219–225. http://www.c-s-a.org.cn/1003-3254/7792.html

## MapReduce Performance Evaluation Model for Hadoop2.x

WU Yue

(Forest Industry Planning and Design Institute, National Forestry and Glassland Administration, Beijing 100010, China)

**Abstract**: MapReduce-based systems are increasingly being used for large-scale data analysis applications. Apache Hadoop is one of the most common open-source implementations of such paradigm. Minimizing the execution time is vital for MapReduce as well as for all data-processing applications, and the accurate estimation of execution time is essential for optimization. In this study, the author created a MapReduce performance model for Hadoop2.x that can precisely estimate the execution time of workload in MapReduce. This model combines a precedence tree model that can capture dependencies between different tasks in one MapReduce job, and a queueing network model that can capture the intra-job synchronization constraints. Such an analytical performance model is a particularly attractive tool as it might provide reasonably accurate job response time at significantly lower cost than the simulation experiment of real data-analysis systems. Furthermore, a clear understanding of systematic job response time under different circumstances is key to making decisions in MapReduce workload management and resource capacity planning.

**Key words**: MapReduce performance model; Hadoop2.x; queuing theory; mean value analysis

基于 MapReduce 的程序被越来越多地应用于大型数据分析的应用中. 程序运行时间的缩短对于 MapReduce 程序以及所有数据处理应用而言至关重要, 而能够准确估算 MapReduce 程序的执行时间是优化程序的重要环节. 因此, 作者需要建立基于 MapReduce 程序数据处理应用的性能模型. 分析基于 MapReduce 程序性能模型的实验成本远低于在真实数据分析系统中设置模拟实验, 也可以得到准确的系统作业响应时间, 准确了解不同状况下的系统作业响应时间是对 MapReduce 作业负载管理和资源分配规划做出决策的重要依据.

基于 MapReduce 程序编写需要将算法调整为两

① 收稿时间: 2020-06-29; 修改时间: 2020-07-27; 采用时间: 2020-07-31; csa 在线出版时间: 2021-01-27

阶段的处理模型, 即 Map 模型和 Reduce 模型. 以这种方式编写的程序会自动在计算集群上并行执行. Apache Hadoop 是最常用的开源 MapReduce 模型之一. 在 Hadoop1.x 版本中, MapReduce 程序模型与资源管理是整合在一起的. 为了达到更高的集群利用率、可靠性和可用性, 支持编程模型多样性、向后兼容性以及弹性资源模型, Hadoop2.x 版本的体系结构进行了重大改进, 引入了 YARN 框架 (Yet Another Resource Negotiator)——独立的资源管理模块. Hadoop2.x 版本的体系结构发生了明显改变, YARN 框架将 MapReduce 编程模型从资源管理结构中分离出来, 集群资源被认为是一个整体, 没有被静态地划分给每个 Map 和 Reduce 作业.

本文定义了一个在 Hadoop2.x 版本中能够准确估算 MapReduce 作业负载执行时间的性能模型. 该模型包括一个优先级树模型与一个排队网络模型, 优先级树模型可以展示一个 MapReduce 作业中不同任务之间的依赖关系, 排队网络模型可以展示 MapReduce 作业内的同步约束.

通过分析 Hadoop2.x 的体系结构, 作者确定了可能影响 MapReduce 作业执行开销的因素, 为 Hadoop2.x 定义了理论上的 MapReduce 开销模型, 该模型展示了不同 MapReduce 作业的优先级以及由于共享资源导致的同步延迟. 通过比较开销模型的估算数据与 MapReduce 的实际执行数据来评估开销模型的准确性.

## 1 适用于 Hadoop1.x 的性能评估模型

有两种方法可以分析 Hadoop1.x 中 MapReduce 作业的性能, 分别为静态 MapReduce 性能模型和动态 MapReduce 性能模型. 静态 MapReduce 性能模型中不考虑由于争用共享资源导致的排队延迟与不同 MapReduce 作业之间的同步延迟. 动态 MapReduce 性能模型中考虑了并发 MapReduce 程序执行与排队延迟.

### 1.1 静态 MapReduce 性能模型

静态 MapReduce 性能模型可以描述 Hadoop1.x 中 MapReduce 作业的执行情况. 性能模型以细粒度的方式描述 MapReduce 作业内各个阶段的数据流和开销情况. Map 作业可分为: Read, Map, Collect, Spill、Merge 五个阶段. Reduce 作业的 Shuffle、Merge、Reduce、write 阶段均有独立的公式. 整个工作的执行时间可以认为是 Map 和 Reduce 作业所有阶段执行时间的总和.

该模型为 MapReduce 作业完成时间与 MapReduce 作业资源分配定义上限值和下限值, 在给定的作业完成期限内分配满足该 MapReduce 作业所需的资源, 以便 MapReduce 操作在要求的期限内完成. 该框架由 3 个相关联的部分组成. (1) 作业配置文件, 其中该应用程序 Map 和 Reduce 阶段的性能特征. (2) 构建一个 MapReduce 性能模型, 该模型根据给定作业截止期限来估算在截止期限内完成作业所需的资源量. (3) 调度程序本身, 它确定 MapReduce作业顺序以及在截止日期之内完成 MapReduce 作业所需的资源量.

MapReduce 作业完成时间与输入数据集的规模及分配资源的数量有关, 定义作业完成时间的上限 $T_J^{Up}$, 作业完成时间下限为 $T_J^{Low}$, 平均完成时间 $T_J^{Avg}$. 假设 $T_J^{Avg}=(T_J^{Up}+T_J^{Low})/2$, 由于误差等原因, 作业平均完成时间会比作业实际完成时间少 15% 左右, 因此基于作业平均完成时间的预测很适合确保作业在截止日期之前完成, 是最接近实际作业完成时间的. 在 Hadoop1.x 中, 用于 Map 和 Reduce 作业的资源数量是预先确定的并不会更改. 在 Hadoop2.x 中, YARN 完全将静态资源分配从 MapReduce 作业中分离出来, 并且不能忽略作业之间的依赖关系, 因此静态 MapReduce 性能模型不再适用.

### 1.2 动态 MapReduce 性能模型

研究 MapReduce 性能模型的最大难点是, 必须精准地捕获到 MapReduce 作业执行过程中不同原因造成的延迟. 特别是, 属于某个 MapReduce 作业的任务可能会发生两种类型的延迟: 由于争用共享资源产生的排队延迟, 在同一 MapReduce 作业中进行协作的任务之间由于优先级约束而导致的同步延迟. 在不考虑同步延迟的条件下, 主要有两种方法可以评估并行应用程序的 MapReduce 作业负载性能. 第一种方法是平均值分析 (MVA). MVA 仅考虑任务由于共享公共资源而导致的排队延迟. 因此, MVA 无法直接应用于具有优先级约束的工作负载, 例如同一个 MapReduce 作业中 Map 和 Reduce 任务的同步. 另一种典型的解决方案是利用马尔可夫链来表示系统的状态, 并对网络模型进行排队, 以计算系统不同状态之间的转换率. 但是, 在 MapReduce 作业中, 系统状态的数量会随着任务数量的增加呈指数增长, 所以此种方法无法应用于具有多个任务的 MapReduce 作业模型中[1].

## 2 Hadoop2.x 的体系结构

### 2.1 YARN 框架的构成

Hadoop2.x 中的最大变化就是出现了 YARN 框架,

它主要负责管理集群资源与作业调度. 在 Hadoop1.x 中, 这个功能是集成在 MapReduce 框架中, 由 JobTracker 组件实现. YARN 框架的基本思想是接替 JobTracker 的两个主要功能, 即资源管理和任务调度/监控, 所以 YARN 拥有面向全局的 ResourceManager 和面向每一个应用程序的 ApplicationMaster. 通过将资源管理与编程模型分离, YARN 将集群资源认为是一个整体, 没有被静态地划分给每个 Map 和 Reduce 作业, 显著提高了集群资源利用率.

YARN 模块包含 3 个主要组件:

(1) ResourceManager(RM): RM 负责整个集群的资源管理和分配, 是一个全局的资源管理系统.

(2) NodeManager(NM): NM 是每个节点上的资源和任务管理器, 它是管理这台机器的代理, 负责该节点程序的运行, 以及该节点资源的管理和监控. YARN 中每个节点都运行一个 NodeManager.

(3) ApplicationMaster(AM): 用户提交的每个应用程序均包含一个 AM, AM 负责与 RM 协商以获取资源 (用 Container 表示)[2].

### 2.2 Hadoop2.x 中的资源管理

了解资源请求过程是构建 Hadoop2.x 性能模型的关键. 在 Hadoop2.x 中, AM 可以以静态模式或者动态模式提出它需要的资源请求. 如果对资源的要求是在应用程序提交时确定的, 并且在 AM 开始运行时该资源请求没有变化, 那么可以使用静态方式请求资源. 例如在 MapReduce 作业中, 通过输入拆分产生的 Map 任务数量与通过用户定义参数产生的 Reduce 数量总和是固定不变的. 动态资源请求方式是指在应用程序提交时无法确定, 而需要 AM 根据用户指定、集群资源的可用性、业务逻辑等条件, 在运行时选择需要请求多少资源. 无论使用哪种资源请求模式, RM 都不会立即分配 Container 给 AM. AM 发送分配的请求后 RM 最终将根据集群容量, 优先级和调度策略分配容器. 当且仅当其原始估算值发生变化并且需要其他容器时, AM 才应再次向 RM 请求容器.

### 2.3 Hadoop2.x 中的作业调度

构建性能模型需要了解在不同节点中为任务分配容器的方式. 通过分析 MapReduce 的源代码 Package (org.apache.hadoop.mapreduce.v2.app.rm) 和 JavaClass (RMContainerAllocator), 可以得到 Map 和 Reduce 任务具有不同的生命周期. Map 任务的生命周期分为 3 个过程: Scheduled、Assigned 和 Completed. Reduce 任务的生命周期分为 4 个过程: Pending、Scheduled、Assigned 和 Completed. 在 Pending 过程中, 资源请求尚未发送给 RM. 在 Scheduled 过程, 资源请求已经发送给 RM 但是并未被分配. 在 Assigned 过程, 资源请求被分配到一个容器中. 在 Completed 过程, 被请求的容器已经完成执行.

此外, AM 可以进行第二级的调度, 并将其获得的容器分配给作业执行计划中的任何任务. 因此, YARN 中的资源分配是后期绑定. AM 仅负责使用容器提供的资源, 而不会一定将该资源分配给最初请求资源的逻辑任务. 当 AM 接收到一个容器时, 它将使该容器与待处理任务集进行匹配, 选择一个输入数据最接近该容器的任务, 首先尝试执行本地数据任务, 然后后退到本地机架[3,4].

## 3 算法设计

在本文中, 作者以 Hadoop1.x 中 MapReduce 作业性能模型为基础加以改进, 使它能够适应 Hadoop2.x 的体系结构. 需要改进方面主要有以下两处:

(1) 与 Hadoop1.0 中为每个 Map 和 Reduce 任务预先配置资源不同, 考虑到 Hadoop2.x 中资源的动态分配, 需要构建一个优先级树.

(2) 在 Map 任务和 Reduce 任务的 Shuffle 阶段因为引入管线会发生同步延迟, 需要在性能模型中及时捕获到这种延迟.

### 3.1 输入参数

为了简单起见, 作者设计了一组数量为 *numNodes* 的计算节点, 所有计算节点都具有相同的技术特征. 工作负载由系统中同时执行的 $N$ 个 MapReduce 作业组成. 每个作业都有 $mi$ 个 Map 任务和 $ri$ 个 Reduce 任务. 由于每次 Shuffle 操作后都会执行部分 Sort 操作, 所以将每对 Shuffle 操作与 Sort 操作进行分组, 形成一个单独的子任务, 这个子任务称为 Shuffle-Sort 任务. 在 Reduce 任务的最后阶段, 在完成所有的部分 Sort 操作后, 将执行最终 Sort 操作, 将最终 Sort 操作和 Reduce 函数分组到一个子任务中, 这个子任务称为 Merge 任务. 因此, Reduce 任务分为两个子任务: Shuffle-Sort 任务和 Merge 任务. 表 1 列出了性能模型的输入参数. 共有两种类型的服务资源: CPU-内存和网络. 用常量 C 表示任务类别的总数, 它的取值为 3 (即 Map 任务、Shuffle-Sort 任务和 Merge 任务).

表 1 输入参数表

| | 符号 | 输入参数 |
|---|---|---|
| 配置参数 | *numNodes* | 集群中的节点数量 |
| | *cpuPerNode* | 每个节点的CPU数量 |
| | *diskPerNode* | 每个节点的磁盘数量 |
| 作业负载参数 | $S_{ik}$ | 资源$k$中$i$类任务的停留时间 |
| | $AvgResponseTime_i$ | $i$类任务的响应时间 |
| | *m* | Map任务的数量 |
| | *r* | Reduce任务的数量 |
| | *MaxMapPerNode* | 每个节点的容器中Map任务的最大数量 |
| | *MaxReducePerNode* | 每个节点的容器中Reduce任务的最大数量 |

### 3.2 改进后的均值分析 (MVA) 算法

作者使用了改进的均值分析算法来解决排队网络模型问题. 假设一个系统有 $C$ 种任务类型和 $K$ 个服务资源. 有一个向量$\vec{N}$, 其中第 $i$ 个分量表示系统中第 $i$ 类任务的数量. $S_{jk}$ 表示服务资源 $k$ 中任务类型 $j$ 的平均资源需求 ($j \in C, k \in K$).

这个算法主要分为 6 个步骤, 如图 1 所示①~⑥.

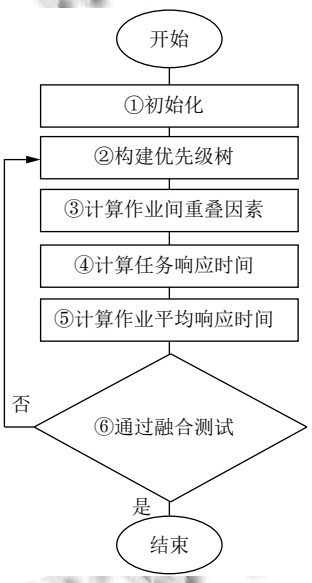


图 1 改进后的均值算法的主要步骤

在①阶段, 初始化每个服务资源中每类任务的平均停留时间和系统中每个任务的平均响应时间. 在②阶段, 基于每个独立任务的平均响应时间构建优先级树. 在③至⑤阶段, 相同作业任务和不同作业任务在执行时间上的叠加会产生排队延迟, 考虑到排队延迟因素带来的影响, 需要重新估算任务平均响应时间. 在⑥阶段, 在新估算出的任务平均响应时间上进行融合测试. 如果融合测试失败了, 使用③至⑤阶段得到的任务响应时间, 回到②阶段重新构建优先级树, 这个优先级树会比之前构建的更加准确. 如果当前的估算值足够接近先前的估算值, 算法将结束, 由此产生最终的作业平均响应时间.

(1) 初始化任务响应时间

初始化过程包含可以并发执行的两个子过程: 初始化每个服务资源中每类任务的平均停留时间和初始化系统中每个任务的平均响应时间. 要初始化任务驻留时间, 可以从相应的实际 Hadoop 作业执行历史中获取了驻留时间的平均值. 要初始化任务响应时间, 可以假设优先执行所有的 Map 任务, 然后执行 Reduce 任务, 于是将所有可用资源先提供给 Map 任务, 再提供给 Reduce 任务. 这种方法可以通过较少的算法迭代产生更多准确的响应时间初始化.

(2) 构建优先级树

在优先级树中, 每个叶子代表一个任务, 每个内部节点都是一个运算符, 描述了任务执行过程中的约束. 使用串行 (S) 和并行 (P) 两种基本操作符号构建的优先级二叉树. 串行 (S) 运算符用于连接顺序执行的任务, 并行 (P) 运算符用于连接并发执行的任务. 优先级树的示例如图 2 所示.

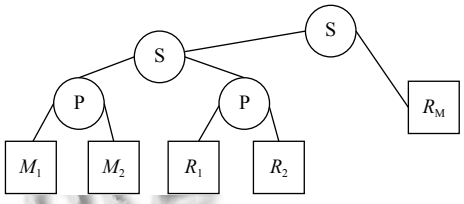


图 2 优先级树示例

构建优先级树的主要目标是捕获作业的执行流程, 确定各个任务的执行顺序是串行还是并行, 以及各个任务之间的依赖性. 在算法的每次迭代时会重新构建优先级树, 用于重新估计任务响应时间[5].

优先级树取决于各个任务的响应时间, 使用任务响应时间线构建. 基于获得的时间线, 可以构造唯一的优先级树. 将所有同时执行的任务中耗时最长的称为一个阶段, 为了能够区分任务是顺序执行还是并发执行, 必须确定时间线中新阶段的开始点. 将时间线划分为多个阶段. 同一阶段内的所有任务并发执行, 而属于不同阶段的任务则顺序执行. 这意味着每个任务的开始或结束都会有一个新阶段开始. 时间线构造算法如算法 1 所示.

算法 1. 时间线构造算法

输入: Map 任务集合 $M$, Reduce 任务集合 $R$, 集群节点集合 $N$, Map 任务所需的容器数 $m$, Reduce 任务所需的容器数 $r$
输出: 时间线 $TL$

{$st$: 开始时间; $et$: 结束时间; $d$: 持续时间; $sd$: Shuffle 阶段持续时间; $an$: 被委派节点;}
1. 当 $i$ 从 1 到 $N$ 时, 循环执行
2. 时间线集合 $TL[i]$ 为空集Ø;
3. 循环结束
4. 当 $m$ 属于 $M$ 时, 循环执行
5: $i$ 为 $TL$ 中的最小值;
$m$ 中被委派节点 $m.an$ 等于 $i$;
$m$ 的开始时间 $m.st$ 等于 $TL[i]$ 中最小值;
$m$ 的结束时间 $m.et$ 等于 $m$ 的开始时间 $m.st$ 加上 $m$ 的持续时间 $m.d$;
$TL[i]$ 等于 $TL[i]$ 与{$m$}的并集;
6. 循环结束
7. 如果设置为 $slow\_start$, 那么
8. 边界值 $border$ 等于 $TL[TL_{min}]$ 的结束时间;
9. 否则
10. 边界值 $border$ 等于 $TL[TL_{max}]$ 的结束时间;
11. 条件判断结束
12. 当 $r$ 属于 $R$ 的时候, 循环执行
13. $i$ 为 $TL$ 中的最小值;
$r$ 中被委派节点 $r.an$ 等于 $i$;
$r$ 的开始时间 $r.st$ 等于 $TL[i]$ 的结束时间与边界值 $border$ 中的较大值;
14. 当 $m$ 属于 $M$ 时, 循环执行
15. 如果 $m.an$ 不等于 $i$, 那么
16. $r.d$ 等于 $r.d$ 加上 $m.sd$ 除以$|R|$;
17. 条件判断结束
18. 循环结束
19. $r$ 的结束时间 $r.et$ 等于 $r$ 的开始时间 $r.st$ 加上 $r$ 的持续时间 $r.d$;
20. $TL[i]$ 等于 $TL[i]$ 与{$r$}的并集;
21. 循环结束
22. 返回值 $TL$;

由于 Map 任务比 Reduce 任务具有更高的优先级, 算法从 1~6 行开始为 Map 任务分配容器. 如果设置为 slow_start 时, 则 Reduce 任务 Shuffle 阶段的开始将与占用率最低的节点上的第一个 Map 任务的结束重合. 因此, Shuffle 阶段尽将会可能早地开始. 如果没有设置为 slow_start 时, Reduce 任务的 Shuffle 阶段将会尽可能晚地开始, 如同算法的第 7~11 行. 算法在第 12~21 行为 Reduce 任务分发了容器.

得到时间线后, 可以基于时间线构建一个二进制优先级树, 为了减少二进制优先级树的最大深度对其每一个 P 子树应用平衡处理. 时间线与优先级树分别如图 3 和图 4 所示.

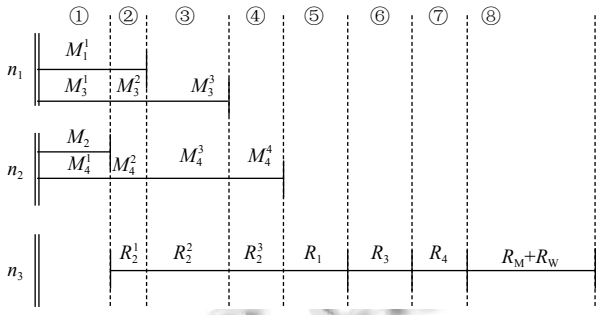


图 3 时间线

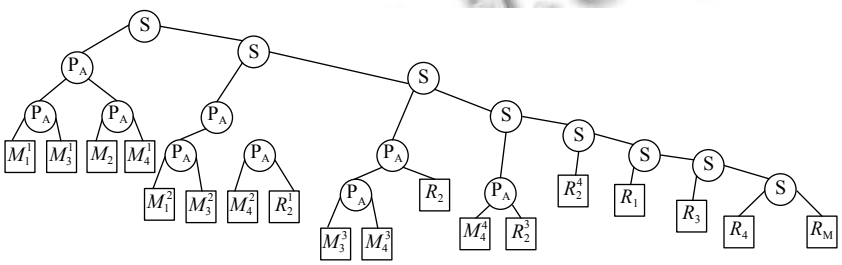


图 4 优先级树

(3) 计算作业内与作业间重叠因子

对于具有多个任务类型的系统, 由于 $j$ 类任务所导致 $i$ 类任务的排队延迟与它们的重叠成正比. 主要有两种类型的重叠因子: 作业内重叠因子$\alpha_{ij},\forall i,j$表示来自同一工作的表示任务的 ID; 工作间重叠因子$\beta_{kr},\forall k,r$表示来自不同工作的任务 ID. 工作内和工作间重叠因素的示例如图 5 所示.

(4) 计算平均作业响应时间

使用基于 Fork/Join 框架的算法来估算作业响应时间, 将并行阶段的作业执行看作 Fork/Join 块, 估算 Fork/Join 的平均响应时间为 $k$ 个任务中平均响应时间的最大值与第 $k$ 次调和函数的乘积. 公式如下:

$$R_{ik} = H_k \cdot \max\left(T_i, T_j\right) \tag{1}$$

其中, $H_k$=∑$s_i$=1/$i$, $s$ 为优先级树子节点的数量.

优先级树是一个二进制树, 所以 $H_k$=3/2, ∀$k$, 父节点的响应时间等于最大子节点的响应时间加上可能的延迟[6].

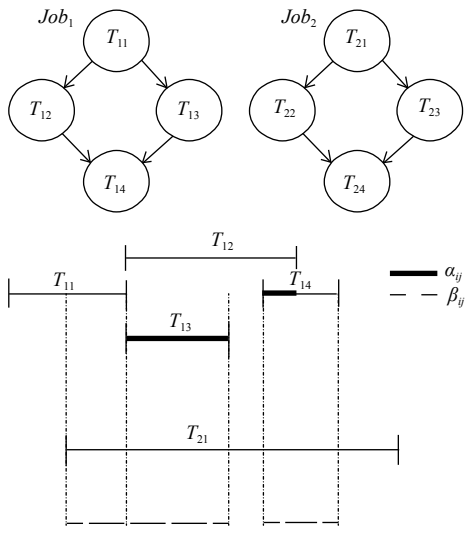


图 5 工作内和工作间重叠因素

(5) 融合测试

在融合测试阶段, 比较前一次迭代产生的总响应时间总值与当前计算产生的响应时间总值, 当二者足够接近时 (例如 *Rcurr-Rprev*<$10^{-7}$), 算法执行完毕. 否则, 算法将返回到优先级树构建阶段重新执行. 经试验测试, 将差值标准定为 $10^{-7}$, 可在精度级别和算法复杂性 (迭代次数) 之间提供良好的折衷方案. 当这个差值值较低时, 计算出的作业响应时间几乎不再改变, 但是算法迭代次数会继续增长[7].

## 4 实验

### 4.1 试验设置

实验中使用基于 Fork/Join 框架的模型与 Hadoop1.x 原生方法的测量结果进行比较. 设计一组实验参数, 在 Map-Reduce 任务中输入大量作业 (例如 word-count 程序), 输入数据经过处理会生成大量中间数据用来分析作业响应时间. 分析每个实验中固定 3 个参数中两个参数的工作响应时间, 每个实验重复 5 次, 然后取响应时间的中位数.

实验参数如下:

(1) 节点数量: 4 个, 6 个, 8 个.

(2) 输入数据量: 1 GB, 5 GB.

(3) 集群中同时执行的任务数量: 1~4 个.

(4) 集群中每个节点的配置相同: XeonE5-2630@2.4 GHz/128 GB /1 TB /4 GiEthernet.

### 4.2 试验结果

使用固定的输入数据, 在集群中使用不同数量的节点 (4 个, 6 个, 8 个), 并发地执行不同数量的作业 (1 个, 4 个). 图中使用实线表示 Hadoop1.x 原生值, 使用虚线表示基于 Fork/Join 框架的计算值. 输入数据量为 1 GB 和 5 GB 的实验结果分别如图 6–图 9 所示. 使用固定输入数据在集群中同时执行不同的作业数量 (从 1 个到 4 个) 的响应时间如图 10 所示.

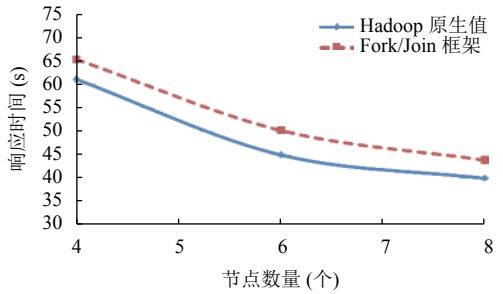


图 6 作业数量 1, 输入数据量 1 GB

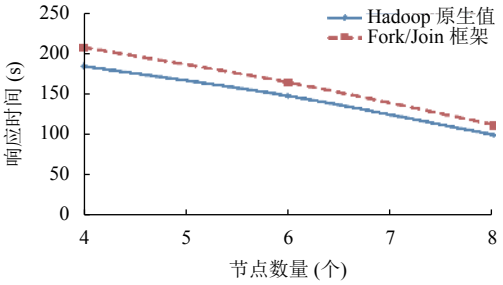


图 7 作业数量 4, 输入数据量 1 GB

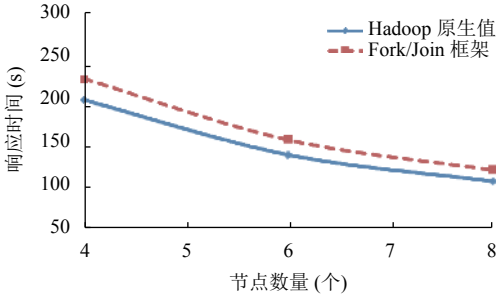


图 8 作业数量 1, 输入数据量 5 GB

通过实验发现, 基于 Fork/Join 框架的算法的误差在 11% 到 13.5% 之间, 当输入数据量为 5 GB 的时候, 误差值达到最大 13.5%. 假设算法的准确性取决于 Map 任务的数量而不是输入数据量. Map 任务越多优先级树越复杂 (深度越大), 所以出现误差的值就越大. 为了证明这个假设, 在实验中增加 Map 任务的数量而不增加输入数据量, 将 Map 任务的数据块从 128 MB 减小为 64 MB. 实验结果如图 11 所示, 当输入数据量为 5 GB, 作业数量为 1 个的时候, 误差值为 17%, 是实验中得到的最大值. 当优先级树的最大深度减小时, 误差也会降低.

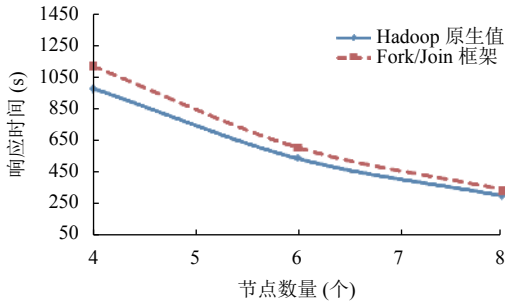


图 9 作业数量 4, 输入数据量 5 GB

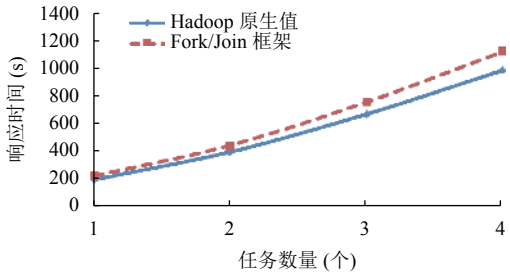


图 10 节点数量 4, 输入数据量 5 GB

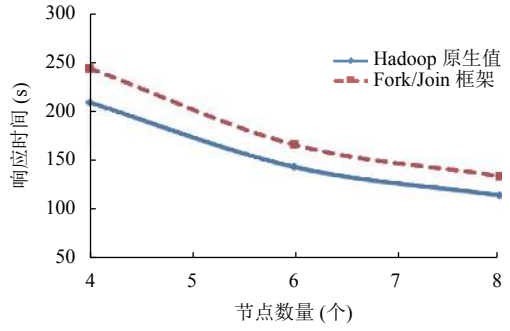


图 11 数据块 64 MB, 作业数量 1, 输入数据量 5 GB

基于 Fork/Join 框架的算法比 Hadoop1.x 原生方法提供了更加准确的结果, 可以进一步调整开销模型, 通过计算作业叠加因素的变化来提高准确性.

## 5 结论

本文解决了为 Hadoop2.x 创建 MapReduce 性能评估模型的问题. 该模型考虑了由于共享资源争用而导致的排队延迟, 以及在同一作业中进行协作的各任务之间由于优先级约束而导致的同步延迟 (Map 阶段和 Reduce 阶段). 该模型的构建方法通过在 Hadoop1.x 性能评估模型的基础上扩展得到, 用优先级二进制树表示作业的执行流程, 用封闭式排队网络捕获物理资源上的争用. 通过对 Hadoop2.x 的资源管理与任务调度方法进行分析, 考虑到 Hadoop2.x架构中资源分配改为动态方式, 作者创建了时间线构造算法, 并在此方法的基础上构建出优先级树.

在实验中从同时执行的不同数量作业的真实 Hadoop 设置值中获得测量值, 使用实验测量值对该模型进行验证: 使用标准数据块 (128 MB) 计算出作业响应时间的平均误差在 11% 和 13.5% 之间. 该模型可用于理论上估计作业响应时间, 所用成本要比实际中设置实验数据低得多. 该模型对于作业负载管理和资源分配规划中的决策也能够起到辅助作用.

## 参考文献

1 李元亨, 邹学玉. Hadoop 综述. 电脑知识与技术, 2018, 14(9): 8–9.
2 司雅楠. Hadoop2.0 平台概述. 科技与创新, 2019, (5): 65–66.
3 郭玉栋, 左金平. 基于 Hadoop 改进的云任务调度算法研究. 晋中学院学报, 2019, 36(3): 56–60. [doi: 10.3969/j.issn.1673-1808.2019.03.013]
4 马生俊, 陈旺虎, 郭宏乐, 等. Hadoop 集群中影响应用性能的因素分析. 小型微型计算机系统, 2018, 39(4): 719–724. [doi: 10.3969/j.issn.1000-1220.2018.04.018]
5 朱洁, 顾烨君, 柳飞, 等. Hadoop 负载树任务调度算法. 软件导刊, 2018, 17(12): 69–72, 76.
6 李耘书, 滕飞, 李天瑞. 基于微操作的 Hadoop 参数自动调优方法. 计算机应用, 2019, 39(6): 1589–1594. [doi: 10.11772/j.issn.1001-9081.2018122592]
7 王凌晖, 解云月, 周美华. Hadoop 分布式存储架构的性能分析. 现代电子技术, 2018, 41(18): 92–95.