文章编号:1671-4598(2016)09-0172-04 DOI:10.16526/j.cnki.11—4762/tp.2016.09.048 中图分类号:TP3 文献标识码:A

异构环境下基于双重预取的 Hadoop 调度算法

孙玉强，陆 勇，王文闻，李媛媛，顾玉宛
（常州大学 信息科学与工程学院，江苏 常州 213000）

摘要：Hadoop 处理海量数据时，无论是 Map 任务还是 Reduce 任务都需要耗费大量的时间传输数据，故提出一种基于双重预取的调度算法；该算法通过估算节点上任务执行的进度来预测 Map 任务的执行节点，然后通知节点提前预取所需的数据，并且在 Map 任务完成的数量达到预定值时，开始为 Reduce 任务预取部分数据；由于在异构的环境下集群中节点的性能各不相同，为此采取了改进的预测模型，以提高任务进度判断的准确性；实验证明，本算法在作业响应时间等方面优于现有的调度算法。

关键词：Hadoop；异构环境；调度算法；双重预取

Scheduling Algorithm Based on Double Prefetching in Heterogeneous Hadoop Clusters

Sun Yuqiang, Lu Yong, Wang Wenwen, Li Yuanyuan, Gu Yuwan
(School of Information Science & Engineering, Changzhou University, Changzhou 213000, China)

Abstract: When Hadoop processing huge amounts of data, both in the Map tasks and Reduce tasks requires a lot of time to transfer data. This paper presents a scheduling algorithm based on double prefetching, the algorithm predicts the node which will execute the Map task by estimating the progress of running tasks, so that the node can prefetch required data for Map tasks. Moreover, the system can also prefetch the data for Reduce tasks while Map tasks are running. Due to the performance of the cluster nodes in heterogeneous environment are not identical, the algorithm adopts an improved prediction model to improve the accuracy of the judgment of task progress. Experiments show that the algorithm is superior to the existing scheduling algorithm with less response time.

Keywords: Hadoop; heterogeneous; scheduling algotithm; double prefetching

0 引言

近年来，随着计算机的普及和信息技术的快速发展，互联网的使用也越来越广泛。与此同时，互联网上的数据也呈现出爆炸式的增长。例如，纽约证券交易所每天产生 1TB 的交易数据。FaceBook 存储着超过 1 亿张照片，约 1PB 存储容量。据预测，2015 年全球产生的数据量将达到近 10 ZB，而 2020 年全球产生的数据量将达到 40 ZB。我们已经生活在一个大数据的时代，越来越多的公司需要面对大数据的处理问题。

传统的的解决方案存在着存储量小、稳定性差、耗时过长等缺点。当前广泛使用的大数据处理模型是由 Google 公司设计的 MapReduce[1] 编程模型。MapReduce 作为并行分布式数据处理框架获得了极大的成功。MapReduce 把海量的数据计算划分为许多小的任务，并且让它们在不同的节点上并行执行。它隐藏了底层处理的细节，为开发分布式应用提供了简单的编程接口。由于其具有良好的可扩展性、容错性、可用性等特点，使得其成为近年来数据处理领域的研究热点。Hadoop[2] 平台作为 MapReduce 的开源实现，由于其良好的性能，已经被 Yahoo!、Amazon 等公司采用。

MapReduce 的处理过程主要分为两步。首先是执行 Map 任务，处理输入数据并生成中间结果，将其存储在本地节点。然后 Reduce 任务远程读取这些中间结果，经过运算产生最终的输出结果。任务的执行需要调度器将用户作业队列中的任务分配给相应的节点，调度算法的优劣对于系统的性能起着至关重要的作用。为此，大量的学者对 Hadoop 的调度算法做了相关的研究。

常见的 Hadoop 调度算法有 FIFO[3] 调度，HOD[4] 调度，公平调度[5] 等算法。陶永才等人提出基于动态负载均衡的 Hadoop 动态延迟调度机制[6]，金嘉晖等人提出基于数据中心负载分析的自适应延迟调度算法[7]，两种方法均提高了作业的响应时间，但都没有考虑节点的异构性。M Zaharia 等人针对异构环境提出推测任务剩余时间的 LATE[8] 算法，李丽英等人基于 LATE 算法提出数据局部性改进的调度算法[9]。但上述算法都缺乏对数据预取方面的考虑。本文提出异构环境下基于双重预取的 Hadoop 调度算法，以提高作业的执行效率。

1 MapReduce 框架分析及问题的提出

如图 1 所示，MapReduce 中任务的执行分为 Map 和 Reduce 两个阶段。在用户提交了一个作业后，输入数据被划分为多个数据块（默认为 64 M），每个数据块对应一个 Map 任务。Map 任务对输入进行处理生成＜key，value＞对作为中间结果。当所有的 Map 任务执行完成之后，Map 输出的中间结果交给 Reduce 任务执行。Reduce 任务可以分为三步。首先执行 Reduce 任务的节点读取中间结果，然后根据 key 进行排序，

收稿日期:2016-02-29； **修回日期**:2016-04-25。
基金项目:国家自然科学基金项目(11271057);江苏省普通高校研究生科研创新计划项目(SCZ1412800004)。
作者简介:孙玉强(1956-),男,博士,教授,主要从事并行计算、软件工程方向的研究。
通讯作者:顾玉宛,女,博士生,通讯联系人,主要从事并行计算和图像处理方向的研究。

最后调用用户编写的 Reduce 函数执行并输出最终结果。

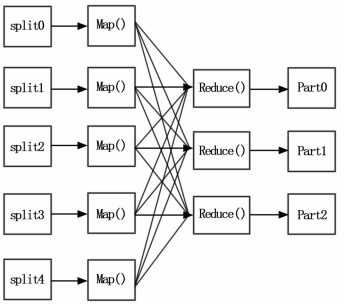

图 1　MapReduce 框架

Hadoop 集群中有一个 JobTracker 负责调度作业运行。当 JobTracker 收到客户端提交的作业后，就把它放在一个队列中。调度器为其创建相应的 Map 和 Reduce 任务。其余的节点作为 TaskTracker 负责具体任务的执行并且通过心跳信号（heartbeat）与 JobTracker 进行通信。当某个 TaskTracker 有空闲资源时，就会向 JobTracker 请求新的任务，此时调度器会给这个节点分配一个 Map 或者 Reduce 任务。

通过分析 MapReduce 的执行流程可以发现，Map 任务执行前如果没有取到相应的数据，会先远程读取所需要的数据。如果数据量非常大，那么 Map 任务的数量也较多，并且它们分布在集群中不同的节点执行，这将导致大量的数据传输开销。另外，Reduce 阶段要等到所有的 Map 任务执行结束后才开始，并且同样需要传输大量数据。这些都会耗费大量的时间。

针对以上问题，本文将采用改进的调度算法，通过双重数据预取的方式来减少任务执行时读取数据的时间，提升 Hadoop 的执行效率。

2　改进的算法

本文提出的改进算法从两方面进行数据的预取。

1）在节点分配 Map 任务之前，通过预测模型找出最快即将完成任务的节点，通知相应的节点进行下次 Map 任务所需数据的预取。

2）在所有 Map 任务完成之前，即将进行 Reduce 任务的节点对 Map 任务已经生成的中间数据进行预取。

2.1　Map 任务的数据预取

在 MapReduce 中，当一个节点被分配 Map 任务后，首先要获取任务所需要的输入数据。在最好的情况下，运行的任务和需要的数据在同一个节点上，则称其满足数据本地性，否则需要从远程节点进行读取。为了减少数据读取的时间，可采用数据预取的方式，使得当前任务的执行与下次任务数据的传输并行执行。

具体步骤如下：

1）TaskTracker 节点检查其正在执行的任务，推测出任务结束还需要的剩余时间，并通知 JobTracker 节点。

2）JobTracker 节点生成一个预调度节点队列 PreNodes，并且设定一个阈值 MapThreshold，把剩余时间低于阈值的节点加入队列。时间越短的在队列中的位置越靠前。

3）JobTracker 从任务队列中取出即将调度的任务，预先将其分配给队列中最前面的节点，并且通知其预取相应的数据。

4）接收到预取数据任务的节点开始预取下一次 Map 任务的数据。

首先，为了让节点推测出任务执行需要的剩余时间，就需要一个推测模型。Hadoop 根据任务的执行进度判断任务执行的快慢。但是在异构的环境下，每个节点的 CPU、内存以及 I/O 性能都不一样，因此上述方法并不适用于判断任务的剩余完成时间。目前有一种针对异构环境的 LATE 算法，其核心思想是通过执行进度判断任务的速率进而推算出任务的剩余完成时间，这正符合我们的目的。

LATE 算法的计算公式如下所示：

$$\text{ProgressRate} = \frac{\text{ProgressScore}}{T} \tag{1}$$

$$\text{TimeRemain} = \frac{1 - \text{ProgressScore}}{\text{ProgressRate}} \tag{2}$$

其中：ProgressScore 是任务执行的进度，量化为百分比。T 是任务已经运行的时间，TimeRemain 是任务完成需要的剩余时间。

ProgressScore 的值基于任务执行的阶段。在此处我们仅仅关注 Map 任务，Map 任务的执行被划分为两个子阶段：1）Map 函数执行，占 2/3；2）sort 和 partition 阶段，占 1/3。但是由于在异构环境下节点差异较大，任务执行的各阶段比例也会有所不同，所以采用固定的划分方式并不合理。为此，本算法在 TaskTracker 执行 Map 任务时，把各阶段的比例信息保存在磁盘中。下次执行任务前，首先读取应用的历史执行信息，动态确定各阶段的比例，使得每个节点都能得到更准确的进度值。ProgressScore 与运行时间 T 的比值则为当前任务的速率 ProgressRate，用任务剩余进度除以速率则可以估算出任务完成需要的时间。

为了实现预取数据的保存，TaskTracker 节点必须做相应的改进。鉴于目前节点的内存都比较大，可直接把数据预取到内存中，加快 Map 任务的执行速度。为此，把节点的内存划分为两部分。一部分存放当前任务的数据，另一部分用于存放下次 Map 任务的数据。当前任务结束后，其所使用的数据空间将用于下次任务预取数据的存储，而当前的预取数据将成为下次任务的输入数据。这两部分空间轮换使用，从而达到数据预取的目的。

如图 2 所示，当 JobTracker 节点预判到 map6 这个任务即将在节点 1 执行时，就会给节点 1 下达预取指令，于是节点 1 从节点 2 开始传输 map6 所需的输入数据。

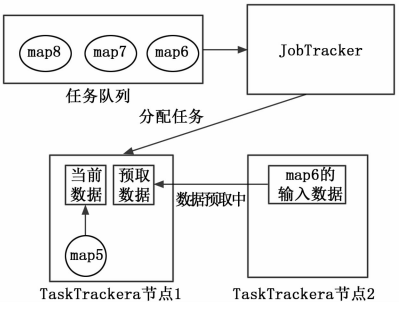

图 2　数据预取模型

该预取算法伪代码描述如下：

```
for each map in TaskTracker do
  TimeRemain = (1-ProgressScore)/ ProgressRate
endfor
sort(nodes)
if nodes. TimeRemain < MapThreshold then
queue. offer(nodes)
end if
for maps in job queue do
   nodes ← map //把 map 任务分配给 node 节点
if nodes 没有 map 任务的数据 then
  nodes begin data perfecting;
endif
end for
```

2.2 Reduce 任务的数据预取

Reduce 任务必须等到所有的 Map 任务执行完成之后才开始执行，第一步是从相应的节点将 Map 生成的中间结果进行拷贝，一样需要数据传输的开销。为此，同样对 Reduce 任务进行数据的预取。

Redecue 任务的数据预取与 Map 任务的预取相似，但不同的是由于 Reduce 任务的输入数据来源于 Map 任务的输出，所以其数据预取不能依靠当前节点 Reduce 任务的剩余时间。本算法采取的策略是，利用式（3）计算已完成任务的 Map 数量与 Map 任务的总数量之比。当比值达到一个设定的阈值 ReduceThreshold 之后，即开始为 Reduce 任务进行预取。

$$Rate_{\text{map}} = \frac{MapNums_{\text{finished}}}{MapNums_{\text{total}}} \tag{3}$$

其中：$MapNums_{\text{finished}}$ 为已完成的 Map 任务数，$MapNums_{\text{total}}$ 为 Map 任务总数量。

在为 Reduce 任务预取中间结果的时候，可能会出现多个节点同时从相同的输出结果预取数据，由此造成 I/O 资源的竞争，降低了预取的效率。为了解决这个问题，采用冲突检测的方式来为 Reduce 任务预取数据。当某个 Reduce 任务预取的数据所在节点已经有其它任务正在预取时，则先跳过这部分数据，转而预取其他节点上的所需数据。如果所有的节点都被其他任务占用，则等待一段时间再发起预取请求。等待时间设定如下：

$$Rate_{\text{data}} = \frac{Data_{\text{finished}}}{Data_{\text{total}}} \tag{4}$$

$$\text{WaitingTime} = \text{rate} \times \text{TIME} \tag{5}$$

式（4）中 $Data_{\text{finished}}$ 是已经预取的数据，$Data_{\text{total}}$ 是需要预取的数据总量，$Rate_{\text{data}}$ 则为任务的数据预取率。TIME 是一个时间常数，表示集群中传输一个数据块所用时间。WaitingTime 则是等待时间，与数据预取率成正比。数据预取率较高的，等待时间也较长，以保证各 Reduce 任务均衡预取。

由于随着时间的推移，Map 任务完成的数量越来越多，所以需要循环地探测执行 Map 任务的节点，获取更多地数据。该过程算法伪代码描述如下：

```
while Rate < ReduceThreadhold do
Rate = MapNumsfinished/ MapNumstotal
end while
while ! (all maps has finashed) do
for each node in NodesWithReduceTask do
if task needs more data do
prefectching data
if 预取冲突 then
WaitingTime = rate * TIME //等待
end if
end if
end for
end while
```

3 实验

本实验采用虚拟机的方式搭建异构环境，各 PC 机采用 100 M 的局域网互联。虚拟机使用 VMware workstation 10.0 安装的 CentOs 6.5 32 位系统。JDK 版本为 1.7.0 _ 45，Hadoop 版本为 0.20.1。具体集群配置如表 1 所示。

表 1 Hadoop 集群环境配置

机架	节点数	虚拟机数(个/台)	CPU
机架 1	1	2	Inter P III
机架 2	2	1	Inter P IV
机架 3	2	2	Inter dual－core

实验中选取的任务是 Sort 和 WordCount。因为这些任务涉及到大量数据的传输，有利于比较算法之间的差异。在 HDFS 中数据块设为默认的 64 M，且每个数据块保存 2 个副本。图 3（a）和图 3（b）分别是 Map 任务和 Reduce 任务的数据本地性比较。

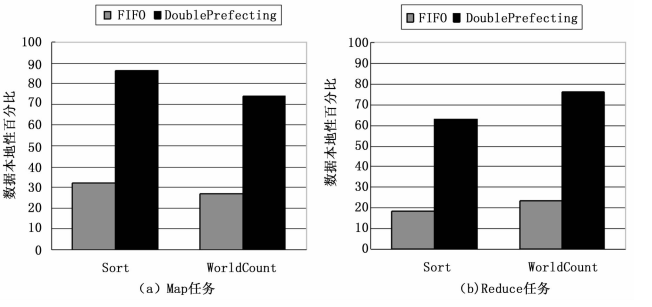

图 3 任务数据本地性对比

通过图 3 可以看出，相比于传统的无数据预取的 FIFO 算法，执行数据预取之后的 Map 任务和 Reduce 任务的数据本地性都得到极大提高。当然，对于作业执行效率的分析最重要的是响应时间。对上述任务设定不同的数据量多次执行后得到图 4 的响应时间图。从图中可以推算出，通过双重的数据预取后作业的响应时间提升了大约 18%。

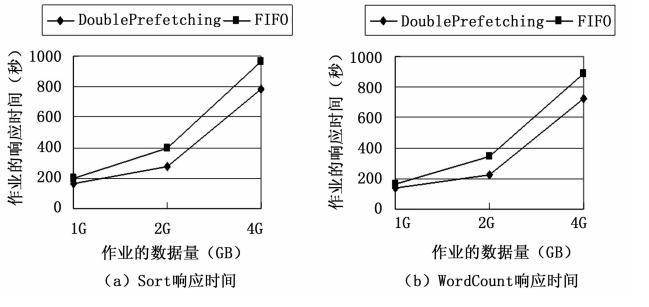

图 4 作业响应时间对比

4　总结

针对 Hadoop 作业执行时的数据本地性问题，本文提出基于双重预取的调度算法。通过预测任务的执行节点，为 Map 任务预取所需的数据。在 Map 任务完成了一定数量后，预先为 Reduce 任务拷贝 Map 已经生成的中间结果，解决了 Reduce 任务的远程调度问题。另外，算法充分考虑异构环境下节点性能的差异，采用改进的 LATE 算法以便更准确地判断任务的执行进度。实验证明，使用数据预取有效提高了作业的响应时间。

参考文献：

[1] Dean J，Ghemawat S. MapReduce：simplified data processing on large clusters [J]. Commun. ACM 51 (1) (2008) 107－113.

[2] White T. Hadoop：The Definitive Guide [Z]. O' Reilly Media，Inc.，2009.

[3] HADOOP－3759：Provide ability to run memory intensive jobs without affecting other running tasks on the nodes [EB/0L]. https：//issues. apache. org/jira/browse/HADOOP－3759.

[4] Apache. Hadoop On Demand [DB/OL]. http：//hadoop. apache. org/common/does/r0. 17. 2/hod. html，2008，20 (8).

[5] Zaharia M，Borthakur D，Sarma J S，et al. Job Scheduling for Multi－user MapReduce Clusters [R]. EECS－2009—55. April 2009.

[6] 陶永才，李文洁，石　磊，等. 基于负载均衡的 Hadoop 动态延迟调度机制 [J]. 小型微型计算机系统，2015，3 (3)：445－449.

[7] 金嘉晖，罗军舟，宋爱波，等. 基于数据中心负载分析的自适应延迟调度算法 [J]. 通信学报，2011，32 (7)：47－56.

[8] Zaharia M，Konwinski A，et al. Improving mapreduce performance in heterogeneous environments [A]. Proc of USENIX conference on Operating systems design and implementation [C]. Berkeley：USENIX Association，2008：29－42.

[9] 李丽英，唐　卓，李仁发. 基于 LATE 的 Hadoop 数据局部性改进调度算法 [J]. 计算机科学，2011 (11)：67－70.

(上接第 171 页)

```
<!-- 定义数据源 -->
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
    <property name="driverClassName">
    <value>oracle.jdbc.driver.OracleDriver</value>
    </property>
    <property name="url">
    <value>jdbc:oracle:thin:@localhost:1521:device</value>
    </property>
    <property name="username">
    <value>mydevice</value>
    </property>
    <property name="password">
    <value>m123456</value>
    </property>
</bean>
```

图 6　JavaBean 组件

4）Hibernate 技术：

Hibernate 是一个开源的对象/关系映射框架。它对 JDBC 进行了轻量级的对象封装。Hibernate 对象将 JavaBean 对象和数据库中表建立对应关系，使数据库中的表对应于 JAVA 中的对象。在对数据库中数据操作时，只需调用 JavaBean 对象而不用写 SQL 语句。Hibernate 体系结构如图 7 所示。

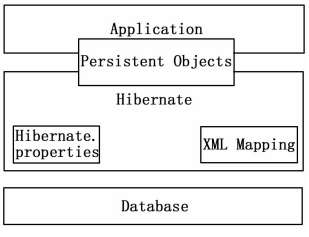

图 7　Hibernate 体系结构

图 8 是 Hibernate 中对数据库操作的 JavaBean 对象代码，实现了对数据库的增删改查操作。

```
<!-- 配置事务属性 -->
<property name="transactionAttributes">
 <props>
  <prop key="delete*">PROPAGATION_REQUIRED</prop>
  <prop key="update*">PROPAGATION_REQUIRED</prop>
  <prop key="create*">PROPAGATION_REQUIRED</prop>
  <prop key="get*">PROPAGATION_REQUIRED,readOnly</prop>
 </props>
</property>
```

图 8　数据库操作 JavaBean 对象

4　软件效果展示

研发的软件经过测试，能够很好地实现预期的功能。包括对数据库中数据的实时读取、浏览器显示以及故障判断等预定功能。系统软件的主要功能界面运行效果如下。

1）系统主界面：

程序启动后进入登录界面，输入指定用户名和密码进入主界面，可以看到主界面上 4 个主要功能模块。

2）设备信息显示：

通过点击主界面上设备管理按钮，可实现按厂家和设备类型对设备进行查询，也可查询到一定时间段内设备的实时状态。

3）故障诊断和查询：

软件设定每隔一段时间自动访问数据库中的状态信息表，提取新数据，通过分析比较诊断出设备是否故障，然后将故障信息显示到界面，同时该信息可导出至 EXCEL 表中。

5　结束语

设备管理软件系统旨在对应用于油井现场的数字化设备进行系统的管理。软件使得作业区和厂区级的管理人员可以获得数字化设备的基本信息及其运行状况信息，统计分析故障产生原因，查看维修情况，从而保证数字化设备的上线率、准确率、完好率，方便管理层能够及时全面地掌握实时的实物资产状况。所研发的系统软件经过测试，具有运行良好、性能稳定、功能齐全等特点，能够很好地满足油田现场对设备管理的要求。

参考文献：

[1] 徐进明，但正刚. JSP 网站开发技术，清华大学出版社，2007.

[2] 李代平. Oracle 9i 应用系统开发技术 [M]. 北京：冶金工业出版社，2004.

[3] 贾素玲，王　强. JSP 应用开发技术 [M]. 北京：清华大学出版社，2009.

[4] 刘振杰，何娟丽. 网页制作技术 [M]. 北京：人民邮电出版社，2009.

[5] 张　伟. Java 语言程序设计 [M]. 北京：电子工业出版社，2008.

[6] 沈应逵. Java Web 数据库系统应用开发与实例 [M]. 北京：人民邮电出版社，2004.

[7] Li D P. Application system development technology Oracle9I [M]. Metallurgical Industry Press，2004.