目前学历*	【 A 】 A. 硕士研究生　　B. 博士研究生　　C. 其他				
姓名*	郑健	学号*	2011110776	导师姓名*	徐六通
学院*	计算机学院	专业*	计算机科学与技术		
班级*	2011111304	研究方向*	下一代电信运营支撑系统		
固定电话		E-Mail*	zhengjian8972@qq.com		
手机*	18810300681				
作者类别*（填选项）	【A】 A. 第一作者参与论坛 B. 老师作为第一作者，选手以第二作者参与论坛 C. 其他情况				
论文方向*（填选项）	【B 】A. 通信 B. 计算机 C. 经管 D. 自动化 E. 理学 F. 人文 G. 教育 H.电子				
论文题目*	基于资源与作业特征的 Hadoop 参数优化方法				

关键字*（4 到 5 个中文词或英文单词）

MapReduce，资源，作业特征，参数优化

论文摘要*

大规模数据分析如今变得越来越热门，其已应用领域已广泛涉及到商务数据分析、科学数据处理和仿真计算等领域。Hadoop 并行计算框架使用了 MapReduce 编程模型通过控制多台机器并行处理大规模数据，目前已成为处理这些数据密集型任务的最佳选择之一。但是，尽管 MapReduce 计算框架能提高计算效率，但用户还是会经常遇到性能问题，由于他们不知道如何针对资源和作业特征调整 Hadoop 中影响执行过程和资源分配的配置参数。本文首先详细分析了 MapReduce 中的数据处理原理，进而研究了相关参数对资源消耗的影响，并提出了一种基于集群资源配置和作业特征的 Hadoop 参数优化的方法，从优化资源利用率的角度提升作业性能。实验证明，这种方法在实际场景中能大幅提升 MapReduce 作业执行效率。

备注	

【注】提交文论前请务必把此表填写完整，在论文正文中，不允许出现任何与作者个人信息相关的内容。作者在提交论文时，需要将此表和论文正文一起提交。文档命名格式为：**论文_所在学院_作者姓名_学号_论文方向_导师。**（注意格式中第一个字段“论文”为固定统一字段，并非是论文题目，选手不必编辑和修改）

另：提交论文时，请作者同提交参赛报名表，及作者信息表(excel)。
参赛报名表命名格式为：**报名表_所在学院_作者姓名_学号_论文方向_导师**
作者电子信息表命名格式为：**信息表_所在学院_作者姓名_学号_论文方向_导师**
三份文档打包压缩后，命名为：**所在学院_论文方向_作者姓名_学号_导师**
然后，统一发送至组委会邮箱：bupt_xueshuluntan@bupt.edu.cn

基于资源与作业特征的 Hadoop 参数优化方法

摘要：大规模数据分析如今变得越来越热门，其已应用领域已广泛涉及到商务数据分析、科学数据处理和仿真计算等领域。 Hadoop 并行计算框架使用了 MapReduce 编程模型通过控制多台机器并行处理大规模数据，目前已成为处理这些数据密集型任务的最佳选择之一。但是，尽管 MapReduce 计算框架能提高计算效率，但用户还是会经常遇到性能问题，由于他们不知道如何针对资源和作业特征调整 Hadoop 中影响执行过程和资源分配的配置参数。本文首先详细分析了 MapReduce 中的数据处理原理，进而研究了相关参数对资源消耗的影响，并提出了一种基于集群资源配置和作业特征的 Hadoop 参数优化的方法，从优化资源利用率的角度提升作业性能。实验证明，这种方法在实际场景中能大幅提升 MapReduce 作业执行效率。

关键字：MapReduce，资源，作业特征，参数优化

一. 引言

如今越来越多的 Web 应用服务着亿万互联网用户，并产生了 PB 级的数据量。相对廉价的存储能力导致了滞留数据的极大增长，从这些数据集中抽取信息和情报——也就是通常所说的数据分析，是一种基于这些巨大数据集的数据密集型应用。数据分析被用于许多场景，如 web 数据挖掘（web 索引和检索）、商务数据挖掘、科学计算和仿真（自然语言处理、图挖掘）等[5]- [9]。

分析这些大规模数据要求有高可靠的解决方案。MapReduce[11]是一种流行的方法，可以进行数据分析通过并行数据处理，将大规模数据分割到多个商用机器上，可以方便地将集群扩展至满足数据处理需求的规模。Hadoop[4]是一个开放源码的基于 Java 语言的云计算框架，该框架使用 MapReduce 的编程模型对海量数据进行跨集群的分布式处理，它的思想来源于 Google Labs 开发的 MapReduce 和 Global File System (GFS)技术，具有高效、可靠和可伸缩的优点。经过多年发展，Hadoop 已经成为大规模数据处理方面事实上的行业标准框架，在云计算领域得到广泛应用，一系列企业级 Hadoop 应用已经在 web 索引、数据挖掘、日志分析、科学仿真、基因信息计算等领域发挥重要作用。

尽管 Hadoop 在云计算中有诸多优点，用户还是会经常遇到性能问题，由于他们不知道如何如何针对资源配置和作业特征调整 Hadoop 参数，以更有效的方式利用计算资源。集群中的计算资源包括 CPU、内存、磁盘 IO 和网络等，大多数情况下，这些资源是被闲置的。对用户来说，参数配置的目标是充分利用计算资源以获得最大的性能。Hadoop 为用户提供了 190 多个可配置参数，其中部分参数对 Job 对系统资源的利用率有重大影响，表 1 提给出了 Hadoop 中几个重要参数的样例。

表 1 HADOOP 中重要参数样例

参数名	默认值	参数说明
io.dfs.block.size	67108864	数据块大小
mapred.child.java.opts	-Xmx200m	Task 内存大小
io.sort.mb	100M	并发拷贝连接数

Hadoop 在安装时为所有参数设置了默认值，但是参数的默认值无法完全适应多变的集群环境和作业需求，通常情况下作业性能无法达到最优。如何通过参数调优提高集群资源利用率、提升作业效率，已成为业内关注的热点问题之一。

辛大欣[3]等人对 Hadoop 参数配置给出了一些经验性建议，这要求用户在不熟悉参数作用机制的情况下根据经验选择参数。然而，在当前集群下的最优配置并不一定适合另一个集群；而运行作业特征的改变也会使得最优参数变得不再合适。Karthik[1]等使用 RS Sizer 监控 Hadoop Job 在不同 map 数和 reduce 个数情况下的资源占用情况，并以此为标准，建立 signature 数据库，未来可根据这个数据库，得出在特定情况下的 Hadoop 参数配置。这近似于一种黑盒方法，仅针对 Map 和 Reduce 个数构建模型，没有提及 Hadoop 中的其余参数，建模过程依赖于大量的测试，在实际生产中不便使用。Shivnath[2]提出了使用动态 Dynamic Profiling 方法，在 Job 运行之前获取抽样数据，通过预先运行 job，获取 Job 占用的资源情况，动态配置 Hadoop 参数。但该方法基于运行数据可均匀抽样的假设，无法完全模拟 Job 运行过程中的资源消耗。Shivnath 还提出了基于竞争的参数配置方法：对同一个 job 开启多个参数不同的 Task，若发现一个 task 运行速度快于其他，则将其他 Task 杀死，剩余的这套配置就是该 job 的最优配置，该方法减少了对消耗模型的需要，但这种仅基于 Job 的初始部分的性能来预测整个 Job 的性能，当 Job 存在多轮 MapReduce 时，初始部分的最优参数有可能不适合剩余 Job，参数配置无法达到最优。

本文认为基于经验的参数配置不够自动化，而简单的自动化参数配置不够精确。后文的主要内容为：首先介绍了 Hadoop 中的 MapReduce 计算框架，详细分析了 MapReduce 中的数据处理机制。然后分析关键参数对资源消耗的影响，并基于 Mapreduce 原理与作业特征提出一套提升集群资源利用率的 Hadoop 参数配置方法，从优化资源利用率的角度提升作业性能。通过实际生产中的场景证明，本文提出的 Hadoop 参数配置方法可使集群资源得到充分利用，并能明显提升作业性能。

二. MAPREDUCE 技术概述

虽然 MapReduce 的编程思想已在十几年前就广为人知，但将 MapReduce 用于大规模数据的并行处理还只是在 2004 年才由 Google 的 Dean 提出[10]。最流行的开源 MapReduce 实现 Hadoop，是由 Yahoo! 为处理 web 索引等数据实现的，它同时实现了基于 GFS 的分布式文件系统 HDFS。在捐献给 Apache 社区后，Hadoop 成为当前大中型企业处理 PB 级数据的事实上的标准工具。本章将详细介绍 Hadoop 中的 MapReduce 计算框架，并分析 MapReduce 中的数据处理原理及相关参数对资源消耗的影响。

2.1 MapReduce 技术概述

MapReduce 是一种计算模型，用于大规模数据集的并行计算。MapReduce 计算模型的核心概念是 Map（映射）和 Reduce（化简）。用户定义 Map 函数将所有输入键值对<inKey,inValue>映射成一系列中间键值对<intermediateKey,intermediateValue>，再由 reduce 收集 intermediateKey 和相应的 intermediateValue 列表，保证同一个 intermediateKey 的所有 intermediateValue 被分到一个组里，也就是获取<intermediateKey, list of intermediateValue>，然后 reduce 对其进行化简，得到<outKey,outValue>。

表 2 MAPREDUCE 程序实例

```
map(String inKey, String inValue):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, “1”);

reduce(intermediateKey, List of intermediateValue):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v inintermediateValueList:
    result += ParseInt(v);
  Emit(AsString(result));
```

假设需要统计大量文档中每个单词出现的次数，用户需要使用类似表 2 中的代码，Map 函数统计每个单词在单个文档中出现的次数，每出现一次，记录数加 1。Reduce 将同一个词的统计值相加，最终得出单词出现的总次数。

2.2 Hadoop MapReduce 技术框架

Hadoop 中的 MapReduce 过程是一系列混合的并行处理过程。Map 阶段在 reduce 阶段之前，但大部分时间内，Map 和 Reduce 是并行处理的。提交作业时，由主节点的 JobTracker 接受作业并实例化作业，将作业放入调度队列中等待调度；执行作业时，由子节点上的 TaskTracker 创建和执行多个 Map 和 Reduce 任务，每启动一个任务，节点的 TaskTracker 都会开启一个新的 JVM 进程，Map 函数或 Reduce 函数会在该进程中被重复调用。一个计算节点只能容纳有限的 map 任务和 reduce 任务，用户可根据 mapred.tasktracker.{map/reduce}.tasks.maximum 调整节点能够并行执行的 Map 和 Reduce 任务数量。图 1 简要展示了 Hadoop 中 MapReduce 的过程。

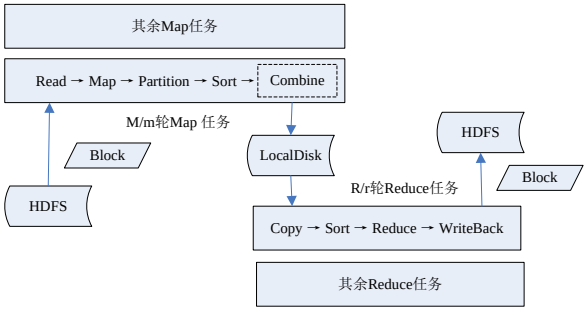

图 1. Hadoop 中 MapReduce 的过程

Hadoop 会为每个 Map 分配一定的内存大小（默认 200M，由 mapred.child.java.opts 设置），Map 会读入一个数据块（默认 64M，由 io.dfs.block.size 确定），因此如果有 M 个数据块，那么总共就会将 M 个 Map 任务加入调度队列，当 M 大于最大并行 Map 数时，会启动 M/m 轮 Map 任务。接着由用户定义的映射函数处理输入，并输出一系列中间<key,value>对。

MapReduce 框架需要把 Map 任务输出的数据分发到 Reduce 任务中，由后者进一步处理。整个过程包括数据划分（Partition）、排序（Sort）、写磁盘、网络传输、合并（Merge）等步骤，合称为 Shuffle，如图 2 所示。Shuffle 涉及 Map 任务和 Reduce 任务，实现了数据分区、排序、分发等功能，是 MapRedceu 框架中的重要部分，其中用到的参数对 MapRedcue 作业性能有重要影响。

(1) Map 端的 Shuffle 过程：

Map 的输出结果会暂时保存在内存的缓冲区（默认大小为 100M，由 io.sort.mb 控制）中，在写入缓冲区前，会根据 Reduce 任务总数 R，对 Map 的 Key 值进行 Hash，得到中间结果的划分值（Partition），之后会根据划分值将中间结果分配给不同 Reduce。这样的目的是保证相同 Key 值的数据能够汇总到同一个 Reduce 中处理；

缓冲区存放中间结果数据以及这些数据的位置索引（默认情况下索引区占缓冲区的 5%，两者比例由 io.sort.record.percent 控制）。在往缓冲区写入数据时，会使用堆排序根据记录的 Key 值与 Partition 值对索引进行排序，当缓冲区内的数据或者索引条数达到最大溢出比时（默认为 80%，由 io.sort.spill.percent 控制），会在本地文件系统中的缓存目录（由 hadoop.tmp.dir 设置）下创建一个临时文件，将该缓冲区中的数据按照排序的索引顺序写入文件，该文件被称为 Spill 文件。

当 Map 任务的输出数据超过一个缓冲区容量时，Map 端的输出会被写出到多个 Spill 文件中，当 Map 处理完所有数据后，会对将这些 Spill 文件合并，并将合并结果写回本地磁盘。值得注意的是，如果只有一个 spill 文件，将跳过合并（Merge）阶段，直接结束 Map 任务。

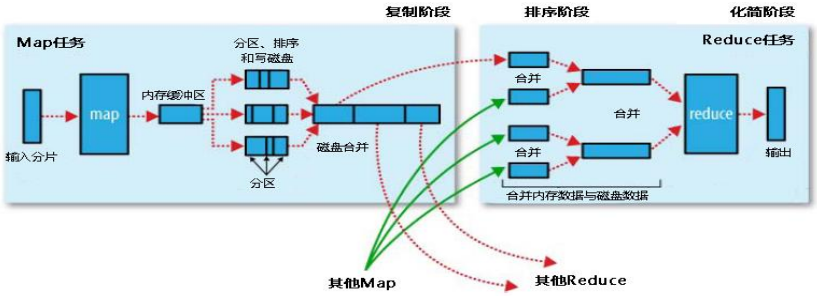

图 2. MapReduce 的 Shuffle 过程

(2) Reduce 端的 Shuffle 过程：

Reduce 任务会在第一轮 Map 任务执行完毕后开始拷贝 Map 端的中间结果数据，每个 Reduce 默认情况下会开启 5 个（由 mapred.reduce.parellel.copies 设置）拷贝线程。

Reduce 接收到不同 Map 任务传来的数据，并且每个 Map 传来的数据都是有序的。接收到的数据不会立刻写入磁盘，而是会先缓存在内存中，默认会使用 reduce 内存的 70% 用于缓存数据（由 mapred.job.shuffle.input.buffer.percent 设定）。与 Map 端类似，Reduce 的缓存区也是达到一定阈值就往磁盘上写。默认情况下，这里有两种情况：(1) 当拷贝的数据大小达到缓存区的 66%时，启动线程对内存中的数据根据 Key 值进行归并排序，并写出到本地磁盘（由 mapred.job.shuffle.merge.percent 设置）；(2) 当从 Map 端取到的分片（Segment）数达到 1000 时（由 mapred.inmem.merge.threshold 设置），也进行归并排序，并写出到磁盘。

如果数据量超过一个缓存区的大小，Hadoop 会使用归并排序将他们合并成为一个有序的大分片。当内存中剩余的 Map 端输出小于等于 0 时（由参数 mapred.job.reduce.input.buffer.percent 设定），Reduce 阶段才能够真正开始。

最后，Reduce 的执行阶段只需要顺序读取这分片文件，就能将相同 Key 值的记录归并到一起处理，通过用户定义的化简函数，得到最终结果并输出。

三. 基于资源与作业特征的参数配置方法

Hadoop 在安装时就在配置文件中为所有参数设置了默认值，这些默认值是根据典型的集群配置和应用设置的。然而，在实际应用场景中，不同的应用对资源有不同的需求，而参数值也要根据集群资源总量来设置。本章我们将通过一些实验结果，基于 MapReduce 运行机制，说明 Hadoop 关键参数对集群资源利用率的影响，并提出这些参数的配置方法。本章的实验使用 4 个节点的 Hadoop 集群，其中有一个子节点同时担任主节点，集群拓扑结构如图 3 所示。每个节点使用两个 Intel(R) Xeon(R) E5-2620@ 2.0GHz CPU，共 24 个处理进程，48G 内存，以及 11 块 7.2krpm 的 NL-SAS 盘，硬盘空间 11*3T，运行 64 位的 CentOS 6.3 Linux 操作系统。使用 Hadoop 自带的 Sort 作为实验场景，数据为 160G 的文本数据，使用 Nmon 工具[12]监控集群资源。

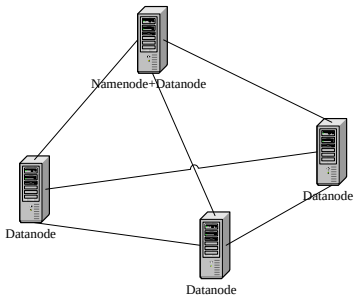

图 4. 实验集群拓扑结构

为了方便说明，需定义 Hadoop 参数变量、集群资源配置变量和 Job 特征变量。表 3 定义了上文提到的影响 MapReduce 的 Hadoop 关键参数，并根据其影响范围将参数分为 IO、Map/Reduce 和 Shuffle，本章分类介绍这些参数的调优方法。表 4 定义了节点个数、CPU 进程数、内存大小等集群资源配置变量。表 5 定义了与作业相关的特征变量。

3.1 IO 相关参数

与 IO 相关的参数主要有 pTmpDir 与 pDataDir，分别用于设置 HDFS 数据存放位置与中间结果目录。当 Map 任务或 Reduce 任务从 HDFS 或本次目录读取数据时，都会产生大量的磁盘 IO 操作。特别是在 Reduce 端从 Map 端拷贝中间结果过程中，Reduce 端将从所有 Map 中复制属于本端的中间结果，会有大量数据需要写入中间结果目录，并在 Merge 为有序分片（Segment）的过程中反复读写，容易产生 CPU 等待 IO 的情况。如图 4 所示，实验集群的 pTmpDir 仅挂载了一个磁盘目录时，Reduce 阶段 CPU 出现了大量等待，在这段过程中，CPU 实际利率用不到 30%。

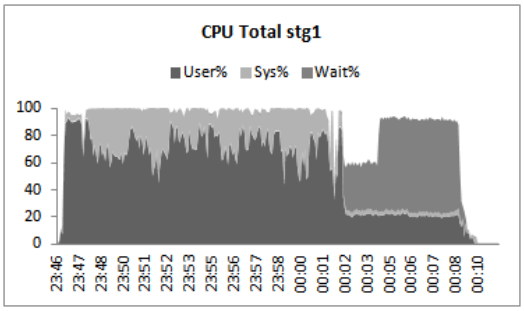

图 4. pTmpDir 挂载磁盘不足，导致 Reduce 阶段出现 CPU 等待 IO 的情况

表 3 HADOOP 参数变量

变量名	参数名	默认值	影响范围
pTmpDir	hadoop.tmp.dir		IO
pDataDir	dfs.data.dir		
pBlockSize	io.dfs.block.size	67108864	Map/Reduce、Shuffle
pTaskJvm	mapred.child.java.opts	-Xmx200m	
pMaxMapsPerNode	mapred.tasktracker.map.tasks.maximum	2	Map/Reduce
pMaxReducePerNode	mapred.tasktracker.reduce.tasks.maximum	2	
pNumMappers	mapred.map.tasks		
pNumReducers	mapred.reduce.tasks		
pSortMB	io.sort.mb	100	Shuffle
pSpillPercent	io.sort.spill.percent	0.8	
pRecPercent	io.sort.record.percent	0.05	
pSortFactor	io.sort.factor	10	
pPareCopies	mapred.reduce.parellel.copies	5	
pShuffleBufPercent	mapred.job.shuffle.input.buffer.percent	0.7	
pShuffleMerPercent	mapred.job.shuffle.merge.percent	0.66	
pMergeThres	mapred.inmem.merge.threshold	1000	
pReduceBufPercent	mapred.job.reduce.input.buffer.percent	0	

表 4 集群资源配置变量

变量名	参数名
sNumNodes	集群节点数
sCpuProcessers	节点 CPU 进程数
sMemory	节点总内存大小
sUsedMemory	操作系统占用的内存

表 5 作业特征变量

变量名	参数名
jNumBlocks	输入数据块数量
jInputMapPairs	输入记录条数
jInputWidth	输入记录平均长度
jMapPairsSel	记录行数转换率
jOuputMapPairs	输出中间结果条数
jMapWidthSel	记录长度转换率
jOutputWidth	输出中间结果长度

为减少 IO 等待的情况，应该为这两个参数设置尽量多的磁盘目录，这些目录应分散在不同的物理磁盘上。Map 端读取数据时，由于数据分散在多个磁盘目录下，会减少单磁盘的 IO 负载；在 Reduce 端读写数据时，将以轮询的方式选择一个中间结果目录写入数据，可提高 IO 效率。实验显示，挂载多个磁盘目录时，CPU 利用率和作业性能得到显著提升，Reduce 阶段的执行时间大大减少，如图 5 所示。

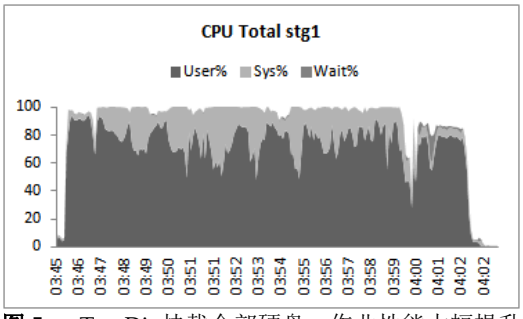

图 5. pTmpDir 挂载全部硬盘，作业性能大幅提升

3.2 Map 与 Reduce 相关参数

执行作业时，由子节点上的 TaskTracker 创建和执行 Map 和 Reduce 任务，这些 Map 和 Reduce 任务将平分子节点上剩余的 CPU 进程资源与内存资源。影响这些资源分配的参数主要有 pMaxMapsPerNode、pMaxReducePerNode、pTaskJvm、pBlockSize。

(1) CPU 进程资源的分配

Map 与 Reduce 的主要计算任务由 CPU 完成，通常情况下，CPU 的利用率决定了任务的执行效率。由于 Map 任务与 Reduce 任务的主要计算过程是先后进行的，在这两个阶段，除去为维护操作系统中其他程序留下的 CPU 进程，Map 与 Reduce 任务都可以独占剩余的 CPU 进程资源，因此为了充分利用 CPU 进程资源，将 pMaxMapsPerNode 和 pMaxReducePerNode 设置为：

$$pMaxMapsPerNode = sCpuProcessers - 1 \quad (1)$$

$$pMaxReducePerNode = pMaxMapsPerNode \quad (2)$$

实验中先后将 pMax{Maps/Reduce}PerNode 设为 12 与 23（实验场景中，sCpuProcessers-1=23），并监控作业运行过程中的 CPU 状态。对比图 6 与图 7 可见，当 pMaxMapsPerNode 和 pMaxReducePerNode 设置为 12 时，CPU 的利用率仅维持在 80%左右；当 pMaxMapsPerNode 和 pMaxReducePerNode 设置为 23 时，CPU 计算资源得到充分利用。

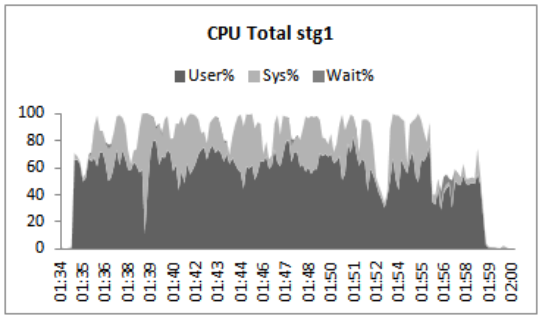

图 6. 最大任务数设置太少，无法充分利用 CPU

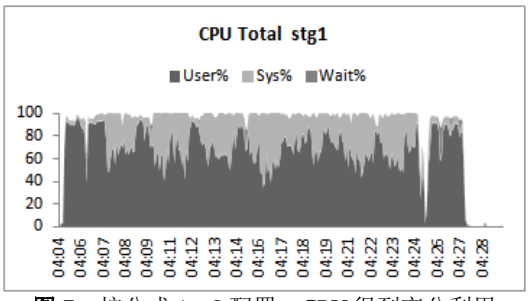

图 7. 按公式 1、2 配置，CPU 得到充分利用

(2) 内存资源分配

内存也是 Map 与 Reduce 任务过程中的重要资源。TaskTracker 在启动 Map 和 Reduce 任务时，会给每个任务分配 pTaskJvm 大小的内存空间，之后的 Map 和 Reduce 计算、Shuffle 过程都将在这个内存空间内执行。由于在第一轮 Map 执行完毕后，Hadoop 就会启动 Reduce 开始拷贝 Map 端输出的中间结果，因此大部分时间内，Map 和 Reduce 进程是并行执行的。为确保内存得到充分利用、又不造成频繁内存换页，它们需要平分操作系统的剩余内存。因此将 pTaskJvm 设为：

$$\text{pTaskJvm}=\frac{(\text{sMemory}-\text{sUsedMemory})}{(\text{pMaxMapsPerNode}+\text{pMaxReducesPerNode})} \quad (3)$$

本文使用 nmon 工具监控内存剩余量，如图 8 所示，在 Hadoop 环境中 sUsedMemory 大约为 5G 左右，内存剩余量大约为 43G，根据公式(3)，在测试集群中设置 pTaskJvm 为 $43/(23+23)\approx 950M$ 。

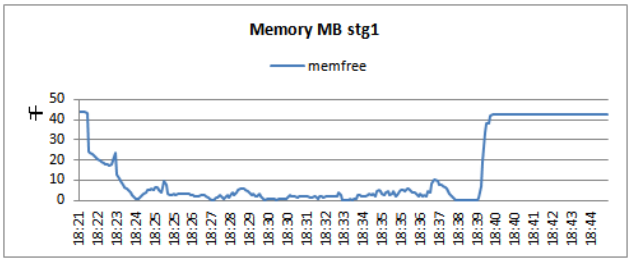

图 8. 使用资源监控工具查看运行时的内存剩余量

(3) Map/Reduce 总数设置

pNumMappers 默认情况下是由 jNumBlocks 确定的，对输入数据的每个块，TaskTracker 都会启动一个 Map 任务。对于 TotalData 大小的输入数据，pNumMappers 与 pBlockSize 的关系如下：

$$\text{pNumMappers=jNumBlocks=TotalData/pBlockSize} \quad (4)$$

Reduce 任务将在第一轮 Map 任务执行完毕后开始拷贝 Map 端的中间结果，拷贝过程是与 Map 任务并行执行的。为了避免多轮 Reduce 拷贝，需将 pNumReducers 设置为：

$$\text{pNumReducers=pMaxReducePerNode} \quad (5)$$

3.3 Shuffle 相关参数

Shuffle 是 MapReduce 框架的重要部分，其中涉及的到的参数对 MapReduce 作业性能有重要影响。在 Map 任务中，以一个数据块为输入，需处理的数据条数为

$$\text{jInputMapPairs=pBlockSize/jInputWidth} \quad (6)$$

本文定义记录行数转换率（jOuputMapPairs ）表示 Map 端输出记录条数与输入记录条数的比值；定义记录长度转换率（jMapWidthSel）表示输出记录长度与输入记录长度的比值：

$$\text{jOuputMapPairs=jInputMapPairs*jMapPairsSel} \quad (7)$$

$$\text{jOutputWidth=jInputWidth*jMapWidthSel} \quad (8)$$

对于 Hadoop 的示例作业 Sort 来说，jMapPairsSel 和 jMapWidthSel 均为 1。

根据不同 Job 的特征，将得到大小约为 jOuputMapPairs * jOutputMapWidth 的中间结果输出，存放于 Map 端的内存缓冲区 pSortMB（单位是 M）中。内存缓冲区被分为两个部分：存储 Key-Value 对的序列化数据区与存储索引的记录区，任意一个区域达到溢出比，都会将缓冲区内的数据按索引排序输出到磁盘上。序列化数据区能容纳的最多纪录条数为：

$$maxSerPairs=\frac{pSortMB*2^{20}*(1-pRecPercent)*pSpillPercent}{jOutputWidth} \quad (9)$$

记录区能容纳的最多纪录条数为：

$$maxRecPairs=\frac{pSortMB*2^{20}*pRecPercent*pSpillPercent}{16} \quad (10)$$

因此，每次溢出时 spill 文件内包含的记录条数为：

$$\text{spillBufferPairs = Min\{maxSerPairs,maxRecPairs,jOuputMapPairs\}} \quad (11)$$

最终得到的 Spill 文件个数为：

$$\text{numSpills = outMapPairs/spillBufferPairs} \quad (12)$$

理想情况下，调整 pBlockSize、pSortMB、pRecPercent 和 pSpillPercent，使 numSpills =1，将跳过 Map 过程中的 Merge 阶段，大大节省运行时间。

在实际应用过程中，可采用数据抽样的方式，在作业正真运行前预先计算 jMapPairsSel 和 jMapWidthSel，并作为作业特征参数保存。当作业正式运行时，可以直接利用这两个参数得出当前环境下的最优 Hadoop 配置，而无需再次进行计算。

若无法通过调整参数使 numSpills=1，那么在 Map 端和 Reduce 端进行 Merge 时，会由 pSortFactor 决定同时合并多少分片。当发现 Reduce 在 Shuffle 阶段的 IO 等待非常高时，就有可能通过调大这个参数来加大以此 merge 时的并发吞吐，优化 Reduce 效率。

在 Reduce 端，Redcue 任务会在第一轮 Map 执行完毕后开启 pPareCopies 个拷贝线程，对于 Map 数量较多的情况下，可适当调高并发拷贝数。Reduce 从 Map 端拷贝分片后，需要将所有分片合并为一个有序的大分片，这个过程将进行大量的磁盘 IO，通常情况下磁盘读写性能将成为瓶颈，因此对与内存相关的参数（pShuffleBufPercent、pShuffleMerPercent、pMergeThres 和 pReduceBufPercent）进行调整，效果并不明显，采用默认值即可。

四. 实验

本章将使用实际生产场景对上一章提出的参数调优方法进行验证。本章的实验使用 10 个节点的 Hadoop 集群，其中包括 9 个子节点和 1 个主节点。每个节点使用两个 Intel(R) Xeon(R) E5530 @ 2.40GHz CPU，共 16 个处理进程，48G 内存，以及 3 块 SAS

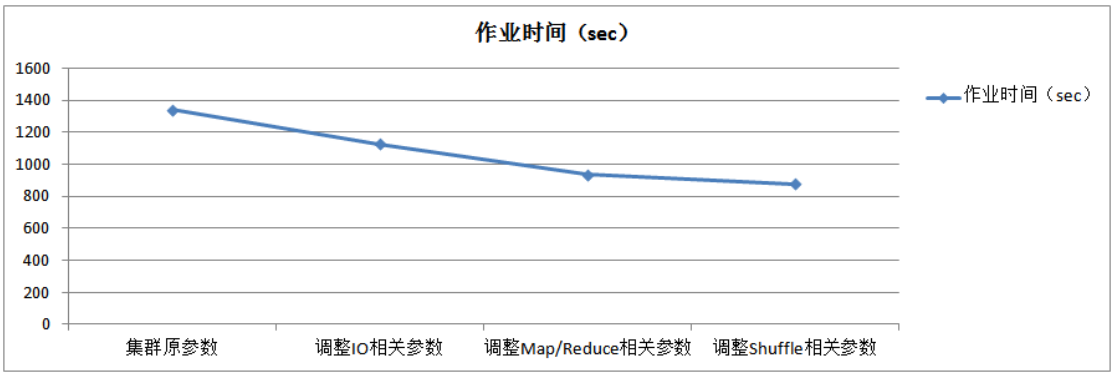

图 9. 使用资源监控工具查看运行时的内存剩余量

盘，硬盘空间为 3*1T，运行 64 位的 Red Hat Enterprise 6.2 操作系统。选择电信日志处理场景中常见的多表关联汇总作业，数据量为 320G，在原有参数下，作业运行时间为 1343 秒。根据上一章的方法调整参数后，作业性能得到大幅提升，参数改进与作业时间如表 6 所示，作业时间对比如图 9 所示。

表 6 HADOOP 参数改进步骤与性能提升

参数改进	作业时间 (秒)	加速系数
调整 IO 相关参数，挂载更多磁盘目录	1125	1.19
根据 CPU 与内存资源调整 Map/Reduce 相关参数	936	1.2
调整 Shuffle 相关参数，跳过 Merge 阶段	878	1.1

挂载更多磁盘目录：原有参数配置中，将 pTmpDir 挂载在根目录下；在新的配置中，在每个目录下，将 pTmpDir 分散挂载在 3 块硬盘上，减小了 Shuffle 阶段的 IO 负载，使作业性能提升为原先的 1.19 倍。根据 CPU 与内存资源合理调整 Map/Reduce 相关参数后，剩余的 CPU 和内存资源被并行的 Map/Reduce 任务平分，计算资源得到充分利用，性能在上一步基础上又提升 1.2 倍。通过调整 Shuffle 参数，使得 Map 端只输出一个 Spill 文件，从而跳过了 Merge 阶段，在原有基础上又提速了 1.1 倍。上面描述的所有改进累积起来使加速系数达到 1.5，作业性能得到较大提升。

五. 总结

目前，在实际生产中经常会由于不合理的 Hadoop 参数配置而照成资源浪费和性能瓶颈。本文基于 MapReducede 运行机制，对 MapReduce 中的数据处理原理及相关参数对资源消耗的影响进行了研究。根据参数作用范围，将参数分为影响 IO 的参数、Map/Reduce 相关参数与 Shuffle 相关参数，分别研究这些参数的在 MapReduce 过程中的作用机制与对集群资源占用的影响。基于 Mapreduce 原理与作业特征提出一套提升集群资源利用率的 Hadoop 参数配置优化方法，通过实际生产中的日志表关联汇总场景证明，本文提出的 Hadoop 参数优化方法可明显提升作业性能。

但是本文提出的参数调优是在理想条件下进行的，只对集群单独运行一个作业的情况给出了配置建议，而实际的集群通常是繁忙的，会出现多个 MapReduce 任务、甚至 MapReduce 与其他程序竞争 CPU、内存、磁盘和网络资源的情况，实际调优效果可能不如理想中明显。因此，具体分析实际集群的资源竞争情况，建立多程序资源竞争情况下的调优准则是下一步的研究点。

参考文献

[1] Kambatla Karthik, Pathak Abhinav, Pucha Himabindu, “Towards optimizing hadoop provisioning in the cloud,” in *Proceedings of the 2009 conference on Hot topics in cloud computing*, pp. 118-123, 2009.

[2] Shivnath Babu, “Towards automatic optimization of MapReduce programs,” in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010, paper, 137-142.

[3] 辛大欣, 刘飞. Hadoop 集群性能优化技术研究[J] 电脑知识与技术，2011，vol. 07(22)：5484-5486

[4] (2008) Apache Hadoop website. [Online]. Available :http://hadoop.apache.org/

[5] Le Yu, Jian Zheng, Wei Chong Shen, Bin Wu, Bai Wang, Long Qian, and Bo Ren Zhang, “BC-PDM: data mining, social network analysis and text mining system based on cloud computing” in In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012,paper, p. 1496-1499.

[6] Panda, B., Herbach, J. S., Basu, S., & Bayardo, R. J. “Planet: massively parallel learning of tree ensembles with mapreduce.” *Proceedings of the VLDB Endowment*, vol.2, pp, 1426-1437, 2009.

[7] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt,S. Madden, and M . S tonebraker , “A comparison of approachesto large-scale data analysis,” in *Proc. ACM SIGMOD Conference*, 2009.

[8] A. S . Das, M . Datar, A. Garg, and S . R ajaram, “Google news personalization: scalable online collaborative filtering,” in *International Conference on World Wide Web*, 2007, paper. 271–280.

[9] Kang, U., Tsourakakis, C. E., & Faloutsos, “Pegasus: mining peta-scale graphs,” in *Knowledge and information systems*, vol. 27(2), pp. 303-325, 2011.

[10] Dean, J. and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” in *OSDI*, 2004: p. 1.

[11] Dean, J. and S. Ghemawat, “MapReduce: a flexible data processing tool,” in *Commun. ACM*, vol. 53(1), pp. 72-77, 2010

[12] (2009) Nmon for linux Web Site. [Online]. Available: http://sourceforge.net/projects/nmon/