

基于应用程序的 MapReduce 性能优化

高莉莎¹, 刘正涛^{2,3}, 应毅³

(1. 南京供电公司, 江苏 南京 210019;

2. 南京航空航天大学 信息科学与技术学院, 江苏 南京 210016;

3. 三江学院 计算机科学与工程学院, 江苏 南京 210012)

摘要:针对 MapReduce 框架执行效率不佳的问题,对 MapReduce 性能优化的多个方案进行了研究。首先阐述了云计算的定义、特征以及专用批处理 PaaS 平台 Hadoop 的组成,之后简单介绍了 MapReduce 框架和 MapReduce 框架下的应用程序开发,接着着重讨论了 MapReduce 性能优化的三个主流方向:系统实现优化、参数调优、应用程序优化。并从应用程序着手,提出多个解决方法,进行了 in-Map Reduce 优化算法、脚本语言/编译语言对比、小文件预处理优化等多个实验,最后对优化技术和实验数据进行了分析。实验结果表明,优化应用程序是提高 MapReduce 性能的有效手段。

关键词:MapReduce;应用程序;性能优化;in-Map Reduce;小文件优化

中图分类号:TP316.4

文献标识码:A

文章编号:1673-629X(2015)07-0096-04

doi:10.3969/j.issn.1673-629X.2015.07.021

Performance Optimization of MapReduce Based on Applications

GAO Li-sha¹, LIU Zheng-tao^{2,3}, YING Yi³

(1. Nanjing Power Supply Company of Jiangsu, Nanjing 210019, China;

2. College of Information Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China;

3. College of Computer Science and Technology, Sanjiang University, Nanjing 210012, China)

Abstract:For the problem of poor execution efficiency under MapReduce framework, multiple solutions on performance optimization of MapReduce is studied. Firstly, the cloud computing definition and its characteristics, Hadoop composition are described in detail, then the framework of MapReduce and application development under the framework is introduced in this paper. Three main directions of performance optimization for MapReduce are also discussed, including the system optimization realized, parameters optimized and application optimization. Furthermore, multiple solutions are put forward from the application viewpoint, including in-Map Reduce optimization algorithm, script language/compile language contrast experiment, tuning for small files. Lastly, analyze optimization technique and experimental data. The experimental results show that the optimized application is an effective means to improve the performance of MapReduce.

Key words:MapReduce; application; performance optimization; in-Map Reduce; tuning for small files

0 引言

2006 年 Google 在一个搜索引擎会议上首次提出“Cloud Computing”。业界比较一致的看法是,云计算能提供动态资源池、虚拟化和高可用性的下一代计算平台。从物理形态上看,云计算接近于硬件集群,其基础设施架构在大规模廉价计算机之上;从技术理论上看,云计算是分布式处理、并行处理、网格计算、效用计

算、虚拟化技术的进一步发展,是这些科学概念综合演进结果,是能向各种互联网应用、本地应用提供硬件服务(HaaS)、基础架构服务(IaaS)、平台服务(PaaS)、软件服务(SaaS)、数据资源服务(DaaS)的系统。

1 云计算与 Hadoop

云计算具有以下特征:

收稿日期:2014-08-28

修回日期:2014-11-28

网络出版时间:2015-06-23

基金项目:江苏省卓越工程师(软件类)计划试点专业(苏教高函[2012]17号);江苏省高等学校软件服务外包类专业嵌入式人才培养项目(苏教高函[2014]14号);江苏省电力公司科技项目(J2014057);三江学院本科工程二期项目(J14021)

作者简介:高莉莎(1982-),女,高级工程师,研究方向为计算机应用技术;刘正涛,副教授,博士研究生,研究方向为 Web 数据空间、Web 数据集成。

网络出版地址:<http://www.cnki.net/kcms/detail/61.1450.TP.20150623.1031.030.html>

(1)高可用性。通过多节点冗余的方式,系统能够容忍节点错误,自动检测失效节点,并将之排除,以保证系统的正常运行。

(2)数据高可靠性。系统由硬件集群向用户提供服务,随着计算机数量的增加,出现错误的概率大大增加,在没有专用可靠性硬件的支持下,采用分布式存储的软件方式来保证数据的可靠性。

(3)可扩展性。系统能够无缝地扩展到大规模集群之上,甚至包含数千个节点同时运行。

(4)经济性。系统由廉价的硬件设备组成的集群替代小型机或大型主机,保证低成本,并能做到自动调配资源,充分使用各节点的计算能力和存储能力,降低能源消耗。

随着相关技术的发展,工业界已经实现了多个云计算实例^[1],例如:Amazon 的 EC2、微软的 Azure、IBM 的“蓝云”、阿里巴巴的“飞天”。

Hadoop^[2]是一种基于批处理技术的开源云计算平台^[3],起源于开源网络搜索引擎 Nutch,在借鉴了 Google 的 GFS^[4]和 MapReduce^[5]的设计思想之后,Hadoop 日渐成熟,于 2008 年 1 月成为 Apache 的顶级项目,并于 2011 年底发布 1.0.0 版本,说明该产品已完全做好了应用于生产的准备。Hadoop 框架由 Java 语言编写,运行在 Linux 操作系统之上,由 Hadoop Common、HDFS^[6]、Hadoop MapReduce 核心部件组成。短短几年间,Hadoop 已经成为大数据领域事实上的标准,有自己完备的生态系统。图 1 显示了 Hadoop 系统中的最主要产品,但在根本上,Hadoop 的核心仍是 MapReduce。

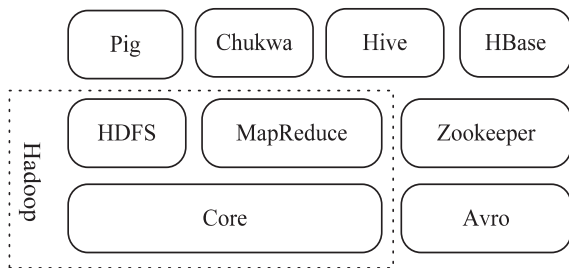


图 1 Hadoop 生态图中的主要产品

2 MapReduce 及其应用开发

MapReduce 是 Google 于 2004 年提出的能处理大数据的并行编程模型,同时也是一种高效的调度模型。“Map”、“Reduce”的概念和主要思想,都是从函数式编程语言和矢量编程语言借鉴而来的。本质上,MapReduce 的过程就是 Divide+Conquer。Map 函数负责分块数据的处理,Reduce 函数负责对分块数据处理的中间结果进行归约,如图 2 所示。通过把问题 Divide,使这些 Divide 后的 Map 运算高度并行,再将 Map

后的结果进行 Reduce,得到最终结果。

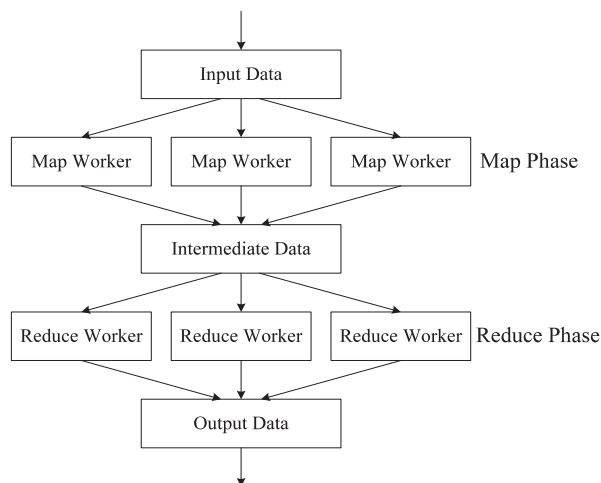


图 2 MapReduce 处理流程图

MapReduce 的最大优势在于屏蔽底层实现细节,有效降低并行编程难度,提高编程效率^[7]。框架处理了集群中的若干底层问题(例如:资源分配调度、数据分块、负载均衡、节点通信、容错),开发人员只需将精力放在应用程序本身。

MapReduce 的总体执行流程简单描述如下:Input→Map→Sort→Reduce→Output^[8]。在 Hadoop 平台上开发人员编写 MapReduce 应用需要实现的接口依次有:InputFormat、Mapper、Partitioner、Combiner、Reducer、OutputFormat。

Partitioner 接口用以指定 Map Task 产生的 Key-Value 对交由哪个 Reduce Task 处理;Combiner 接口完成 Map 节点内的规约。但实际上,开发人员只需要完成 Map 接口和 Reduce 接口,Partitioner 接口和 Combiner 接口是可选的。一般情况下 InputFormat 接口和 OutputFormat 接口使用系统默认的即可。对于少数简单的应用,甚至连 Reduce 接口都不用实现。这大大降低了并行编程的技术难度和工作量。

3 MapReduce 的性能优化

当前,MapReduce 性能优化的相关研究工作主要集中在三个方面。

第一,从系统实现角度进行优化。Yang 等在 MapReduce 框架中加入一个 Merge 操作,形成新的 MapReduceMerge 框架^[9],实现两个数据库的合并操作。文献[10]研究了计算模型、I/O 模型、数据解析、索引、调度这 5 个影响 Hadoop 效率的因素,通过提出相应的解决方案,使 Hadoop 效率提高了 2.5~3.5 倍。百度使用 C++语言对 Hadoop 中关键组件(Map、Shuffler 和 Reducer 等)进行了重写,经内部测试,发现效率提升约 20%^[11]。Sangwon 等提出 Prefetching(预取)和 Pre-shuffling(预混洗)机制来提高 MapReduce 性能^[12],在

不同规模、不同负载的集群下进行测试,发现有大约 73% 的效率提升。

第二,针对具体应用、硬件配置及集群规模进行参数调优。文献[13]尝试从自动化参数配置的角度对运行在 Hadoop 上的应用程序进行效率优化,这是一种 Hadoop 优化的新思路。在 Hadoop 的 190 多个配置项中,大约有 25 个对应用程序的效率有显著影响。例如,对于较大集群以下参数可适当调大:dfs. namenode. handler. count、mapred. job. tracker. handler. count、io. sort. mb、dfs. datanode. handler. count、mapred. reduce. parallel、dfs. block. size/mapred. min. split. size;而参数 io. sort. factor、mapred. child. java. opts、io. sort. record 等,需根据实际情况和具体应用进行配置。再如,当 Map 端数据量过大时,可以进行数据压缩,设置 mapred. compress. map. output/mapred. output. compress 参数为 true,同时改 Zlib 压缩方式为 LZO 压缩方式(Intel 内部测试表明,LZO 压缩方式的作业运行时间明显快于 Zlib 压缩方式)。

第三,根据具体应用需求从应用程序角度进行优化。MapReduce 逐行解析数据文件是 Hadoop 框架的一个特点,在大数据量的情况下,实现应用程序时规避这个弱点,是一种优化思路。但这需要根据具体应用区别对待,也需要在实践中积累经验并注意总结。

三种优化方式中,第一种往往效果明显,但难度较大;第二种需要经过大量测试不断摸索;第三种成本投入小但回报较高。

4 应用程序的优化技术

4.1 in-Map Reduce(本地规约)

为了提高系统的可靠性,MapReduce 框架在 Map 阶段会将中间结果写到磁盘上;因为绝大部分 Map Task 与 Reduce Task 处在集群的不同物理节点上,Reduce 阶段前需要跨节点拉取 Map 结果数据。磁盘 I/O 和网络数据传递严重降低了 MapReduce 的性能^[14]。解决磁盘 I/O 和网络延迟的代价相对比较昂贵,减少中间结果的数据量也可以有效提高算法效率。

通过使用 Map 阶段内置的 Combiner 操作,可以减少从 Map 传递到 Reduce 的中间 Key-Value 对的数量和规模,但 Combiner 没有在第一时间减少 Map 产生的 Key-Value 对的数量,更好的做法是在 Map 函数中直接进行本地规约,把相同 Key 的所有 Key-Value 对合并,这被称为“in-Map Reduce”。经典 WordCount(单词计数)程序的 in-Map Reduce 算法如下所示:

```
method map(text)
Map<String, int> map
while text.hasMoreWord()
```

```
word = text.getWord()
map. put( word, map.get( word)+1)
for Object obj:map. keySet()
emit( obj.getKey(), obj.getValue())
```

在 Hadoop 集群上(4 台普通 PC:1 台 Master+3 台 Slave。安装软件为:Ubuntu 12+JDK 1.6+Hadoop 0.20.203),进行了以下 2 组实验。

实验 1:使用 Python 语言完成原始算法和 in-Map Reduce 算法的 WordCount 程序,各运行两次。其执行结果如表 1 所示。

表 1 实验 1 测试结果

	Total Time	Combine Time	Sort Time
Original 1	287.18	47.25	34.13
Original 2	316.68	42.73	34.01
Optimized 1	170.73	3.33	1.67
Optimized 2	177.81	2.46	1.12

从实验结果可以看出:由于 in-Map Reduce 算法第一时间进行了数据规约,将宽带资源的占用降低,相较于原始算法,显著改善了 Combine 和 Sort 的时间,有效提高了系统整体的执行效率,如图 3 所示。

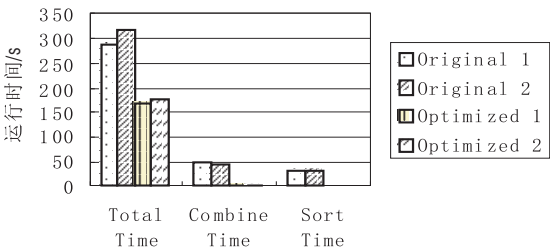


图 3 原始算法和优化算法的运行时间比较

4.2 不使用脚本语言

众所周知,在性能上解释性语言要慢于编译语言。为了有数量级上的认识,做了以下实验。

实验 2:使用 Python 语言和 Java 语言完成原始算法和 in-Map Reduce 算法的 WordCount 程序,各运行两次。系统执行的 Total Time 如表 2 所示。

表 2 实验 2 测试结果

	Original	Optimized
Python 1	287.18	170.73
Python 2	316.68	177.81
Java 1	201.36	78.35
Java 2	172.81	81.19

从实验结果可以看出:Original Python 比 Original Java 落后 60%,Optimized Java 甚至比 Optimized Python 快两倍,如图 4 所示。有理由相信,如果是数据量更大、业务更复杂的任务,差距会更大。

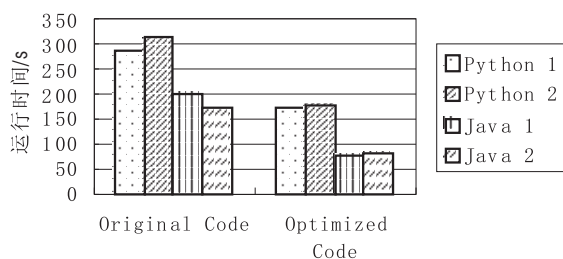


图4 Python 代码和 Java 代码的运行时间比较

4.3 使用 Java 编程中提高效率的技巧

从 4.2 的实验可以看出,编译语言比脚本语言快很多,所以在 Hadoop 平台上应该尽量使用 Java 语言,Java 编程中提高效率的技巧皆可运用在 MapReduce 编程中。

使用 StringBuffer 代替 String。当需要频繁处理字符串时,使用 StringBuffer 类型,而不是 String,因为 StringBuffer 是可修改的,而 String 是只读的,会因产生临时对象而降低性能。

根据数据处理特征选择适合的 Writable 类型。MapReduce 框架内置的 Text 对象使用起来很方便,但它在由字符串转换成文本或由数值转换成文本时都是效率很低的。当处理那些非文本的数据时,建议使用二进制的 Writable 类型(如 IntWritable、FloatWritable)。二进制 Writable 类型能避免文件转换的消耗,中间结果占用的内存空间也更少。

重用 Writable 对象。Java 程序员可能会在每次输出时都创建 Writable 对象。例如:

```
public void map(...) {  
    ...  
    for(String word;words) {  
        context.write(new Text(word),new IntWritable(1));  
    }  
}
```

以上代码的确很简洁,但会导致 JDK 分配出众多的短周期对象,JVM GC 也要做很多额外的回收工作。效率更高的写法是:

```
public void map(...) {  
    ...  
    Text wordText=new Text();  
    IntWritable one=new IntWritable(1);  
    for (String word;words) {  
        wordText.set(word);  
        context.write(wordText, one);  
    }  
}
```

4.4 海量小文件的预处理

Hadoop 最初的设计目的是为了流式访问大文件,如果存储大量小文件,系统的扩展性和性能都将受到

极大影响。Hadoop 会为每个小文件保存 Metadata 信息,由于存在多个副本,还要为其分配多个 DataNode。系统开销占据了大部分时间,用于传输文件内容的时间很少。这是造成文件存储性能下降的主要原因。所以,MapReduce 处理大量小文件的速度远远小于处理同等大小的大文件的速度。

为了解决该问题,可以在 Master 节点上添加一个小文件预处理模块。它的设计思想是:将很多小文件合并成一个大文件,之后再将大文件 Copy 到 HDFS 中。操作的具体过程是:需要 Hadoop 处理的文件不是直接上传到 HDFS,而是先上传到本地的文件系统,预处理模块会先判断该文件是否小于设定的文件大小阈值,如果小于阈值,则交给预处理模块处理;否则,Copy 到 HDFS 中。

Apriori-MR 算法是基于 MapReduce 编程模型的并行频繁集算法^[15],在进行点菜单关联规则的数据挖掘时,遇到了大量的小文件,总共大约有 18 000 个,每个文件的大小为 6~10 kB。在由 1 个 NameNode、3 个 DataNode 组成的 Hadoop 环境中,未优化前存入集群总耗时需要 25~30 min。经过小文件预处理模块的优化,将 18 000 个小文件合并成 5 个文件,每个文件大约 30 M 左右,存入 Linux 本地文件系统、合并及存入 Hadoop 集群总耗时缩短为 75 s。

4.5 其他优化技术

为 Job 添加一个 Combiner。通常情况下,Combiner 与 Reducer 相同。该方法能减少 Shuffle 阶段从 Map Task 拷贝给 Reduce Task 的数据量。

缓存外部文件。有些外部文件(如配置文件、数据字典)会被应用程序频繁使用,或需要在所有 Task 之间共享,可考虑将这些文件放置到分布式缓存内。

避免不必要的 Reduce 任务。对于某些特殊的应用,数据在处理前已经分区了,则只需要自定义 InputSplit,将单个分区对应单个 Map 输入,仅 Map 处理数据,Reduce 设置为空即可。

5 结束语

作为云计算的核心,MapReduce 技术在大数据领域发挥着越来越重要的作用。文中详细讨论了当前优化 MapReduce 性能的三种方案。在应用程序层面提出若干优化技术,包括:in-Map Reduce 算法、使用 Java 语言、重用 Writable 对象、小文件预处理等,并且做了测试。

实验证明:应用程序级别的优化技术具有投入小回报高的特点,是提高 MapReduce 性能行之有效的

- ternational conference on wavelet analysis and pattern recognition. Beijing: IEEE, 2007: 1333–1340.
- [24] Chen Changhong, Liang Jimin, Zhao Heng, et al. Frame difference energy image for gait recognition with incomplete silhouettes[J]. Pattern Recognition Letters, 2009, 30(11): 977–984.
- [25] Xue Zhaojun, Ming Dong, Song Wei, et al. Infrared gait recognition based on wavelet Transform and support vector machine[J]. Pattern Recognition, 2010, 43(8): 2904–2910.
- [26] Dadashi F, Araabi B N, Soltanian-Zadeh H. Gait recognition using wavelet packet silhouette representation and transductive support vector machines[C]//Proc of 2nd international congress on image and signal processing. Tianjin: IEEE, 2009: 1–5.
- [27] 赵子健, 吴晓娟. 基于近似时空切片向量的步态识别方法研究[J]. 模式识别与人工智能, 2005, 18(5): 608–614.
- [28] 刘志勇, 冯国灿, 邹小林. 一种基于静态和动态特征的步态识别新方法[J]. 计算机科学, 2012, 39(4): 261–264.
- [29] Wang Liang, Ning Huazhong, Tan Tieniu, et al. Fusion of static and dynamic body biometrics for gait recognition[J]. IEEE Trans on Circuits and Systems for Video Technology, 2004, 14(2): 149–158.
- [30] Nandic C, Ravi-Kumar C N. An approach to gait recognition[C]//Proceedings of international symposium on biometrics and security technologies. Islamabad: [s. n.], 2008: 1–3.
- [31] 林敏, 陈淑清. 基于特征组合的步态识别算法研究[J]. 广西工学院学报, 2012, 23(2): 60–64.
- [32] 柴艳妹, 韩文英, 刘灿涛, 等. 融合理论在步态识别中的应用研究[J]. 计算机科学, 2012, 39(12): 272–277.
- [33] Gregory S, Trevor D. On probabilistic combination of face and gait cues for identification[C]//Proceedings of the 5th international conference on automated automatic face and gesture recognition. Washington: [s. n.], 2002: 169–174.
- [34] Kusakunniran W, Li Hongdong, Wu Qiang, et al. Cross-view and multi-view gait recognitions based on view transformation model using multilayer perceptron[J]. Pattern Recognition Letters, 2012, 33(7): 882–889.
- [35] Lee B, Hong S, Lee H, et al. Gait recognition system using decision-level fusion[C]//Proceedings of Fifth IEEE international conference on industrial electronics and applications. Taichung: IEEE, 2010: 313–316.
- [36] Imed B, Mark N. Exploratory factor analysis of gait recognition[C]//Proceedings of 2008 eighth IEEE International conference automatic face and gesture recognition. Amsterdam: IEEE, 2008: 1–6.
- [37] Popoola O P, Wang Kejun. Video-based abnormal human behavior recognition—a review[J]. IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews, 2012, 42(6): 865–878.
- +++++
- (上接第 99 页)
- 参考文献:**
- [1] 张建成, 宋丽华, 鹿全礼, 等. 云计算方案分析研究[J]. 计算机技术与发展, 2012, 22(1): 165–167.
- [2] White T. Hadoop 权威指南[M]. 周敏奇, 王晓玲, 金澈清, 等, 译. 第 2 版. 北京: 清华大学出版社, 2011.
- [3] 周洪波. 云计算: 技术、应用、标准和商业模式[M]. 北京: 电子工业出版社, 2011.
- [4] Ghemawat S, Gobioff H, Leung S T. The Google file system[C]//Proc of ACM symposium on operating systems principle. New York: ACM, 2003: 29–43.
- [5] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107–113.
- [6] Shvachko K, Kuang H, Radia S, et al. The hadoop distributed file system[C]//Proc of 2010 IEEE 26th symposium on mass storage systems and technologies. [s. l.]: IEEE, 2010: 1–10.
- [7] 李建江, 崔健, 王聃, 等. MapReduce 并行编程模型研究综述[J]. 电子学报, 2011, 39(11): 2635–2642.
- [8] 张密密. MapReduce 模型在 Hadoop 实现中的性能分析及改进优化[D]. 成都: 电子科技大学, 2010.
- [9] Yang H, Dasdan A, Hsiao R L, et al. Map-reduce-merge: simplified relational data processing on large clusters[C]//Proceedings of the 2007 ACM SIGMOD international conference on management of data. [s. l.]: ACM, 2007: 1029–1040.
- [10] Jiang D, Ooi B C, Shi L, et al. The performance of mapreduce: An in-depth study[J]. Proceedings of the VLDB Endowment, 2010, 3(1–2): 472–483.
- [11] Hadoop C++ extention[EB/OL]. (2012-07-03) [2014-10-15]. <https://issues.apache.org/jira/browse/MAPREDUCE-1270>.
- [12] Seo S, Jang I, Woo K, et al. HPMR: prefetching and pre-shuffling in shared MapReduce computation environment[C]//Proc of IEEE international conference on cluster computing and workshops. [s. l.]: IEEE, 2009: 1–8.
- [13] Babu S. Towards automatic optimization of MapReduce programs[C]//Proceedings of the 1st ACM symposium on cloud computing. [s. l.]: ACM, 2010: 137–142.
- [14] 彭辅权, 金苍宏, 吴明晖, 等. MapReduce 中 Shuffle 优化与重构[J]. 中国科技论文, 2012, 7(4): 241–245.
- [15] 应毅, 任凯, 刘正涛. 基于云计算技术的数据挖掘[J]. 微电子学与计算机, 2013, 30(2): 161–164.

基于应用程序的MapReduce性能优化

作者：[高莉莎](#)，[刘正涛](#)，[应毅](#)，[GAO Li-sha](#)，[LIU Zheng-tao](#)，[YING Yi](#)

作者单位：[高莉莎, GAO Li-sha\(南京供电公司, 江苏 南京, 210019\)](#)，[刘正涛, LIU Zheng-tao\(南京航空航天大学 信息科学与技术学院, 江苏 南京 210016; 三江学院 计算机科学与工程学院, 江苏 南京 210012\)](#)，[应毅, YING Yi\(三江学院 计算机科学与工程学院, 江苏 南京, 210012\)](#)

刊名：[计算机技术与发展](#)

英文刊名：[Computer Technology and Development](#)

年，卷(期)：2015(7)

引用本文格式：[高莉莎](#).[刘正涛](#).[应毅](#).[GAO Li-sha](#).[LIU Zheng-tao](#).[YING Yi](#) [基于应用程序的MapReduce性能优化](#)[期刊论文]-[计算机技术与发展](#) 2015(7)