PostgreSQL 数据库在不同调节器中的能效比较①

王 萌, 杨良怀, 王英姿, 潘 建

(浙江工业大学 计算机科学与技术学院, 杭州 310014)
(浙江省可视媒体智能处理技术研究重点实验室, 杭州 310014)

摘 要: 近年来, 能效数据库系统成为数据库领域的一个研究议题.CPU 动态电压频率调节(DVFS)是一种有效的动态功率节能技术. 探寻 PostgreSQL 数据库在 ACPI 不同调节器下查询操作的性能、能耗、功率之间潜在联系, 发现动态功耗管理与数据库系统的能效关系, 通过运行 TPC-H 测试基准生成的数据库与相应 22 个查询, 总结出调节器对数据库查询处理各种操作的影响.实验结果表明, DVFS 可以对 DBMS 进行动态功耗管理是有效的, 查询处理的不同操作具有各自特性, 利用这些特性来设计效率更高的调节器是颇有前途的.

关键词: 数据库系统; 能效; ACPI; 动态电压频率调节

Energy Efficiency Comparisons for PostgreSQL Under Different Governors

WANG Meng, YANG Liang-Huai, WANG Ying-Zi, PAN Jian

(School of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310014, China)
(Key Laboratory of Visual Media Intelligent Process Technology of Zhejiang Province, Hangzhou 310014, China)

Abstract: Energy-efficient DBMS has become a hot research topic for recent years. CPU DVFS is an effective dynamic power management scheme. This paper looks into the relationships between performance, energy and power in PostgreSQL under different governors with the aim to discover the power saving effectiveness in DBMS from hardware perspective. After running 22 TPC-H benchmark queries, we summarize the effects of 4 governors on different query processing operators. The experiment results show that CPU DVFS is effective on DBMS energy-efficiency, and different query operators have their respective power usage characteristics and can be exploited to design more efficient customized governors.

Key words: database system; energy efficiency; ACPI; DVFS

1 引言与相关工作

1.1 概述

如果计算机能耗继续增长, 其能耗成本将超过硬件的成本[1].能耗增长必然导致发热, 其直接负面影响就是系统稳定性大幅降低[2].针对计算机系统的能耗问题, 大量研究已在 CPU、内存、磁盘存储系统等多个层次硬件上展开[3-7]; 在软件层面也开展了一些研究, 如文献[8].

在过去, 数据库管理系统(DBMS)的研究是以性能为目标.近年来, 能效计算也在数据库领领域引起了重视.由于能效的重要性, 事务处理委员会(TPC)在 09 年底公布了 TPC-Energy[9]规范, 把能效度量(瓦/性能)增加到所有的基准测试中.在数据库软件方面, 文献[10]指出了三种可能的能效途径: 能量知晓的优化、资源使用统筹以及软件部件的重新设计.在利用硬件节能方面, Bale[11]针对不同的数据库系统执行数据库查询时的峰值功率进行了剖析, 得出 CPU 是系统动态功率的主要部件.

1.2 DVFS 的相关研究

DVFS(Dynamic Voltage and Frequency Scaling), 即动态电压与频率调节, 是近年来被广泛应用的一种有效的处理器节能技术[12].DVFS 可以根据处理器当前的工作量来调节电压与频率[13].在 CMOS 逻辑电路中, 处理器的功率主要有动态功率(dynamic power)和静态功率(static power)两部分组成[14].其中, 动态功率是指

① 基金项目:国家自然科学基金(61070042);浙江省自然科学基金(Y1090096)
收稿时间:2012-09-05;收到修改稿时间:2012-10-17

随电压与频率的改变而改变的功率, 它与当前电压的平方成正比, 与当前频率成正比, 也就是说, 动态功率与频率的三次方成正比[2].而静态功率是指当没有任何指令运行时处理器所消耗的基本功率.

在计算机系统中, 能效是指消耗单位能量所完成的有效工作负载[3].所以, 能效可以看作是系统性能与功率的比例.文献[7]等通过能量正比性的概念, 指出现有的系统需要对能效进行优化.Hsu 等[6]通过实验得出结论: 对于 CPU 密集型的任务, 系统节能的机会极少, 这时只有通过增加处理器的个数来实现节能.

1.3 数据库节能相关工作

Lang 等[15]第一次提出了数据库能效优化方法, 他们的主要思想是“用性能换能耗”, 并以此为基础提出了两种降低数据库管理系统能耗的方法: 一种是 PVC(Processor Voltage/frequency Control), 这项技术可以使得 CPU 接受来自软件的指令并在处理器各能耗级中进行切换; 另一种方法是 QED, 它的核心思想是把查询任务积累到一定数量后开始处理, 利用共享公共子表达式的方法来减少重复的查询操作从而降低能耗.两种方法均获得明显的节能效果.

Xu 等[16]认为, 仅仅依靠硬件的改良还不足以做到对数据库系统的“能效管理”, 他们提出了两种提高数据库能效的方法, 即增加吞吐量和适当降低系统性能.文献[16]还提出了能耗代价评估模型, 将传统的数据库代价估算模型与能量结合在一起, 并通过实验证实了这个模型的有效性.

针对不同的数据库系统, Bale[11]使用一组数据库查询操作, 分析了它们在不同数据库上的峰值功率, 并总结出不同的功耗模式.但作者并没有深入到分析各个操作具体功耗规律.

本文使用不同 ACPI 调节器考察 PostgreSQL 中查询执行的性能与能耗情况, 并总结出调节器对数据库查询处理各种操作的影响, 为后续研究工作的展开提供指导.

本文的剩余部分组织如下: 第 2 节介绍能效比较方法, ACPI 调节器相关特性以及实验系统; 第 3 节给出了实验结果, 对其进行了分析和比较; 最后, 我们进行了小结.

2 能效实验方法

2.1 调节方法

1997 年, Intel、Microsoft 和 Toshiba 推出了一个开放的工业标准——高级配置与电源接口(ACPI)[17].它允许操作系统控制电源管理.提供若干调节器(governor), 可以动态调整处理器的频率与电压, 从而改变处理器的能耗. ACPI 提供了不同的处理器状态 *G*0~*G*3, 各状态的能耗随编号依次递减. *G*0 态即正常工作状态, 它包括处理器 *C* 态, 处理器 *P* 态和处理器 *T* 态. *C* 态(Executing CPU Power State)包括 *C*0~*C*n 态, 能耗依次递减; *P* 态(Performance State)可以将 CPU 的频率与电压设置为给定的任一离散值. *T* 态(Throttle State)只能调节频率. *G*1~*G*3 态是睡眠态.

在 Linux 系统中提供了 cpufreq 这个模块用来实现对频率和电压的控制.这个模块允许用户自定义调节策略, 使 CPU 在 *P* 状态各级之间进行调节.它提供了调节器机制, 主要包含以下几种:

(1) Performance: CPU 一直处于 *P*0 态来保证系统的最优性能.

(2) Powersave: CPU 一直处于 *Px* 状态(*x* 为处理器所能达到的最大状态号)来保证系统的最低能耗.

(3) Ondemand: 系统根据当前 CPU 的利用率动态的调节运行频率, 当利用率低于预设下限阈值时, 降低频率来节能; 反之, 则频率升至预设值来保证性能.

(4) Conservative: 系统采用类似 Ondemand 的调节方式, 但规定电压/频率升降必须逐级转换.若 CPU 利用率超过上限阈值, 调节器调高一级频率; 反之则调低一级频率.

(5) Userspace: 用户可以利用 ACPI 给定的接口来编写程序, 根据应用程序的各种上下文、条件构建专门的调控策略, 自由控制处理器的电压与频率.

本文针对前 4 种调节器考察 PostgreSQL 查询操作的性能与能耗情况, 并总结出各个查询操作的特性.

2.2 实验系统

我们利用杭州远方光电信息有限公司的 PF9808B 数字功耗仪, 搭建面向能效的查询处理与优化的算法试验平台, 如图 1 所示. PostgreSQL 运行在被测系统(数据库服务器)中, 数字功耗仪按每秒 1 次采样记录被测系统功耗, 具体采样数据由监控测量系统通过串口读取.PostgreSQL 系统中加载 TPC-H 测试基准所生成的数据库, 运行 TPC-H 基准测试程序生成的查询语句.为排除能耗仪采样程序的干扰, 采用双机通信的方式进行实验.

表 1 是被测系统硬件参数, PostgreSQL 数据库与 ACPI 均采用默认配置. 8 个数据表的大小如表 2 所示.

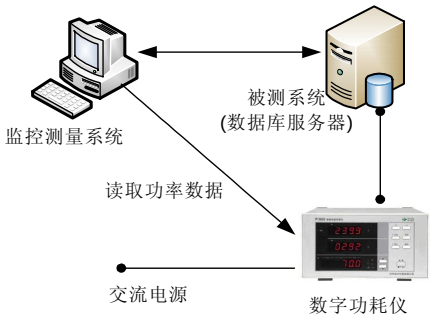


图 1　功耗试验示意图

表 1　实验环境

硬件环境	CPU	Intel® Core™2 Quad Processor Q8200	频率范围: 1.20GHz~2.80GHz 额定功率: 95W
	内存	Kingston DDR2	容量: 2GB*2 额定功率: 3W*2
	硬盘	Seagate ST3500418AS	容量: 1TB 额定功率: 7W
	能耗仪	远方 PF9808B	采样频率: 1s
软件环境	OS	Ubuntu Linux 10.04 LTS	
	数据库	PostgreSQL 8.4	
	Benchmark	TPC-H 2.8.0	

表 2　数据表的大小

数据表名	大小	数据表名	大小
Customer	468.1 MB	Part	466.6 MB
Lineitem	2.0 GB	Partsupp	2.0 GB
Nation	2.2 KB	Region	389 BYTE
Orders	2.0 GB	Supplier	27.2 MB

3　实验结果与分析

对 22 个查询执行后进行实验后发现，调节器对查询操作的影响主要有以下几类:

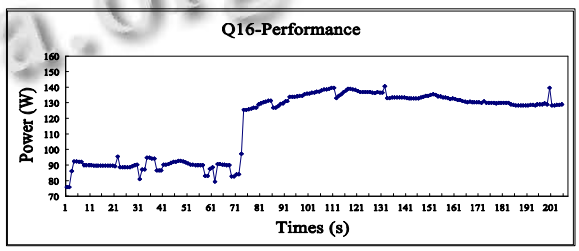
(1) 查询在 Powersave 下运行最慢，即 CPU 处理器频率越低，查询执行的速度越慢.这样的查询占大多数，以查询 16 为例，该查询是在 *WHERE* 子句中加入子查询，并采用 *IN* 和 *NOT IN* 对子查询结果作筛选，最后用 *COUNT* 统计查询结果并对结果进行分组、排序.查询 16 的实验结果如表 3 所示.

表 3　查询 16 的实验结果

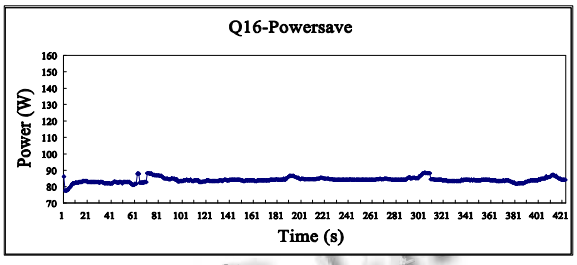
	Perfermance	Powersave	Ondemand	Conservative
时间	206.0 s	427.4 s	207.7 s	207.0 s
能耗	24190.9 J	35578.5 J	23722.1 J	23643.6 J
节能百分比	--	-47.1%	1.9%	2.3%
动态能耗	7978.7 J	1942.1 J	7376.1 J	7352.7 J
节能百分比	--	75.7%	7.6%	7.8%

显然, Powersave 调节器下功率始终稳定在 82W 左右，这是由于 Powersave 只允许 CPU 一直以最低的频率运行，不会出现功率大幅度波动.而其他三种调节器下，功率在 70s 左右跃升至 130W 左右，这是由于出现了哈希连接与外部归并排序两种操作，其中哈希连接需要进行键值匹配，而外部归并排序需要比较键值的大小，这两种同属 CPU 密集型操作，故 CPU 利用率升高，从而导致峰值功率出现.

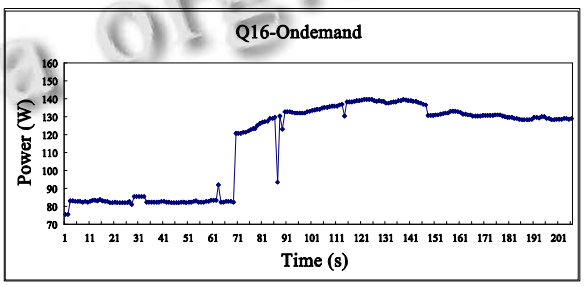
从表中可以看出：查询 16 在 Ondemand 与 Conservative 下执行可以在不牺牲性能的基础上节约 2%的总能耗和 8%的活动能耗，这是因为 Performance 下 CPU 始终处于最高频率，在内存密集型操作出现时，CPU 空转消耗的能量比低频率态的 CPU 能耗要低.另外由于 Powersave 下运行时间延长，静态能耗增加，虽然 CPU 功率较低，但过长的运行时间使得查询所需要的能耗总体增加.



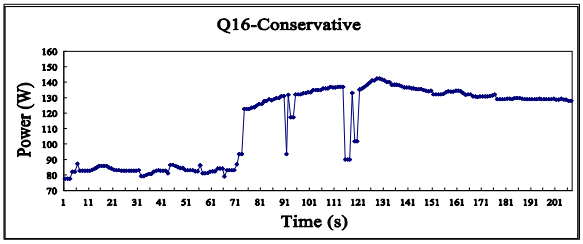
(a) Performance



(b) Powersave



(c) Ondemand



(d) Conservative

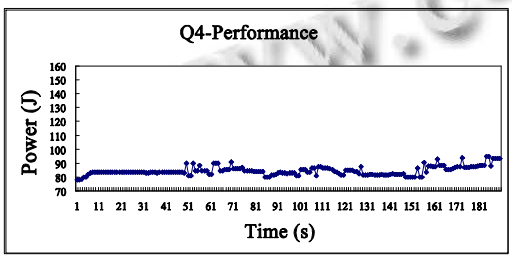
图 2　查询 16 功率图

(2) 查询在四种调节器下执行时间大体相同.以查询 4 为例, 该查询是在 WHERE 条件中采用子查询, 并用 EXISTS 对子查询结果作筛选, 最后对查询结果进行分组、排序. 查询 4 的实验结果如表 4 所示:

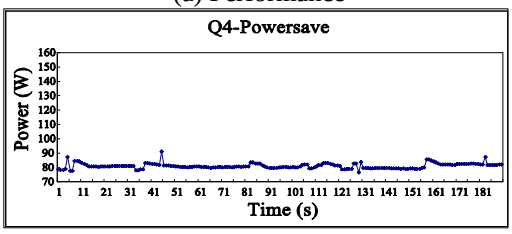
表 4 查询 4 的实验结果

	Perfermance	Powersave	Ondemand	Conservative
时间	192.0 s	191.8 s	191.9 s	192.4 s
能耗	16018.9 J	15429.0 J	15602.7 J	15528.4 J
节能百分比	--	3.7%	2.6%	3.1%
动态能耗	908.5 J	334.3 J	500.2 J	386.5 J
节能百分比	--	63.2%	44.9%	57.5%

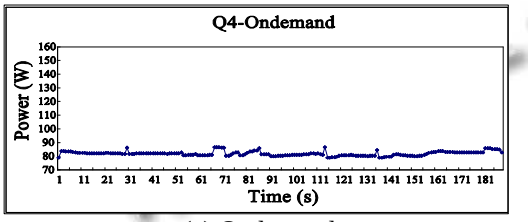
显然, 无论在哪种调节器下, 系统的功率始终保持在 80W 左右, 在 Powersave、Ondemand 和 Conservative 下, 系统的动态能耗都有大幅度的降低.查询 4 中的操作主要是顺序扫描, 属于内存密集型操作, 不需要太高的 CPU 利用率.



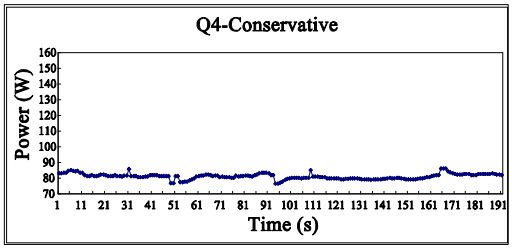
(a) Performance



(b) Powersave



(c) Ondemand



(d) Conservative

图 3 查询 4 功率图

结果表明, 查询 4 在 Powersave 下执行可以节约 4%的总能耗与 63%的活动能耗且执行时间不受影响.

(3) 查询在 Performance 下反而运行最慢.以查询 2 为例, 结果如表 5 所示.此类查询较为罕见, 仅在查询 2 与查询 19 中出现, 具体原因在后面深入探讨.从数据中可知: Performance 下执行查询 2, 速度慢, 能耗高, Powersave 可以节约 8%的总能耗和 65%的活动能耗, 节能效果明显.

表 5 查询 2 的实验结果

	Perfermance	Powersave	Ondemand	Conservative
时间	232.6 s	224.9 s	226.0 s	225.9 s
能耗	19854.8 J	18230.2 J	18747.2 J	18429.6 J
节能百分比	--	8.2%	5.6%	7.2%
动态能耗	1549.2 J	530.6 J	961 J	651.3 J
节能百分比	--	65.8%	38.0%	58.0%

由前文可知, 若负载一定, 则消耗能量越少, 能效就越大.综上分析, 对于第一类查询, Ondemand 和 Conservative 可以收获最大的能效, 而对于第二类与第三类查询, Powersave 可以收获最大的能效.

为了进一步深入研究各个数据库操作在不同调节器下的能效, 我们对每个查询进行深入的剖析.PostgreSQL 提供了 EXPLAIN ANALYZE 命令来查看查询的语法分析.

以查询 2 为例, 该查询是在 WHERE 条件中加入子查询.其查询结果如表 5 所示, 其功率图谱如图 4 所示.经 *EXPLAIN ANALYZE* 后的结果如表 6 所示:

表 6 查询 2 执行结果分析

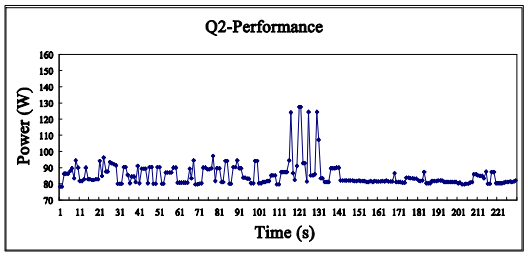
	Performance	Powersave	Ondemand	Conservative
Merge join	98 s	74 s	79 s	76 s
Index scan	9 s	6 s	7 s	7 s
Materialize	3 s	9 s	7 s	8 s
Ex-merge sort	2 s	5 s	4 s	4 s
Hash join	120 s	122 s	119 s	118 s

从表中可以看出, 在 Performance 下, 哈希连接、外部归并排序执行速度最快.扫描和归并连接反而最慢.这也是其总体运行时间在 Performance 下偏慢的原因.

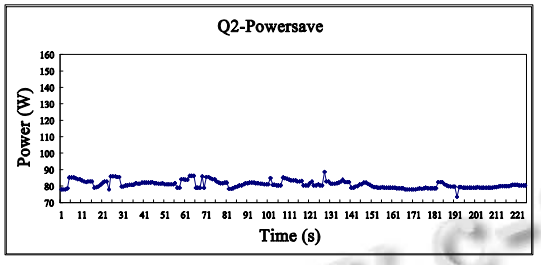
对 22 个查询进行分析后, 我们将 PostgreSQL 数据库基本操作分为两类: 第一类是在 Performance 下运行会加快的操作, 主要有: 哈希连接、外部归并排序、等, 它们包含长时间的 CPU 密集型操作, 所以它们需要较高的 CPU 频率; 第二类是不受 Performance 影响的操作, 主要是顺序/索引扫描, 它们基本属于内存密集型操作, 不需要高 CPU 频率.

给定的四种调节器是基于当前 CPU 利用率来调节 CPU 频率, 它们是事后被动调节; 而我们的实验结果说明, 可以利用查询计划中操作的特征来实现主动

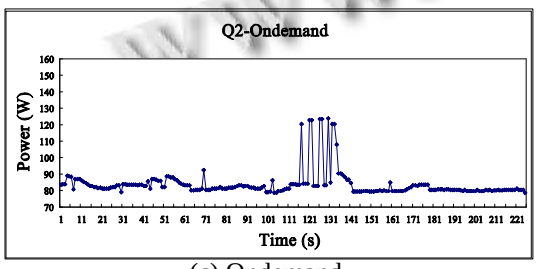
调节, 从而获得更高能效.



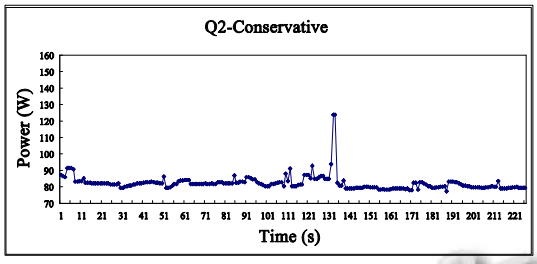
(a) Performance



(b) Powersave



(c) Ondemand



(d) Conservative

图 4 查询 2 功率图

4 结论与展望

本文通过 Linux 系统中 4 个调节器对 PostgreSQL 数据库管理系统执行 TPC-H 22 个查询的性能、能量、功率进行剖析, 总结出 PostgreSQL 数据库操作的一些功率特性, 探寻动态功耗管理与数据库系统的能效关系.从实验结果分析可知, 处理器频率变化会造成运行时间的变化, 在性能与能耗之间做出权衡是 DBMS 动态功耗管理中十分重要的议题; 查询处理的不同操作具有各自特性, 利用这些特性来设计效率更高的调节器是颇有前途的.

在将来的工作中, 我们将分析查询操作特征, 设计查询敏感的频率调节算法, 利用 Userspace 实现更好的调节效果.

参考文献

1 Barroso LA. The price of performance. ACM Queue, 2005,3(7):47–53.

2 Albers S. Energy-efficient algorithms. Communications of the ACM, 2010,53(5):86–96.

3 Barroso LA, Hölzle U. The case for energy-proportional computing. IEEE Computer, 2007,40(12):33–37.

4 Ishihara T, Yasuura H. Voltage scheduling problem for dynamically variable voltage processors. Proc. of the International Symposium on Low Power Electronics and Design. 1998: 197–202.

5 Qu G. What is the limit of energy saving by dynamic voltage scaling? Proc. of the IEEE/ACM International Conference on Computer-aided Design. 2001: 560–563.

6 Hsu CH, Feng W. Effective Dynamic Voltage Scaling through Accurate Performance Modeling. http://sss.lanl.gov/pubs/tr03-7582.pdf.

7 Yao F, Demers A, Shenker S. A scheduling model for reduced CPU energy. Proc. of the 36th IEEE Symposium on Foundations of Computer Science. 1995: 374–382.

8 Zeng H, Ellis CS, Lebeck AR, Vahdat A. Ecosystem: managing energy as a first class operating system resource. SIGPLAN Notices, 2002,37(10):123–132.

9 TPC-Energy Specification. http://www.tpc.org/tpc_energy/default.asp.

10 Harizopoulos S, Shah M, Ranganathan P. Energy efficiency: The new holy grail of data management systems research. Conference on Innovative Data Systems Research. 2009.

11 Bale M. Power Profiling of Database Engines.http://dsl.serc.iisc.ernet.in/publications/thesis/mahesh.pdf.

12 Hung CM, Chen JJ, Kuo TW. Energy-efficient real-time task scheduling for a DVS system with a non-dvs processing element. Proc. of the 27th IEEE International Real-Time Systems Symposium. 2006: 303–312.

13 Cong J, Gururaj K. Energy efficient multiprocessor task scheduling under input-dependent variation. Proc. of the Conference on Design, Automation and Test in Europe.

(下转第 24 页)

有中间数据被写入磁盘等待处理. 因此, 磁盘I/O的效率直接关系了 map/reduce 数据处理的效率. 文献[6]和文献[7]给出了两种基于 map/reduce 的优化方案, 本质是开发基于 map/reduce 程序的分类算法, 提高数据分析的效率. 这两种方法, 没有从本质上解决大数据处理所面临的 I/O 压力.

表 2 Replication 值对 HDFS 效率的影响

存储 457M 数据	Replication =1	Replication =2	Replication =3
所需时间（秒）	310	473	580

本文给出了两种优化 map/reduce 性能的方法. 根据对 Hadoop 系统的研究, 提出了数据局部性优化和 I/O 缓存优化, 本质上都是提高磁盘 I/O 性能. 数据局部性优化提供了 combiner 函数, 减少了 map 到 reduce 的 I/O. I/O 缓存优化是配置 Hadoop 数据 I/O 缓冲区的大小, 从默认的 4096 字节, 增加到 65536 字节. 实验证明, map/reduce 程序性能有了明显提高. 表 3 列出了两种典型的优化方案.

表 3 map/reduce 程序处理时间比较

单位（秒）	第一组数据	第二组数据	第三组数据
未优化	284	223	251
局部优化	217	157	154
I/O 优化	134	140	149

4 结语

本文应用Hadoop云计算模型, 提高了数据存储的安全性, 数据分析的效率, 满足了应用的要求, 提供了更好的用户体验. 同时, 分析了 map/reduce 计算模型的瓶颈, 从而根据应用对环境进行优化, 并进行验证, 得出map/reduce程序的优化应从I/O性能着手. 实验结果表明, HDFS 文件系统能够很好的容错, map/reduce 具有高效的数据分析能力, 完全可以替代传统的单机的数据存储、分析. 今后要实践解决的问题是, 如何快速的跨平台向 Hadoop 提交数据, 降低数据移植给 Hadoop 带来的效率影响.

本文的局限性在于资源有限, 只能构建虚拟集群, 并没有构成规模. 所以, 主要验证了 Hadoop 投入生产使用的可行性, 以及在技术上遇到的问题, 通过优化系统配置和程序代码, 提高数据分析的效率.

参考文献

1 TOM Wbite. Hadoop: The Definitive Guide. US: O'Reilly. 2005.

2 Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Communications of the ACM, 2005,51(1): 107−113.

3 Dhruba B. The Hadoop Distributed File System: Architecture and Design.2007.

4 Dean J, Ghemawat S. Distributed programming with Mapreduce. In: Oram A, Wilson G, eds. Beautiful Code. Sebastopol: O'Reilly Media, Inc., 2007: 371−384.

5 李丽英,唐卓,李仁发.基于 LATE 的 Hadoop 数据局部性改进调度算法.计算机科学,2011,11.

6 丁光华,周继鹏,周敏.基于 MapReduce 的并行贝叶斯分类算法的设计与实现.微计算机信息,2010,9.

7 李应安.基于 MapReduce 的聚类算法的并行化研究.微计算机信息,2010,9.

8 Hadoop. http://wiki.apache.org/hadoop/Hbase/PerformanceEvaluation.

(上接第 20 页)

2009: 411−416.

14 Jejurikar R, Gupta R. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. Proc. of the 42nd Annual Design Automation Conference. 2005: 111−116.

15 Lang W, Patel J. Towards eco-friendly database management Systems. Proc. of the 4th Biennial Conference on Innovative Data Systems Research. 2009.

16 Xu Z. Building a power-aware database management system. Proc. of the 4th SIGMOD PhD Workshop on Innovative Database Research. 2010: 1−6.

17 Brown L, Keshavamurthy A, Li DS. ACPI in Linux: Architecture, Advances and Challenges. Intel Open Source Technology Center, 2005.