基于参数优化的 Hadoop 云计算平台①

李 寒, 唐兴兴

(桂林电子科技大学 计算机科学与工程学院, 桂林 541004)

摘 要: 传统的数据分析, 很难满足现阶段大数据处理效率的要求. Hadoop 云计算技术的应用, 实现了海量数据存储和分析, 提高了数据存储和分析的效率. 在总结传统系统利弊的基础上, 以 Hadoop 分布式文件系统(HDFS)取代现有的单机数据存储, 以 map/reduce 应用程序取代传统的单机数据分析, 并对其做出优化. 实验证明, Hadoop 系统架构在生产上部署、投入使用的可行性.

关键词: 云计算; Hadoop; 数据分析; map/reduce

Hadoop Cloud Computing Model Based on Parameters Optimization

LI Han, TANG Xing-Xing

(School of Computer Science and Engineering, Guilin University of Electronic Technology, Guilin 541004, China)

Abstract: The traditional data analysis, it is difficult to meet data processing efficiency requirements at this stage. Application of Hadoop cloud computing technology, realization of data storage and analysis, improve the efficiency of data storage and analysis. This paper is on the basis of summing up the pros and cons of traditional system, with Hadoop Distributed File System (HDFS) to replace the existing stand-alone data storage, map/reduce application instead of the traditional stand-alone data analysis, and make the optimization. The experiment approves that the feasibility of Hadoop system architecture in the production deployment and using.

Key words: cloud computing; Hadoop; data analysis; map/reduce

云计算在大数据处理方面, 尤其针对几百 MB、几百 GB、甚至几百 TB 大小的文件, 有了很好的应用, 目前已经有存储 PB 级数据的 Hadoop 集群了[1]. Google 关于 GFS、MapReduce[2]、BigTable 的三篇论文是云计算领域的经典. Apache 按照这三篇论文, 用 Java 实现了开源的云计算 Hadoop 系统, 性能上 Hadoop 不比 Google 优良, 却也不影响 Hadoop 被业界广泛接受.

1 Hadoop系统介绍

Hadoop 由两个核心构件组成, Hadoop 分布式文件系统 HDFS(Hadoop Distributed File System)和 map/reduce 计算模型.

表 1 Apache 与 Google 云计算产品性能比较[8]

Experiment Run	Mapfile0.19.0	BigTable
Random reads	768	1212
Random reads(mem)	Lag badly	10811
Random writes	Lag badly	8850
Sequential reads	Lag badly	4425
Sequential writes	7519	8547
Scans	55555	15385

HDFS 保留了传统文件系统特征的同时, 也有海量数据存储、高性价比、可靠性、可扩展性等云计算领域的特征. HDFS 集群是由一个名字节点(NameNode)和多个数据节点(DataNode)构成. 一个大数据文件被分成多个块, 这些块存储在 DataNode 中, 由 NameNode 确定每

① 收稿时间:2012-08-07;收到修改稿时间:2012-09-11

个块与DataNode的映射, 并且命令DataNode进行文件或目录操作, 如open、read、write、close等. NameNode的容错很重要, NameNode停机会造成HDFS数据的丢失, 安全起见Hadoop会有NameNode的备份, 一旦NameNode停机或异常, SecondaryNameNode便会接管NameNode的工作, 而用户却不容易觉察到明显的中断[3].

map/reduce计算模型由一个调度协调任务的JobTracker和多个执行来自JobTracker指令的TaskTracker组成[4]. map/reduce程序提交给JobTracker后, JobTracker获取当前网络拓扑中最优的数个节点, 将map和reduce任务交付给所选节点的TaskTracker. 任务执行过程中, 通过心跳机制, TaskTracker向JobTracker报告任务进度, 通过更改heartbeat.recheck.interval属性可以设置心跳的时间. 当TaskTracker不能完成任务或任务失败的时候, JobTracker会选取效率高的TaskTracker的结果, 或者将任务重新下发给其他节点的TaskTracker.

图1, map/reduce计算模型提供了任务的分解(map)和规约(reduce)来进行分布式计算. map函数对文件的每一行进行处理, 产生(Key/Value)对. reduce以map的输出为输入, 将相同的Key值规约至同一reduce, reduce进一步处理后, 产生新的数据集. 形式化过程如下:

map: (k1, v1) → (k2, v2) → (k2, list(v2))

reduce: (k2, list(v2)) → (k3, v3)

map将(k1, v1)经过处理, 产生数据集(k2, v2), 归纳为(k2, list(v2)); 将(k2, list(v2))交予reduce处理, (k2, list(v2))成为reduce函数的输入, 产生map/reduce计算的结果(k3, v3).

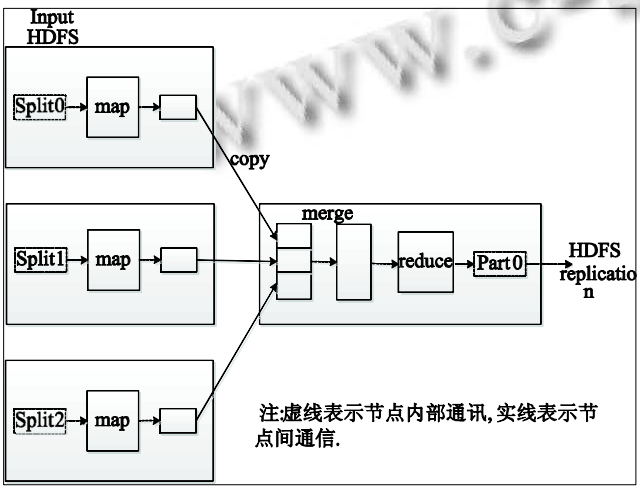

图1 map/reduce计算模型

2 云计算技术

数据存储和分析效率, 是判断服务优劣程度的重要指标. 多年来, IT界内存容量和磁盘容量大幅增加, 然而数据存储和数据分析的效率却未能满足企业的要求. 将云计算产品Hadoop应用于数据的存储、分析, 能提高数据存储和分析的效率. 传统的数据存储、分析, 将一张表存储在一台机器上, 一条SELECT语句被一台机器执行, 影响效率的关键因素是单机的性能. HDFS将一个大文件分块存储在集群的各个节点上, 完成数据存储的分解; map/reduce将一个数据分析任务下发到各个节点上, 完成任务的分解. 各个节点并行的进行数据存储和分析, 最终通过归约各个节点的结果完成任务. Hadoop系统强大的数据处理能力, 鲜明的云计算的特征, 已经被业界广泛的接受, 并被应用于生产.

2.1 HDFS 的数据存储

数据文件有效记录1221215条, 大小为457MB, 每一行是数据库的一条记录, 将数据备份至HDFS文件系统. 图2从Web端展示了HDFS文件系统, 它类似于Unix文件系统, 有访问控制、属主、属组等.

Go to parent directory

Name	Type	Size	Replication	Block Size	Modification Time	Permission	Owner	Group
acctinfo.out	dir				2012-05-26 11:21	rwxr-xr-x	hadoop	supergroup
acctinfo.txt	file	456.65 MB	1	64 MB	2012-05-26 10:43	rw-r--r--	hadoop	supergroup

图2 HDFS分布式文件系统

Block Size属性表示HDFS的块大小, 由于Hadoop的优点是大数据处理, 当处理TB、PB级别数据的时候, 可以根据需要调节为256M、512M等. 它的配置直接影响了分配给map阶段每个Split的大小, 从而影响map过程的效率. 配置Block Size属性, 理论上应从以下几点考虑:

第一, GB、TB级别的大数据处理, 分片小了, 增加了map过程的开销.

第二, 分片大了, 负载均衡的效果不理想.

第三, NameNode的内存压力, 随着块大小的增加, NameNode内存压力变小.

当数据量过大的时候, 或者集群规模不大, 可选择大一点的分片. 3个节点的机群, 要处理1GB的数据, 不妨设置分片128M或者256M. 实际应用中, 分片大小的选取, 要从数据量和节点个数考虑. 另外, 处理小数据量时也不必设置小于64M(比如32M), 就如同

Windows 下, 不必将块大小设置为小于 512byte.

Replication 属性配置了数据块的复本数, 体现了 HDFS 容错机制, 一旦有节点意外停机, 用户可以从其他节点读取数据块. 实验证明, 随着 Replication 值增加, 实际写入的数据量是原数据量的 Replication 倍, 导致 HDFS 的写速度降低. 因此, 在不同场合, 应对 Replication 属性有所取舍.

2.2 map/reduce 程序设计

map/reduce 程序设计的关键, 是构建与应用相关的 key/value 键值, 也就是 map 函数输出的中间结果. 另外, 中间结果的优化也影响了数据分析的效率.

map 函数对源数据提取需要分析的字段, 过滤无效数据, 进行预处理后, 交由 reduce 进行规约. map 函数中间输出(key, V), 被写入 context. reduce 对 map 的输出 context 的内容进行归类, 同一 key 值的(key, V)放在一个列表中生成(key, list(V)), 然后对(key, list(V))进行规约 V_{out}=result(list(V)), 任务结果为(key, V_{out}). map/reduce 伪码如下:

Step1: map 函数	1. read line 2. line=substring(line) 3. while line.hasmore ;do key=whichlevel() done 4. context.write(key,value)
Step2: combiner 函数	Call reduce()
Step3: reduce 函数	1. input(key，list(value)) 2. OBJECT V_{out}; 3. foreach V in list(value)；do V_{out} +=result(value) done 4. context.write(key, V_{out})

为了优化 map/reduce 程序, 对 map 函数的输出在当前节点进行中间处理, 然后交予 reduce 函数, 这种方式称为 Hadoop 数据局部性. Hadoop 提供了一个优化数据局部性的函数 combiner, 实质上 combiner 也是一个 reduce 规约函数, 它不影响 reduce 的处理结果. map 函数完成, 在当前节点进行 combiner, 实质上是减少了 map 函数和 reduce 函数之间的数据传输, 因此提高了效率. 为此, 文献[5]提出了一种改进的数据局部性算法. 本实验中, 用 reduce 函数作为 combiner 函数.

3 实验结果

3.1 实验平台

图 3, 在配置 Pentium Dual-core CPU T4300 2.1GHz, 4G DDR3 1600mHz 内存, 250G WDC ATA 硬盘, 32bit-Windows7 sp1 的机器上, 运行三台虚拟机 fedora101、fedora102、fedora103, 操作系统是 32 位 Fedora Core 15. fedora101 是 Master 节点, fedora101、fedora102、fedora103 是 Slave 节点. 由于实验资源有限, Master 节点 fedora101, 也运行了 Slave 节点的实例. 但是, 在实际生产上, 尽量避免这样做, 不仅 Master 与 Slave 要物理上隔离, SecondaryNameNode 也要与 NameNode 隔离.

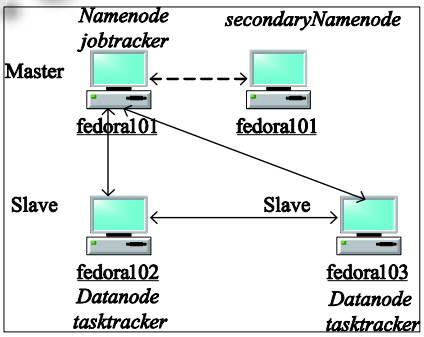

图 3 实验环境系统架构

3.2 实验分析及优化

Hadoop 的性能表现在两个方面: 一是 HDFS 大数据存储的效率; 二是 map/reduce 程序大数据分析的效率. 以此将应用分为载入数据和分析数据两个阶段, 这两个阶段的效率直接影响了用户感受. 对于云计算来说, 影响这两个效率的瓶颈是网络传输效率和磁盘 I/O, 在网络环境理想的 LAN 中, 暂不予考虑网络传输效率.

HDFS 的 Replication 属性很大程度上影响了数据载入效率, 是磁盘 I/O 对性能的影响. 它确保了在发生数据块、磁盘、机器故障后数据不丢失. 当系统发现一个错误的块, 会从其他节点读取另一个复本, 保证复本数回到 Replication 的值. 当 Replication 过大时, 会影响写数据的效率, 因为数据量比原数据大了 Replication 倍, 并且在 WAN 中应考虑网络数据传输的开销. 当应用仅仅是为了分析数据时, 可以将 Replication 设置为 1. 表 2 列出了不同 Replication 值对 HDFS 效率的影响.

当数据达到 GB 规模, 内存已不能满足需求, 必定

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.