第33卷 第22期 计 算 机 工 程 2007年11月
Vol.33 No.22 Computer Engineering November 2007
·软件技术与数据库· 文章编号：1000—3428(2007)22—0083—03 文献标识码：A 中图分类号：TP391

基于MapReduce的分布式光线跟踪的设计与实现

郑欣杰，朱程荣，熊齐邦
(同济大学计算机科学与技术系，上海 200331)

摘 要：提出了基于MapReduce架构实现分布式光线跟踪渲染的方案。该方案基于Hadoop实现，利用MapReduce架构简化了分布式程序设计。使用分布式计算进行光线跟踪，充分利用了现有低端硬件设备的处理能力。实验表明，该方案通过并行计算大大加快了渲染速度。
关键词：MapReduce；Hadoop；分布式计算；光线跟踪

Design and Implementation of Distributed Ray Tracing Using MapReduce

ZHENG Xin-jie, ZHU Cheng-rong, XIONG Qi-bang
(Dept. of Computer Science and Technology, Tongji University, Shanghai 200331)

【Abstract】 An approach based on Hadoop is presented to speed up ray tracing by utilizing MapReduce. It improves the rendering speed considerably by employing distributed computing, and simplifies the design of the distributed application by using MapReduce paradigm. Hadoop implementation provides a convenient and flexible framework for distributed computing on a cluster of commodity machines. The design and implementation is presented. The results and some related discussions are also given.
【Key words】 MapReduce; Hadoop; distributed computing; ray tracing

随着计算量的激增，分布式计算应运而生。由于分布式计算将计算任务分配到各台机器上并行处理，整体运算速度得到很大提高。但该方法在实际操作中，还存在诸多问题，例如如何有效地分割输入的数据和合理分配计算任务等。因此，如果有一种通用的分布式计算模型，由底层封装任务分配、并行处理、容错支持等细节，而由用户集中精力解决分布式计算的任务表述，就能极大地简化分布式程序的设计。而由Google实验室提出的MapReduce[1]正是这样一种模型。同时，Apache开源社区的Hadoop[2]为该模型提供了Java实现，即也为分布式计算提供了平台。

MapReduce解决的主要是海量数据的处理，但其也可用于其他一般性的分布式计算。光线跟踪是一种能真实模拟现实的3D图像渲染算法，它能实现完美的画质，但需要大量的运算。raja(ray-tracer in Java)[3]是SourceForge上一个开源的光线跟踪实现。基于raja所进行的分布式改进的工作很多，例如Fraja，Straja等，但这些改进都是从头进行的，包括从头设计分布式实现细节等。鉴于此，本文试图利用MapReduce模型对raja进行分布式改进。

1 MapReduce概述

MapReduce是Google实验室提出的一种新的分布式程序设计模型，用于在机群上对海量数据进行并行处理。Map和Reduce是该模型中的两大基本操作，可以由用户指定。利用该模型编写的程序具有分布式特性，可以在分布式机群上进行并行处理。MapReduce的输入是一系列键值对，输出也由键值对构成。用户将需要进行的处理工作分解为两个功能块：Map和Reduce。

用户通过指定的Map函数对输入进行处理，产生大量的中间键值对。MapReduce底层实现将所有具有相同键值I的键值对集合到一起，传递给Reduce函数。

用户通过指定的Reduce函数得到了一系列以I为键的键值对。Reduce函数由这些键值对产生一个新的键值对集合。通常这个集合要比输入小一些，而且，一次Reduce调用只产生一个输出，或者不产生输出。

Google使用C++实现了MapReduce，并已在Google内部得到了广泛的运用。Hadoop是Lucene Apache开发出的MapReduce的开源实现，它完全使用Java编写。

Hadoop为在大量低端机器组成的机群上进行分布式计算提供了一个方便的平台。它使用SSH对远程主机管理脚本提供支持，还可通过配置rsync来简化安装更新过程。运行过程中Hadoop框架自动为远程主机配置分布式应用，此外，Hadoop还带有jetty服务器，可在运行时从Web页面监视每台机器上的任务运行情况，这些都为分布式应用的部署及管理提供了极大的便利[4]。

2 实验环境

表1 分布式渲染实验环境

编号	硬件配置	操作系统
1	Intel Celeron 2.00GHz, 1 024MB memory	Ubuntu 6.06 dapper
2	AMD Sempron 2200+ 1.50GHz, 1 024MB memory	Ubuntu 5.06 hoary
3	Intel Pentium4 1 300MHz, 384MB memory	FC3
4	Intel Pentium 3 1 000 MHz, 512MB memory	RH9
5	Intel Pentium 3 800 MHz, 512MB memory	Magic Linux 1.2
6	Intel Pentium 3 733 MHz, 256MB memory	Ubuntu 6.06 dapper
7	Intel Pentium 3 866 MHz, 128MB memory	RH8

本文利用实验室7台机器进行分布式计算，7台机器的

作者简介：郑欣杰(1982－)，男，硕士研究生，主研方向：信息安全及容错计算；朱程荣，副教授；熊齐邦，教授
收稿日期：2006-12-09 **E-mail：**raul07tj@163.com

配置不一，安装系统为 Linux 不同版本，如表 1 所示。此外，使用的 Hadoop 版本是 0.4.0，raja 版本是 0.4.0-pre4。

3 设计实现

要进行分布式光线跟踪，首先要将构成图片的所有像素点进行分割。而后将分割信息通过输入传递给并行的 Map 操作，各 Map 操作对各自的像素块进行独立渲染，最后将生成的像素颜色连同像素坐标传递给主程序，由主程序根据这些数据对图像进行生成。

根据 MapReduce 模型，在分布式渲染前首先必须确定 Map 任务总数。在配置文件中设定单机执行的 Map 数，在程序中进行读取，并乘以机器数就可算出所需 Map 总数。然后将图像宽度按 Map 数进行等分，实际产生的 Map 数根据图像分割块数确定。每次 Map 操作都对一块图像进行渲染运算。为此，为每个 Map 任务创建一个输入文件，将渲染起始宽度与终止宽度写入文件，作为 Map 操作的输入。此外，对所有渲染都需要的其余参数，简单常量(如反锯齿级别等)可通过 Hadoop 中的 JobConf 配置直接传递，复杂的如 Camera 等类实例参数，可将其持久化后写入分布式文件系统中，由所有 Map 操作共享。最后必须为 JobConf 设置各类配置信息，然后启动任务：

```
    jobConf.setInputFormat(SequenceFileInputFormat.class);//设置输
//入文件格式
    jobConf.setInputKeyClass(IntWritable.class);//输入中键代表渲染
//起始宽度
    jobConf.setInputValueClass(IntWritable.class);//值代表终止宽度
    jobConf.setMapperClass(RayTracerMapper.class);//设置 Map 操作
//实现类
    jobConf.setReducerClass(RayTracerReducer.class);//设置 Reduce
//操作实现
    jobConf.setNumMapTasks(numMaps); //设置 Map 操作数
    jobConf.setNumReduceTasks(1); //设置 Reduce 操作数
    …
    JobClient.runJob(jobConf); //启动 MapReduce 操作
    在 Mapper 中，首先必须获得各项参数：
    // Map 操作前通过 configure 从 JobConf 中获得简单参数
    public void configure(JobConf job) {
    antialiasLevel = job.getInt("RayTracer.renderer.antialiasLevel", 0);
//反锯齿度
    …}
    //map 操作中的初始化
    public void map(WritableComparable key, Writable value,
OutputCollector output, Reporter reporter) throws IOException {
    //从输入中取得任务始末坐标
    int fromWidth = ((IntWritable) key).get();
        int toWidth = ((IntWritable) value).get();
    …
        SequenceFile.Reader reader = new SequenceFile.Reader(fileSys,
jobArgs, job); // 用于从分布式文件系统中取得 camera 等复杂参数
        IntWritable jobKey = new IntWritable();
        JobArgsWritable ja = new JobArgsWritable(); //自定义类，传
//递复杂参数
        reader.next(jobKey, ja);
        reader.close();
        //获得复杂参数
    Camera camera = ja.camera;
    …
```

取得所有渲染参数后，进行光线跟踪渲染计算。基于图像渲染自身的特点，为提高效率，可将单个 Map 操作的输出直接写至分布式文件系统的不同文件，每个文件都采用特定的命名规则，使其与该 Map 操作所渲染的区域相对应，这样在最后的主程序中就容易从分布式文件系统中取得各个区域的渲染结果。程序中省略了 Reduce 操作。RayTracerReducer 类实际上实现的是一个空操作。此外，由于 Map 操作产生的结果是像素信息，在中间过程也没有必要进行合并，同样省略了 Combiner 的设置。

在并行进行 Map 操作时，Hadoop 默认的进度显示是根据输入文件被处理的程度。由于本文仅通过输入文件传递渲染起点以及终点，默认的进度不能很好地表示实际处理进度，因此可通过修改 Hadoop 中 Task 类的 getReporter 方法，为其返回的 Reporter 添加可设置进度的 progress 方法，同时禁用 MapTask 类中读输入数据时的进度汇报，最后在 Map 操作代码中对 progress 加以调用即可。

当所有任务计算完毕后，主程序从分布式文件系统中取回所有渲染结果，并据此生成最后的渲染图像。

4 结果与分析

本文对 raja 自带的 PenInWater 场景进行了实验，实验参数如表 2 所示。

表 2 PenInWater 实验参数

参数名称	参数值
分辨率	640×480
递归深度	20
反锯齿	4

在未采用分布式渲染的情况下使用 raja 原程序时，本文选取了两台机器做渲染实验，结果如表 3 所示。

表 3 PenInWater 单机渲染实验结果

机器编号	渲染时间
1	1h 32m5s 863ms
7	2h 42m25s 316ms

在 7 台机器进行并行渲染的情况下，通过改变并行 Map 数量，得到如表 4 所示的结果。

表 4 PenInWater 分布式渲染实验结果

单台运行 Map 数	总渲染时间
1	1h 2m 21s 268ms
2	33m 8s 211ms
3	21m 27s 769ms
4	19m 48s 924ms
5	19m 50s 495ms
10	18m 27s 281ms
15	17m 32s 821ms
20	17m 42s 46ms

由表 4 的数据可知，尽管 7 台机器配置不一，最快的并行渲染速度还是比 1 号机(处理能力较强)单机渲染速度要提高了约 5.25 倍。此外，随着 Map 数量的增加，总渲染时间明显缩短。这是因为随着 Map 数的增加，在不同处理能力的机器上的任务分配日益均衡，从更大程度上开发了并行度。从 4 号机上获得的 CPU 利用率图像如图 1～图 3 所示。

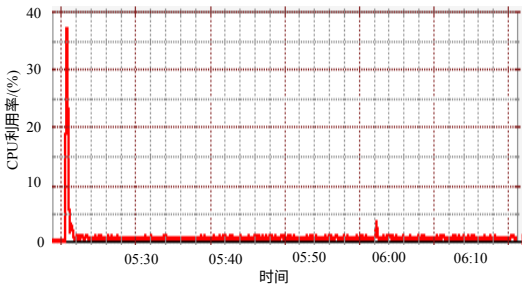


图 1 单机运行 1 个 Map 时 4 号机 CPU 的利用率

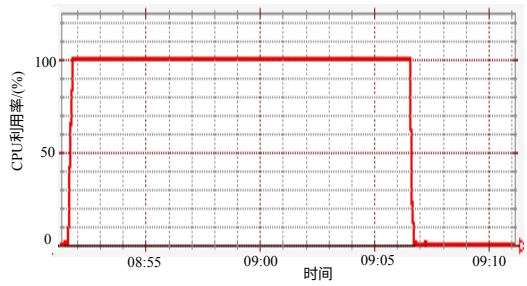


图 2　单机运行 5 个 Map 时 4 号机 CPU 的利用率

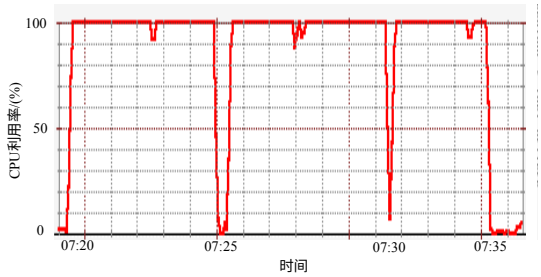


图 3　单机运行 15 个 Map 时 4 号机 CPU 的利用率

在配置为单机运行 1 个 Map 操作时，由于 Hadoop 默认每台机器可以运行 2 个 Map 操作，因此任务分配极不平衡，4 号机没有获得任何任务，CPU 利用率的一次上扬仅仅是因为任务启动过程中的一系列开销。当每台机器配置为运行 5 个 Map 操作时， 4 号机的 CPU 利用率得到极大提高。在配置为运行 15 个 Map 操作时，4 号机的运算能力已趋于饱和。

在实际应用中，可尝试运用 MapReduce 的 Backup 概念来提高运算速度。Hadoop 实现也提供了这一功能(称为 Speculative Execution)。但是 Hadoop 的分布式文件系统不能处理多任务对同一文件的同时写，为了开启这一功能，可对代码进行修改，使 Map 操作通过 Collecotor 传出单个像素的计算结果，由 Reduce 操作进行恒等变换，再将其直接拷贝至输出，这无形中增加了开销。尽管如此，Backup 任务并没有带来明显的效果，特别在提高单机运行的 Map 数以后，各机器的处理能力已都趋于饱和，几乎已经没有了进行 Backup 运行的空间。此外，Hadoop 还提供了较好的容错性能，若单机运行失效，10 分钟之内未能产生应答，Hadoop 自动将该机上的任务重新安排运行。这一点也在实验中得到了验证。

5　结束语

本文提出了基于 Hadoop、采用 MapReduce 技术加速光线跟踪渲染的方法。Hadoop 的实现则在低端计算机组成的机群上为分布式计算提供了方便灵活的平台。

参考文献

1 Dean J, Ghemawat S. MapReduce: Simplifed Data Processing on Large Clusters[C]//Proc. of the 6th Symposium on Operating System Design and Implementation, San Francisco. 2004.

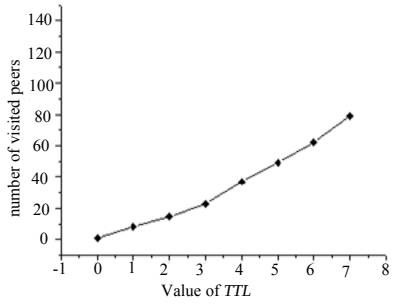
2 Cutting D. Scalable Computing with MapReduce[C]//Proc. of O'Reilly Open Source Convention, Poland. 2005.

3 The Raja Project[EB/OL]. (2003-05). http://raja.sourceforge.net.

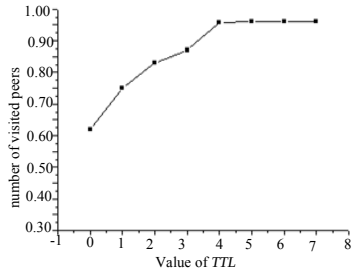
4 Apache Lucene Hadoop[EB/OL]. (2006-11). http://lucene.apache.org/hadoop.

(上接第 72 页)

中，查询结果的质量按照 Q=|Sreal∩Sapprox.|/|Sreal|定义，由此得到的是查询数据结果集合中正确结果的比例。



(a)P2P 中被访问结点的数量与 *TTL* 值的关系



(b)实验结果的查询质量与 *TTL* 值的关系

图 4　成本与集群规模大小的关系

图 4(a)显示：随着 *TTL* 值的增大，系统中被访问结点的数量不断增加。这个现象是合乎逻辑的，因为访问的集群越多，越有利于查询到包含相关数据的结点；图 4(b)显示：随着 *TTL* 值的增大，实验结果的质量在进一步地改善；但是当 *TTL* 达到一定的值之后，查询质量趋于稳定，不再增加。因此，在 *TTL* 值相对小的情况下，范围查询策略可以得到较高的数据查询结果。

4　结论

对比较复杂的多维数据查询请求的研究在 P2P 系统中很重要。本文对 EIR-tree 综合架构中的多维数据范围查询进行了探讨，对推动相关的理论与应用研究意义重大。

参考文献

1 Li Xiuqi, Wu Jie. Cluster-based Intelligent Searching in Unstructured Peer-to-Peer Networks[C]//Proceedings of the 25th IEEE International Conference on Distributed Computing Systems, 2005.

2 Gu Tao, Tan E, Hung Keng Pung. A Peer-to-Peer Architecture for Context Lookup[C]//Proceedings of the 2nd Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services. 2005.

3 曾碧卿. 分布式计算中并行 I/O 调度策略研究[D]. 广州: 华南师范大学, 2005.

4 Risse T. A Self-organizing Data Store for Large Scale Distributed Infrastructures[C]//Proceedings of the 21th International Conference on Data Engineering, 2005.

5 Ala'a Qasim Al-Namiy, Faris S. Majeed. Improving Query Answering in Peer-to-Peer Data Searching[C]//Proceedings of the 19th International Conference on Advanced Information Networking and Applications. 2005.