一种基于分布式平台 Hadoop 的矩阵相乘算法①

冯 健, 倪 明, 赵建波

(中国电子科技集团公司 第三十二研究所, 上海 200233)

摘 要: 为了解决单节点上大矩阵相乘运算量过大的问题, 提出了一种基于分布式平台 Hadoop 的矩阵相乘算法. 算法采用了字节文件作为输入以及最优化的分片方式, 去掉了不必要的 Reduce 过程, 极大地减少了输入数据量, 拥有简洁的算法流程和可拓展性. 实验结果表明该算法很好地解决了矩阵相乘运算量过大的问题, 当输入数据量接近集群最佳负载量的情况下取得了很好的加速比.

关键词: Hadoop 平台; 分布式矩阵乘法; 输入格式; 分片方式

Algorithm of Distributed Matrix Multiplication Based on Hadoop

FENG Jian, NI Ming, ZHAO Jian-Bo

(The 32nd Research Institute of China Electronics Technology Group Corporation, Shanghai 200233, China)

Abstract: In order to release the burden of large matrix multiplication on a single node, a distributed matrix multiplication algorithm based on Hadoop is proposed. The algorithm uses the input file with binary format, applys optimal split, removes the unnecessary Reduce phase. The algorithm can greatly reduce the amount of input data, has simple algorithm flow and good scalability. Experiment results demonstrate that the algorithm greatly reduces the computation of matrix multiplication and achieves good speedup when the amount of input data is around the optimum loading of the cluster.

Key words: Hadoop platform; distributed matrix multiplication; input format; split

传统的矩阵运算并行化研究主要通过改进计算机的体来系结构来充分挖掘高性能计算机的并行能力. 但这样会使得计算机的功耗、复杂度极大增加, 不一定能带来和投入相应比例的性能提升. 在大数据面前, 单台机器的计算能力太过弱小, 适当地采用分布式计算成为了一种值得考虑的方案.

分布式计算相对于传统的高性能计算机具有许多优势: (1)成本低. 如今的高性能计算机基本上都是靠政府的财力支持才能运转的, 而分布式集群使用普通 PC 就可以构建. (2)容错能力强. 分布式集群由于物理上的分散造成了单个节点的失败不会影响整个集群的运转, 通过适当的软件方法可以迅速地恢复正常状态. (3)面向普通大众. 高性能计算机只在实验室中使用, 而分布式计算则面向所有可以联网的人.

Hadoop 是一款开源的分布式系统基础架构, 它提供了一种大容量、可拓展、高可靠的分布式存储系统 HDFS(Hadoop Distributed File System)和一种高性能的分布式计算平台 MapReduce[1], 使用户不必过多了解底层细节就可以开发出高性能的分布式应用.

分布式矩阵乘法的研究普遍使用了矩阵分块法. [2]采用了 Server-Client 机制, Server 根据集群内的 Client 数量把左矩阵按行平均分块后给每个 Client 发送一个子块和一份右矩阵, Client 完成计算后把结果送回 Server. 在 Hadoop 中若采用上述分块机制, 会因为右矩阵的分布式存储特性增大右矩阵到 Client 的网络传输时间. [3]把左右矩阵分解成了矩阵元素相乘的形式, 相当于把每个矩阵元素都划分为一个子矩阵, 这样的划分太细, 增加了 JobClient(任务客户端)的计算负担. [4]提到了输入数据分片要考虑计算本地化, 却没有说明如何通过分片实现计算本地化.

① 基金项目:国家“863”计划基金重点项目 (2009AA012201);上海市科委科技攻关基金重大项目(08dz501600)
收稿时间:2013-05-21;收到修改稿时间:2013-06-24

本文紧密结合 Hadoop 平台的特性, 提出了一种新的分布式矩阵相乘算法, 算法采用字节文件输入、优化的分片策略且具有简洁的算法流程和可拓展性, 实验证明了这种算法具有较好的性能.

1 相关概念

1.1 HDFS

HDFS 是 Hadoop 平台专有的分布式文件系统, HDFS 的存储空间被划分为很多个 Block 分布在集群节点中. Block 是节点上的一个固定大小的存储块, 是 HDFS 的基本读写单位.

1.2 MapReduce

MapReduce 是 Hadoop 平台上的分布式计算框架, 图 1 是 MapReduce 的执行流程, 主要包括了 Map 和 Reduce 两个过程.

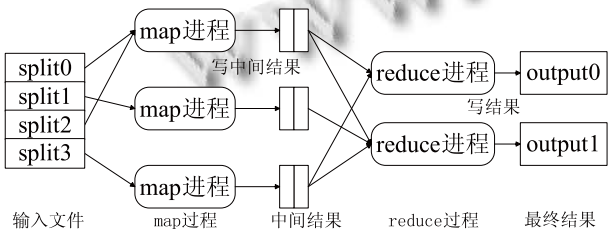

图 1　MapReduce 框架流程

首先在 JobClient 节点上对输入数据进行分片, 图 1 中的输入文件被分为 4 个片. 接着 MapReduce 会在集群内较空闲的节点上开启 Map 进程, Map 进程会读取距离最近的片进行处理, 把生成的中间结果暂存在本地, 经过排序、分组等操作后传送给 Reduce 进程. Reduce 进程进一步处理后生成最终结果[5].

1.3 分片

片(split)是 MapReduce 中重要的逻辑概念, 简单地说一个片就是一个 Map 进程要处理的所有数据的描述, 包括数据在输入文件中的偏移和数据的大小. 分片就是把输入文件从逻辑上划分为多个片的过程. 分片以及任务提交都是通过 JobClient 节点完成的.

1.4 矩阵分块乘法

根据线性代数中分块矩阵的思想, 任何行数和列数较高的矩阵, 在运算时可以通过矩阵分块把大矩阵的运算化为小矩阵的运算. 矩阵分块乘法如下定义[6]:

设 A 为 $m\times l$ 矩阵, B 为 $l\times n$ 矩阵, 分块成

$$A=\begin{bmatrix} A_{11}\cdots A_{1t} \\ \vdots \quad \vdots \\ A_{s1}\cdots A_{st} \end{bmatrix}\ (1),\ B=\begin{bmatrix} B_{11}\cdots B_{1r} \\ \vdots \quad \vdots \\ B_{t1}\cdots B_{tr} \end{bmatrix} \tag{1}$$

那么

$$C=AB=\begin{bmatrix} C_{11}\cdots C_{1r} \\ \vdots \quad \vdots \\ C_{s1}\cdots C_{sr} \end{bmatrix} \tag{2}$$

其中

$$C_{ij}=\sum_{k=1}^{t}A_{ik}B_{kj}(i=1,...,s;\ j=1,...,r) \tag{3}$$

2 算法分析

2.1 算法流程

图 2 给出了算法的流程. 输入文件 *x1_y1_x2_y2.raw* 中的矩阵数据在 JobClient 上被分片, 之后该文件上传到 HDFS 中分布式保存. Map 进程会从其它节点或者本地读取片的内容至内存, 采用串行算法执行子矩阵相乘后将结果写入输出文件. 下面详细分析本算法的核心部分.

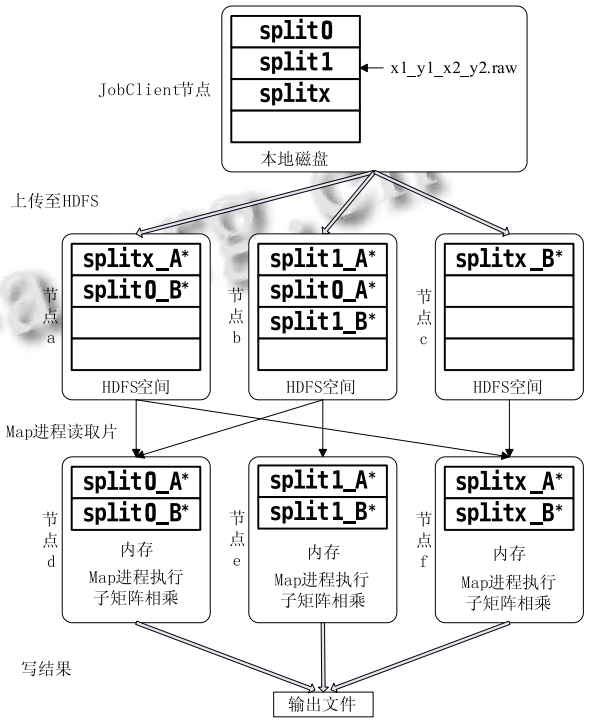

图 2　算法流程

2.2 输入格式优化

不失一般性, 设输入矩阵 A、B 的元素为 Java 语

言中 8 字节长的浮点型.

MapReduce 的默认输入格式是文本文件. 图 3 的文本文件中每个矩阵元素存储为字符串, 串的长度为有效位数, 元素间用空格隔开. 由于矩阵元素有效位数不相同, 导致了每个元素的串长度不相等, 这给分片优化造成了不便, 也使得输入文件较大. [3,4]中都采用了文本文件保存矩阵数据.

$$a_{11}\cdots a_{1l}$$
$$\cdots\cdots$$
$$a_{m1}\cdots a_{ml}$$
$$b_{11}\cdots b_{l1}$$
$$\cdots\cdots$$
$$b_{1m}\cdots b_{lm}$$

图 3 文本文件的内容

本算法采用了字节文件 *x*1_*y*1_*x*2_*y*2.*raw* 来存储矩阵数据. 图 4 所示, 在文件中矩阵 A 的元素按行主序依次存放, B 的元素按列主序紧接着 A 的最后一个元素依次存放, 每个矩阵元素都为 8 字节长度. 同时把矩阵 A、B 的行列数 *x*1、*y*1、*x*2、*y*2 写入文件名中.

$$a_{11}\cdots a_{1l}a_{21}\cdots a_{(m-1)l}a_{m1}\cdots a_{ml}b_{11}\cdots b_{l1}b_{12}\cdots b_{l(n-1)}b_{1n}\cdots b_{ln}$$

图 4 字节文件 *x*1_*y*1_*x*2_*y*2.*raw* 的内容

字节文件相比于文本文件可以非常方便地计算出 A、B 中任意元素在文件中的偏移. 例如 B(5,10)的偏移可以表示为(*x*1**y*1+9**x*2+4)*sizeof(double), 这种特性便于后续的分片优化. 而文本文件需要解析出 A(1,1)到 B(6,10)所有元素的串长度才可以得出偏移, 不仅方法复杂而且会增加集群负载.

此外在输入矩阵阶次相同时, 字节文件的大小比文本文件小 30%以上, 不仅节省 HDFS 空间而且减少了 Map 进程读取数据的网络延时. 表 1 显示在不同阶次下两种格式矩阵数据的大小比较, 假定文本文件中每个元素是 3 位精度 6 位有效数字的浮点型, 单位 MB.

表 1 两种输入格式数据量的比较

阶数(方阵)	1000	2000	4000
字节文件	7.629	30.518	122.070
文本文件	11.444	45.777	183.105

2.3 分片优化

矩阵分块采用图 5 中的模式, A 分块为多行一列, B 分块为一行多列. 优化的分片方式基于矩阵 A、B 的分块模式, 把 A 的一个行子矩阵 A*和 B 的一个列子矩阵 B*划为一个片.

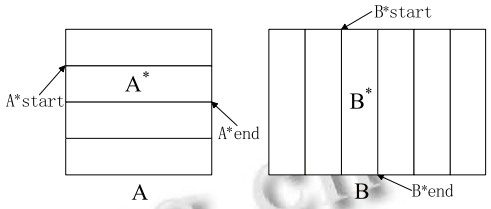

图 5 矩阵 A、B 的分块(分片)模式

片大小的讨论. 片是对 HDFS 中数据的划分, Block 是对 HDFS 存储空间的划分. 片过大会使得 Map 进程读取片的内容时访问多个 Block, 如果被访问 Block 不在本地, 无疑会加大 Map 进程的延时. 片太小, 一次 Block 读取的有效数据太少导致读取效率太低. 所以最高效的片大小等于 Block 大小, 默认的 Block 大小是 64M[7].

由于矩阵 B 是列主序存储的, 根据上述分片方式并结合图 5 可以看出, 片的内容 A*和 B*各自在输入文件内占据一块连续区域. 采用 64M 的片大小后, 一个片最多分布在 4 个 Block 中, 也就是说 Map 进程在读取一个片时, 最坏情况下有 4 次非本地访问, 保证了网络延时的上界.

这样分片有以下好处: (1)一个片由两个子矩阵组成, 分块和分片一次性完成, 减少了延时. 如果采用文件文件输入, 必须先分片后分块. (2)可以使 Map 进程之间的执行相互独立. 子矩阵 A*和 B*相乘的结果刚好是矩阵 C 的一个子矩阵, 把 A*和 B*划分为一个片后, A*和 B*相乘刚好由一个 Map 进程来完成, 使得 Map 进程之间相互无关.

2.4 自定义分片类

Hadoop 内置的分片方法无法满足上述分片需求, 需要自定义 MatrixInputFormat 类来完成分片, 其主要任务是计算片的大小和片内容在输入文件中的偏移量. 过程如下:

(1) 计算每个子矩阵占的行数和列数.

$$\begin{cases} SizeA\times RowsA^{*}/x1+SizeB\times ColsB^{*}/y2=64M \\ RowsA^{*}/ColsB^{*}=SizeA/SizeB \end{cases} \quad (4)$$

其中 *SizeA*、*SizeB* 分别是矩阵 A、B 的大小, *x*1、*y*1 分别是 A 的行数和 B 的列数, 64M 是片大小. 解出的 *RowsA**不一定能被 *x*1 整除, *ColsB**也不一定能被 *y*2 整除, 无法保证子矩阵大小相等, 需要进一步调整.

$$\begin{cases} mod = x1\%RowsA^* ,\ div = x1 / RowsA^* \\ if\ (mod < RowsA^* / 2)\ then\ \{ \\ \quad actualRows = RowsA^* + mod / div \\ \quad mdl = mod\%div\ \} \end{cases} \tag{5}$$

$$\begin{cases} mod = y2\%ColsB^* ,\ div = y2 / ColsB^* \\ if\ (mod < ColsB^* / 2)\ then\ \{ \\ \quad actualCols = ColsB^* + mod / div \\ \quad mdl = mod\%div\} \end{cases} \tag{6}$$

公式(5)对 *RowsA**进行调整, 目的是把余下的 *mod* 行平分到每个子矩阵, 使子矩阵大小尽量相等. 如果式(5)中判断条件为假, 则矩阵 A 分块为(*div*+1)个子矩阵, 前 *div* 个子矩阵都是 *RowsA**行, 最后一个子矩阵有 *mod* 行, *RowsA**和 *mod* 之间相差不超过 *RowsA**/2; 如果条件为真, 则 A 分块为 *div* 个子矩阵, 前 *mdl* 个子矩阵都是(*actualRows*+1)行, 后(*div*-*mdl*)个子矩阵都有 *actualRows* 行,子矩阵最大相差 1 行. 公式(6)是对 *ColsB**的调整, 同理不再赘述.

上述调整保证了片大小平均约为 64M, 平衡了集群内节点间的负载量, 片的两部分 A*和 B*也能够保持合理的大小比例. 例如取 3000 阶的矩阵 A 和 4000 阶的矩阵 B, 解得 *RowsA**=829, *ColsB**=1474, 按(5)式调整后矩阵 A 分为 4 个子矩阵, 前 3 个子矩阵有 829 行, 后一个有 513 行; 矩阵 B 分为 3 个子矩阵, 前 2 个子矩阵有 1474 列, 后一个有 1052 列.

(2) 确定片内容的偏移

描述一个片的内容需要 4 个变量[8], 分别指定 A*的起始和结尾, B*的起始和结尾, 如图 5 所示. 为了便于描述, 假定上一步计算得出的 A 的所有子矩阵都是 *RowsA**行, B 的所有子矩阵都是 *ColsB**列.

$$\begin{cases} A^*start = IndexOfA^* \times RowsA^* \times y1 \times sizeof\ (\text{double}) \\ A^*end = A^*start + RowsA^* \times y1 \times sizeof\ (\text{double}) \\ B^*start = SizeA + IndexOfB^* \times ColsB^* \times x2 \times sizeof\ (\text{double}) \\ B^*end = B^*start + ColsB^* \times x2 \times sizeof\ (\text{double}) \end{cases} \tag{7}$$

式(7)计算出了片的内容在输入文件中的偏移量. *IndexOfA**、*IndexOfB**是任意子矩阵 A*、B*的子矩阵序号, 例如图 4 中这两个变量的值分别是 1 和 2.

自定义分片类对性能影响的分析. 自定义分片操作比Hadoop内置的分片操作要复杂, 因为一个片需要更多的信息来进行描述而且需要更多的计算量. 但自定义分片类并不会成为 JobClient 性能的瓶颈: 第一, MatrixInputFormat 类中的分片操作只是对输入文件进行逻辑分片没有对 HDFS 中输入文件的读取或写入这样耗时的 IO 操作. 第二, 即便输入矩阵为 100 万阶, 最终产生的片数目也不超过 25 万, 一次分片仅仅是生成一个片对象和几次运算, 片对象总共占用内存空间也不到 20MB, 和内内置的分片操作具有相同的时空复杂度.

2.5 Map 过程

分片结束后 MapReduce 会为每个片开启一个 Map 进程, Map 进程根据片的偏移信息从文件中把片的内容 A*和 B*读取到本地, 然后执行实际的子矩阵相乘, 如图 2 所示. 由矩阵分块乘法可知, A*和 B*相乘的结果就是结果矩阵 C 的一个子块, Map 进程只要把结果写到输出文件的指定偏移位置就可以.

本算法中一个片的内容最多分布在 4 个 Block 中, Map 进程最多进行 4 次非本地 Block 读取, 理想情况下有 2 次本地读取和 2 次非本地读取, Block 访问次数为常数级别 O(1). 而[3,4]中一个 Map 进程至少需要 O(矩阵阶次)量级的 Block 访问, 相比而言本算法可以极大减少非本地 Block 访问带来的网络延时.

由于分片和分块在 JobClient 上一次性完成, 相对于[3,4]省去了一个耗时的 Reduce 步骤, 而且 Map 进程之间相互无关, 使得本算法显得较为简洁.

一般来说 Map 进程运行的时间应该大于创建 Map 进程所需要的时间, 这样才是有意义的. 这就要求 Map 进程处理的数据不能太少. 本算法中一个分片约为 64M, 相当于两个 2048 阶的方阵相乘, 经过多次测试, 在单节点上的计算耗时约为 210 秒, 这样的计算量对于 Map 进程是完全足够的, 也不至于让 Map 进行运行太久.

3 实验与结果分析

3.1 实验环境

实验环境是拥有 7 个节点的 Hadoop 集群, 其中 1 个作为控制节点和 JobClient, 集群内每个节点配置一个 Map 进程. 节点的软硬件配置为: Ubuntu10.04、Hadoop-1.1.0、Xeon E5620 2.40GHz、内存 4GB、硬盘

500GB、百兆网卡.

根据实验环境和集群大小，输入矩阵依此为1000、1500、2000、2500、3000、3500、4000、4500、5000 和 5500 阶的方阵. 实验结果图中 A 曲线代表本算法, B 代表[4]提出的算法.

3.2 实验结果与分析

图 6 是在单节点上执行时间复杂度为 $O(n^3)$的普通串行矩阵相乘的耗时，单节点是实验集群中的一台 PC. 由于 CPU 运算能力和内存的限制，随着输入矩阵阶次的增加，耗时增加的越来越快.

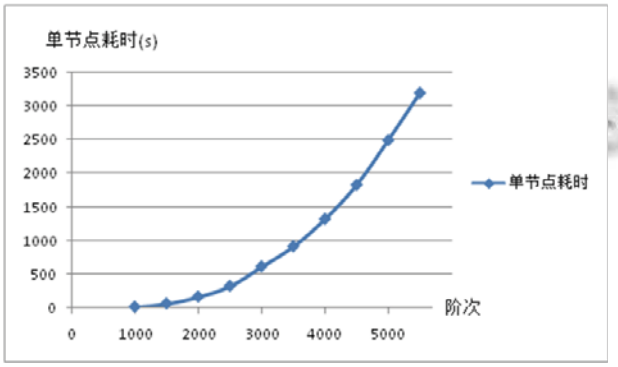

图 6 单节点上串行算法的耗时

图 7 是分布式算法的耗时情况. 对于 A 曲线，当阶次在 2000 至 5000 之间时，单节点耗时是分布式算法耗时的两倍以上. 说明了分布式算法比单节点上的串行算法有很大的速度优势.

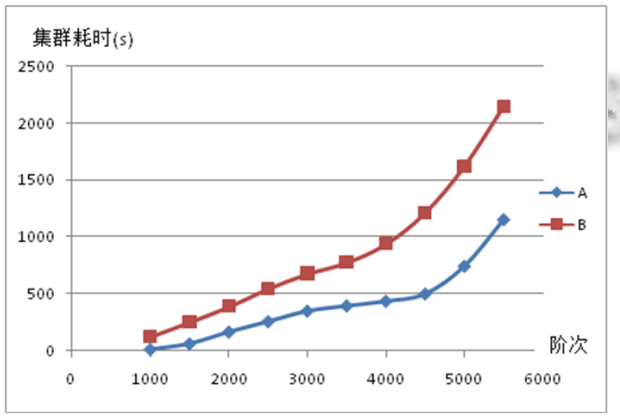

图 7 分布式的算法耗时

A、B 曲线对比表明本算法比[4]中算法要快. [4]由于使用文本文件输入，无法把分块和分片一次性完成，多出了分块过程和一次完整矩阵数据的网络传输，增加了延时.

图 8 是分布式算法相对于单节点上串行算法的加速比，曲线 A、B 具有大致相同的形状. 可以看出阶次超过 5000 后，加速比已经开始下降了，这种变化主要是集群的负载量决定的. 一个 Hadoop 集群内节点数量是有限的，决定了集群的负载量是有上限的，当 Map 进程的数量略小于集群节点数量时，可以取得最好的加速比. 本实验集群有 6 个工作节点，一个 Map 进程处理的数据量约为 64M，则集群的最佳负载量约为 380M 的，相当于 4900 阶的矩阵，所以图 8 中的加速比在接近 5000 阶次时达到最大.

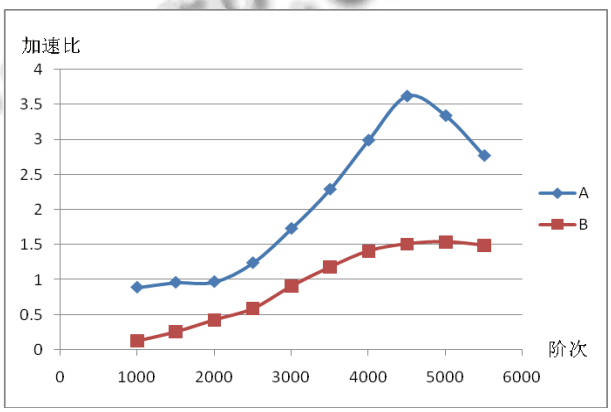

图 8 分布式算法的加速比

在阶次小于 2000 时加速比小于 1，是节点处理的数据量太少所致，因为在 2.5 节中已经说明了集群内一个节点处理的数据不能太少，太少就无法体现分布式系统在大数据运算上的优势.

当阶次处于 2000 和 5000 之间，加速比随着阶次成比例地增加，阶次大于 5000 后，加速比虽然下降了，但依然大于 1. 说明只要输入数据量接近集群的最佳负载量，就可以取得很好的加速比.

本算法可以根据实际情况进一步拓展. 如果节点的计算能力有限，可以采用较小的片；在 Map 进程中可以采用子矩阵相乘并行算法，进一步提升性能；如果选用的片很大，可以在 Map 进程中开启级联任务.

4 结语

本文针对大矩阵相乘运算量大的问题，提出了一种基于 Hadoop 的分布式矩阵相乘算法，取得了很好的加速比. 算法有以下特点:

(1) 采用了字节文件，相比于文本文件输入可以减少 30%以上的存储空间占用.

(2) 自定义分片方式, 保证了片的最佳大小, 使集群中节点的计算量处于较均衡、正常的水平.

(3) 相比于[3,4]以及其他 Hadoop 应用, 省去了 Reduce 过程且 Map 进程相关无关, 算法流程更加简洁.

(4) 算法可适当地进行拓展以适应集群环境并提高性能.

Hadoop 作为刚刚兴起的计算平台, 还有很多特性需要去发掘和研究, 这些特性会有助于提升 Hadoop 平台上算法的表现, 以后会在这方面进一步研究.

参考文献

1 White T. Hadoop: The Definitive Guide. California. yahoopress. 2012: 1–15.

2 Islam S eds. An empirical distributed matrix multiplication algorithm to reduce time complexity. international association of engineers. Proc. of the International Multi Conference of Engineers and Computer Scientists 2009 Vol II.2009. HongKong. International Association of Engineers. 2009. 2171–2173.

3 曾大军.云平台下大型矩阵乘法运算处理方案设计.科技广场,2012,5:1–3.

4 张骏.一种基于 MapReduce 并行框架的大规模矩阵乘法运算的实现.计算机应用与软件,2012,6:1–4.

5 Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters. USENIX. OSDI-2004. San Francisco, California, USA. USENIX. 2004. 137–150.

6 同济大学数学系.同济线性代数.北京:高等教育出版社,2007:47–48.

7 Xie J. Improving Performance of Hadoop Clusters. UMI Dissertation Express. 2011:6–7.

8 王谦.HADOOP 作业启动性能优化实践[硕士学位论文].北京:北京交通大学,2012:9–12.

(上接第 222 页)

3D 渲染 API Stage3D, 以及基于 Stage3D 的多个 3D 引擎, 并基于其中的一个引擎 Flare3D 实现了一个 B/S 架构的、包含编辑和展示功能的 3D 场景组件, 相对于早期的 Flash 平台, 该组件在 3D 场景的实时渲染性能方面得到了极大的提升,可以在普通显卡上支持百万级别的三角形面的实时渲染. 且使用简单, 部署方便, 可广泛应用于增强现实、立体场景、电子商务等高级应用中.

参考文献

1 何伟明,罗立宏.基于 Flash 的三维商品展示.广东工业大学学报(社会科学版),2009,B6:317–318.

2 王立英.基于 Flash 的在线三维商品展示系统的研究与实现[硕士学位论文].成都:电子科技大学,2011.

3 陈忻.FLASH 三维游戏开发探索[硕士学位论文].杭州:浙江大学,2008.

4 陈宁.一种基于 Away3d 的 Web 三维虚拟装配系统.黑龙江科技信息,2012,14:42–43.

5 杨逸文.基于 BlazeDS 的烟草移动服务综合监控系统.计算机应用与软件,2012,29(5):224–227.

6 吕海东,陆永林.基于 Flex 和 BlazeDS 推技术实现 WEB 方式实时监控系统.自动化技术与应用,2010,29(1):34–37.