基于 CUDA 的 k-means 算法并行化研究

刘端阳 郑江帆 沈国江 刘 志
(浙江工业大学计算机科学与技术学院 杭州 310023)

摘 要 k-means 算法在面对大规模数据集时,计算时间将随着数据集的增大而成倍增长。为了提升算法的运算性能,设计了一种基于 CUDA(Compute Unified Device Architecture)编程模型的并化行 k-means 算法,即 GS_k-means 算法。对 k-means 算法进行了并行化分析,在距离计算前,运用全局选择判断数据所属聚簇是否改变,减少冗余计算;在距离计算时,采用通用矩阵乘加速,加快计算速度;在簇中心点更新时,将所有数据按照簇标签排序分组,将组内数据简单相加,减少原子内存操作,从而提高整体性能。使用 KDDCUP99 数据集对改进算法进行实验,结果表明,在保证实验结果的准确性的情况下,改进算法加快了计算速度,与经典的 GPUMiner 算法相比加速比提升 5 倍。

关键词 k-means,CUDA,并行计算,大数据
中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.11.047

Study on Parallel K-means Algorithm Based on CUDA

LIU Duan-yang ZHENG Jiang-fan SHEN Guo-jiang LIU Zhi
(College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China)

Abstract When k-means algorithm is confronted with large dataset, the computation time will grow exponentially with the increase of the dataset. In order to improve the computational performance of this algorithm, this paper designed a parallel k-means algorithm based on CUDA (Compute Unified Device Architecture) programming model, called GS_k-means. According to the parallel analysis of k-means algorithm, the global selection is used to determine whether the cluster of data is changed before distance calculation, thus reducing redundant computation. The calculation speed is accelerated by using the universal matrix multiplication in distance calculation. When the cluster center is updated, all data are grouped by cluster tag, and the data in the group are simply added, thus reducing the atomic memory operation and improving the overall performance. The experimental results on KDDCUP99 dataset show that on the condition of ensuring the accuracy of the experimental results, the improved algorithm the calculation speed and is five times faster than the classical GPUMiner algorithm.

Keywords k-means, CUDA, Parallel computation, Big data

1 引言

聚类技术被运用到许多领域,比如计算机视觉、基因组分析等。聚类的目标是对大量的输入数据进行分组以产生多个集合,分组结果使得集合内的数据相似度高,集合之间的数据相似度低,因此数据点在一个集合内通常有着相似的特征。k-means 算法[1]是最为流行的聚类算法之一,被广泛应用于数据分析、模式识别[2]、图像分析和网络入侵检测[3]等领域。由于 k-means 算法在每次迭代过程中会计算各个数据点到每个聚簇中心的距离,因此当任务的数据量尺寸和维度非常庞大时,该算法将会非常耗时。如果任务的数据总量为 N,簇的数目为 k,算法经过 m 次迭代之后收敛,那么算法在计算距离阶段就要进行 $k*N*m$ 次运算。因此,对于计算密集型的 k-means 算法,运用 CUDA 并行化该算法,能够节省大量的计算时间。CUDA[4]是显卡厂商 NVIDIA 公司推出的一款通用并行计算架构,它使得 GPU 能够解决复杂的计算问题。CUDA 提供了硬件的直接访问接口,并不依赖传统的图形 API 实现 GPU 的访问,降低了图形处理器的编程难度。

目前,国内外已经有很多基于 CUDA 架构的 k-means 算法并行化研究。文献[5]将每个数据到质心距离的计算任务放在 GPU 端执行,而将质心的更新放在 CPU 端执行,这种方法的效率很高而且也很普遍,但是增加了数据在 GPU 端和 CPU 端之间的传输时间。文献[6]根据算法运行时输入数据的不同维度对数据进行划分,低维度的数据采用传统的并行

到稿日期:2017-10-30 返修日期:2018-02-07 本文受浙江省自然科学基金(LY16F020033),国家自然科学基金(61771430)资助。
刘端阳(1975-),男,博士,副教授,主要研究方向为数据挖掘、分布式计算,E-mail:ldy@zjut.edu.cn;郑江帆(1993-),男,硕士,主要研究方向为数据挖掘、并行计算,E-mail:980702938@qq.com;沈国江(1975-),男,博士,教授,主要研究方向为大数据、智能交通,E-mail:gjshen1975@zjut.edu.cn;刘 志(1969-),女,博士,教授,主要研究方向为大数据、人工智能,E-mail:lzhi@zjut.edu.cn(通信作者)。

方式，高维度的数据在计算距离时采用分块矩阵乘的方法，这是一种并行化实现的新观点。文献[7]根据算法运行时数据量的大小对数据进行划分，面对小数据量时，将其一次读入设备进行计算，面对庞大的数据时，将数量分批次读入设备进行计算，这种流式计算在面对大数据时简单有效，但没有充分利用 GPU 内部的存储器体系。文献[8]提出了一种基于 GPU 的并行化 CluStream 框架，其在面对高速度和高维度的实时数据流时可以发现具有任意形状的集群并可适当处理异常值。文献[9]在标记算法的数据点时引入了三角不等式的思想，减少了不必要的距离计算，从而提升了整体的性能。文献[10]提出了基于特征权重的全局 k-means 算法和基于熵权重的全局 k-means 算法在 GPU 上的实现。文献[11]针对算法容易陷入局部最优的问题，优化了初始簇中心的选择，并且在 GPU 上加以实现，但在计算距离时的并行性并不高。文献[12-13]将算法分别在 CUDA，MPI 以及 OpenMP 平台上实现，侧重于比较不同平台下算法的性能提升，只利用 CUDA 进行了并行的距离计算，仍有大量计算需要在 CPU 端执行，并行度不高。文献[14]提出了一种自适应的共享内存架构，它允许 GPU 上的流多处理器(Stream Multiprocessor，SM)共享其他 SM 资源中未使用的部分，提升了资源利用率，但也加大了编程难度。

基于上述国内外研究现状，本文提出了 GS_k-means 算法，该算法充分利用了 CUDA 函数库，发挥了 GPU 良好的运算能力。针对大量的重复距离计算问题，采用基于上下界的全局选择判断数据点在本轮迭代中是否会改变所属聚簇，从而减少了冗余计算。针对复杂的距离计算问题，将计算的公共部分提取出来，剩余部分利用 cublas 库的通用矩阵乘计算，加快了运算速度。针对每次迭代中簇中心点的更新问题，将数据按照同一簇标签排序，然后分组，将数据点的累计转化为每个组内的简单相加，从而最小化原子操作，提升整体性能。最后通过实验证明了在保证算法鲁棒性的前提下所提算法获得了良好的加速效果。

2　背景知识

2.1　k-means 算法

k-means 算法是目前最为流行的聚类算法之一。对于一个包含 n 个数据点的集合 $R=\{r_1,r_2,\cdots,r_n\}$，每个数据点的维度为 d，k-means 算法的目标是将数据点集合 R 划分到 $k(k<n)$ 个聚簇中，k 个簇的集合表示为 $S=\{S_1,S_2,\cdots,S_k\}$，划分结果使得目标函数 $\sum_{i=1}^{k}\sum_{x_j\in S_i}\|x_j-c_i\|^2$ 达到最小，c_i 表示聚簇 S_i 的中心点。这样的划分结果使得相似度高的对象聚在一个簇内，而不同簇之间的数据对象的相似度低。

算法 1　基于 CPU 的串行 k-means 算法

输入：n 个数据点的对象集合，k 个聚类个数，最大迭代次数 m，中心点收敛阈值 t

输出：k 个聚类集合

1. 从输入的 n 个数据对象中随机选取 k 个数据对象作为初始的聚类中心。
2. 分别计算其余的 n－k 个数据对象到 k 个聚类中心的距离，并将数据对象分配到与其距离最近的簇。
3. 当所有数据对象分配到簇之后，重新计算 k 个簇的新聚类中心。
4. 判断 k 个新聚类中心，若聚类中心的变化小于 t 或者迭代次数达到 m，则转步骤 5，否则转步骤 2。
5. 输出 k 个聚类集合。

假如每个输入对象的维度为 d，在一次迭代中：步骤 1 随机选取聚类中心的时间复杂度可以忽略不计；步骤 2 中计算距离的时间复杂度为 nkd，分配到各个簇的时间复杂度为 nk；步骤 3 的时间复杂度为 nd。当数据量 n 很大或者面对高维度数据时，运用 CUDA 计算架构并行化 k-means 算法通常可以取得很好的加速效率。

2.2　CUDA 编程模型

CUDA 是 NVIDIA 公司推出的一款通用并行计算架构，它的出现改变了传统的 GPU 程序编程方式，使用 CUDA 时不需要将程序任务转换为 GPU 图形处理任务，降低了开发难度。CUDA 架构包含计算核心和存储器体系，每个 GPU 的计算核心由多个 SM 组成，并且每个 SM 由多个标量处理器(Stream Processor，SP)组成。GPU 中提供了可供线程访问的多级存储器，各个线程具有私有的本地存储器。每个线程块都有共享存储器，共享存储器对线程块内的所有线程共享数据，并且它的生命周期与线程块相同，读写速度快。常量存储器和纹理存储器是两种只读存储器，可以针对不同的用途进行优化。全局存储器接收从 CPU 端传入的大量数据。

在 CUDA 编程模型中，GPU 被认为是能够并行执行大量线程的协处理器。一个简单的源程序包括运行在 CPU 上的主机端代码和运行在 GPU 上的内核(kernel)代码。Kernel 函数通过 _global_ 类型限定符定义，并通过 Kernel<<<*blocks*,*threads*>>>(*parameters*)调用，*blocks* 代表线程块的数量，*threads* 代表每个线程块中包含的线程数，*parameters* 代表函数调用时的参数列表。计算密集型和数据并行的内核函数代码运行在 GPU 上。所有线程被组成许多线程块，每个线程块中的线程会在同一个 SM 中并行执行。同一个线程块的线程可以通过块内的共享内存共享数据，并且可以通过 _syncthreads()函数同步数据。CUDA 编程模型中的另一个重要概念就是线程束(thread warp)，它由 32 个并行的线程组成，并且是每个流多处理器的基本调度单元。当一个线程束执行结束之后，流多处理器会接着调度另一个线程束继续执行，线程束一次执行同一条指令，因此当线程束中的 32 条线程有相同的执行路径时，可以达到最大的加速效率。当遇到条件分支时，线程束中的线程会产生不同的执行路径，这将会增加该线程束执行指令的总时间。为了不浪费资源，即线程束内的线程被充分利用，线程的数量经常设置为 32 的整数倍。访问内存时每半个线程束即 16 个线程作为一个组执行，若每半个线程束访问的是合并数据，则访问操作会在一个指令内执行完。只有合理地分配线程块以及线程的数量，才可以将 GPU 的利用率最大化。

3　基于 CUDA 的 k-means 算法并行优化

分析串行 k-means 算法的流程可知，由于算法的时间开

销大部分集中在数据点到质心的距离计算以及簇中心点的更新上，因此本文将在这两个步骤上进行并行优化，并设计一个基于 CUDA 的选择器以减少不必要的计算。下面将分别详细介绍 GS_k-means 算法在 3 个方面的优化过程。

3.1 基于 CUDA 的全局选择器

通常情况下，k-means 算法在每轮迭代时，很多数据点和其距离的簇中心与上一轮相同，因此在距离计算之前加上一个选择器来判断是否与上一轮迭代相同，如果相同，则可以略过本轮距离的计算。很多文献通过三角不等式来判断是否可以减少冗余计算，三角不等式的概念如下：令 $d(x,y)$ 代表 x 和 y 之间的距离，距离计算可采用欧几里得方法，对于另一个目标 z，有 $d(x,z)\leqslant d(x,y)+d(y,z)$。在 k-means 算法中，给定一个点 x 和两个簇中心 c_1,c_2，若知道 x 与 c_1 的距离以及 c_1 与 c_2 的距离，则可以通过三角不等式判断 x 与 c_2 的距离区间：

$$|d(x,c_1)-d(c_1,c_2)|\leqslant d(x,c_2)\leqslant d(x,c_1)+d(c_1+c_2) \tag{1}$$

基于三角不等式的思想，设计一种基于距离上下界的全局选择器。首先给出如下字符定义：令 C 代表所有聚类中心的集合，c 代表集合中的一个聚类中心，对于一个给定的数据点 x，$e(x)$ 代表与该数据点距离最近的聚类中心，C'，c' 和 $e'(x)$ 代表下一轮迭代中与上一轮对应的数据对象，令 $\phi(c)$ 代表 $d(c,c')$，即中心点更新后点 c 的距离偏移量。全局选择器是通过一个简单的条件来判断数据点 x 所属聚类中心是否改变。对于每一个数据点 x，该算法包含一个全局上界和一个全局下界，分别如式(2)和式(3)所示：

$$ue(x)\geqslant d(x,e(x)) \tag{2}$$

$$le(x)\leqslant d(x,c),\forall c\in C-e(x) \tag{3}$$

初始化上下界时，可以将 x 与 $e(x)$ 的距离作为上界，x 与第二近的簇中心的距离作为下界。对于一个数据点 x 和它所属的聚类中心 $e=e(x)$，如果在中心点更新后点 x 不改变其所属的聚类，则需满足条件：

$$ue(x)+\phi(e)\leqslant le(x)-\max_{c\in C}\phi(c) \tag{4}$$

证明：对于一个聚类中心 $c\in C-e$，令 c' 代表 c 经过中心点更新之后的该簇中心，e' 代表 e 经过中心点更新之后的该簇中心，基于三角不等式的概念，有：

$$\begin{aligned}d(x,c')&\geqslant d(x,c)-d(c,c')=d(x,c)-\phi(c)\\&\geqslant d(x,c)-\max_{c\in C}\phi(c)\\&\geqslant le(x)-\max_{c\in C}\phi(c)\end{aligned} \tag{5}$$

同理有：

$$\begin{aligned}d(x,e')&\leqslant d(x,e)+d(e,e')=d(x,e)+\phi(e)\\&\leqslant ue(x)+\phi(e)\end{aligned} \tag{6}$$

所以当满足条件式(4)时，得到：

$$\begin{aligned}d(x,e')&\leqslant ue(x)+\phi(e)\\&\leqslant le(x)-\max_{c\in C}\phi(c)\\&\leqslant d(x,c')\end{aligned} \tag{7}$$

即 $d(x,e')\leqslant d(x,c')$。因此，点 x 在中心点更新之后与所属簇中心的距离小于所有与其他簇中心的距离，由此判断点 x 所属聚类没有变化。

经过每轮的迭代之后，需要计算每个聚类中心距离的改变量 $\phi(c)$。对于没有改变聚簇的数据点，由于没有计算它到各个聚类中心的距离，因此下一轮的上界 $ue'(x)$ 可表示为：

$$ue'(x)=ue(x)+\phi(e) \tag{8}$$

下一轮的下界 $le'(x)$ 可表示为：

$$le'(x)=le(x)-\max_{c\in C}\phi(c) \tag{9}$$

这样便可以避免任何的数据点与聚类中心之间的距离计算。以上过程包含可以并行计算的部分，对于 n 个输入数据点和初始的 k 个聚类中心，将数据拷贝到 GPU 内存，由于线程块内的线程共享 k 个聚类中心，因此可将聚类放在共享内存中以提高存取速率；接着可以开启 n 个线程以计算每个数据点到 k 个聚类中心的距离，从而得到该数据点的全局上界 $ue(x)$ 和全局下界 $le(x)$；然后并行地计算簇更新之后的新聚类中心，得到该数据点对应的 $\phi(e)$ 和 $\max_{c\in C}\phi(c)$，在下一轮迭代前开启 n 个线程进行全局选择，若满足判断条件则无需更新所属聚类，若不满足则进行距离计算。在本轮聚类中心更新之后须重新计算该数据点的上下界，对于满足判断条件的数据点，上下界可以用式(8)和式(9)计算，对于不满足条件的数据点，则可以根据基于 CUDA 的距离计算得到上下界。基于 CUDA 的全局选择算法的整体流程如算法 2 所示。

算法 2 基于 CUDA 的全局选择算法

输入：n 个数据点的对象集合，聚类个数 k，最大迭代次数 m，中心点收敛阈值 t

输出：k 个聚类集合

1. 在主机端内存初始化数据点集 R，随机选取 k 个点作为初始聚类中心。
2. 在设备端分配数据点集、聚类中心、全局上下界以及中心点偏移量的内存空间。
3. 将主机端数据拷贝到设备端内存。
4. 在设备端调用内核程序，若数据点 x_i 不满足全局选择器判定条件，则开启线程计算数据点到各个聚类中心的距离，得到全局上下界，并根据最近的距离判断数据点所属的簇，然后将其划分到最近的簇。若满足判断条件，则保持所在的簇，使用式(8)和式(9)计算全局上下界。
5. 所有线程同步之后，在设备端进行中心点的更新。
6. 中心点更新之后，与之前的中心点进行比较，并进行偏移量计算，从而得到 $\phi(e)$ 和 $\max_{c\in C}\phi(c)$。
7. 若聚类中心改变量小于给定阈值或达到最大迭代次数，则结束循环，否则转步骤 4。

3.2 基于 Cublas 的距离计算

Cublas Library 是基于 NVIDIA GPU 的通用函数库，它实现了 Blas(基本线性代数子程序)，允许用户访问 GPU 中的计算单元，使用时在 GPU 中分配所需的矩阵向量空间并填充数据，然后调用其中的函数以加速向量、矩阵的线性运算。在目前大数据的背景下，通常是直接利用 CUDA 提供的各类函数达到加速线性运算的目的，对于 k-means 算法中庞大的距离计算量，本文利用 Cublas 库中的矩阵乘法函数 cublasS-

gemm()来加速距离计算。该函数能实现向量与矩阵，矩阵与矩阵之间的运算，并且拥有很好的加速比。

调用 cublasSgemm()函数时有许多参数，调用方法为 cublasSgemm(*handle*,*transa*,*transb*,*m*,*n*,*k*,*alpha*,*A*,*lda*,*B*,*ldb*,*beta*,*C*,*ldc*)，在 cublas 里面所有的矩阵都是按照列优先方式存储，*lda*,*ldb*,*ldc* 代表矩阵 A,B,C 的行数，矩阵 A 的维度为 $m*k$，B 的维度为 $k*n$，C 的维度为 $m*n$，*transa* 和 *transb* 代表矩阵 A 和矩阵 B 是否转置。函数实现的矩阵运算为：

$$C=alpha*OP(A)*OP(B)+beta*C \tag{10}$$

为了满足该运算的格式，对欧几里得距离计算公式进行平方运算，则 k-means 算法的距离计算可以转化为：

$$\begin{aligned}|x-y|^2 &= x^2+y^2-2x*y\\ &= alpha*(x*y)+beta*(x^2+y^2)\end{aligned} \tag{11}$$

其中，*alpha* 为 -2，*beta* 为 1，满足函数运算的表达式。在设计 cuda 核函数时，若有 n 个数据点，每个数据点的维度为 d，则总共有 k 个聚类中心，矩阵 A 为 x，代表 $n*d$ 维的数据点向量，矩阵 B 为 y，代表 $k*d$ 维的聚类中心矩阵。由于 x 和 y 都是按照列优先的方式存储的，因此为了得到每个数据点到各个聚类中心的距离矩阵，需要将 x 转置，参数列表中的 m 赋值为 n，n 赋值为 k，k 赋值为 d。在调用函数之前先要计算矩阵 C 即 x^2+y^2 的值，对于 x^2，可以开启 n 个线程，每个线程计算一个数据点向量 d 个维度上的平方和。对于 y^2，可以开启 k 个线程，每个线程计算一个聚类中心 d 个维度上的平方和。最后，将两者相加得到一个维度为 $k*n$ 的矩阵 C，每一列代表一个数据点分别与 k 个聚类中心的距离。

算法 3　基于 cublas 的距离计算

输入：n 个数据点矩阵 x，k 个聚类中心的矩阵 y

输出：n 个数据点分别到 k 个聚类中心的距离矩阵 z

1. 设备端分配矩阵 x 和矩阵 y 的内存空间。
2. 从主机端向设备端拷贝数据。
3. 开启 n 个线程，计算矩阵 x 的平方 x^2。
4. 开启 k 个线程，计算矩阵 y 的平方 y^2，并将其与步骤 3 的结果相加，得到 x^2+y^2。
5. 令 alpha 为 -2，beta 为 1，调用矩阵乘函数 cublasSgemm()得到结果矩阵 z。

3.3　基于同一标签排序分组的中心点更新

通过 3.2 节的距离计算得到了数据点集到每个聚类中心的距离矩阵，通过距离矩阵得到所属聚簇。并行中心点更新的过程如下：

1)调用 getNewLabels()函数，函数定义为：_global_static void getNewLabels()。

函数的输入为上一步得到的距离矩阵、数据点数量 n 以及聚类簇数目 k，输出为每个数据点所属簇中心的向量。在 GPU 中为每个线程块开启 128 个线程，共分配 $(n-1)/128+1$ 个线程块，每个线程负责一个数据点的计算，最后根据最小距离得到所属的聚类中心。由于部分数据在全局选择之后没有进行距离计算，因此在得到簇中心向量之后加上这部分数据的所属聚类，最终得到所有数据的簇中心向量。

2)调用 sortByKey()函数，函数定义为：_gl-obal_static void sortByKey()。

函数的输入为在第 1)步中得到的簇中心向量 *labels*，以及一个标记向量 $index=(1,2,\cdots,n)$，该标记向量的主要作用是定位每个数据点。在 CUDA 中提供了 Thrust 库，该库中提供了一个 sort_by_key()的函数，这个函数可以根据 *key* 值进行排序。在 *labels* 中从向量起始到向量结束的所属簇中心是杂乱无序的，该步骤将 *labels* 作为 *key*，将 *index* 作为 *value*，调用 sort_by_key()函数之后，可以将所有相同簇中心的数据点归入同一分组中，并得到 *new_labels* 向量。通过 sort_by_key()函数得到 k 组具有相同簇中心的数据点和 *new_index* 向量，该向量记录了经过排序之后每个数据点在原来 *index* 向量中的位置。

3)调用 getNewCentroids()函数，函数定义为：_global_static void getNewCentroids()。

函数的输入为所有的数据点向量 *data*，以及在步骤 2)中得到的 *new_labels* 和 *new_index*。GPU 开启合理的二维线程格以及二维线程块，然后为每个线程分配同组内的 $(n-1)/(blockDim.y*gridDim.y)$ 个数据点，并计算这些数据点在 $threadIdx.x$ 维度上的和，最后组内数据点分别在 d 个维度上累加求和，根据组内数据点总数得到每个维度的均值，输出更新之后的簇中心。

当更新之后的簇中心与原有簇中心的差值小于收敛阈值时，算法结束，GS_k-means 算法的整体流程如图 1 所示。算法先将数据从 CPU 拷贝至 GPU，进行一次距离计算以及中心点更新，初始化全局上下界，每轮迭代时所有数据先通过全局选择器进行判断，若满足条件则跳过距离计算，否则进行基于矩阵乘的距离计算并找到所属聚簇。汇总两部分数据进行排序分组，并计算新的中心点，当满足迭代跳出条件时算法结束。

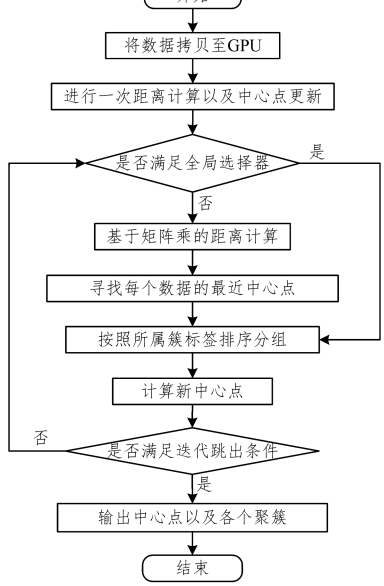

图 1　GS_k-means 算法流程图

Fig. 1　Flowchart of GS_k-means algorithm

4 实验与分析

4.1 实验环境

本文实验平台为 Intel Core i5-4590 CPU，主频为 3.30 GHz，系统内存为 8GB，GPU 平台为 NVIDIA GTX750 Ti，计算能力为 5.0，拥有 5 个流多处理器，640 个 SP。软件环境为 Windows10，Visual studio 2010，CUDA7.0。

4.2 实验结果

本文实验所使用的数据集为 KDDCUP1999 提供的网络入侵检测数据集[15]，该数据集包含大量真实的网络入侵数据，数据集中每个元素包含 41 个维度，每个维度的数据属性为浮点数，总共有 24 种攻击类型。由于每个维度上的度量单位不同，需要对输入数据进行标准化和归一化，本文使用文献[16]的方法进行数据的预处理。

1)固定 d，k，改变 n 对加速的影响

固定属性维度 d 为 41，聚类中心数目 k 为 24，收敛阈值设置为 0.00001，改变数据点总数 n 的大小，实验分析 GS_k-means 算法相对于单 CPU 条件下的 k-means 算法的加速比，以及 GS_k-means 算法相对于 GPUMiner 算法[17]的加速比，其中 GPUMiner 算法是采用位图的思想来实现 k-means 算法的加速。GS_k-means 算法与单 CPU 版本的 k-means 算法相比最大可达到 220 倍的加速比，而与 GPUMiner 相比最大可达到 5 倍的加速比。实验结果如表 1 所列，其中数据集 n 的单位为千条(k)，如 20k 为 20 千条，即 2 万条。从表 1 可以得知，当数据集 n 小于 100k 时，GS_k-means 算法相对于 GPUMiner 算法的加速比增长一般，加速效果更为明显，这是由于 GPUMiner 算法中对位图数据的维护导致了额外的开销。当 n 增大时，由于全局选择器在迭代后期作用越来越大，因此需要计算的数据点将会减少，从而取得良好的加速比。

表 1 与 CPU 和 GPUMiner 的性能比较结果

Table 1 Performance comparison results with CPU and GPUMiner

(单位：s)

n	CPU	GPUMiner	GS_kmeans	与 CPU 的加速比	与 GPUMiner 的加速比
20k	155.21	4.85	1.72	90.24	2.82
50k	203.13	6.21	1.91	106.35	3.25
100k	343.82	7.98	2.62	131.23	3.05
200k	609.33	15.64	3.65	166.94	4.29
400k	1042.11	23.72	4.73	220.32	5.01

2)固定 n，d，改变 k 对加速的影响

固定数据点 n 的大小为 50k，属性维度 d 为 41，改变聚类数目 k 的大小，对比分析 GS_k-means 算法和 GPUMiner 算法。当聚类数目 k 的值增大到 20 左右时加速比达到最大，继续增大 k 值时加速比会降低，这是由于 GS_k-means 算法在进行全局选择判定时，聚类数目 k 越大则数据点所属聚类中心在下一轮迭代时发生改变的概率就越大，需要计算距离的中心点也就越多。在分组排序阶段，排序的开销随着组类的增多而增大，这两点也是本文算法需要继续完善的地方。实验结果如图 2 所示。

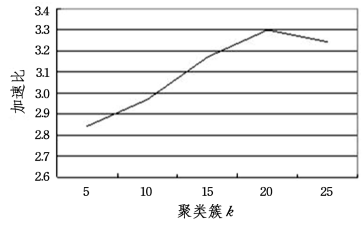

图 2 改变 k 对加速比的影响

Fig. 2 Effect of changing k on acceleration ratio

3)算法准确性的验证

为了验证算法的鲁棒性，选取 1 万条数据作为训练集，其中 9800 条为正常数据，200 条为入侵数据。另选 1000 条数据作为测试集，其中包含正常数据以及未知入侵数据。实验分别运用经典的基于 CPU 的 k-means 入侵检测算法和 GS_k-means 算法，在聚类结果中计算每个攻击类别的检测率和误报率，取 10 次实验后的平均值，实验结果如表 2 所列。

表 2 入侵检测的鲁棒性检测结果

Table 2 Robustness detection results of intrusion detection

(单位：%)

攻击类别	CPU 检测率	GS_k-means 检测率	CPU 误报率	GS_k-means 误报率
Dos	86.7	87.0	11.6	11.3
Probe	83.2	83.1	7.2	7.9
R2L	79.8	79.2	11.5	12.0
U2R	80.8	81.5	9.5	8.9

从结果来看，GS_k-means 算法在加速的情况下，在检测 4 类攻击上与标准 CPU 版本的 k-means 算法相比，检测率和误报率大致相同，主要是因为初始聚类中心的随机选择导致了略微偏差，但基本保持了良好的准确性。

结束语 本文在经典串行 k-means 算法的基础上，利用 GPU 的强大并行计算能力，提出了一种基于 CUDA 的并行 k-means 算法，即 GS_k-means 算法。该算法能有效处理大规模数据情况下的数据聚类问题，通过引入一个全局选择器，减少了不必要的距离计算；利用矩阵乘的思想加速了距离矩阵的运算；利用分组排序方法优化了中心点的更新，与经典的 GPUMiner 算法相比获得了 5 倍的加速比。然而，GPU 和 CPU 之间的数据传输和获取方面还有待研究，因此下一步会尽可能优化数据的存储，减少获取的延时，从而提升算法的性能。

参 考 文 献

[1] MACQUEEN J B. Some Methods for Classification and Analysis of MultiVariate Observations[C]// Proceedings of Berkeley Symposium on Mathematical Statistics and Probability. 1967: 281-297.

[2] LE V H, KIM S R. K-strings algorithm, anewapproach based on Kmeans[C]// Conference on Research in Adaptive and Convergent Systems. ACM, 2015: 15-20.

[3] KUMAR G R, MANGATHAYARU N, NARASIMHA G. An improved k-Means Clustering algorithm for Intrusion Detection using Gaussian function[C]// The International Conference on Engineering &Mis. ACM, 2015: 1-7.

[4] NVIDIA Corporation. CUDA Technology[OL]. http://www.nvidia.com/CUDA.

[5] ZECHNER M, GRANITZER M. Accelerating K-Means on the-Graphics Processor via CU-DA[C]// First International Conference on Intensive Applications and Services. IEEE Computer Society, 2009: 7-15.

[6] LI Y, ZHAO K, CHU X, et al. Speeding up K-Means Algorithm by GPUs[C]// International Conference on Computer and Information Technology. IEEE, 2010: 115-122.

[7] ZHONG S, LIN S, XU G, et al. The expansibility research of K-Means algorithm under the GPU[C]// IEEE International Conference on Software Engineering and Service Science. IEEE, 2017: 734-737.

[8] HUANG P, LI X, YUAN B. A Parallel GPU-Based Approach to Clustering Very Fast Data Streams[C]// ACM International on Conference on Information and Knowledge Management. ACM, 2015: 23-32.

[9] WU J, HONG B. An Efficient k-Means Algorithm on CUDA[C]// IEEE International Symposiumon Parallel and Distributed Processing Workshops and Phd Forum. IEEE Computer Society, 2011: 1740-1749.

[10] HUI Z N. Multidimensional data clustering alogorithm research and GPU acceleration based Global K-means[D]. Xi'an: Xidian University, 2012. (in Chinees)

惠转妮. 基于 Global K-means 的多维数据聚类算法研究及其 GPU 加速[D]. 西安：西安电子科技大学，2012.

[11] KAKOOEI M, SHAHHOSEINI H S. A parallel k-means clustering initial center selection and dynamic center correction on GPU[C]// Electrical Engineering. IEEE, 2015: 20-25.

[12] BHIMANI J, LEESER M, MI N. Accelerating K-Means clustering with parallel implementations and GPU computing[C]// High Performance Extreme Computing Conference. IEEE, 2015: 1-6.

[13] BAYDOUN M, DAWI M, GHAZIRI H. Enhanced parallel implementation of the K-Means clustering algorithm[C]// International Conference on Advances in Computational TOOLS for Engineering Applications. IEEE, 2016: 7-11.

[14] ABBASITABAR H, SAMAVATIAN M H, SA-RBAZI-AZAD H. ASHA: An Adaptive Shared-Memory Sharing Architecture for Multi-Programmed GPUs[J]. Microprocessors & Microsystems, 2016, 46: 264-273.

[15] HETTICH S, BAY S D. KDD cup 1999 data[EB/OL]. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[16] WANG Q, MEGALOOIKONOMOU V. A clustering algorithm for intrusion detection[J]. Proc Spie, 2008, 5812(5812): 31-38.

[17] FANG W, LAU K K, LUM, et al. Parallel datamining on graphics processors[OL]. http://11130.126.143.33/sites/default/files/papers/358/gpuminer.pdf.

（上接第 271 页）

[5] HSIN H F, LEOU J J, LIN C S, et al. Image inpaintingusing-structure-guided priority belief propagation and label transformations[C]// International Conference on Pattern Recognition. IEEE, 2010: 4492-4495.

[6] MANSFIELD A, PRASAD M, ROTHER C, et al. Transforming Image Completion[C]// British Machine Vision Conference. 2011.

[7] KANG Y, LEE K T, EUN J, et al. Stacked Denoising Autoencoders for Face Pose Normalization[C]// International Conference on Neural Information Processing. Springer Berlin Heidelberg, 2013: 241-248.

[8] XIE J, XU L, CHEN E. Image denoising and inpainting with deep neural networks[C]// International Conference on Neural Information Processing Systems. 2012: 341-349.

[9] PATHAK D, KRAHENBUHL P, DONAHUE J, et al. Context Encoders: Feature learning by inpainting[C]// 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016: 2536-2544.

[10] YANG C, LU X, LIN Z, et al. High-Resolution Image Inpainting Using Multi-scale Neural Patch Synthesis[C]// IEEE Conference on Computer Vision and Pattern Recognition. IEEE, 2017: 4076-4084.

[11] HUANG R, CHANG L, LIC G, et al. Adaptive Deep Supervised Autoencoder Based Image Reconstruction for Face Recognition [J]. Mathematical Problems in Engineering, 2016, 2016(5): 1-14.

[12] HINTON G E, OSINDERO S, TEH Y W. A fast learning algorithm for deep belief nets[J]. Neural Computation, 2006, 18(7): 1527.

[13] BENGIO Y. Learning deep architectures for AI[J]. Foundations and Trends in Machine Learning, 2009, 2(1): 1-127.

[14] ZHANG C, ZHANG Z Y. Improving multiview face detection with multi-task deep convolutional neural networks[C]// IEEE Winter Application and Computer Vision Conference. USA, 2014: 1036-1041.

[15] LÄNGKVIST M, KARLSSON L, LOUTFI A. A review of unsupervised feature learning and deep learning for time-series modeling[J]. Pattern Recognition Letters, 2014, 42(1): 11-24.

[16] ZHANG Y, CHEN Q Y, ZHANG Y J. Deep learning and its new progress in object and behavior recognition[J]. Journal of Image & Graphics, 2014, 19(2): 175-184. (in Chinese)
郑胤，陈权崎，章毓晋. 深度学习及其在目标和行为识别中的新进展[J]. 中国图象图形学报，2014，19(2)：175-184.

[17] RUMELHART D E, HINTON G E, WILLIAMS R J. Learning representations by back-propagating errors[J]. Nature, 1986, 323: 533-536.

[18] PEARSON K. Mathematical contributions to the theory of evolution (III): Regression, heredity, and panmixia[J]. Prceedings of the Royal Society of London, 1998, 187(4): 253-318.