计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200800215

基于近似匹配的移动边缘计算缓存管理方法

郦睿翔　毛莺池　郝　帅
河海大学计算机与信息学院　南京 211100
(loye317@sina.com)

摘　要　针对终端用户产生大量相同或相似计算请求的情况,可以通过近似匹配在边缘服务器缓存空间中查找相似数据,选取可复用的计算结果。现有算法大多未考虑数据分布不均的问题,导致计算量和时间开销较大,对此文中提出基于动态局部敏感哈希算法与加权k近邻算法的缓存数据选择策略(Cache Selection Strategy based on Dynamic-LSH algorithm and Weighted-KNN algorithm,CSS-DLWK)。其中,Dynamic-LSH 算法能够针对数据分布不均的问题,根据数据分布的变化动态调整哈希桶粒度,从缓存空间中选出与输入数据相似的数据集合;Weighted-KNN 算法以距离和样本数为权重,对由 Dynamic-LSH 算法获取的相似数据集合进行数据再选取,得到与输入数据最相似的数据,获取相应的计算结果以供复用。仿真实验结果表明,在 CIFAR-10 数据集中,与基于 A-LSH 算法与 H-KNN 算法的缓存选取策略相比,CSS-DLWK 策略的平均选取准确率提高了 4.1%;与传统的 LSH 算法相比,其平均选取准确率提高了 16.8%。CSS-DLWK 策略能够在可接受的数据选取时间开销内,有效地提高可复用数据选取的准确率,从而减少边缘服务器的重复计算。

关键词:移动边缘计算;缓存替换;近似匹配;数据复用;局部敏感哈希算法
中图法分类号　TP399

Cache Management Method in Mobile Edge Computing Based on Approximate Matching

LI Rui-xiang,MAO Ying-chi and HAO Shuai
School of Computer and Information,Hohai University,Nanjing 211100,China

Abstract　For the case of massive identical or similar computing requests from end users,a search of similar data in the cache space of the edge server by approximate match can be applied to select computing results that can be reused. Most of the existing algorithms do not consider the uneven distribution of data,resulting in a large amount of calculation and time overhead. In this paper,a cache selection strategy based on dynamic-locality sensitive hashing (LSH) algorithm and Weighted-k nearest neighbor (KNN) algorithm(CSS-DLWK) is proposed. The Dynamic-LSH algorithm can deal with uneven data distribution by dynamically adjusting the hash bucket size accordingly,thereby selecting data sets that are similar to the input data from the cache space. Then,regarding distance and sample size as weights,the weighted-KNN algorithm re-selects the data in the similar data sets acquired by the dynamic-LSH algorithm. From this approach,the data most similar to the input data are obtained,and the corresponding computing result is acquired for reuse. As demonstrated by simulation experiments,in the CIFAR-10 dataset,CSS-DLWK increases the ave-rage selection accuracy by 4.1% compared to the cache selection strategy based on A-LSH and H-KNN algorithms. The improvement is 16.8% compared to traditional LSH algorithms. Overall,with acceptable time costs in data selection,the proposed strategy can effectively improve the selection accuracy of reusable data,thereby reducing repetitive computation in the edge server.

Keywords　Mobile edge computing,Cache replacement,Approximate matching,Data reuse,Locality sensitive hashing algorithm

1　引言

随着边缘智能的普及,终端设备上出现了越来越多的人工智能应用,如淘宝中的“拍立淘”功能,终端用户可以利用“拍立淘”拍摄身边的物体,并上传到边缘服务器进行处理。同一个边缘服务器下,由于地理位置相近,“拍立淘”会拍摄到大量相似的照片,将这些相似图片上传到边缘服务器会得到相同的计算结果,存在重复计算的问题。由于边缘服务器的计算能力是有限的[1],此类重复计算会消耗计算资源,并延长终端用户的等待时延[2]。

到稿日期:2020-08-30　返修日期:2020-11-07　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发计划(2018YFC0407105);国家自然科学基金重点项目(61832005);华能集团重点研发课题(HNKJ17-21)
This work was supported by the National Key R&D Program of China(2018YFC0407105),Key Project of National Nature Science Foundation of China(61832005) and Key Technology Project of China Huaneng Group (HNKJ17-21).
通信作者:毛莺池(yingchimao@hhu.edu.com)

为了解决这个问题，可以通过近似匹配在边缘服务器缓存空间中查找相似数据，选取可复用的计算结果以减少重复计算，降低终端用户的等待时延[3]。以上过程会面临两个挑战：一方面，由于在图像识别和语音识别的场景中几乎不存在两个完全相同的图像和语音，只能查找最相似的数据而非完全相同的数据，因此传统的基于精确匹配的缓存选择策略不再适用[4]；另一方面，终端用户每天都会产生海量的数据，在海量数据中查找相同或相似的数据需要耗费大量的时间，并且由于数据维度的增加，搜索难度会更高[5-6]。

常用于构建高维数据索引结构的方法有 LSH[7]（Locality Sensitive Hashing）算法，R-Tree[8]，KD-Tree[9] 和 VP-Tree[10] 等。这些空间索引数据结构构造方法中，LSH 算法最有效，其他方法的复杂度都会受到数据维度的影响。使用 LSH 算法进行近似数据查找时会得到输入数据的相似数据集合，之后的一般做法是遍历相似数据集合，找出与输入数据距离最近的数据作为返回结果。但是，该方法的查找准确率较低，为了提高查找准确率，可以使用 KNN 算法对相似数据集合中的数据进行筛选。

本文针对边缘服务器缓存空间中数据近似匹配精确度低、效率差的问题，提出了基于桶粒度动态调整的 Dynamic-LSH 算法，能够从大量高维数据中快速准确地找到输入数据的相似数据集合，该算法能有效应对数据分布不均对 LSH 算法查找性能的影响。本文还提出了基于距离和样本数的 Weighted-KNN 算法，从由 Dynamic-LSH 算法得到的相似数据集合中准确找出与输入数据最近似的缓存数据，并获取相应的计算结果，将此数据返回给用户，从而减少重复计算。最后，通过与传统方法进行比较，验证了 CSS-DLWK 在可接受的时间开销内，能够有效地提高可复用数据选取的准确率。

本文第 2 节介绍了边缘计算系统中缓存管理的相关工作；第 3 节描述了基于近似匹配的移动边缘计算缓存管理方法的流程框架；第 4 节分别给出了基于 Dynamic-LSH 和 Weighted-KNN 的数据选取技术细节；第 5 节通过仿真实验评估缓存管理策略的有效性；最后总结全文。

2 相关工作

在移动边缘计算环境下，边缘服务器缓存终端用户的计算结果以供复用，当再次收到相同的计算请求时，从缓存空间中选取可复用数据，从而减少重复计算。传统的缓存数据选取策略一般是基于精确匹配的，认为只有完全相同的数据才是可复用的。对于同一类型的计算任务，如果终端用户的输入数据是相同的或高度相似的，以至于对应的计算结果相同，那么就会存在冗余计算，可以利用冗余消除技术解决此问题。现有的冗余消除一般是基于精确匹配实现的，例如 Sanadhya 等[11] 将冗余消除应用于通信系统，用于减少网络中的重复数据，提高网络的性能。DuBois 等[12] 将冗余消除应用于存储系统，通过消除存储系统中的冗余数据来提高空间利用率。这里的冗余数据均是指完全相同的数据。

在移动边缘计算环境下，终端用户产生的数据并不是完全相同的，在计算机视觉和语音识别领域，由于用户使用场景的差异性，往往不存在完全相同的图片数据和语音数据，因此精确的数据匹配并不适用于此类场景。

Kannan 等[13] 为了减少应用程序中的重复计算，先对应用程序工作流的每一步数据进行处理得到 TO（Transformation Output）数据，然后使用相似性度量来衡量 TO 数据与早期同一阶段 TO 数据的相似程度，但是若仅仅使用欧氏距离或余弦距离等方式来衡量数据之间的相似度，则在高维数据中此方法的准确率较低。Tang 等[14] 设计了一种安全的消除重复计算的系统，此系统在定义重复计算时使用了基于精确匹配的哈希算法，无法减少数据相似但不相同的情况下的重复计算。Mastorakis 等[3] 提出了一种应用于网络边缘的缓存框架，该框架简化了服务调用并改进了边缘冗余计算，并且能够针对不完全相同的计算任务的计算结果进行复用，大大提高了缓存资源的复用率。但是，在边缘环境中计算任务的输入数据分布与用户请求的动态变化对选取可复用数据的准确率有较大的影响，而上述工作并没有深入研究此方面的问题。

近几年，随着强化学习的快速发展，很多研究者开始在缓存策略中引入强化学习算法，将传统的缓存替换策略问题建模为马尔可夫决策（Markov Decision Process，MDP），然后利用强化学习算法进行求解。Zhang 等[15] 基于强化学习中的 DDPG 算法提出改进的 SDDPG（Supervised Deep Deterministic Policy Gradient）算法，采用有监督的深度确定性策略梯度算法来决定缓存的替换策略。Zhu 等[16] 使用深度强化学习中的 A3C（Asynchronous Advantage Actor-Critic）算法来预测缓存的访问趋势进而得到缓存替换策略，但在实际应用中缓存文件的大小对缓存策略的性能有显著的影响，其没有考虑此因素。Zhong 等[17] 受强化学习能够解决复杂控制问题的启发，提出了一种基于深度强化学习的缓存算法，该算法以最大化缓存的长期命中率为目的，但同样没有考虑缓存大小对缓存策略的影响。

上述基于强化学习的策略是解决缓存替换问题的一种新的方式，但仍处于研究阶段，效果不稳定。Guo 等[18-19] 将输入数据相似且计算结果相同的计算任务称为“模糊冗余”，通过研究发现由于空间相关性、时间相关性或语义相关性，终端用户上传的计算任务存在很多的近似数据，相同设备中的同一类型的应用程序往往会收到相同或相似的输入数据，经过相似的函数处理，计算结果具有可复用性。为了减少输入数据相似或相同的重复计算，Guo 等提出了利用 LSH 算法和 KNN 算法的近似匹配策略在缓存空间中选取相似的数据，但是对 LSH 算法进行改进时没有考虑不同数据分布对 LSH 查询性能的影响，对 KNN 算法进行改进时没有考虑与输入数据距离不同的数据对结果的不同影响[20]。

3 整体框架

3.1 问题陈述

对于图像识别、语音识别等应用，由于一般不存在完全相同的两个输入数据，为了实现相似计算任务的计算结果复用，需要对高维输入数据进行近似最近邻搜索，以找出最相似的数据。以 Google Lens[21] 为例，该功能是先拍摄周围物体并将拍摄结果上传到边缘服务器对图像中的物体进行识别，然后将信息返回给用户。在同一个区域中，由于街景相似，不同

设备上的 Google Lens 在运行时会将大量相同或相似的图片上传到同一个边缘服务器上，导致边缘服务器中存在重复计算，因此需要对缓存数据进行近似匹配。

近似匹配指给定一个数据集合 $P=\{x_i\in R^d\}_{i=1}^n$、一个查询数据 $q\in R^d$ 和一个近似比率 $c>1$，则数据 q 的近似最近邻搜索返回一个数据 $p\in D$，满足 $dist(p,q)\leqslant c\times dist(p^*,q)$，其中 p^* 是在数据集合 P 中查询数据 q 的最近邻数据。在实际应用中，不同的应用程序会产生不同的输入数据类型(如图像、音频和文本)，因此，近似匹配的第一步是将异构的原始输入数据在度量空间中转换为多维向量，在此基础上可以计算向量之间的欧氏距离来衡量它们的相似性。

在高维空间中搜索近似匹配项时，本文使用哈希表(Hash table)将数据表示成(*key*，*value*)结构，并设计一种特殊的哈希函数，使得相似度很高的数据以较高的概率分到同一个哈希桶内，得到输入数据的相似数据集合，之后遍历相似数据集合，找出与输入数据距离最近的数据作为结果返回。为了能够提高查找的准确率，可以使用 KNN 算法对相似数据集合中的数据进行筛选。

在移动边缘计算环境下，由于受到不同用户和不同场景的影响，得到的数据分布具有很大的差异性，会影响查找的复杂度和准确度。而 KNN 算法在应对数据分布不均或各类数据没有明显占主导地位的情况时性能表现不佳。

本文针对数据分布不均的问题，根据数据分布的变化动态调整算法中的哈希桶粒度，从缓存空间中选出与输入数据相似的数据集合，并以距离和样本数为权重，对 LSH 算法获取的相似数据集合进行数据再选取，得到与输入数据最相似的数据。

3.2 算法流程

本文提出的缓存数据选取流程如图 1 所示。首先将用户计算任务的输入数据和相应的计算结果作为一个键值对(*key*，*value*)存入边缘服务器的缓存空间中，当再次接收到用户计算任务时，通过 CSS-DLWK 在缓存空间中选出与用户计算任务的输入数据最近似的 *key* 值，进而获取相应的 *value* 作为可复用的计算结果，从而减少重复计算，降低用户的等待时延。

在选取可复用的计算结果时，首先利用 SIFT(Scale-invariant Feature Transform)算法[22]对高维输入数据进行特征提取，将其转化为特征向量。该方法通过检测空间极值来定位图片中的特征点，再为这些点赋予特征方向值以描述图片。SIFT 算法能够有效提取目标图片特征，且包含丰富的信息，适用于在海量特征数据库中进行快速、准确的匹配。

利用 Dynamic-LSH 算法在边缘服务器的缓存空间中构造数据结构，并将现有数据集放入该数据结构中。当边缘服务器接收到终端用户请求的输入数据时，利用 Dynamic-LSH 算法查找与该数据在同一个哈希桶中的相似数据集合。由于移动边缘计算环境下的数据分布不均且实时变化，为了应对此问题，Dynamic-LSH 算法通过感知相邻哈希桶中数据的分布情况，动态调整哈希桶的大小，从而有效减小数据分布不均的问题对 LSH 算法查找性能的影响。

在由 Dynamic-LSH 算法得到相似数据集合后，一般的做法是基于最近欧氏距离选取最相似数据，以此作为可复用的数据，但是此做法随机性较强，导致数据选取的准确率较低。为了提高数据选取的准确率，利用 Weighted-KNN 算法对由 Dynamic-LSH 算法输出的相似数据集合进行数据再选取。由于数据集合中的数据以(*key*，*value*)的形式出现，将数据根据 *value* 的不同分成不同的类别，然后 Weighted-KNN 算法根据数据集合中的数据与输入数据之间的距离和每类数据的个数为每个数据赋予权重，选出权重最大的一类数据的类别作为输入数据的类别，此类别下的数据即为可复用的数据。

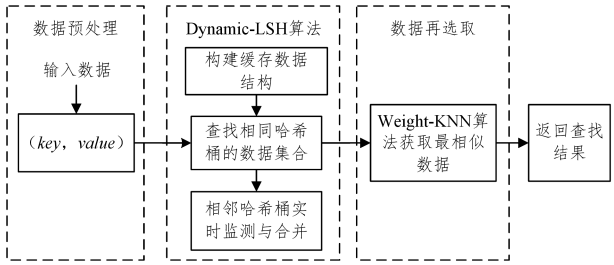


图 1 CSS-DLWK 算法流程

Fig. 1 CSS-DLWK algorithm flow

4 基于 Dynamic-LSH 与 Weighted-KNN 的数据选取

4.1 基于 Dynamic-LSH 的数据初选

LSH[7]算法常用于构建高维数据索引结构，传统的 LSH 算法随机选取 L 组哈希函数族 $G=\{g_1,g_2,\cdots,g_L\}$，每组哈希函数族由 w 个哈希函数组成 $g(\boldsymbol{v})=(h_1,h_2,\cdots,h_w)$，单个哈希函数的公式为[23]：

$$h_i(\boldsymbol{v})=\left|\frac{\boldsymbol{a}_i\cdot\boldsymbol{v}+\boldsymbol{b}_i}{r}\right| \tag{1}$$

其中，r 为哈希桶粒度大小；$\boldsymbol{b}_i$ 符合从 0 到 r 上的均匀分布，此参数增强了哈希函数的随机性，有利于消除哈希桶边界的影响；$\boldsymbol{a}_i$ 取自高斯分布的一个 d 维向量。每一个哈希函数 $h_i(\boldsymbol{v}):R^d\rightarrow Z$ 将 d 维向量 $\boldsymbol{v}$ 映射为一个整数，则向量 $\boldsymbol{v}$ 经过每组哈希函数族运算都能得到一个 w 维向量，一个 LSH 算法构造的数据结构由 L 个哈希表组成。对于两个高维向量，经过 LSH 算法的投影和量化计算，只要有一组哈希函数族使得两个向量哈希到同一个哈希桶中，就认为这两个向量是近邻的。两个向量 $\boldsymbol{v}_1$ 和 $\boldsymbol{v}_2$ 在 LSH 算法下整体的冲突(分配到同一个哈希桶)概率为：

$$\begin{aligned}p(c)&=\Pr(h(\boldsymbol{v}_1)=h(\boldsymbol{v}_2))\\&=\int_0^r\frac{1}{c}f_p\left(\frac{x}{c}\right)\left(1-\frac{x}{r}\right)\mathrm{d}x\end{aligned} \tag{2}$$

其中，$c=\|\boldsymbol{v}_1-\boldsymbol{v}_2\|_p$，$f_p(x)$ 是 p-稳态分布绝对值的概率密度函数。由公式可知两个向量发生冲突的概率随着它们之间欧氏距离的减小而增大。使用 L 组哈希函数族时，只需要有一组函数族将两个向量放在同一个桶中，即认为两个向量是近邻的，从而大大提高了两个向量碰撞的概率。

LSH 算法中的哈希函数是随机生成的，没有考虑数据的分布结构[7]。图 2 给出了在数据分布不均匀时选取固定的桶粒度参数 r 对 LSH 算法性能的影响。图 2 中有 3 个形状，即圆形、星形和正方形，同一类型的数据(*value* 相同的缓存数据)之间的欧氏距离较近，用同一种形状表示，被大概率放置

在同一个桶中；不同类型的数据（*value* 不同的缓存数据）之间的欧氏距离较远，用不同形状表示，只有很小的概率被放置在同一个桶中。其中，星形和正方形数据的分布较密集，圆形数据分布较稀疏，因此桶粒度 r 值对前者来说过大，导致不同类型的数据经过哈希函数计算后被放置到同一个桶，提高了后续查找的时间复杂度；桶粒度 r 值对圆形数据来说过小，使得该类数据被放置在多个桶中，导致无法全部选出有效数据。

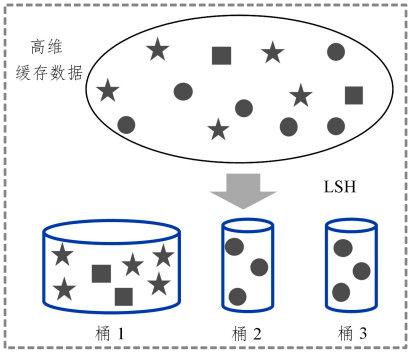


图 2 固定的哈希桶粒度对 LSH 算法性能的影响

Fig. 2 Effect of fixed bucket granularity on performance of LSH algorithm

为了使 LSH 算法在不同的数据分布下都有较好的查询准确率，Dynamic-LSH 算法首先采取较为严格的哈希策略，使得同一个桶中的数据之间都是高度相似的，没有低相似度的数据；然后测量相邻哈希桶之间的距离，将距离相近的桶合并，在增加同一个桶中高相似度数据的数量的同时减少低相似度数据的数量，从而降低 LSH 算法的时间复杂度，提高 LSH 算法查找近似数据的准确度。

为了便于算法的描述，首先定义 Dynamic-LSH 算法中相关的概念。

定义 1 哈希桶 $\boldsymbol{b}_i$ 的质心为 $\boldsymbol{C}_i=\frac{1}{m}\sum_{i=1}^{m}key_i$，桶 $\boldsymbol{b}_i$ 中数据向量的集合为 $(key_1,key_2,\cdots,key_m)$。

定义 2 哈希桶 $\boldsymbol{b}_x$ 与 $\boldsymbol{b}_y$ 之间的距离为 $db_{(\boldsymbol{b}_x,\boldsymbol{b}_y)}=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$，其中桶 $\boldsymbol{b}_x$ 的质心向量为 $\boldsymbol{C}_x=(x_1,x_2,\cdots,x_n)$，桶 $\boldsymbol{b}_y$ 的质心向量为 $\boldsymbol{C}_y=(y_1,y_2,\cdots,y_n)$。

定义 3 数据分布的疏密度指获取数据集中每个数据与第 k 近邻的数据之间的欧氏距离，记为 D_k，表示该点处的数据分布疏密程度。D_k 越大，表示数据分布越稀疏，D_k 越小，表示数据分布越紧密，其中 k 的取值是 KNN 算法在该数据集下获取最佳性能的值。

定义 4 哈希桶 $\boldsymbol{b}_i$ 中数据的离群率为 $outlier_i=1-\min_j\cos(key_j)$，$j\in[1,m]$，其中 $\cos(key_j)=\frac{\langle key_j,\boldsymbol{C}_i\rangle}{|key_j||\boldsymbol{C}_i|}$，$\boldsymbol{C}_i$ 为桶 $\boldsymbol{b}_i$ 的质心，桶 $\boldsymbol{b}_i$ 中的数据向量集合为 $\{key_1,key_2,\cdots,key_m\}$。

Dynamic-LSH 算法的要点描述如下：

（1）确定严格的哈希策略。选取尽量小的初始桶粒度 r_0，使 LSH 算法在数据分布最紧密的情况下表现最佳。

（2）动态合并相似桶。计算相邻的桶 $\boldsymbol{b}_x$ 与 $\boldsymbol{b}_y$ 之间的距离 $db_{(\boldsymbol{b}_x,\boldsymbol{b}_y)}$，当距离小于一定的桶间距阈值 $d_{\text{threshold}}$ 时，就可认为两个桶中的数据是邻近的，然后将这两个桶合并，合并时不需要移动桶中的数据，只需要将两个桶标记为合并状态即可。

考虑到随着数据规模的增长，每一个哈希桶中都会有大量的数据用于计算桶质心，其时间复杂度将会大幅度提高，因此可以采用随机抽样调查的方式抽取部分数据来计算桶质心。

基于 Dynamic-LSH 算法进行相似数据查找的示意图如图 3 所示，该算法通过测量相邻哈希桶之间的距离，动态合并相似桶以得到相似数据集合。

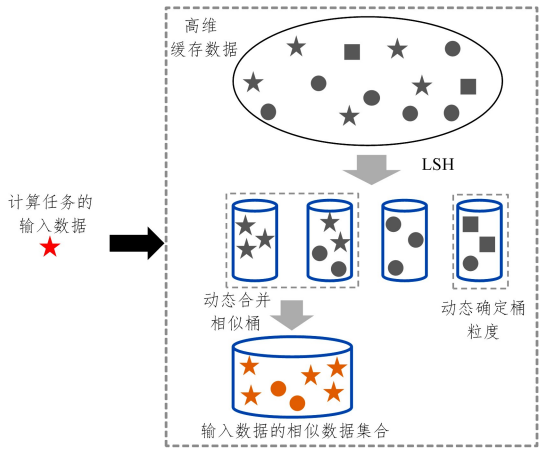


图 3 基于 Dynamic-LSH 算法的相似数据匹配

Fig. 3 Similar data matching based on Dynamic-LSH

4.1.1 初始桶粒度 r_0 的取值

由式（2）可知，参数 r 是用来衡量哈希函数桶粒度的参数；参数 c 是局部敏感因子，可以用来衡量数据集中数据的分布情况，为了使 LSH 算法在数据分布最紧密的情况下表现最佳，参数 c 要在数据集中数据分布紧密度 D_k 的最小值处取值，因此 $c=\lambda\times\min(D_k)$。在选取 λ 的值时首先应初始化 λ、置信概率 p_0 和离群率阈值 $outlier_0$。根据式（2）在 $p(c)=p_0$，$c=\lambda\times\min(D_k)$ 下得到相应的 r 值，利用 LSH 算法对当前数据集中的数据进行哈希操作，计算 LSH 中每个哈希桶的质心，进而获取每个桶的离群率 $outlier_i$，其中 $i\in[1,n]$，n 为哈希桶数量，找出所有桶中最大的离群率 $\max_i outlier_i$，$i\in[1,n]$。

如果 $\max_i outlier_i>outlier_0$，则返回当前 λ 值作为适用于最紧密数据分布情况下哈希桶粒度的衡量标准，否则 $\lambda=\lambda+1$ 并继续执行寻找最佳的 λ 值。得到合适的 λ 值后，由 $c=\lambda\times\min(D_k)$ 得到 c 值，在置信概率 p_0 下由式（2）得到相应的 r_0 值即为初始桶粒度值。

4.1.2 桶间距阈值 $d_{\text{threshold}}$ 的取值

$d_{\text{threshold}}$ 的值与 Dynamic-LSH 算法的查找精度和时间复杂度密切相关，该值越大，相邻的哈希桶合并的概率越大，合并之后的桶中找到与输入数据最近似的缓存项的概率越大，即查找精度越高，但是时间复杂度也会越高。为了保证相似数据都出现在同一个哈希桶中而又不会使桶中数据太多，并且保证该值随着数据分布的变化而动态调整，这里取 $d_{\text{threshold}}=mean(D_k)$，其中 $mean(D_k)$ 表示数据集中 D_k 分布的平均值。后续的实验部分会验证该取值的有效性。

在执行 Dynamic-LSH 算法时，首先初始化参数，包括置信概率p_0、η_0、数据集和哈希桶集合$\{b_1, b_2, \cdots, b_n\}$，并获取数据集中每个数据与第 k 近邻的数据之间的欧氏距离，记为D_k，表示该点处的数据分布疏密程度。由 $c=\lambda\times min(D_k)$得到 c 值，令 $p(c)=p_0$，由式(2)得到相应的初始桶粒度r_0，构建桶粒度为r_0的 LSH 索引结构。

接收用户请求任务的输入数据key_0，通过 Dynamic-LSH 算法将key_0哈希到桶b_i中，并得到桶中数据作为key_0的相似数据集合，将该集合作为后续 KNN 算法的输入。如果桶b_i的新增数据比例大于η_0，计算b_i与b_{i-1}、b_{i+1}的桶间距，如果大于$d_{threshold}$，则合并桶。

4.2 基于 Weighted-KNN 的数据再选取

现有的 KNN 算法普遍忽视了与输入数据不同距离的数据对 KNN 算法准确度的影响，认为同一类数据中的每一个数据的权重一致[24]。事实上，集合中的数据与输入数据之间距离的远近决定了该数据与输入数据的相似程度，因此在 KNN 算法中应充分考虑此因素的影响[25]。基于以上分析，本文提出基于距离和样本数的 Weighted-KNN 算法，对由 Dynamic-LSH 算法获取的相似数据集合中的数据进行更加准确的再查找，从而有效地提高数据选取的准确率。

在定义每个数据的权值时，Weighted-KNN 算法不仅考虑了每类数据包含的样本数，还考虑了每个数据与输入数据之间的欧氏距离。具体地，某类数据的个数越多，权值就越大，离输入数据的欧氏距离越远，权值就越小。对于输入数据key_0和由 Dynamic-LSH 算法得到的相似数据集合，首先计算相似数据集合中的数据与key_0之间的距离，取距离最近的 k 个数据，则可以得到集合：

$$data\_set_k=\{(key_1, value_1), (key_2, value_2), \cdots, (key_k, value_j)\}$$

其中，j 表示 k 个数据共可分为 j 个类别。

Weighted-KNN 算法在执行时初始化相似度阈值θ_0，给定key_0和 k 个最近邻数据集合$data_set_k$。在$data_set_k$中，将 $value$ 值相同的数据看作同一类数据，统计每一类数据包含的样本数，$value_m$类别的样本数记为n_m，其中 $m\in[1,j]$。定义类别为$value_m$的数据$(key_i, value_m)$的权重为：

$$w_i^m=\frac{n_m}{d(key_i, key_0)} \tag{3}$$

定义类别为$value_m$的判别函数为：

$$f_m=\sum_{i=1}^{n_m} w_i^m=\sum_{i=1}^{n_m}\frac{n_m}{d(key_i, key_0)} \tag{4}$$

令$\boldsymbol{f}_m$表示以f_m为模的坐标轴向量，计算和向量 $\boldsymbol{FF}=\boldsymbol{f}_1+\boldsymbol{f}_2+\cdots+\boldsymbol{f}_j$，可以得到$\boldsymbol{f}_m$与 $\boldsymbol{F}$ 的余弦值$\theta_m=\frac{\langle \boldsymbol{f}_m, \boldsymbol{F}\rangle}{|\boldsymbol{f}_m||\boldsymbol{F}|}$，$m\in[1,j]$。

取 j 个余弦值中的最大值$\theta_{\max}$，对应的类别记为$value_{\max}$。如果$\theta_{\max}>\theta_0$，返回$value_{\max}$作为输入数据key_0的类别；否则返回 null，查询失败。

通过 Weighted-KNN 算法进行数据再选取，得到返回值 $value$，$value$ 存在如下两种情况：

(1)如果 $value=$null，说明当前缓存空间中不存在与输入数据最相似的数据，需要边缘服务器对该输入数据对应的任务进行计算。

(2)如果 $value\neq$null，说明在当前缓存空间中找到了与输入数据最相似的数据，并将该返回值 $value$ 当作输入数据的计算结果直接返回给终端用户。

5 实验设计与结果分析

5.1 实验环境和数据集

实验使用的数据集为 ImageNet[26] 和 CIFAR-10[27]。ImageNet 数据集包含两万多种类别的图片，共有数千万张图片。该数据集是按照 WordNet 的层次结构组织的，WordNet 是一种包含语义信息的特殊词典，它将每一个具有相同意义的词条组成一个集合作为同义词集合(synset)，并为每一个集合提供简要的定义。ImageNet 数据集为每一个 synset 提供了上千幅图像，一共有 21 841 个 synset。

CIFAR-10 数据集是用于普适物体识别的小型数据集，该数据集中包括 6 万张图片，共有 10 个类别的 RGB 彩色图片，每个类别有 6 000 张图片，每一张图片标注了当前的图片大小和所属类别，因此可以用于多分类和检索的场景。

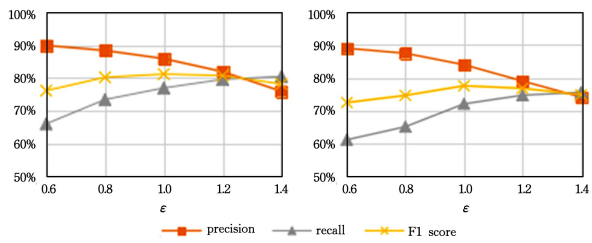
5.2 参数取值分析

(1)桶间距阈值$d_{\text{threshold}}$的取值分析

首先对 Dynamic-LSH 算法中的$d_{\text{threshold}}$参数的取值进行实验分析，该值是影响 Dynamic-LSH 算法表现性能的关键因素。选用 $F1\ score$ 作为衡量 Dynamic-LSH 算法综合性能的指标，其是查准率 $precision$ 和查全率 $recall$ 的调和平均值，计算公式为：

$$F1\ score=\frac{2\times precision\times recall}{precision+recall} \tag{5}$$

选取数据集的 90%作为缓存文件存入边缘服务器，将剩余的 10%作为查询数据。图 4 给出了不同$d_{\text{threshold}}$取值对 Dynamic-LSH 的查准率、查全率和 $F1\ score$ 值的影响，其中横坐标表示$d_{\text{threshold}}$的值与 $mean(D_k)$的比值，记为 ε。



(a)ImageNet 数据集 (b)CIFAR-10 数据集

图 4 $d_{\text{threshold}}$取值对 Dynamic-LSH 算法性能的影响

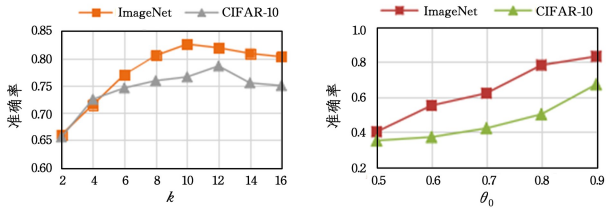
Fig. 4 Effect of $d_{\text{threshold}}$ on performance of Dynamic-LSH

由图 4 可知，在 ImageNet 数据集和 CIFAR-10 数据集中，随着 ε 的增大，Dynamic-LSH 算法的 $F1\ score$ 值将先增大后减小。这是因为当 ε 值过小时，$d_{\text{threshold}}$的值过小，此时距离较近的相邻哈希桶无法合并，使得相似度较高的数据无法放在同一个桶中，导致 Dynamic-LSH 算法无法查找到全部的相似数据；当 ε 值过大时，$d_{\text{threshold}}$的值过大，此时桶间距较大的哈希桶会进行合并，使得同一个桶中包含较多低相似度的数

据，导致 Dynamic-LSH 算法查找到的数据集合中包含较多与输入数据之间相似度较低的数据，降低了算法的性能；当 $\varepsilon=1.0$，即 $d_{\text{threshold}}=mean(D_k)$ 时，$F1\ score$ 最大，Dynamic-LSH 算法的综合性能最优。

(2)参数 k 和相似度阈值 θ_0 的取值分析

为了评估参数 k 和相似度阈值 θ_0 的取值对 Weighted-KNN 算法准确率的影响，选取数据集的 90% 作为缓存文件存入边缘服务器，将剩余的 10% 作为查询数据。实验结果如图 5 所示。



(a)k 对 Weighted-KNN 算法准确率的影响　(b)θ_0 对 Weighted-KNN 算法准确率的影响

图 5　不同参数取值对 Weighted-KNN 算法准确率的影响

Fig. 5　Effect of parameters on accuracy of Weighted-KNN

由图 5(a)可知，随着 k 值的增大，Weighted-KNN 算法的准确率先增大后减小，这是因为只有当 k 的取值符合数据分布情况时才能取得最高的准确率，过低或过高的 k 值都会导致准确率下降。在 ImageNet 数据集中，当 $k=10$ 时，Weighted-KNN 算法的准确率最高。在 CIFAR-10 数据集中，当 $k=12$ 时，Weighted-KNN 算法的准确率最高。

由图 5(b)可知，随着 θ_0 值的增大，Weighted-KNN 算法的准确率逐渐增大，这是因为只有余弦值大于 θ_0 的数据才能被选中并返回给用户，否则返回 null。θ_0 值过大时，会使得相似度较高的数据无法被选中，导致符合终端用户需求的目标数据无法被选中，因此 θ_0 值不宜过大。基于以上实验结果，在 ImageNet 数据集中取 $\theta_0=0.8$，在 CIFAR-10 数据集中取 $\theta_0=0.9$。

5.3　算法性能分析

为了评估 CSS-DLWK 的性能，选择如下缓存数据选取策略做为对比。

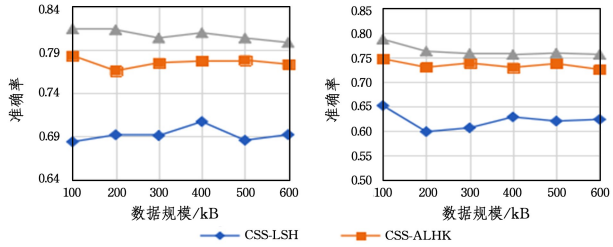
(1)基于 LSH 算法的缓存数据选取策略(Cache Selection Strategy based on LSH algorithm，CSS-LSH)。其通过 LSH 算法选取相似数据集合，然后计算输入数据与相似数据集合中数据之间的距离，选取距离最近的数据的计算结果返回给终端用户。

(2)基于 A-LSH 算法与 H-KNN 算法[17]的缓存数据选取策略(Cache Selection Strategy based on A-LSH algorithm and H-KNN algorithm，CSS-ALHK)。

实验中，选取不同数据规模的数据集进行实验，选取数据集的 90% 作为缓存文件存入边缘服务器中，剩余的 10% 作为查询数据。CSS-DLWK 策略的参数取值为：$d_{\text{threshold}}=mean(D_k)$，$p_0=0.95$，$\eta_0=0.2$，$outlier_0=0.25$；在 ImageNet 数据集中，$k=10$，$\theta_0=0.8$；在 CIFAR-10 数据集中，$k=12$，$\theta_0=0.9$。

(1)准确率对比

不同数据集下的数据选取准确率对比如图 6 所示。



(a)ImageNet 数据集　(b)CIFAR-10 数据集

图 6　不同数据集下的数据选取准确率对比

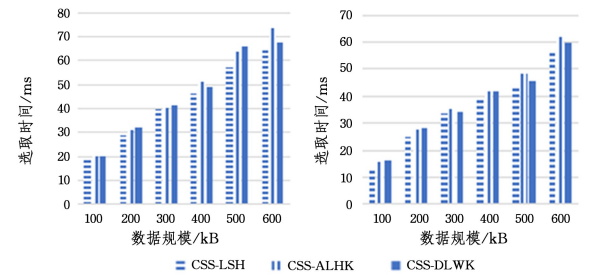
Fig. 6　Comparison of data selection accuracy under different data sets

由图 6(a)所示的 ImageNet 数据集上的实验结果可知，随着数据规模的增大，CSS-DLWK 的缓存数据选取的准确率均高于其他策略，与 CSS-ALHK 策略相比平均选取准确率提高了 3.9%，与 CSS-LSH 策略相比平均选取准确率提高了 22.9%，说明 CSS-DLWK 策略有较好的选取准确率。

为了进一步验证 CSS-DLWK 策略的通用性，在 CIFAR-10 数据集上进行补充实验，结果如图 6(b)所示。由实验结果可知，CSS-DLWK 策略依然具有最优的数据选取准确率，与 CSS-ALHK 策略相比平均选取准确率提高了 4.1%，与 CSS-LSH 策略相比平均选取准确率提高了 16.8%，证明 CSS-DLWK 策略有较好的通用性与鲁棒性。

(2)选取时间对比

不同数据集下的数据选取时间对比如图 7 所示。



(a)ImageNet 数据集　(b)CIFAR-10 数据集

图 7　不同数据集下的数据选取时间对比

Fig. 7　Comparison of data selection time under different data sets

由图 7 所示的 ImageNet 与 CIFAR-10 数据集上的实验结果可知，随着数据规模的增大，CSS-LSH，CSS-ALHK 与 CSS-DLWK 进行数据选取时消耗的时间随之上升。在不同的数据规模下，CSS-ALHK 与 CSS-DLWK 的数据选取时间稍稍长于 CSS-LSH 的选取时间。对比 CSS-ALHK 与 CSS-DLWK 的数据选取时间可知，二者在数据规模较小的条件下选取时间相近；扩大数据规模后，CSS-ALHK 的数据选取时间显著高于 CSS-DLWK。由此可得出结论，CSS-DLWK 数据的选取时间开销处于可接受的范围内。

结束语　本文针对边缘计算环境下存在相同或相似的计算任务的问题，提出基于 Dynamic-LSH 算法与 Weighted-KNN 算法的缓存数据选取策略 CSS-DLWK。为了减少重复计算并降低用户时延，本文设计了动态调整桶粒度的 Dynamic-LSH 算法，该算法能够从大量高维数据中快速准确地找

到输入数据的相似数据集合；为了提高数据选取的准确率，本文提出了基于欧氏距离和样本数的 Weighted-KNN 算法，该算法从由 Dynamic-LSH 算法得到的相似数据集合中准确找出与输入数据最接近的缓存数据，并将此数据返回给用户，从而进一步减少重复计算。最后，通过仿真实验证明，CSS-DL-WK 在可接受的数据选取时间开销内，有效地提高了可复用数据选取的准确率。

参 考 文 献

[1] SHI W, CAO J, ZHANG Q, et al. Edge computing: Vision and challenges[J]. IEEE Internet of Things Journal, 2016, 3(5): 637-646.

[2] GUO Y, LIU F, CAI Z, et al. Edge-based efficient search over encrypted data mobile cloud storage[J]. Sensors, 2018, 18(4): 1189.

[3] MASTORAKIS S, MTIBAA A, LEE J, et al. ICedge: When Edge Computing Meets Information-Centric Networking[J]. IEEE Internet of Things Journal, 2020, 7(5): 4203-4217.

[4] SANADHYA S, SIVAKUMAR R, KIM K H, et al. Asymmetric caching: improved network deduplication for mobile devices [C]// Proceedings of the 18th Annual International Conference on Mobile Computing and Networking. ACM, 2012: 161-172.

[5] GUO Y, LIU F, CAI Z, et al. Edge-based efficient search over encrypted data mobile cloud storage[J]. Sensors, 2018, 18(4): 1189.

[6] SUN S Y, YAO W B, LI X Y. DARS: A dynamic adaptive replica strategy under high load Cloud-P2P[J]. Future Generation Computer Systems, 2018, 78: 31-40.

[7] ANDONI A, INDYK P. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions[C]// 2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06). IEEE, 2006: 459-468.

[8] BECKMANN N, KRIEGEL H P, SCHNEIDER R, et al. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles[C]// Proceedings of the ACM SIGMOD Confe-rence on Management of Data. 1990: 322-331.

[9] SAMET H. The design and analysis of spatial data structures [M]. Reading, MA: Addison-Wesley, 1990.

[10] YIANILOS P N. Data structures and algorithms for nearest neighbor search in general metric spaces[C]// Soda. 1993: 311-321.

[11] SANADHYA S, SIVAKUMAR R, KIM K H, et al. Asymmetric caching: improved network deduplication for mobile devices [C]// Proceedings of the 18th Annual International Conference on Mobile Computing and Networking. ACM, 2012: 161-172.

[12] DUBOIS L, AMALDAS M, SHEPPARD E. Key considerations as deduplication evolves into primary storage[J]. White Paper, 2011, 223310.

[13] KANNAN K, BHATTACHARYA S, RAJ K, et al. Seesaw-similarity exploiting storage for accelerating analytics workflows [C]// 8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'16). 2016.

[14] TANG Y, YANG J. Secure deduplication of general computations[C]// 2015 USENIX Conference on Annual Technical Conference. 2015: 319-331.

[15] ZHANG Z, TAO M. Accelerated Deep Reinforcement Learning for Wireless Coded Caching[C]// 2019 IEEE/CIC International Conference on Communications in China (ICCC). IEEE, 2019: 249-254.

[16] ZHU H, CAO Y, WANG W, et al. Deep reinforcement learning for mobile edge caching: Review, new features, and open issues [J]. IEEE Network, 2018, 32(6): 50-57.

[17] ZHONG C, GURSOY M C, VELIPASALAR S. A deep reinforcement learning-based framework for content caching[C]// 2018 52nd Annual Conference on Information Sciences and Systems (CISS). IEEE, 2018: 1-6.

[18] GUO P, HU B, LI R, et al. Foggycache: Cross-device approximate computation reuse[C]// Proceedings of the 24th Annual International Conference on Mobile Computing and Networking. ACM, 2018: 19-34.

[19] GUO P, HU W. Potluck: Cross-application approximate deduplication for computation-intensive mobile applications[C]// Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems. 2018: 271-284.

[20] LI Y, CHENG B. An improved k-nearest neighbor algorithm and its application to high resolution remote sensing image classification. [C]// 2009 17th International Conference on Geoinformatics. IEEE, 2009: 1-4.

[21] PEREZ S. Lets smartphone cameras understand what they see [EB/OL]. https://techcrunch.com/2017/05/17/google-lens/.

[22] LOWE D G. Distinctive Image Features from Scale-Invariant Keypoints[J]. International Journal of Computer Vision, 2004, 60(2): 91-110.

[23] ANDONI A, INDYK P. E2LSH: Exact euclidean locality-sensitive hashing [EB/OL]. http://web.mit.edu/andoni/www/LSH/.

[24] COVER T, HART P. Nearest Neighbor pattern classification [J]. IEEE Transactions on Information Theory, 1967, 13(1): 21-27.

[25] SHAH J, SMOLENSKI B, YANTORNO R, et al. Sequentialknearest neighbor pattern recognition for usable speech classification[C]// 2004 12th European Signal Processing Conference. Vienna, 2004: 741-744.

[26] RUSSAKOVSKY O, DENG J, SU H, et al. ImageNet Large Scale Visual Recognition Challenge[J]. International Journal of Computer Vision, 2015, 115(3): 211-252.

[27] KRIZHEVSKY A, HINTON G. Learning Multiple Layers of Features from Tiny Images[R]. University of Toronto, 2009.

LI Rui-xiang, born in 1996, master candidate, is a student member of China Computer Federation. His main research interests include edge computing, and federated learning.

MAO Ying-chi, born in 1976, Ph.D, professor, is a senior member of China Computer Federation. Her main research interests include cloud computing and edge computing, mobile sensing systems and internet of things.