WHERE $T_1.k$=$T_2.fk$ AND $T_1.fk$=$T_3.k$ AND $T_1.k$<100

GROUP BY $T_1.k$

其中,T_1,T_2,T_3 分别表示了数据库中的源表,即查询计划的各个操作可能用到的源表;$T_1.k$ 表示 T_1 表的主键,同理,$T_3.k$ 表示 T_3 表的主键;$T_1.fk$ 表示了 T_1 表的外键,同理,$T_2.fk$ 表示了 T_2 表的外键.

查询优化器产生一个查询计划只要毫秒到秒级别的时间,因此,这些信息不难获取.查询计划本质上是一棵以各种类型的操作为节点的多叉树,每个节点上都有操作的相关信息.例如,图 6 中左侧 Hash Join 节点代表的是连接类型下的一种操作,其连接的条件是 $T_1.k$=$T_2.fk$.

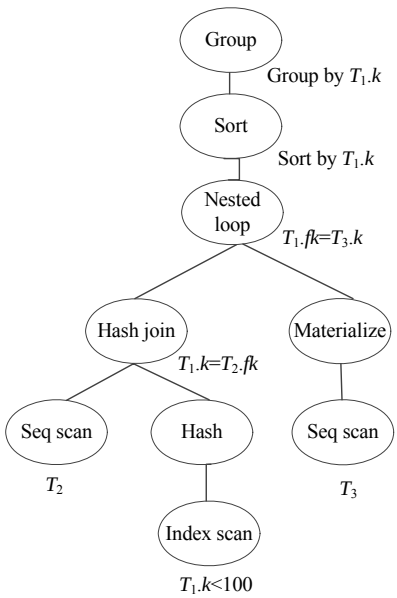

Fig.6 A typical query plan

图 6 查询计划树

在将查询计划编码成 LSTM 能接收的格式时,为了保留查询计划树的结构信息,本文采用后序编码,将其编码成一个操作序列 S_{op}={op_0,op_1,…,op_{m-1}},op_i 是操作序列 S_{op} 中第 i 个操作,m 表示查询计划中操作的个数.图 6 中查询计划树的后续遍历可以表示成:

$$S_{op}=\{T_2,\sigma T_1.k<100,Hash,\overline{T}=T_1\triangleright\triangleleft T_2,T_3,Materialize,\overline{T}\triangleright\triangleleft T_3,Sort,Group\}.$$

σ表示选择操作,$\triangleright\triangleleft$表示表的连接操作,$\overline{T}$ 表示 T_1,T_2 连接后的结果.使用多叉树后序遍历,一棵计划树可以被转换成独一无二的 S_{op}.反之亦然,因为在每一个节点中都保存了源表信息,即树中的叶子节点,所以 S_{op} 也能转换成一棵独一无二的计划树.

对于一个查询计划中的各个操作,经过后序遍历生成操作序列,遍历生成操作序列时,对于查询计划中的各个操作提取关键特征,将每个操作转换成向量 v.v 包含 5 个部分:

(1) n_0 代表操作的类型,例如 Hash Join,Nested Loop 等.PostgreSQL 中共有 34 种操作类型,表 1 给出了操作清单.因此,n_0 是一个 34 位的向量,该操作类型对应的位设置为 1,其他位设置为 0;

(2) n_1 代表操作在数据库中对应的源表.假设数据库有 k 个表,那 n_1 就有 k 位.计划树的叶子节点带有源表信息.例如,图 6 中左边叶子节点的源表是 T_2,因此,该叶子节点操作的 n_1 中 T_2 对应的位设置为 1,其他位设置为 0.子节点的源表信息会传递给父节点.例如,图 6 中 Hash Join 操作的源表是 T_1 和 T_2,分别来自它的两棵子树;

(3) n_2 代表操作在数据库中对应源表中涉及的列.假设数据库中所有表共 m 列,那 n_2 就有 m 位.例如,图 6

中 Hash Join 涉及表 T_1 中主键和表 T_2 中的外键,那么这两列对应的位就会被设置为 1,其余的位设置为 0;

(4) n_3 代表操作对应结果行的平均宽度,即每行所占字节数.假设操作的输出仅包含一列且该列的类型是 integer,那么结果行的平均宽度是 4 字节.将宽度的范围划分成 c 个区间,宽度落在哪个区间内,该区间对应的位就设置为 1,其余位设置为 0;

(5) n_4 代表操作对数据的选择率,即结果数据量占总量的百分比.这部分考虑了数据分布对于开销的影响.由于 TPC-H 的数据集分布是均匀的,因此本文实验部分不包含 n_4,但实际数据库中的数据分布很有可能是不均匀的.可以使用等宽直方图或等高直方图来计算选择率.等高直方图更加准确,因为频度较高的属性值,所处的桶范围就小,因此更为近似;相反,频度较低的属性值,等高直方图的近似相对不准确,但对于估算结果来说,频度高的属性值更重要.多列条件筛选时,选择率可以是多列选择率的乘积,也可以收集多列统计信息.前者的实现方式更为简单,后者估算结果更为准确,但需要维护统计信息.

Table 1 Operation list

表 1 操作清单

编号	类型	编号	类型	编号	类型
1	Result	13	Seq scan	25	Materialize
2	Insert	14	Index scan	26	Sort
3	Update	15	Index only scan	27	Group
4	Delete	16	Bitmap index scan	28	Aggregate
5	Append	17	Bitmap HeapScan	29	WindowAgg
6	Merge append	18	Tid scan	30	Unique
7	Recursive union	19	Subquery scan	31	SetOp
8	BitmapAnd	20	Function scan	32	LockRows
9	BitmapOr	21	Values scan	33	Limit
10	Nested loop	22	Cte scan	34	Hash
11	Merge join	23	Worktable Scan		
12	Hash join	24	Foreign scan		

向量 v 的前 3 个部分描述了查询的结构,后 2 个部分跟踪了查询过程中包含的数据规模.因此,当数据库系统中数据量动态变化时,模型的输入的一部分也会产生变化,因此模型的输出也会受到影响.同时,用户在此期间产生的历史查询会作为新的训练集继续训练模型,因而模型会随着数据库数据量的变化而变化.现在用一个向量来表示一个特定的操作,即 $S_v=\{v_0,v_1,\ldots,v_{m-1}\}$ 表示一个特定的计划,v_i 是后序编码中第 i 个操作.在这节的其余部分里,将展示如何使用 S_v 来训练 LSTM 模型.

3.2 LSTM模型

LSTM 有 3 个门:输入门、遗忘门和输出门,它可以自适应地记住过去的状态来加强它的学习过程.本文应用了 LSTM 的一个变种——peephole connections[14],如图 7 所示.

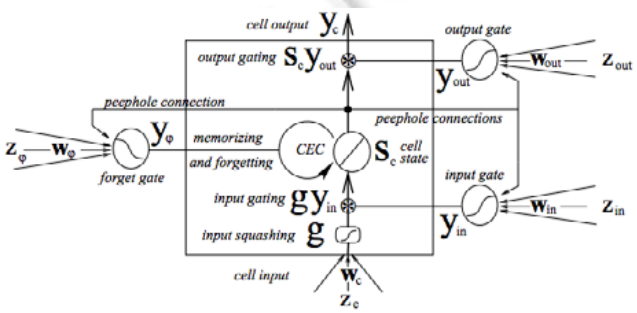

Fig.7 LSTM memory block with peephole connections from the CEC to the gates[14]

图 7 使用 peephole connections 的 LSTM 结构[14]

peephole 连接起了门与 CEC(constant error carousel),可以理解为 3 个门的输入信息增加了 Cell 的状态.这

个变种可以在没有任何短训练样本的帮助下学习间隔 50 个或 49 个时间步的序列之间的细微差别,比普通 LSTM 学习长序列的能力要强[12].而查询计划形成的序列也很长,因而选用了这种网络.

peephole connections 有多个实现的版本,只在细节上有所差异.本文所采用的 peephole connections 是 DL4J 中的版本,见公式(4)~公式(8).

定义 I 是输入的个数,K 是输出的个数,H 是隐藏层中细胞的个数,C 是块(block)中细胞的个数.w_{ij} 是 i 单元~j 单元之间的权重.在 t 时间, j 单元的值经过激励函数输出的值定义为 y_j^t .b_j 表示 j 单元的偏移量.下标 t、ϕ 和ω分别表示输入门、遗忘门和输出门.下标 c 表示 C 中的记忆细胞(memory cell). s_c^t 表示 t 时间细胞 c 的状态.下标 h 代表隐藏层中其他块的细胞输出.f 是门的激励函数,g_{in} 代表细胞输入的激励函数,g_{out} 代表细胞输出的激励函数.

公式(4)、公式(5)、公式(7)表示了 3 个门前向传播的计算方式,公式(6)、公式(8)表示 cell 的状态和 cell 的输出.关于反向传播的公式见文献[23].

$$y_\iota^t = f\left(\sum_{i=1}^{I} w_{i\iota} x_i^t + \sum_{h=1}^{H} w_{h\iota} y_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} s_c^{t-1} + b_\iota\right) \tag{4}$$

$$y_\phi^t = f\left(\sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} y_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} s_c^{t-1} + b_\phi\right) \tag{5}$$

$$s_c^t = y_\phi^t s_c^{t-1} + y_\iota^t g\left(\sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} y_h^{t-1} + b_c\right) \tag{6}$$

$$y_\omega^t = f\left(\sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} y_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} s_c^t + b_\omega\right) \tag{7}$$

$$y_c^t = y_\omega^t g_{out}(s_c^t) \tag{8}$$

由公式(4)~公式(8)可推导出 LSTM 隐藏层的参数个数为

$$Num_{parameters} = [(I + H + C + 1) \times 3 + I + H + 1)] \times H \tag{9}$$

使用第 3.1 节介绍的 S_v 作为模型的输入.模型试图去预测查询计划每一步的运行开销,再与真实的结果比较计算出误差来调节网络.

S_t 代表子计划在时间 t 已经被执行了.对于计划 S_v=$\{v_0,v_1,\ldots,v_{m-1}\}$来说,$S_t$=$\{v_0,v_1,\ldots,v_{t-1}\}$($t$<$m$)是它的一个子计划.将时间范围划分为 n 个区间,记为$\{c_0,c_1,\ldots,c_{n-1}\}$.

定义 S_t 的开销为$\delta(S_t)$,$\delta(S_t)$落入某区间 c_i 的概率表示为 $P_{S_t,c_i} = P(\delta(S_t) \in c_i)$,最终的预测结果见公式(10):

$$C(S_t) = MAX(P_{S_t,c_i})(i = 0,1,...,n-1) \tag{10}$$

例 1:对于图 6 中展示的查询计划,在预测了它的子计划 S_2=$\{T_2,\sigma T_1.k$<100,$Hash\}$后,有了扫描表 T_2 和选择 T_1 特定元组的开销预测.假设 S_2 的开销是$\delta(S_2)$.

接下来,模型要预测子计划 $S_3 = \{T_2,\sigma T_1.k < 100, Hash, \overline{T} = T_1 \triangleright\!\!\triangleleft T_2\}$ 的开销.假设过程 $\{\overline{T} = T_1 \triangleright\!\!\triangleleft T_2\}$ 产生了开销δ'.S_3 的开销实际上是$\delta(S_2)$+δ',模型通过训练可以掌握该规则.因为 LSTM 可以通过当前的输入和上一时刻的状态来预测当前的结果,所以子计划被作为上一时刻的状态来预测总查询计划的开销.

3.3 网络结构设计

网络设计要确定层数、每个隐藏层的节点数和激活函数、输出层的激活函数和代价函数,以及一些重要参数,如优化算法、初始学习率等.本文设计的网络结构如图 8 所示:第 1 层是输入层,中间两层是隐藏层,最后一层是输出层.隐藏层所用的激活函数是 *sigmoid*,输出层的激活函数是 *soft*max.两个隐藏层都是 100 节点.输入层的节点数取决于特征向量的长度,见表 3、表 4.

优化算法采用了随机梯度下降方法,初始学习率设为 0.05.

代价函数是交叉熵代价函数,见公式(11)(损失越小,表示预测效果越好.训练过程就是不断减小损失的过程):

$$J=-\sum_{k=1}^{n}\sum_{i=1}^{c}y_{ki}\log(\hat{y}_{ki}) \tag{11}$$

公式(11)中,样本数量为 n,类别数量是 c,y_{ki} 是样本 k 属于类别 i 的概率,$\hat{y}_{ki}$ 是模型预测样本 k 属于类别 i 的概率.

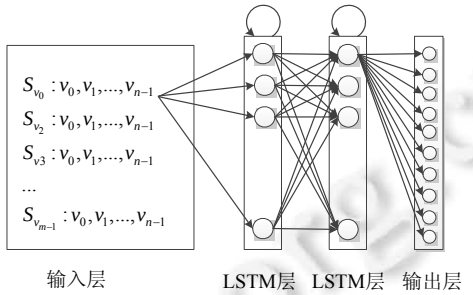

Fig.8 Network structure

图 8 网络结构

4 实 验

4.1 实验环境

为了评估模型的性能,本文使用 DL4J 实现了基于 LSTM 的开销预测模型.训练数据和测试数据来源于 TPC-H 决策支持基准[22],采用了 PostgreSQL 数据库.

本文做了两组实验:一组使用查询优化器产生的 cost 进行训练,一组收集查询真实的运行时间作为训练集.分别称为模拟数据和真实数据.

原型系统采用 Java 语言开发,系统环境为 Ubuntu 14.04.2 LTS,Linux 3.13.0-99-generic (x86_64),2x Intel(R) Xeon(R) CPU E5-2650 v2@2.60GHz,内存 96GB,硬盘 SEAGATE ST9300603SS SAS 10K,300GB.

4.2 训练集收集

大部分 DBMS(如 PostgreSQL 和 SQLServer)都支持 explain 命令,可以得到建议的执行计划和查询优化器估计的 I/O 开销.为了使用实际运行数据作为训练数据集,可以使用 explain analyze 命令去运行和收集计划中每一个操作真实的运行时间.

Table 2 The experimental data range

表 2 实验数据范围

实验组别	开销单位	数量		开销范围		
		训练集	测试集	平均	最小	最大
1	任意单位(arb. unit)	120 000	30 000	4.015E29	3.199E9	6.910E30
2	毫秒(ms)	8 000	2 000	387.122	2.154	6738.856

大部分由 TPC-H 模板产生的查询语句执行开销较小,并不满足均匀覆盖短时间查询和长时间查询的条件.为了产生长时间查询,实验根据数据库中的 8 张表自定义了查询模板.通过改变数据库配置和表的连接顺序,利用模板生成了数万条的查询计划.更改数据库配置,是为了引导查询优化器产生不同的查询计划.实验中随机更改 PostgreSQL 提供的 28 项设置,以引导查询优化器同时产生“好”和“不好”的计划.

特别注意的是:在收集每一条查询计划的运行时间时,都需要在执行该条计划前清理缓存,以确保数据的准确性.在 Linux 上,可以通过关闭 PostgreSQL 数据库,使用 drop_caches 清理系统层面的缓存,再重启 PostgreSQL 的方法清理缓存.另外,训练集在训练前要随机打乱,避免训练数据之间的相关性很大影响训练结果.

4.3 模型评估

本节介绍了两组实验中对模型的评估.表 3、表 4 总结描述了两次实验中网络各层的详细信息.表 3 中输入大小为 120,代表第 3.1 节中被转换成向量 v 的操作长度.实验中的向量 v 有 4 个部分:n_0 有 34 位,代表了 PostgreSQL 数据库中 34 种操作类型;n_1 有 8 位,代表了 TPC-H 中的 8 张表;n_2 有 61 位,代表了表中的所有列;n_3 有 17 位,表示结果行的平均宽度范围被划分成 17 个区间.4 部分总共 120 位.

实验使用 80%的数据作为训练集,20%的数据作为测试集.表 5 中第 1 组实验使用了 12 万条查询计划进行训练,3 万条独立的查询计划测试;第 2 组实验采用了真实运行时间数据,8 000 条查询计划用于训练,2 000 条独立的查询计划用于测试.

公式(12)用于评估模型预测的正确率,Num_{total} 指的是预测的总次数,Num_{hit} 指预测开销所在的区间与真实开销所在区间一致的预测次数.Accuracy 并不代表查询时间值的差异,而是代表了时间开销间隔类别的差异.

$$Accuracy=Num_{hit}/Num_{total} \tag{12}$$

实验结果见表 5:两组实验的正确率都高于 71%,模拟数据训练下模型的正确率高于 78%.分析预测结果发现:不正确的预测结果误差范围较小,对实际应用场景的执行影响很小.模型的训练时间代价见表 6,在本文系统环境下,实验 2 模型的训练代价仅为 1 小时,达到了 71%的正确率.

实验中模型终止迭代评分为 3.78.终止迭代次数见表 6 中,各为 2 000 次.

Table 3 Network layer information of the first experiment

表 3 实验 1 网络层级信息

层级	类型	输入大小	层大小	参数个数	激励函数
1	GravesLSTM	120	100	88 700	*sigmoid*
2	GravesLSTM	100	100	80 700	*sigmoid*
3	RnnOutput	100	10	1 010	*softmax*

Table 4 Network layer information of the second experiment

表 4 实验 2 网络层级信息

层级	类型	输入大小	层大小	参数个数	激励函数
1	GravesLSTM	112	100	85 500	*sigmoid*
2	GravesLSTM	100	100	80 700	*sigmoid*
3	RnnOutput	100	10	1 010	*softmax*

Table 5 Model evaluation results

表 5 模型评估结果

实验组别	数量		正确率(%)
	训练集	测试集	
1	120 000	30 000	78.244
2	8 000	2 000	71.258

Table 6 Model training time cost

表 6 模型训练时间代价

实验组别	迭代次数	时间		
		数据加载	训练时间	测试时间
1	2 000	72.503s	65.035h	458.33s
2	2 000	2.343s	1.001h	6 372ms

图 9 和图 10 是两次实验对于每个子计划预测的命中率,第 1 类的时间区间是(−∞,0],负数的时间实际是没有意义的,因此这个区间只包含 0.为了让每一个计划向量的长度相同,采用最大字长,不足最大长度的计划向量在前面补 0,这样的做法使得训练集和测试集都存在一定数量开销为 0 的操作.例如实验 2 中,2 000 条计划有 2000×6 个操作,其中,开销为 0 的个数为 5 005,命中的数量是 4 977,模型预测的准确率是 99.441%,但由于其数量之大影响了图 10 中其他数据的显示,因此图中没有给出.图 9 同理.

实验还比较了查询优化器与本文模型的预测速度.第 2 组实验中,对于 2 000 条测试数据,查询优化器用时 846ms,本文模型用时 6 372ms.本文模型的应用场景是负载管理,与查询优化器的代价模型有所不同,本文第 2 节给出了详细的说明.因此两者的速度要求并不相同.实验中本文模型用时是查询优化器的 8 倍.但随着深度学习算法的优化,和硬件的提升,未来模型时间性能的提升是必然的.

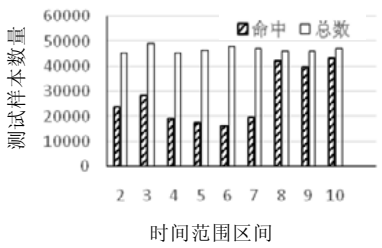

Fig.9 The hit condition of the first experiment

图 9 实验 1 的命中情况

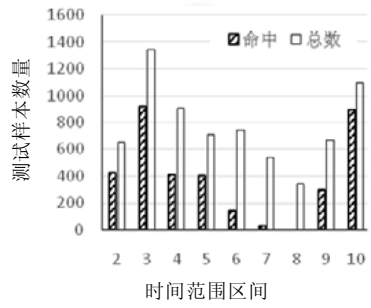

Fig.10 The hit condition of the second experiment

图 10 实验 2 的命中情况

5 结束语

本文提出一种基于循环神经网络的查询开销预测模型.给出一个特定的查询计划,在计划实际执行前,模型就能够产生该查询计划实际运行时间区间的预测,不管是短时间查询还是长时间查询,模型的预测结果都较为准确.特别地,本文设计了一种编码方法,能够将具有复杂结构的查询计划转换成特征向量.通过实验验证,模型的正确率高于 71%,说明了本文模型的可行性.后续工作将继续研究模型在 PostgreSQL 以外的关系型数据库上的性能表现以及如何将模型扩展应用到关系型数据库以外的数据库上,例如 MongoDB、HBase 等.

References:

[1] Wu W, Chi Y, Zhu S, Tatemura J, Hacigümüs H, Naughton JF. Predicting query execution time: Are optimizer cost models really unusable? In: Proc. of the 2013 IEEE 29th Int'l Conf. on Data Engineering (ICDE). New York: IEEE, 2013. 1081−1092. [doi: 10.1109/ICDE.2013.6544899]

[2] Tozer S, Brecht T, Aboulnaga A. Q-Cop: Avoiding bad query mixes to minimize client timeouts under heavy loads. In: Proc. of the 2010 IEEE 26th Int'l Conf. on Data Engineering (ICDE). New York: IEEE, 2010. 397−408. [doi: 10.1109/ICDE.2010.5447850]

[3] Xiong P, Chi Y, Zhu S, Tatemura J, Pu C, HacigümüŞ H. ActiveSLA: A profit-oriented admission control framework for database-as-a-service providers. In: Proc. of the 2nd ACM Symp. on Cloud Computing. New York: ACM Press, 2011. 15. [doi: 10.1145/2038916.2038931]

[4] Mishra C, Koudas N. The design of a query monitoring system. ACM Trans. on Database Systems (TODS), 2009,34(1):1−51. [doi: 10.1145/1508857.1508858]

[5] Wasserman TJ, Martin P, Skillicorn DB, Rizvi H. Developing a characterization of business intelligence workloads for sizing new database systems. In: Proc. of the 7th ACM Int'l Workshop on Data Warehousing and OLAP. New York: ACM Press, 2004. 7−13. [doi: 10.1145/1031763.1031766]

[6] Ganapathi A, Kuno H, Dayal U, Wiener JL, Fox A, Jordan M, Patterson D. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In: Proc. of the 2009 IEEE Int'l Conf. on Data Engineering (ICDE 2009). New York: IEEE, 2009. 592−603. [doi: 10.1109/ICDE.2009.130]

[7] Elnaffar S, Martin P, Horman R. Automatically classifying database workloads. In: Proc. of the 11th Int'l Conf. on Information and Knowledge Management (CIKM 2002). New York: ACM Press, 2002. 622−624. [doi: 10.1145/584792.584898]

[8] Akdere M, Çetintemel U, Riondato M, Upfal E, Zdonik SB. Learning-Based query performance modeling and prediction. In: Proc. of the 2012 IEEE 28th Int'l Conf. on Data Engineering. New York: IEEE, 2012. 390−401. [doi: 10.1109/ICDE.2012.64]

[9] Ahmad M, Aboulnaga A, Babu S, Munagala K. Interaction-Aware scheduling of report-generation workloads. The VLDB Journal—The Int'l Journal on Very Large Data Bases, 2011,20(4):589−615. [doi: 10.1007/s00778-011-0217-y]

[10] Duggan J, Cetintemel U, Papaemmanouil O, Upfal E. Performance prediction for concurrent database workloads. In: Proc. of the 2011 ACM SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2011. 337−348. [doi: 10.1145/1989323.1989359]

[11] Wang W, Zhang M, Chen G, Jagadish HV, Ooi BC, Tan KL. Database meets deep learning: Challenges and opportunities. ACM SIGMOD Record, 2016,45(2):17−22. [doi: 10.1145/3003665.3003669]

[12] Pavlo A, Angulo G, Arulraj J, Lin H, Lin J, Ma L, Santurkar S. Self-Driving database management systems. In: Proc. of the 8th Biennial Conf. on Innovative Data Systems Research (CIDR). 2017.

[13] Gers FA, Schmidhuber J, Cummins F. Learning to forget: Continual prediction with LSTM. Neural Computation, 2000,12(10):2451−2471. [doi: 10.1162/089976600300015015]

[14] Gers FA, Schraudolph NN, Schmidhuber J. Learning precise timing with LSTM recurrent networks. Journal of Machine Learning Research, 2003,3(1):115−143. [doi: 10.1162/153244303768966139]

[15] Ioannidis YE. Query optimization. ACM Computing Surveys (CSUR), 1996,28(1):121−123. [doi: 10.1145/234313.234367]

[16] Scott DW. On optimal and data-based histograms. Biometrika, 1979,66(3):605−610. [doi: 10.1093/biomet/66.3.605]

[17] Ioannidis YE, Poosala V. Balancing histogram optimality and practicality for query result size estimation. ACM SIGMOD Record, 1995,24(2):233−244. [doi: 10.1145/568271.223841]

[18] Haas PJ, Naughton JF, Seshadri S, Stokes L. Sampling-Based estimation of the number of distinct values of an attribute. In: Carey M, ed. Proc. of the 21th Int'l Conf. on Very Large Data Bases (VLDB'95). New York: ACM Press, 1995. 311−322. [doi: 10.1145/223784.223841]

[19] Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014.

[20] Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078, 2014.

[21] Sutskever I, Vinyals O, Le QV. Sequence to sequence learning with neural networks. In: Ghahramani Z, ed. Proc. of the 27th Int'l Conf. on Neural Information Processing Systems (NIPS 2014). Massachusetts: MIT Press Cambridge, 2014. 3104−3112.

[22] TPC-H benchmarks. http://www.tpc.org

[23] Graves A, Schmidhuber J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Networks, 2005,18(5):602−610.

毕里缘(1993－),女,上海人,硕士生,主要研究领域为数据库.

伍赛(1980－),男,博士,副教授,CCF 专业会员,主要研究领域为数据库.

陈刚(1973－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为数据库.

寿黎但(1974－),男,博士,教授,博士生导师,CCF 专业会员,主要研究领域为数据库.

陈珂(1977－),女,博士,副研究员,CCF 专业会员,主要研究领域为数据库.

胡天磊(1982－),男,博士,副教授,主要研究领域为数据库.