DOI: 10.11817/j.issn.1672-7207.2018.11.014

一种基于 Bitmap 的活动时间冲突查询算法

沈瑛，陈望远，侯晨煜，徐锦婷，曹斌，董天阳，范菁

(浙江工业大学 计算机科学与技术学院，浙江 杭州，310023)

摘要：提出 1 种基于 Bitmap 的活动时间冲突查询算法。首先对原始数据预处理以构建 Bitmap 索引结构，然后构建两阶段查询算法：第 1 阶段遍历 Bitmap 索引得到满足各个活动持续时间的候选时间区间和候选用户集合，并过滤其中的无效用户、调整候选时间；第 2 阶段完成冲突区间组合优化，获得不冲突条件下活动组织的全局最优方案；最后，以 8 628 个用户的 50 000 条真实数据(时间跨度为 1 月)进行实验，分为单活动及多活动场景，以用户数量、时间范围、活动数量、持续时间等为测试指标，对比本文算法与滑动时间窗口法测试结果。研究结果表明：本文提出的算法能够满足大规模、涉及时间冲突的活动组织查询的时效性要求，该算法查询速度比滑动时间窗口法的查询速度快，单活动场景下其查询响应速度约为滑动时间窗口法的 100 倍。

关键词：查询服务；活动时间冲突；Bitmap 索引；全局最优；时间区间

中图分类号：TP391.1 **文献标志码：**A **文章编号：**1672−7207(2018)11−2738−07

A bitmap index based activities conflict query algorithm

SHEN Ying, CHEN Wangyuan, HOU Chenyu, XU Jinting, CAO Bin, DONG Tianyang, FAN Jing

(College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China)

Abstract: A Bitmap index based activities conflict query algorithm was proposed. Firstly, original data was preprocessed by the algorithm to construct Bitmap index structure, then the two-phase query algorithm was constructed. During the first phase, the Bitmap index was traversed using the query algorithm to pick out candidates of time ranges and users which meet the activity duration requirements. Then invalid users were filtered and time ranges were adjusted. During the second phase, time intervals were optimized to find out non-conflict global optimized activity schemes. Experiments in single activity and multiple activities scenes with variations in user number, activity number, and time range and time duration were conducted on a dataset with 50 000 records of 8 628 users collected in one month. The performance of the proposed algorithm and sliding time window algorithm were compared. The results show that the proposed algorithm can meet time efficiency requirement of large-scale conflicted activity organization query. The proposed Bitmap based algorithm has an obvious advantage over sliding time window algorithm in query speed. In single activity scene, the query speed of the proposed algorithm is nearly 100 times faster than that of sliding time window algorithm.

Key words: query service; activity time conflict; Bitmap index; global optimization; time interval

收稿日期：2018−01−08；**修回日期：**2018−04−16

基金项目(Foundation item)：国家重点研发计划项目(2016YFB1001403)；国家自然科学基金资助项目(61602411)；浙江省重大科技专项重点工业项目(2015C01034, 2017C01013)；杭州市重大科技创新项目(20152011A03) (Project(2016YFB1001403) supported by the National Key Research & Development Program of China; Project(61602411) supported by the National Natural Science Foundation of China; Projects(2015C01034, 2017C01013) supported by Key Research and Development Program of Zhejiang Province; Project(20152011A03) supported by Major Science and Technology Innovation Program of Hangzhou)

通信作者：沈瑛，副教授，从事时序数据挖掘和可视化、信息安全研究；E-mail: shenying@zjut.edu.cn

在旅行、系列会议组织等面向大规模、多用户、多活动的方案查询应用场景中，查询效率依赖于支持时间冲突[1]协调的查询算法，即给定多个用户和对应的空闲时间区间，有多个活动要举办且每个活动需要持续一定时间，查询时间上不冲突且参与活动总人次最多的合理活动组织方案。一般可采用固定滑动时间法[2]，通过设置固定的活动持续时间窗口，然后沿着时间线逐步获取每个时间区间及该区间内的用户，并优化每个活动的查询结果，得到可行的活动时间组合方案，但这种方法在时序数据量较大时查询耗时较长。Bitmap 索引[3]是数据库中常用的多维数据查询处理技术，它将对象的索引信息采用游程编码压缩技术[4]存储成一系列二进制位向量[5]，具有空间开销较小、查询响应速度快的优势。本文作者提出 1 种基于 Bitmap 的活动时间冲突查询算法；首先对用户的有效时间区间数据进行预处理，建立 Bitmap 索引结构，然后遍历索引结构，获取每个活动的候选时间区间和对应的候选用户集合，最后对查询结果进行组合优化得到无冲突的活动全局最优方案，基于真实的用户时间数据，评估不同的影响因素对查询算法的影响，验证算法的有效性。

1　相关工作

活动时间冲突问题类似于时态数据库中的时态聚集查询问题，主要包括时态查询语言中对时态聚集的支持[6]、时态聚集索引[7]、时态聚集实例化、时态聚集查询算法[8]等。其中，时态聚集查询算法非常关键，它涉及时态分组的问题，必须先把时间序列划分为时间区间并分组计算聚集值。

典型的时态聚集查询算法可根据有、无索引分为 2 类。无索引的算法通过预扫描数据在内存中保存局部结果[9−11]：有的基于聚集树求得聚集结果[9]，但节点插入时会影响算法整体性能；有的算法基于平衡树[10]的叶节点存储聚集结果，搜索效率和调整效率较高，但需要元组排序开销较大；还有些基于并行算法[12]。基于时态索引的算法一般将局部聚集计算结果以索引形式存于外存储器中[13]。

然而，上述时态聚集查询算法都不能直接应用于解决活动时间冲突问题。这是因为活动时间冲突问题的场景与典型时态聚集查询存在查询目的、约束和处理逻辑上的差异。1) 查询目的不同。传统时态聚集查询通常是针对某一时刻或时间区间的用户聚合值，但活动时间冲突问题需要查询的是活动时间不发生冲突时，总用户数最大的时间区间。2) 查询约束不同。活动时间冲突问题的约束为待查询时间区间必须满足的活动持续时间和查询时间范围。3) 查询处理逻辑不同。活动时间冲突查询的最佳结果中的用户存在时间区间必须完全覆盖待查询的最优时间区间，而不仅是有交集。

2　问题定义

2.1　数据说明

1) 原始数据集。时序数据是以规律的时间顺序采集的时间序列的集合[14]。本文所研究的原始用户时序数据即为用户在 1 个月内的空闲时间区间。它的元素是一个二元组，第 1 分量为用户的唯一标识，第 2 分量为用户的所有空闲时间区间的列表，每个空闲时间为从开始时刻到结束时刻的 1 个时间片。

2) 持续时间。持续时间是由活动组织者预先设定的举办活动需要的最短时长。若某用户存在空闲时间区间包含某个活动持续时间的情形，即该用户能参加该活动，则认为用户对该活动有效；否则为不能参加该活动的无效用户。

例如某晚会的持续时间设为 2 h，而用户 A 在 2017−07−07T20:00:00—22:00:00 有空，用户 B 在 2017−07−08T19:30:00—21:00:00 有空，那么用户 A 是该晚会的有效用户之一，用户 B 不能参与该晚会。

3) 时间范围。时间范围可看作是一个大的时间区间，在活动查询前设定，是所有要举办的活动在时间上的约束。

例如某个活动要求在 2017−07−07T09:00:00—21:00:00 范围内举办，其时间范围可描述为 [2017−07−07T09:00:00, 2017−07−07T21:00:00]。

4) 预处理数据集。以 1 个活动的持续时间对用户时序数据进行预处理后，可以构建用户信息表，如表 1 所示。用户信息表存储所有满足持续时间的有效用户和对应的有效时间区间。

表 1　用户信息表

Table 1　User Table

用户	空闲时间区间
u00183	[2013−07−20T00:31:34, 2013−07−20T00:36:29], [2013−07−20T00:42:29, 2013−07−20T00:54:45], …
u00190	[2013−07−10T07:22:28, 2013−07−10T07:24:15], …
u00200	[2013−07−07T22:39:36, 2013−07−07T22:46:12], …

2.2 活动时间问题定义

基于上述基本概念，活动时间冲突问题可以定义下述概念。

冲突。若 2 个时间区间 I_A 和 I_B 满足 $I_A \cap I_B \neq \varnothing$，则认为这 2 个时间区间发生冲突。

活动时间冲突问题。给定 1 个数据集，包含了多个用户 U 的若干空闲时段 I。若有多个活动(活动总个数 $n \geqslant 1$)需在某时间范围 R 内举办，则活动的持续时间 $D_i(D_i>0,\ i \in N,\ N=\{1, 2, 3, \cdots, n\})$及活动的最优活动时间 $O_i\ (O_i>0, i \in N)$应满足如下条件：

1) $O_i \geqslant D_i$。

2) 对任意 O_k，O_k 与其他 O_i 之间不发生冲突；$i, k \in \{1, 2, 3, \cdots, n\}$。

3) 覆盖 O_i 的活动的总参与人次最多。

3 基于 Bitmap 的活动时间冲突查询算法

为提高活动冲突查询算法的性能，本文作者以用户信息表构建 Bitmap 索引，使得每个有效用户时间区间的时刻对应 1 个 Bitmap，并设计相应的算法来支持活动冲突查询。

基于 Bitmap 的活动时间冲突查询算法需要 2 个输入：一是用户信息表中的所有用户以及对应的时间区间信息；二是查询的时间范围和根据多个活动的查询要求给定的多个持续时间。算法的输出是能使全部活动参与人次最大化的每个活动的最优活动举办时间和参与所有活动的总人次。

在查询之前先通过输入的用户信息表数据构造 Bitmap 索引，基于 Bitmap 索引的查询算法可以分为 2 个阶段：第 1 阶段遍历 Bitmap 索引得到满足各个活动持续时间的候选时间区间和候选用户集合；第 2 阶段对各个活动的候选时间区间进行组合优化，找出组合后所有活动的用户总人次最大并且时间上互不发生冲突的最优时间区间。

3.1 Bitmap 索引结构

本文中需要构建的 Bitmap 索引由用户信息表中用户时间区间的某一时刻以及该时刻对应的 Bitmap 位串 2 部分组成。先根据用户信息表中的用户输入顺序编制每个用户在索引比特位串中的位置 B_u。若 B_u 为 1，则表示当前时刻处于用户的某个空闲区间 I 内；若用户无空闲时间，则 B_u 为 0。根据表 1 中的用户信息可以得到 Bitmap 索引结构，如表 2 所示(其中 t_i 为时间，$i \in N$)。

表 2 Bitmap 索引结构表

Table 2 Bitmap index structure

时间	对应的 Bitmap 位串
t_1	[1100…01]
t_2	[1100…11]
t_3	[1010…00]
⋮	⋮

3.2 Bitmap 索引初始构造算法

构建 Bitmap 索引的过程如下：读取所有用户的时间序列数据和持续时间窗口大小，给每个用户标记用户位置 B_u，得到用户位置的映射表，并得到所有用户区间的开始和结束时刻。给每个时刻建立 1 个 Bitmap 位串并赋值，若用户的有效时间区间包含当前时刻，则将位串中该用户对应的比特位置 1，否则置 0。最后得到每个用户的 1 个时刻对应 1 个 Bitmap 位串的索引映射表。

上面构建好的 Bitmap 索引结构存在存储空间消耗巨大及索引长度过长的缺点。以 50 000 条记录和近万用户数为例，构建 Bitmap 索引需要超过 1 G 的存储空间。查询算法中需要得到其中 2 个时刻对应的 Bitmap 位串进行按位与运算，并提取“1”的比特位对应的用户，加入候选用户集。如果不进行压缩处理，则需要遍历近万个位置和用户，会大大降低查询的时间效率，因此，很有必要对上述 Bitmap 索引结构进行压缩操作。下面将 Bitmap 位串按游程压缩，即以连续相同的比特值和相同位的数量取代原始子串。

3.3 Bitmap 查询算法

3.3.1 遍历 Bitmap 索引获取中间结果

遍历 Bitmap 索引获取中间结果为查询算法的第 1 阶段，具体又分为 3 步。

步骤 1：获取候选时间区间和候选用户集合。首先对所有用户在线时间区间内所有起始、结束时刻按时间排序，构成时刻表。遍历时刻表，若当前时刻是某用户在线时间区间的起始时刻，则从这个起始时刻往后搜寻第 1 个满足活动持续时间区间的终止时刻，从而以起始时刻、结束时刻构成 1 个候选时间区间。将候选时间区间的起始、结束时刻对应的用户位串进行压缩，并 2 串进行按位与运算，遍历结果串中所有值为 1 的比特位，提取对应的用户，加入候选用户集。

步骤 2：过滤候选用户集中的无效用户。步骤 1 得到的候选用户集合中可能存在某些无效用户，其空闲时间区间不能完全覆盖候选区间。本步骤主要过滤无效用户，处理逻辑如下：对每个候选用户设置 1 个初值为 0 的计数器，从头开始检查该用户的所有空闲

区间集合，判断是否能覆盖候选区间的起始时刻。1) 若当前区间不能覆盖候选区间的起始时刻，则计数器加 1，并检查当前区间是否包含候选区间的结束时刻。若是，则判定该用户为无效用户，保存此时的计数器值，结束对该用户的区间检查。若否，则继续检查该用户的下 1 个区间。2) 若当前区间能覆盖候选区间的起始时刻，则继续判断它是否覆盖候选区间的结束时刻。若是，则该用户不是无效用户，保存此时的计数器值，结束对该用户区间集合的遍历。若否，则该用户是无效用户。

通过上述过滤方法，就能够过滤掉候选用户集合中的无效用户。

步骤 3：调整候选时间区间。当完成前 2 个步骤后，遍历排序后的时刻列表找到起始时刻后的第 2 个结束时刻。重复步骤 1 和步骤 2，得到新的候选区间及候选用户集合。比较前后 2 次的候选用户集合，若元素个数相同，则将初次候选区间的结束时刻替换为新候选区间的结束时刻，然后，继续找到时刻表候选区间起始时刻中的第 3 个结束时刻，继续重复步骤 1 和步骤 2，直到后 1 个候选用户集合的元素个数小于前 1 个候选用户集合的元素个数为止，最终得到优化后的候选时间区间 I_c 和对应的候选用户集合 U_c。候选区间和候选用户集合映射表如表 3 所示。其中，I_{c_i} 为不同的候选区间，U_{c_i} 为不同的候选用户，$i \in N$。

表 3　候选区间和候选用户集合映射表

Table 3　Candidate interval and candidate user set map

候选区间	候选用户集
I_{c_1}	$\{U_{c_1}, U_{c_2}, U_{c_3}, U_{c_4}\}$
I_{c_2}	$\{U_{c_2}, U_{c_3}, U_{c_4}, U_{c_5}\}$
I_{c_3}	$\{U_{c_1}, U_{c_2}, U_{c_3}\}$
I_{c_4}	$\{U_{c_3}, U_{c_4}\}$

3.3.2　冲突区间组合优化算法

第 1 阶段查询得到多个活动的候选时间区间和对应的候选用户集合。为了在特定时间范围内得到所有活动的具体举办时间段，并保证参与所有活动的总人次最大化，需要通过第 2 阶段对结果组合优化。

例如，有 3 个持续时间分别为 1，2 和 3 h 的活动，首先需要进行 3 次独立查询，通过遍历 Bitmap 索引结构分别得到 3 个活动对应的候选区间和候选用户集合，然后对这 3 个活动对应的候选区间和候选用户集合进行组合优化，即分别从 3 个活动对应的候选时间区间集合中选取 3 个候选区间，在保证所选 3 个候选时间区间在时间上不发生冲突的情况下，使得 3 个候选区间对应的候选用户集合中的用户数加起来最大。

活动冲突组合优化算法流程如下。

算法 1：活动冲突组合优化算法

输入：每个活动的候选区间和候选用户集合映射表 T_{c_i} (i=1, 2, 3, …)，表中每个候选区间对应覆盖它的候选用户数量。

输出：结果映射表 T_{op}，表中多个不冲突活动的具体举办时间对应这些活动的最大的总参与人次。

$P=\varnothing$, $L_I=\varnothing$, $m_{ax}=0$, $S=0$
FOR I_{c_i} IN T_{c_i}
　　L_I.add(I_{c_i})
　　$S=S+T_{c_i}$.get(I_{c_i})
　　IF $S > m_{ax}$
　　　IF not Conflicted(L_I)
　　　　P.put(L_I, S)
　　　　$m_{ax} = S$
　　$T_{op}=\varnothing$
　　FOR l IN P
　　　IF P.get(l) == m_{ax}
　　T_{op}.put(l, m_{ax})
RETURN T_{op}

其中，P 为方案中间结果；m_{ax} 为最大总参与人数；S 为当前最大总参与人数；I_{c_i} 为第 i 个活动的候选时间区间；L_I 为候选活动列表，$i \in N$。

4　实验评估

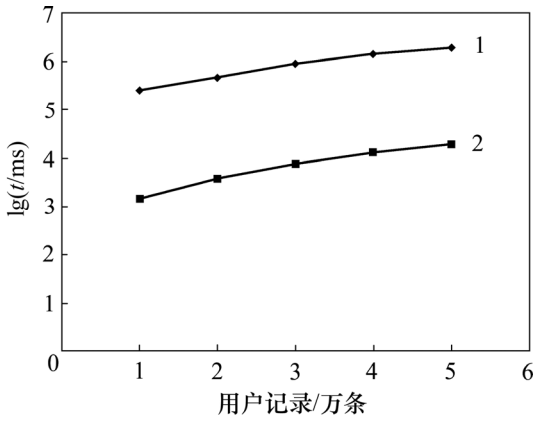
下面针对基于 Bitmap 索引的活动冲突查询算法的性能指标进行实验评估。本实验的原始数据来自用户行为日志，包括 8 628 个用户的 50 000 条时间序列数据(时间跨度为 1 个月)。在实验过程中，根据实验的影响因素对原始数据进行调整。首先在单个活动查询请求的情况下，将 Bitmap 索引算法与传统的滑动时间窗口方法进行比较。采用控制变量的方法即改变其中 1 个影响因素并保持其他影响因素不变，对 2 种查询算法进行实验对比分析。然后，通过活动数量、活动持续时间和时间范围这几个因素来研究基于 Bitmap 的活动时间冲突查询算法在活动冲突查询请求下的性能，探究其在不同影响因素下的效果以及算法在不同阶段所消耗的时间。本实验硬件如下：2.70 GHz 双核英特尔酷睿 i5 处理器，8 GB 内存，128 GB 的固态硬盘，操作系统为 MacOS Sierra 10.12.1。

4.1　单活动场景下 2 种查询算法性能比较

活动数量为 1 个是活动冲突问题的特殊情况，此时活动不发生冲突。为了比较 Bitmap 查询算法与普通

的滑动时间窗口算法，本文研究在单活动查询请求情况下不同数据量(用户记录)和活动持续时间对 2 种查询算法的影响。

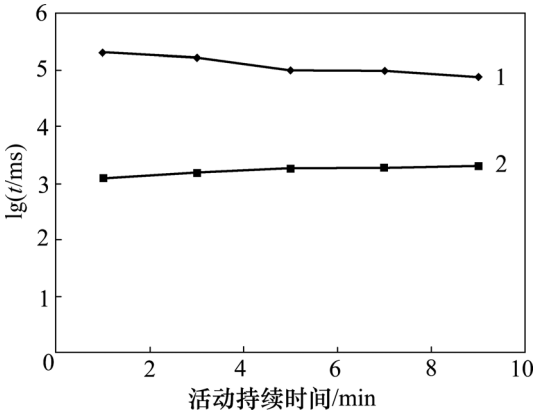
单活动场景下不同数据量及不同持续时间下滑动时间窗口算法和基于 Bitmap 索引算法的性能比较分别如图 1 和图 2 所示。算法运行时间单位为 ms，由于 2 种算法耗时相差过大，取运行时间的对数 lg t 进行比较。



1—滑动时间窗口算法；2—Bitmap 查询算法。

图 1 不同数据量下 2 种查询算法性能对比

Fig. 1 Performance comparison of two kind of query algorithms with different data size



1—滑动时间窗口算法；2—Bitmap 查询算法。

图 2 不同持续时间下 2 种查询算法性能对比

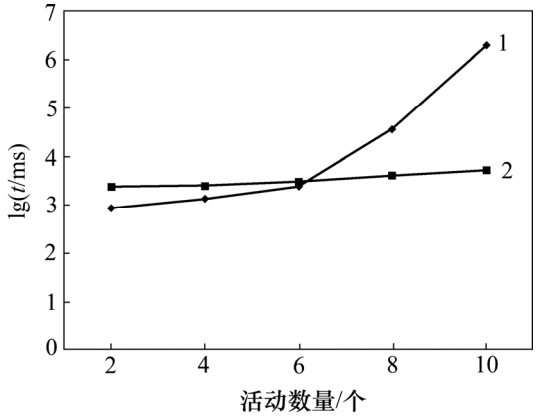
Fig. 2 Performance comparison of two kind of query algorithms with different duration time

从图 1 可以看出：随着用户数据量不断增大，2 种查询算法的查询耗时都逐渐增加。但 Bitmap 查询算法耗时约为滑动时间窗口算法耗时的 1/100。这是因为普通的滑动窗口算法需要遍历每 1 个粒度的时间点。例如，当时间粒度为 1 s 时，用户信息表中的时序数据范围从 12:11:00 到 12:12:00，滑动时间窗口将会进行 60 次移动。然而，Bitmap 索引算法中最外面的循环只需要遍历用户信息表所有开始时间，因此，耗时较少。然而，随着数据量增加，用户信息表的用户时间区间数量也会增加，从而导致 2 种查询算法的耗时增加。

从图 2 可以看出：滑动时间窗口算法对活动持续时间的变化不敏感，而 Bitmap 查询算法的耗时则随着活动持续时间增加而缓慢增加。然而，Bitmap 索引算法的查询时间还是比滑动时间窗口算法的查询时间小 2 个数量级。

4.2 活动冲突场景下 2 种查询算法性能比较

在实验中，活动持续时间窗口和查询时间范围固定，仅逐步增加活动数量来探究 2 种算法执行 1 次查询所需的运行时间。不同活动数量下 2 种查询算法的性能对比如图 3 所示。从图 3 可以看出：随着活动数量不断增大，Bitmap 查询算法的查询耗时缓慢增加，而滑动时间窗口查询算法的查询时间则在某个活动数量后急剧增加。最终，Bitmap 查询算法的查询耗时比普通的滑动时间窗口算法的查询耗时快近 2 个数量级。当活动数量较少时，2 种算法的运行耗时都很短并且很接近，滑动时间窗口算法的耗时比 Bitmap 查询算法的要少，这是因为此时冲突时间组合部分的耗时会随着活动数量不断增加而增加。滑动时间窗口算法在冲突组合部分得到的候选时间要比 Bitmap 查询的算法更多，因此，候选区间匹配的次数也更多，这会增加耗时。



1—滑动时间窗口算法；2—Bitmap 查询算法。

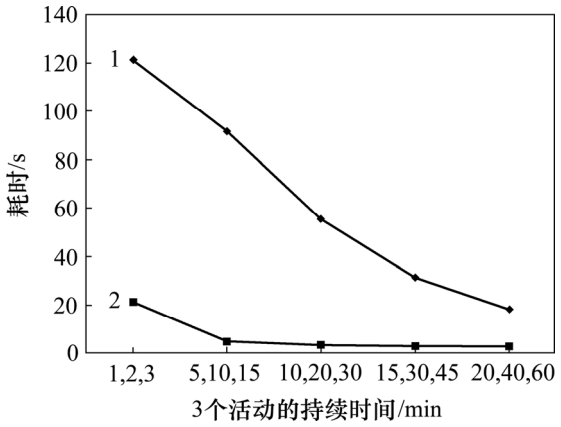
图 3 不同活动数量下 2 种查询算法的性能对比

Fig. 3 Performance comparison of two kind of algorithms with different numbers of activities

当活动数量固定为 3 个时，在不同活动持续时间下，2 种查询算法的耗时比较如图 4 所示。从图 4 可以看出：当活动持续时间不断增加时，Bitmap 查询算法的运行耗时一直维持在很小的范围内。而在活动持续时间较短时，滑动时间窗口算法耗时较长，随着活

动时间增加，滑动时间窗口算法的耗时也不断减少，但还是略多于 Bitmap 查询算法的耗时。这是因为当活动持续时间比较短时，满足它的有效用户时间区间数量会比活动持续时间长时的要多。而单次活动查询情况下滑动时间窗口的遍历过程是按每个时间粒度来遍历的，由于 Bitmap 查询算法索引结构逻辑与运算速度较快，且其遍历的是每个有效用户的开始时间点，因此，运行耗时会比滑动时间窗口算法的耗时要少。

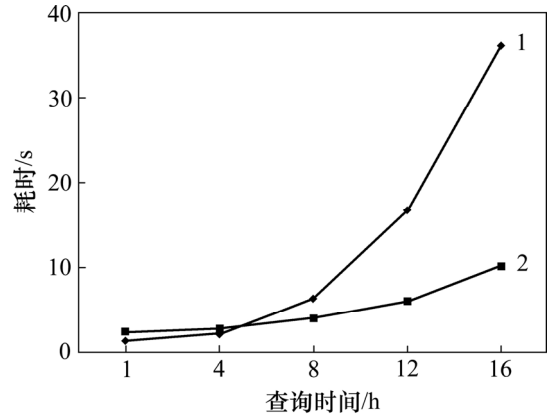
图 5 所示为不同查询时间范围下 2 种查询算法的性能比较。从图 5 可以看出：随着时间范围不断扩大，滑动时间窗口的运行耗时急剧增加，而 Bitmap 查询算法的运行耗时缓慢增加。当时间范围扩大到一定程度后，Bitmap 查询算法的性能明显优于滑动时间窗口算法的性能。查询时间范围扩大会导致有效用户时间区间数量增多，这会增加 2 种算法的耗时。随着时间范围增大，Bitmap 查询算法中遍历 Bitmap 索引阶段的时间增长速度远小于比滑动时间窗口法的增长速度。



1—滑动时间窗口算法；2—Bitmap 查询算法。

图 4　不同活动持续时间下 2 种查询算法的性能对比

Fig. 4　Performance comparison of two kind of algorithms in different activity duration



1—滑动时间窗口算法；2—Bitmap 查询算法。

图 5　不同查询时间范围下 2 种查询算法的性能对比

Fig. 5　Performance comparison of two kind of algorithms in different query time range

4.3　不同因素对基于 Bitmap 的活动时间冲突查询算法性能的影响

下面研究不同因素对 Bitmap 查询算法执行 1 次查询过程中遍历 Bitmap 索引阶段和冲突区间组合优化阶段运行时间的影响。在该算法的实际应用中，Bitmap 索引的是在线下提前构造的，当接收 1 个活动冲突查询请求时，Bitmap 查询算法再根据构造好的 Bitmap 索引结构进行查询，因此，在算法的查询过程中不考虑 Bitmap 索引的构造时间。下面针对活动数量、活动的持续时间和查询时间这 3 个因素对 Bitmap 查询算法性能的影响进行分析。

图 6 所示为活动数量对 Bitmap 查询算法性能的影响。从图 6 可以看出：随着活动数量不断增多，Bitmap 查询算法的运行耗时也缓慢增加。图 6 中遍历 Bitmap 索引的时间先逐渐增加后减少也说明活动数量对 Bitmap 索引遍历阶段(第 1 阶段)的影响不大。而当活动数量超过 4 个时，冲突区间组合优化阶段(第 2 阶段)耗时急剧增加，说明该阶段对活动数量的变化非常敏感。这是因为，活动数量增多会直接导致冲突区间组合优化的循环次数增加，影响算法的整体查询时间。

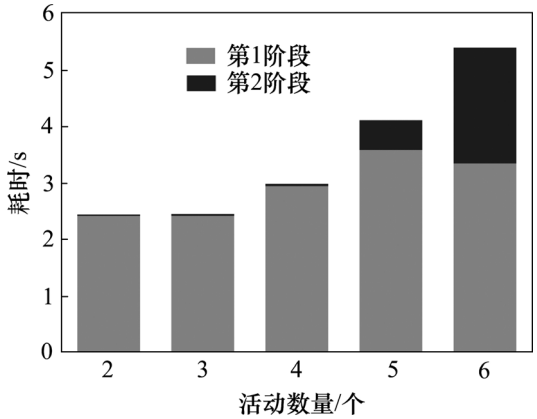


图 6　活动数量对 Bitmap 查询算法性能的影响

Fig. 6　Effect of activity number on performance of Bitmap query algorithm

图 7 所示为不同活动持续时间对 Bitmap 查询算法性能的影响。从图 7 可以看出：随着活动持续时间不断增加，算法的运行耗时迅速减少，同时第 1 阶段的耗时也迅速减少。而第 2 阶段的运行耗时基本不变。在活动持续时间较短时，由于满足活动持续时间的有效用户时间区间会增加，所以，每个活动得到的候选时间区间个数会增加。而这将导致第 2 阶段的冲突判定次数增加，从而使查询时间增加。

图 8 所示为不同查询时间对 Bitmap 查询算法性能的影响。从图 8 可以看出：随着查询时间范围不断扩

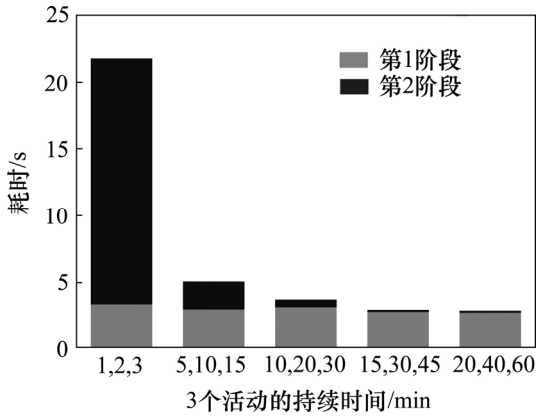


图 7 不同活动持续时间对 Bitmap 查询算法性能的影响

Fig. 7 Effects of different activity time duration on performance of Bitmap query algorithms

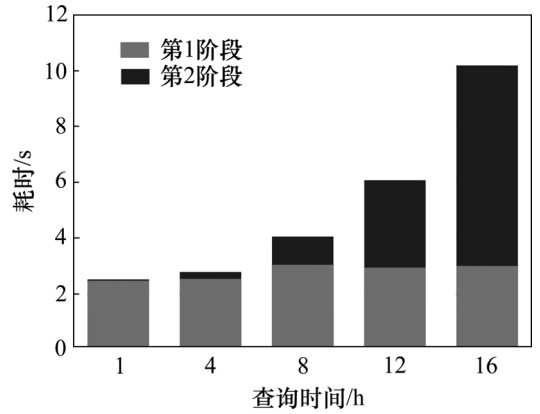


图 8 不同查询时间范围对 Bitmap 查询算法性能的影响

Fig. 8 Effects of different query time range on performance of Bitmap query algorithms

大，Bitmap 查询算法的运行耗时也不断增加。第 1 阶段的耗时也不断增加，而且该阶段在整个查询耗时的占比也不断提高；而第 2 阶段的运行耗时基本稳定不变。查询时间范围扩大会导致有效用户时间区间数量增大，直接导致第 1 阶段得到的候选时间区间变多，所以，第 2 阶段的冲突判定次数也会变多，从而影响查询性能。

5 结论

1) 提出基于 Bitmap 的活动时间冲突查询算法。该方法底层采用 Bitmap 数据结构对用户区间信息数据构建索引；通过遍历索引并结合冲突区间组合优化来提高查询性能，得到活动时间的全局最优化方案。

2) 利用真实用户数据集，通过控制变量法对 Bitmap 索引算法和滑动时间窗口算法的运行效率进行比较。在单活动场景下，Bitmap 索引算法比滑动时间查询算法约快 100 倍。

3) 基于 Bitmap 索引的活动时间冲突解决方法能够更好地满足含冲突的大规模活动组织查询的时效性要求。

参考文献：

[1] 李永峰, 周兴社, 杜可君, 等. 基于时间约束网络的智能活动规划[J]. 计算机科学, 2011, 38(2): 179−183.
LI Yongfeng, ZHOU Xin, DU Kejun, et al. Activity planning based on temporal constraint network[J]. Computer Science, 2011, 38(2): 179−183.

[2] CHAN H L, LAM T W, LEE L K, et al. Continuous monitoring of distributed data streams over a time-based sliding window[J]. Algorithmica, 2012, 62(3/4): 1088−1111.

[3] CHAN C Y, IOANNIDIS Y E. Bitmap index design and evaluation[J]. ACM SIGMOD Record, 1998, 27(2): 355−366.

[4] CHEN Zhen, WEN Yuhao, CAO Junwei, et al. A survey of bitmap index compression algorithms for big data[J]. Tsinghua Science and Technology, 2015, 20(1): 100−115.

[5] SESHADRI V, HSIEH K, BOROUM A, et al. Fast bulk bitwise AND and OR in DRAM[J]. IEEE Computer Architecture Letters, 2015, 14(2): 127−131.

[6] FENG Wei, ZHANG Chao, ZHANG Wei, et al. STREAMCUBE: hierarchical spatio-temporal hashtag clustering for event exploration over the twitter stream[C]//International Conference on Data Engineering (ICDE). Seoul, South Korea: IEEE, 2015: 1561−1572.

[7] FENG Jun, LU Chunyan, WANG Ying, et al. Sketch RR-tree: a spatio-temporal aggregation index for network-constrained moving objects[C]// Proceedings of the 3rd International Conference on Innovative Computing Information and Control(ICICIC). Washington D C, USA: IEEE Computer Society, 2008: 1899−1903.

[8] JOHN A, SUGUMARAN M, RAJESH R S. Indexing and query processing techniques in spatio-temporal data[J]. ICTACT Journal on Soft Computing, 2016, 6(3): 1198−1217.

[9] KLINE N, SNODGRASS R T. Computing temporal aggregates[C]// Proceedings of the Eleventh International Conference on Data Engineering(ICDE). Washington D C, USA: IEEE Computer Society, 1995: 222−231.

[10] KLINE R N. Aggregation in temporal databases[D]. Tucson, USA :University of Arizona, Department of Computer Science , 1999:213.

[11] KIM J S, KANG S T, KIM M H. On temporal aggregate processing based on time points[J]. Information Processing Letters, 1999, 71(5/6): 213−220.

[12] MOON B, LOPEZ I F V, IMMANUEL V. Efficient algorithms for large-scale temporal aggregation[J]. IEEE Transactions on Knowledge and Data Engineering, 2003, 15(3): 744−759.

[13] LIAO T W. Clustering of time series data—a survey[J]. Pattern Recognition, 2005, 38(11): 1857−1874.

[14] LI Yinan, HE Bingsheng, LUO Qiong, et al. Tree indexing on flash disks[C]// Proceedings of the 25th International Conference on Data Engineering(ICDE).Washington D C, USA: IEEE Computer Society, 2009: 1303−1306.

(编辑 伍锦花)