嘉宾介绍

|  |  |
| --- | --- |
| 王晓玲 | **嘉宾介绍**： 华东师范大学计算机科学与软件工程学院，教授，博士生导师，副院长。教育部新世纪人才计划获得者，上海市教育部科技创新计划获得者。她于1997、2000和2003年在东南大学分别获得计算机软件学士、硕士和博士学位。2003年任职于复旦大学计算机科学系，于2006年晋升为副教授。2008年加入华东师范大学软件学院。研究方向为面向数据密集型计算的数据管理系统、数据服务技术及应用。她目前是中国计算机学会数据库专委会委员。  **演讲主题**：《PostgreSQL的Window分析函数源码优化》  Window(窗口)函数作为关系数据库领域中内数据库分析技术的一种解决方案, 其精妙的语义特征使其能代替自连接和相关子查询等完成传统复杂查询功能，现已被广泛应用到互联网应用的数据管理和分析中。在目前互联网应用步入大数据时代的背景下，针对高吞吐和实时响应等需求，已有的Window(窗口)函数的处理性能已经出现了瓶颈。在这个工作里，我们与EMC2的曹逾博士进行了合作，针对窗口函数的优化技术，提出了基于临时窗口和组共享思想的窗口函数优化方案，在尽量减少数据读取和计算次数的前提下，优化了窗口函数的执行过程，提升了函数的执行效率。通过与PostgreSQL数据库原有的窗口函数实现以及现有商业数据库进行性能上的对比，验证了新算法的有效性。 |

面向MAX/MIN优化的SQL Window函数处理

马建松1) 王晓玲1) 曹逾2)

1)(华东师范大学软件学院 数据科学与工程研究院, 上海)

2)( EMC Labs Beijing, 北京)

摘 要 随着互联网用户数量的不断增长，互联网应用逐渐进入了大数据时代。如何存储和分析这些大数据成为了互联网应用中的难题。融合数据存储和数据分析的内数据库分析(In-Database Analytics)技术，受到了越来越多企业和研究者的关注。Window(窗口)函数作为关系数据库领域中内数据库分析技术的一种解决方案, 其精妙的语义特征使其能代替自连接(Self Join)和相关子查询(sub Queries)等完成传统复杂查询功能，现已被广泛应用到互联网应用的数据管理和分析中。在目前互联网应用步入大数据时代的背景下，针对高吞吐和实时响应等需求，已有的Window(窗口)函数的处理性能已经出现了瓶颈。本文基于PostgreSQL数据库中原有MAX/MIN Window(窗口)函数执行框架，提出了一种基于临时窗口的专门针对MAX/MIN函数优化的方法，来优化SQL Window查询针对MAX/MIN函数的处理，并给出了查询代价的分析模型，从理论上证明了该算法的性能。通过与现有商业数据库SQLServer进行性能上的对比，验证了该方案的有效性。

关键词 Window函数；查询处理；性能优化；MAX/MIN；

Optimizing the MAX/MIN of Window Functions

MA Jian-Song1)  WANG Xiao-Ling1) CAO Yu2)

1)(Institute for Data Science and Engineering, Software Engineering Institute, East China Normal University, Shanghai)

2)( EMC Labs Beijing, China)

**Abstract** With the growing number of Internet users, the Internet application gradually enters the era of big data. How to store and analyze the big data becomes a problem in the Internet application. In-Database Analytics as a technology that integrate data storage with data analysis, has attracted the attention of many enterprises and researchers. Window Function, as a solution of In-Database Analytics in the field of relational database, make it take the place of Self Join and Sub Queries to complete the traditional complex queries with its subtle semantic characteristics and is widely used in the current enterprise data management and analysis in the Internet application. Under the background of big data, Window Function has the bottleneck in the face of such demand as high throughput and real-time response. In this paper, we design a new algorithm dependent on Temporary Window for the MAX/MIN window functions contraposing the original framework in PostgreSQL. We design a query cost analysis model and theoretically proved the performance of the algorithm. We did the performance comparison between the new algorithm and the existing commercial database SQL Server and proved the effectiveness of the proposed algorithm.

**Key words** window function; query processing; performance optimization; MAX/MIN

# 引言

## 1.1 Window函数简介

随着互联网的普及和互联网用户数量的不断增加，互联网应用逐渐进入了大数据时代。大数据时代的来临，使的互联网应用面临着数据的爆发式增长。由于用户数据中包含了丰富的用户行为模式信息，因此，大数据量的用户数据对互联网应用来说变的越来越重要。如何存储和分析这些大数据成为了互联网应用中的难题。随着用户对数据处理效率要求的不断提高，融合数据存储和数据分析的内数据库分析(In-Database Analytics)技术，受到了越来越多企业和研究者的关注。Window(窗口)函数作为关系数据库领域中内数据库分析技术的一种解决方案,最初以扩展文档的形式被引入到SQL:1999，之后不久，SQL:2003就正式规范了Window(窗口)函数的标准，并在后续的标准版本中有所丰富与扩展。Window函数拥有SQL语句惯有的简洁构成模式，但是其精妙的语义特征使其能代替自连接(Self Join)和相关子查询(Sub Queries)等完成传统复杂查询的功能。

在数据处理中，窗口概念的引入，使得计算被应用到特定的数据集合(data set)上，或者说是一个窗口所包含的元组(tuple)之上。在此基础上，随之产生了一系列常规分析函数，例如，平均值(average)、累积求和(sum)、最大值(max）、最小值（min）、排序(ranking)、百分比(percentile)等。这些函数都可以用一条SQL语句精确、直观、有效的表达出来。

Window函数由一个分析函数和一个窗口定义子句构成。在标准SQL的定义中，使用窗口定义函数所作用于的上下文区间，窗口的具体规范则由一个OVER子句来定义。

例1. 给定一张员工工资表empsalary(如表1所示)，包含三个属性，empno、depno和salary(其中empno是主键)。empno指的是员工的员工编号(唯一的)，depno指的是部门编号，salary指的是员工工资。

如上是一条带Window函数的SQL查询语句。表示将所有员工按部门号划分，按员工编号排序，将每位员工的工资与前后20位比较，得到工资最高的员工的工资。

表1 员工工资表(empsalary表)



例1中的SQL语句部分查询结果如下表(表2)中所示：

表2 SQL语句部分查询结果表



如例1中的SQL语句所示，Window函数包含一个分析函数(例1中的SQL语句中为MAX函数)和一个规定窗口大小的OVER子句。其中OVER子句中包含了三个组成部分：

1. PARTITION BY子句：PARTITION BY子句用来定义数据分区，形式为PARTITION BY *expr\_list*。后面的属性值*expr\_list*决定了数据表按照哪些属性进行划分，*expr\_list*的值相同的元组同属于一个分区。
2. ORDER BY子句：ORDER BY子句用来定义数据排序模式，形式为ORDER BY *order\_list*。后面的属性值*order\_list*决定了数据表按照哪些属性进行排序。
3. ROWS子句：ROWS子句用来定义Window函数中一个窗口的大小，形式为ROWS BETWEEN *pre\_value* and *post\_value*。其中*pre\_value*包含了UNBOUNDED PRECEDING(指窗口的起始位置为分区的第一个元组)、*value* PRECEDING(指窗口的起始位置为当前行的前*value*个元组)和CURRENT ROW(指窗口的起始位置为当前行)三种，*post\_value*包含了UNBOUNDED FOLLOWING(指窗口的终止位置为分区的最后一个元组)、*value* FOLLOWING（指窗口的终止位置为分区的后*value*个元组）和CURRENT ROW（指窗口的终止位置为当前行）三种。

数据表中的每一个元组作为当前行都对应一个窗口，窗口的大小由OVER子句确定，其定义跟当前行相关的一个元组的集合（元组的集合通常包括一个或者多个元组，也可以是整张数据表）。

现今，主流商业数据库系统中都实现了Window函数以支持数据分析任务，比如DB2的联机分析处理(online analytical processing，OLAP)函数，Oracle的解析函数(analytic function)和SQL Server的Window函数等。

随着互联网应用逐渐步入大数据时代，Window函数也逐渐被应用于各类互联网应用的数据管理和数据分析中，如商务智能的查询报表和各类分析应用。越来越多的企业级数据的查询和分析逐渐将旧的查询替换成Window函数的方案。Window函数在数据查询和分析中的应用使的查询处理更加高效，尤其是可以有效地消除效率低下的自连接(Self Join)和相关子查询(Sub Queries) [1，2]，而且查询处理过程中可以尽可能的减少临时表的使用。

尽管Window函数在数据查询和分析中可以尽可能的提高查询效率，其执行框架也设计的足够精简，但由于其应用场景的多样性使其在实现时并没有得到足够的优化。在目前互联网应用步入大数据时代的背景下，针对高吞吐和实时响应等需求，已有的Window函数的处理性能已经出现了瓶颈。因此，针对特定的数据分析场景，需要设计出更适合Window函数执行的优化方法。

## 1.2 相关工作

当前，主流的数据仓库厂商，如Teredata，Greenplum等，针对高吞吐量、实时响应(比如作弊检测、风险控制)等应用都提供了不同程度的内数据库分析支持。 Window函数针对每个窗口都会计算出一列额外的属性。GROUP BY及其三类扩展：GROUP SETS、ROLLUP和CUBE，是将数据表中的数据按照不同的划分方式进行分组，然后对每个组内的元组进行各种数据操作，其操作方式跟Window 函数类似。但是，如[3,4,5]等关于GROUP BY及其扩展的优化工作，并不适用于Window 函数的执行框架。主要有两方面的原因：1）GROUP BY及其扩展的计算模式并不是和Window 函数的计算模式一样。Window 函数在保留原始数据的基础上，计算出额外属性列，输出结果可以包含原始表的详细信息，而GROUP BY及其扩展是在数据表分组的基础上得出数据操作信息，因此GROUP BY及其扩展的输出最多保留分组信息。2）GROUP BY及其扩展中，数据操作是针对整个分组的，而Window函数的语意特性更强，在分组的基础上可以指定任意物理上或逻辑上的窗口大小。

数据流领域也引入了窗口的概念。在数据流领域，由于数据是持续不断产生的，因此，处理的数据具有数据量大和持续更新等特性。在数据流领域，由于数据其特有的特征，将整个数据作为操作的对象是基本不可能的，常用的数据操作方法是使用滑动窗口(sliding window)来处理最新到达的数据，并在窗口范围内进行数据上的操作。针对数据流上的窗口操作，尤其是聚合(aggregate)操作，已经有了大量的研究工作。如[6]对经典的top-k问题做了研究，[7]则在滑动窗口中加入了语义信息。由于数据流处理过程中，其数据访问和结果要求与关系数据库存在差异，因此许多优化策略在关系数据库中并不适用。[8]则针对滑动窗口计算过程中存在的很多重叠部分，提出了一种基于计算共享思想的优化方法。在连续查询领域(continuous query)，[9，10，11，12]也是基于这种计算共享思想对计算过程进行了相应的优化。

重排序和顺序调用是Window函数执行过程中的两个重要阶段。[13，14，15，16, 17]是重点关注Window函数执行优化工作的几篇文献。其中，文献[13]提出了一种基于全排序(full sort)的重排序方法，文献[14]则在此基础上提出了更高效的哈希排序(hash sort)和分段排序(segment sort)。[15]则从排序顺序(sort order)和排序共享(sort share)的角度，提出了一种协同排序技术，其排序代价极为接近全局最优。[16]基于窗口分组共享的思想针对顺序调用阶段进行了优化，[17]基于segment树共享的思想针对顺序调用阶段进行了优化，都是一个通用框架，但不能保证每种类型的函数处理都能达到最优。

## 1.3 知识预定义

为了尽可能全面、细致的量化Window函数在数据库中的执行过程的消耗，在描述Window函数的消耗模型时需要用到以下几个参数：

表1 参数表

|  |  |
| --- | --- |
| 参数 | 参数释义 |
|  | 数据表中元组个数 |
|  | 一个分区中的第个窗口 |
|  | 窗口的大小 |
|  | 窗口的起始位置 |
|  | 窗口的终止位置 |
|  | 分区中第个元组的位置 |
|  | 分区中第个元组的值 |
|  | 一个元组的数据读取和计算消耗 |
|  | 更新临时窗口信息的消耗 |

# Window函数执行过程

## 2.1 两阶段执行框架

Window函数的执行过程被分为两个部分：1）重排序阶段：根据PARTITION BY子句和ORDER BY子句将表进行划分和重排序。2）顺序调用阶段：得到重排序后的表，对窗口内的元组依次调用转移函数(transition function)，依次得到转移值(transition value)，拥有最终计算函数(final function)的Window函数通过调用最终计算函数得到每个窗口的最终结果值。

在重排序阶段，MAX/MIN Window函数跟其它Window函数一样，主要是针对PARTITION BY子句和ORDER BY子句对表进行分割和重排序。由于MAX/MIN Window函数并没有最终计算函数，因此每个窗口得到的最终转移值即为该窗口的最终结果值。因此，对于MAX/MIN Window函数，在顺序调用阶段，主要是对重排序后的表中的每一个窗口中的元组去顺序调用转移函数求转移值的过程。

## 2.2 顺序调用阶段的执行过程

顺序调用阶段，主体的计算过程是由转移函数完成的。对于像AVG等拥有最终计算函数的Window函数，最终结果是由最终计算函数获得。对于像MAX/MIN等函数，并不拥有最终计算函数，其最终结果就是最终获得的转移值。



图1 PostgreSQL中MAX/MIN Window函数在顺序调用阶段的执行过程

定义1. 窗口(Window)：由OVER子句定义的包含一系列元组的集合(包含了一个起始位置和一个终止位置，并且拥有其对应的转移值)，。其中指一个分区中的第个窗口，指窗口的起始位置，指窗口的终止位置，指窗口的转移值。

对于例1中的SQL查询，PostgreSQL中的Window函数在顺序调用阶段的执行过程如图1所示(桔黄色的元组表示此窗口的当前行，黑色实体箭头表示转移值的计算过程)：对于第一个窗口，窗口的起始位置为，终止位置为。转移函数依次作用于窗口中的每一个元组之上，获得的最终转移值即是窗口要求的最终结果值。对于窗口，起始位置为，终止位置为。相对于窗口来说，只比其多出了位置处的这一个元组。因此，对于窗口来说，并不需要重新计算至的元组的转移值。只需将窗口的转移值赋予窗口的转移值，并计算处的元组的转移值，得到的最终转移值即是窗口要求的最终结果值。从窗口开始，窗口的起始位置不再是，开始逐渐依次增加。对于窗口，计算过程与窗口类似，转移函数依次作用于窗口中的每一个元组之上，最终的转移值即是要求的最终结果值。后面的、等窗口的计算过程同窗口类似。

算法1 ***PostgreSQL顺序调用算法***

输入: *经过重排序的表T’*

输出: *每一个元组所对应的窗口的MAX/MIN函数值*

1. **FOR** 表*T’中的每一个分区P* **DO**
2. **FOR** 分区P中的每一个窗口**DO**
3. 初始化、、;
4. **IF**  **THEN**
5. ;
6. **FOR** each row in (] **DO**
7. transfunc();
8. **ELSE**
9. **FOR** each row in [] **DO**
10. transfunc();
11. **RETURN** ;

PostgreSQL中的MAX/MIN Window函数，在顺序调用阶段，转移函数执行过程的具体算法如算法1所示：首先，初始化当前窗口的参数(、、)，并将读指针置于窗口的起始位置。当前窗口的起始位置与上一个窗口的起始位置相比，如果位置不相同，则遍历窗口内的所有元组，并计算其相应的转移值(第8-10行)。如果位置相同，则只需遍历与上一个窗口相比新增的元组，并计算其相应的转移值(第4-7行)。

## 2.3 消耗模型

为了更好的发现MAX/MIN Window函数在顺序调用阶段执行过程中存在的瓶颈，我们建立一个量化模型去刻画其在数据库执行过程中的消耗。

首先，我们假设表中共有个元组()，个元组处在同一分区中，且在Window函数顺序调用阶段的执行过程中，每一个元组进行数据读取和转移函数计算的总消耗为。

由于表中的每一个元组作为当前行时都会唯一的确定一个窗口，因此，包含有个元组的表，在Window函数执行过程中共有个窗口的函数值需要计算。在此，我们假设个窗口的大小都是，且每个窗口的起始位置相对于上一个窗口都向下平移一个元组。则传统计算框架中，Window函数在顺序调用阶段的消耗为：

从中可以看出，在每一个元组的数据读取和计算消耗固定的情况下，执行过程中的瓶颈主要存在于元组重复的进行数据读取和计算。

# 基于临时窗口的MAX/MIN Window函数优化

MAX/MIN Window函数是不拥有最终计算函数的Window函数。因此，每个窗口的最终转移值即是其要求的Window函数值。优化MAX/MIN Window函数的核心思想是在函数执行过程中维持一个临时窗口，其中包含了此临时窗口对应的临时转移值，用于共享计算。以此来减少在顺序调用过程中重复进行的数据元组的读取和计算消耗。

## 3.1 优化执行过程

定义2. 临时窗口(Temporary Window)：临时产生的包含一系列元组的集合(包含了一个起始位置和一个终止位置，并且拥有其对应的临时转移值)，。其中指临时窗口的起始位置，指临时窗口的终止位置，(Temporary Transition Value)临时窗口的临时转移值。



图2 计算过程

对于例1中的SQL查询，其优化算法的执行过程如图2所示(图中紫色元组为每个窗口的最大值所处位置)：对于第一个窗口，转移函数依次作用于窗口中的每一个元组之上，与此同时，并记录下最大转移值的位置()来作为临时窗口的起始位置()，记录下窗口的终止位置()作为临时窗口的终止位置()，以及记录下最大转移值作为临时窗口的临时转移值()。对于窗口，起始位置为，终止位置为。相对于窗口来说，只比其多出了位置处的这一个元组。因此，对于窗口来说，并不需要重新计算至处的元组的转移值。只需将窗口的转移值赋予窗口的转移值，并计算处的元组的转移值，得到的最终转移值即是窗口要求的最终结果值。

由于小于（假设小于），因此我们只需将临时窗口的终止位置()的值由更新为即可。后面的窗口的执行过程与窗口的计算过程类似。对于窗口，其起始位置与上一个窗口相比向下移动了一个元组，但是并没有超过记录的临时窗口的起始位置，因此决定了，窗口中的前20个元组中的最终转移值为记录下来的临时窗口的临时转移值()。对于前20行元组，我们无需重复的进行数据读取和调用转移函数，只需将临时窗口的临时转移值()赋给窗口的转移值()即可。接下来，我们只需对处的元组进行数据读取和调用转移函数，并更新临时窗口的终止位置。同理，窗口的计算过程也是与窗口的计算过程类似。但是，对于窗口来说，其起始位置()超过了临时窗口的的起始位置()，因此，临时窗口已经不再适用于窗口的计算过程。因此，窗口的计算过程与窗口的类似，转移函数依次作用于窗口中的每一个元组之上，与此同时，并记录下最大转移值的位置()作为临时窗口的起始位置()，记录下窗口的终止位置()作为临时窗口的终止位置()，记录下最大转移值作为临时窗口的临时转移值()。对于窗口，由于其起始位置()没有超过临时窗口的起始位置()，因此，窗口的计算过程与窗口类似。后面的窗口的计算过程以此类推。

计算过程中，不仅需要更新临时窗口的终止位置，同时还需要更新临时窗口的临时转移值和其起始位置。例如，对于例1中的SQL查询，我们假设前11个窗口的计算过程如图3中所示一样(即前21行中的最大值为的值)，并且的值大于的值。因此，对于窗口来说，由于其起始



图3 更新过程

位置()并没有超过临时窗口的起始位置()，因此，处到处的元组并不需要重复的进行数据读取和调用转移函数，只需对处的元组进行数据读取和调用转移函数即可。由于的值大于的值，因此我们需要将临时窗口的临时转移值更新为的值，并且将临时窗口的起始位置由更新为。对于窗口来说，其起始位置()没有超过临时窗口的起始位置()，因此，计算过程类似于窗口。同理，后面的窗口的计算过程以此类推。

PostgreSQL中的MAX/MIN Window函数在顺序调用阶段执行过程的具体算法如算法2所示：对于数据表中的每一个分区，初始化临时窗口的参数(、、)(第2行)。对于分区中的每一个窗口，初始化当前窗口的参数(、、)，并将读指针置于窗口的起始位置(第4行)。当前窗口的起始位置与上一个窗口的起始位置相比，如果位置相同，则只需遍历与上一个窗口相比新增的元组，并计算其相应的转移值。与此同时，如果转移值不等于临时窗口的临时转移值，就更新临时窗口的起始位置和临时转移值。所有元组遍历结束后更新临时窗口的结束位置(第5-12行)。如果位置不相同，则比较当前窗口的起始位置与临时窗口的起始位置。如果当前窗口的起始位置不大于临时窗口的起始位置，首先将临时窗口的临时转移值赋予当前窗口的转移值，并遍历临时窗口的终止位置后直到当前窗口的终止位置处的元组及计算相应的转移值。与此同时，如果转移值不等于临时窗口的临时转移值，就更新临时窗口的起始位置和临时转移值。所有元组遍历结束后更新临时窗口的结束位置(第13-20行)。如果当前窗口的起始位置大于临时窗口的起始位置，遍历窗口内的所有元组，并计算其相应的转移值。与此同时，如果转移值不等于临时窗口的临时转移值，就更新临时窗口的起始位置和临时转移值。所有元组遍历结束后更新临时窗口的结束位置(第21-27行)。

算法2 ***TW*** *(Temporary Window)****顺序调用优化算法***

输入: *经过重排序的表T’*

输出: *每一个元组所对应的窗口的MAX/MIN函数值*

1. **FOR** 表*T’中的每一个分区P* **DO**
2. 初始化;
3. **FOR** 分区*P*中的每一个窗口 **DO**
4. 初始化、、;
5. **IF** **THEN**
6. ;
7. **FOR** each row in (] **DO**
8. transfunc();
9. **IF THEN**
10. ;
11. ;
12. ;
13. **ELSE IF THEN**
14. ;
15. **FOR** each row in (] **DO**
16. transfunc();
17. **IF THEN**
18. ;
19. ;
20. ;
21. **ELSE**
22. **FOR** each row in [] **DO**
23. transfunc();
24. **IF THEN**
25. ;
26. ;
27. ;
28. **RETURN** ;

## 3.2 消耗模型

与PostgreSQL原有顺序调用阶段的消耗模型类似，我们假设表中共有个元组()，个元组处在同一分区中，且在Window函数的执行过程中，每一个元组的数据读取和计算消耗为。假设个窗口的大小都是，且每个窗口的起始位置都相对于上一个窗口向下平移一个元组。更新一次临时窗口信息的消耗为。

对于第一个窗口，我们需要将从窗口起始位置到窗口终止位置处的所有元组依次进行数据的读取和计算，并且更新临时窗口的信息。因此，对于第一个窗口的消耗为：

其中指的是每次遍历一个元组时需要更新临时窗口信息的平均概率。

对于后面的窗口，当窗口的起始位置没有超越临时窗口的起始位置时，共用临时窗口的信息(同时更新临时窗口的信息)。当窗口的起始位置超过临时窗口的起始位置时，无可避免的需要重新遍历窗口内的所有元组。因此。剩余窗口的消耗为：

其中指的是一个窗口需要重新遍历全部元组的平均概率。

总消耗为：

令，则得：

当时，。

当时，如果当前窗口从新完全遍历了窗口内的所有元组，则上一个窗口在遍历第二个元组时没有执行更新临时窗口信息的操作。如果当前窗口只遍历了窗口内的第二个元组，则上一个窗口在遍历第二个元组时一定执行了更新临时窗口信息的操作。由此可得，则

由此可以看出当越小，即重新遍历所有元组的窗口越少时越小，也就是当数据按照一种从小到大(或从大到小)排列时消耗越少。

当时，函数简写形式如下：

由于，。令，，，可得如下函数图像(图4)：其中横轴为，纵轴为，竖轴为。

随着窗口和数据量的逐渐增加，对整个函数下移的影响要远比其他三个参数对函数上移的影响大。因此，窗口越大，数据量越多，优化效果越明显。

由于指的是每次遍历一个元组时需要更新临时窗口信息的概率，指的是一个窗口需要重新遍历全部元组的概率，因此、反映了一种数据分布趋势。由以上分析可以看出，数据分布与取值函数越趋于一致，优化效果会越好。

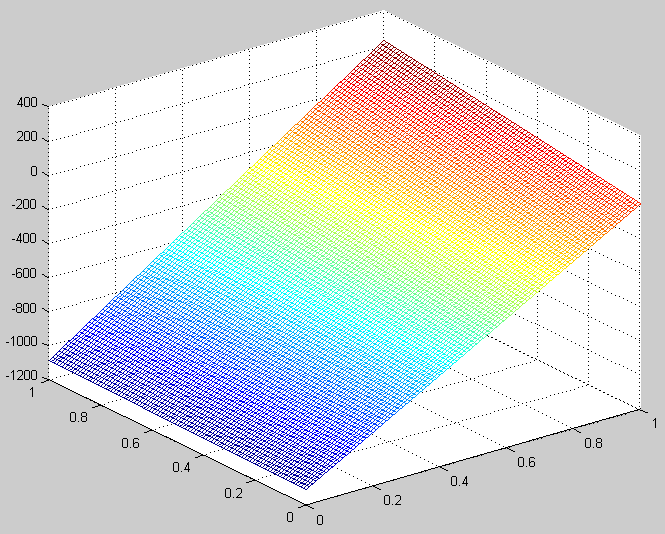


图4 消耗差函数图像

# 实验结果与分析

## 4.1 实验环境

我们更改了PostgreSQL 9.3.6的内核，实现了基于临时窗口的MAX/MIN Window函数优化算法。我们将其与Microsoft SQL Server 2012(Express Edition)共同部署在一台ThinkPad X220i电脑上。CPU型号是Intel(R) Core(TM) i3-2310M CPU@2.10Hz，内存4GB 1333MHz DDR3。所有数据库的工作内存(work\_mem)设置为500MB。

## 4.2 实验数据

本文的实验数据是使用TPC-H DBGEN生成的。指令如下：

生成的是TPC-H中的表“order”，指令1生成的数据集大小为170MB，包含1500000条元组，指令2生成的数据集大小是1.7GB，包含15000000条元组。

## 4.3 SQL查询语句

本文实验采用的SQL查询如下所示：

其中指的是MAX/MIN Window函数。我们通过更改参数来改变实验中窗口的大小。

## 4.4 实验对比简介

* PG：PostgreSQL中原有的顺序调用算法
* TW：TW (Temporary Window)顺序调用优化算法
* SQLServer：MS SQL Server 2012(Express Edition)

## 4.5 实验结果

本文的实验对比方法有三种，1）PostgreSQL数据库，原有执行框架；2）实现了基于临时窗口的MAX/MIN Window函数优化算法的PostgreSQL数据库；3）Microsoft SQL Server 2012(Express Edition)。



图5 MAX Window函数执行时间(170MB)



图6 MAX Window函数执行时间(1.7GB)

图5是本文实验采用的SQL查询在170MB数据集上的SQL查询执行时间对比图。其中横轴为，大小从10到500，纵轴为SQL查询执行时间，单位为秒(s)。由图中可以看出，采用优化算法TW的执行效率要远好于PG和SQLServer。而且，随着窗口的增大，TW的执行效率的提升越来越明显，其基本稳定在5s之内，相比之下，PG和SQLServer的执行时间已经增长到几百秒。

图6是本文实验采用的SQL查询在1.7GB的数据集上的SQL查询执行时间对比图。其中横轴为，大小从10到500，纵轴为SQL查询执行时间，单位为秒(s)。与图5结果类似，采用优化算法TW的执行效率要远好于PG和SQLServer。图6中三种数据库的SQL查询执行时间，相比于图5 都要有所上升，这是因为数据集的增大导致了查询处理过程中相应窗口的数量也随之增加，使的查询处理时间随之上升。



图7 MAX Window函数执行时间

图7是TW在不同数据集和不同的窗口大小下SQL查询执行时间结果图，其中横轴是，纵轴是SQL查询执行时间。由图7可以看出，基于临时窗口的MAX/MIN Window函数优化算法TW，在数据集大小一定的情况下，SQL查询的执行时间随着窗口大小的变化基本维持稳定，并不会出现较大幅度的改变。相比于PG、SQLServer在数据集一定的情况下，SQL查询执行时间随着窗口的增大而呈指数级增长，TW的这一特性，在数据处理中会更加有优势。

# 结束语

针对MAX/MIN SQL Window函数在顺序调用阶段的执行，本文提出了一种基于临时窗口的优化算法。利用执行过程中产生的一些临时结果，极大的避免了数据库中元组的重复读取和调用转移函数，从而大大提高了MAX/MIN SQL Window函数在顺序调用阶段的执行时间和效率。与现有的算法相比，该算法在保证结果正确的基础上极大提高了运行效率，其独有的特性在处理大数据时也有明显的优势。

与[16]中的组共享和[17]中的segment树共享思想相比，本文的优化算法又具有一定的局限性，因为并不是一个普适的Window函数优化算法，因此具有函数识别性，在进行实际部署时需要具备函数识别器。

未来，Window函数优化在保证结果正确性的前提下，会尽可能的考虑不同函数的执行特性，在普适性和效率之中做出最适合的取舍。

参 考 文 献

[1] Calisto Zuzarte, Hamid Pirahesh, Wenbin Ma, Qi Cheng, Linqi Liu, and Kwai Wong. Winmagic: subquery elimination using window aggregation. In Proceedings of the 2003 ACM SIGMOD international conference on Management of data, 2003, pages 652–656

[2] Srikanth Bellamkonda, Rafi Ahmed, Andrew Witkowski, Angela Amor, Mohamed Zait, and Chun-Chieh Lin. Enhanced subquery optimizations in oracle. In Proceedings of the VLDB Endowment, 2009, 2(2):1366–1377

[3] Sameet Agarwal, Rakesh Agrawal, Prasad Deshpande, Ashish Gupta, Jeffrey F. Naughton, Raghu Ramakrishnan, and Sunita Sarawagi. On the computation of multidimensional aggregates. In Proceedings of the 22th International Conference on Very Large Data Bases, 1996, pages 506–521

[4] Damianos Chatziantoniou and Kenneth A. Ross. Querying multiple features of groups in relational databases. In Proceedings of the 22th International Conference on Very Large Data Bases, 1996, pages 295–306

[5] Zhimin Chen and Vivek Narasayya. Efficient computation of multiple group by queries. In Proceedings of the 2005 ACM SIGMOD international conference on Management of data, 2005, pages 263–274.

[6] Cheqing Jin, Ke Yi, Lei Chen, Jeffrey Xu Yu, and Xuemin Lin. Sliding-window top-k queries on uncertain streams. The International Journal on Very Large Data Bases, 2010, 19(3):411–435

[7] Jin Li, David Maier, Kristin Tufte, Vassilis Papadimos, and Peter A. Tucker. Semantics and evaluation techniques for window aggregates in data streams. In Proceedings of the 2005 ACM SIGMOD international conference on Management of data, 2005, pages 311–322

[8] Jin Li, David Maier, Kristin Tufte, Vassilis Papadimos, and Peter A. Tucker. No pane, no gain: efficient evaluation of sliding-window aggregates over data streams. SIGMOD Record, 2005, 34(1):39–44

[9] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The cql continuous query language: semantic foundations and query execution. The International Journal on Very Large Data Bases, 2006, 15(2):121–142

[10] Shenoda Guirguis, Mohamed A. Sharaf, Panos K. Chrysanthis, and Alexandros Labrinidis. Optimized processing of multiple aggregate continuous queries. In Proceedings of the 20th ACM international conference on Information and knowledge management, 2011, pages 1515–1524

[11] Arvind Arasu and Jennifer Widom. Resource sharing in continuous sliding-window aggregates. In Proceedings of the Thirtieth international conference on Very large data bases, 2004, pages 336–347

[12] Yan-Nei Law, Haixun Wang, and Carlo Zaniolo. Relational languages and data models for continuous queries on sequences and data streams. ACM Trans. Database Syst., 2011, 36(2):8:1–8:32

[13] Srikanth Bellamkonda, Tolga Bozkaya, Bhaskar Ghosh, Abhinav Gupta, John Haydu, Sankar Subramanian, and Andrew Witkowski. Analytic functions in oracle 8i. Technical report, 2000.

[14] Yu Cao, Chee-Yong Chan, Jie Li, and Kian-Lee Tan. Optimization of analytic window functions. In Proceedings of the VLDB Endowment, 2012, 5(11):1244–1255

[15] Yu Cao, Ramadhana Bramandia, Chee-Yong Chan, and Kian-Lee Tan. Sort-sharing aware query processing. The International Journal on Very Large Data Bases, 2012, 21(3):411–436

[16] Jiansong Ma, Yu Cao, Xiaoling Wang, Chaoyong Wang, Cheqing Jin, Aoying Zhou. PGWinFunc: Optimizing Window Aggregate Functions in PostgreSQL and Its Application for Trajectory Data. In Proceedings of 31st IEEE International Conference on Data Engineering, 2015, pages: 1448-1451

[17] Viktor Leis, Kan Kundhikanjana, Alfons Kemper, Thomas Neumann. Efficient processing of window functions in analytical SQL queries. In Proceedings of the VLDB Endowment, 2015, [8(10)](http://dblp.uni-trier.de/db/journals/pvldb/pvldb8.html" \l "LeisKK015): 1058-1069