Journal of Computer Applications
计算机应用,2020,40(2):410－415
ISSN 1001-9081
CODEN JYIIDU
2020-02-10
http://www.joca.cn

文章编号:1001-9081(2020)02-0410-06
DOI:10.11772/j.issn.1001-9081.2019081908

TiDB的多索引访问优化

兰　海[1],韩　珂[1],申　砾[2],崔　秋[2],彭煜玮[1*]
(1.武汉大学 计算机学院,武汉 610072;　2.北京平凯星辰科技发展有限公司,北京 100096)
(*通信作者电子邮箱 ywpeng@whu.edu.cn)

摘　要:当查询条件涉及多个已建立索引的属性时,TiDB不能利用多个索引产生更优的执行计划。为了解决此问题,在研究现有数据库解决方案(如PostgreSQL和MySQL等)后,在TiDB中提出一种同时利用多个索引的新类型数据访问路径,称为MultiIndexPath。首先,设计算法生成一个查询可能的MultiIndexPath,并产生该路径的物理计划MultiIIndexPlan,然后计算物理计划的代价;其次,结合TiDB的架构与实现,提出MultiIndexPlan的通用执行框架;最后,当条件为合取范式时,提出Pipeline执行方案。整个工作基于TiDB 3.0实现并进行若干实验,结果表明:当条件为析取范式时,所提方案的性能比原TiDB至少有一个数量级提升;当条件为合取范式时,性能也优于原TiDB。

关键词:分布式数据库;TiDB;优化器;索引
中图分类号:TP311.132　　**文献标志码:**A

Accessing optimization with multiple indexes in TiDB

LAN Hai[1],　HAN Ke[1],　SHEN Li[2],　CUI Qiu[2],　PENG Yuwei[1*]
(1.*School of Computer Science*,　*Wuhan University*,　*Wuhan Hubei* 610072,　*China*;
2.*PingCAP*,　*Beijing* 100096,　*China*)

Abstract: When the query condition involves multiple indexed attributes, TiDB cannot effectively use multiple indexes to generate a more efficient execution plan. After studying the existing solutions of databases, such as PostgreSQL and MySQL, a new type of data access path using multiple indexes simultaneously was proposed in TiDB to solve the problem, namely MultiIndexPath. Firstly, a possible access path named MultiIndexPath for a certain query was generated and its physical plan named MultiIIndexPlan was created, then the cost of this plan was calculated. Secondly, the general execution framework of MultiIndexPath was proposed after combining the architecture and implementation of TiDB. Finally, the Pipeline execution plan was proposed with the condition of conjunctive normal form. The whole work was implemented based on TiDB 3.0 and several experiments were conducted. Experimental results show that the performance of the proposed scheme is improved by at least one order of magnitude compared with the original TiDB when the condition is the disjunctive normal form. With the condition of conjunctive normal form, the performance of the scheme is also better than that of the original TiDB.

Key words: distributed database; TiDB; optimizer; index

0　引言

在大数据时代,各个公司需要更高的数据处理和分析能力,保证迅速地做出商业决策以及用户响应,数据处理引擎因而成为了各个公司的核心系统。随着数据持续增多、处理性能要求提升、处理场景和类型的多元化,对数据处理引擎提出各方面的挑战[1],因此,在传统关系数据库外,发展出了NoSQL[2-3]、NewSQL[4-6]、流式处理[7-9]等各类数据处理引擎。其中,TiDB[10]是受Google的F1[11]以及Spanner[12]系统启发的一款开源分布式混合事务分析处理(Hybrid Transaction/Analytical Processing, HTAP)数据库,结合传统关系数据库管理系统(Relational Database Management System, RDMS)和NoSQL的特性,兼容MySQL语法,支持无限水平扩展,具有强一致性和高可用性。TiDB现在逐渐被许多商业公司在业务系统中使用。

当前TiDB的优化器对大部分查询请求都能生成性能极佳的物理计划,但仍有不足之处。下面通过举例来描述其中之一,同时也是本文中要解决的问题。假设有表1所示的模式。

查询"SELECT * FROM t1 WHERE a1<1 OR a2>10"发送给TiDB,当前TiDB优化器生成的物理计划为在t1表上的全表扫描。假定t1表上有100万个元组,且满足条件"a1<1"和"a2>10"的元组各仅有一个。使用全表扫描需要访问100万个元组。如果利用索引"i1"(后文以i1(a)表示名为i1建立在

收稿日期:2019-10-18;**修回日期:**2019-11-20;**录用日期:**2019-11-22。　**基金项目:**国家重点研发计划项目(2016YFB1000701)。
作者简介:兰海(1993—),男,四川成都人,硕士研究生,CCF会员,主要研究方向:数据库系统、分布式系统;　韩珂(1995—),男,河南南阳人,硕士研究生,主要研究方向:大规模并行处理数据库、族谱数据管理;　申砾(1985—),男,河北石家庄人,硕士研究生,主要研究方向:数据库、分布式系统;　崔秋(1986—),男,天津人,硕士研究生,主要研究方向:分布式数据库;　彭煜玮(1980—),男,湖北天门人,副教授,博士,CCF会员,主要研究方向:数据库系统、数字水印。

列a上的索引)获取满足条件“a1<1”的元组,利用索引“i2”获取满足条件“a2>10”的元组,然后将两个结果进行并操作,即得到结果。在这种执行方式中仅需要访问4个元组,其中两个为数据元组,两个为索引元组(TiDB中索引非树状结构,索引扫描没有中间节点访问的开销)。同样的环境下,后者性能提升明显。

表1　模式信息

Tab. 1　Mode information

对象	创建语句
表	CREATE TABLE t1(a1 int, a2 int, a3 int);
索引	CREATE INDEX i1 ON t1(a1);
索引	CREATE INDEX i2 ON t1(a2);

从上述例子可知,当约束条件涉及多个索引属性时,首先利用单个索引获取结果,然后将结果进行并(条件为析取范式(Disjunctive Normal Form, DNF))或者交(条件为和取范式(Conjunctive Normal Form, CNF))操作以获取最终数据元组的方案优于用全表扫描或单个索引扫描的执行方式。

本文研究在TiDB中利用多个索引提供更优的数据访问方案。将利用多个索引进行数据访问的路径称为MultiIndexPath,具体根据约束条件类型又分为MultiIndexOrPath以及MultiIndexAndPath。

在TiDB上增加MultiIndexPath的支持并不容易,存在如下难点:第一,如何将路径生成算法与TiDB系统融合,以生成可能的MultiIndexPath。第二,索引选择。如果表上的一个约束条件有多个可使用索引,如何选择其中一个。第三,代价模型。由于TiDB将计算与存储分离,两者通过远端程序调用(Remote Procedure Call, RPC)进行通信以及数据传输,增加了网络开销;同时,TiDB运用大量并行执行技术,增加了代价模型的建模难度。

针对以上难点,本文首先基于TiDB现有的优化器机制实现了MultiIndexPath的生成算法。其次,当有多个索引可选时,采用启发式方法,进行索引选择,后文通过实验证明了该方法的有效性。第三,在充分考虑了网络以及并行等因素后,给出了MultiIndexPath的代价模型。最后,在TiDB的执行架构基础上,实现了物理计划的执行框架;并进一步结合TiDB架构特点,提出了一种Pipeline模式的执行算法。

1　相关研究

单机关系型数据库针对上述问题已经能生成利用多个索引的物理计划,但在实现方案上均有所不同。

MySQL提供了IndexMerge[13]。当前MySQL支持三种方式利用多个索引:Intersection(交)、Union(并)以及Sort_union(排序并)。交、并操作需要每个索引返回的表元组按照Rowid排序,然后将从不同索引获取的表元组利用归并操作得到最终结果。

PostgreSQL中用Bitmap Index Scan以及Bitmap(位图)操作[14]来实现多个索引的使用:首先利用Bitmap Index Scan构建每个索引要访问的物理页的位图,然后利用位图的交或者并确定最终要访问的页,最后访问页并从中取得满足条件的元组。

商业数据库DB2中,首先利用单个索引获取满足索引条件的Rowid,然后再将这些不同的Rowid集合执行交或者并操作获取最终Rowid集合,最后将利用这些Rowid进行数据获取。商业数据库Oracle中有多个方式利用多个索引,如IndexJoin以及Bitmap Merge。其中IndexJoin将不同索引返回的结果根据Rowid来进行Join,然后返回。Bitmap Merge则利用Oracle中的Bitmap索引,通过Bitmap的交或者并位操作来获取最终满足结果的Bitmap,再将Bitmap变为Rowid,通过Rowid获取最终的表数据。对商业数据SQL Sever中or的情况,首先通过多个IndexSeek获取满足条件的索引元组,再利用Sort+Concatenation或者MergeJoin+Aggregate。对于and情况也是利用类似的方式。

2　TiDB的数据获取

TiDB主要包括三个核心部件:TiDB Server、PD Server和TiKV Server[15]。其中:TiDB Server是计算层,负责SQL语句的执行;TiKV Server是存储层,为TiDB提供了分布式存储;PD Server负责集群管理,包括集群相关的元数据存储、全局事务的管理以及TiKV集群的负载均衡。

以表1中模式为例,如果查询为“SELECT * FROM t1 WHERE a1<2 AND a3=3”,该查询语句有两种可能的表上数据访问方式:全表扫描和索引扫描。

2.1　全表扫描

TiDB Server通过PD了解表上数据分布位置后,将全表扫描计划以及过滤条件“a1<2 AND a3=3”发给对应的TiKV。TiKV对表t1上的元组逐一扫描并判断是否满足条件,将满足条件的元组返回给TiDB Server,最后由TiDB Sever返回给用户。

2.2　索引扫描

TiDB的索引扫描包含两种:Index-only扫描以及一般索引扫描。Index-only扫描适合于查询目标列被索引覆盖的场景。一般索引扫描先用索引获取满足条件的元组Rowid,然后利用Rowid去获取对应的表数据。

3　整体框架与代价模型

为能在TiDB中利用多个索引协同完成查询,本文提出了一种新的访问路径:MultiIndexPath。

3.1　整体流程

TiDB中构建逻辑计划时,将生成一个全表扫描路径以及所有可能的索引扫描路径。MultiIndexPath的生成位于逻辑优化结束后、物理优化开始前。在此过程中,优化器会根据表上的条件以及索引信息生成可能的MultiIndexPath加入备选路径中。进行物理优化时,对每个MultiIndexPath备选路径根据4.2小节中的代价模型估算代价,最终选中代价最低的物理计划(记为MultiIndexPlan)。

MultiIndexPlan有两种可能的执行方案:1)利用索引扫描得到实际元组,对各个索引得到的实际元组执行并或者交操作;2)利用索引获取表上数据的Rowid,然后对Rowid进行交并操作,再根据结果获取对应的表数据。

本文基于以下几点考虑选择了后一种方案:1)第一种方

案需要获取表元组的Rowid值执行交并，增加了一次对元组的解析操作；2）由于TiDB Server利用RPC从TiKV获取实际元组，重复元组会耗费额外的网络带宽；3）TiDB中索引扫描的流程是从TiKV先获取Rowid，然后通过Rowid取表数据，这种执行模式为第二种方案提供了实现基础。

MultiIndexPlan的执行如图1所示。其中涉及到三类协程（routine）：

1）IndexWorker：每个索引对应一个，执行索引扫描，获取满足条件的Rowid；

2）AndOrWorker：对返回的Rowid集合进行交并得到最终的Rowid集；

3）TableWorker：根据AndOrWorker得到的Rowid集，获取表元组。

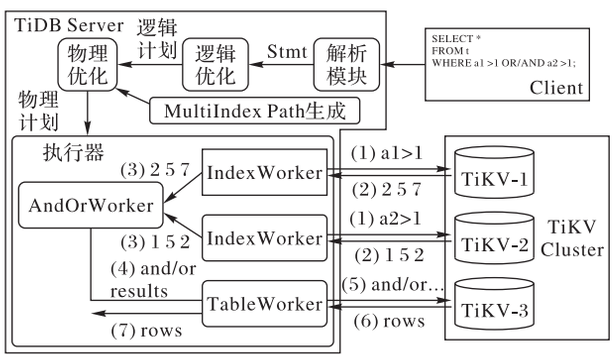

图1 MultiIndexPlan执行框架

Fig. 1 MultiIndexPlan execution framework

3.2 代价模型

MultiIndexPlan的代价包括：索引扫描代价、Rowid传输代价、表数据扫描代价、过滤条件计算代价、表元组传输代价以及执行交并操作的代价。代价模型中用到的符号说明见表2。

表2 代价模型的符号说明

Tab. 2 Symbol description of cost model

标识	简称	含义
IndexTupleCost	ITC	索引元组扫描代价
NetworkCost	NC	网络传输代价
TableScanCost	TSC	数据元组扫描代价
SelectionCost	SC	过滤条件执行代价
MergeCost	MC	集合交并代价
networkFactor	nF	网络传输代价因子
scanFactor	sF	元组扫描代价因子
selectionCPUFactor	selF	过滤条件代价因子
mergeFactor	mF	集合交并代价因子
indexRowCount	iRC	所有索引元组总个数
dataRowCount	dRC	最终结果元组总个数
afterSetOPRowCount	aSC	交并操作结束后总Rowid个数
TotalCost	TC	总代价

在3.1节中描述的本文所采用的执行模型中，多个IndexWorker并行进行Rowid扫描，得到的Rowid由AndOrWorker处理，最初增加并行度能够提高性能，但过高的并行度会让AndOrWorker成为瓶颈，无法带来性能提升。因此本文的代价模型中仍然按串行执行的方式计算代价：

$$TC = ITC + NC + TSC + SC + MC$$

$$ITC = iRC*sF$$

$$NC = iRC*nF + dRC*nF$$

$$TSC = dRC*sF$$

$$SC = aSC*selF$$

$$MC = iRC*mF$$

4 路径生成算法

路径生成分为两个阶段：第一阶段生成所有可能的MultiIndexOrPath；第二阶段生成所有可能的MultiIndexAndPath。

4.1 MultiIndexOrPath 生成

MultiIndexOrPath生成算法如算法1所示，输入需要访问的表上的索引集合Is以及条件数组PCs，输出为所有可能的备选MultiIndexOrPath（记为CP）。

算法1 GenMultiIndexOrPaths。

```
输入  Is, PCs;
输出  CP。
1)  CP=[]
2)  foreach cond in PCs
3)    imPaths=[]
4)    if isOrCond(cond) is false
5)      contiue
6)    newCs=flatten(cond, "or")
7)    foreach newC in newCs
8)      if isAndCond(newC) is true
9)        andCs=flatten(newC,"and")
10)     else
11)       andCs=[]{newC}
12)     ixPaths=genIxPaths(andCs, Is)
13)     if ixPaths is nil
14)       imPaths=[]
15)       break
16)     imParPath=genParIxPath(ixPaths)
17)     imPaths.add(imParPath)
18)   if imPaths is not empty
19)     oneP=genIxMOr(imPaths, PCs)
20)     CP.add(oneP)
21) return CP
```

算法依次处理条件数组的每一项，对每一个数组元素cond，首先检查该条件是否由OR连接的表达式：如果不是则取数组的下一个条件进行处理（见算法第4）行）；如果是OR连接表达式，则将其展开。例如"((a1>1 OR a2<10) OR a3=5)"，将展开为[a1>1, a2<10, a3=5]。如果a1>1、a2<10、a3=5都能够利用索引来进行数据获取，从它们三者就能够得到一个MultiIndexOrPath备选路径。如果展开项中有一个不能利用索引，则该cond不能生成MultiIndexOrPath（见算法12）~15）行）。

每个子表达式可能有多个可用索引，当前采用启发式规则选择其中一个索引：1）优先选择覆盖更多表达式中属性的索引；2）优先选择索引列数目最多的索引；3）如果通过上述两条规则都无法确定，则随机选择一个索引。当单个索引扫描路径生成后，生成最终的MultiIndexOrPath（见算法19）行）。

4.2 MultiIndexAndPath 生成

MultiIndexAndPath生成算法如算法2所示，输入为索引信息Is、条件数组PCs以及已经被用于生成MultiIndexOrPath

的条件数组UCs。生成MultiIndexAndPath至少要两个AND连接的条件,若条件数组中除去已用于MultiIndexOrPath生成的条件少于两个,则无法生成MultiIndexAndPath,直接返回(算法1)、2)行)。

如果剩下的条件多于两个,首先将已经生成MultiIndexOrPath的条件加入tableFilters数组中,其次从条件数组中移除它们(算法4)、5)行)。针对新得到的条件数组的每一个条件表达式,生成所有可能的索引路径。如果无法生成索引路径,则将该条件加入tableFilters数组中,然后进行下一个条件的处理。如果有多个索引路径生成,则基于启发式规则选择一个索引路径返回(算法6)~12)行)。当处理完所有条件后,得到的索引路径多于2个则生成MultiIndexAndPath(算法13)~16)行)。

算法2　GenMultiIndexAndPaths。

```
输入　Is,PCs,UCs;
输出　Path。
1)　if PCs.len - UCs.len<2
2)　　retur null
3)　tableFilters.add(UCs)
4)　newCs=PCs.remove(UCs)
5)　imPaths=[]
6)　foreach cond in newCs
7)　　ixPaths=getIxPaths([]{cond},Is)
8)　　if ixPaths is null
9)　　tableFilters.add(cond)
10)　　continue
11)　imParPath=genParIxPath(ixPaths)
12)　imPaths.add(imParPath)
13)　if imParPath.len<2
14)　　return null
15)　Path=genIxMAnd(imPaths,tableFilters)
16)　return Path
```

5　执行优化

在MultiIndexPlan执行过程中,需要通过索引获取Rowid并作交并操作,再根据结果Rowid去获取实际的数据。在作交并操作时,可通过有序集的归并或者使用位图方式来实现。两个方法分别在许多数据库都被采用,具体见第2章描述。获取所有Rowid建立位图或者获取所有Rowid后进行排序再归并的操作不能以Pipeline模式执行。尽管上述两种执行方式将对表上元组的随机扫描变成顺序扫描,在以传统磁盘为介质的环境中可以大幅度提升性能;同时各自都保证了集合操作结果的正确性。TiDB的数据存储层中的数据获取方式与第2章中描述的数据库有所不同。在TiDB中,数据存储管理由TiKV完成,在最底层由RocksDB进行数据存储。RocksDB对外提供的数据获取接口包括了prefixseek、get以及next方式。

如果数据连续性比较好,则第一个数据用prefixseek获取,剩余的数据用next获取是效率更高的方法。如果数据连续性不好,用一个prefixseek加多个next的方式性能会比用多个prefixseek的方式更差。例如,要从1 000 000行中返回100行,平均每行之间相隔10 000个Rowid,如果第二个数据采用next的方式来获取,则需要先执行9 999个无用的next。如果第二个数据采用prefixseek的方式获取,则只需要一个prefixseek操作。两种方式的代价比约为9 999×1∶8(单次prefixseek和next的耗时比约为8∶1)。此外,采用一个prefixseek加多个next的执行方式还需要额外对所有Rowid进行排序的代价,而且这是一个断流点。

综合以上分析以及TiDB在Rowid上的天然不连续性,本文认为一个prefixseek加多个next的数据获取方式不适合于MultiIndexPlan的执行,所以,本文提出对任意序的Rowid均能以Pipeline模式进行表元组获取的执行方式。

对于MultiIndexOrPath,采用一个集合来记录当前已经发送给TableWorker的Rowid,每当索引(无论哪个索引)返回一个新的Rowid,先检查其是否在集合中:如果存在(表示该Rowid已经被访问过),则跳过该Rowid;如果不存在,将其加入集合中,并将该Rowid发送给TableWorker进行表上元组获取。

对于MultIndexAndPath,以图2所示的例子说明算法原理。对每个索引增加一个集合,用于记录当前该索引已经返回,但是还没有进行表上数据获取的Rowid。在这个例子中记为set1以及set2。假设上面的"*ix*1"和"*ix*2"索引交替返回Rowid。如果一个新的Rowid从索引"*ix*1"返回,首先检查该Rowid是否在set2中,如果存在,从set2中删除该Rowid,并发送给TableWorker;如果没有在set2中,将其加入到set1中。同理对"*ix*2"返回的索引也是类似的处理流程。用表3来描述上例的处理流程。

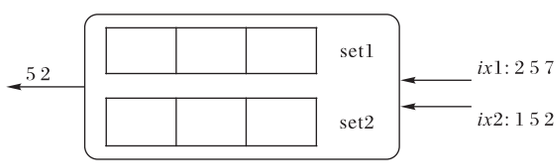

图2　MultiIndexAnd执行示例

Fig. 2　Example of MultiIndexAnd execution

表3　MultiIndexPlan执行流程示例

Tab. 3　Example of MultiIndexPlan execution process

新Rowid	set1	set2	TableWorker
2(*ix*1)	[2]	[]	[]
1(*ix*2)	[2]	[1]	[]
5(*ix*1)	[2,5]	[1]	[]
5(*ix*2)	[2]	[1]	[5]
7(*ix*1)	[2,7]	[1]	[]
2(*ix*2)	[7]	[1]	[2]

关于存储Rowid集合的数据结构,本文首先采用Bitmap来实现。实际测试也表明该方案执行效率很高,但是TiDB中的Rowid为int64,所以可能会存储$2^{64}-1$个位,这会使得Bitmap的尺寸超过内存容量。本文最终选择普通的Hashmap方式实现,效率仍旧可以得到保证,对内存的占用也较低。

6　实验与结果分析

将本文方法在TiDB 3.0版本中进行了实现,并通过多方面的实验验证了其效果,接下来就对实验的方法和结果进行介绍。

6.1 实验环境

采用4台服务器组成集群，每台机器处理器为Intel i7-7700，内存16 GB，存储为256 GB的SSD；服务器之间通过千兆网连接；其中3台服务器用于TiKV集群，另一台服务器上搭建PD和TiDB Server。

实验中用到的数据如表4所示，由2个人工生成的数据集构成。在数据集的每个列上均建有索引。查询语句模板如表5所示。

表4 数据集描述

Tab. 4 Description of datasets

数据集	元组数量(万)	模式描述
T200	200	3列，每列为整型
T200M	200	8列，每列为整型

表5 SQL语句模板

Tab. 5 Statement templates of SQL

模板	SQL
DNF	SELECT * FROM T200 WHERE C1< $1 OR C2> $2;
CNF-1	SELECT * FROM T200M WHERE C1< $1 AND C2> $2;
CNF-2	SELECT * FROM T200M WHERE C1< $1 AND C2< $2;

6.2 结果及分析

本实验主要验证下面几个方面：1)添加多索引扫描方式后，针对本文所考虑的场景，查询性能相比未添加该方式是否有性能提升；2)验证在生成路径时的启发式规则的正确性。

6.2.1 性能提升

第一个实验修改$1与$2的值，使得模板DNF语句的选择率从0.1%到80%。图3展示了优化前、后TiDB对DNF语句的响应时间变化情况。在选择率低于60%的情况下，多索引访问的方式优于TiDB原来的执行方式，并且在选择率低于8%时，有数量级的性能提升。原来TiDB的全表扫描，执行时间与表上数据元组数量相关，因而随着表上数据量增多，这种性能提升会更加地明显。当选择率高于4%时，会看到优化后的响应时间会随着选择率成倍增长而对应地倍增，主要原因是随着选择率增加，结果元组增多，而网络开销与结果元组的数目成正比，因而网络开销增大，并占主要部分。

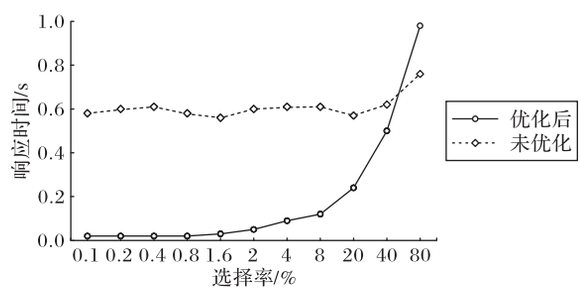

图3 DNF测试

Fig. 3 Test of DNF

第二个实验修改$1和$2的值，使得每个索引的选择率从1%到50%。图4展示了优化前、后TiDB对CNF语句的响应时间变化情况。其中，CNF-1语句的查询结果为空，CNF-2语句的查询结果非空。上述选择率是指单个索引上的选择率。实验结果表明，对于两种查询语句，多索引的访问方式都略优于原TiDB的扫描方式。

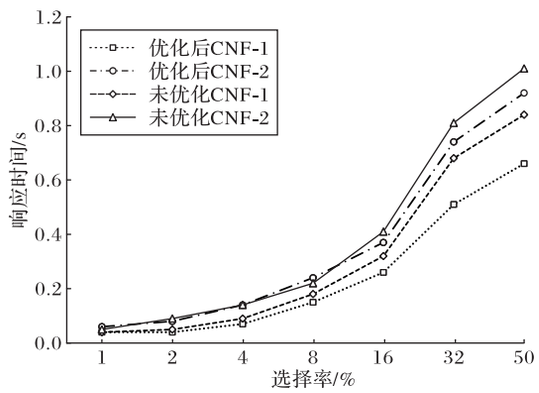

图4 CNF测试

Fig. 4 Test of CNF

6.2.2 启发式索引选择方式

按4.2.1节所述，适合一个条件的索引有多个时，将依照启发式规则选择索引，方便后面能够进行多个索引扫描的合并，减少并行。图5以及图6分别表示在DNF条件以及CNF条件下，优化后不同的并行度对响应时间的影响。从图5和图6中看出，在误差允许范围内，随着并行度的增加，性能并没有得到相应的提升，其原因是随着并行度的增加，耗费了更多的物理资源，如处理器、网络带宽等，抵消了性能收益。

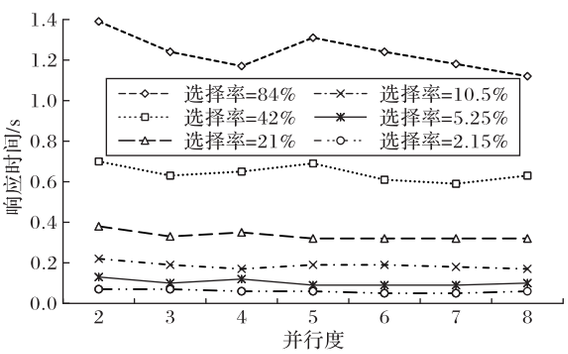

图5 DNF并行度测试

Fig. 5 Test of parallelism degree of DNF

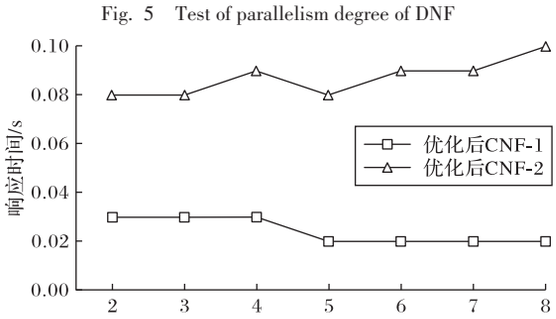

图6 CNF并行度测试

Fig. 6 Test of parallelism degree of CNF

7 结语

在约束条件含有多个索引属性时，TiDB不能利用多个索引提供更好的物理计划。为了解决该问题，本文首先在TiDB中设计并实现了MultiIndexPath的生成算法，并提出了一种Pipeline模式的MultiIndexPlan执行算法。实验结果表明在本文所考虑的场景中，利用多个索引的数据访问有明显的性能提升。

参考文献 (References)

[1] 崔斌,高军,童咏昕,等. 新型数据管理系统研究进展与趋势[J]. 软件学报, 2019, 30(1):164-193. (CUI B, GAO J, TONG Y X, et al. Progress and trend in novel data management system[J]. Journal of Software, 2019, 30(1):164-193.)

[2] SIVASUBRAMANIAN S. Amazon dynamoDB: a seamlessly scalable non-relational database service[C]// Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2012: 729-730.

[3] GEORGE L. HBase: the definitive guide: random access to your planet-size data[M]. Sebastopol, CA: O'Reilly Media, Inc., 2011: 1-522.

[4] KEMPER A, NEUMANN T. HyPer: a hybrid OLTP&OLAP main memory database system based on virtual memory snapshots[C]// Proceedings of the IEEE 27th International Conference on Data Engineering. Piscataway: IEEE, 2011: 195-206.

[5] FÄRBER F, CHA S K, PRIMSCH J, et al. SAP HANA database: data management for modern business applications[J]. ACM SIGMOD Record, 2012, 40(4): 45-51.

[6] STONEBRAKER M, WEISBERG A. The VoltDB main memory DBMS[J]. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2013, 36(2): 21-27.

[7] TOSHNIWAL A, TANEJA S, SHUKLA A, et al. Storm@twitter[C]// Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. New York: ACM, 2014: 147-156.

[8] NOGHABI S A, PARAMASIVAM K, PAN Y, et al. Samza: stateful scalable stream processing at LinkedIn[J]. Proceedings of the VLDB Endowment, 2017, 10(12): 1634-1645.

[9] CARBONE P, EWEN S, HARIDI S, et al. Apache flink™: stream and batch processing in a single engine[J]. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2015, 36(4):28-38.

[10] PingCAP. PingCAP 首页[EB/OL]. [2019-03-23]. https://pingcap.com/. (PingCAP. Homepage of PingCAP[EB/OL]. [2019-03-23]. https://pingcap.com/.)

[11] SHUTE J, VINGRALEK R, SAMWEL B, et al. F1: a distributed SQL database that scales[J]. Proceedings of the VLDB Endowment, 2013, 6(11): 1068-1079.

[12] CORBETT J C, DEAN J, EPSTEIN M, et al. Spanner: Google's globally distributed database[J]. ACM Transactions on Computer Systems, 2013, 31(3): Article No. 8.

[13] MySQL. Index merge optimization [EB/OL]. [2019-03-23]. https://dev.mysql.com/doc/refman/8.0/en/index-merge-optimization.html.

[14] PostgreSQL. Combining multiple indexes[EB/OL]. [2019-03-23] https://www.postgresql.org/docs/current/indexes-bitmap-scans.html.

[15] PingCAP. TiDB architecture [EB/OL]. [2019-03-23]. https://pingcap.com/docs-cn/stable/architecture/.

This work is partially supported by the National Key Research and Development Program of China (2016YFB1000701).

LAN Hai, born in 1993, M. S. candidate. His research interests include database system, distributed system.

HAN Ke, born in 1995, M. S. candidate. His research interests include massive parallel processing database, genealogical data management.

SHEN Li, born in 1985, M. S. His research interests include database, distributed system.

CUI Qiu, born in 1986, M. S. His research interests include distributed database.

PENG Yuwei, born in 1980, Ph. D., associate professor. His research interests include database system, digital watermark.