

大数据计算基础课程报告

题 目 ： 并行SQL-over-NoSQL的系统优化

专 业 计算机科学与技术

学 号

姓 名

课 程 大数据基础

日 期 12月8日

# 摘 要

**关键词**：spark-sql BaaV模型 Scan-Free KBA查询计划 交错并行功能

为了优化现有的SQL-over-NoSQL系统，本次实验参考论文《Block as a Value for SQL over NoSQL》，并基于spark sql2.4.4源代码进行实验设计与测试。本实验使用了Zidian的设计思想。基于关系代数模型的思想，本实验设计实现 Block-as-a-Value（BaaV）模型，包括schema和Extension, Shift 和 Join 三种运算符。在BaaV模型的基础上，本实验实现Scan-Free优化，对于给定的SQL查询进行优化判断，并对可优化的查询生成Scan-Free的查询计划。本实验使用RDD实现了KBA查询计划的交错并行（interleaved parallelization）执行的功能，包括有界通信、并行可扩展性与横向可扩展性。通过实验设计与测试用例，本次实验验证了BaaV模型，Scan-Free优化和KBA查询计划的交错并行执行的功能的成功实现。在对spark sql的优化效果方面，本次实验参考了论文中的实验数据，凭经验验证Zidian改进了现有的SQL-over-NoSQL系统平均减少2个数量级。

**目 录**

[摘 要 I](#_Toc29049371)

[第1章 绪 论 4](#_Toc29049372)

[1.1 研究问题的背景 4](#_Toc29049373)

[1.2 研究问题的挑战 4](#_Toc29049374)

[1.3 当前研究工作的不足之处、 4](#_Toc29049375)

[1.4 本文的工作要解决的问题以及方法 4](#_Toc29049376)

[1.5 本文的贡献 5](#_Toc29049377)

[1.6 章节安排 5](#_Toc29049378)

[第2章系统/方法框架 6](#_Toc29049379)

[2.1 系统框架 6](#_Toc29049380)

[2.2 各部分简介 6](#_Toc29049381)

[第3章BaaV模型 7](#_Toc29049382)

[3.1定义 7](#_Toc29049383)

[3.2 BaaV模型设计 8](#_Toc29049384)

[3.3运算符 9](#_Toc29049385)

[第4章Scan-Free 优化 12](#_Toc29049386)

[4.1设计思想 12](#_Toc29049387)

[4.2确定无扫描查询 13](#_Toc29049388)

[4.3 生成免扫描的KBA计划 17](#_Toc29049389)

[第5章 KB查询计划的交错并行执行的功能 19](#_Toc29049390)

[5.1 并行化KBA计划实现的功能 19](#_Toc29049391)

[5.2 实现 20](#_Toc29049392)

[第6章实验 21](#_Toc29049393)

[6.1 实验设计 21](#_Toc29049394)

[6.2 实验结果 22](#_Toc29049395)

[第7章 相关工作（创新性） 23](#_Toc29049396)

[7.1操作符的创新设计 23](#_Toc29049397)

[7.2 RDD的创新使用 24](#_Toc29049398)

[7.3 TPC- H性能评价 24](#_Toc29049399)

[第8章 结论 24](#_Toc29049400)

# 第1章 绪 论

## 1.1 研究问题的背景

键值（KV）存储已在行业中广泛使用。KV存储支持类似于字典的数据访问，以作为键值对检索和存储数据，从而提供水平可伸缩性，容错性和透明分片。近来，很多新型数据库系统常常使用键值存储（Key-Value Store）作为其底层存储引擎，然后在其上实现 SQL 查询。这有时被称为 SQL-over-NoSQL 方案。然而，这样的实现方案常常难以应付复杂的分析查询。

## 1.2 研究问题的挑战

为了支持大规模查询，系统通常是基于SQL-NoSQL架构，并通过SQL查询来进行数据分析。该架构将数据持久存储在KV存储群集中，并行地在计算群集（作为单独的层）中回答查询。相比于DBMS，该架构需要更多的数据访问和更大通信成本。因此如何减少数据访问和通信成本，并使现有的SQL-over-NoSQL系统在回答SQL查询时是否像DBMS一样有效，成为当前SQL-over-NoSQL架构研究的一大挑战。

## 1.3 当前研究工作的不足之处、

目前基于SQL-over-NoSQL体系结构的数据库系统例如Google的Spanner，Facebook的MyRocks，Hive和SparkSQL 虽然提供了潜在的键值存储优势，但在评估SQL查询时性能不如传统的DBMS。主要有以下原因：

1. 扫描成本高。通常大多数SQL over NoSQL系统是基于TaaV模型。在系统中一个关系作为一组KV对（k，t）进行存储在分布式哈希表（DHT）。DHT支持高效点通过给定的键k的get函数进行访问，获取整个元组t。对于大多数SQL查询，本文必须通过扫描整个表来获取元组t。
2. 通信负荷大。大多数SQL over NoSQL系统经常从KV存储中检索到大量数据（甚至整个关系）并由计算层进行处理。对于并行执行中的数据，这会导致沉重的通信成本。

## 1.4 本文的工作要解决的问题以及方法

（1）设计实现 Block-as-a-Value（BaaV）模型，包括 schema和所有论文中提到Extension, Shift 和 Join 三种运算符。

（2）实现论文中提到的 Scan-Free 优化。首先判断一个给定的 SQL 查询是否能够进行 Scan-Free 优化，然后对可以优化的查询生成 Scan-Free 的查询计划。

（3）实现论文中提到的 KBA 查询计划的交错并行（interleaved parallelization）执行的功能。尽可能利用系统现有的功能，如 SQL 编译和查询优化等，以尽量少地改动已有的代码。

## 1.5 本文的贡献

本文介绍了Zidian，一个用于键值（KV）存储的中间件，以通过NoSQL加速SQL查询评估。Zidian提出以键为键，整个元组为值即块模型BaaV。本文将关系代数扩展到BaaV。

BAAV下，Zidian大大降低数据访问和通讯成本。本文提供特征（充分和必要的条件）以（a）保持结果查询，即可用的BaaV存储覆盖的查询，（b）免扫描查询，即无需评估即可查询的查询扫描任何表，以及（c）有界查询，即查询可以通过访问有限数量的数据来回答。

在并行处理中，Zidian保证（a）不扫描无扫描查询，（b）受限查询的受限通信成本；（c）并行可伸缩性，即添加处理器时加快速度。而且，Zidian可以插入现有的SQL-No-SQL系统并保留水平可伸缩性。

## 1.6 章节安排

**1.6.1 第一节 数据模型**

本文介绍BaaV，将KV存储中的关系表示为关键块。本文扩展关系代数到BaaV存储，以利用BaaV模型回答SQL查询时。此外，本文定义了BaaV查询计划中的免扫描查询和有界查询，加快No- SQL上SQL系统的SQL评估。

**1.6.2 第二节 Scan-Free**

本文描述了无扫描（限定边界）查询的特征，即，本文开发了充分必要的条件来确定SQL查询在BaaV存储上是否无扫描（限定边界）。 虽然问题是无法确定的，但特征描述提供了可以有效检查的此类查询的有效语法。 此外，本文提供了一种生成查询计划的算法，该算法可确保避免对无扫描（限制有界）的查询进行扫描（限制访问有限数量的数据）。

**1.6.3 第三节 并行化**

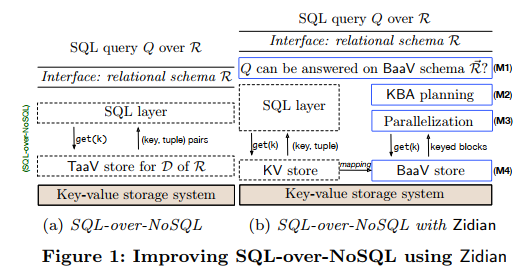
有界和可扩展性（第7节）。本文建议在并行回答查询时交错数据访问和计算，而不是先获取所有数据然后计算答案。通过这种策略，本文表明zidian不需要扫描就可以进行无扫描查询，并且不需要为受限查询增加通信成本。而且，在BaaV下，zidian保证了并行可伸缩性，并保留了SQL-No-SQL系统的水平可伸缩性。

**1.6.4 第四节 实验**

使用基准TPC-H [42]和实际数据，本文评估了Zidian的有效性。本文平均发现以下内容。 1）Zidian的无扫描查询效率分别比SoH，SoK和SoC高2.8×102、1.7×102和8.1×102，非无查询查询分别高2.0×102、1.5×102和3.6×102 。 2）使用Zidian，当数据集增长时，系统会为有界查询带来稳定的计算和通信成本。 3）Zidian具有平行可扩展性，并且可以很好地与数据集进行扩展，例如，在SoH之上的Zidian平均对8名员工的128GB数据集进行freefree和non-free-free查询分别需要27.7和65.4秒，而1.7×103和2.1× SoH没有zidian的103秒。 （4）zidian保留了基础KV系统的水平可扩展性，以应对KV工作量。

# 第2章系统/方法框架

## 2.1 系统框架



采用的系统框架是zidian。zidian用上图右侧所示的四个主要模块M1-M4扩展了SQL-NoSQL。

## 2.2 各部分简介

2.2.1 符号表

|  |  |
| --- | --- |
| 符号 | 定义 |
| R(Z) | TaaV下的关系模式 |
| R~<X, Y> | BaaV下的键值（KV）模式 |
| R (resp. R~) | 关系模式（BaaV模式） |
| att(R~) (resp. pk(R~ )) | R~的属性（resp.主键） |
| D (resp. D) | 关系型数据库（resp. relations） |
| |D|(resp.||D||) | D中的元组数（重复值） |
| D~(resp.D~) | BaaV存储（resp.KV）实例 |

2.2.1 M1-M4部分

在M1，Zidian允许用户在关系数据库D上进行常规SQL查询Q。它检查Q是否可以在BaaV存储D〜中回答。

在M2, Zidian扩展了现有的SQL层，为查询Q生成KBA计划ξ。它标识在边界或边界上scan-free的Q，并保证分别为此类查询生成scan-free和边界的KBA计划。

在M3，zidian收到KBA计划ξ后，通过并行化ξ生成并行的KBA计划ξp。

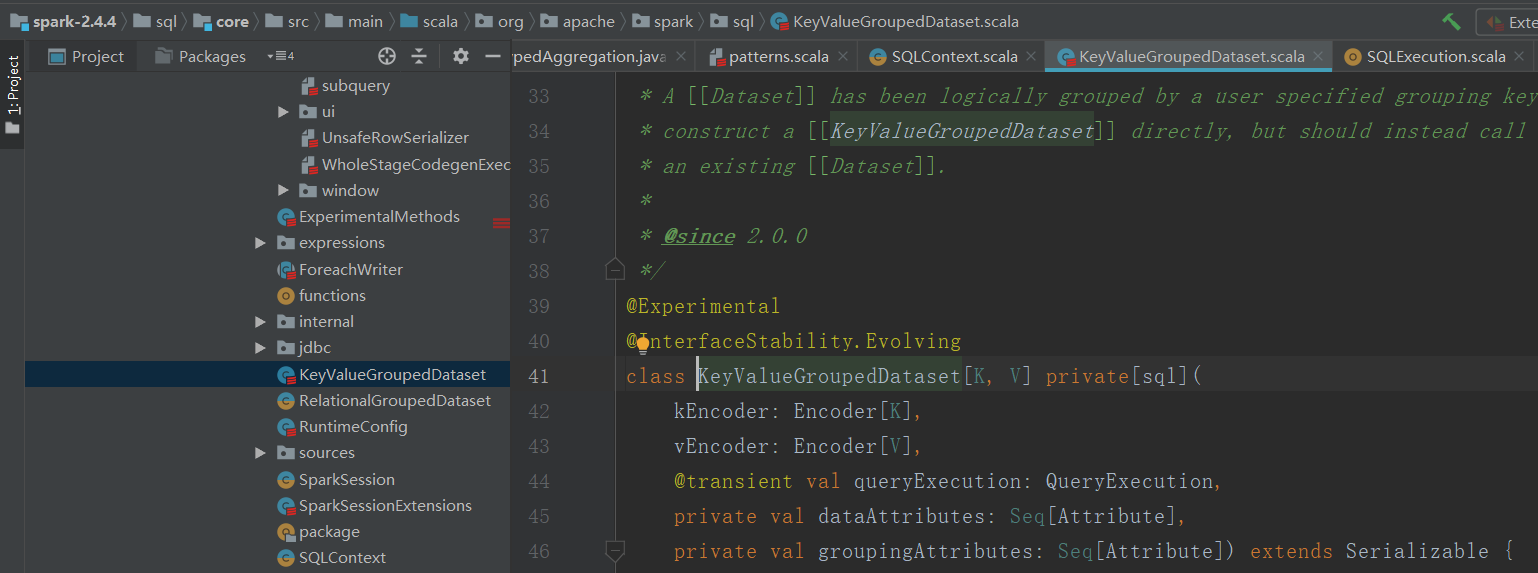
在M4，Zidian维护了一个BaaV存储D〜，该存储从模式R的常规数据库D映射到BaaV方案R〜。

# 第3章BaaV模型

## 3.1定义

在Baav模型中，设KV模式<X，Y>的实例D~，即一组关键块对（k，B）；设D是D~的关系版本。对于块B中的每个Y元组t，将（k，t）称为D~的元组。则D是D~的所有元组的集合。

BaaV模型将关系表示为键控块（k，B），其中k是部分元组的块（集合）B的键。BaaV模型使用了一个用于键值（KV）存储的中间件，Zidian。Baav模型提供（a）保存结果查询（b）免扫描查询(c) 有界查询。



## 3.2 BaaV模型设计

3.2.1 BaaV架构设计

Zidian提供了一种算法（T2B），该算法在给定架构R，一个R上的数据库D，QCS的集合Σ和存储预算b的情况下计算BaaV架构R〜，以便：（a）R〜上D的映射D〜不大于b；（b）如果b≥|D|（D的大小）则R〜支持Σ中的所有QCS Z [X]：对于任何X值，都可以从D〜提取R的数据库D中所有相关的Z值，甚至当b足够大时也无需扫描。

当b允许时，R〜是条件（I）为R保留的数据。作为实现方式的额外灵活性，Zidian还公开了一个界面，供用户使用建议的KV模式修改R〜，从而允许在环设计。

3.2.2算法T2B的概述

（1）T2B首先通过将每个QCS Z[X]视为KV模式<Z,X\Z>来生成初始BaaV模式R〜0。对于QCS在Σ中由QCS提取的任何查询Q，Q在R〜0上都是免扫描的。

（2）然后删除R〜0中的冗余KV模式：KV模式R〜(X,Z)，如果X在R〜0中为冗余对于任何查询Q，当且仅当Q在R〜0 \ {R〜}上不扫描时，Q在R〜0上不扫描。当R〜0中存在多个冗余KV模式时，算法T2B通过使用排名函数来一一删除它们，该函数选择对R〜0上的查询评估效率具有最小估计影响的一个。

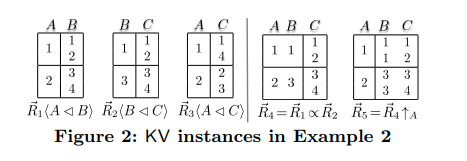
（3）如果步骤（2）生成的BaaV模式R〜'上的D超过预算b，T2B迭代地合并R〜'中的KV模式以减小映射的大小，同时保留R〜'上的所有免扫描查询，直到大小在b之内。在每次迭代中，T2B都会选择一对KV架构，其对查询评估效率的估计影响最小。

## 3.3运算符

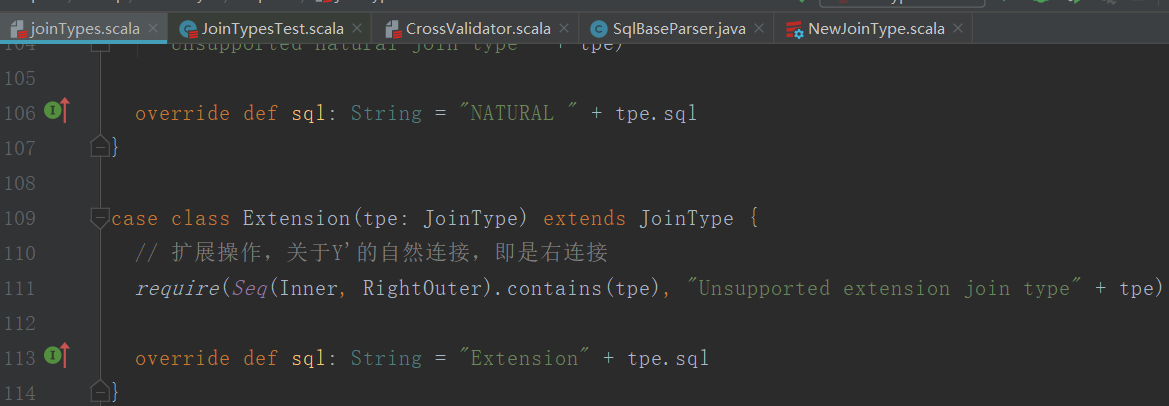
（1）Extension∝

设KV模式<X，Y>的实例D〜1，<Y',Z>的D〜2。如果Y'⊆XY，则D〜1与D〜2的Extension，记为D~ 1 ∝ D~ 2，是在新的KV模式<XY, Z>下D1 ⋈Y' D2的映射。其中D1、D2是KV实例D〜1、D〜2的关系版本，⋈Y' 是RA关系代数属性Y'的自然连接。

D〜1∝D〜2用D〜2具有相关的Z属性扩展了D〜1，并使用D〜1的值作为键来提取。可以将D〜1 ∝ D〜2视为BaaV特有的特殊“联接”：它联接D〜1和D〜2，但不访问D〜2中的所有数据，即不扫描D〜2。



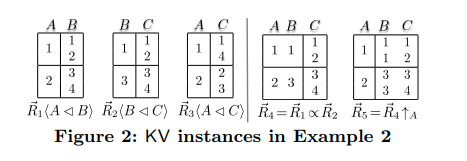
例如：如图2，R~4 = R~1 ∝ R~2



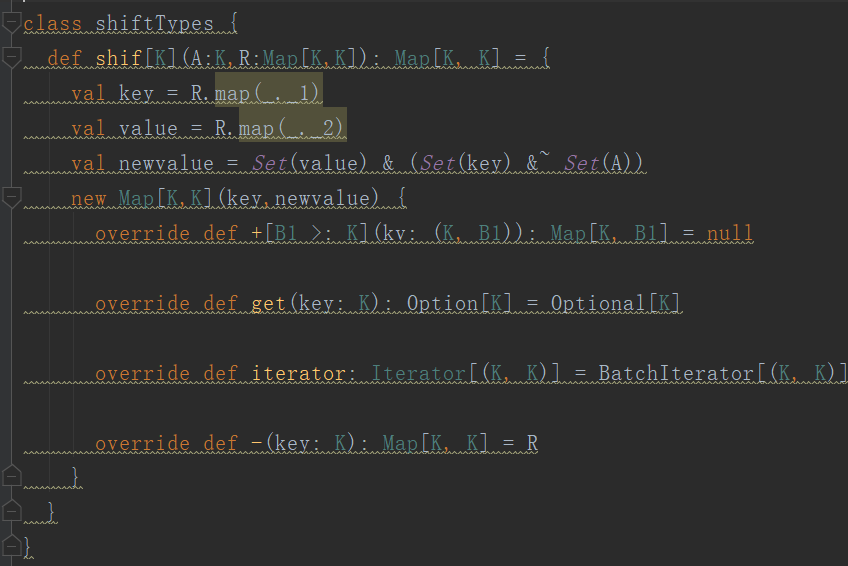
（2）Shift↑

设KV实例<X，Y>为D〜，并且X'⊆XY，D〜随X'的shift（偏移），由D〜↑X’表示，是<X',XY\X'>的KV实例D〜'，与D〜相同的关系版本。

直观地讲，D〜↑X'通过将X'设置为键来重新分配D〜的键和值属性。它允许本文对齐KV实例，并在具有不同键属性分布的KV实例上启用类似集合的操作（例如，并集和差值）。



例如：如图2，R~5 = R~4↑X’

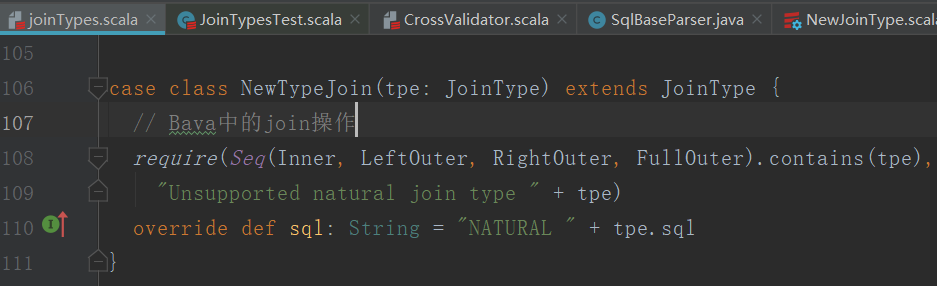


（3）Join ⋈/✶

设<X1，Y1>的D〜1和<X2，Y2>的D〜2的KV实例。D〜1和D〜2在属性X（X⊆X1Y1 ∩ X2Y2）上的join（自然连接），表示为D〜1 ⋈X D〜2，是KV模式<X1X2，Y1Y2>上D〜1和D〜2的关系版本的join的映射。

自然连接（join）运算主要是将某些选择跟笛卡尔积运算合并在一起表示，它会将两个关系模式中都出现的属性上的相等性进行选择，最后还要去除重复属性，用 ⋈ 来表示。

代码：



# 第4章Scan-Free 优化

## 4.1设计思想

**定义Q中所有属性的集合GET（Q，R〜）**。令XCQ为集合Q选择条件下属性A的集合，其中A = 常数c。下面的规则以归纳法定义GET（Q，R〜）：（a）XCQ⊆GET（Q，R〜）;

（b）如果A∈GET（Q，R〜）并且A = B由Q的选择条件C通过等传递性确定，则B∈GET（Q，R〜）;（c）如果X⊆GET（Q，R〜），R〜<X，Y>∈R〜，则Y⊆GET（Q，R〜）。集合GET（Q，R〜）不包含其他内容。

**可验证的组合VC（Q，R〜）**。集合GET（Q，R〜）标识Q的属性，其值可以使用免扫描计划从R〜中获取。但是，要回答Q，不仅要保留这些值，还必须保留它们的组合。

本文定义了GET（Q，R〜）中的属性集的集合VC（Q，R〜），以便可以使用D的映射D〜上的无扫描访问计划来检查R的D中的值组合。用R〜Q表示R〜中的KV模式，其属性全部包含在GET（Q，R〜）中，即R〜Q = { S〜∈R | 〜attr（S〜）⊆GET（Q，R〜）}。然后（从5.2节的条件（I）调用clo（））：VC（Q，R〜）= {clo（S，〜R〜Q）| S~ ∈ R~ Q}。

## 4.2确定无扫描查询

(1)检查是否可以使用scan-free从R〜中检索回答Q所需的属性。本文定义Q中所有属性的集合GET（Q，R〜），可以使用免扫描计划从R〜检索这些属性，称为R〜的可检索属性。

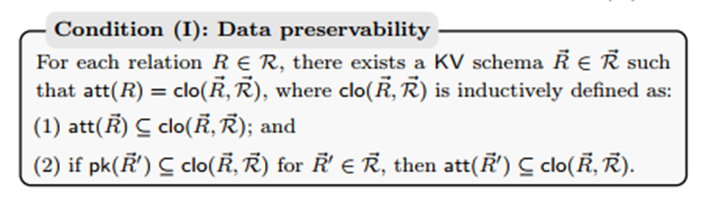
下面的规则以归纳法定义GET（Q，R〜）：

（a）XCQ ⊆ GET（Q，R〜）。

（b）如果A∈GET（Q，R〜）并且由Q的选择条件C通过等传递性可以确定A = B，则B∈GET（Q，R〜）。

（c）如果X ⊆ GET（Q，R〜），R〜<X，Y> ∈ R〜，则Y⊆GET（Q，R〜）。集合GET（Q，R〜）不包含其他内容。

（2） 检查其组合是否被保留。本文在GET（Q，R〜）中定义属性集的集合VC（Q，R〜），可以验证其组合。用R〜Q表示R〜中的KV模式，其属性全部包含在GET（Q，R〜）中，即R〜Q = { S〜∈R | 〜attr（S〜）⊆ GET（Q，R〜）}。然后（从5.2节的条件（I）调用clo（））：VC（Q，R〜）= {clo（S，〜R〜Q）| S~ ∈ R~ Q}。

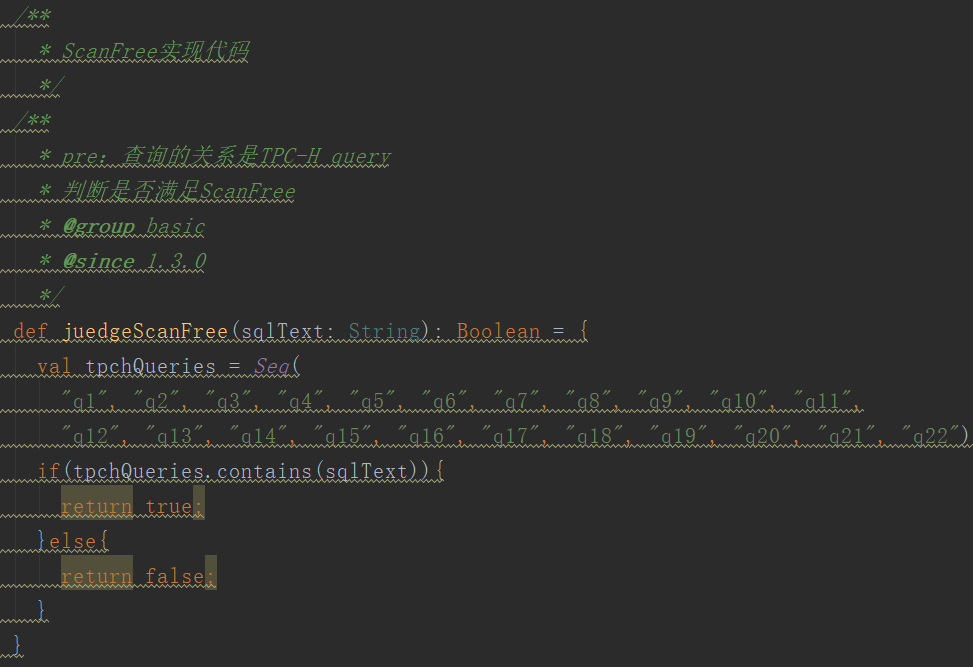


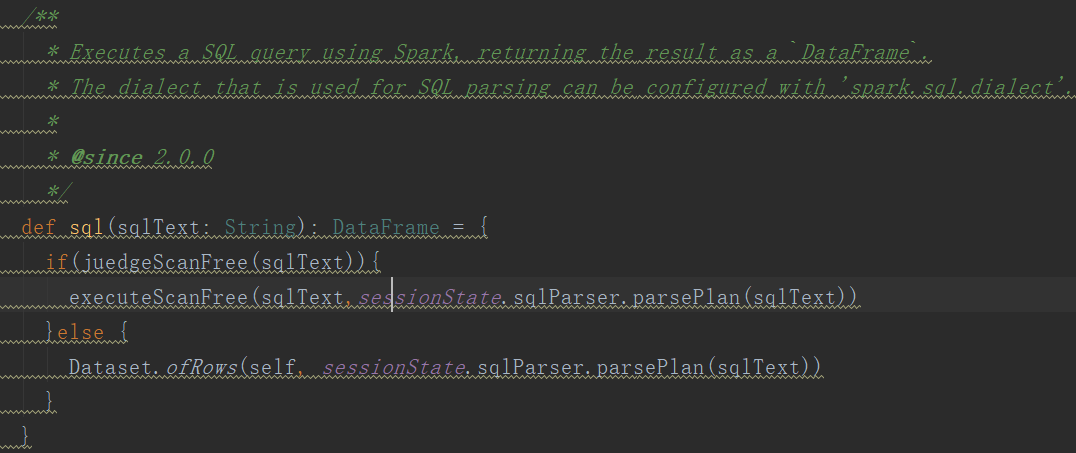
|  |
| --- |
| 条件（III）：  对于Q的最小等效查询min（Q）中的每个关系R，都存在一个集合W∈VC（min（Q），R〜）使得XRmin（Q）⊆W。 |

最小代价供货商查询min（Q）代码

|  |
| --- |
| select  s\_acctbal, s\_name, n\_name, p\_partkey, p\_mfgr, s\_address, s\_phone, s\_comment /\*查询供应者的帐户余额、名字、国家、零件的号码、生产者、供应者的地址、电话号码、备注信息 \*/  from part, supplier, partsupp, nation, region //五表连接  where  p\_partkey = ps\_partkey  and s\_suppkey = ps\_suppkey  and p\_size = [SIZE] //指定大小，在区间[1, 50]内随机选择  and p\_type like '%[TYPE]' //指定类型，在TPC-H标准指定的范围内随机选择  and s\_nationkey = n\_nationkey  and n\_regionkey = r\_regionkey  and r\_name = '[REGION]' //指定地区，在TPC-H标准指定的范围内随机选择  and ps\_supplycost = ( //子查询  select  min(ps\_supplycost) //聚集函数  from  partsupp, supplier, nation, region //与父查询的表有重叠  where  p\_partkey = ps\_partkey  and s\_suppkey = ps\_suppkey  and s\_nationkey = n\_nationkey  and n\_regionkey = r\_regionkey  and r\_name = '[REGION]'  )  order by //排序  s\_acctbal desc,  n\_name,  s\_name,  p\_partkey; |

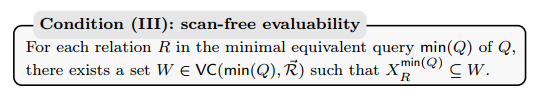
条件（III）是要对SPC Q进行扫描的充分必要条件。





（3）例子

|  |
| --- |
| **Q1’:**  select PS.suppkey, SUM(PS.supplycost)  from PARTSUPP as PS, SUPPLIER as S, NATION as N  where PS.suppkey = S.suppkey and S.nationkey  = N.nationkey and N.name = “GERMANY”  **R~ 1（R〜'1是条件（II）为Q'1保留的结果）:**  SUPPLIER' <nationkey, suppkeyi>  PARTSUPP′ <suppkey, (partkey, supplycost)>  NATION' <name, nationkey> |



在Q′ 1和 R~ 1中, Q′ 1 在R~1上无扫描: min(Q′ 1) = Q′ 1,

GET(Q′ 1, R~ 1) = {N.name, N.nationkey,S.nationkey, S.suppkey, PS.suppkey, PS.supplycost}, XQ′ 1 N={N.name, N.nationkey}, XPS Q′ 1 = {PS.supplycost, PS.suppkey},XQ′ 1S = {S.nationkey, S.suppkey}, VC(Q′ 1, R~ 1) = {XNQ′ 1, XPS Q′ 1,XQ′ 1S }。

## 4.3 生成免扫描的KBA计划

4.3.1 KBA计划

**KBA计划。** 将关系代数操作符进行组合，把一个操作符应用到其他的一个或多个操作符之上形成一个表达式树。

KBA计划类似于RA计划树。 与以叶节点为关系的常规RA计划相反，在KBA计划中，（1）每个叶要么是一个常数（常数键控块），要么是一个KV实例；（2）每个中间节点都是KBA的运算符之一。特别是，对于KBA计划ξ= R〜1 ∝ R〜2，R〜1是ξ的唯一叶节点，而R〜2被视为∝的参数。

用ξ（D〜）表示ξ在BaaV存储D〜上的执行结果，其中ξ和D〜在相同的BaaV模式R〜上。下面我们确定了KBA计划的两个特殊类别。

**免扫描的KBA计划。** 如果KBA计划ξ的所有叶节点都是常量，则它在BaaV架构R〜上是免扫描的。 直观地讲，执行此类ξ不会对R〜的任何BaaV存储进行扫描。

4.3.2 chase-based方法

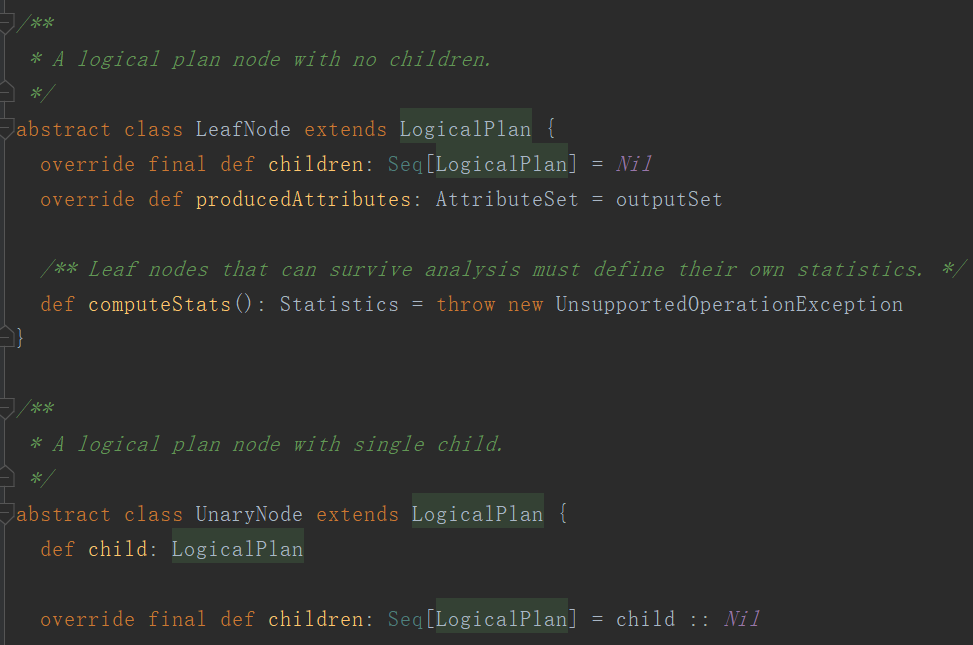
为了在R〜上生成Q的免扫描的KBA计划ξQ，Zidian采用chase-based方法。

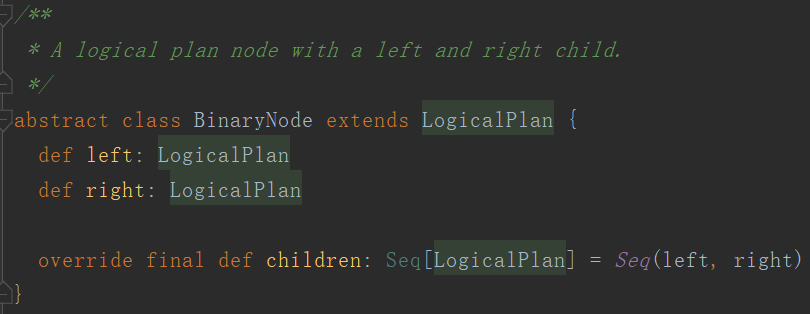
（1） Zidian首先生成常规的RAaggr计划ξQ。请注意，ξQ的叶子节点是Q中的关系。

（2） Zidian然后用KBA计划替换ξQ的所有叶子节点，更具体地说，它生成用于计算GET（Q，R〜）和VC（Q，R〜）的规则的应用序列ℓ（第6.1节），称为Q和R〜的追逐序列。

1. 如果Q在R〜上不是免扫描的，则ξQ中必须存在叶节点R使得XRQ 不属于W，对于所有W∈VC（Q，R〜）。对于这样的关系R，必须存在R〜∈R’〜，使得条件（II）满足XQR ⊆ clo（R，R’〜），因为R’〜是为Q保留的结果。zidian简单地用R〜替换ξQ中的R。
2. 由（2）和（3），所有改写后的ξQ中的叶子节点是R〜的常数或KV模式，即ξQ成为KBA计划。

4.3.3 通过LogicalPlan实现KBA树





# 第5章 KB查询计划的交错并行执行的功能

5.1 并行化KBA计划实现的功能

（1）有界通信。如果ξ在D〜上是无扫描的，则ξ的并行执行将保持在无扫描状态，而且，如果D〜的边界为有界，则并行计划会产生有界的数据访问和有界的通信成本。

（2）并行可扩展性。并行化的ξ，对于计算和通信成本都是并行可扩展的。

注：可扩展性是一种对软件系统计算处理能力的设计指标，高可扩展性代表一种弹性，在系统扩展成长过程中，软件能够保证旺盛的生命力，通过很少的改动甚至只是硬件设备的添置，能实现整个系统处理能力的线性增长，实现高吞吐量和低延迟高性能。

（3）横向可扩展性。zidian保留了现有的SQL-No-SQL系统的水平可伸缩性，将其作为添加新存储节点时的系统I / O吞吐量来衡量。

Spark sql是一种SQL-No-SQL系统，保留了水平可伸缩性。

5.2 实现

RDD是一种有容错机制的特殊数据集合，可以分布在集群的结点上，以函数式操作集合的方式进行各种并行操作。

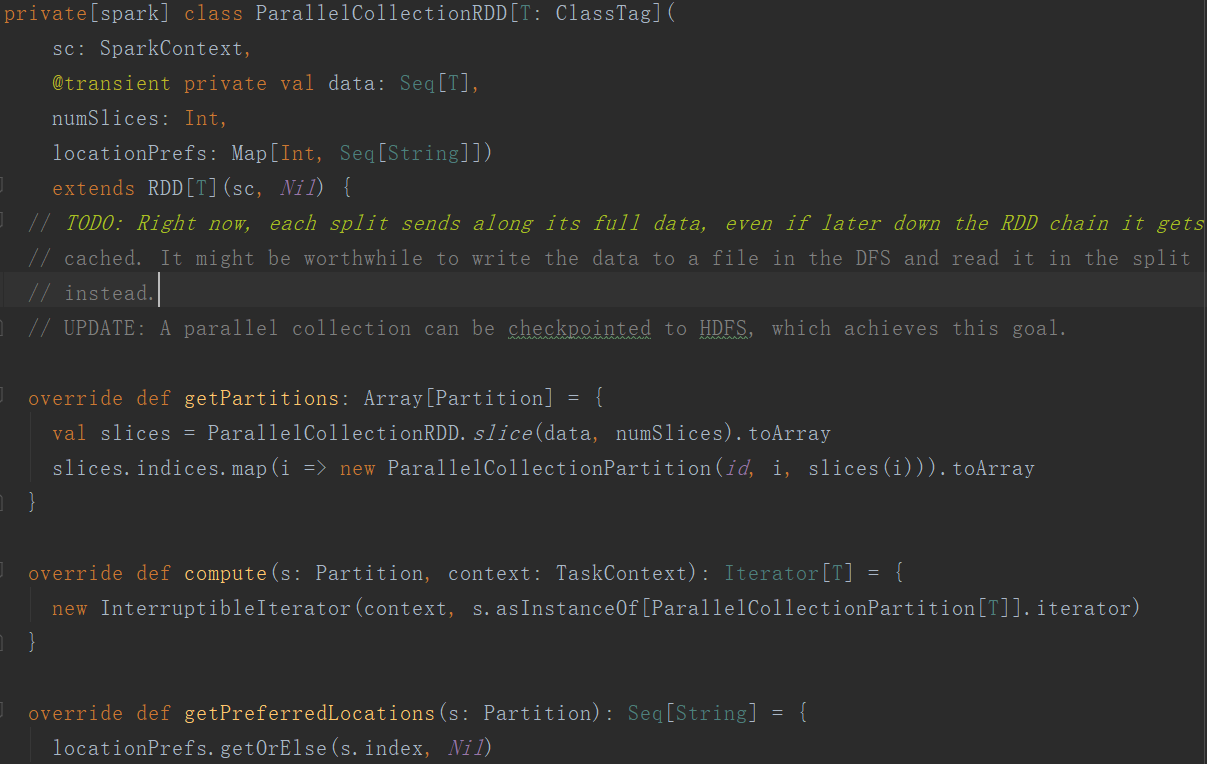
使用ParallelCollectionRDD对ξ实现并行化。RDD具有容错机制，并且只读不能修改，可以执行确定的转换操作创建新的RDD。RDD 具有以下属性：

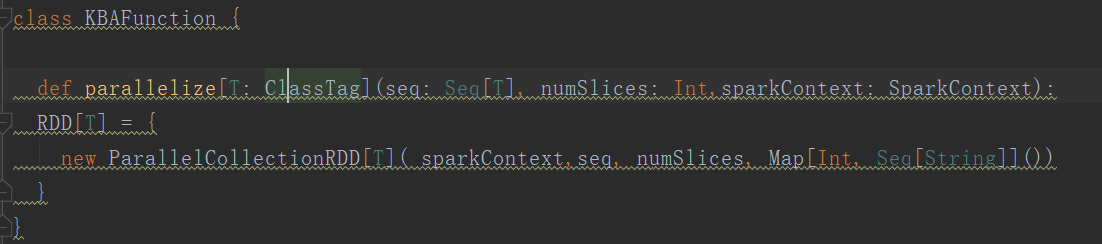
1)只读：不能修改，只能通过转换操作生成新的 RDD。

2)分布式：可以分布在多台机器上进行并行处理。

3)弹性：计算过程中内存不够时它会和磁盘进行数据交换。

4)基于内存：可以全部或部分缓存在内存中，在多次计算间重用。

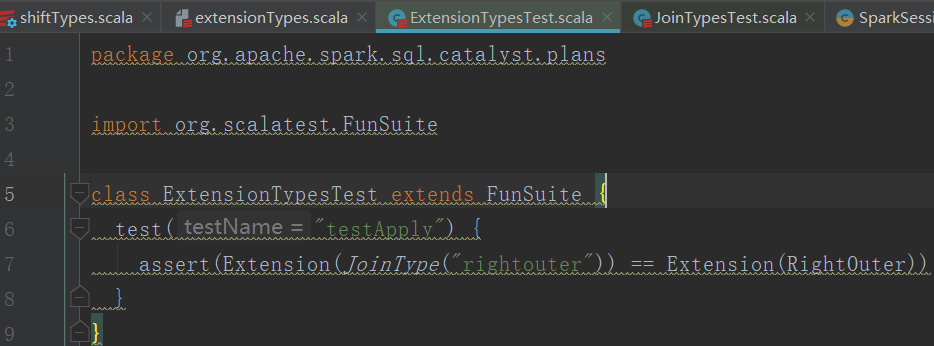




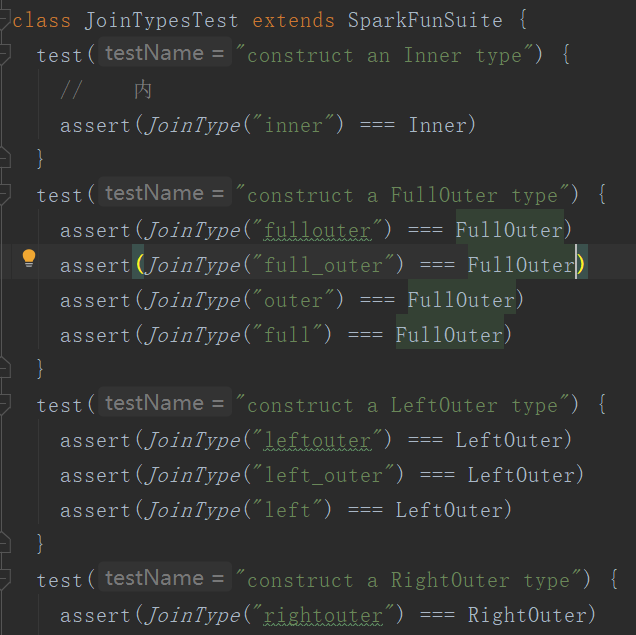
# 第6章实验

## 6.1 实验设计

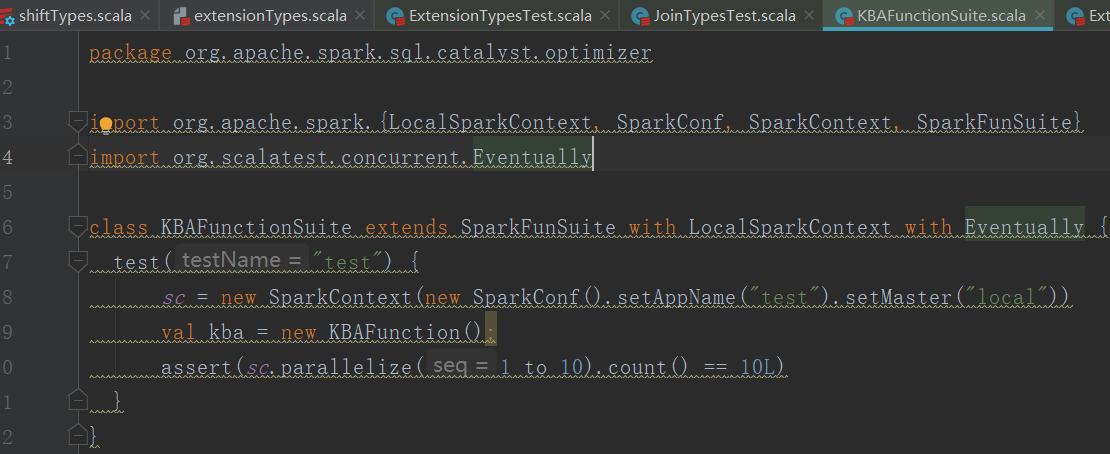
（1）Extension运算符的测试



（2）Join运算符的测试

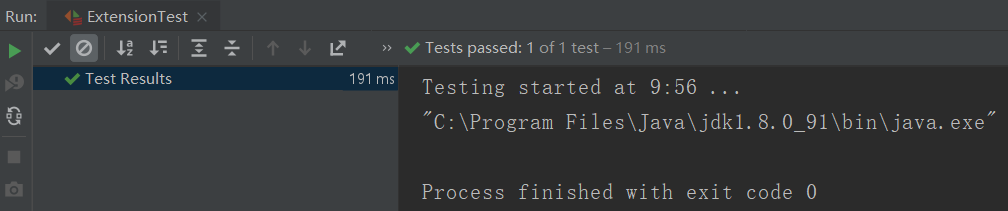


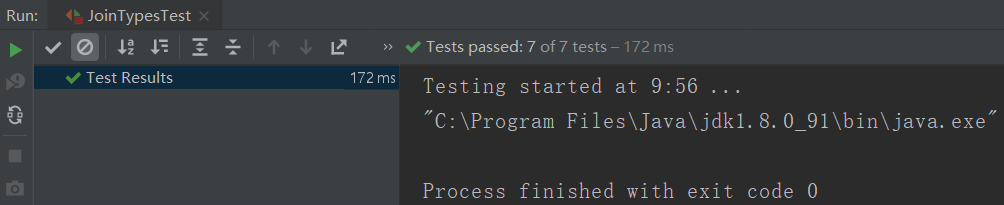
（3）KB查询计划的交错并行执行的功能测试



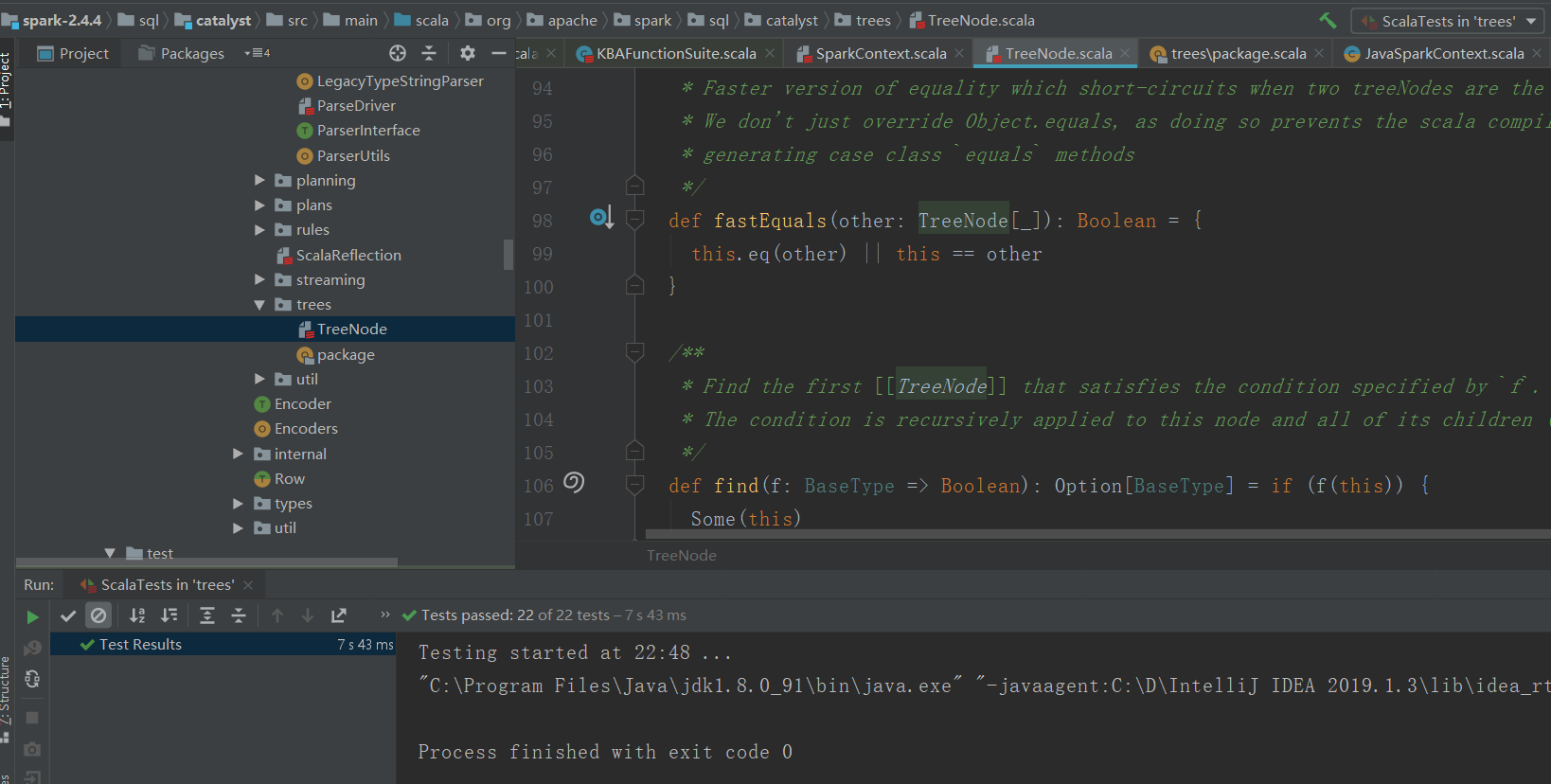
## 6.2 实验结果

Baav模型测试结果：

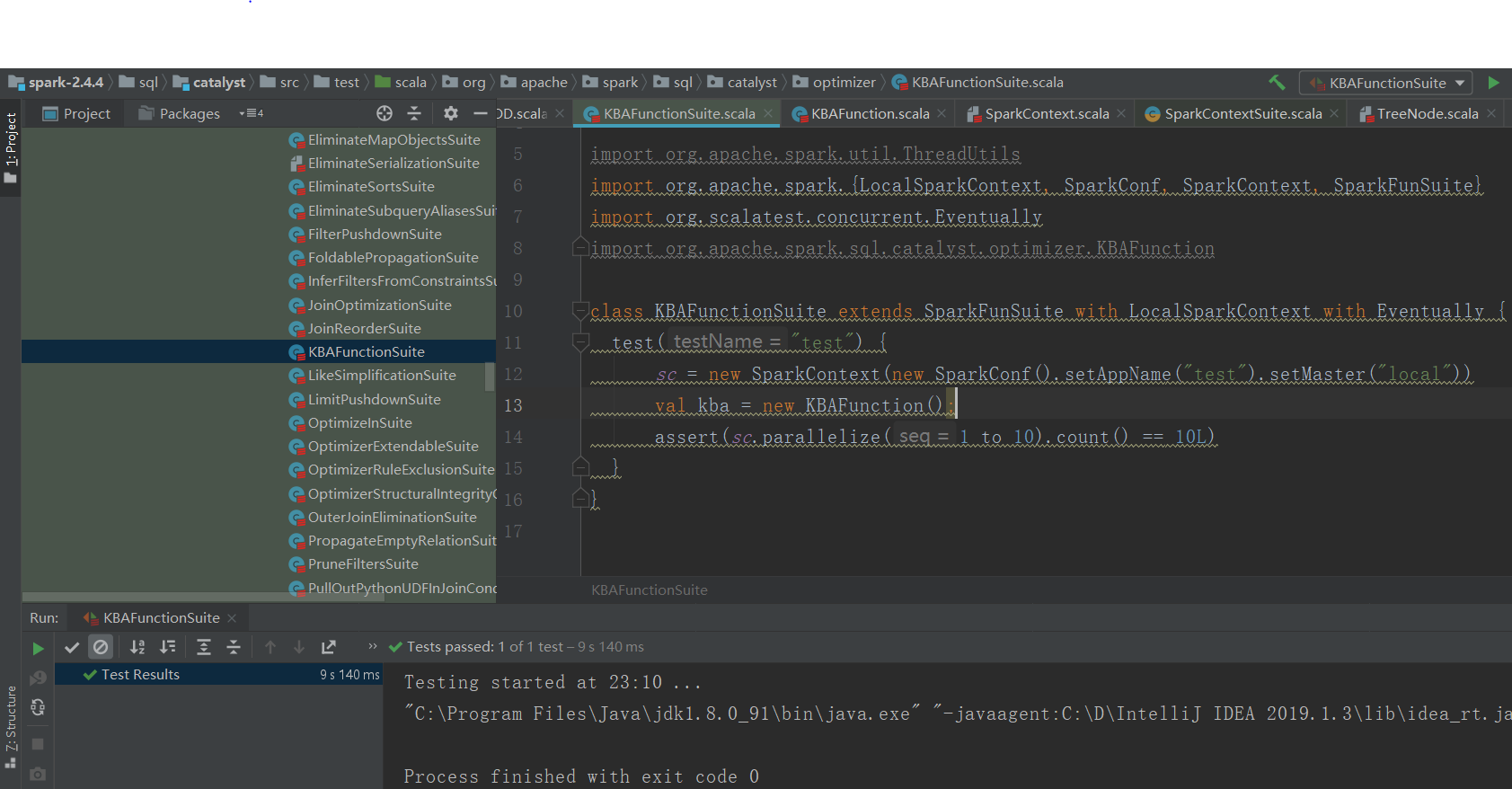




Scan-Free测试结果：



KBA测试结果：



# 第7章 相关工作（创新性）

7.1操作符的创新设计

在关系代数有join、并、差、笛卡尔积、投影和选择等操作。虽然spark sql是SQL-over-NoSQL系统并基于键值存储，但基本操作符与关系数据库相同。这些基本操作符在键值的操作上并不灵活。为了进一步减小查询代价，本实验设计了新的操作符Extention和Shift。Extension操作符可以联接两个键值对实例，但不需要访问所有键值对中的所有数据。Shift操作符可以在具有不同键属性分布的键值对实例上启用类似集合的操作（例如，并集和差值）。

7.2 RDD的创新使用

在参考论文中，作者基于关系代数树的思想设计了KBA计划树，实现了KBA查询和交错并行执行的功能。树结构能实现交错并行执行的功能，但树结构相对复杂，并且查询效果不够好。

本实验使用了Spark中最基本的数据抽象——RDD，实现交错并行执行的功能，包括有界通信、并行可扩展性与横向可扩展性。Spark中最基本的数据抽象是RDD。RDD是一个不可变、可分区、里面的元素可并行计算的集合。相比于KBA查询计划，RDD具有数据流模型的特点：自动容错、位置感知性调度和可伸缩性。并且，RDD允许用户在执行多个查询时显式地将工作集缓存在内存中，后续的查询能够重用工作集，这极大地提升了查询速度。

7.3 TPC- H性能评价

TPC-H是事务处理性能委员会制定的基准程序之一，主要目的是评价特定查询的决策支持能力TPC-H中使用SF描述数据量，1SF对应1GB单位，SF由低到高依次是1、10、30、100、300、1000、3000、10000。TPC-H测试围绕22个SELECT语句展开，模拟决策支持系统中的数据库操作，测试数据库系统复杂查询的响应时间，以每小时执行的查询数作为度量指标。在sparksql现有的测试的基础上，采用TPC-H作为测试基准，来测试衡量改进后的统查询决策支持方面的能力，使得实验结果更具说服力。

# 第8章 结论

本实验借鉴了Zidian的设计思想，可以通过加快SQL查询的应答速度来改进现有的SQL-No-NoSQL系统。BaaV模型大大降低改进后的系统的数据访问和通讯成本。scan-free查询使得改进后的系统无需评估即可查询的查询扫描任何表，减少了盲目扫描和通信成本。在并行处理中，改进后的系统具有有界通信、并行可扩展性与横向可扩展性的属性，保证不扫描无扫描查询，并且保证有界查询的通信成本有限。结合论文中的实验结果和本实验结果可以证明，在SQL查询应答方面，基于Zidian思想改进后的spark sql性能大大优于现有的spark sql系统。

**参考文献**

[1] Block as a Value for SQL over NoSQL.Yang Cao.Wenfei Fan.Tengfei Yuan.