PG加速
(并行计算、JIT)

阿里云
digoal

目录

并行计算

并行计算相关参数

- max_worker_processes　　整个实例最多有多少工人
- max_parallel_workers　　同一时间窗口内，最多有多少工人被分配给并行计算
- max_parallel_workers_per_gather　一个并行计算子任务，最多分配多少工人
- parallel_leader_participation　　领导要不要和工人一起干
- parallel_setup_cost　　工人启动成本，唤醒工人的成本
- parallel_tuple_cost　　工人提成成本，功能每处理一条记录需要给多少提成
- min_parallel_index_scan_size　　触发并行计算的最小索引
- min_parallel_table_scan_size　　触发并行计算的最小表
- alter table xx set (parallel_workers =32);　扫描这张表相关的一个并行计算子任务，最多分配多少工人
- 是否启用并行?
  - **与代价相关，CBO**
  - **与参数开关有关，例如关闭了并行，设置了works_per_gather=0, 或表级 workers=0**
  - **与当前剩余多少工人有关， max_parallel_workers-当前窗口已分配工人数**
- 自动并行度计算:　**min(coalesce (**table_parallel_workers, compute_by(rel_size)),

max_parallel_workers_per_gather,

max_parallel_workers-当前窗口已分配工人数**)**

强制并行度24

```
max_worker_processes=128;  -- 启动数据库时设置
set max_parallel_workers=128;
set max_parallel_workers_per_gather=24;
set parallel_leader_participation=off;
set parallel_setup_cost=0;
set parallel_tuple_cost=0;
set min_parallel_index_scan_size=0;
set min_parallel_table_scan_size=0;
set alter table xx set (parallel_workers =24);
explain (analyze,verbose,timing,costs,buffers) query;
-- build索引相关
set max_parallel_maintenance_workers=n;  -- 建立索引时的并行度
set maintenance_work_mem='xGB';
```

例子

- 10亿，无索引条件过滤
- 10亿，哈希、分组聚合
- 10亿，自定义函数计算
- 10亿，产生分析中间表
- 10亿，无索引排序
- 10亿，JOIN（nestloop, merge, hash）
- 10亿，创建索引
- 10亿，索引扫描
- 10亿，子查询
- 10亿，CTE
- 10亿，分区表与分区表 智能并行 JOIN

测试表

create table t1 (id int, c1 int2, c2 int2, c3 int2, c4 int2, c5 int, c6 text, c7 timestamp);

insert into t1 select generate_series(1,1000000000), random()*10, random()*100, random()*1000, random()*10000, random()*100000, md5(random()::text), clock_timestamp();

create table t2 (id int, c1 int2, c2 int2, c3 int2, c4 int2, c5 int, c6 text, c7 timestamp);

insert into t2 select generate_series(1,1000000000), random()*10, random()*100, random()*1000, random()*10000, random()*100000, md5(random()::text), clock_timestamp();

alter table t1 set (parallel_workers =32);

alter table t2 set (parallel_workers =32);

强制并行

```
set max_parallel_workers=128;
set max_parallel_workers_per_gather=24;
set parallel_leader_participation=off;
set parallel_setup_cost=0;
set parallel_tuple_cost=0;
set min_parallel_index_scan_size=0;
set min_parallel_table_scan_size=0;
alter table t1 set (parallel_workers =24);
alter table t2 set (parallel_workers =24);
```

并行无索引条件过滤、分组聚合、哈希聚合

```
select c1,count(*) from t1 where c2<=50 group  by c1;
set enable_hashagg =off;
select c1,count(*) from t1 where c2<=50 group  by c1;
```

并行自定义函数计算

```
create or replace function udf(text) returns boolean as $$
  select hashtext($1) < 10000;
$$ language sql strict immutable parallel safe;
select count(*) from t1 where udf(c6);
```

并行产生分析中间表

```
create unlogged table t_1 as select c1,count(*) from t1 group by c1;
```

并行无索引排序

```
select * from t1 order by id desc limit 10;
select * from t1 order by id limit 10;
select * from t1 order by c1 desc limit 10;
select * from t1 order by hashtext(c6) limit 10;
select * from t1 order by hashtext(c6) desc limit 10;
```

并行哈希JOIN

```
select t1.c1,count(*) from t1 join t2 using (id) where t2.c2<5 group by t1.c1;
select t1.c1,count(*) from t1 join t2 using (id) group by t1.c1;
```

并行创建索引

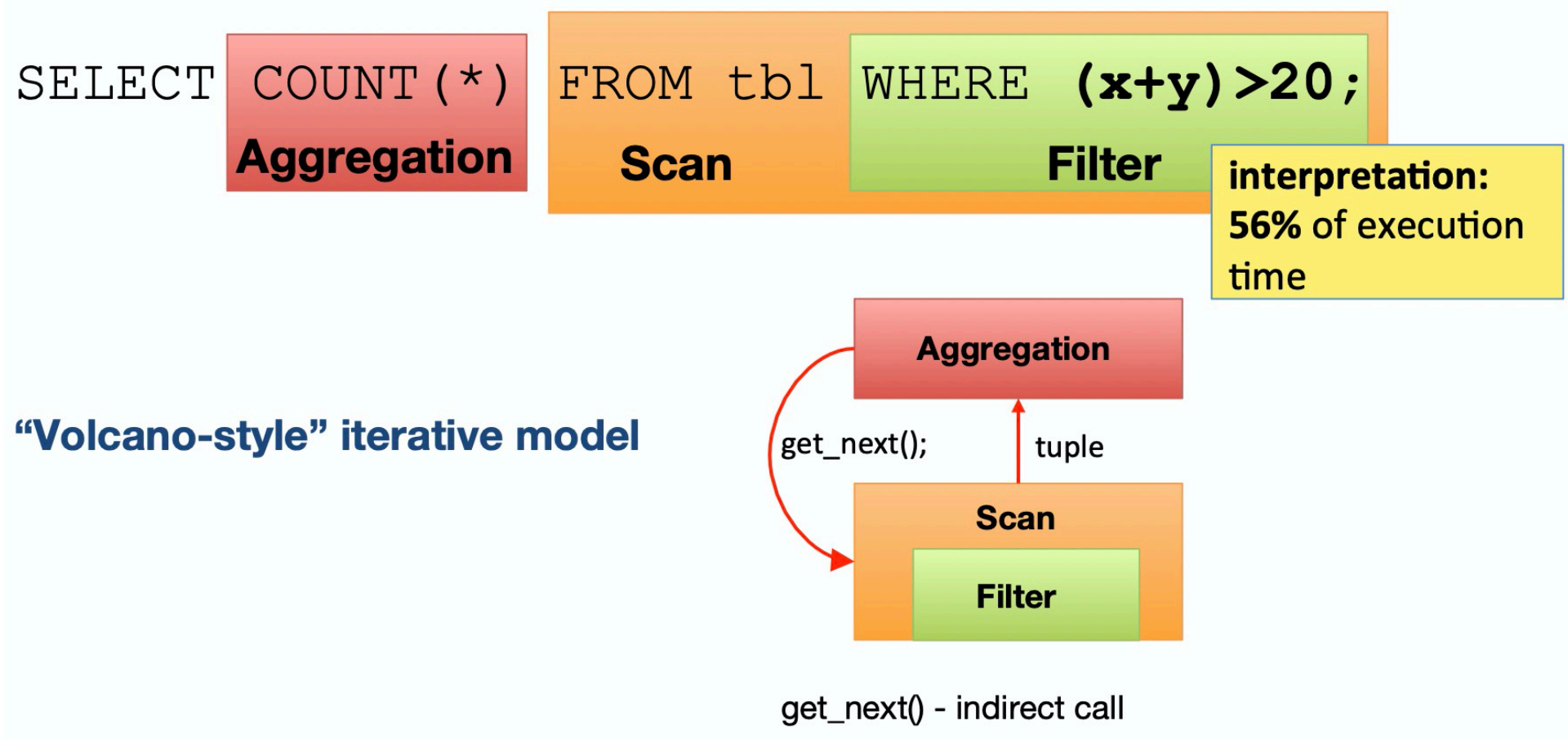
```
set max_parallel_maintenance_workers=8;
set maintenance_work_mem='2GB';
create index idx_t1_1 on t1 using btree (c1);
```

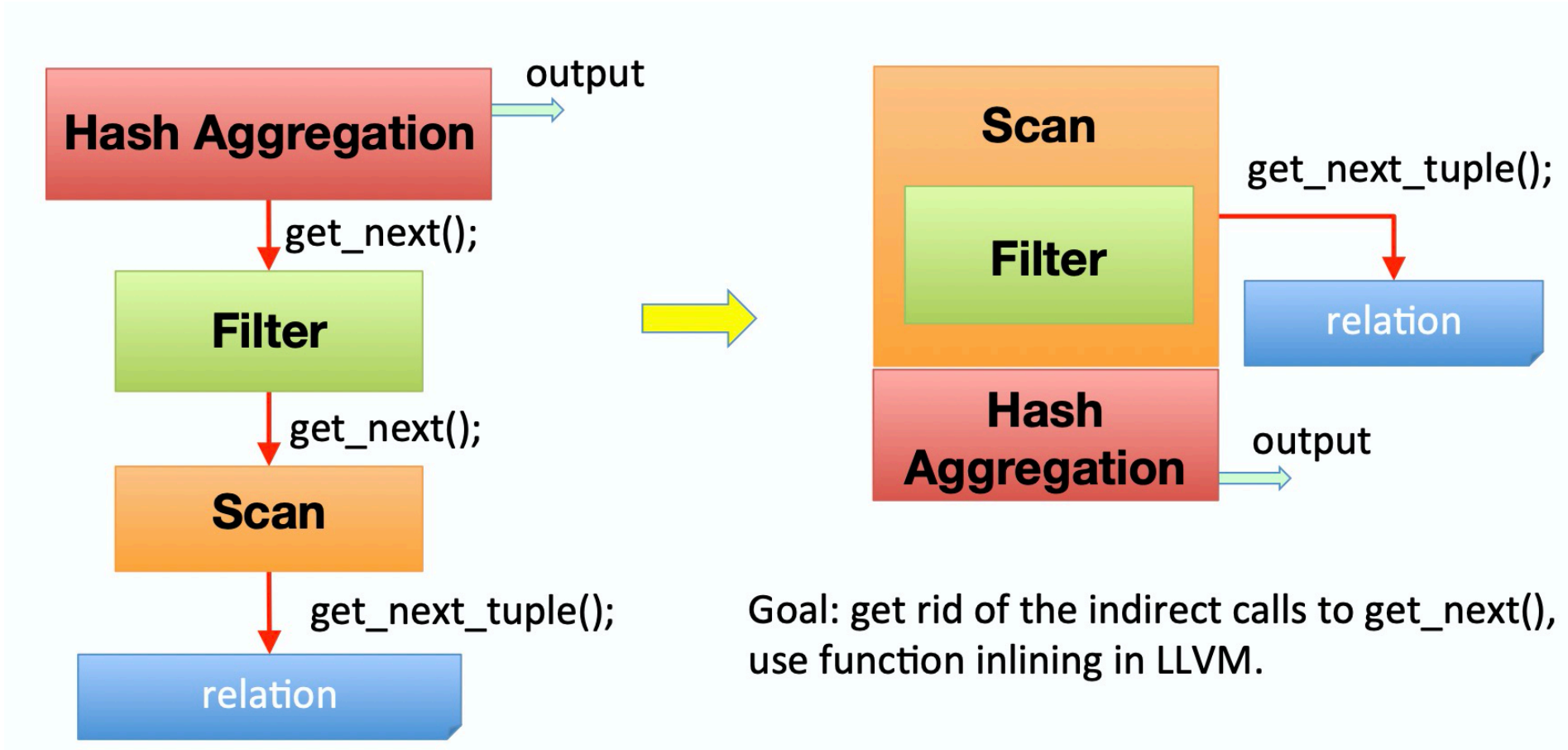
并行索引扫描

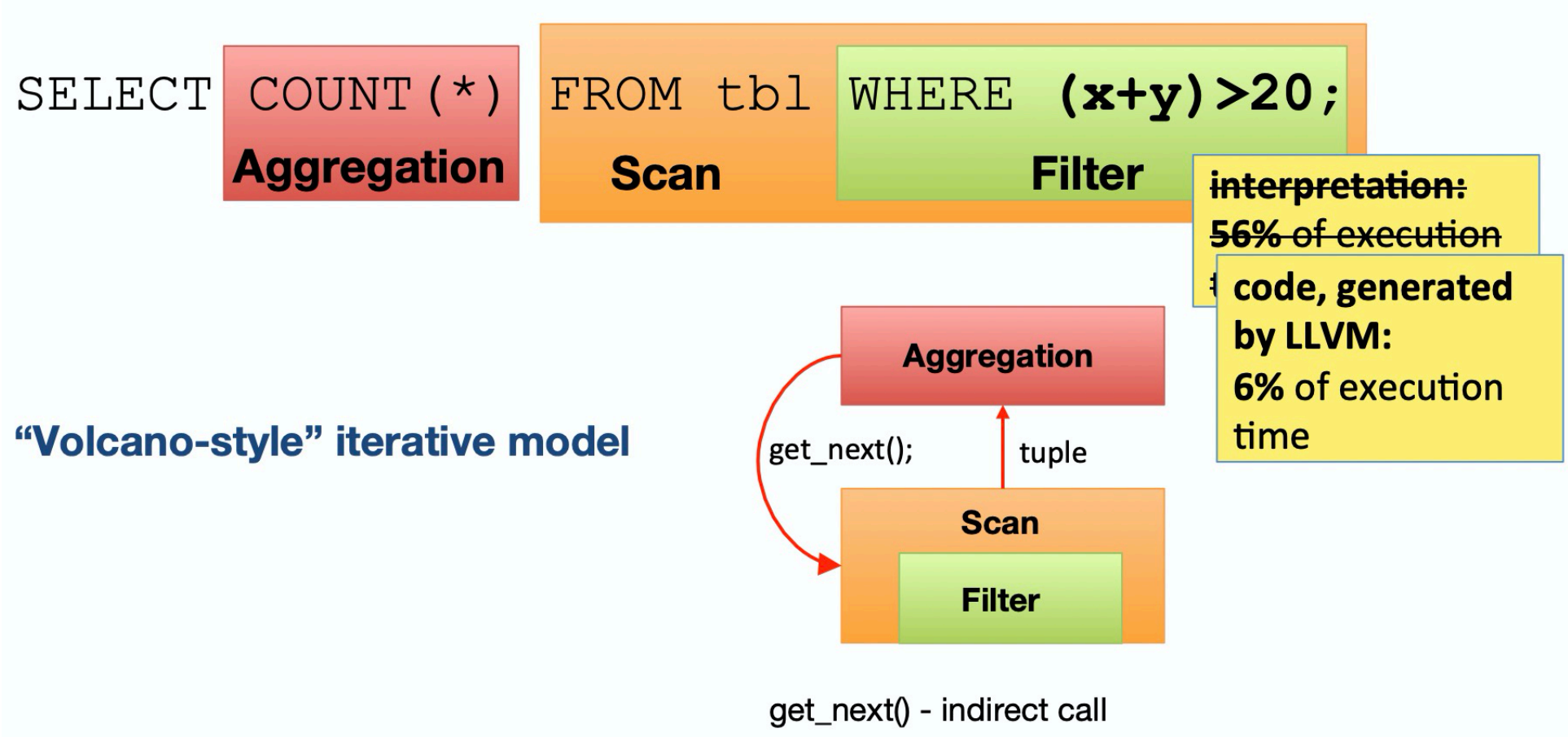
```
select count(*) from t1 where c1=1;
```

- 10亿， 并行nestloop join
- https://github.com/digoal/blog/blob/master/201903/20190317_09.md
- 10亿， 并行merge join
- https://github.com/digoal/blog/blob/master/201903/20190317_10.md
- 10亿，并行子查询
- https://github.com/digoal/blog/blob/master/201903/20190318_02.md
- 10亿，并行CTE
- https://github.com/digoal/blog/blob/master/201903/20190318_04.md
- 10亿，分区表与分区表 智能并行 JOIN
- https://github.com/digoal/blog/blob/master/201903/20190317_12.md
- 10亿，分区表与分区表 智能并行 聚合
- https://github.com/digoal/blog/blob/master/201903/20190317_13.md

JIT


SELECT
COUNT(*)
Aggregation
FROM tbl
Scan
WHERE (x+y)>20;
Filter
interpretation:
56% of execution time
"Volcano-style" iterative model
Aggregation
get_next();
tuple
Scan
Filter
get_next() - indirect call


Hash Aggregation
output
get_next();
Filter
get_next();
Scan
get_next_tuple();
relation
Scan
Filter
get_next_tuple();
relation
Hash Aggregation
output
Goal: get rid of the indirect calls to get_next(), use function inlining in LLVM.


SELECT
COUNT(*)
Aggregation
FROM tbl
Scan
WHERE (x+y)>20;
Filter
interpretation:
56% of execution
code, generated by LLVM:
6% of execution time
"Volcano-style" iterative model
Aggregation
get_next();
tuple
Scan
Filter
get_next() - indirect call

JIT 参数

- jit 是否开启jit
- jit_provider 谁出品的llvm编译器
- jit_above_cost 超过这个代价的SQL启用jit
- jit_inline_above_cost 超过这个代价的SQL启用jit inline 动态编译
- jit_optimize_above_cost 超过这个代价的SQL启用jit optimize代码动态编译
- jit_expressions 非cost控制：是否开启jit表达式动态编译
- jit_tuple_deforming 非cost控制：是否开启jit tuple deform动态编译
- jit_profiling_support 是否支持jit的profile，用于诊断jit优化效果
- jit_dump_bitcode 开发者参数，用于打印jit的动态编译代码
- jit_debugging_support 开发者参数，用于输出jit debug信息

例子

```
create table test(id int, c1 int, c2 int, c3 int, c4 int, c5 int, c6 int, c7 int, c8 int, c9 int);
insert into test select generate_series(1,10000000),1,2,3,4,5,6,7,8,9;
explain verbose select avg(c1),min(c1),max(c1),count(*), avg(c2),min(c2),max(c2), avg(c3),min(c3),max(c3),
avg(c4),min(c4),max(c4), avg(c5),min(c5),max(c5), avg(c6),min(c6),max(c6), avg(c7),min(c7),max(c7),
avg(c8),min(c8),max(c8), avg(c9),min(c9),max(c9)
from test
where c1<1 or c2<1 or c3<1 or c4<1 or c5<1 or c6<1 or c7<1 or c8<1 or c9<10
group by c1 order by c1;
```

参考资料

- 并行计算文档
  - https://github.com/digoal/blog/blob/master/201903/20190318_05.md
- MySQL手册
  - https://www.mysqltutorial.org/
  - https://dev.mysql.com/doc/refman/8.0/en/
- PG 管理、开发规范
  - https://github.com/digoal/blog/blob/master/201609/20160926_01.md
- PG手册
  - https://www.postgresql.org/docs/current/index.html
  - https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-vs-mysql/
- GIS手册
  - http://postgis.net/docs/manual-3.0/

一期开课计划(PG+MySQL联合方案)

- - 2019.12.30 19:30 RDS PG产品概览，如何与MySQL结合使用
- - 2019.12.31 19:30 如何连接PG，GUI，CLI的使用
- - 2020.1.3 19:30 如何压测PG数据库、如何瞬间构造海量测试数据
- - 2020.1.6 19:30 MySQL与PG对比学习(面向开发者)
- - 2020.1.7 19:30 如何将MySQL数据同步到PG（DTS）
- - 2020.1.8 19:30 PG外部表妙用 - mysql_fdw, oss_fdw（直接读写MySQL数据、冷热分离）
- - 2020.1.9 19:30 PG应用场景介绍 - 并行计算，实时分析
- - 2020.1.10 19:30 PG应用场景介绍 - GIS
- - 2020.1.13 19:30 PG应用场景介绍 - 用户画像、实时营销系统
- - 2020.1.14 19:30 PG应用场景介绍 - 多维搜索
- - 2020.1.15 19:30 PG应用场景介绍 - 向量计算、图像搜索
- - 2020.1.16 19:30 PG应用场景介绍 - 全文检索、模糊查询
- - 2020.1.17 19:30 PG 数据分析语法介绍
- - 2020.1.18 19:30 PG 更多功能了解：扩展语法、索引、类型、存储过程与函数。如何加入PG技术社群

本课程习题

- 并行计算适合什么应用场景
- 并行度是优化器自动计算的，取决于哪些因素
- 如何设定强制并行度
- 为什么分析型场景使用JIT可以提升性能，推拉模型的区别是什么

技术社群

PG技术交流钉钉群(3500+人)