openLooKeng：
如何实现跨源跨域的高性能融合分析

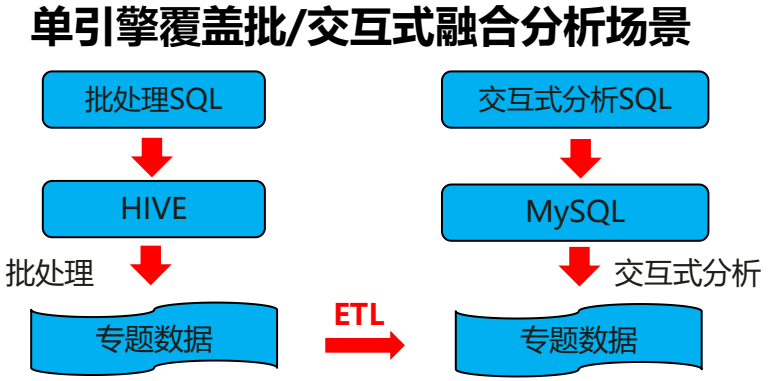
余吉文

openLooKeng Community Tech PM & User Group Member

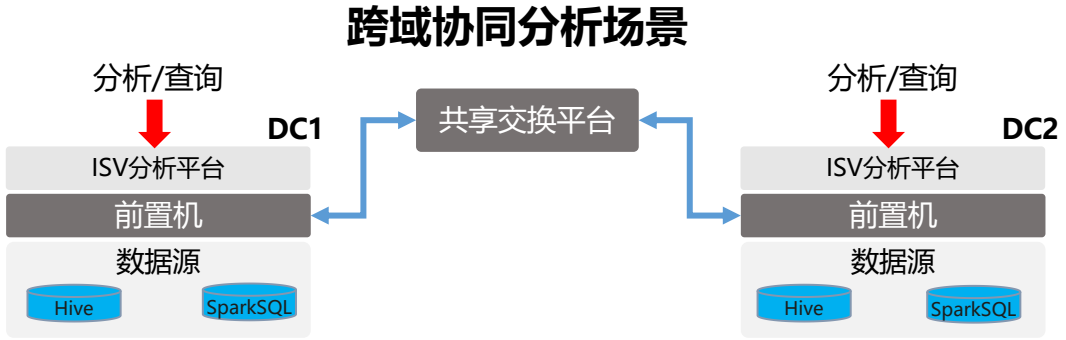
目录

目录

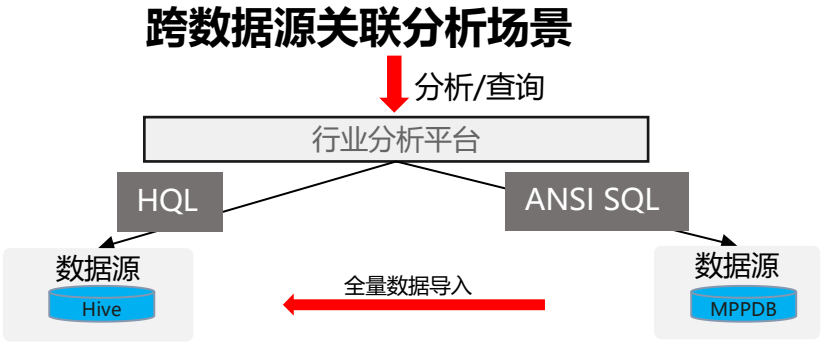
大数据分析面临的挑战



痛点1：两个烟囱，两份数据，管理复杂



痛点3：工作流人工处理，难以支撑T+0分析



痛点2：引擎接口不统一，编程模型复杂

三大需求：

- 批/交互式融合分析
- 跨数据源关联查询
- 跨域协同分析

openLooKeng-面向大数据的融合分析引擎

安平
警务大数据

政府
政务大数据 | 部委大数据

金融
金融数据湖

运营商
运营商大数据

大企业
企业数据湖

数据源

- 关系型数据
- 日志数据
- 外部数据
- 传感器(IoT)
- WEB
- 社交媒体
- 3rd party

入湖

数据使能：数据集成 | 数据开发 | 数据治理 | 虚拟数仓

计算引擎

查询引擎

批计算	流计算	融合分析	图计算	搜索
Hive, Spark	Flink	openLooKeng	GraphBase, GeoMesa, HBase	ElasticSearch

YARN

AI：机器学习 | 深度学习 | 推理引擎

数据管理

数据目录 — Catalog

数据安全 — Security

数据存储

数据存储 — HDFS（TXT | ORC | Parquet | Carbon）

分布式存储（FS-HDFS | 对象 | 文件）

管理

- 安全管理
- 租户管理
- 配置管理
- 性能管理
- 故障管理

鲲鹏服务器 | X86服务器 | 虚拟机 | 云主机

openLooKeng：统一高效的数据虚拟化融合分析引擎，让大数据变简单

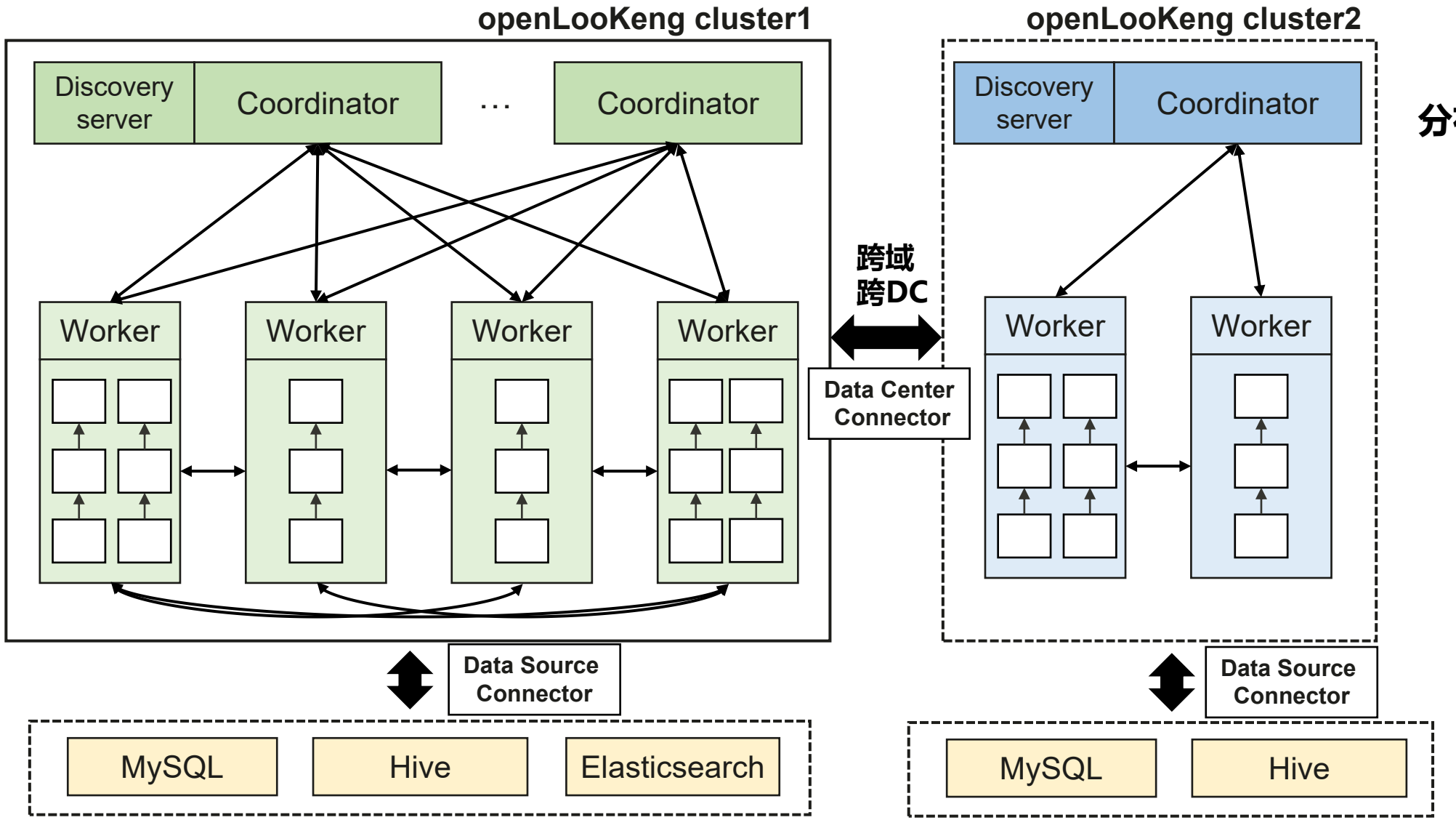
统一入口，化繁为简，单一引擎支持多场景

内核增强，高性能查询

跨源关联分析，数据消费零搬移

跨域协同计算，广域网的部署，局域网的体验

openLooKeng架构



分布式处理系统，MPP架构

高可用性，无单点故障

向量化列式处理引擎

基于内存的流水线处理

目录

交互式查询特点

随机性
用户SQL不可预测，具有很强的随机性

数据量适中
数据存在data warehouse中

并发支持
一般需要提供50-100并发支持

交互式查询

所需结果集小
通常所需结果集较小，使用limit或者取消

端到端时间敏感
秒级/分钟级返回，查询涉及资源不敏感

跨源跨域
需要跨DC或者跨源支持

交互式查询特点

随机性
用户SQL不可预测，具有很强的随机性

数据量适中
数据存在data warehouse中

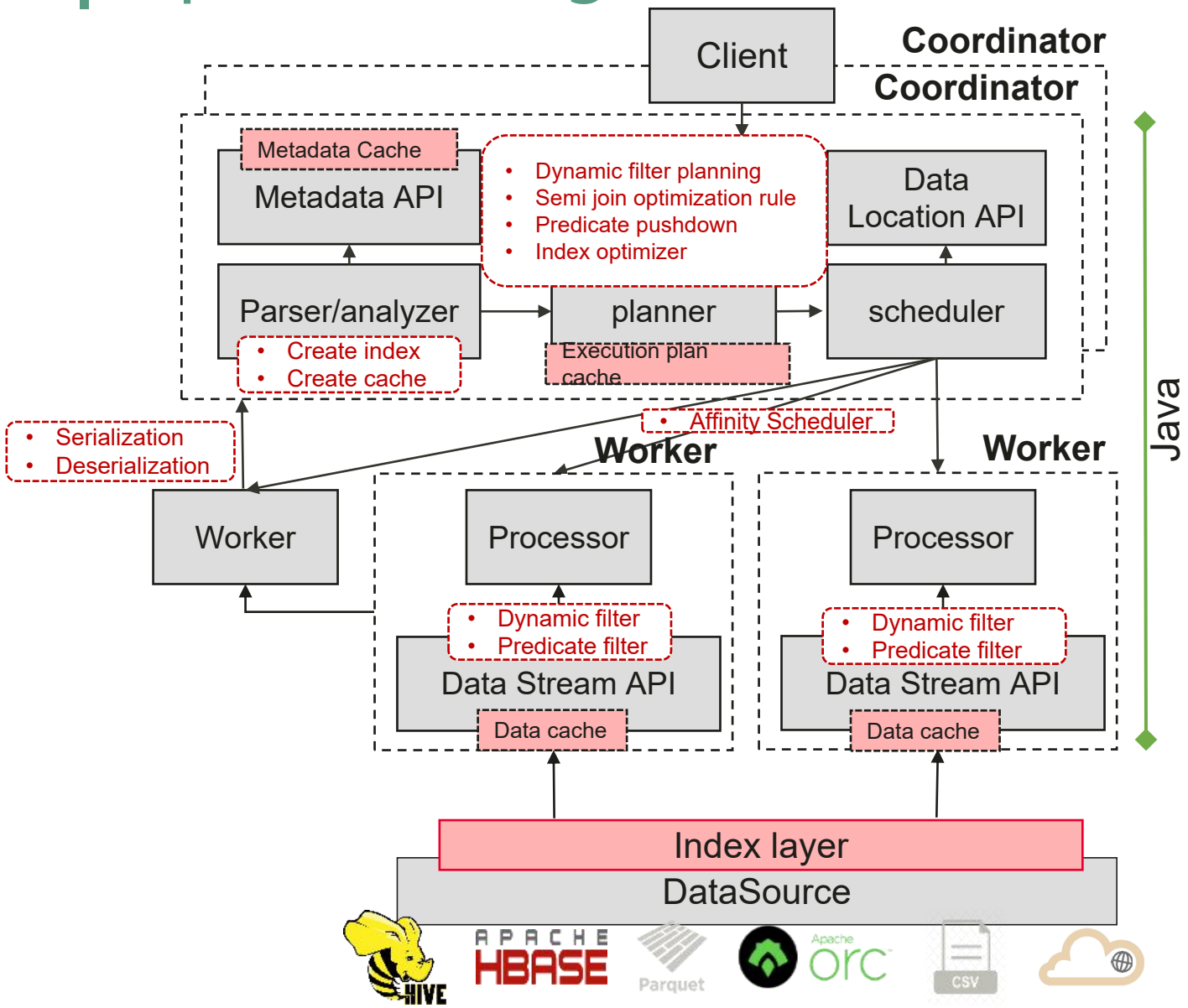
并发支持
一般需要提供50-100并发支持

所需结果集小
通常所需结果集较小，使用limit或者取消

端到端时间敏感
秒级/分钟级返回，查询涉及资源不敏感

跨源跨域
需要跨DC或者跨源支持

openLooKeng交互式场景关键技术



- **数据源侧，更适应openLooKeng**
  - 分桶/分区
  - 小文件合并
  - 查询字段排序
- **引擎层，增强交互式查询能力**

- 缓存加速：
  - 执行计划缓存
  - 元数据缓存
  - 增量列式缓存
- 优化器：
  - 谓词下推
  - **动态过滤**
  - RBO&CBO
  - **CTE**
- 自适应调度器

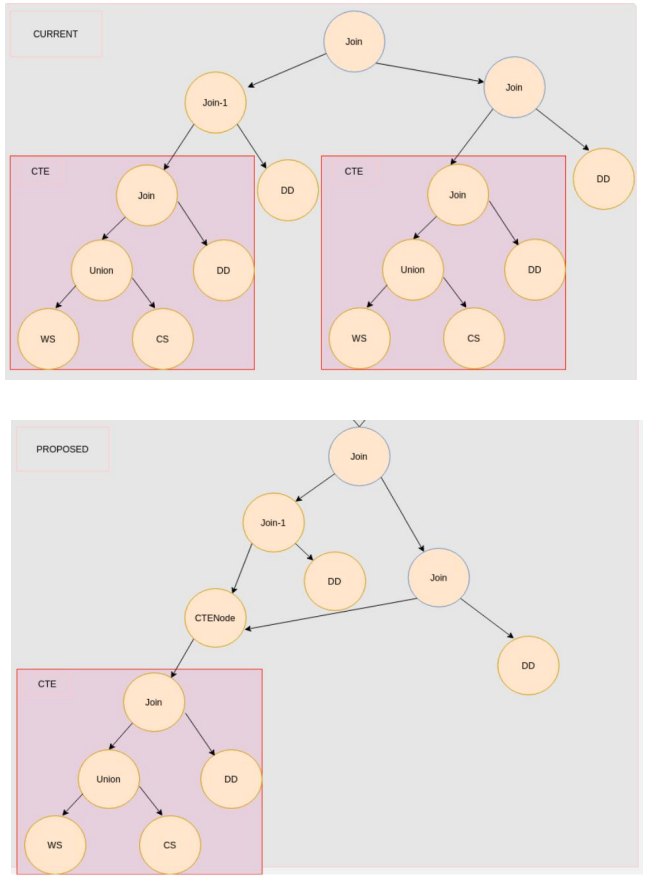
- **额外层，加速交互式查询**
  - **Heuristic index layer**

  (bitmap/bloomfilter/min-max)
  - Data cache layer
  - 序列化&反序列化

CTE （Common Table Expression）执行计划优化

```sql
--Q47
with v1 as(
 select i_category, i_brand,
        s_store_name, s_company_name,
        d_year, d_moy,
        sum(ss_sales_price) sum_sales,
        avg(sum(ss_sales_price)) over
          (partition by i_category, i_brand,
          avg_monthly_sales,
        rank() over
          (partition by i_category, i_brand,
 from item, store_sales, date_dim, store
 where ss_item_sk = i_item_sk and
       ss_sold_date_sk = d_date_sk and
       ss_store_sk = s_store_sk and
       (
 group by i_category, i_brand,
          s_store_name, s_company_name,
          d_year, d_moy),
 v2 as(
 select v1.i_category, v1.i_brand, v1.s_store_name, v1.s_company_name
        ,v1.d_year, v1.d_moy
        ,v1.avg_monthly_sales
        ,v1.sum_sales, v1_lag.sum_sales psum, v1_lead.sum_sales nsum
 from v1, v1 v1_lag, v1 v1_lead
 where v1.i_category = v1_lag.i_category and
       v1.i_category = v1_lead.i_category and
       v1.i_brand = v1_lag.i_brand and
       v1.i_brand = v1_lead.i_brand and
       v1.s_store_name = v1_lag.s_store_name and
       v1.s_store_name = v1_lead.s_store_name and
       v1.s_company_name = v1_lag.s_company_name and
       v1.s_company_name = v1_lead.s_company_name and
       v1.rn = v1_lag.rn + 1 and
       v1.rn = v1_lead.rn - 1)
  select  *
 from v2
 where  d_year = 2000 and
        avg_monthly_sales > 0 and
        case when avg_monthly_sales > 0 then abs(sum_sales - avg_monthly_sales) / avg_monthly_sales else null end > 0.1
 order by sum_sales - avg_monthly_sales, psum
 limit 100;
```

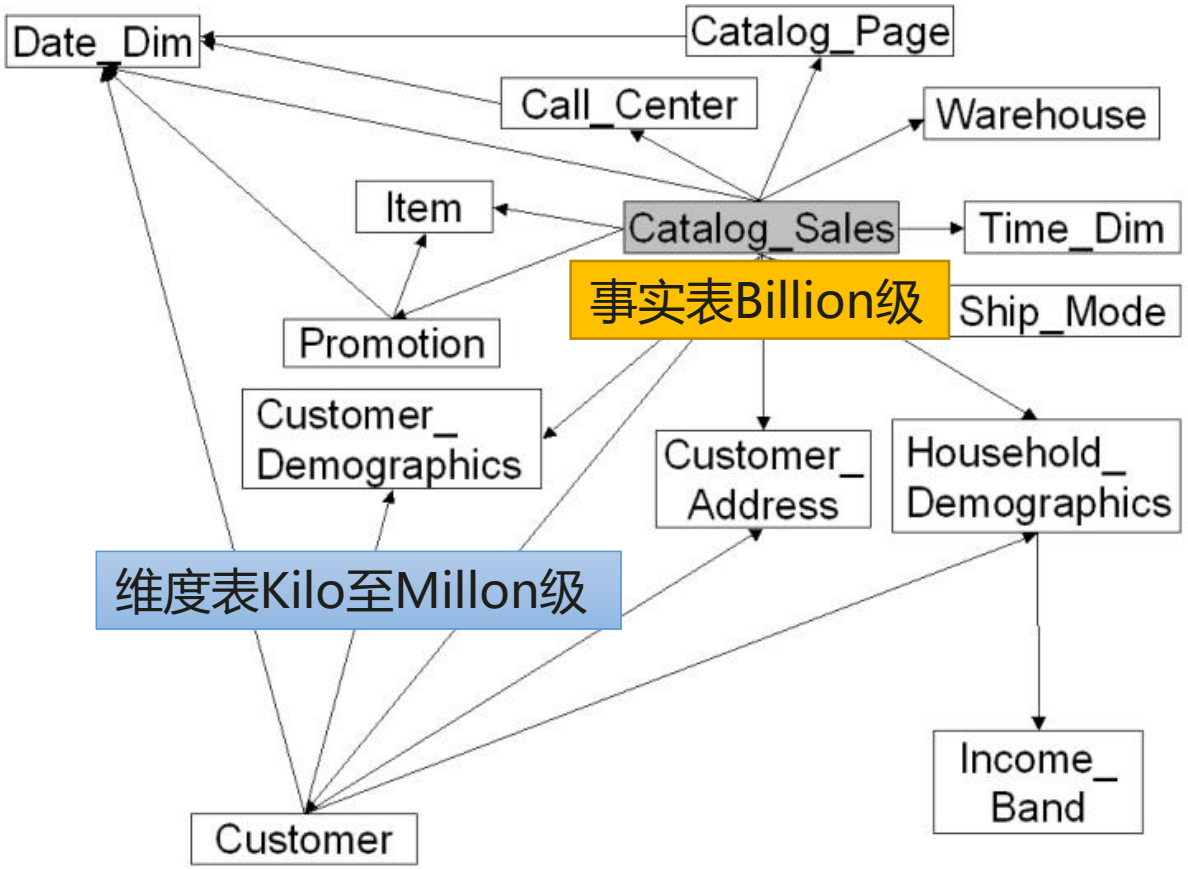
以TPC-DS Q47为代表的查询
查询性能提升**50%+**
（10TB，180s 降至 63s）



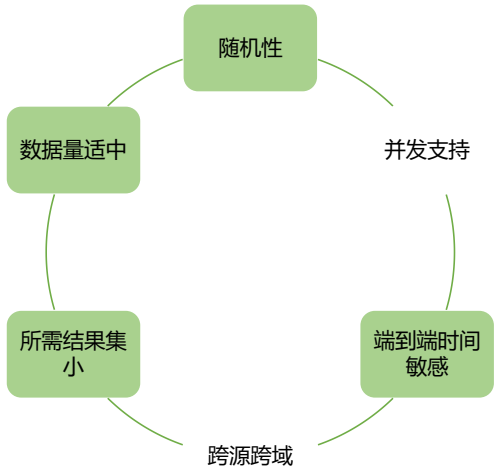
Single-stream scenario	tpcds-10TB		
	不开启CTE	开启CTE	性能提升
Total(s)	10591	8873	16%

TPC-DS场景分析

TPC-DS是TPC标准组织推出的一个广泛使用的行业标准决策支持基准，用于评估数据处理引擎的性能



Catalog Sales表ER图



场景维度分析

TPCDS数据模型

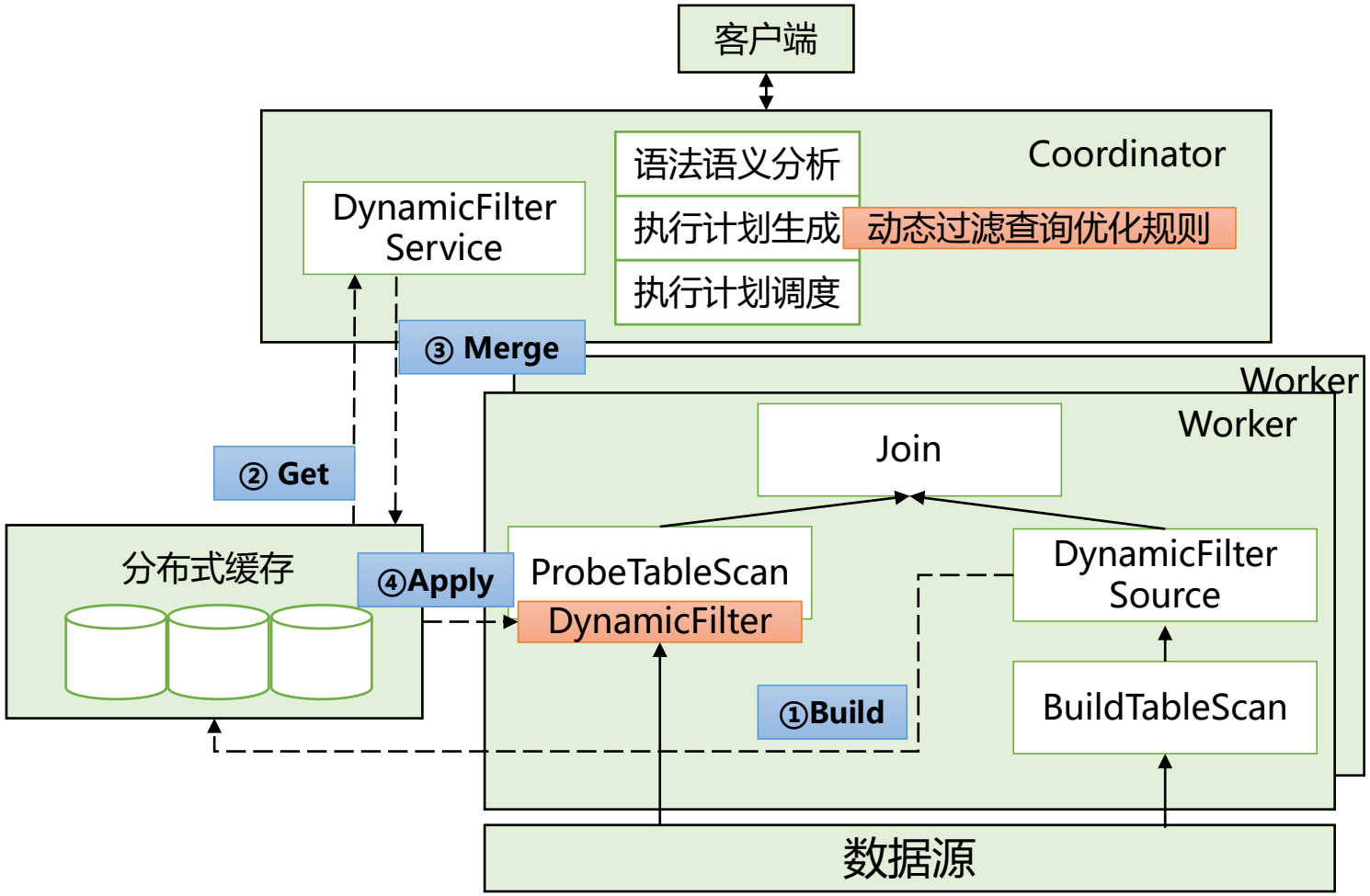
- 包含7张事实表和17张维度表
- 事实表数据量极大，而维度表相对较小
- **查询很少有谓词直接应用到事实表**
- **事实表查询条件通过维度表相连接得到**

带来的挑战

- **传统谓词下推等优化很难应用**
- **Join数据量巨大导致执行时间过长**

Dynamic Filtering

依靠join条件以及build侧表读出的数据，运行时生成动态过滤条件（dynamic filters），应用到probe侧表的table scan阶段，从而减少参与join操作的数据量，有效地减少IO读取与网络传输

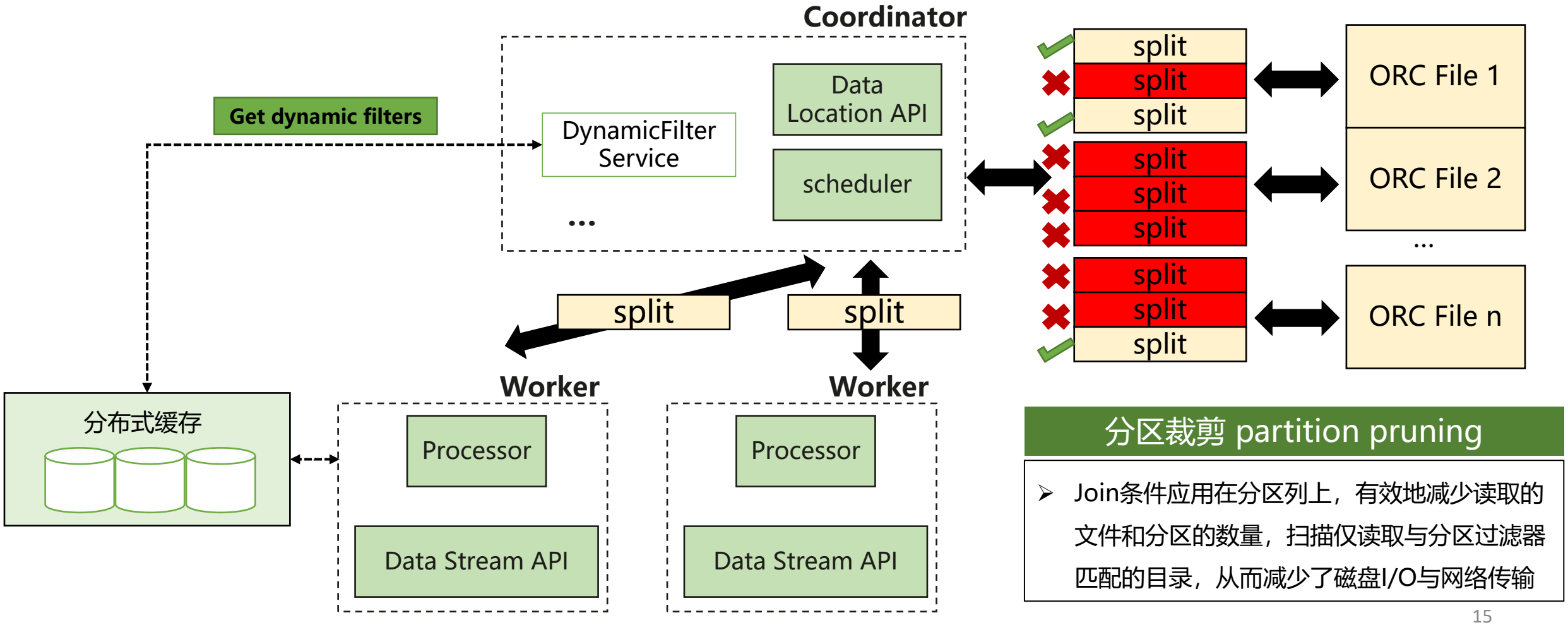


Dynamic Filtering

- 添加DynamicFIlterSource算子，搜集build侧数据
- 依赖分布式缓存进行DF的处理
- 适用于inner join & right join
- **适用于join选择率较高的场景**

Dynamic Filtering

/user/hive/warehouse/tpcds_bin_partitioned_orc_1000.db/store_sales
/**ss_sold_date_sk=2452638**/000997_0



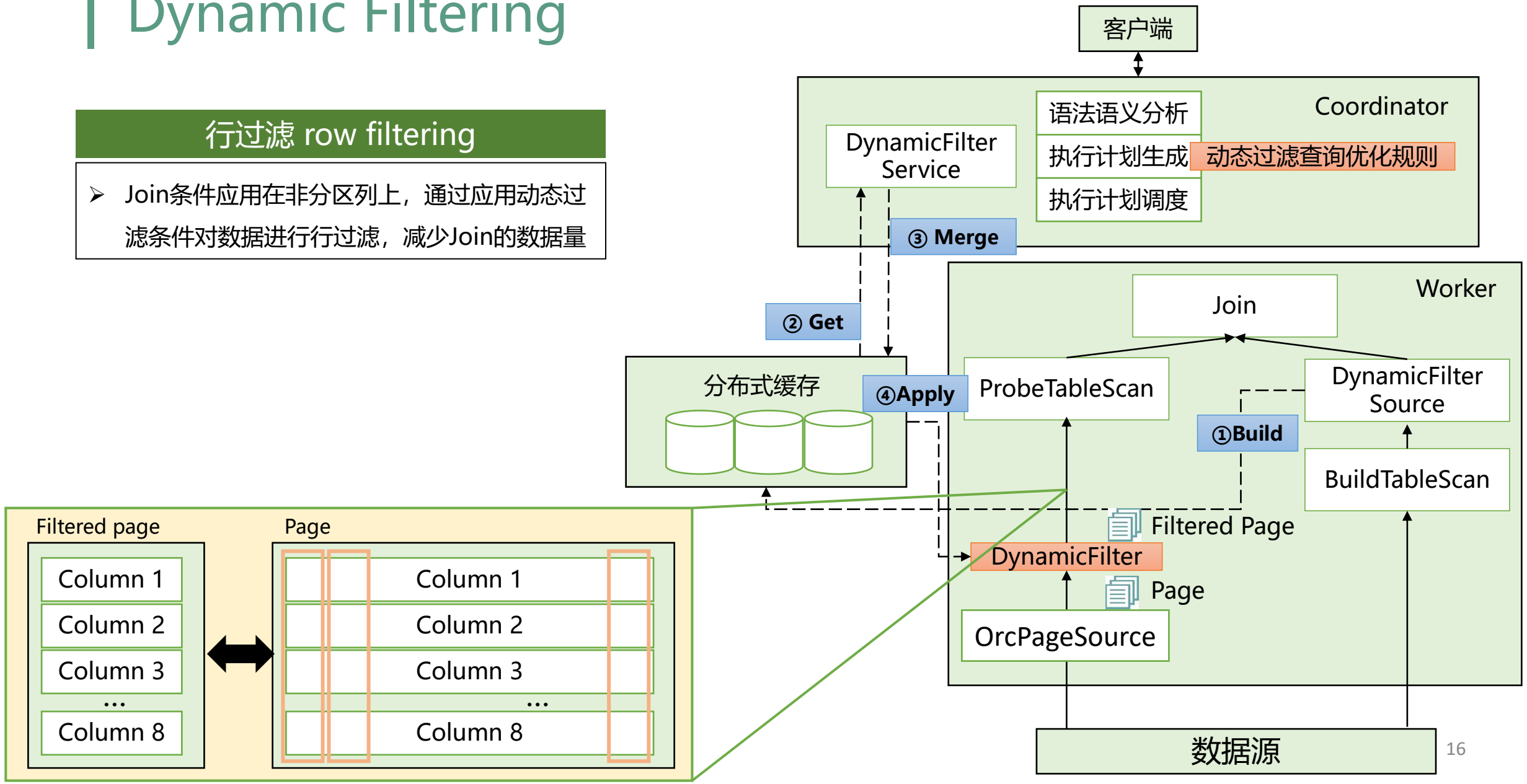
分区裁剪 partition pruning

- Join条件应用在分区列上，有效地减少读取的文件和分区的数量，扫描仅读取与分区过滤器匹配的目录，从而减少了磁盘I/O与网络传输

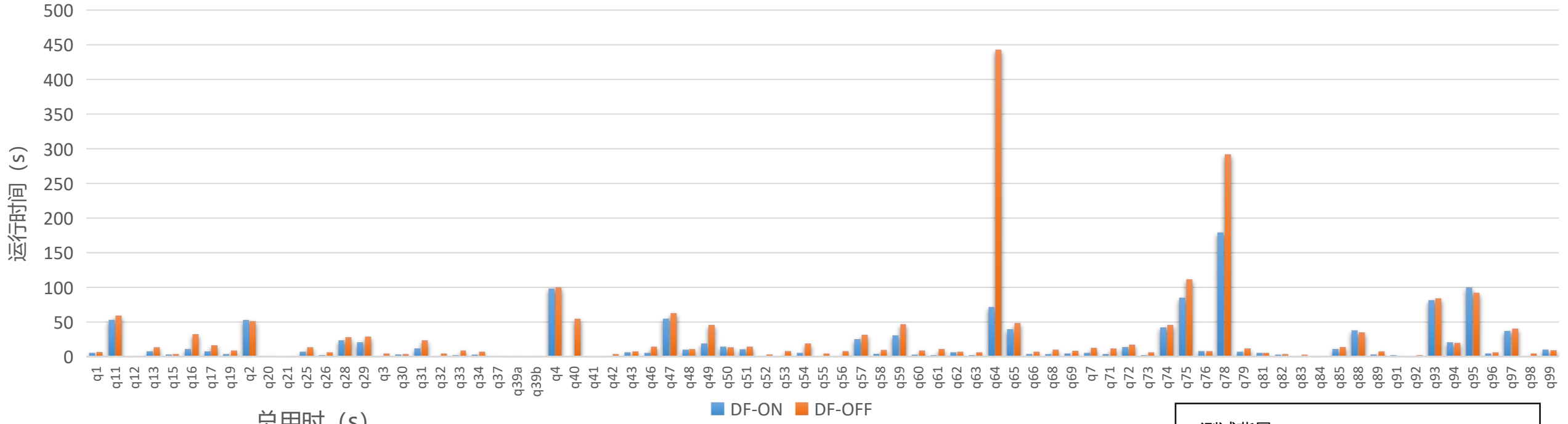
Dynamic Filtering

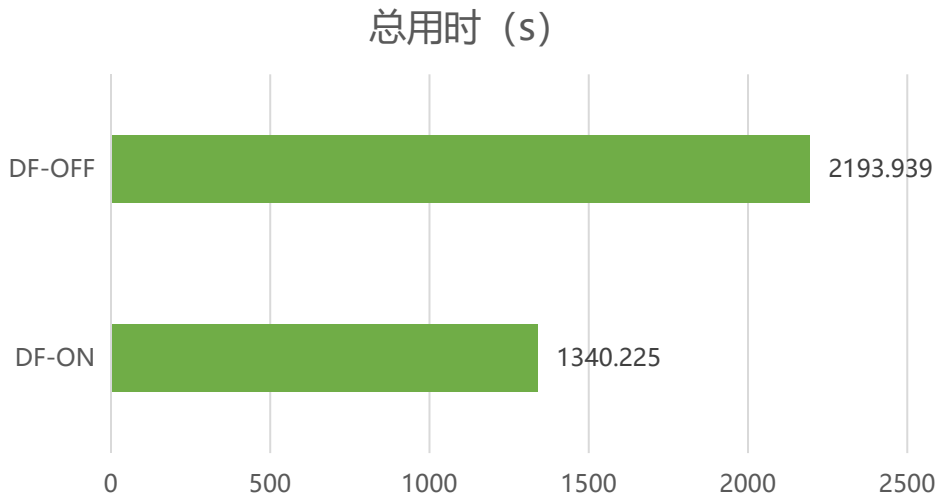
行过滤 row filtering

- Join条件应用在非分区列上，通过应用动态过滤条件对数据进行行过滤，减少Join的数据量



性能测试



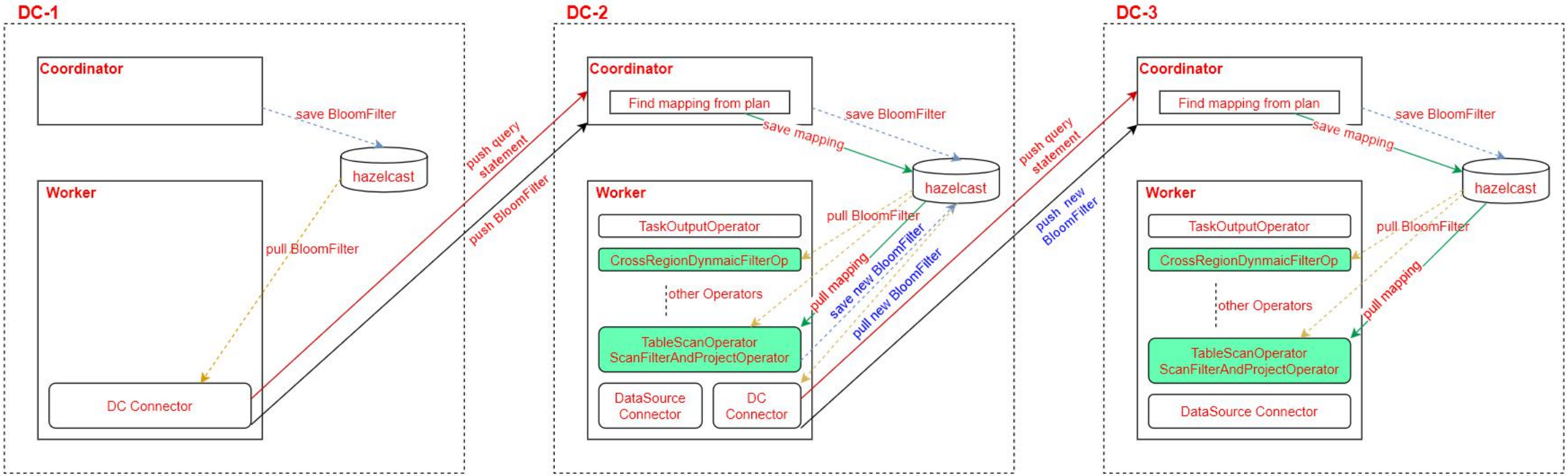


结论及后续优化

- 结论
  - TPC-DS测试用例总用时openLooKeng开启动态过滤，执行时间减少**38.9%**
- 后续优化
  - Predicate pushdown优化
  - Dynamic filtering等待及应用优化

测试背景：
数据集：2TB TPCDS
节点：11计算节点
内存：376GB
Cpu：2*Gold 6140 CPU @ 2.30GHz
OS：RedHat 7.3
*openLooKeng基于master分支

跨域Dynamic Filtering



DC-2 Coordinator： 1）将DC-1的BF filter以QueryId为Key存入到hazelcast；2）判断当前query是否存在跨域dynamic filter，存在，设置session中的cross-region-dynamic-filter；3）CN生产执行计划Plan，从Plan中Query的列名到Plan的outputSymbols的映射关系，存入hazelcast；4）判断Plan的TableScanNode是否存在DC table，如存在，则标记，可能存在继续下推BF filter的可能。

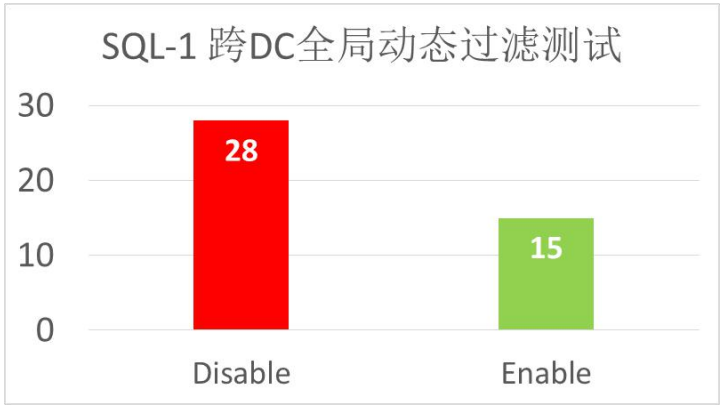
DC-2 Worker： 1) CrossRegionDynamicFilterOp从hazelcast中取出BF filter和outputSymbols，判断是否存在过滤列，存在则应用filter对Page进行过滤；2）TableScanOperator应用filter和步骤一类似；3）如果TableScanNode存在DC table，则生成新的BF filter并存入hazelcast，用于发送给下一级DC。

跨域Dynamic Filtering 性能

测试环境：每个DC是一个单节点openLooKeng，内存200GB，CPU：2*Gold 6140 CPU @ 2.30GHz，OS：RedHat 7.3

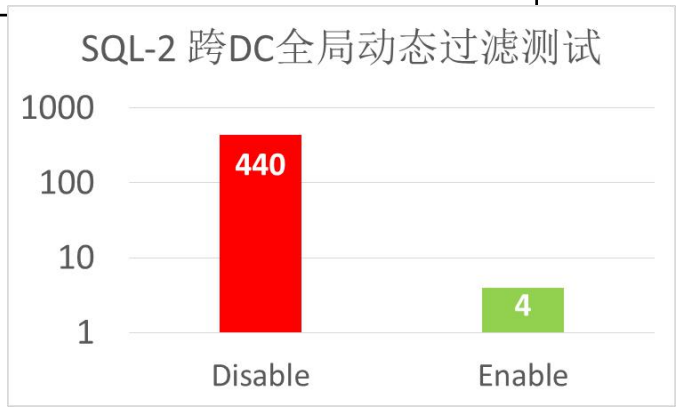
SQL-1：

```
Select a.* From
(
        Select collect_place,  first_time, last_time from
        dc.vdm.test.identity_net_stat_10million
        Union
        Select collect_place,  first_time, last_time from
        dc.vdm.test.identity_net_stat_10million
) a
Join hive.test.identity_net_stat_10million b
On a.first_time = b.first_time
Where  b.msidsn =  '13812345678'
```

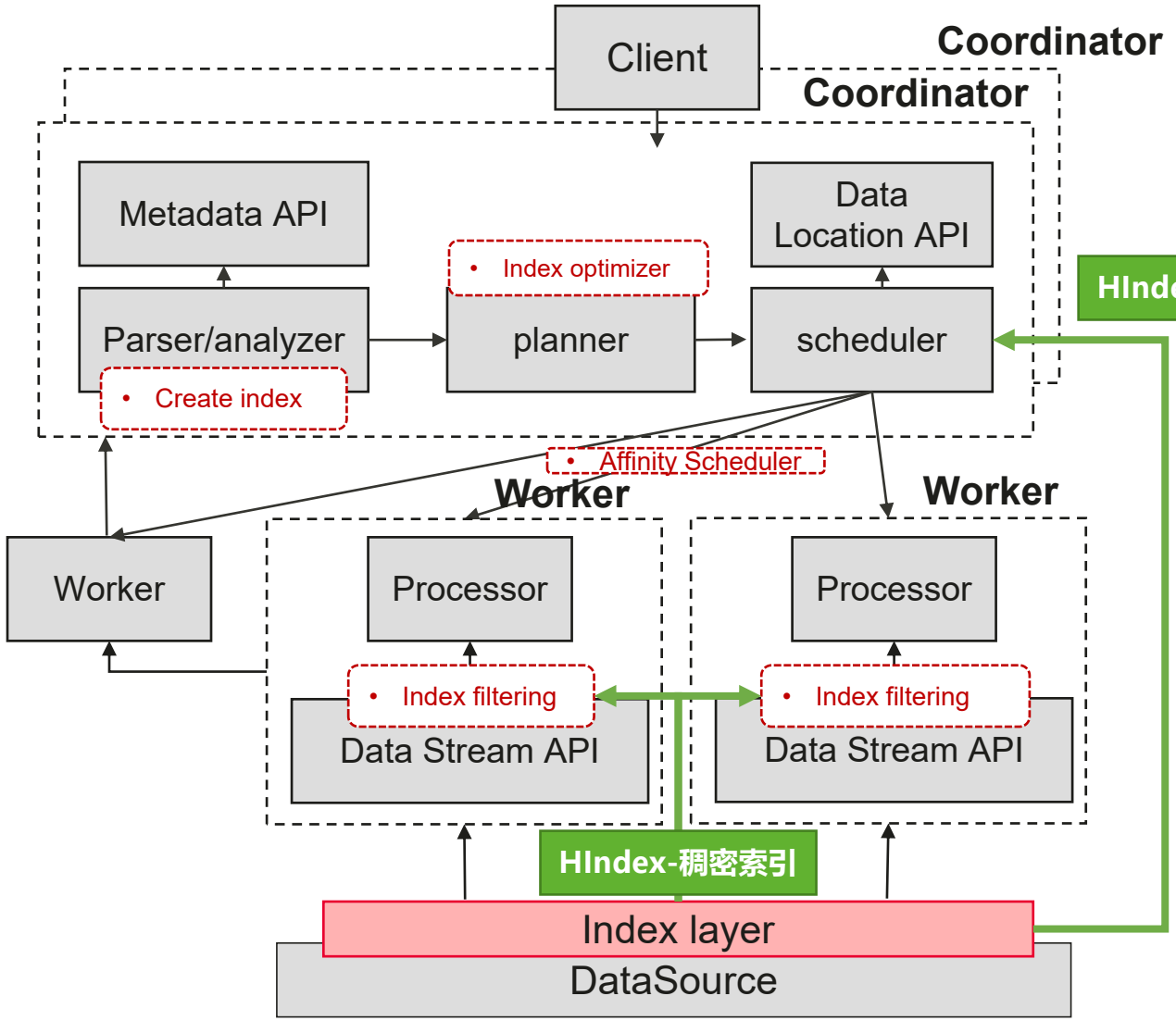


SQL-2

```
Select count(a.first_time) From
(
        Select collect_place,  first_time, last_time From
        dc.vdm.test.identity_net_stat_10million m
        Join
        Select collect_place,  first_time, last_time From
        dc.vdm.test.identity_net_stat_10million n
        On m.collect_place =  n.collect_place
) a
Join hive.test.identity_net_stat_10million b
On a.first_time = b.first_time
Where  b.msidsn =  '13812345678'
```



Heuristic index架构

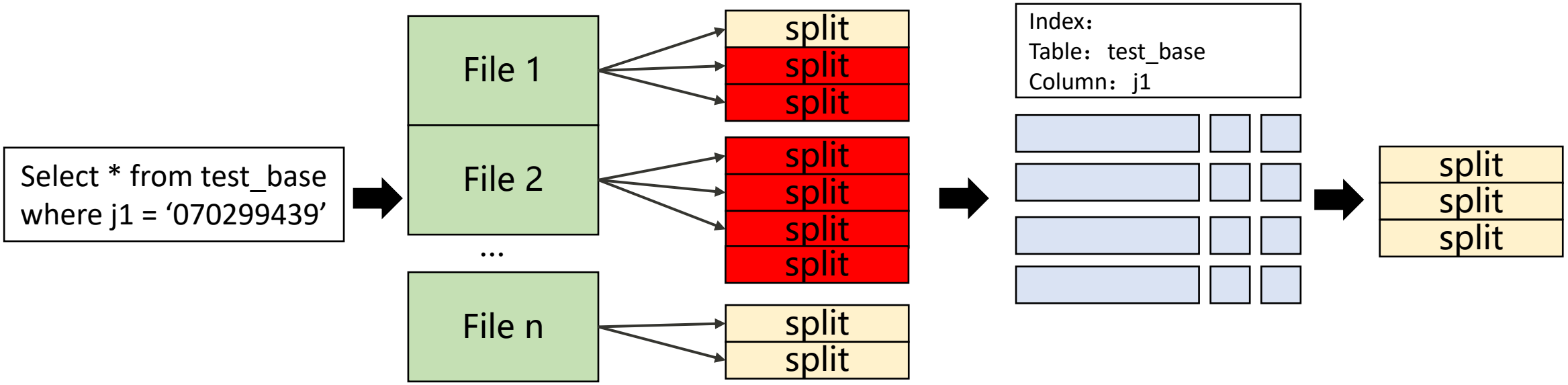


Heuristic index

- 提供统一的索引框架
- 支持多种索引结构
  - 稀疏索引：Bloomfilter、Min-Max
  - 稠密索引：Bitmap，Btree
- 任务调度阶段：
  - 裁剪Split,减少调度到Worker的任务数
  - 支持基于索引的亲和性调度
- 数据读取阶段：
  - 减少加载到计算侧内存的数据量

Heuristic index- Bloom filter索引

Bloom filter索引，确定每个split是否包含要搜索的值，并只对可能包含该值的split进行读操作

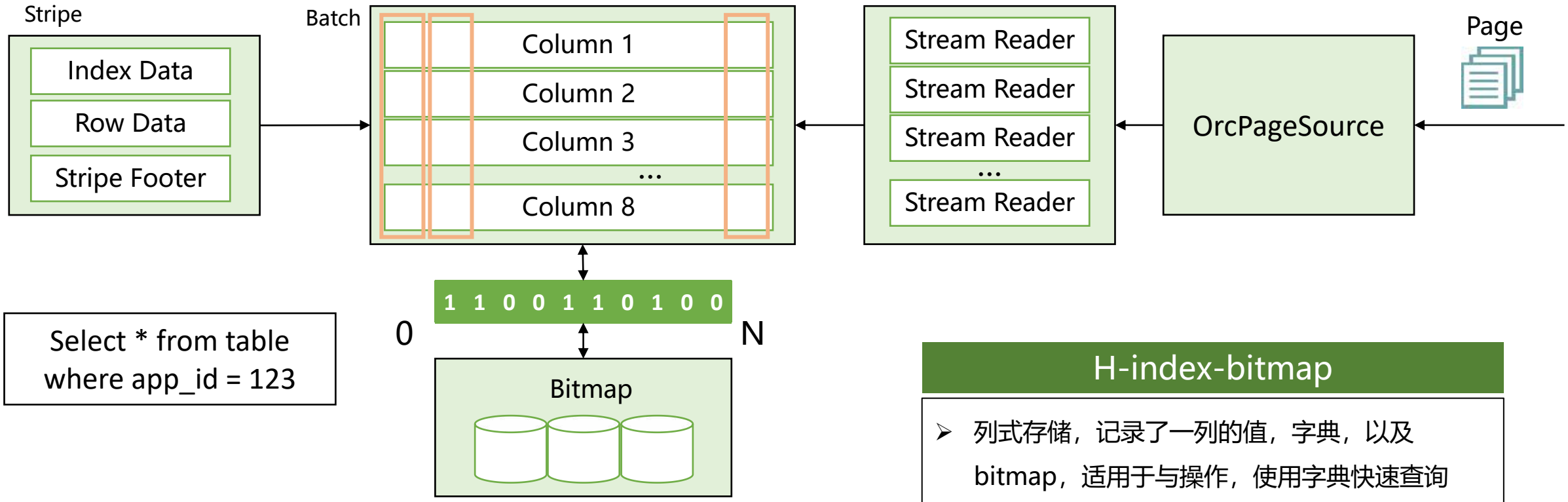


H-index-bloom filter

- 可以快速判断一个集合中有无某个值（只支持等于符号）
- 需要预先通过create index进行索引创建
- 通过在coordinator侧过滤，减少不必要的split生成与处理

Heuristic index – 位图索引

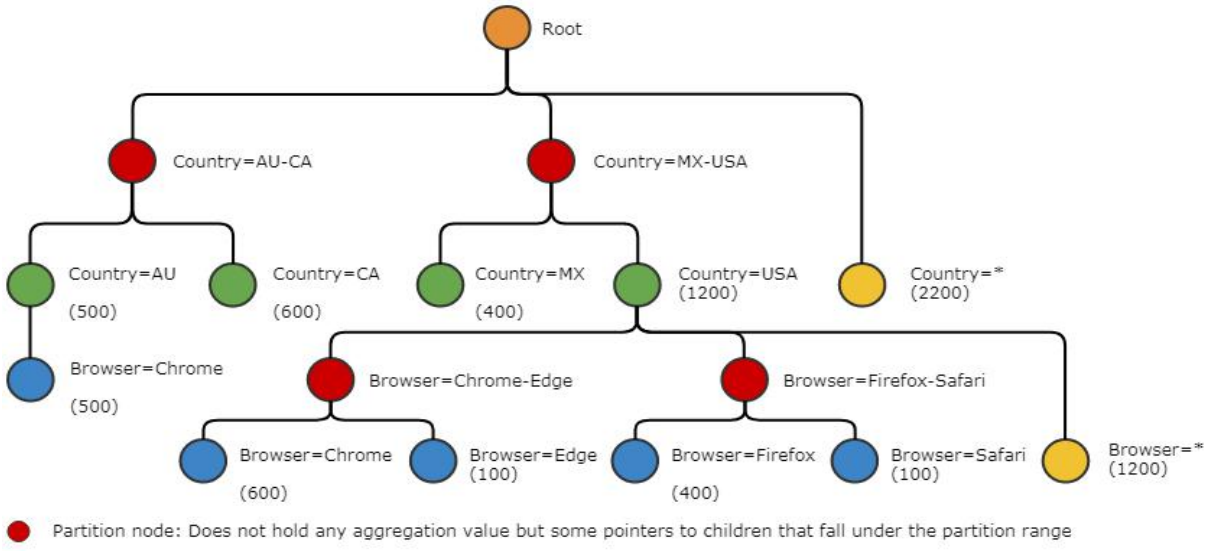
Bitmap索引，通过为谓词列保留外部位图索引，可以过滤掉与谓词不匹配的行

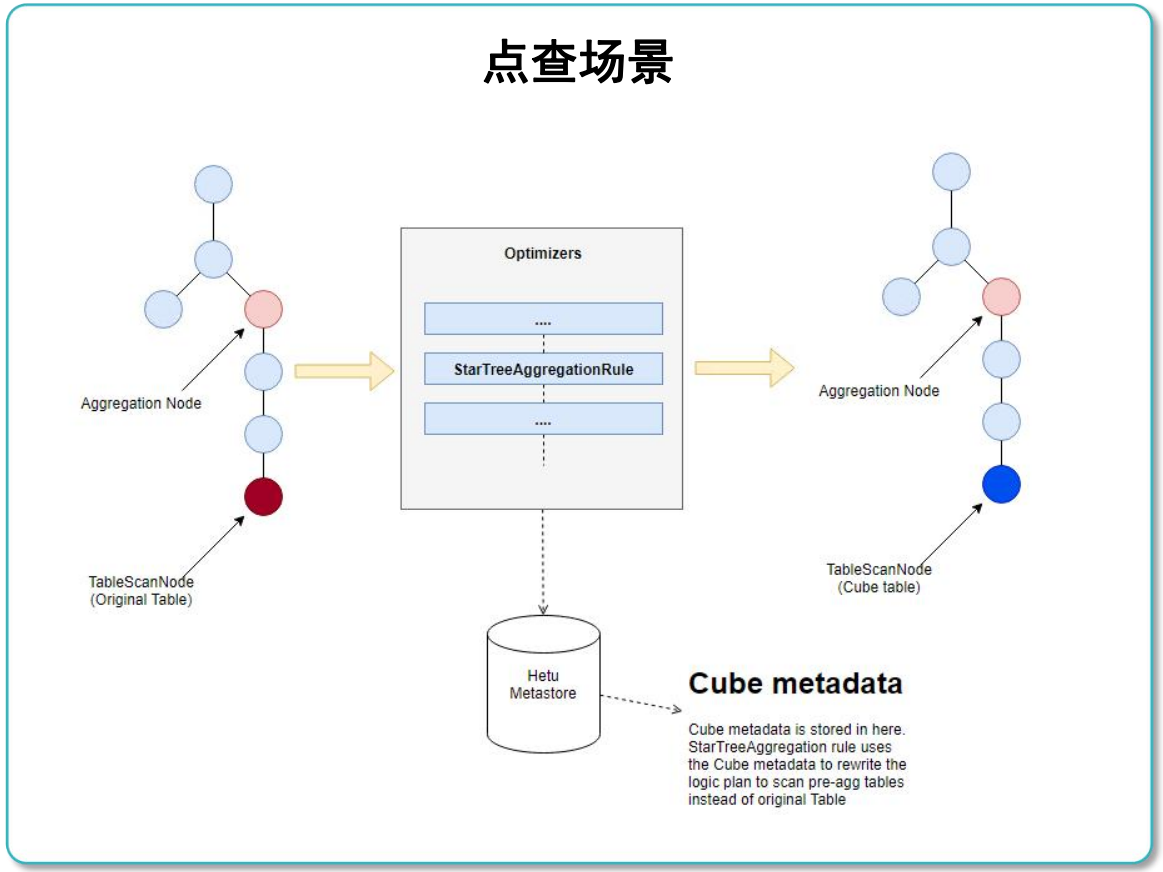


H-index-bitmap

- 列式存储，记录了一列的值，字典，以及bitmap，适用于与操作，使用字典快速查询
- 需要预先通过create index进行索引创建

Heuristic index – StarTree index

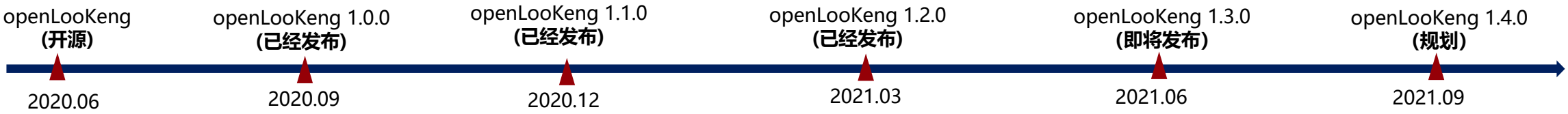




Star Tree技术：通过预聚合技术，降低高基维的查询的延迟，达到ms级别的查询要求

目录

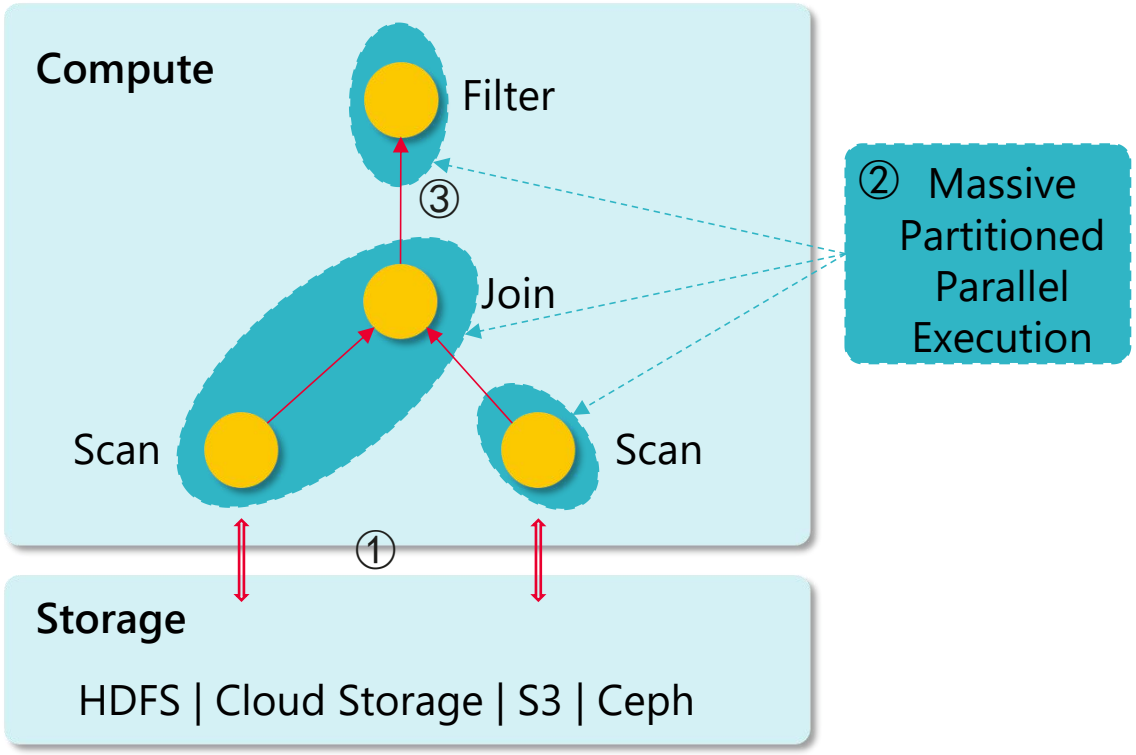
openLooKeng 开源一周年 – 每3个月一个正式版本



2020年6月30日openLooKeng 0.1.0版本在社区发布，提供统一SQL接口，具备跨源/跨域分析能力，支持交互式查询场景，同时构筑了启发式索引、动态过滤、高可用AA、弹性伸缩、动态UDF等竞争力特性

版本	1.0.0	1.1.0	1.2.0	1.3.0
概要	**支持IUD for ORC，支持数据虚拟集市**	**跨DC的动态过滤增强**	**DM优化+通用算子下推框架**	**资源隔离+可靠性增强**
高性能	算子下推、动态过滤增强、执行计划缓存	启发式索引增强、针对TPC-DS性能优化	Data Management优化、基于星树索引的预聚合、CTE Reuse	CBO增强，支持Sorted Source Aggregator，减少内存使用
南北向生态	北向SQL语法转换工具支持HQL/Impala语法 南向支持10+数据源	南向对接更多数据源：openGauss、MongoDB、ElasticSearch7.x、hive metastore用户透传	北向兼容性增强 南向提供新的通用算子下推框架，更加简单、灵活和高效 HBase Connector性能优化	新增hudi connector，GreenPlum connector，clickhouse connector JDBC Connector支持多分片查询 memory connector 功能增强
企业级	容器化部署、Try-me、SQL Editor	高并发能力增强、对接Ranger权限管理、Admin Dashboard	细粒度权限管控、查询重试增强	资源隔离,可靠性增强(task level recovery)

从openLooKeng看大数据引擎性能的优化方向



① 数据加载：

- 存算协同：存算间缺乏有效协同

② 数据计算：

- 有效计算：算力消耗在无关控制流上
- 加速比：大规模集群的加速比小于0.5
- CPU/网络：受限数据处理逻辑无法充分利用

③ 数据交换：

- 数据交换格式：不能根据网络特性进行自动调整
- 序列化：序列化、反序列化的性能损耗
- Zero-Copy：内存和操作系统缓冲区的数据拷贝

目录

总结

- openLooKeng愿景是让大数据更简单
  - 统一SQL入口，数据免搬迁
  - 高性能的交互式查询能力
  - 多源异构数据源融合分析
  - 跨域跨DC融合分析
- 优化技术

  openLooKeng从如下三个方面进行交互式查询优化：
  - 让数据源更好的适配引擎（分区/小文件合并）
  - 引擎自身优化（动态过滤/RBO/CBO/谓词下推/缓存）
  - 添加额外层加速（Heuristic index layer/Cache layer/序列化/反序列化）

Thank you

openLooKeng, Big Data Simplified

openLooKeng微信公众号

openLooKeng微信小助手