一种支持实时复杂查询和分析的NoSQL系统

中科院计算所
INSTITUTE OF COMPUTING TECHNOLOGY

王树鹏

中国科学院计算技术研究所

题 纲

- 系统需求与现有方案
- 技术方案
- 应用案例

系统需求概述

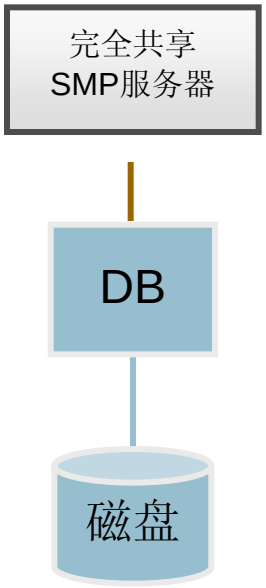
- 数据及系统特点
  - 结构化：每条记录包含10个字段左右，每条记录的大小大约是几百字节
  - 数据量巨大：达到千亿级以上，达到PB级
  - 加载速度快：达到百万条/s的规模
  - 系统规模：可以扩展到上千个节点

系统需求概述

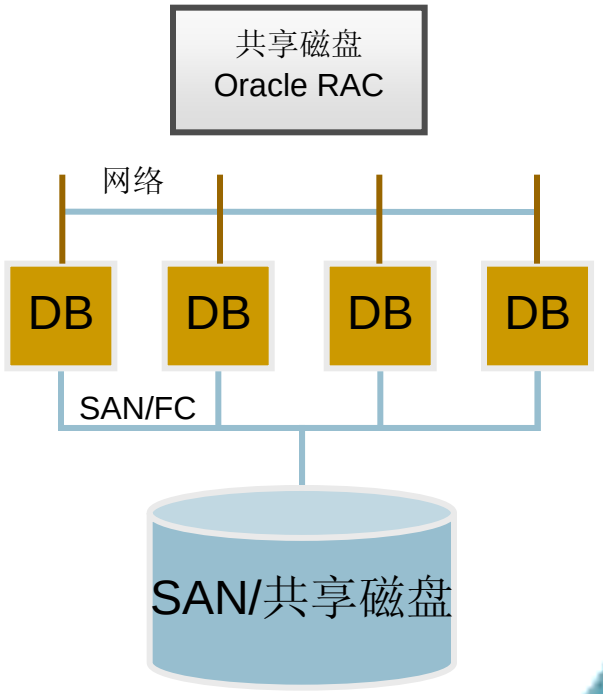
- 对数据的访问需求
  - 提供SQL访问接口
  - 支持大规模结果集：达到千万条规模
  - 支持按多列的实时查询（秒级）
    - 支持多列之间的逻辑比较关系，例如AND、OR、NOT等
    - 支持多列之间的的算术比较关系，例如=、>、<等
  - 支持统计、聚合、分组、排序等操作（秒级）
    - ORDER BY ASC (DESC)，GROUP BY, TOP, LIMIT
    - SUM，COUNT，AVG，MAX，MIN
  - 数据不更新，但需要对数据批量删除

现有方案(1)—关系数据库

- 关系型数据库都主要关注了CA，即一致性和可用性
- 性能、可扩展性上都比较差
- 无法满足可扩展性和性能的要求



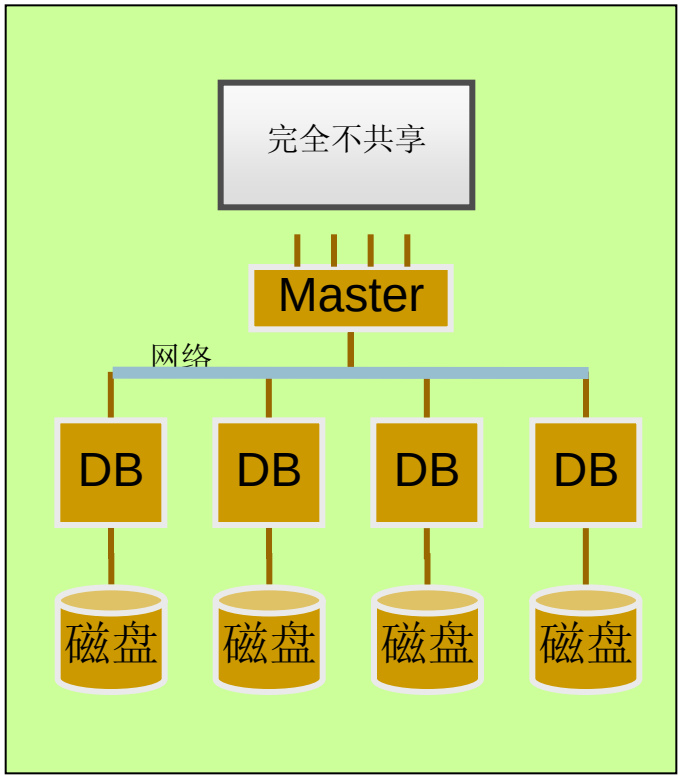
单机数据库结构



Oracle RAC

现有方案(2)—关系数据库集群

- 数据分片(sharding)或者功能分区
- 将数据按照不同的策略进行划分：功能、字段值范围、HASH等
- 优点：能够有效的解决可扩展性的问题
- 缺点：shard的扩容比较复杂；联合多个shard的表数据查询复杂。



Parallel DBMS

现有方案(3)-NoSQL方案

- NoSQL（非关系型）
  - NoSQL ≠No SQL，而是No Relationship，Not Only SQL
- 系统特点
  - 可以处理超大规模的数据，可支持到千亿规模
  - Sharing-Noting架构，可扩展性强
  - 数据加载速度快，并可随节点个数线性增长

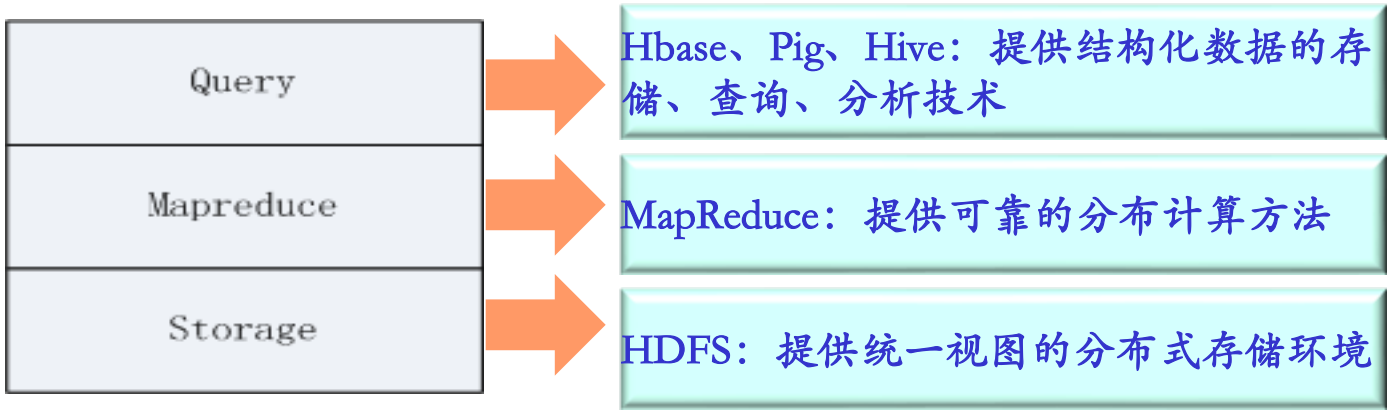
现有方案(3)-NoSQL方案

- 根据特定应用场景的需要设计开发了很多NoSQL系统
  - 分布式KV型：例如：Dynamo，PNUTS、Flare
  - CF型： 例如：Bigtable，Cassandra和Hbase。
  - 文档型：例如：MongoDB, coutchDB

现有方案(3)-NoSQL方案

- 现有No-SQL数据管理系统检索能力差
  - K/V型：仅支持基于Key的查询，无法做多关键字查询以及根据Value的复杂查询
  - Column-Based型：扩展了KV数据模型的表述能力，但是仅支持关键字查询，时间区间查询，不支持针对属性的复杂查询以及统计、分析等操作

现有方案(4)—Hadoop+MR+HIVE



- 面向非实时的分析型应用
- 速度慢，无法满足实时性的要求

现有方案分析

PDBMS、No-SQL数据库、Hadoop局限性分析！

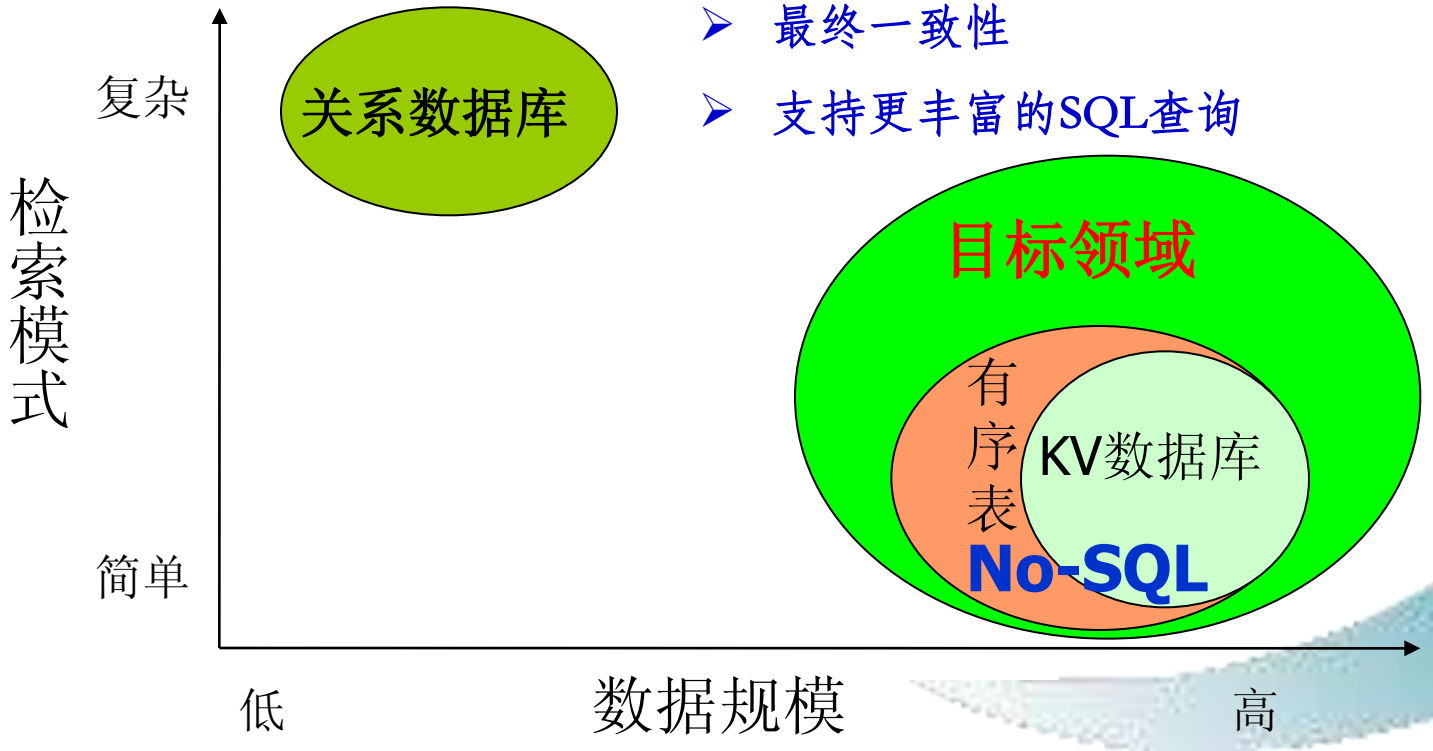
RDBMS	Hadoop+MR+Hive	No-SQL
当节点规模扩大时，由于关系模式的约束，子表 维护、数据错误等原因导致关系数据库的性能急剧下降！	MapReduce无索引的检索方式与“pull”模式的中间数据处理流程导致检索效率低下！	仅支持基于Row_Key的查询，不支持多列查询，统计分析等复杂查询；针对大返回结果集的查询效率低！

现有方案分析

系统分类		典型系统	特点概述
关系型数据库		DBMS-X，Verita，GreenPlum,AsterData	具备检索复杂性，但是不具有扩展性
Hadoop		HIVE，PIG，HadoopDB etc	具备扩展性，但是检索效率低
No-SQL	**Local Host-Key Value**	TC(KC),BDB	不具备扩展性
	Hash-based Key Value	Dynamo,Pnuts,voldemort,falre	具备扩展性，但是不支持区间查询
	Column-family	Hbase, Hypertable, Cassandra, Memcachedb，levelDB etc	具备扩展性，但是不支持多列查询
	Document based DB	MongoDB, coutchDB	加载、检索效率低

系统设计目标

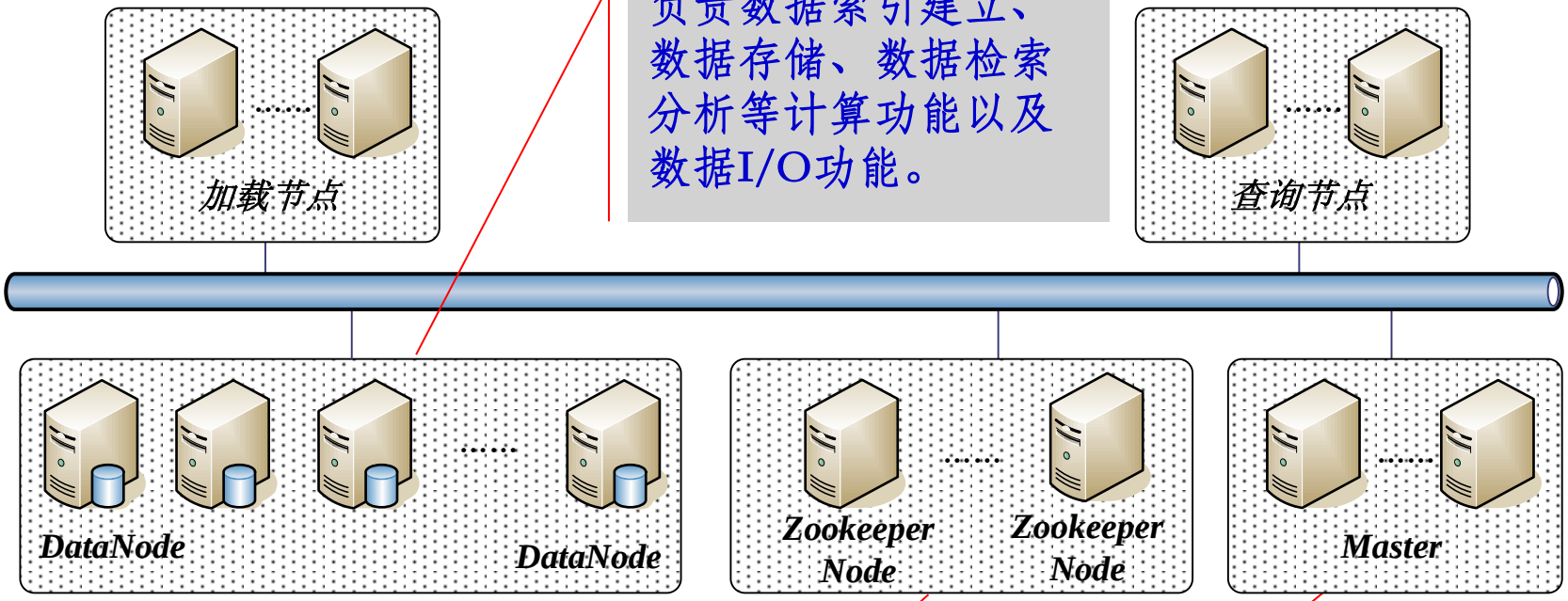
- 改善传统数据库的可扩展性差，并发性差的问题
- 解决NoSQL数据的检索能力差的问题，增加多列查询、统计排序等功能
  - 系统规模：万亿条
  - 最终一致性
  - 支持更丰富的SQL查询



题 纲

- 系统需求与现有方案
- 技术方案
- 应用案例

系统物理架构



负责数据索引建立、数据存储、数据检索分析等计算功能以及数据I/O功能。

负责对全局信息、节点状态信息的管理

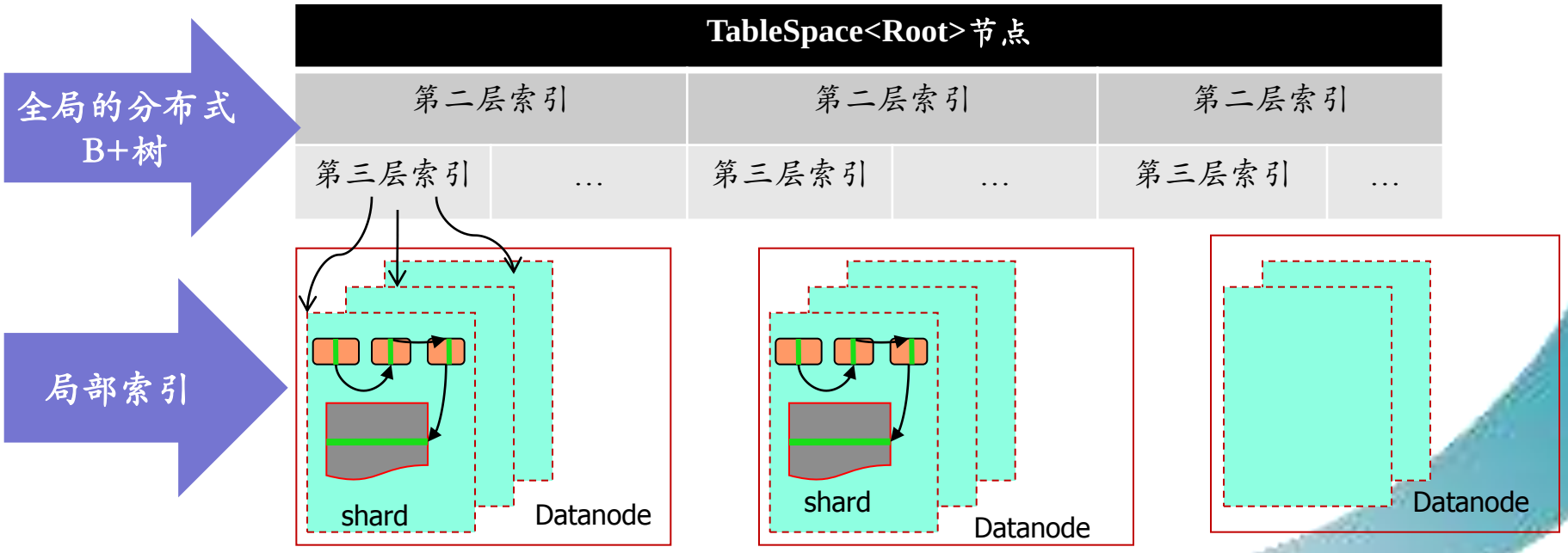
负责对集群中各节点进行管理和协调功能

索引结构

- 分布式Hash表
  - 随机读，不支持范围查询
  - 实例：Tair，Memcache，Dynamo，Cassandra
- 分布式B+ Tree
  - 随机读和顺序扫描，支持范围查询；
  - 顺序划分不均匀，需要叶子节点分裂合并
  - 实例：Bigtable & HBase，Google Megastore

索引结构

- 系统采用全局索引+局部索引机制
  - 全局索引：分布式B+树结构，支持对目标数据的快速定位
  - 局部索引：保存在每个node内部，支持复杂的查询和统计

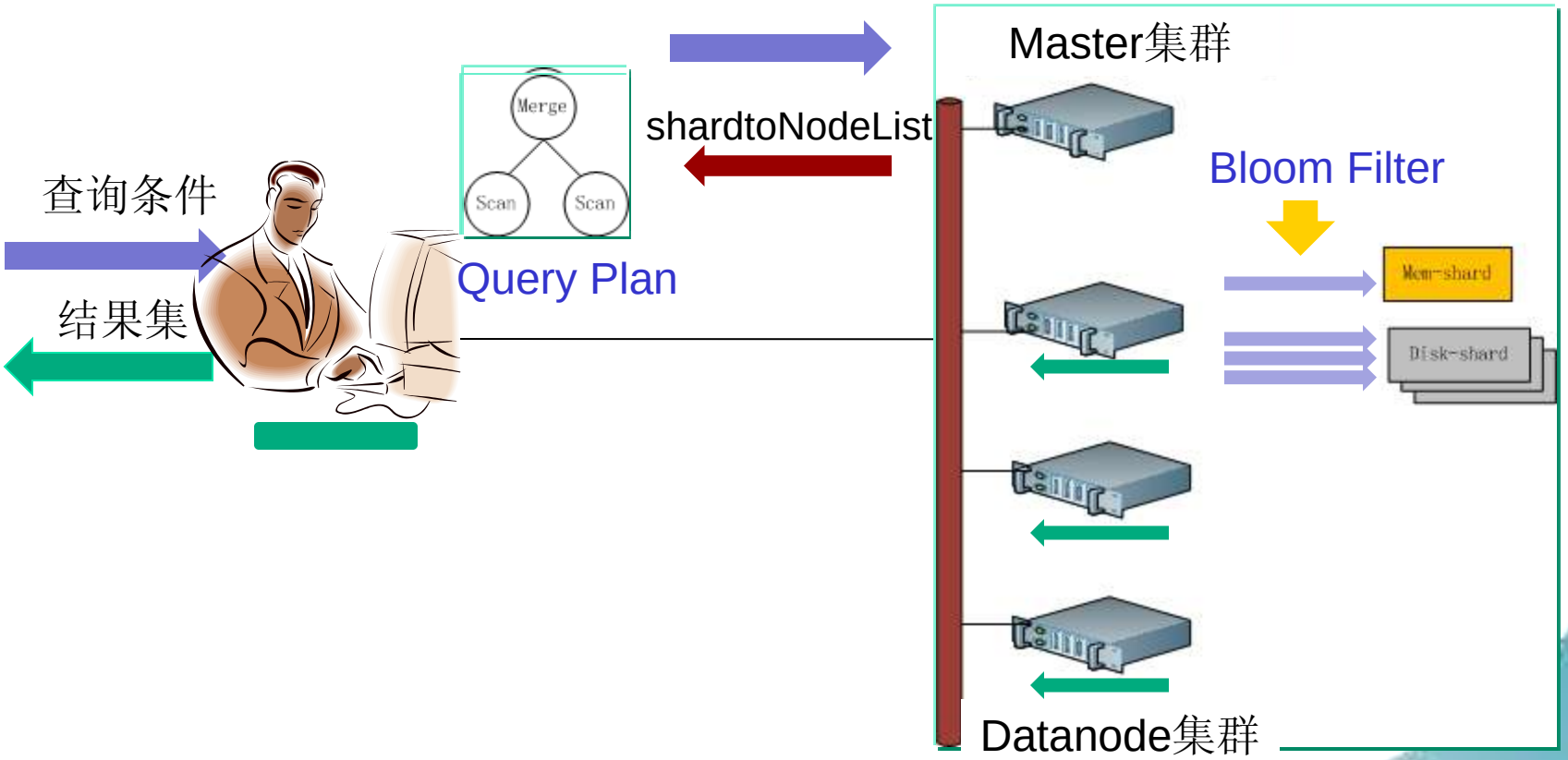


存储及I/O方式

- 列存储
  - 每个Shard由一个或者多个Store File组成，每个store File保存一个列
- I/O并行
  - 将数据分配到Data Node的多块磁盘上；
  - 数据I/O时并行从多块磁盘访问；
  - 实现多块磁盘IOPS和带宽的聚合。

分布式查询流程

- 分级、并行检索机制

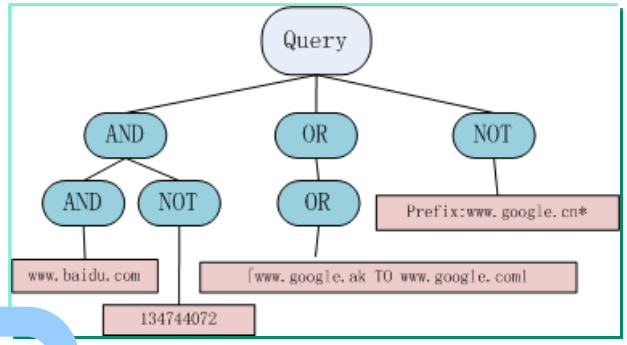


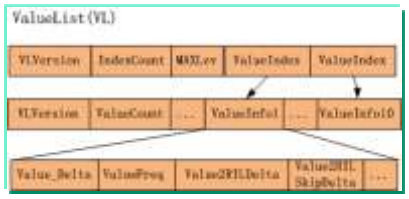
分布式查询流程

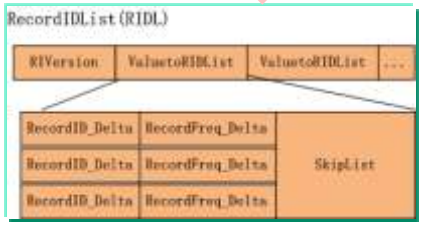
- Datanode独立接收查询规划树
- 根据语法树叶节点的属性值检索局部索引
- 根据语法树内节点合并、过滤规则，生成局部结果集

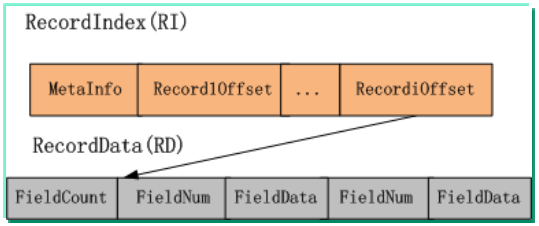
```
SELECT * FROM TABLE WHERE
(DOMAIN=‘www.baidu.com’ AND NOT VALUE
=‘8.8.8.8’)OR(DOMAIN = [‘www.google.ak’ TO
‘www.google.coml’)AND NOT DOMAIN=‘www.google.cn*’;
```

语法解析









结果集

分布式查询规划—查询规划

- 投影与选择：π，θ——shard
  - π :select DOMAIN,VALUE…
  - θ:where DOMAIN=‘www.baidu.com’ and TYPE=A…
  - 在Datanode上，针对每个shard并发处执行
- 分组与排序：Order By，Group BY——datanode+client
  - 在Datanode上，第二级汇总处执行
  - 在Client一级级处汇总多Datanode结果，做最后的排序、分组
- 聚合函数——client
  - 在Client第一级汇总处完成，汇总每个Datanode返回的结果集后，进行分组与排序，执行具体的聚合函数，再运行排序与topk处理

系统扩展性分析

- 存储规模分析
  - 系统采用share-nothing结构，增加存储节点，可以增加集群的计算能力并线性扩展存储空间
  - 读写完全并行化

系统扩展性分析

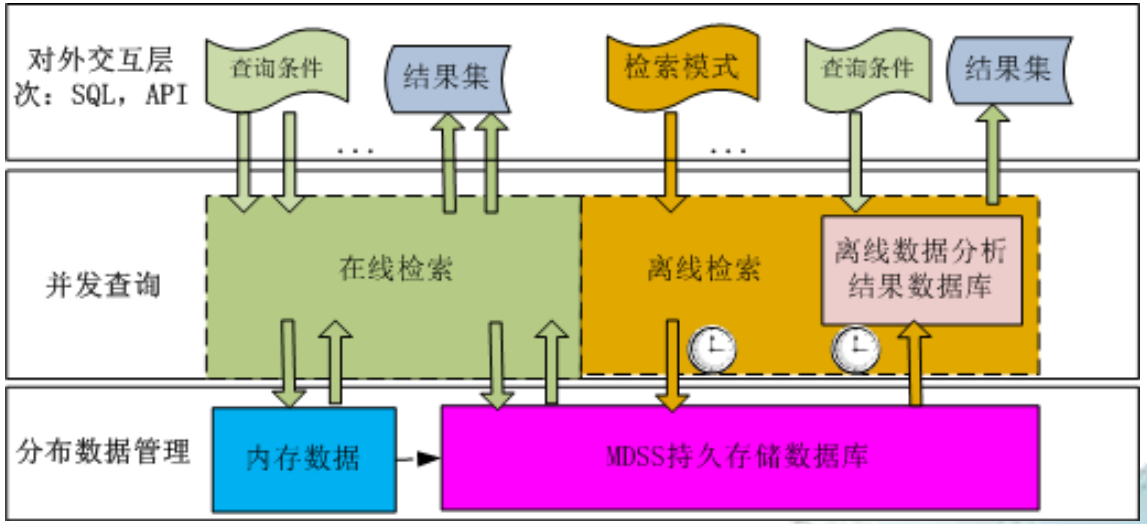
- 加载能力分析
  - 在所有Datanode上加载数据，系统加载效率可以线性提升
- 检索效率分析
  - 在二级并发处会线性提高检索效率
    - 数据规模一定，在仅有选择，投影检索条件时，检索效率会随着存储节点的增加而线性增加

系统可靠性分析

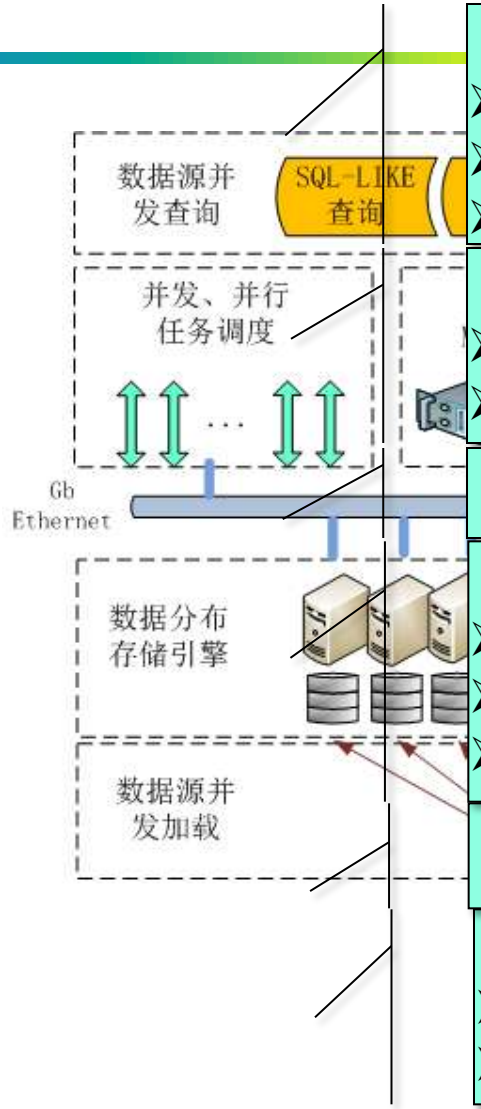
- 元数据管理集群化，防止单点失效问题
  - 元数据信息存储在zookeeper集群和DataNode集群上，具备数据容错功能，在线数据恢复，提供集群高可用性服务
  - Master点不存储元数据信息，可实现快速失效切换
- 引入commit-log，提供记录级别的原子操作
  - 防止内存数据的掉电丢失
  - 以记录为单位，提供前滚或回滚操作，保证记录写操作的原子性
- 引入副本容错机制
  - 支持多副本
  - 采用差量压缩技术，提高存储效率
- 无单点失效问题

系统功能概述—查询方法

- 对外提供在线检索和离线检索两类查询方法
  - 在线检索
    - 通过SQL-LIKE语言描述检索规则，利用SHELL或API接口返回结果
    - 可实时查询内存数据或历史数据
  - 离线检索
    - 用户定义查询任务，定时启动任务，结果导入离线数据分析库
    - 在离线数据分析库中查询获得相对复杂、耗时的数据查询



系统功能概述—功能汇总



（1）支持SQL语言的核心功能

- 支持多列查询、分组、排序、聚合等查询功能
- 提供必要的函数和数据类型定义
- 提供SHELL交互界面WebService接口

（2）支持分布式并发查询功能

- 支持二级并发机制：节点之间和shard之间
- 采用分布式查询规划

（3）支持在线查询与定时复杂任务查询

（4）全局索引+局部索引的分布索引式机制

- 全局索引采用分布式B+Tree
- 局部建立了面向多个属性的索引
- 数据以shard为单位进行列存储

（5）支持副本容错功能，无单点失效影响

（6）支持并发加载功能

- 采用带外传输机制写入数据到Datanode
- Datanode并发加载方式

题 纲

- 系统需求与现有方案
- 技术方案
- 应用案例

谢　谢