
AWS Summit
AWS技术峰会2015 · 上海
亚马逊
aws

Amazon Redshift深入解析

陈新

AWS解决方案架构师

议程

- Redshift简介
- 数据加载的最佳实践
- 查询语句优化的最佳实践
- 表结构设计的最佳实践
- 新特性介绍
- 应用迁移的注意事项
- AWS案例分享

Redshift是什么？

Amazon Redshift

性能优越
价格低廉
更加简便

PB级数据仓库

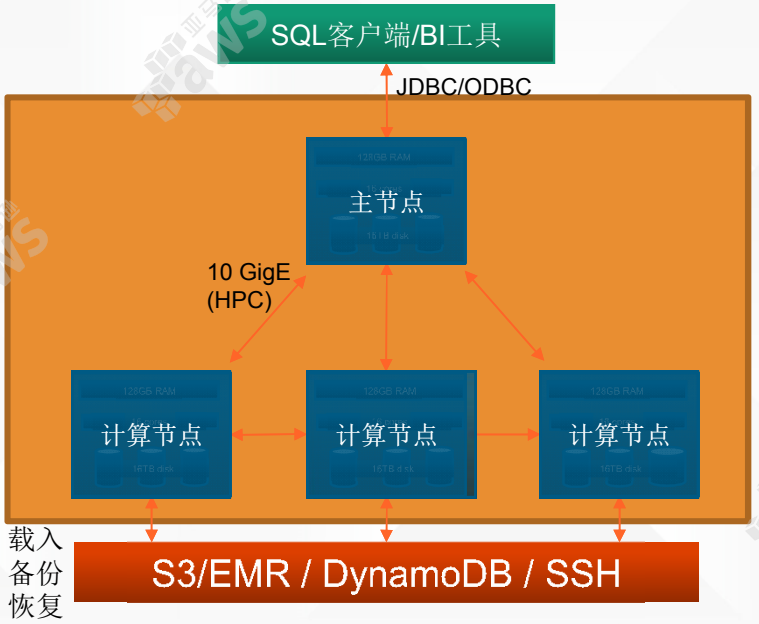
大规模并行处理（MPP）

关系型数据仓库（SQL）

管理简便、大幅扩容

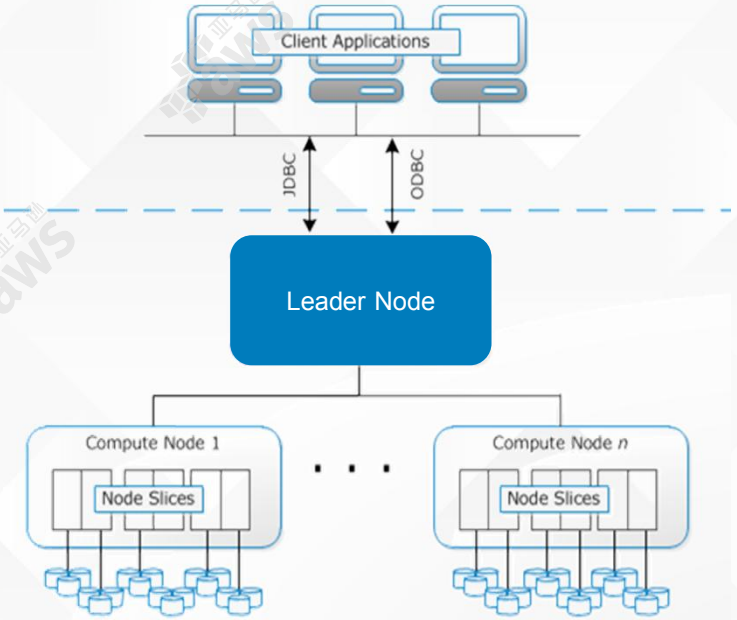
关于Redshift 的整体架构

- 主节点
  - SQL Endpoint
  - 元数据 (metadata)
  - 优化查询
- 计算节点
  - 列式存储
  - 并行查询
  - 可通过 S3 加载、备份和恢复数据
  - 可从 DynamoDB、EMR、SSH并行加载数据
- 三种型号
  - DS1（DW1）– HDD，容量从2TB 到 2PB
  - DS2 - HDD，容量从2TB 到 2PB
  - DC1（DW2）– SSD，容量从160GB 到 326TB



Redshift计算节点的架构剖析

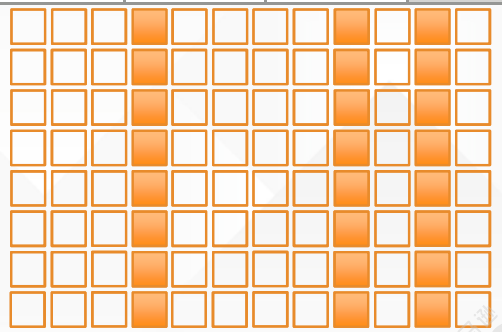
- 每个节点分为多个分片（slices）
  - 每个内核一个分片
  - DS1 – XL有2 个分片, 8XL 有16个分片
  - DS2 - XL 有4个分片， 8XL 有36个分片
  - DC1 – L有2个分片, 8XL 有32个分片
- 每个分片拥有独立的内存、CPU和磁盘空间
- 每个分片处理并行工作负载中的一部分任务



Redshift 为什么快？

- 列式存储
- 数据压缩
- Zone maps
- 直连的存储
- 大数据块

ID	Age	State	Amount
123	20	CA	500
345	25	WA	250
678	40	FL	125
957	37	WA	375

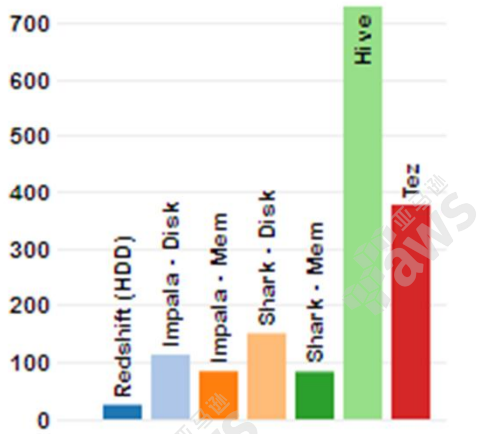


第三方测试报告验证Redshift的高性能

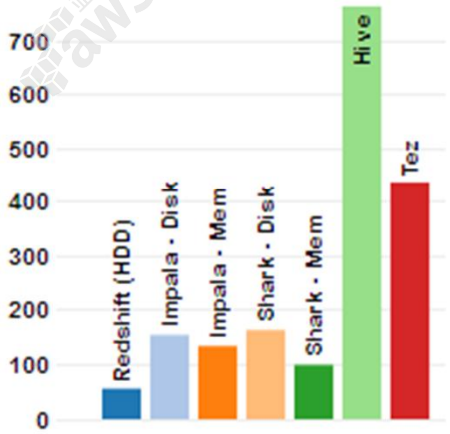
2. Aggregation Query

```
SELECT SUBSTR(sourceIP, 1, X), SUM(adRevenue) FROM uservisits GROUP BY SUBSTR(sourceIP, 1, X)
```

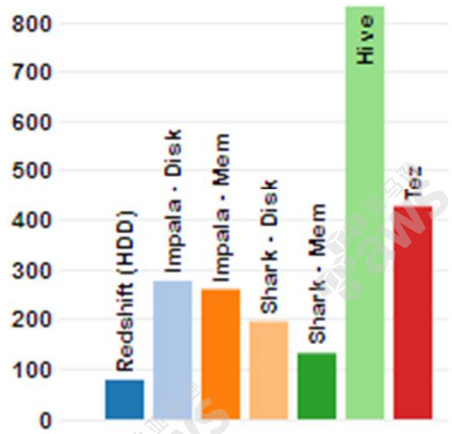
Query 2A
2,067,313 groups



Query 2B
31,348,913 groups



Query 2C
253,890,330 groups

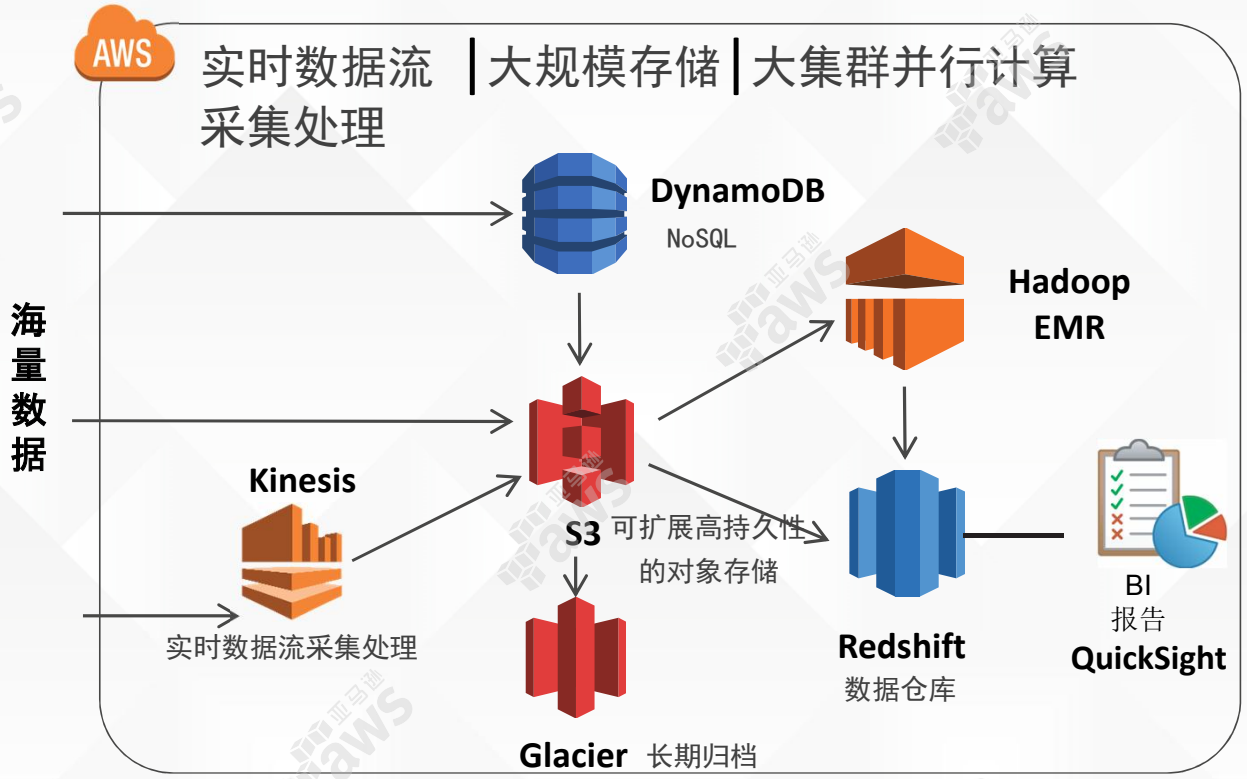


amplab UC BERKELEY

https://amplab.cs.berkeley.edu/benchmark/

aws 亚马逊

AWS 基于云的完整大数据服务



适合Redshift 的场景

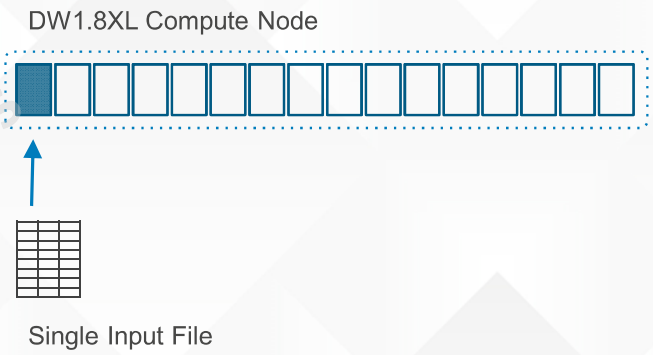
✓	在线分析处理 (OLAP) 大量数据的复杂查询
✗	在线交易处理 (OLTP) 数据量小的简单查询 数据库如RDS和DynamoDB 是更好的选择
✗	非结构化数据 数据可能需要进行预处理加载到Redshift (可以使用EMR 运行 MapReduce)

部分RedShift 客户

数据加载的最佳实践

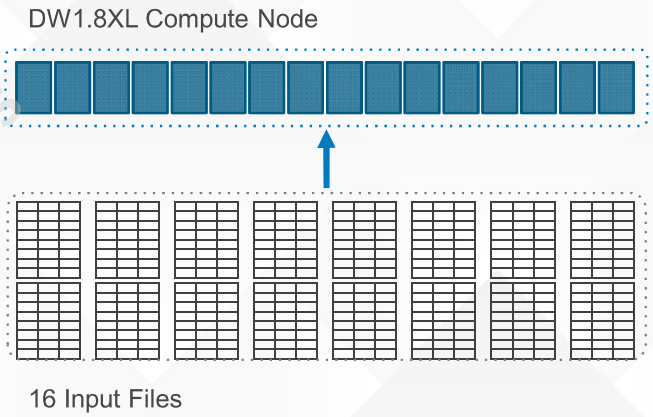
使用多个源文件来提高吞吐量

- 使用 COPY 命令
- 每个分片一次可以加载一个文件
- 只有一个源文件意味着只有一个分片可以处理输入的数据
- 你只能获得 6.25MB/s的性能，而不是100MB/s



使用多个源文件来提高吞吐量

- 使用 COPY 命令
- 你需要至少与集群中的分片数量相当的源文件
- 通过使用16个源文件,所有的分片都得到了利用，最大化吞吐量
- 每个节点可以获得100MB/s的性能;增加节点性能还可以得到线性的增长

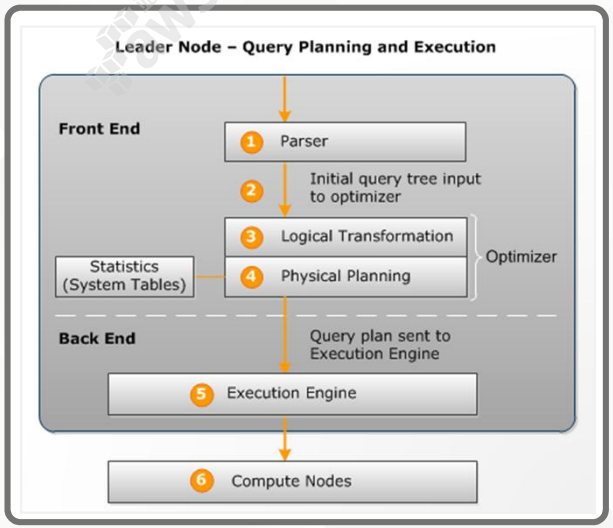


关于主键和manifest文件

- Redshift 不维护主键约束
  - 相同的数据如果加载多次， Redshift 不会报错
  - 如果你在DML语句中申明了主键约束， 优化器就会认为该列的数据是不重复的
- 使用manifest文件来精确控制加载哪些文件，以及如果源文件不存在时的处理逻辑
  - 定义一个保存在S3上的 JSON 格式的manifest 文件
  - 确保集群仅加载你想要加载的文件

在每次数据加载之后分析sort/dist key列

- Redshift的查询优化器依赖最新的统计信息
- 在每次数据加载之后更新Sort/Dist键列的统计信息来实现最佳的性能

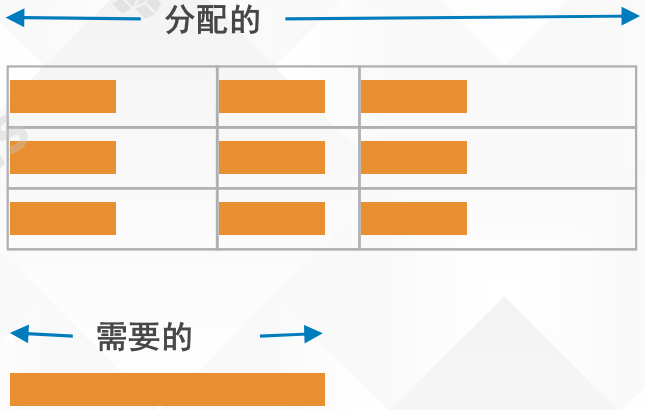


自动列压缩在大部分情况下适用

- 更好的性能，更低的成本
- 加载到空表时Redshift会自动拷贝样本数据
  - 样本数据多达 100,000 行，基于样本数据分析并自动选择每个列的最优化压缩算法
- 如果你需要定时执行ETL过程，并且用到了临时表或Staging表，建议关闭自动列压缩的功能
  - 使用analyze compression命令确定正确的列压缩参数
  - 将这些列压缩参数写到DML代码中

尽量减小列的宽度

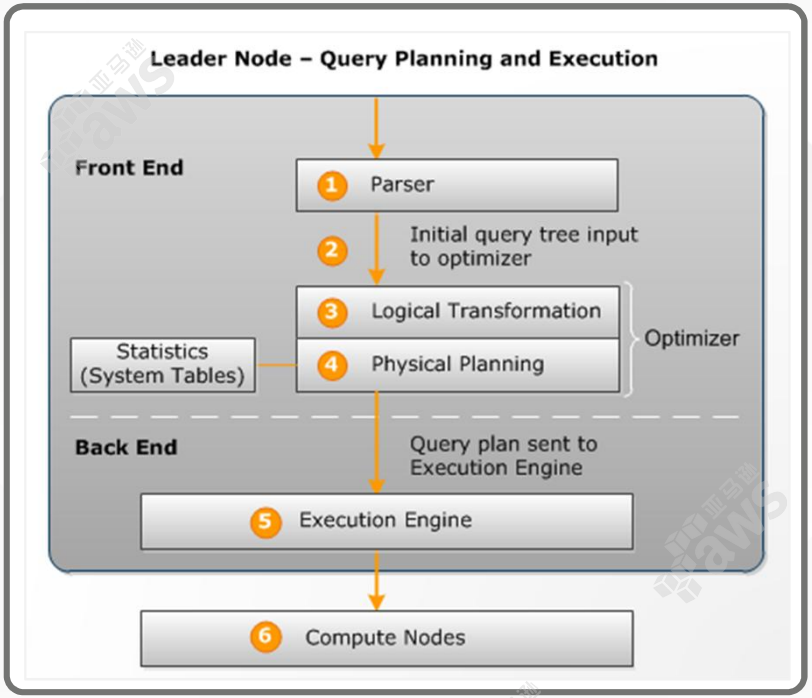
- 在查询和加载的过程中，系统会基于列的宽度分配缓存
- 比实际所需要的更大的列宽度意味着内存的浪费
- 内存中能保存的数据行就会更少；查询操作溢出到磁盘的概率就会增加。



关于数据的保健

- 定期对表做Analyze操作
  - 每次数据加载之后针对经常访问的列
  - 每周针对所有的列
  - 从SVV_TABLE_INFO(stats_off) 了解过时的状态信息
  - 从 stl_alert_event_log 了解缺失的状态信息

- 定期对表做Vacuum操作
  - 建议每周做一次
  - 根据unsorted 数据块的数量
  - 根据SVV_TABLE_INFO(unsorted, empty)
  - 如果unsorted数据占比例很高的话，Deep copy可能是更快速的方法
    (20% unsorted 通常采用deep copy更快速)



查询语句优化的最佳实践

高效的查询语句

- 避免使用 “select *”
- 使用CASE 表达式实现列的转换
- 避免交叉关联（成为“嵌套循环 ” 关联)
- 如果查询条件中用到了某张表，建议使用子查询
- 尽可能限制结果集的大小

高效的查询语句

- 选择成本最低的查询条件操作符 (尽量避免“LIKE”)
- 查询条件中避免使用函数
- 查询条件中使用第一个排序列 (最大的表)
- 在 group by 子句中使用排序键

表结构设计的最佳实践

排序键(Sort Key)的选择

- 经常查询最近的数据吗? 选择时间戳列
- 用到范围过滤吗？选择该列
- 用到相等过滤吗？选择该列
- 经常需要关联两张表吗? 选择关联列作为排序键和分布键

数据分布类型（Distribution style）

- 尽量避免数据的重新分布
- 选择事实表和某一个维度表（最大的）的关联列做分布键
- 某些维度 (例如小表) 可以使用 ALL
- 如果没有用到表关联的话可以使用EVEN 方式 (这是默认的方式)
- 小心某些有大量NULL值的列，可能会有数据分布不均

主键和外键

- 定义主键和外键约束 (Redshift不会维护该约束，但是优化器可以利用该约束)
- 由于Redshift不会维护这些约束，用户需要自己保证这些约束的有效性 (例如,主键列没有重复的数据)

使用基于时间的视图

- 范例: 按月分表结合UNION ALL 视图
- 好处:
  - 更好的方法移除旧的数据 (生命周期管理)
  - Vacuum操作可以在每张表上单独执行
- 需要注意的问题:
  - 视图与表管理会导致锁

新特性介绍

用户定义的函数（UDF）

- 您可以使用Python 2.7写UDF
- 语法基本上与PostgreSQL UDF 语法相同
  - 在UDF内禁止执行系统与网络调用

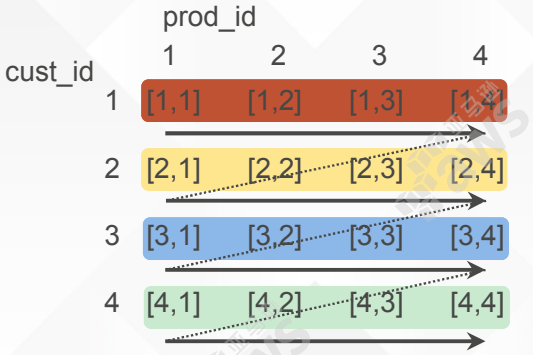
- 预先提供了 Pandas, NumPy, SciPy 库
  - 您还可以导入自己的库来提供更大的灵活性

UDF 示例– URL 解析

```
CREATE FUNCTION f_hostname (VARCHAR url)
  RETURNS varchar
IMMUTABLE AS $$
  import urlparse
  return urlparse.urlparse(url).hostname
$$ LANGUAGE plpythonu;
```

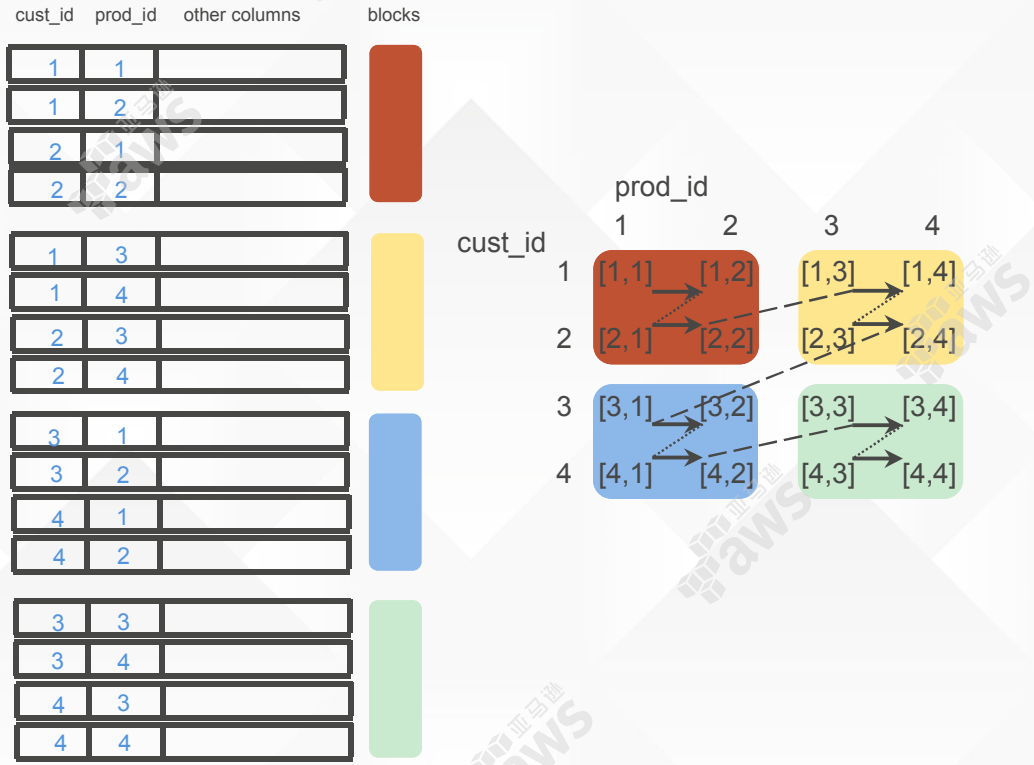
无需使用复杂的正则表达式，您可以导入标准的Python URL解析库并在SQL中使用

关于复合排序键



- Redshift中的记录保存在数据块中
- 假设每个块有4个记录
- 含有某个cust_id 的记录都存储在同一个块中
- 但是, 含有某个prod_id 的记录分布在四个块中

关于交叉排序键（Interleaved Sort Keys）



- 含有某个cust_id 的记录分布在两个块中
- 含有某个prod_id 的记录分布在两个块中
- 数据按两个key相同的措施排序

如何使用该特性

```
[[ COMPOUND | INTERLEAVED ] SORTKEY ( column_name [, ...] ) ]
```

- 新的关键字 ‘INTERLEAVED’ 用于定义排序键列
  - 现有的语法仍然有效，效果保持不变
  - 您可以选择多达8个列，并使用任意的列或者所有的列进行查询
- 查询语句保持不变

应用迁移的注意事项

传统的ETL/ELT

- 每张表一个源文件 ,大表可能有几个源文件
- 有许多的Update操作
- 每个作业都清空数据，然后再加载
- 依赖主键来防止重复的加载
- 高并发的加载作业
- 用小表来控制作业流

在Redshift上需要注意的地方

- Update操作通过delete + insert实现
  - Delete操作只是做一个删除标记
- Commit的成本很高
  - 8XL，每个节点上面要写4GB数据
  - 会镜像整个数据字典
  - 集群级别的串行操作

在Redshift上需要注意的地方

- 预先做汇总也会有帮助
- 保持低并发度
- 不要直接连接到Redshift运行仪表盘应用
- WLM 只是为Session之间分配内存, 而不是优先级调度
- 压缩也能带来速度的提升
- Distkey, Sortkey 和数据类型的选择很重要

开源的工具

- https://github.com/awslabs/amazon-redshift-utils

- **管理脚本**
  - 用于诊断集群的工具
- **管理视图**
  - 管理集群的工具，生成Schema DDL等
- **列压缩算法工具**
  - 帮助你针对已经加载了数据的Schema自动选择最优的列压缩算法
- **ANALYZE与Vacuum工具**
  - 帮助您自动化VACUUM 和ANALYZE操作
- **Unload 与Copy 工具**
  - 帮助您在Redshift集群或数据库之间迁移数据

常用的系统表和视图

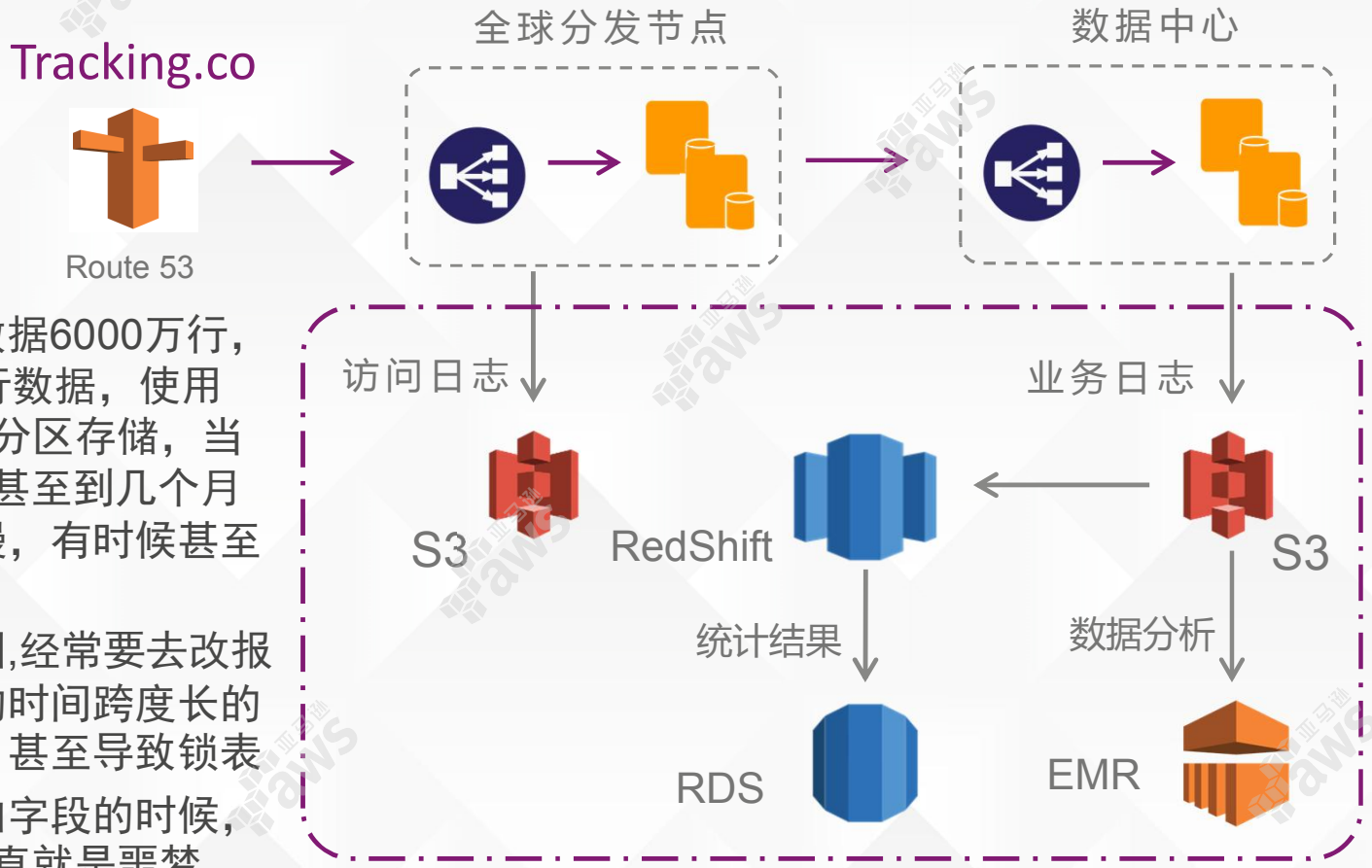
- SVV_TABLE_INFO
- STL_COMMIT_STATS
- STL_ALERT_EVENT_LOGS
- STL_WLM_QUERY

AWS案例分享

亚马逊 aws

Mobvista: 离线日记统计

Mobvista.



- 历史聚合报表数据6000万行，巅峰一天40万行数据，使用mysql按时间戳分区存储，当时间跨到1个月甚至到几个月的时候就会很慢，有时候甚至无法响应
- 基于业务的原因,经常要去改报表数据，当改的时间跨度长的时候，会很慢，甚至导致锁表
- 后期要对表增加字段的时候，对mysql来说简直就是噩梦

客户案例

Mobvista: Redshift使用效果

数据仓库

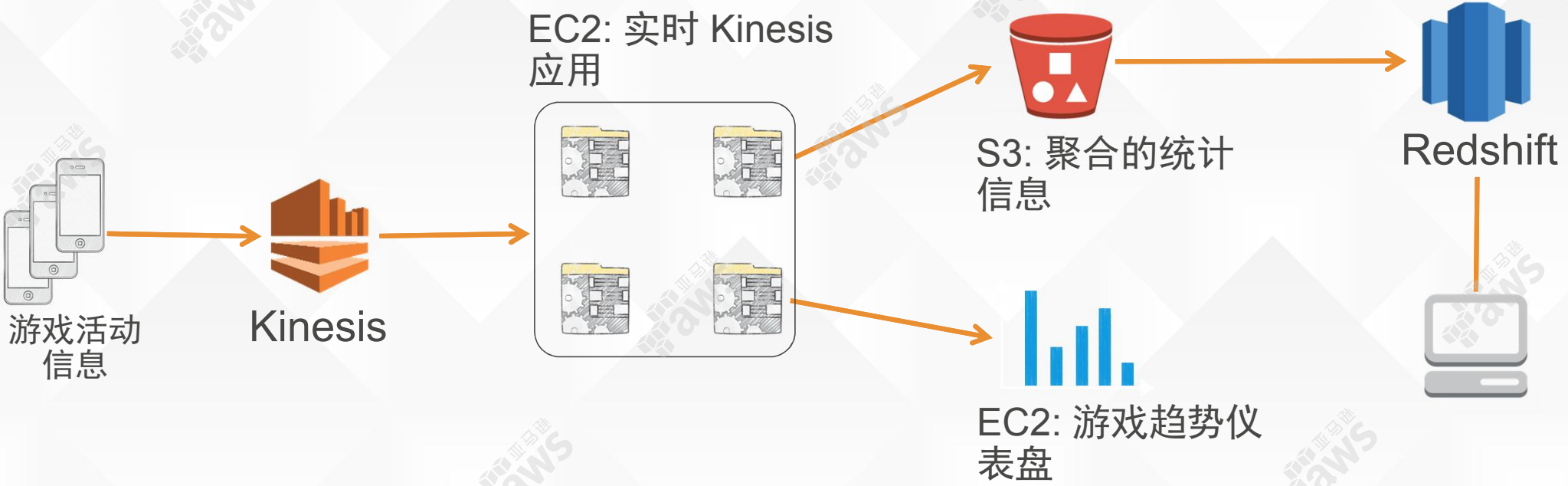
1. 列存储技术 ，区域压缩，亿级数据快速查询
2. 表自动分区；节点无缝升级扩展，无需停机
3. 可靠性强，自动备份
4. 兼容 PostgreSQL, ODBC,迁移成本低

- 8个dc1.large
- 一天数据新增大概为1亿行，占存储空间大概为10G，原始数据为20-30G
- 点击单表数据量已达60亿
- 非点击表的查询基本上都是秒级别响应
- 对点击表几天内数据的少维度聚合统计基本上都能在10秒能响应,多维度而复杂查询响应时间为30-60秒

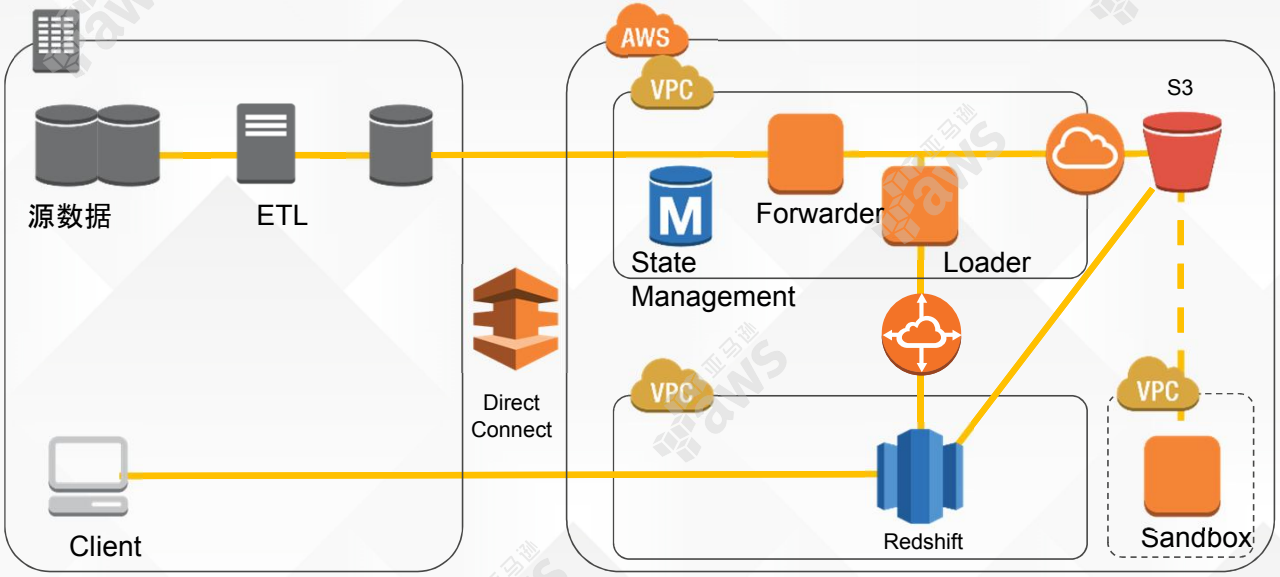
操作	描述（1.8亿条）	性能
Create table as	复制表	3分钟
update	更新3千万条	90秒
query	多维度聚合统计	5秒内
join	跟万级别表	几乎没影响

客户案例

SuperCell - Clash of Clans 游戏数据实时分析



NTT DOCOMO 企业数据仓库



- 在企业内部数据中心生成PB级别的数据，传递到云端的Redshift上进行分析
- 通过高速、冗余的专线连接到AWS
- 通过VPC、VPN、 CloudTrail、数据库审计、网络和存储加密等技术满足严格的安全要求

客户案例

Thank You