

Pivotal 践行见远技术篇系列

Pivotal®



Greenplum 精粹文集





迎接数据世界带来的挑战

此时此刻，我们正在经历着一场空前绝后的信息大爆炸。业界潮流也沿着开放、开源的方向走向了大数据和云计算时代。Pivotal，正是这场革命的领头羊。Greenplum 十来年的快速发展不是偶然发生的，与其在技术路线上始终与整个 IT 行业的技术演进保持高度一致密不可分。

这本精辑，汇集了 Pivotal 中国团队在大数据实践过程中对 Greenplum 的技术思考、实战教程、大咖日志和实践分享。可以说此书是专业性和实用性的完美融合。





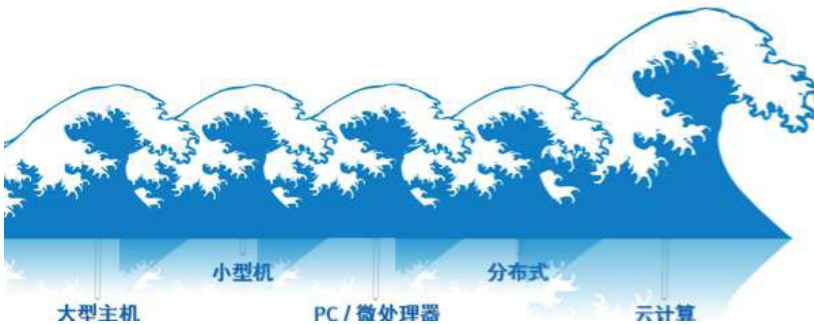
一、Greenplum 的前生今世	1
二、Greenplum 背后的帝国	20
三、Greenplum 硬件选型篇	24
四、Greenplum 实施经验谈	29
五、Greenplum 系统表的维护及修复技巧	34
六、Greenplum 的开发和优化	40
七、加密 Greenplum 中的静态数据	50

一、Greenplum 的前生今世

1. Greenplum 的起源

Greenplum 最早是在 10 多年前（大约在 2002 年）出现，基本上和 Hadoop 是同一时期（Hadoop 约是 2004 年前后出现的，早期的 Nutch 可追溯到 2002 年）。

互联网行业经过之前近 10 年的由慢到快的发展，累积了大量信息和数据，数据在爆发式增长，这些海量数据急需新的计算方式，需要一场计算方式的革命。



传统的主机计算模式在海量数据面前，除了造价昂贵外，在技术上也难于满足数据计算性能指标，传统主机的 Scale-up 模式遇到了瓶颈，SMP（对称多处理）架构难于扩展，并且在 CPU 计算和 IO 吞吐上不能满足海量数据的计算需求。

分布式存储和分布式计算理论刚刚被提出来，Google 的两篇著名论文发表后引起业界的关注，一篇是关于 GFS 分布式文件系统，另外一篇是关于 MapReduce 并行计算框架的理论，分布式计算模式在互联网行业特别是搜索引擎和分词检索等方面获得了巨大成功。

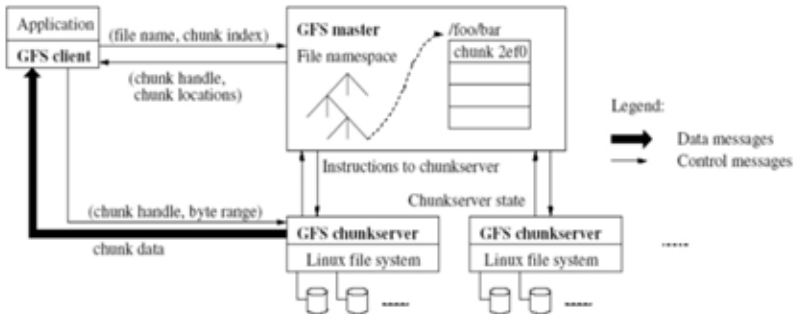


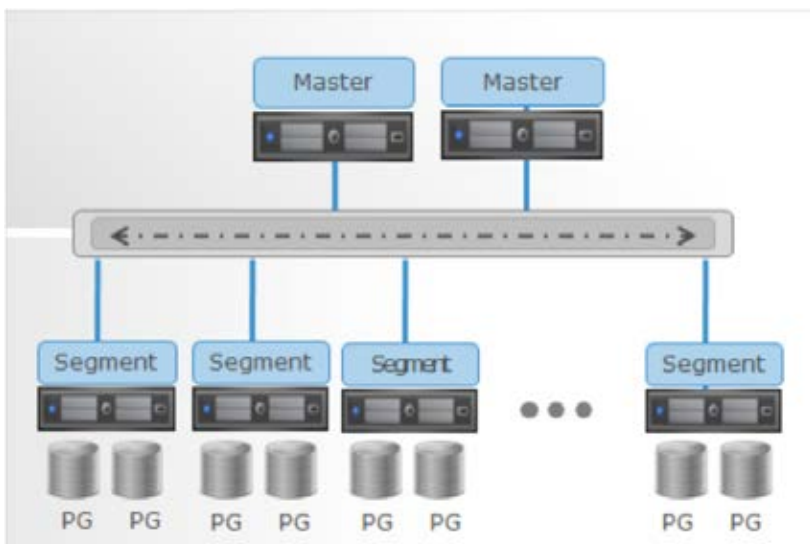
Figure 1: GFS Architecture

由此，业界认识到对于海量数据需要一种新的计算模式来支持，这种模式就是可以支持 Scale-out 横向扩展的分布式并行数据计算技术。

当时，开放的 X86 服务器技术已经能很好的支持商用，借助高速网络（当时是千兆以太网）组建的 X86 集群在整体上提供的计算能力已大幅高于传统 SMP 主机，并且成本很低，横向的扩展性还可带来系统良好的成长性。

问题来了，在 X86 集群上实现自动的并行计算，无论是后来的 MapReduce 计算框架还是 MPP（海量并行处理）计算框架，最终还是需要软件来实现，Greenplum 正是在这一背景下产生的，借助于分布式计算思想，Greenplum 实现了基于数据库的分布式数据存储和并行计算（GoogleMapReduce 实现的是基于文件的分布式数据存储和计算，我们会在后面比较这两种方法的优劣性）。

话说当年 Greenplum（当时还是一个 Startup 公司，创始人家门口有一棵青梅——greenplum，因此而得名）召集了十几位业界大咖（据说来自 google、yahoo、ibm 和 TD），说干就干，花了一年多的时间完成最初的版本设计和开发，用软件实现了在开放 X86 平台上的分布式并行计算，不依赖于任何专有硬件，达到的性能却远远超过传统高昂的专有系统。



Greenplum 集群架构

大家都知道 Greenplum 的数据库引擎层是基于著名的开源数据库 Postgresql 的(下面会分析为什么采用 Postgresql,而不是 mysql 等等),但是 Postgresql 是单实例数据库,怎么能在多个 X86 服务器上运行多个实例且实现并行计算呢?为了这,Interconnect 大神器出现了。在那一年多的时间里,大咖们很大一部分精力都在不断的设计、优化、开发 Interconnect 这个核心软件组件。最终实现了对同一个集群中多个 Postgresql 实例的高效协同和并行计算,Interconnect 承载了并行查询计划生产和 Dispatch 分发(QD)、协调节点上 QE 执行器的并行工作、负责数据分布、Pipeline 计算、镜像复制、健康探测等等诸多任务。

在 Greenplum 开源以前,据说一些厂商也有开发 MPP 数据库的打算,其中最难的部分就是在 Interconnect 上遇到了障碍,可见这项技术的关键性。

2. Greenplum 为什么选择 Postgresql 做轮子

说到这，也许有同学会问，为什么 Greenplum 要基于 Postgresql? 这个问题大致引申出两个问题：

1) 为什么不从数据库底层进行重新设计研发？

所谓术业有专攻，就像制造跑车的不会亲自生产车轮一样，我们只要专注在分布式技术中最核心的并行处理技术上面，协调我们下面的轮子跑的更快更稳才是我们的最终目标。而数据库底层组件就像车轮一样，经过几十年磨砺，数据库引擎技术已经非常成熟，大可不必去重新设计开发，而且把数据库底层交给其它专业化组织来开发（对应到 Postgresql 就是社区），还可充分利用到社区的源源不断的创新能力和资源，让产品保持持续旺盛的生命力。

这也是我们在用户选型时，通常建议用户考察一下底层的技术支撑是不是有好的组织和社区支持的原因，如果缺乏这方面的有力支持或独自闭门造轮，那就有理由为那个车的前途感到担忧，一个简单判断的标准就是看看底下那个轮子有多少人使用，有多少人为它贡献力量。

2) 为什么是 Postgresql 而不是其它的？

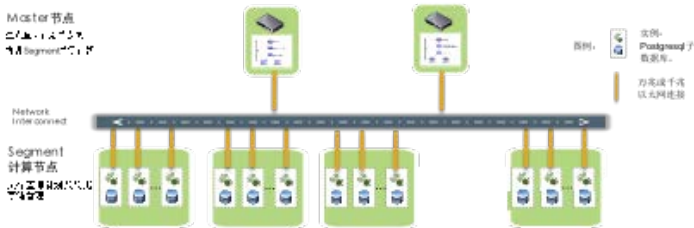
我想大家可能主要想问为什么是 Postgresql 而不是 Mysql?（其实，还有很多开源关系型数据库，但相比这两个主流开源库，实在不在一个起跑线上）。我们无意去从技术点上 PK 这两个数据库孰优孰劣，我相信它们的存在都有各自的特点，它们都有成熟的开源社区做支持，有各自的庞大的 fans 群众基础。我们认为，Greenplum 选择 Postgresql 有以下考虑：

Postgresql 号称最先进的数据库（官方主页 “The world’s most advanced open source database”），且不管这是不是自我标榜，就从 OLAP 分析型方面来考察，以下几点 Postgresql 确实胜出一筹。

- 1) PG 有非常强大 SQL 支持能力和非常丰富的统计函数和统计语法支持, 除对 ANSI SQL 完全支持外, 还支持比如分析函数(SQL2003 OLAP window 函数), 还可以用多种语言来写存储过程, 对于 Madlib、R 的支持也很好。这一点上 MYSQL 就差的很远, 很多分析功能都不支持, 而 Greenplum 作为 MPP 数据分析平台, 这些功能都是必不可少的。
- 2) Mysql 查询优化器对于子查询、复制查询如多表关联、外关联的支持等较弱, 特别是在关联时对于三大 join 技术: hash join、merge join、nestloop join 的支持方面, Mysql 只支持最后一种 nestloop join (据说未来会支持 hash join), 而多个大表关联分析时 hash join 是必备的利器, 缺少这些关键功能非常致命, 将难于在 OLAP 领域充当大任。我们最近对基于 MYSQL 的某内存分布式数据库做对比测试时, 发现其优点是 OLTP 非常快, TPS 非常高 (轻松搞定几十万), 但一到复杂多表关联性能就立马下降, 即使其具有内存计算的功能也无能为力, 就其因估计还是受到 mysql 在这方面限制。
- 3) 扩展性方面, Postgresql 比 mysql 也要出色许多, Postgres 天生就是为扩展而生的, 你可以在 PG 中用 Python、C、Perl、TCL、PLSQL 等等语言来扩展功能, 在后续章节中, 我将展现这种扩展是如何的方便, 另外, 开发新的功能模块、新的数据类型、新的索引类型等等非常方便, 只要按照 API 接口开发, 无需对 PG 重新编译。PG 中 contrib 目录下的各个第三方模块, 在 GP 中的 postgis 空间数据库、R、Madlib、pgcrypto 各类加密算法、gptext 全文检索都是通过这种方式实现功能扩展的。
- 4) 在诸如 ACID 事物处理、数据强一致性保证、数据类型支持、独特的 MVCC 带来高效数据更新能力等还有很多方面, Postgresql 似乎在这些 OLAP 功能上都比 mysql 更甚一筹。
- 5) Postgresql 许可是仿照 BSD 许可模式的, 没有被大公司控制, 社区比较纯洁, 版本和路线控制非常好, 基于 Postgresql 可让用户拥有更多自主性。反观 Mysql 的社区现状和众多分支 (如 MariaDB), 确实有些混乱。

相信这些特点已经足够了，据说很多互联网公司采用 Mysql 来做 OLTP 的同时，却采用 Postgresql 来做内部的 OLAP 分析数据库，甚至对新的 OLTP 系统也直接采用 Postgresql。

相比之下，Greenplum 更强悍，把 Postgresql 作为实例（该实例非 Oracle 实例概念，这里指的是一个分布式子库架构在 Interconnect 下），在 Interconnect 的指挥协调下，数十个甚至数千个 Sub Postgresql 数据库实例同时开展并行计算。而且，这些 Postgresql 之间采用 share-nothing 无共享架构，从而更将这种并行计算能力发挥到极致，除此之外，MPP 采用两阶段提交和全局事务管理机制来保证集群上分布式事务的一致性，Greenplum 像 Postgresql 一样满足关系型数据库的包括 ACID 在内的所有特征。



从上图可以看到，Greenplum 的最小并行单元不是节点层级，而是在实例层级。安装过 Greenplum 的同学应该都看到每个实例都有自己的 Postgresql 目录结构，都有各自的一套 Postgresql 数据库守护进程（甚至可以通过 UT 模式进行单个实例的访问）。正因为如此，甚至一个运行在单节点上的 GreenplumDB 也是一个小型的并行计算架构，一般一个节点配置 6~8 个实例，相当于在一个节点上有 6~8 个 Postgresql 数据库同时并行工作，优势在于可以充分利用到每个节点的所有 CPU 和 IO 能力。

Greenplum 单个节点上运行能力比其它数据库也快很多，如果运行在多节点上，其提供性能几乎是线性的增长，这样一个集群提供的性能能够很轻易的达到传统数据库的数百倍甚至数千倍，所管理数据存储空间达到 100TB~ 数 PB，而你在硬件上的投入，仅仅是数台一般的 X86 服务器和普通的万兆交换机。

Greenplum 采用 Postgres 作为底层引擎，良好的兼容了 Postgresql 的功能，Postgresql 中的功能模块和接口基本上 99% 都可以在 Greenplum 上使用，例如 odbc、jdbc、oledb、perltdbi、python psycopg2 等，所以 Greenplum 与第三方工具、BI 报表集成的时候非常容易；对于 postgresql 的 contrib 中的一些常用模块 Greenplum 提供了编译后的模块开箱即用，如：oraface、postgis、pgcrypt 等，对于其它模块，用户可以自行将 contrib 下的代码与 Greenplum 的 include 头文件编译后，将动态 so 库文件部署到所有节点就可进行测试使用了。有些模块还是非常好用的，例如：oraface，基本上集成了 Oracle 常用的函数到 Greenplum 中，曾经在一次 PoC 测试中，用户提供的 22 条 Oracle SQL 语句，不做任何改动就能运行在 Greenplum 上。

最后，需要强调的是：Greenplum 绝不仅仅只是简单的等同于“Postgresql+interconnect 并行调度 + 分布式事务两阶段提交”，Greenplum 还研发了非常多的高级数据分析管理功能和企业级管理模块，如下这些功能都是 Postgresql 没有提供的：

- 外部表并行数据加载
- 可更新数据压缩表
- 行、列混合存储
- 数据表多级分区
- Bitmap 索引
- Hadoop 外部表
- Gptext 全文检索
- 并行查询计划优化器和 Orca 优化器
- Primary/Mirror 镜像保护机制
- 资源队列管理
- WEB/Brower 监控

3. Greenplum 的艺术 -- Parallel Everything

前面介绍了 Greenplum 的分布式并行计算架构，其中每个节点上所有 Postgresql 实例都是并行工作的，这种并行的 Style 贯穿了 Greenplum 功能设计的方方面面：



外部表数据加载是并行的、查询计划执行是并行的、索引的建立和使用是并行的、统计信息收集是并行的、表关联（包括其中的重分布或广播及关联计算）是并行的，排序和分组聚合都是并行的，备份恢复也是并行的，甚而数据库启停和元数据检查等维护工具也按照并行方式来设计。得益于这种无所不在的并行，Greenplum 在数据加载和数据计算中表现出强悍的性能，某行业客户对此深有体会：同样 2TB 左右的数据，在 Greenplum 中不到一个小时就加载完成了，而在用户传统数据仓库平台上耗时半天以上。

在该用户的生产环境中，1 个数百亿表和 2 个 10 多亿条记录表的全表关联中（只有 on 关联条件，不带 where 过滤条件，其中一个 10 亿条的表计算中需要重分布），Greenplum 仅耗时数分钟就完成了，当其它传统数据平台还在为千万级或亿级规模的表关联性能发愁时，Greenplum 已经一骑绝尘，在百亿级规模以上表关联中展示出上佳的表现。

Greenplum 建立在 Share-nothing 无共享架构上，让每一颗 CPU 和每一块磁盘 IO 都运转起来，无共享架构将这种并行处理发挥到极致。相比一些其它传统数据仓库的 Sharedisk 架构，后者最大瓶颈就是在 IO 吞吐上，在大规模数据处理时，IO 无法及时 feed 数据给到 CPU，CPU 资源处于 wait 空转状态，无法充分利用系统资源，导致 SQL 效率低下：

一台内置 16 块 SAS 盘的 X86 服务器，每秒的 IO 数据扫描性能约在 2000MB/s 左右，可以想象，20 台这样的服务器构成的机群 IO 性能是 40GB/s，这样超大的 IO 吞吐是传统的 Storage 难以达到的。



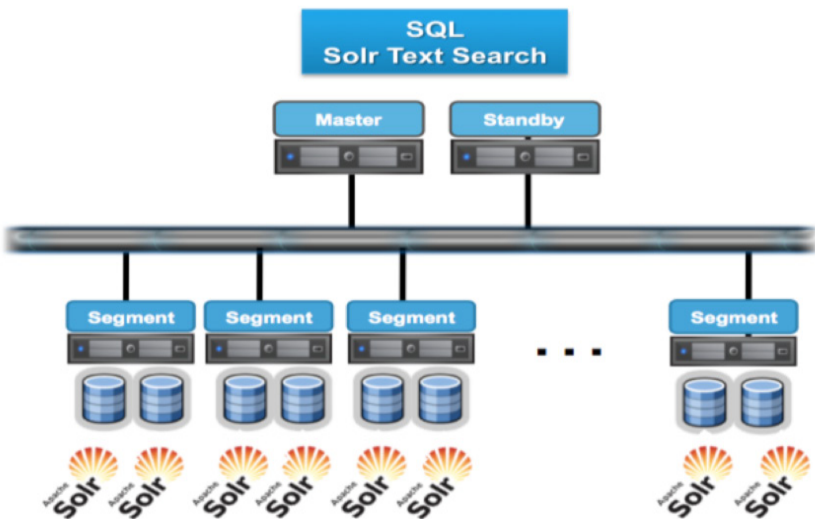
(MPP Share-nothing 架构实现超大 IO 吞吐能力)

另外，Greenplum 还是建立在实例级别上的并行计算，可在一次 SQL 请求中利用到每个节点上的多个 CPU CORE 的计算能力，对 X86 的 CPU 超线程有很好的支持，提供更好的请求响应速度。在 PoC 中接触到其它一些国内外基于开放平台的 MPP 软件，大都是建立在节点级的并行，单个或少量的任务时无法充分利用资源，导致系统加载和 SQL 执行性能不高。

记忆较深的一次 PoC 公开测试中，有厂商要求在测试中关闭 CPU 超线程，估计和这个原因有关（因为没有办法利用到多个 CPU core 的计算能力，还不如关掉超线程以提高单 core 的能力），但即使是这样，在那个测试中，测试性能也大幅低于 Greenplum（那个测试中，各厂商基于客户提供的完全相同的硬件环境，Greenplum 是唯一一家完成所有测试的，特别在混合负载测试中，Greenplum 的 80 并发耗时 3 个多小时就成功完成了，其它厂商大都没有完成此项测试，唯一完成的一家耗时 40 多小时）。

前文提到，得益于 Postgresql 的良好扩展性（这里是 extension，不是 scalability），Greenplum 可以采用各种开发语言来扩展用户自定义函数（UDF）（我个人是 Python 和 C 的 fans，后续章节与大家分享）。这些自定义函数部署到 Greenplum 后可用充分享受到实例级别的并行性能优势，我们强烈建议用户将库外的处理逻辑，部署到用 MPP 数据库的 UDF 这种 In-Database 的方式来处理，你将获得意想不到的性能和方便性；例如我们在某客户实现的数据转码、数据脱敏等，只需要简单的改写原有代码后部署到 GP 中，通过并行计算获得数十倍性能提高。

另外，GPTXT（lucent 全文检索）、Apache Madlib（开源挖掘算法）、SAS algorithm、R 都是通过 UDF 方式实现在 Greenplum 集群中分布式部署，从而获得库内计算的并行能力。这里可以分享的是，SAS 曾经做过测试，对 1 亿条记录做逻辑回归，采用一台小型机耗时约 4 个多小时，通过部署到 Greenplum 集群中，耗时不到 2 分钟就全部完成了。以 GPEXT 为例，下图展现了 Solr 全文检索在 Greenplum 中的并行化风格。



最后，也许你会有问题，Greenplum 采用 Master-slave 架构，Master 是否会成为瓶颈？完全不用担心，Greenplum 所有的并行任务都是在 Segment 数据节点上完成后，Master 只负责生成和优化查询计划、派发任务、协调数据节点进行并行计算。

按照我们在用户现场观察到的，Master 上的资源消耗很少有超过 20% 情况发生，因为 Segment 才是计算和加载发生的场所（当然，在 HA 方面，Greenplum 提供 Standby Master 机制进行保证）。

再进一步看，Master-Slave 架构在业界的大数据分布式计算和云计算体系中被广泛应用，大家可以看到，现在主流分布式系统都是采用 Master-Slave 架构，包括：Hadoop FS、Hbase、MapReduce、Storm、Mesos..... 无一例外都是 Master-Slave 架构。相反，采用 MultipleActive Master 的软件系统，需要消耗更多资源和机制来保证元数据一致性和全局事务一致性，特别是在节点规模较多时，将导致性能下降，严重时可能导致多 Master 之间的脑裂引发严重系统故障。

4. Greenplum 不能做什么？

Greenplum 最大的特点总结就一句话：基于低成本的开放平台基础上提供强大的并行数据计算性能和海量数据管理能力。这个能力主要指的是并行计算能力，是对大任务、复杂任务的快速高效计算，但如果你指望 MPP 并行数据库能够像 OLTP 数据库一样，在极短的时间处理大量的并发小任务，这个并非 MPP 数据库所长。请牢记，并行和并发是两个完全不同的概念，MPP 数据库是为了解决大问题而设计的并行计算技术，而不是大量的小问题的高并发请求。

再通俗点说，Greenplum 主要定位在 OLAP 领域，利用 Greenplum MPP 数据库做大数据计算或分析平台非常适合，例如：数据仓库系统、ODS 系统、ACRM 系统、历史数据管理系统、电信流量分析系统、移动通信令分析系统、SANDBOX 自助分析沙箱、数据集市等等。

而 MPP 数据库都不擅长做 OLTP 交易系统，所谓交易系统，就是高频的交易型小规模数据插入、修改、删除，每次事务处理的数据量不大，但每秒钟都会发生几十次甚至几百次以上交易型事务，这类系统的衡量指标是 TPS，适用的系统是 OLTP 数据库或类似 GemFire 的内存数据库。

5. Greenplum MPP 与 Hadoop

MPP 和 Hadoop 都是为了解决大规模数据的并行计算而出现的技术，两种技术的相似点在于：

-
- 分布式存储数据在多个节点服务器上
 - 采用分布式并行计算框架
 - 支持横向扩展来提高整体的计算能力和存储容量
 - 都支持 X86 开放集群架构
-

但两种技术在数据存储和计算方法上，也存在很多显而易见的差异：

-
- MPP 按照关系数据库行列表方式存储数据（有模式），Hadoop 按照文件切片方式分布式存储（无模式）。
 - 两者采用的数据分布机制不同，MPP 采用 Hash 分布，计算节点和存储紧密耦合，数据分布粒度在记录级的更小粒度（一般在 1k 以下）；Hadoop FS 按照文件切块后随机分配，节点和数据无耦合，数据分布粒度在文件块级（缺省 64MB）。
 - MPP 采用 SQL 并行查询计划，Hadoop 采用 Mapreduce 框架。
-

基于以上不同，体现在效率、功能等特性方面也大不相同。

1) 计算效率的比较

先说说 Mapreduce 技术。

Mapreduce 相比而言是一种较为蛮力计算方式（业内曾经甚至有声音质疑 MapReduce 是反潮流的），数据处理过程分成 Map-〉 Shuffle-〉 Reduce 的过程，相比 MPP 数据库并行计算而言，Mapreduce 的数据在计算前未经整理和组织（只是做了简单数据分块，数据无模式），而 MPP 预先会把数据有效的组织（有模式），例如：行列表关系、Hash 分布、索引、分区、列存储等、统计信息收集等，这就决定了在计算过程中效率大为不同：

• MAP 效率对比

Hadoop 的 MAP 阶段需要对数据再解析，而 MPP 数据库则会直接取行列表，效率高。

Hadoop 按 64MB 拆分文件，而且数据不能保证在所有节点都均匀分布，因此，MAP 过程的并行化程度低；MPP 数据库按照数据记录拆分和 Hash 分布，粒度更细，数据分布在所有节点中非常均匀，并行化程度很高。

HadoopHDFS 没有灵活的索引、分区、列存储等技术支持，而 MPP 通常利用这些技术大幅提高数据的检索效率。

• MASHuffle 效率对比

Hadoop Shuffle 对比 MPP 计算中的重分布 -- 由于 Hadoop 数据与节点的无关性，Shuffle 是基本避免不了的；而 MPP 数据库对于相同 Hash 分布数据不需要重分布，节省大量网络和 CPU 消耗。

Mapreduce 没有统计信息，不能做基于 cost-base 的优化；MPP 数据库可以利用统计信息很好地进行并行计算优化。例如，MPP 对于不同分布的数据可以在计算中基于 Cost 动态决定最优执行路径，如采用重分布还是小表广播。

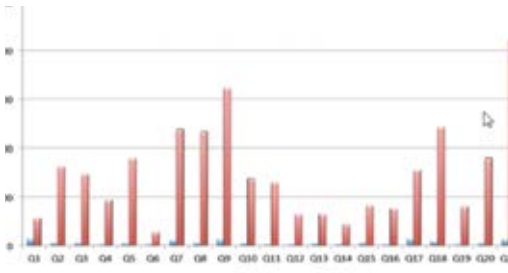
- Reduce 效率对比：

对比于 MPP 数据库的 SQL 执行器 -executor，Mapreduce 缺乏灵活的 Join 技术支持；MPP 数据库可以基于 COST 来自动选择 Hash join、Merger join 和 Nestloop join，甚至可以在 Hash join 通过 COST 选择小表做 Hash，在 NestloopJoin 中选择 index 提高 join 性能等等。

MPP 数据库对于 Aggregation（聚合）提供 Multiple-agg、Group-agg、sort-agg 等多种技术来提供计算性能；Mapreuce 需要开发人员自己实现。

另外，Mapreduce 在整个 MAP->Shuffle->Reduce 过程中通过文件来交换数据，效率很低，MapReduce 要求每个步骤间的数据都要序列化到磁盘，这意味着 MapReduce 作业的 I/O 成本很高，导致交互分析和迭代算法开销很大，MPP 数据库采用 Pipeline 方式在内存数据流中处理数据，效率比文件方式高很多。

总结以上几点，MPP 数据库在计算并行度、计算算法上比 Hadoop 更加 SMART，效率更高；在客户现场的测试对比中，Mapreduce 对于单表的计算尚可，但对于复杂查询，如多表关联等，性能很差，性能甚至只有 MPP 数据库的几分之一甚至几百分之一，下图是基于 MapReduce 的 Hive 和 Greenplum MPP 在 TPCH 22 个 SQL 测试性能比较：（相同硬件环境下）



某国内知名电商在其数据分析平台上做过验证：同样的硬件条件下，MPP 数据库比 Hadoop 性能快 12 倍以上。

2) 功能上的对比

MPP 数据库采用 SQL 作为主要交互式语言，SQL 语言简单易学，具有很强数据操纵能力和过程语言的流程控制能力，SQL 语言是专门为统计和数据分析开发的语言，各种功能和函数琳琅满目，SQL 语言不仅适合开发人员，也适用于分析业务人员，大幅简化了数据的操作和交互过程。

而对 MapReduce 编程明显是困难的，在原生的 Mapreduce 开发框架基础上的开发，需要技术人员谙熟于 JAVA 开发和并行原理，不仅业务分析人员无法使用，甚至技术人员也难以学习和操控。为了解决易用性的问题，近年来 SQL-ON-HADOOP 技术大量涌现出来，几乎成为当前 Hadoop 开发使用的一个技术热点趋势。

这些技术包括：Hive、Pivotal HAWQ、SPARK SQL、Impala、Prest、Drill、Tajo 等等很多，这些技术有些是在 Mapreduce 上做了优化。例如 Spark 采用内存中的 Mapreduce 技术，号称性能比基于文件的 Mapreduce 提高 10 倍；有的则采用 C/C++ 语言替代 Java 语言重构 Hadoop 和 Mapreduce（如 MapR 公司及国内某知名电商的大数据平台）；而有些则直接绕开了 Mapreduce 另起炉灶，如 Impala、hawq 采用借鉴 MPP 计算思想来做查询优化和内存数据 Pipeline 计算，以此来提高性能。

虽然 SQL-On-Hadoop 比原始的 Mapreduce 虽然在易用上有所提高，但在 SQL 成熟度和关系分析上目前还与 MPP 数据库有较大差距。

上述系统，除了 HAWQ 外，对 SQL 的支持都非常有限，特别是分析型复杂 SQL，如 SQL 2003 OLAPWINDOW 函数，几乎都不支持。以 TPC-DS 测试（用于评测决策支持系统（大数据或数据仓库）的标准 SQL 测试集，99 个 SQL）为例，包括 SPARK、Impala、Hive，只能支持其中的 1/3 左右。

由于 HADOOP 本身 Append-only 的特性，SQL-On-Hadoop 大多不支持数据局部更新和删除功能 (update/delete)；例如 Spark 计算时，需要预先将数据装载到 DataFrames 模型中；

基本上都缺少索引和存储过程等特征

除 HAWQ 外，大多对于 ODBC/JDBC/DBI/OLEDB/.NET 接口的支持有限，与主流第三方 BI 报表工具的兼容性不如 MPP 数据库

SQL-On-Hadoop 不擅长于交互式 (interactive) 的 Ad-hoc 查询，大多通过预关联的方式来规避这个问题；另外，在并发处理方面的能力较弱。高并发场景下，需要控制计算请求的并发度，避免资源过载导致的稳定性问题和性能下降问题。

3) 架构灵活性的对比

前面提到，为保证数据的高性能计算，MPP 数据库节点和数据之间是紧耦合的，相反，Hadoop 的节点和数据是没有耦合关系的。这就决定了 Hadoop 的架构更加灵活，存储节点和计算节点的无关性，这体现在以下 2 个方面：

- 扩展性方面

Hadoop 架构支持单独增加数据节点或计算节点，依托于 Hadoop 的 SQL-On-Hadoop 系统，例如 HAWQ、SPARK 均可单独增加计算层的节点或数据层的 HDFS 存储节点，HDFS 数据存储对计算层来说是透明的；

MPP 数据库扩展时，一般情况下是计算节点和数据节点一起增加的，在增加节点后，需要对数据做重分布才能保证数据与节点的紧耦合（重新 Hash 数据），进而保证系统的性能；Hadoop 在增加存储层节点后，虽然也需要 Rebalance 数据，但相较 MPP 而言，不是那么紧迫。

- 节点退服方面

Hadoop 节点宕机退服对系统的影响较小，并且系统会自动将数据在其它节点扩充到 3 份；MPP 数据库节点宕机时，系统的性能损耗大于 Hadoop 节点。Pivotal 将 GPDB 的 MPP 技术与 Hadoop 分布式存储技术结合，推出了 HAWQ 高级数据分析软件系统，实现了 Hadoop 上的 SQL-On-Hadoop。

与其它的 SQL-on-HADOOP 系统不同，HAWQ 支持完全的 SQL 语法 和 SQL2003 OLAP 语法及 Cost-Base 的算法优化，让用户就像使用关系型数据库一样使用 Hadoop。底层存储采用 HDFS，HAWQ 实现了计算节点和 HDFS 数据节点的解耦，采用 MR2.0 的 YARN 来进行资源调度，同时具有 Hadoop 的灵活伸缩的架构特性和 MPP 的高效能计算能力。

当然，有得也有所失，虽然 HAWQ 的架构比 GreenplumMPP 数据库灵活，但在获得架构优越性的同时，其性能比 Greenplum MPP 数据库要低一倍左右。不过，得益于 MPP 算法的红利，HAWQ 的性能仍大幅高于其它基于 MapReduce 的 SQL-On-Hadoop 系统。

4) 选择 MPP 还是 Hadoop ?

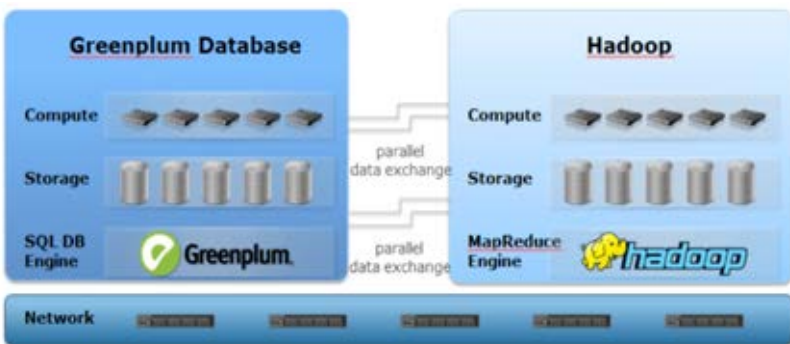
HadoopMapReduce 和 SQL-On-Hadoop 技术目前都还不够成熟，性能和功能上都有很多待提升的空间。相比之下，MPP 数据在数据处理上更加 SMART，要填平或缩小与 MPP 数据库之间的性能和功能上的差距，Hadoop 还有很长的一段路要走。就目前来看，我们认为这两个系统都有其适用的场景。

简单来说，如果你的数据需要频繁的被计算和统计、并且你希望具有更好的 SQL 交互式支持和更快计算性能及复杂 SQL 语法的支持，那么你应该选择 MPP 数据库，SQL-on-Hadoop 技术还没有足够成熟。特别如数据仓库、集市、ODS、交互式分析数据平台等系统，MPP 数据库有明显的优势。

而如果你的数据加载后只会被用于读取少数次的任务和用于少数次的访问，而且主要用于 Batch（不需要交互式），对计算性能不是很敏感，那 Hadoop 也是不错的选择，因为 Hadoop 不需要你花费较多的精力来模式化你的数据，节省数据模型设计和数据加载设计方面的投入。这些系统包括：历史数据系统、ETL 临时数据区、数据交换平台等等。

切记，千万不要为了大数据而大数据（就好像不要为了创新而创新一个道理），否则，你项目最后的产出与你的最初设想可能将差之千里，行业内不乏失败案例。

最后，提一下，GreenplumMPP 数据库支持用“Hadoop 外部表”方式来访问、加载 HadoopFS 的数据，虽然 Greenplum 的 Hadoop 外部表性能大幅低于 MPP 内部表，但比 Hadoop 自身的 HIVE 要高很多（在某金融客户的测试结果，比 HIVE 高 8 倍左右），因此可以考虑在项目中同时部署 MPP 数据库和 Hadoop，MPP 用于交互式高性能分析，Hadoop 用于数据 Staging、MPP 的数据备份或一些 ETL batch 的数据清洗任务，两者相辅相成，在各自最擅长的场景中发挥其特性和优势。



6. Greenplum 未来的发展之路

过去十年，IT 技术潮起潮落发生着时刻不停的变化，而在这变化中的不变就是走向开放和开源的道路，即将到来的伟大变革是云计算时代的到来，任何 IT 技术，从硬件到软件到服务，都逃不过要接受云计算的洗礼，不能赶上时代潮流的技术和公司都将被无情的淘汰。大数据也要拥抱云计算，大数据将作为一种数据服务来提供（DaaS-Dataas A Service），依靠云提供共享的、弹性、按需分配的大数据计算和存储的服务。

Greenplum MPP 数据库从已一开始就是开放的技术，并且在 2015 年年底已经开源和成立社区（在开源第一天就有上千个 Download），可以说，Greenplum 已经不仅仅只是 Pivotal 公司一家的产品，我们相信越来越多组织和个人会成为 Greenplum 的 Contributor 贡献者，随着社区的发展将推动 GreenplumMPP 数据库走向新的高速发展旅程。（分享一下开源的直接好处，最近我们某用户的一个特殊需求，加载数据中有回车等特殊字符，我们下载了 GP 外部表 gpfdist 源代码，不到一天就轻松搞定问题）

Greenplum 也正在积极的拥抱云计算，Cloud Foundry 的 PaaS 云平台正在技术考虑把 Greenplum MPP 做为 DaaS 服务来提供，对于 Mesos 或其它云计算技术的爱好者，也可以考虑采用容器镜像技术 + 集群资源框架管理技术来部署 Greenplum, 从而可以实现在公共计算资源集群上的 MPP 敏捷部署和资源共享与分配。

总之，相信沿着开放、开源、云计算的路线继续前行，Greenplum MPP 数据库在新的时代将保持旺盛的生命力，继续高速发展。

二、Greenplum 背后的帝国



在 Pivotal 中国的发展历史上，有一个产品名字必须被铭记，那就是今天在中国 Massively Parallel Processing 大规模并行处理 (MPP) 数据库领域当之无愧的领头羊 – Greenplum。不夸张的讲，今天 Greenplum 服务的客户已经遍及我们每个人生活中的方方面面，在中国的电信、银行、保险、证券、交通、

物流、互联网和制造业当中，在几百个 Greenplum 数据库中运行的数据就像奔腾的石油，滚滚而来，不断输送到各个不同的战线。

今天的大数据再也不是几年前热炒的概念，越来越多的客户开始从 MPP 项目开始着手，逐渐利用混搭的技术来构造未来的数据之湖。但是毫无疑问的是，MPP 数据库市场的迅速发展过程也见证了 Greenplum 在中国的成长之路。

自 2008 年 12 月进入中国，到目前为止，在国内已经发展了将近两百个客户。在大数据领域，Greenplum 以其优越的性能为越来越多的客户所理解与接受。作为中国大数据市场的探路先锋，Pivotal 将以一个我们亲力打造的大数据项目为例，与大家一起分享 Greenplum 在项目实施过程中为客户创造的那些真实的价值，如何与客户一起开拓大数据之路。

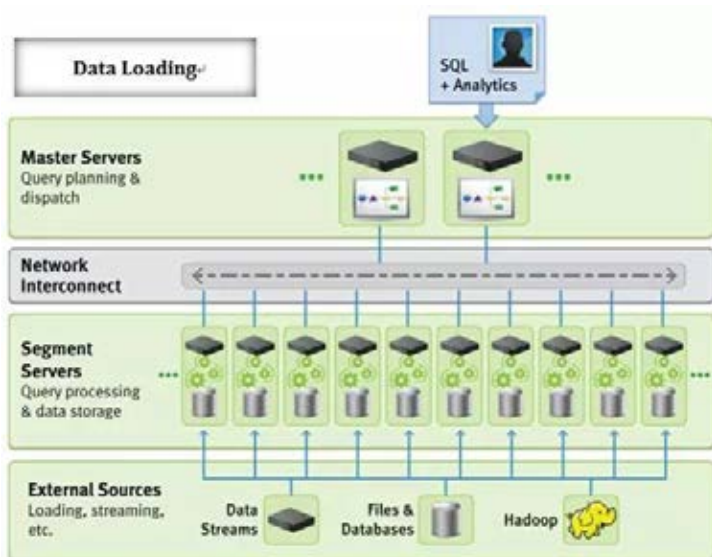
在国内某大型金融机构的大数据处理平台，使用 Greenplum 数据库产品支撑其 ODS 及各类集市应用。项目从 2013 年 6 月份开始到 2015 年底，生产环境已经由最初的一套集群发展到 10 多套，装机数量也从最初的 50 台发展到现在的数百台。短短两年半时间，服务器数量、集群数量、支撑的应用数量都飞速增长。

Greenplum 数据库在该客户发展如此迅猛，与产品在高吞吐、开放性、易扩展等方面的卓越表现是分不开的。

1. 高吞吐

该客户大数据平台的 ODS 区，接入了源端近百个业务系统的生产数据，每天需要加载进来的数据大概 5TB 左右。标准化处理完成后，需要给后端的公共访问、类别繁多的沙箱类应用供数。

每月月初，业务繁忙时段，保守估计平均每天需要给下游系统提供 10TB 的压缩数据。如此大规模数据处理，加上严格的数据时效性要求，不选择 Greenplum 这种吞吐性能特别优秀的产品，很难满足业务部门的要求。

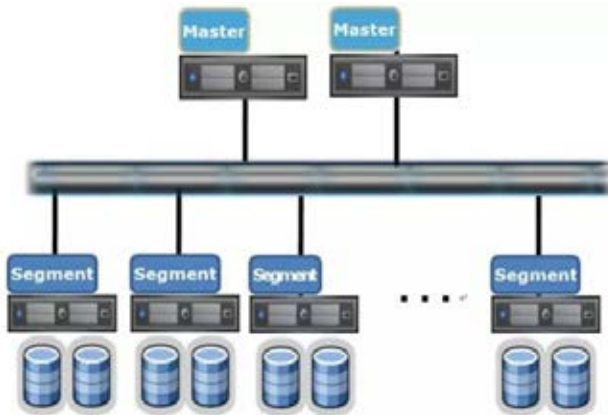


2. 开放性

在短短的两年半时间里，该客户已经选择了多个厂商的 PC 服务器用于部署 Greenplum，包括 DELL、HP、IBM 等等，且在单个集群混合了不同厂商不同型号的服务器。Greenplum 数据库在这些硬件上的表现都很稳定。

开放性的特性给客户带来的好处，不只是硬件厂商和型号的选择范围，也包含工程实施过程的便利性。2014 年，该客户大数据平台需要进行数据搬迁，Greenplum 采用了旧环境数据备份、传输、新环境恢复的方案，停机时间实际只花了不到 4 天。

相比较而言，其他封闭式系统，需要压缩并备份数据，倒腾出整套设备搬迁到新数据中心，然后再导入新数据，影响或暂停业务几十天。两种方案从工程复杂度、人力投入、业务影响来说，开放式架构所带来的便利和优势体现的淋漓尽致。



3. 可扩展

该客户单个 Greenplum 集群，从最初的 50 节点，经历了两次扩展，最终扩展到了上百节点。每次扩容，数据库的数据容量不但得到提升，业务人员更能直观的感受得到相同模型运行速度得到提升，尤其是大机构的大模型更为明显。

第一次扩容是从 50 节点到 74 节点，完成 30TB 业务数据的导出、传输、导入，以及 70TB 左右的索引数据创建，实际停机时间大约 3 天左右。

第二次扩容则到了上百节点。考虑到升级操作的可控性并缩短升级和扩容时间，仍然采用新初始化集群的方案操作，在同一集群中初始化了一个新数据库，将数据从旧库导出后，再导入新库。该方案在 2 天之内完成 57TB 压缩数据的加载和 130TB 索引数据的创建。



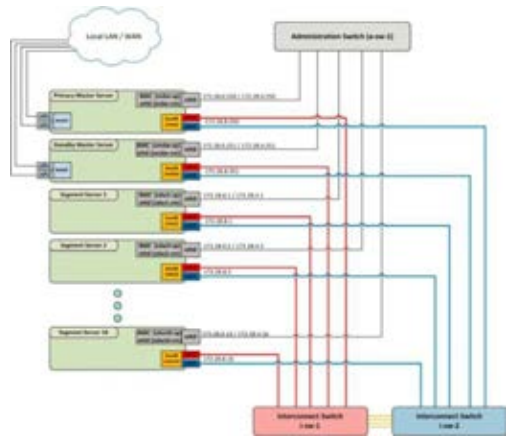
随着中国大数据市场的井喷趋势，作为中国第一代大数据的实践者，Pivotal 已经看到并且能够深刻的感觉到，Pivotal Greenplum 必将成为更多客户的选择，Greenplum 的开源也将让全世界的企业都可以立刻享用到开放技术的卓越之处，Pivotal 开放的技术趋势，和

更多新的大数据产品的逐渐融合，也正在引领着下一个大数据的潮头。我们期待着架构先进、性能优越的 Greenplum 数据库能够将我们的实践和真知灼见转化成能够为更多客户服务的宝贵价值！

三、Greenplum 硬件选型篇

前面我们通过某金融行业客户 Greenplum 的使用案例介绍了 Greenplum 产品在高吞吐、开放性、易扩展等方面的卓越表现及其带给客户的高价值。下面我们将从最佳实践方面介绍下 Greenplum 在具体规划实施时，如何选取硬件。

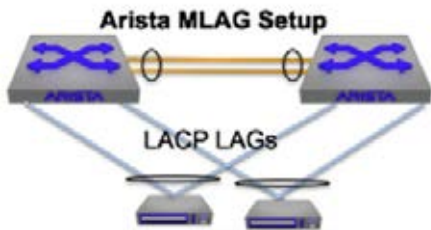
Greenplum 是通过软件将多台 x86 服务器的硬件组织在一起同时对外提供服务，从而达到高速处理的能力，为了达到这样的目的，硬件之间要彼此互通，并且整体性能上必须是均衡的，不同组件之间不能存在明显的性能短板，Greenplum 内部互连的示意图如下图所示：



1. 网络交换机

从上图中可以看出 Greenplum 并不是完全 sharednothing 的架构，Greenplum 是通过以太网将多台物理机连在一起，也就是网络资源是大家共享的，所以在部署 Greenplum 集群的时候，一定要规划好网络设备的接入，在达到性能最大的同时，也要考虑大流量对现有业务系统是否造成影响。

Greenplum 建议采用以太网万兆交换机，并通过设定跨设备链路聚合组 (MC-LAG Multi-ChassisLink Aggregation Group) 的方式将两台交换机连在一起，在服务器上将网卡通过 LACP (IEEE 802.3ad/802.1ax Link Aggregation Control Protocol) 协议做绑定形成链路聚合组 (LAG Link Aggregation Group)，如下图所示：



这样做的好处是，所有链路设备同时对外提供服务，并互相备份，网络设备达到了最大的高可用和吞吐，任意网络硬件故障都不会影响集群的正常使用。

- Greenplum 软件本身不支持 RDMA (Remote Direct Memory Access) 协议，所以如果基于 Infiniband 交换机对 Greenplum 集群组网，必须在服务器上安装硬件厂商提供的网络驱动，并通过 IPoIB(InternetProtocol over InfiniBand) 协议进行转换，之前遇到某客户因为驱动和服务器硬件兼容问题，压力一大，服务器自就会自动重启，最后又将网络设备改回万兆交换机。
- 网卡绑定采用 mode=4 (802.3ad)，流量传输 hash 策略选用 `xmit_hash_policy=layer3+4` 【 $((\text{sourceport XOR dest port}) \text{ XOR } ((\text{source IP XOR dest IP}) \text{ AND } 0\text{xffff}) \text{ modulo } \langle\text{slavecount}\rangle)$ 】，这样能保证流量均匀的打在多块网卡上，另外注意采用 mode4 绑定的时候，一定要把交换机设置在 802.3ad 模式下。

2. 主节点服务器

Greenplum 集群是有 master 架构，关于有、无 master 架构业界一直有所争论。

从功能上而言 master 节点是对外服务的入口，用户所有的请求都必须先经过 master，所以 master 节点的可用性直接关系到集群的稳定，但从实践经验来看，由于 master 节点只存元数据，只负责 SQL 的解析、分发以及最终计算结果的展现，所以承担的负载一般都非常小，故障率也极低，在我们维护阿里 Greenplum 集群 3 年的时间里，以及接触到的客户中，基本上没有碰到由于 master 故障导致集群不可用的情况，唯一一次，还是因为客户误操作同时将 master 和 standby 节点的数据目录 `rm -rf`。

Master 节点推荐采用硬件规格如下：两块万兆网卡（一般多为单网卡双网口）用于内部互连，1-2 块千兆网卡用于带外管理和接入客户网络，内存 DDR4 64GB 以上（推荐 256GB），硬盘 6 块（600GB 或 900GB 10K RPM SAS 盘，采用 RAID5 或者 RAID10，需要预留单独的 hotspare 盘），CPU 2 路 8 核及以上（主频 2.5G HZ 以上），1 块 RAID 卡（要求 cache 大小 1GB 以上，并带有掉电保护功能）

3. 计算节点服务器

由于计算节点真正的负责计算，计算节点的硬件性能直接影响到整体集群的性能，Greenplum 建议单个集群最开始搭建的时候最好选用相同规格的计算服务器，后续扩容也要保证新加机器的性能不能低于原有机器，这是由于 MPP 架构本身存在木桶效应，单台机器的性能短板，很可能导致整体集群变慢，虽然 Greenplum 可以根据具体的硬件配置，初始化的时候调整每台计算节点部署的实例数，但从具体实施看，极少有客户这么做。

Segment 节点推荐采用硬件规格如下：两块万兆网卡（一般多为单网卡双网口）用于内部互连，1-2 块千兆网卡用于带外管理和接入客户网络，内存 DDR4 64GB 以上（推荐 256GB），硬盘 24 块（600GB 或 900GB 10K RPM SAS 盘，采用 RAID5 或者 RAID10，需要预留单独的 hotspare 盘），CPU 2 路 8 核及以上（主频 2.5GHZ 以上），1-2 块 RAID 卡（单块 RAID 卡的 cache 大小 1GB 以上，并带有掉电保护功能，RAID 卡应为多通道，目前接触的硬件厂商中，单通道支持的最大磁盘数为 16 块）

-
- 硬盘尽量选用 SAS 盘，从实践经验看，硬盘故障是 Greenplum 集群中最为常见的故障类型，而 SAS 盘相比 SATA 盘在性能和稳定性上都明显的高于 SATA 盘。
 - RAID 卡一定要带 cache，否则做完 RAID 后写的性能会非常差，曾遇到一客户，把 GP 迁移到更大容量、更多磁盘的机器后，集群性能反而下降，就是因为新采购的机器 RAID 卡没有 cache。
-

- 加强硬件的监控，所有故障中，最怕硬件半死不活的状态，曾遇到一客户，由于硬盘发生坏道，但 RAID 卡并未将其标记为 down，导致坏盘在读写性能非常差的情况下仍然对外提供服务，最终将整个集群拖慢。
- 如果客户的实际应用还存在大量较高并发的小 IO 操作，比如随机查询，可以考虑 SSD+SAS+ 表空间的方式，并将随机 IO 类应用对应的表放在 SSD 设备上，从而有效的隔离底层 IO，达到更好 SLA。某电信用户存在大量对 400 个字段以上大宽表的随机查询场景，之前由于 IO 资源争用，在批量作业调起时随机查询响应时间显著增加，通过这种方式改造后，达到了很好的效果。
- 预留一台硬件服务器作为整个集群的灾备机，提前装好操作系统和数据库软件，并放在与现有集群相同的网络环境中，一旦硬件出现故障，我们可以迅速采取相应的修复措施，如果底层 RAID 没有损坏，在单台机器数据量过大比如接近 10T 的情况下，我们可以直接将磁盘插入到灾备机，由于 RAID 信息写在磁盘上，对调磁盘后，所有数据信息仍然保留，这样就能避免数据同步带来的性能损耗，这种方式要求集群所有机器采用相同规格的 RAID 卡。

以下是我们新一代一体机硬件和机柜配置，大家可以参考：

Component	Masters	Segment Servers
Processor	2 E5-2680v3	2 E5-2680v3
Memory	256GB of DDR4 @ 2133MHz (8x32GB DIMMs)	256GB of DDR4 @ 2133MHz (8x32GB DIMMs).
Size	1U	2U
Sockets	Dual Socket	Dual Socket
Cores	24	24
Network	Dual (one internal, one for customer network)	Single 10G Dual Port NIC for internal connection
Controller	12Gb x8 SAS controller (LSI base, intel branded)	1 x 12Gb x8 SAS controller (LSI based, intel branded) w/ 12Gb SAS expander
Disks	(6) 1.8Tb 10k RPM SAS	(24) 1.8Tb 10k RPM SAS

4. ETL 服务器

ETL 服务器是数据的临时存放区，由于 Greenplum 服务器并行加载的特点，数据可以直接通过网络从 ETL 服务器导入到 Greenplum 计算节点，所以 ETL 服务器网络和磁盘 IO 的性能直接关系到数据加载和卸载的性能，官方的测试数据 16 台计算节点 Greenplum 集群，加载性能可以达到 16TB/小时。

ETL 服务器推荐采用的硬件规格：两块万兆网卡（一般多为单网卡双网口），1 块千兆网卡用于带外管理，内存 DDR4 64GB 以上，硬盘 12 块（600GB 或 900GB 10K RPM SAS 盘，采用 RAID5，预留 hotspare 盘），CPU 2 路 6 核以上（主频 2G HZ 以上），1 块 RAID 卡（单块 RAID 卡的 cache 大小 1GB 以上，并带有掉电保护功能）。

- ETL 服务器最好和 Greenplum 接入相同的二层交换机，如果跨交换机，确保网络不存在性能瓶颈。
- ETL 服务器网卡尽量选用万兆网卡，某香港航空公司曾抱怨 Greenplum 加载性能缓慢，结果发现加载时把所有 ETL 服务器的千兆网卡都打满了。

由于每个客户实际的情况不同，我们介入产品实施时，往往是硬件已经采购，并且网络布线完成，由于底层硬件的一些限制，并不能充分发挥出 Greenplum 的最大性能优势，如果客户之前能根据业务特点规划好硬件采购，并合理的实施，可以避免后面很多的问题。

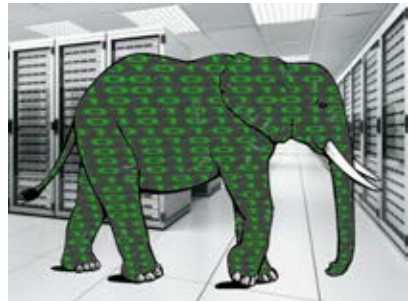
Greenplum 不挑硬件：无论是 Cisco 还是华三的交换机；无论是 IBM、DELL、HP 还是华为、浪潮的 PC 机；无论是刀片还是 PC 在国内外都有大量的案例；Greenplum 不挑系统：无论是 RedHat、CentOS 还是 SuSe，Greenplum 都可以畅快的运行，你甚至可以在自己的 MAC 笔记本上直接安装、玩耍；但 Greenplum 确实依赖于底层的硬件：只有合理的硬件搭配、准确的规划实施、定期的运行维护加上完善的软硬件监控，才是真正保证企业级数据仓库成功实施关键。

1. 前期规划的重要性

在硬件选型上，我们讲究达到平衡。是要在性能、容量、成本等多个方面综合考虑，取得平衡。不能一味追求容量而忽略了整体性能，忽略了日后维护和扩容的成本；也不能一味追求性能而忽略了成本，而让采购部门望而却步。

搭建 Greenplum 集群，首要考虑单个 Segment host 上的实例个数，随着现在单台 PC 服务器的处理能力不断提升，规划实例个数已经不能简单的按照 CPU 核数来推算。单台服务器的实例个数与数据库的并发处理能力是成反比的。因此，规划单台服务器的实例个数需要综合考虑服务器配置、生产环境的运行负载压力、跑批用户和前段查询用户并发需求等各个方面。大多数场景下，4 或 6 个为宜。

同样，作为整体架构设计的重要组成部分，ETL 服务器、监控管理，备份策略如何规划，如何高效组网都得在前期考虑好。在我们的成功案例中，同一个企业级数据平台中 Greenplum 集群和 Hadoop 集群配合运作的案例越来越多。在中国移动的大数据架构规范中，云化 ETL 是一个重要的组成部分。云化 ETL 就是构架在 Hadoop 集群之上。Greenplum 提供了专用产品模块 gphdfs，Greenplum 通过 gphdfs 可以直接与 HDFS 上的数据进行交互，并且可以同时发挥 Greenplum 和 Hadoop 两者并行处理的优势。



2. 数据模型设计的重要性

实施 Greenplum 的项目，有的是从其他数据库产品迁移过来的数据模型，有的是新设计的数据模型。无论是哪种情况，设计时请重点关注 Greenplum 的特性，要充分发挥 Greenplum 所长。

- **分布键：**

均匀为第一大原则，选取更有业务意义的字段，并非必须选择原库的主键（PK）。

- **压缩表使用：**

大表都要采用压缩存储，既节省空间也节省 IO 资源。长远来看还可降低阵列卡和磁盘的故障率。

- **行存还是列存：**

列存储有更高的压缩率，合适于聚合运算，但不合适于宽表。一个数据库中不应只有一种存储方式，每张表应依据实际情况设计存储方式。

- **临时表：**

对于程序中所使用到的临时表和中间表，上述 3 点规则同样适用。

- **分区：**

Greenplum 的分区原理与其他数据库无异。表的子分区个数不宜过多，子分区粒度不易过细，子分区之间无需均匀。

- **索引：**

在 Greenplum 中，可以使用索引但不能滥用。与 OLTP 类型数据库不同，Greenplum 在绝大部分关联场景中不会用到索引。只有部分小结果集的查询场景中需要使用索引优化。

3. 日常运维的重要性

切忌运而不维！想要一个数据库长久健康的运行，是离不开一个称职的 DBA。从其他数据库的 DBA 转为 Greenplum 的 DBA 并不是一件很困难的事，成功转成 Greenplum DBA 的工程师越来越多。

现在企业客户中搭建的 Greenplum 集群服务器数量是越来越大，在电信行业和银行业，搭建 50 台服务器以上的 Greenplum 集群越来越多。而集群服务器数量越多也就代表故障发生率越高。作为 Greenplum 的 DBA 和运维人员，不单只关注 Greenplum 本身，还要关注集群中各硬件的状况，及时发现及时处理。硬盘状态、阵列卡状态、硬件告警、操作系统告警、空间使用率等都是应关注的重点。这些都可通过厂商提供的工具，编写监控程序，SNMP 协议对接企业监控平台等手段提升日常巡检和监控的效率。



针对 Greenplum，DBA 需要关注重点：

- Greenplum 的状态：Standby master 的同步状态往往容易被忽略。通过监控平台或者脚本程序，能够及时告警则最好。
- 系统表：日常系统表维护（vacuum analyze），在系统投产时就应该配置好每天执行维护。

- 统计信息收集：统计信息的准确性影响到运行效率，用户表应该及时收集统计信息。在应用程序中增加手机统计信息的处理逻辑，通过脚本定时批量收集统计信息，或者两者相结合。针对分区表日常可按需收集子分区的统计信息，可节省时间提升效率。
- 表倾斜：表倾斜情况应该 DBA 的关注点之一，但无需每天处理。
- 表膨胀：基于 postgresql 的 MVCC 机制，表膨胀情况不能忽视。重点应该关注日常更新和删除操作的表。
- 报错信息：在日志中错误信息多种多样，大部分不是 DBA 需要关注的。应该重点关注 PANIC、OOM、Internal error 等关键信息。

Greenplum 已经开源了，我们的生态圈在迅速地壮大，Greenplum 的爱好者、拥护者人数也在不断地壮大。在使用和探索 Greenplum 的路途中，我们通过一点经验介绍，希望大家少走弯路。在产品实施过程中的关键阶段，还应该更多地寻求专业顾问的支持。

五、Greenplum 系统表的维护及修复技巧



Greenplum 与其他所有关系型数据库一样，拥有一套管理数据库内部对象及关联关系的元数据表，我们称之为 Greenplum 系统表。Greenplum 的产品内核是基于 postgresql 数据库基础上开发完成的，因此，Greenplum 系统表很多继承于 postgresql 数据库。

Greenplum 的系统表大致可分为这几类：

1) 数据库内部对象的元数据

如：pg_database、pg_namespace、pg_class、pg_attribute、pg_type、pg_exttable 等。

这类系统表既涵盖了全局的对象定义，也涵盖了每个数据库内的各种对象定义。这类系统表的元数据不是分布式的存储，而是每一个数据库实例（不论是 master 实例还是 segment 实例）中都各有一份完整的元数据。但也有例外，如：gp_distribution_policy（分布键定义）表则只在 master 上才有元数据。对于这类系统表，各个实例之间元数据保持一致十分重要。

2) 维护 Greenplum 集群状态的元数据

如：gp_segment_configuration、gp_configuration_history、pg_stat_replication 等。

这类系统表主要由 master 实例负责维护，就如 segment 实例状态管理的两张表 gp_segment_configuration 和 gp_configuration_history 的数据是由 master 的专用进程 fts 负责维护的。

3) Persistent table

如：gp_persistent_database_node、gp_persistent_filespace_node、gp_persistent_relation_node、gp_persistent_tablespace_node。

这类系统表同样是存在于每一个数据库实例中。在每个实例内，persistenttable 与 pg_class/pg_relation_node/pg_database 等系统表有着严格的主外键关系。这类系统表也是 primary 实例与 mirror 实例之间实现同步的重要参考数据。

在 Greenplum 集群出现故障时，会有可能导致系统表数据有问题。系统表出现问题会导致很多种故障产生，如：某些数据库对象不可用，实例恢复不成功，实例启动不成功等。针对系统表相关的问题，我们应该结合各个实例的日志信息，系统表的检查结果一起定位问题，本文将介绍一些定位、分析及解决问题的方法和技巧。

1. 检查工具

Greenplum 提供了一个系统表检查工具 gpcheckcat。该工具在 \$GPHOME/bin/lib 目录下。该工具必须要在 Greenplum 数据库空闲的时候检查才最准确。若在大量任务运行时，检查结果将会受到干扰，不利于定位问题。因此，在使用 gpcheckcat 前建议使用限制模式启动数据库，确保没有其他应用任务干扰。

2. 分析方法和处理技巧

- 1) 遇到临时 schema 的问题，命名为 pg_temp_XXXXX，可以直接删除。通过 gpcheckcat 检查后，会自动生成对临时 schema 的修复脚本。由于临时 schema 的问题会干扰检查结果，因此，处理完后，需要再次用 gpcheckcat 检查。
- 2) 如遇个别表对象元数据不一致的情况，通常只会影响该对象的使用，不会影响到整个集群。如果只是个别实例中存在问题，可以通过 Utility 模式连接到问题实例中处理。处理原则是尽量不要直接更改系统表的数据，而是采用数据库的手段去解决，如：drop table/alter table 等。
- 3) persistent table 问题，这类问题往往比较棘手，影响也比较大。依据 gpcheckcat 的检查结果，必须把 persistent table 以外的所有问题修复完之后，才可以着手处理 persistent table 的问题。

针对 persistent table 再展开讲述几种问题的处理技巧：

- 1) 报错的 Segment 实例日志中出现类似信息

```
"FATAL","XX000","Number of freeTIDs 191, do not match maximum free order number 258, for 'gp_persistent_relation_node'"
```

该错误可能会导致实例启动失败，数据库实例恢复失败等情况。首先可在问题的实例（postgresql.conf）中设置参数 gp_persistent_skip_free_list=true。让出问题的实例先启动起来，再进行 gpcheckcat 检查。

在 gpcheckcat 的结果中应该能找到类似的问题:

```
[INFO]:- [FAIL] gp_persistent_relation_node <=> gp_relation_
node
[ERROR]:-gp_persistent_relation_node <=> gp_relation_node
found 442 issue(s)
.....
[ERROR]:-sdw5:40000:/data1/primary/gpseg24
[ERROR]:- relfilenode | ctid | persistent_tid
[ERROR]:- 13795767 | (205,18) | None
[ERROR]:- 13795768 | (205,17) | None
[ERROR]:- 13795769 | (7444,134) | None
[ERROR]:- 13799293 | (89,226) | None
.....
[INFO]:-[FAIL] gp_persistent_relation_node <=> pg_class
[ERROR]:-gp_persistent_relation_node <=> pg_class found 442
issue(s)
.....
[ERROR]:-sdw5:40000:/data1/primary/gpseg24
[ERROR]:- relfilenode | nspname | relname | relkind | relstorage
[ERROR]:- 13795741 | None | None | None | None
[ERROR]:- 13795741 | None | None | None | None
[ERROR]:- 13795741 | None | None | None | None
[ERROR]:- 13795741 | None | None | None | None
```

从上述检查结果可以看出 persistent table 的部分数据和其他系统表对应关系不正确。处理方法就是要修复 persistent table 数据。

2) 报错的实例日志中出现类似信息

FATAL:Global sequence number 1131954 less than maximum value 1131958 found in scan ('gp_persistent_relation_node')

该问题可能会导致实例启动失败。可在问题的实例（ postgresql.conf ）中设置参数 `gp_persistent_repair_global_sequence=true`，便可修复相应问题，让相应实例正常启动。

3) 报错的实例日志中出现类似信息

Persistent 1663/17226/21248339, segment file #1, current new EOF is greater than update new

EOF for Append-Only mirror resync EOFs recovery (current new EOF 1419080, update new EOF 1416416), persistent serial num 5725616 at TID (3353,1)

该问题会出现在 AO 表中，表示个别实例上的数据文件被损坏。问题可能会导致进程 PANIC，实例启动失败。首先可在问题的实例（ postgresql.conf ）中设置参数 `gp_crash_recovery_suppress_ao_eof=true`。让出问题的实例先启动起来。再进行 `gpcheckcat` 检查。确定问题所在并修复。而通常出问题的 AO 表已经损坏，建议 `rename` 或者删除。

4) 在 gpcheckcat 的检查结果中如果出现如下信息

```
[INFO]:-[FAIL] gp_persistent_relation_node <=> filesystem
[ERROR]:-gp_persistent_relation_node <=> filesystem found
52 issue(s)
.....
[ERROR]:-sdw1:40000:/data1/primary/gpseg0
[ERROR]:- tablespace_oid | database_oid | relfilenode_oid |
segment_file_num | filesystem | persistent | relkind | relstorage
[ERROR]:- 17001 | 272379 | 121899432 | 0 | t | f | None | None
[ERROR]:- 17012 | 272379 | 121692973 | 1 | t | f | r | c
[ERROR]:- 17012 | 272379 | 121693149 | 1 | t | f | r | c
[ERROR]:- 17012 | 272379 | 121694359 | 1 | t | f | r | c
```

检查结果表明文件系统中存在部分数据文件在系统表中没有对应的关系，也就是文件系统中有多余的数据文件。这种情况不会影响 Greenplum 集群的正常运作，可以暂时忽略不处理。

修复 persistent table 表的问题，不可手工修改，只能够使用 Greenplum 提供的修复工具 gppersistentrebuild 进行修复。工具提供了备份功能，在操作修复之前必须要执行备份操作。然后通过 gppersistentrebuild，指定待修复的实例的 contentid 进行修复。

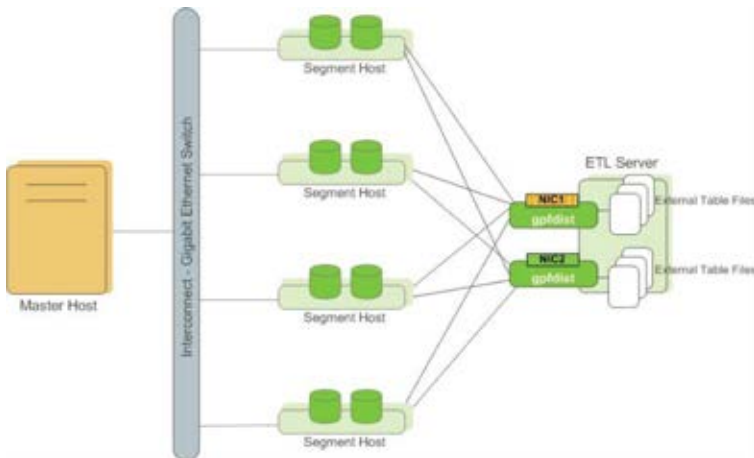
另外，如果 primary 实例与 mirror 实例之间是处于 changetracking 状态。一旦 primary 实例进行了 persistent table 的修复操作，primary 实例与 mirror 实例之间只能执行全量恢复操作（gprecoverseg -F）。

上面所介绍的一些 GUC 参数，都是在修复系统表过程中临时增加的参数，待集群恢复正常之后，请将所修改过的 GUC 参数值恢复回原有默认状态。

六、Greenplum 的开发和优化

1. 外部表加载性能分析

下图是 GP 用户很熟悉的 GP 外部表原理示意图，下面分析一下外部表加载的性能：



首先，数据的流向：

Segment Host 从 gpfdist 服务请求数据，不同的 Segment Host 之间需要互相传输数据，加载过程中数据的重分布和 Mirror 策略都需要占用 Segment Host 之间的网络资源。

这里需要假设一下硬件指标：

主流配置，每个节点机器的磁盘 IO 性能都可达到 1GB/S 以上 (10 块磁盘，如果 RAID 卡很优秀，甚至可以达到 2.0GB)，每个节点的网络带宽达到 1GB/S 以上 (1 个万兆)，CPU 主频在 2.6GHz 到 3GHz 左右，每台机器部署 4~6 个 Primary Instance(生产经验值)。

接下来，评估一下外部表加载数据的性能：

- 1) 一个 gpfdist 服务只使用一个 CPU 核的资源。
因此，通常，一个 gpfdist 服务每秒可以处理约 400MB~450MB 的 TXT 文本，如果是 CSV 格式，每秒可以处理约 300MB~350MB。
- 2) 一个 PrimaryInstance 在处理一个外部表加载时，只使用一个 CPU 核的资源。

通常，一个 Primary Instance 在处理外部表的数据时，每秒处理大约 10MB~15MB 的数据。如果采用深度压缩算法，处理能力还会再打些折扣。

- 3) 根据外部表工作原因，PrimaryInstance 从 gpfdist 服务抓取数据 (D) 是随机的，与分布策略无关。假设数据分布平坦且顺序随机，系统中总共有 N 个 Primary Instance，则，D 中有 $(N-1)/N$ 的数据需要发送到其他节点。因此，节点之间的网络流量占用大约为 gpfdist 输出带宽的 2 倍。再考虑 Mirror 配置，节点之间的网络带宽约为 gpfdist 输出带宽的 3 倍。

综上所述，在主流软硬件环境下，外部表加载，最先成为瓶颈的往往一定是 CPU，如果 gpfdist 性能不足，可以增加 gpfdist 数量；如果节点数量成为瓶颈，可以增加外部表加载的并发度或者增加 Primary Instance 数量。

如果不是主流硬件环境，对于瓶颈的分析，请参照上述标准评估。通常，40 个左右的 Primary Instance 处理性能匹配一个 gpfdist 服务的处理性能，上了万兆网卡，网络就永远不会成为加载时的瓶颈，磁盘一般也不会成为加载时的瓶颈。有兴趣的可以去看看 COPY 的源码，那里也许有可以优化的地方。

2. 执行计划对性能的影响

所谓执行计划，就是 GP 根据 SQL 和相关统计信息规划出的一种执行线路图。之后的执行将严格按照该线路图执行（至少目前的产品特点如此），执行计划的优劣直接决定了 SQL 性能。在 pgAdmin3 中看到如下执行计划节点时，需要警惕：

广播 (Broadcast Motion)

该操作的代价是，全部数据需要在每个 Primary Instance 上有一份完整的拷贝。如果一张表或者中间结果的尺寸是 10GB，集群有 100 个 Primary Instance，该操作等于在集群中传播 1TB(10GB × 100) 的数据！所以，对于那些尺寸较大的表或者中间结果出现这样的操作，都是需要坚决杜绝的。

循环匹配 (Nested Loop)

该操作的代价是，从每个 Primary Instance 上来看，两个集合呈现笛卡尔积方式的关联。这种情况一般还会伴随着广播一起出现，如果是 2 张 100 万数量级的表采用循环的方式关联，计算量为 100 万 × 100 万 = 1 万亿。这种操作，除非你确认计算量是可控的，否则，花多大的代价来避免都是值得的，比如非等值关联。靠硬算，再强的 CPU 也有玩不转的时候。

汇集 (Gather Motion)

该操作是将 Segment 节点上的数据汇总到 Master 节点再进行后续的计算。如果该操作不是出现在最后阶段，就等于放弃了 MPP 的并行优势，由 Master 自己承担余下的所有任务！这里需要提到一个参数 `gp_dynamic_partition_pruning`，如果在使用分区表时遇到这个问题，可以尝试关闭该参数。

重分布 (Redistribute Motion)

该操作会将数据在所有节点之间做一次重新分布，对于超大表或者网络配置不是极高的环境应该尽量避免。不过，这个操作是不可能不发生的，需要尽量避免大表重分布——这与分布键的选择有直接关系。建议，不要选择组合分布键，因为，在两表 JOIN 时，只有关联字段包含所有分布键，才能确保数据不需要任何移动就满足 LocalJoin 的要求。

然后，再来说说，执行计划根据什么来决定数据在节点之间如何移动。当两张表关联时，到底是选择重分布还是选择广播。需要分以下几种情况来说明：

-
- 1) 两张表的关联字段都包含了分布键，此场景不需要在节点之间移动数据。
 - 2) 有一张表的关联字段包含分布键，另外一张不包含分布键。
 - 3) 两张表的关联字段都没有包含分布键（此处所说的包含分布键，意思是包含所有用作分布键的字段，即分布键的超集）。
-

后两种场景，都需要有数据移动，GP 的执行规划器会选择哪一种呢？下面详细分析：

在执行计划中，每个有结果集的节点都会有 rows 与 width 两个指标，这两个指标的乘积表示执行计划对每个 Primary Instance 上此节点数据尺寸的评估，单位为字节。我们假设集群中 PrimaryInstance 的数量为 N，缺省情况下，执行规划器使用 N 作为 Primary Instance 数量来评估选择广播还是重分布。

有一个参数 `gp_segments_for_planner` 可以从执行计划的层面改变 `N`，从而影响下一步的执行计划。例如：

A 表与 B 表关联，A 表的尺寸信息为 `rows:rA`，`width:wA`，B 表的尺寸信息为 `rows:rB`，`width:wB`。

执行规划器会计算以下数值：

$rA \times wA \times N$ ——重分布 A 表代价

$rB \times wB \times N$ ——重分布 B 表代价

$rA \times wA \times N \times N$ ——广播 A 表代价

$rB \times wB \times N \times N$ ——广播 B 表代价

执行计划会根据以上 4 种代价，在符合逻辑关系的组合中，选择代价最低的路径。如果我们发现执行计划不如我们期望的那样，我们可以进行的干预是：

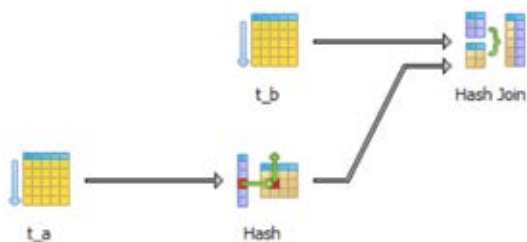
对相关的表收集统计信息，或者修改 `session` 级别的 `gp_segments_for_planner` 参数的值，不要试图在更高级别修改该参数，那样是极其危险的。

再者，执行计划为什么会选择不够高效的方法，由于执行计划是基于统计信息生成的，而统计信息不可能涵盖所有细节。通常，以下几种情况会导致评估失真：

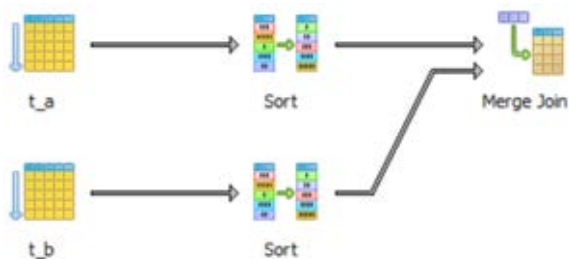
-
- 1) 过多的 WHERE 条件但并没有过滤很多数据。
 - 2) 过多的 JOIN 条件但并没有真正降低匹配数据量。
 - 3) 用重复率极低的字段做汇总。
-

因此，你需要很了解你的数据，这样才能达到优化的最高境界。

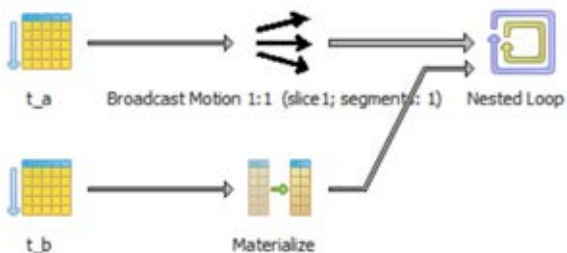
接下来说一说关联算法方面的话题。



这种是 Hash 关联，将一张表按照关联字段 Hash 到内存中，形成 Key-Value 散列，在扫描另外一张表的时候到散列中做匹配。



这种是 Merge 关联，会将关联的两张表先按照关联字段做排序，通常在内存不吃紧的情况下，排序算法不如 Hash 算法快。



这种是 Nested Loop 关联，等于是两张表做了笛卡尔积复杂度的匹配。

我们来看一下，在内存资源充足的情况下，三种场景的复杂度：

Hash Join: $o(N)$
Merge Join: $o(N \times \log N)$
Nested Loop Join: $o(N \times N)$

因此，在绝大多数情况下我们希望走 Hash Join 方法。如果 Join 方法不如我们期望，要找找原因，比如数据量评估错误，关联条件不对等。

通常，smallint、int、bigint 之间的关联是不能走等值关联的，也就是说，Hash Join 和 Merge Join 都是无法选择的，只能选择 Nestloop Join，这很致命。因此，如果你不能确保所有相关联的字段具有相同的数据类型，这三种数据类型中，建议永远不要使用 smallint 和 bigint，分别用 int 和 numeric 代替。

接下来说一说 SQL 语句的注意事项：

-
- 1) 要注意使用 LeftJoin(Right Join 类似) 的场景。避免大表 Hash。此场景会导致执行计划将右边的大表 Hash 到内存中，有时是性能拖累。应该将 SQL 改写为如下形式：
原 SQL: `selects.* ,b.xxx from small s left join big b on s.id=b.id`
修改: `select s.* ,b.xxx from small s left join (select b.* from big b where b.id in(select idfrom small)) b on s.id=b.id` 你会发现性能有很大提升。
 - 2) 尽量不要写出自己理解起来都有困难的 SQL。比如：
`select ta a left join tb b on a.id=b.id where b.timeflag=' 20140825' ;`
你会发现执行计划中没有 left join，因为这个 where 条件导致 left join 直接被执行计划降级为 inner join 了。一般来说 inner join 后面的 where 条件可以被提前到 table scan 节点，而 left join 的 where 条件是对结果的 filter，结果中要求 b 表字段符合 not null 的条件，自然也就没有 left join 的逻辑了。
-

3. 倾斜对性能的影响

关于数据分布倾斜，推荐一个方法：

```
with pg_cls as(
  selectoid,relnamespace,relname,
    (gp_statistics_estimate_reltuples_relpages_oid(oid)::bigint[])[1]+10^5 tuples
  from gp_dist_random('pg_class')where relkind='r' and relstorage<>'x'
)select n.nspname||'.'||c.relname rel_name,
  min(c.tuples)min_tuples,avg(c.tuples)avg_tuples,max(c.tuples) max_tuples
  from pg_cls,c,pg_namespace n,gp_distribution_policy p
  where c.relnamespace=n.oidand c.oid=p.localoid
    and (n.oid>16384 orn.nspname='public') and n.nspname not like 'pg_tmp_%'
    and c.oid not in(selectparrelid from pg_partition)
group by 1 havingmax(c.tuples)/avg(c.tuples)>2 ormin(c.tuples)/avg(c.tuples)<0.5
```

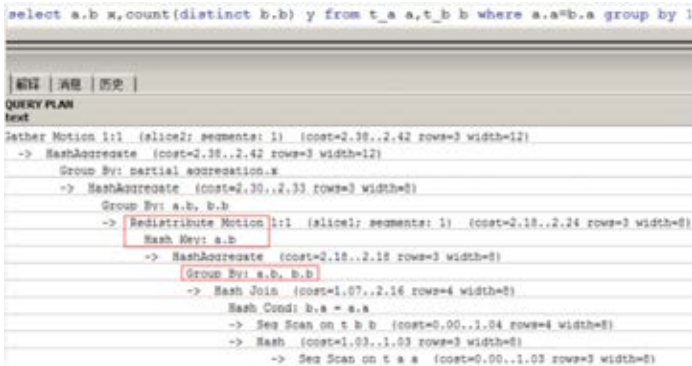
这种方法的优点是，速度快，至于其中的几个 Magic Number，逻辑能力稍强的人很容易明白，就不做过多解释。关键在于 gp_statistics_estimate_reltuples_relpages_oid 这个函数，该函数兼顾了准确性和性能，如果你希望有更多的用法，可以和 pg_partition，pg_partition_rule 等表做关联。

关于计算倾斜，基本上 2 个思路：

-
- 1) 执行计划错误，尝试收集统计信息，尝试修改某些执行计划相关的参数，以干涉执行计划。
 - 2) 执行计划看起来没有什么不正常，但某些步骤的中间结果出现严重倾斜的重分布。建议学会重写 SQL，将原有逻辑拆开，用更复杂的 SQL 逻辑换取低开销的执行计划。
-

比如：

```
select a.b x,count(distinct b.b) y from t_a a,t_b b where a.a=b.a group by 1
```

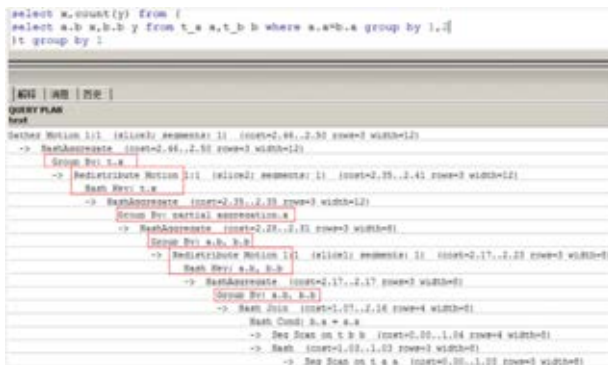


```
QUERY PLAN
text
Gather Motion #1: (slice:2; segments: 1) (cost=2.39..2.42 rows=3 width=12)
-> HashAggregate (cost=2.39..2.42 rows=3 width=12)
  Group By: partial aggregation,a
  -> HashAggregate (cost=2.39..2.33 rows=3 width=0)
    Group By: a,b, b,b
    -> Redistribute Motion #1: (slice:2; segments: 1) (cost=2.18..2.24 rows=3 width=0)
      Hash Key: a,b
      -> HashAggregate (cost=2.18..2.18 rows=3 width=0)
        Group By: a,b, b,b
        -> HashJoin (cost=1.07..2.16 rows=4 width=8)
          Hash Cond: b.a = a.a
          -> Seq Scan on t_b b (cost=0.00..1.04 rows=4 width=8)
          -> Hash (cost=1.03..1.03 rows=3 width=0)
            -> Seq Scan on t_a a (cost=0.00..1.03 rows=3 width=0)
```

如果 a,b,b 的唯一性很高，a,b 枚举数量很少，且 b,b 按照 a,b 的分布倾斜极其严重，那么在第一次 group by a,b,b,b 之后将基本上不会降低数据量，之后按照 a,b 的重分布操作将是极度倾斜的。这种场景并非故意设计出来，对于一些按照行业，区域等严重倾斜维度汇总的场景来说，是很常见的。

该 SQL 可以做如下修改：

```
select x,count(y) from (
select a.b x,b.b y from t_a a,t_b b where a.a=b.a group by 1,]
)t group by 1;
```



```
QUERY PLAN
text
Gather Motion #1: (slice:2; segments: 1) (cost=2.98..2.98 rows=3 width=12)
-> HashAggregate (cost=2.98..2.98 rows=3 width=12)
  Group By: a,b
  -> Redistribute Motion #1: (slice:2; segments: 1) (cost=2.39..2.41 rows=3 width=12)
    Hash Key: a,b
    -> HashAggregate (cost=2.39..2.33 rows=3 width=12)
      Group By: partial aggregation,a
      -> HashAggregate (cost=2.39..2.33 rows=3 width=0)
        Group By: a,b, b,b
        -> Redistribute Motion #1: (slice:2; segments: 1) (cost=2.17..2.23 rows=3 width=0)
          Hash Key: a,b, b,b
          -> HashAggregate (cost=2.17..2.17 rows=3 width=0)
            Group By: a,b, b,b
            -> HashJoin (cost=1.07..2.16 rows=4 width=8)
              Hash Cond: b.a = a.a
              -> Seq Scan on t_b b (cost=0.00..1.04 rows=4 width=8)
              -> Hash (cost=1.03..1.03 rows=3 width=0)
                -> Seq Scan on t_a a (cost=0.00..1.03 rows=3 width=0)
```

这样，中间结果的倾斜就消除了，因为在按照 a,b 重分布之前，已经按照 a,b 做了汇总，如果汇总还有其他 avg, sum, count 之类的指标，请参考数学知识进行拆解。

4. 关于函数

首先，Greenplum 更喜欢动态 SQL，就是没有占位符的那种，因为所有的变量都可以在编译时完全获取，尤其是分区条件，相信已经有大批开发人员在 function 中深受此问题困扰多年。

其次，你也许和我一样，更喜欢在 function 中能够直接将变量写进 SQL，同时，极度厌烦字符串拼接的方式 (双竖线) 去拼凑 SQL。

不但如此，你还希望在 pgAdmin3 中能够高亮显示你的所有 SQL，哇，太酷了！！

那么，plperl/plperlu 函数语言是你不二的选择，赶快拥抱吧，抛弃 sql 和 plpgsql。建议你在 plperl 中使用 qq{} (double quote) 来表示多行字符串，其中可以直接使用各种变量表达式。如果你愿意，你还可以把一些通用函数包装起来，打个 pm 包放到 perl 的系统目录中，然后你的 function 就很美观了。同时，你还可以很容易的使用 eval 以及 elog(level,msg) 打出 DEBUG, LOG, INFO, NOTICE, WARNING, ERROR 多种级别的信息。

还有 plpythonu，这个是 Greenplum 自带的 language，一些通用的函数，用 plpythonu 来编写也是极其不错的选择。

关于这两种语言，可以参考 PostgreSQL 文档的【服务器端编程】章节，以及 Perl 和 Python 语言。

Greenplum 是开放的数据库，又是开源的数据库，可以分享的知识其实真的很多，如果你这方面的知识基础还不是很很高，可以多读一些文章然后收藏起来，慢慢进步。

七、加密 Greenplum 中的静态数据

近几年，数据外泄的问题甚为猖獗，针对这一现象和相关的监管要求，很多公司都在努力提高数据安全性并对静态数据启用加密功能，这同样也适用于大数据，包括 Pivotal Greenplum 的用户。



Protegrity 是 Pivotal Greenplum 中的一个强大的替代性默认加密功能，可提供功能数据加密，更好地保障用户安全。在本文中，我们将解释与 Protegrity 的解决方案相比传统 Greenplum 加密功能的工作方式，展示 Greenplum 上的 Protegrity 设置，告诉读者如何加密静态数据并提供 SQL 代码的示例以调用 Protegrity 的加密功能。

1. 默认的 Greenplum 加密和 Protegrity 的功能数据加密



2015 年，Pivotal 发布了一份关于加密 Greenplum 数据库中数据的基础知识白皮书，提供了有关如何使用 PostgreSQL pgcrypto 扩展包加密 Greenplum 中数据的指导。虽然本方法在许多情况下都适用，但是使用 pgcrypto 加密静态数据会比较麻烦，具体取决于

组织和用户的需求。例如，你必须首先使用 GPG 创建密匙才能利用 pgcrypto 扩展包加密纯文本 / 字节，然后利用密匙对每个 INSERT 执行如下的 SQL 命令。

```
INSERTINTO userssn(username, ssn)
SELECT robotccs.username,pgp_pub_encrypt(robotccs.ssn, keys.
pubkey) AS ssn FROM (
VALUES ('Alice', '123-45-6788'), ('Bob','123-45-6799')) AS
robotccs(username,ssn)
CROSS JOIN (SELECT dearmor('-----BEGIN PGP PUBLIC KEY
BLOCK----- Version:GnuPG v2.0.14 (GNU/Linux)
mQENBFS1Zf0BCADNw8Qvk1V1C36Kfcwd3Kpm/dijPfRyyEwB6Pq
KyA05jtWiXZTh2His1ojSP6LI0cSkIqMU9LAlncecZhRlhBhuVgKIGSgd
9texg2nnSL9Admqik/yX ...
/UUZB+dYqCwtvX0nnBu1KNcMk2AkEcFK3YoliCxomdOxhFOv9AKj
jojDyC65KJciPv2MikPS2fKOAg1R3LpMa8zDEtI4w3vckPQNrQnNy
Utfj6ZoCv =XZ8J
-----END PGP PUBLICKEY BLOCK-----' AS pubkey) AS keys
```

此外，使用 pgcrypto 需要一个独立的密匙管理方案。虽然 PL/SQL 触发器机制可以更轻松地利用 pgcrypto，但是鉴于 Greenplum 的 MPP 架构，无法结合使用 pgcrypto 与触发器机制。所以，Pivotal 与 Protegrity 合作，一起简化了 Greenplum 中静态数据的加密操作。

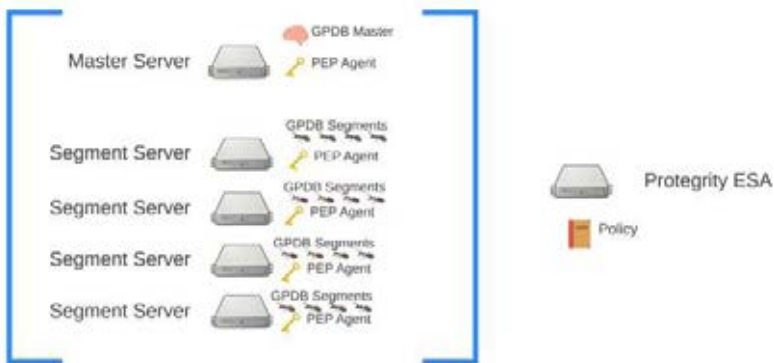
Protegrity 可提供一组在专门的转型流程中加密数据的功能。启用功能之后，物理数据将会采用新的格式保存在磁盘上以满足静态数据的加密要求。解密数据、确定用户是否可以访问全部或者部分的数据也是采用一个相似的流程。

此功能通过在加密之前捕获用户信息并将凭证传递至安装在所有 Greenplum 节点上的本地 Protegrity 代理完成加密流程。此流程与维护、存取和应用策略 (Policy) 目录的服务器通信，在加密期间，此功能将访问策略以应用、确定用户的数据访问权限，并根据策略解密和掩盖数据。

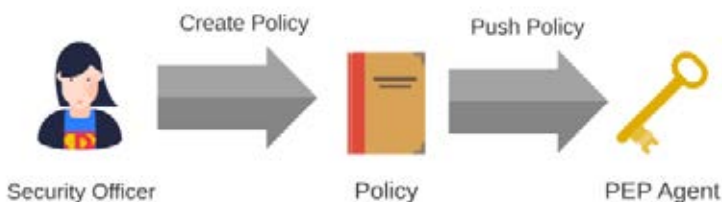
这些策略在集中的 Protegrity 企业安全管理员 (ESA) 服务中创建，并与用户有关，可向数据安全管理员提供一个中心点以便维护各个平台的数据策略。这样一来，管理员无需登录数据库就可以更改策略。此外，ESA 还可帮助划分权限，确保操作用户在未获得安全管理员许可的情况下不能访问数据。

2. 架构图概述

在较高级别主要部件是 Pivotal Greenplum 集群、Protegrity PEP 代理和 Protegrity ESA。



安全管理员将与 Protegrity ESA 交互。他们将宣布数据策略，确定将使用什么样的加密算法来保存数据。此外，他们将定义谁可以访问并解密数据的用户策略并明确一些规则，确定用户是否可以查看所有数据，或仅能看到经过隐蔽处理的部分数据。然后，这些策略将被下推到所有在 Pivotal Greenplum 服务器上运行的 PEP 代理处。



在 PivotalGreenplum 集群上，每当用户发出一个查询使用 Protegrity 所提供的功能时，查询操作就将立即检索数据，然后该功能将联系本地的 PEP 代理。此功能可向 PEP 代理告知使用查询的用户，这样，它就可以查看 ESA 提供的目录，了解适用于该用户的规则及其尝试访问的数据。它将取部分数据检查用户是否具有访问权限，然后 PEP 代理将对数据运行加密或解密并返回数值。



除了在数据库中执行此功能以外，Protegrity 还提供一些工具来加密数据库外的数据。这应作为 ETL 流程的一部分，在数据加载到 Pivotal Greenplum 之前转换数据，通过在摄取数据的时候消除解密动态数据的需求来加快提取数据的速度。

3. 在 Pivotal Greenplum 中设置 Protegrity

在使用任何 Protegrity 产品的时候都需要先安装 Protegrity ESA 服务器。Protegrity ESA 的安装不在本文的讨论范围之内，不过你可以从 Protegrity 客户团队中轻松获取相关信息。

在安装 ESA 之后，下一步将安装 Pivotal Greenplum 的 Protegrity 数据库保护器，这包括在所有的 Greenplum 节点和主节点上安装 PEP（Protegrity 执行点）服务器流程。这也意味着 Greenplum 集群内的所有服务器都应能够到达 Protegrity ESA 服务器，无论其是否联网。

Protegrity 可提供一份安装指南《Protegrity 数据库保护器 – Pivotal Greenplum》。该文件列明了在 Greenplum 主档和节点上安装 PEP 服务器所需执行的步骤，它还包含应该如何导入 Greenplum 必要的 UDF（用户自定义函数，即内嵌入数据库的小节点例程）以操作数据并与 PEP 应用程序通信的相关说明。

为了验证安装是否成功，请查看 Protegrity 功能是否已创建。此外，请检查 `pty_whoami` 功能是否返回了执行该功能的用户 ID。

如果这些功能都正常则说明你已成功安装，并且已经准备好开始隐蔽部分数据。

4. 使用 Protegrity 数据保护器加密静态数据

现在我们来讨论一个数据库保护器所提供的功能的例子。在本例中我们制作了一个简单的表格，用 ID 识别行、SSN 识别个人且包含个人提供的评分及其评分的日期。

在用户（`gpuser`）希望访问包括加密的 SSN 的完整数据的时候，他们可以使用视图

`v_sample_ssn`，这是 `sample_ssn_parts` 表的一个简单视图，包含应用于 SSN 字段的功能。该功能使用 `gpuser` 身份向 ESA 发出请求以访问策略并获得密匙。它应用于策略和数据密匙上，并将解密适当的数据以便用户进行查看。

此外还应用了 Protegrity ESA 内的其他规则，可确保向任何尝试通过查看功能查看数据的其他用户隐藏数据。在恶意管理员访问数据并试图获取社会安全号码的时候它将仅显示最后 4 位数，隐藏并保护剩余的敏感数据。

Protegrity 可以使用各种加密算法，但此种数据标记对数据科学家而言尤其适用。不断对数据启用常用加密将增加额外的处理费用，且常用加密通常会导致在解密之前无法使用数据。与此相反，标记可以将数据转变为一种形式，即隐藏其实际数值，与此同时数据仍可为分析算法所用。

5. 利用 Protegrity 数据保护器标记化实施 SQL

一旦 ESA 安装完毕 PEP 流程就将运行，且 Protegrity 功能已安装，下一步就是与 Protegrity 管理员一起设置数据保护策略。管理员必须在 Protegrity 安全管理器中为将使用信用证标记的 SSN 元素创建一个策略，允许 gpuser 解除数据的标记，并让所有其他用户能够查看数据的隐藏版本。此后，需要将该策略配置下推到 PEP 代理，该代理作为在数据库内执行的功能在 Greenplum 集群内的每个服务器上运行。所有这些都将在 Greenplum 的外部进行管理以便很好地完成职责划分。

一旦完成此设置我们就已准备好在数据库内执行我们需要完成的任务。首先，我们需要一个表格以保存数据：

```
CREATETABLE sample_ssn_parts (  
  id INT PRIMARYKEY,  
  ssn VARCHAR,  
  rating INT,  
  rating_date DATE )  
DISTRIBUTED BY (id)
```

它不能保存磁盘上的原始数据，需要通过 Protegrity 提供的功能访问进入 SSN 字段的数据以对数据进行标记化，并在提取数据的时候解除标记。如果我们希望简单地将标记过的数据放入到数据库中，那么我们将使用：

```
INSERT INTO sample_ssn_parts ( id, ssn, rating, rating_date )  
VALUES ( 1,pty_varcharins('123-45-6789,'ssn'), 2, '2016-04-01');
```

此操作将调用标记化功能，然后联系本地 PEP 服务器，以确定哪些标记了的数值可以放入到实际的表格中。

如果你尝试使用明码检索该数据，那么 SSN 将显示为完全不同的号码，即其标记化的数值。

为了检索原始形式的数据需要对该数据启用 Protegrity 功能。

```
SELECT id, pty_varcharsel(ssn,'ssn') as ssn, rating, rating_date  
FROM sample_ssn_parts  
WHERE id = 1
```

在选定数据后，应当利用 PEP 流程检查功能，查看用户是否可以访问数据，如果可以的话应显示哪种格式。在此情况下，gpuser 执行了此查询并显示了原始插入的数值。SSN 数值对于任何其他用户来说都只会看到 ###-##-6789 这样的显示内容，这依据于 ESA 上创建且下推到本地 PEP 流程的策略。

当你每次输入查询码的时候，添加这些功能会有点麻烦，而且你可能会希望限制用户对基础数据表的访问。如果创建一个视图自动应用功能的话就方便得多。

```
CREATE VIEW v_sample_ssn_parts AS  
SELECT id,pty_varcharsel(ssn,'ssn') as ssn, rating, rating_date  
FROM sample_ssn_parts
```

现在任何人都可以输入如下的查询信息：

```
SELECT id, ssn, rating, rating_date
FROM v_sample_ssn_parts WHERE id = 1
```

自动应用该功能，而且大多数用户都会访问此视图。接下来，我们需要能够更轻松地进行插入、更新和删除数据。最理想的是应用的功能尽可能实现透明化，通过对表格创建 INSERT、UPDATE 和 DELETE（插入、更新和删除）可以实现这点。

```
CREATE OR REPLACE RULE insert_sample_ssn AS ON
INSERT TO v_sample_ssn_parts
DO INSTEAD
INSERT INTO sample_ssn_parts ( id, ssn, rating, rating_date )
VALUES ( NEW.id,pty_varcharins(NEW.ssn,'ssn'), NEW.rating,
NEW.rating_date);
```

```
CREATE OR REPLACE RULE update_sample_ssn AS ON
UPDATE TO v_sample_ssn_parts
DO INSTEAD
UPDATE sample_ssn_parts
SET ssn= pty_varcharins(NEW.ssn,'ssn'),
    rating = NEW.rating,
    rating_date = NEW.rating_date
WHERE id = NEW.id;
CREATE OR REPLACE RULE delete_sample_ssn AS ON
DELETE TO v_sample_ssn_parts
DO INSTEAD
DELETE FROM sample_ssn_parts
WHERE id = OLD.id
```

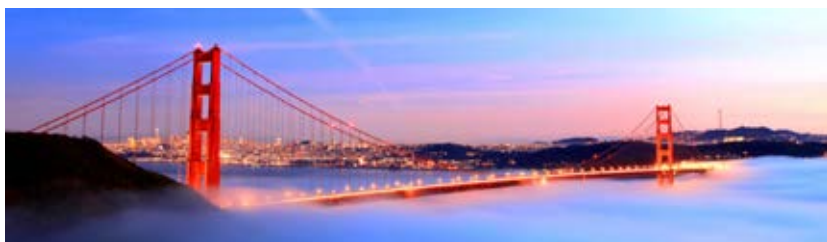
应注意的一点是：为了让这些规则有效表格中的元组为唯一标识，规则可以捕获完成的工作，但它需要返回，并需要对会被原始查询影响的特定记录应用设定的逻辑。

SSN 的所有数据都将以其标记化的格式保存在实际的 `sample_ssn` 表中，同时，用户可以访问 `v_sample_ssn` 并将其视为正常的表格对待。如果数据科学家希望对此数据运行集群算法而不是通过视图传递，那么他们将访问 `sample_ssn` 中的原始数据以避免日常的费用，但仍不会访问实际的 SSN 编号。

正如你所见，Protegrity 的数据库保护器可以提供一种绝佳的方法标记并加密静态数据，管理它们在数据库外的系统中的权限，并以无需解密即可使用的方式模糊数据，是不是相当方便呢？

公司简介

Pivotal 成立于 2013 年 4 月，由 EMC、VMware 和 GE 共同投资成立。公司总部位于美国硅谷，专注于下一代企业级云计算与大数据基础平台，以及下一代应用程序运行框架支撑实现，在敏捷与快速应用程序开发、数据科学、云计算、开放源代码软件、大规模并行处理和实时数据系统领域颇有建树。2016 年 5 月，又获得了来自福特和微软的共同投资，目前公司整体估值达到 28 亿美金。



作为世界上新一代企业级“平台即服务 (PaaS)”，Pivotal Cloud Foundry 结合 Pivotal Big Data Suite 以及 Pivotal Labs 的敏捷开发实验室，使得构建创新型企业成为现实。Pivotal 在世界各地各社区拥有数以百万计的开发人员，每天都有数十亿用户触及 Pivotal 的技术。在塑造了硅谷最有价值的公司软件开发文化十多年之后，如今 Pivotal 引领全球技术浪潮，改变着世界上软件的构建方式，也推动了许多最受人们喜爱的世界品牌的软件创新。全球财富 500 强中，超过三分之二的企业都是 Pivotal 的用户。



平台 就在你手边

Pivotal 正开拓云计算视野，加快产品上市速度，并促使应用持续可用。



数据的商业价值

利用 Pivotal 的端到终端的数据解决方案套件推动决策过程，存储你的大数据，实时分析，并构建合适的应用程序。



构建产品的更好方式

利用 Pivotal 的世界级团队、开源软件和敏捷开发方法，改变用户体验，交付经过市场检验的创新产品。

Pivotal®

颠覆世界软件开发方式

融合领先的敏捷开发服务
云原生平台及开源大数据产品套件
加速客户的创新周期

pivotal.io/cn



关注Pivotal官方微信码: pivotal_china



关注Pivotal官方微博