第 5 章

GaussDB 架构

GaussDB 是华为公司数据库产品品牌名。华为公司从开始自研数据库至今已经有近 20 年历史，其中经历了早期发展、GaussDB 的诞生和发展、数据库产业化三个阶段。本章简明介绍华为公司自研数据库的历程，并给出一些 GaussDB 的里程碑时间点。GaussDB 的发展历史是中国数据库发展历程的典型案例。

GaussDB 以云服务形式提供商业版本，并将在 2020 年中期推出开源数据库产品 openGauss(社区网址为 https://opengauss.org)。

5.1 GaussDB 发展历史

本节首先概要介绍华为自研数据库的早期发展历史及 GaussDB 的诞生和发展，然后介绍华为高斯数据库三个系列产品：GMDB 内存数据库、GaussDB 100 OLTP 数据库和 GaussDB 200 OLAP 数据库的发展历史。

5.1.1 概述

华为公司研究数据库是从满足生产实践出发，从研发用于满足局限场景的较简单架构数据库产品开始，逐步向通用性、可规模商用的数据库产品演进，到 2019 年终于正式发布面向企业客户场景的通用分布式数据库产品，其发展历史如图 5-1 所示。

1. 华为自研数据库的早期发展阶段

华为公司研究和开发数据库技术及产品，最早可追溯到 2001 年。当时，华为公司中央研究院 Dopra 团队为了支撑华为所生产的电信产品(交换机、路由器等)，启动了

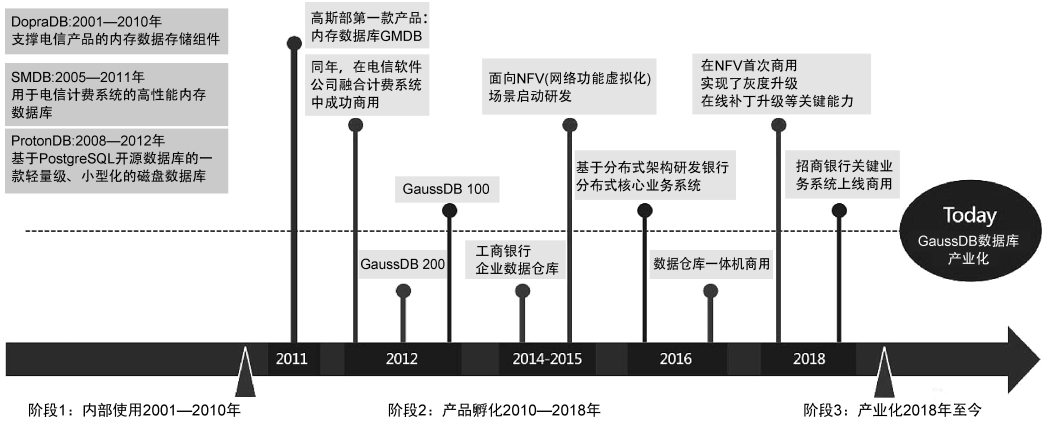


图 5-1　GaussDB 发展历程图

内存数据存储组件 DopraDB 的研发，从此开启了华为自研数据库的历程。DopraDB 后来随着业务和组织的切换，成为华为高斯数据库团队的 GMDB V1 系列产品。

2005 年，华为的通信产品需要一个以内存处理为中心的数据库，评估了当时最高性能的内存数据库软件，发现其性能和特性无法满足业务诉求，便启动了 SMDB (Simple Memory DataBase)的开发。

2008 年，华为核心网产品线需要在产品中使用一款轻量级、小型化的磁盘数据库，于是华为基于 PostgreSQL 开源数据库开发 ProtonDB，这是华为与开源数据库 PostgreSQL 数据库的第一次亲密接触。

2. GaussDB 的诞生和发展阶段

2011 年“数字洪水”即将到来，华为铸造“方舟”应对，组建了 2012 实验室。华为公司认为在数字洪水时代，ICT(Information and Communications Technology，信息和通信技术)软件技术栈中数据库是不可缺少的关键技术，因此将原来分散在各个产品线的数据库团队及业务重新组合，在 2012 实验室中央软件院下成立了高斯部，负责华为公司数据库产品和技术的研发。高斯部得名于纪念大数学家高斯(Gauss)。

高斯部的数据库产品研发历史按照场景和产品特点可分为三个系列。

- GMDB(内存数据库)；
- GaussDB 100 OLTP 数据库；
- GaussDB 200 OLAP 数据库。

3. 数据库产业化阶段

随着华为在 2019 年对业界正式发布高斯数据库，华为自研数据库进入了第三阶段，即数据库产业化阶段。华为高斯数据库后续的规划主要围绕如下方面展开。

1）数据库生态

作为一款通用性、规模商用的数据库产品，生态是重中之重，华为将围绕两个方向来解决数据库生态问题。

（1）技术上采取“云化＋自动化”方案。通过数据库运行基础设施的云化将 DBA（数据库管理员）和运维人员的日常工作自动化，解决如补丁、升级、故障检测及修复等工作带来的开销。传统数据库随着业务负载变化越跑越慢的问题，依赖 DBA 监控和优化来解决。而通过在数据库内部引入 AI 算法，实现免 DBA 自动数据优化，将进一步降低对人工的依赖。

（2）商业上开展与数据库周边生态伙伴的对接与认证，解决开发者/DBA 数据难获取、应用难对接等生态难题，减少企业客户使用华为高斯数据库面临的后顾之忧。

数据库产业生态全景如图 5-2 所示。

图 5-2　数据库产业生态全景

2）技术竞争力

数据库作为“软件皇冠上的明珠”，其技术含量十分高，因此要想在市场上击败竞争对手，必须持之以恒地在关键技术上进行大规模投资。华为高斯数据库将在如下方

向构筑竞争力。

(1) 分布式。构筑世界领先的分布式事务能力和跨 DC(Data Center,数据中心)高可用能力,解决传统关系数据库的扩展性、可用性不足等瓶颈。

(2) 云化架构。未来 10 年云数据库将成为市场主流,华为高斯数据库需要构筑满足公有云、私有云和混合云场景的云化架构,满足各种企业场景的云数据库诉求。

(3) 混合负载。过去由于数据库性能不足,架构缺乏隔离性,一个数据库实例难以在满足 SLA(Service Level Agreement,服务水平协议)前提下,同时支撑不同业务负载(交易型、分析型)的运行。随着硬件性能的提升和新数据架构理论的创新,在一套数据库中运行多种负载已经成为行业趋势,这不但简化了系统部署、消除了数据复制或搬迁带来的数据一致性问题,同时也提升了系统的可靠性和实时性。

(4) 多模异构。传统数据库围绕关系数据进行管理,随着移动互联网、IoT(Internet of Things,物联网)、人工智能的普及应用,新类型数据(时序、图、图像等)成为接下来十年数据库系统主要的管理类型,这需要支持多模数据管理的新型数据库。通用处理器随着晶体管制程逐步走到极限,而异构加速器(FPGA/GPU/NPU 等)大放异彩,在 AI(人工智能)等场景大量使用,如何通过改造优化数据库架构,实现充分利用“通用处理器+异构加速器”算力优势,是高斯数据库重点发展方向之一。

(5) AI+DB。2010 年起随着大数据量和大计算量的普及,AI 算法精度和适用范围足以支撑在特定场景(如数据库参数调优、SQL 执行优化等)下解决问题;另一方面,随着深度神经网络的普及化,对过去无法有效处理的图像、语音、文本等非结构化数据,已经能很好地从中抽取结构化信息,如何将其用在数据库中解决非结构化数据的高效管理也成为当前研究的热点。

5.1.2 GMDB 内存数据库历史

2012 年,华为高斯部成立后,结合电信软件公司在 SMDB 长期使用中面临的“开发效率低、数据一致性弱”等关键痛点,立项开发了高斯部成立后的第一款产品:GMDB V2 系列。GMDB V2 与 GMDB V1 最大差别在于,它是一款支持 SQL/关系模型和 ACID 能力的全功能内存数据库。GMDB V2 最终于 2012 年起在融合计费系统中成功商用,到 2018 年,基于 GMDB V2 内存数据库产品的融合计费系统所支撑的用户数超 20 亿。

2016 年起，华为高斯部面向核心网产品线 NFV（Network Function Virtualization，网络功能虚拟化）场景，启动分布式内存数据库产品 GMDB V3 系列的研发。2018 年 GMDB 在 NFV 首次商用，并在电信行业的 NFV 场景第一个实现了灰度升级（意指不停止业务实现服务在线升级）、在线补丁升级等关键能力。

5.1.3　GaussDB 100 OLTP 数据库历史

2012 年起，华为高斯部启动了 GaussDB 100 的研究工作。GaussDB 100 早期版本 V1 系列是基于 PostgreSQL V8 发展而来的，主要是面向华为公司内各产品线在操作管理类系统中所使用的 OLTP 类型磁盘数据库场景。该系列产品在华为公司大量商用。

随着互联网、移动互联网业务的兴起，网络数据量和业务量均呈现爆炸式增长，传统集中式数据库已经无法满足大容量、高扩展的诉求。2016 年起，华为高斯部启动分布式 OLTP 数据库的研发工作，分布式 OLTP 数据库具备分布式事务强一致、高性能、高扩展、高可用等特点，可以满足金融、电信、能源等主流行业核心业务系统的要求。目前 GaussDB 分布式 OLTP 数据库已针对金融、政府等高端客户商用上线。

5.1.4　GaussDB 200 OLAP 数据库历史

2012 年，华为高斯部启动了 PteroDB（羽龙）项目，孵化面向企业数据仓库场景的 MPP 架构 OLAP 数据库。2014 年华为公司成功击败竞争对手进入工商银行总行下一代 EDW（Enterprise Data Warehouse，企业数据仓库）联合创新项目。经过工商银行 2 年孵化，GaussDB 200 于 2016 年开始进入商用，逐步替换了友商数据仓库一体机产品。2019 年一季度，工商银行总行最后一台友商数据仓库一体机下线、业务负载全面由 GaussDB 200 承载。

2019 年 5 月 15 日，华为公司正式向业界宣布 GaussDB 品牌，揭开了 GaussDB 产业化的帷幕。

华为高斯部除了数据库产品的研发之外，也将部分技术研究成果发表在 VLDB（International Conference on Very Large Data Bases）、SIGMOD（The ACM Special

Interest Group on Management of Data)、ICDE(International Conference on Data Engineering)等数据库顶级会议中。

华为高斯数据库团队在数据库领域顶级学术会议中所发表的论文(非全集)如图 5-3 所示(VLDB 论文)、图 5-4 所示(SIGMOD 论文)。

FusionInsight LibrA: Huawei's Enterprise Cloud Data Analytics Platform

Le Cai; Jianjun Chen; Jun Chen, Yu Chen, Kuorong Chiang, Marko Dimitrijevic, Yonghua Ding
Yu Dong; Ahmad Ghazal, Jacques Hebert, Kamini Jagtiani, Suzhen Lin, Ye Liu, Demai Ni·
Chunfeng Pei, Jason Sun, Yongyan Wang; Li Zhang; Mingyi Zhang, Cheng Zhu
Huawei America Research

ABSTRACT

Huawei FusionInsight LibrA (FI-MPPDB) is a petabyte scale enterprise analytics platform developed by the Huawei database group. It started as a prototype more than five years ago, and is now ... over the globe, i... cial institutions. ... have been mainl... fast evolving Chi...

This paper de... some of its majo... top four requirem... alytics on the clo... over heterogeneo... to utilize powerfu... We present our l... cluding online e... SQL on HDFS, ... nally, we present ... the effectiveness ...

PVLDB Referen... Le Cai, Jianjun C... Marko Dimitrijevi... Jacques Hebert, K... Chunfeng Pei, Jun... Cheng Zhu. Fus... Data Analytics Pl... DOI: https://doi.o...

1. INTROD...

Huawei Fusion... scale enterprise d... Previous version...

*The authors, L... and Li Zhang, ... this work was d...
†Dr. Jianjun ... jun.chen1@huaw...

This work is lice... NonCommercial-No... of this license, visit h... any use beyond those... info@vldb.org. ... Proceedings of the V... Copyright 2018 VLE... DOI: https://doi.org/...

was launched in 2015 and has been adopted by many customers over the globe, including some of the world's largest financial institutes in China. With the help of the success of Huawei FusionInsight MPPDB, FusionInsight products started appearing in Gartner magic quadrant from 2016.

Hassium: Hardware Assisted Database Synchronization

Hillel Avni
Huawei Technologies
hillel.avni@huawei.com

Aharon Avitzur
Huawei Technologies
aharon.avitzur@huawei.com

PHyTM: Persistent Hybrid Transactional Memory

Hillel Avni
Huawei Technologies
European Research Institute
hillel.avni@huawei.com

Trevor Brown
University of Toronto
tabrown@cs.toronto.edu

ABSTRACT

We present Hassiu... rency control, whi... HTM (Hardware T... for overreaction to ... combine the benefi... control (OCC) wit... 2PL (Two-Phase Lo... capable multi-core ... 2X throughput hike ... and guaranties and ... under increasing n... 2PL methods fail t...

1. INTRODU...

Multi-core in-mer... icant performance ... proposals often fail ... and they suffer fro... [18].

This paper pres... scheme which synt... rency control and ... progress guarantees ... the ability to maint... of threads (level of ... sium works around ... then continue to d... superior under cont...

1.1 Using HT...

The first obstacl... method for in-mer... limitation. All HT... associativity, this s... transactions will al... capacity aborts).

ABSTRACT

Processors with hardware support for transactional memory (HTM) are rapidly becoming commonplace, and processor manufacturers are currently working on implementing support for upcoming non-volatile memory (NVM) technologies. The combination of HTM and NVM promises to be a natural choice for in-memory database synchronization. However, limitations on the size of hardware transactions and the lack of progress guarantees by modern HTM implementations prevent some applications from obtaining the full benefit of hardware transactional memory. In this paper, we propose a *persistent hybrid TM* algorithm called PHyTM for systems that support NVM and HTM. PHyTM allows hardware assisted ACID transactions to execute concurrently with pure software transactions, which allows applications to gain the benefit of persistent HTM while simultaneously accommodating unbounded transactions (with a high degree of concurrency). Experimental simulations demonstrate that PHyTM is fast and scalable for realistic workloads.

1. INTRODUCTION

Non-volatile memory (NVM) is an upcoming technology that promises to revolutionize computer memory. It is not currently commercially available, but manufacturers have developed prototypes, and have released performance information about these prototypes to the public. NVM is expected to become cheaper, faster and more power efficient than DRAM, and will likely become ubiquitous.

Researchers have just begun to understand how machines with NVM should be programmed. The programming model for NVM is still in flux, but the following model is currently leading. Systems can contain only NVM, or a combination of DRAM and NVM. Data is asynchronously flushed to NVM at any time, and without the programmer's knowledge. A programmer can also explicitly cause data to be flushed to NVM by invoking a primitive called *Flush*. Another primitive called a *persistence barrier* is provided to allow a process to block until data has been flushed to NVM.

There is significant controversy over whether processor cache and registers will be volatile or non-volatile. Some researchers are investigating ways to provide enough residual power to flush this data to NVM in the event of a power failure [20]. This approach could allow applications to avoid any runtime overhead associated with providing persistence (since the entire processor state would be persistent). However, hardware designers are skeptical about its feasibility, citing concerns about the amount of energy necessary to flush processor cache (since the cache is very large, and processors are complex and power-hungry). They suggest that future hardware will only use residual energy to flush data in volatile buffers on NVM-controllers [14]. Operating under this assumption, Intel is currently designing new and efficient flush instructions (CLWB and CLFLUSHOPT) with NVM in mind [13]. These new instructions assume volatile caches, and allow the flushed data to stay in cache, to avoid cache misses. They also accelerate flushes to NVM.

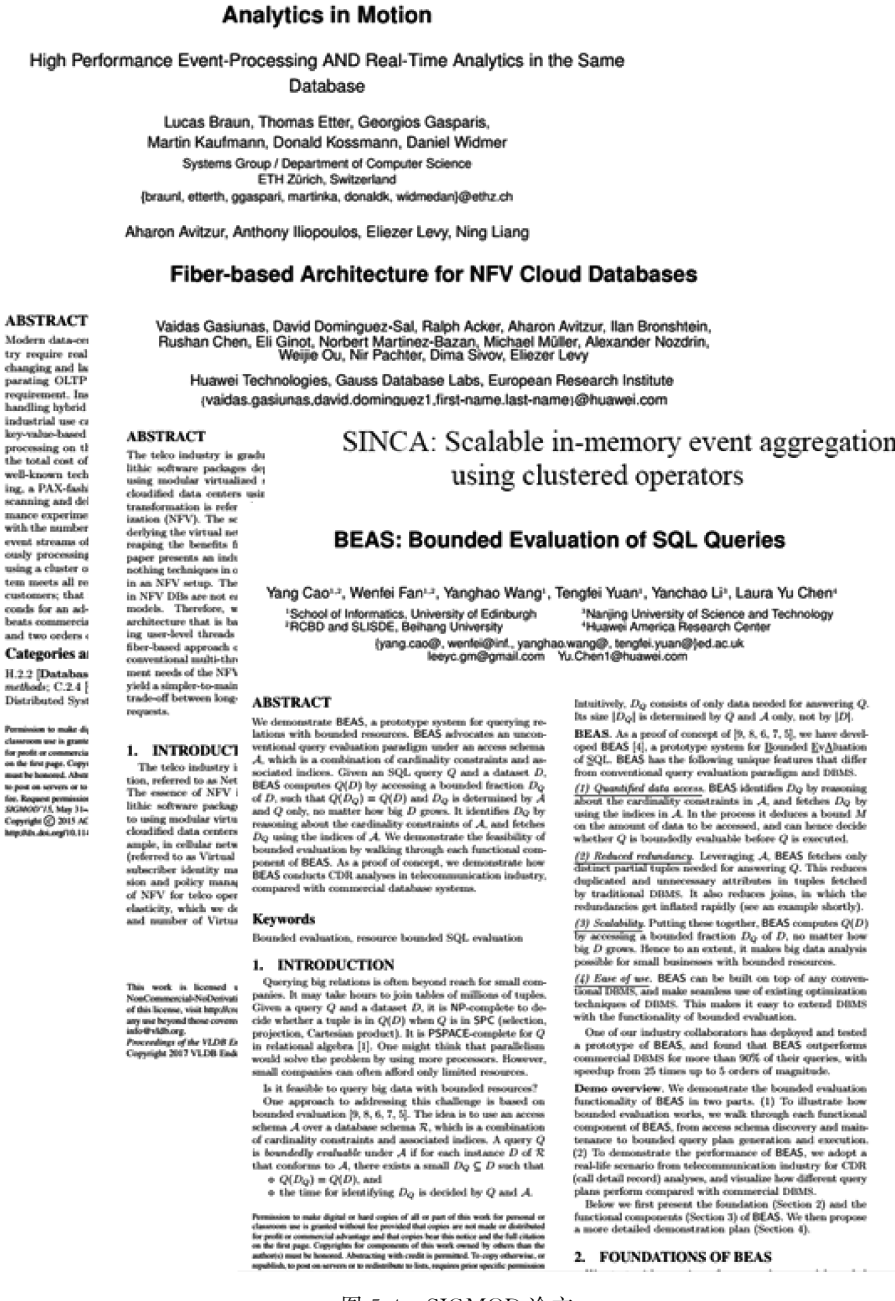
We consider a system in which the processor cache and registers are volatile. In such a system, the key challenge is to ensure that NVM is always left in a consistent state if a power failure occurs and the cache and registers are cleared.

Another recent technology called hardware transactional memory (HTM), which brings database-style transactions to shared memory, was recently implemented in Intel processors. (HTM has also been implemented in production systems by IBM, and in various research systems. We focus on Intel's implementation.) HTM allows programmers to execute arbitrary blocks of code in transactions, which either commit and appear to occur atomically, or abort and have no effect on the contents of memory. Intel's implementation of HTM is best effort, which means that no transaction is ever guaranteed to commit. Thus, a non-transactional fallback path must be provided by a programmer to be executed if a transaction aborts sufficiently many times. The simplest fallback path one can imagine simply reexecutes the body of a transaction after taking a global lock (that prevents other processes from performing transactions). However, this naive approach does not work with NVM.

The interplay between HTM implementations and NVM proposals is particularly interesting, because transactions must appear to be atomic, but writes performed by the fallback path can trickle asynchronously to NVM at any time. Therefore, the fallback path must be carefully designed to avoid exposing partial effects of an in-flight transaction to other processes in the event of a power failure. An additional

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-nd/4.0/. For

图 5-3　VLDB 论文



Analytics in Motion

High Performance Event-Processing AND Real-Time Analytics in the Same Database

Lucas Braun, Thomas Etter, Georgios Gasparis, Martin Kaufmann, Donald Kossmann, Daniel Widmer
Systems Group / Department of Computer Science
ETH Zürich, Switzerland
{braunl, etterth, ggaspari, martinka, donaldk, widmedan}@ethz.ch

Aharon Avitzur, Anthony Iliopoulos, Eliezer Levy, Ning Liang

Fiber-based Architecture for NFV Cloud Databases

ABSTRACT

Modern data-ce
try require real
changing and la
parating OLTP
requirement. In
handling hybrid
industrial use-c
key-value-based
processing on th
the total cost of
well-known tech
ing, a PAX-flash
scanning and de
mance experime
with the number
event streams of
ously processing
using a cluster o
tem meets all re
customers; that
conds for an ad-
beats commercia
and two orders

Categories a

H.2.2 [Databas
methods; C.2.4 [
Distributed Syst

Permission to make di
classroom use is grante
for profit or commercia
on the first page. Copyr
must be honored. Abstr
to post on servers or to
fee. Request permissio
SIGMOD'15, May 31–
Copyright © 2015 AC
http://dx.doi.org/10.11

Vaidas Gasiunas, David Dominguez-Sal, Ralph Acker, Aharon Avitzur, Ilan Bronshtein, Rushan Chen, Eli Ginot, Norbert Martinez-Bazan, Michael Müller, Alexander Nozdrin, Weijie Ou, Nir Pachter, Dima Sivov, Eliezer Levy

Huawei Technologies, Gauss Database Labs, European Research Institute
{vaidas.gasiunas,david.dominguez1,first-name.last-name}@huawei.com

ABSTRACT

The telco industry is gradu
lithic software packages de
using modular virtualized
cloudified data centers usi
transformation is refer
ization (NFV). The s
derlying the virtual ne
reaping the benefits fi
paper presents an indu
nothing techniques in o
in an NFV setup. The
in NFV DBs are not e
models. Therefore, w
architecture that is ba
ing user-level threads
fiber-based approach
conventional multi-thr
ment needs of the NFV
yield a simpler-to-main
trade-off between long-
requests.

1. INTRODUCT

The telco industry i
tion, referred to as Net
The essence of NFV
lithic software packag
to using modular virtu
cloudified data centers
ample, in cellular netw
(referred to as Virtual
subscriber identity ma
sion and policy manag
of NFV for telco oper
elasticity, which we de
and number of Virtua

This work is licensed
NonCommercial-NoDerivati
of this license, visit http://cr
any use beyond those covere
info@vldb.org.
Proceedings of the VLDB En
Copyright 2017 VLDB Endo

SINCA: Scalable in-memory event aggregation using clustered operators

BEAS: Bounded Evaluation of SQL Queries

Yang Cao[1,2], Wenfei Fan[1,2], Yanghao Wang[1], Tengfei Yuan[3], Yanchao Li[3], Laura Yu Chen[4]
[1]School of Informatics, University of Edinburgh
[2]RCBD and SLISDE, Beihang University
[3]Nanjing University of Science and Technology
[4]Huawei America Research Center
{yang.cao@, wenfei@inf., yanghao.wang@, tengfei.yuan@}ed.ac.uk
leeyc.gm@gmail.com Yu.Chen1@huawei.com

ABSTRACT

We demonstrate BEAS, a prototype system for querying relations with bounded resources. BEAS advocates an unconventional query evaluation paradigm under an access schema $\mathcal{A}$, which is a combination of cardinality constraints and associated indices. Given an SQL query Q and a dataset D, BEAS computes $Q(D)$ by accessing a bounded fraction D_Q of D, such that $Q(D_Q) = Q(D)$ and D_Q is determined by $\mathcal{A}$ and Q only, no matter how big D grows. It identifies D_Q by reasoning about the cardinality constraints of $\mathcal{A}$, and fetches D_Q using the indices of $\mathcal{A}$. We demonstrate the feasibility of bounded evaluation by walking through each functional component of BEAS. As a proof of concept, we demonstrate how BEAS conducts CDR analyses in telecommunication industry, compared with commercial database systems.

Keywords

Bounded evaluation, resource bounded SQL evaluation

1. INTRODUCTION

Querying big relations is often beyond reach for small companies. It may take hours to join tables of millions of tuples. Given a query Q and a dataset D, it is NP-complete to decide whether a tuple is in $Q(D)$ when Q is in SPC (selection, projection, Cartesian product). It is PSPACE-complete for Q in relational algebra [1]. One might think that parallelism would solve the problem by using more processors. However, small companies can often afford only limited resources.

Is it feasible to query big data with bounded resources?

One approach to addressing this challenge is based on bounded evaluation [9, 8, 6, 7, 5]. The idea is to use an access schema $\mathcal{A}$ over a database schema $\mathcal{R}$, which is a combination of cardinality constraints and associated indices. A query Q is *boundedly evaluable* under $\mathcal{A}$ if for each instance D of $\mathcal{R}$ that conforms to $\mathcal{A}$, there exists a small $D_Q \subseteq D$ such that

- $Q(D_Q) = Q(D)$, and
- the time for identifying D_Q is decided by Q and $\mathcal{A}$.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission

Intuitively, D_Q consists of only data needed for answering Q. Its size $|D_Q|$ is determined by Q and $\mathcal{A}$ only, not by $|D|$.

BEAS. As a proof of concept of [9, 8, 6, 7, 5], we have developed BEAS [4], a prototype system for Bounded EvAluation of SQL. BEAS has the following unique features that differ from conventional query evaluation paradigm and DBMS.

(1) Quantified data access. BEAS identifies D_Q by reasoning about the cardinality constraints in $\mathcal{A}$, and fetches D_Q by using the indices in $\mathcal{A}$. In the process it deduces a bound M on the amount of data to be accessed, and can hence decide whether Q is boundedly evaluable before Q is executed.

(2) Reduced redundancy. Leveraging $\mathcal{A}$, BEAS fetches only distinct partial tuples needed for answering Q. This reduces duplicated and unnecessary attributes in tuples fetched by traditional DBMS. It also reduces joins, in which the redundancies get inflated rapidly (see an example shortly).

(3) Scalability. Putting these together, BEAS computes $Q(D)$ by accessing a bounded fraction D_Q of D, no matter how big D grows. Hence to an extent, it makes big data analysis possible for small businesses with bounded resources.

(4) Ease of use. BEAS can be built on top of any conventional DBMS, and make seamless use of existing optimization techniques of DBMS. This makes it easy to extend DBMS with the functionality of bounded evaluation.

One of our industry collaborators has deployed and tested a prototype of BEAS, and found that BEAS outperforms commercial DBMS for more than 90% of their queries, with speedup from 25 times up to 5 orders of magnitude.

Demo overview. We demonstrate the bounded evaluation functionality of BEAS in two parts. (1) To illustrate how bounded evaluation works, we walk through each functional component of BEAS, from access schema discovery and maintenance to bounded query plan generation and execution. (2) To demonstrate the performance of BEAS, we adopt a real-life scenario from telecommunication industry for CDR (call detail record) analyses, and visualize how different query plans perform compared with commercial DBMS.

Below we first present the foundation (Section 2) and the functional components (Section 3) of BEAS. We then propose a more detailed demonstration plan (Section 4).

2. FOUNDATIONS OF BEAS

图 5-4　SIGMOD 论文

5.2 GaussDB 架构概览

GaussDB 采用了分层解耦、可插拔架构，能够同时支持 OLTP、OLAP 业务场景。

5.2.1 数据库架构变化

数据库架构经历了几个大的变化：单机数据库、集群数据库、云分布式数据库。GaussDB 面向云分布式数据库设计，采用分层解耦、可插拔架构，一套代码，同时支持 OLTP、OLAP 业务场景，如图 5-5 所示。

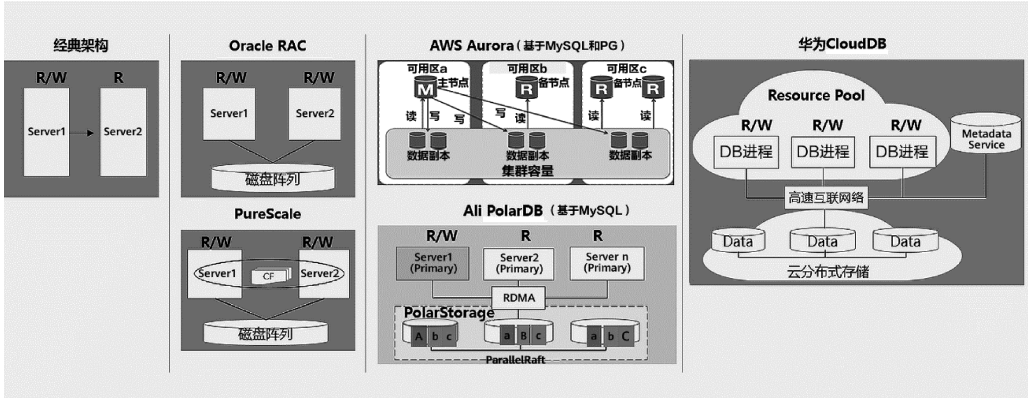


图 5-5 数据库架构变化

5.2.2 GaussDB 关键技术架构

GaussDB 采用分布式关键技术架构，实现一套代码同时支持 OLAP 和 OLTP 业务场景。主要特点如下：

（1）支持 SQL 优化、执行、存储分层解耦架构。

（2）基于 GTM（Global Transaction Management，全局事务控制器）和高精度时钟的分布式 ACID 强一致。

（3）支持存储技术分离，也支持本地架构。

(4) 支持可插拔存储引擎架构。

GaussDB 未来关键技术架构,如图 5-6 所示。

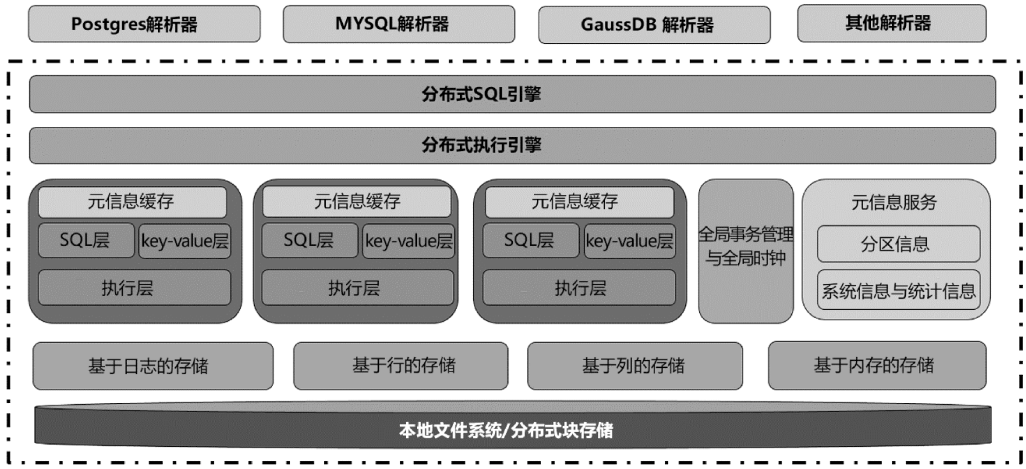


图 5-6　GaussDB 未来关键技术架构

5.3　GaussDB 100 OLTP 数据库架构

OLTP 是传统的关系数据库的主要应用,包括基本的增加、删除、修改、查询事务处理,例如银行交易。

5.3.1　设计思想与目标客户

OLTP 业务场景主要分为两大类:一类是金融银行业务场景,一类是互联网业务场景。但应用 OLTP 业务要满足 5 个重要的需求:

(1) 故障业务中断时间,RTO(Recovery Time Objective,恢复时间目标,指业务停止服务的最长时间)尽可能短,最好是 RTO=0。

(2) 任何故障,数据不错、不丢失,RPO(Recovery Point Objective,数据恢复点目标,指业务系统的数据丢失量)=0。

(3) 并发和性能满足业务诉求。

(4) 易于运维,最好是自动诊断、自动修复。

(5) 易于调优,最好是自调优。

GaussDB OLTP 数据库基于这 5 个关键需求设计,分层解耦:

(1) 采用并行恢复和存储层异步回放机制优化 RTO,目前支持 AZ(Availability Zone,可用区域)内 RTO<10s; AZ 故障,RTO<60s。

(2) 采用多副本 RAFT(一种分布式一致性协议)复制机制保证数据的可靠性,即 RPO=0。

(3) 支持线程池和采用高精度时钟去中心化,支持高并发和线性扩展性,满足高并发和性能的诉求。

(4) 运维能力上基于统计数据分析、推理,实现自运维能力,降低运维门槛。

(5) 采用基于 AI 的自调优参数和 ABO(AI Based Optimization,基于人工智能的查询优化)优化器提供自调优能力,降低调优门槛。

5.3.2 分布式强一致的架构

分布式强一致必须有一个全局的时间戳信息,否则很难保证数据的一致性。GaussDB 100 实现了基于 GTM 的分布式 ACID[原子性(Atomicity)、一致性(Consistency)、隔离性(Isolation)、持久性(Durability)]设计。

GTM 仅处理全局时间戳请求,CSN(Commit Sequence Number,待提交事务的序列号)是一个 64 位递增无符号数,它的递增,几乎都是 CPU++和消息收发操作。

不是每次都写 ETCD(Editable Text Configuration Daemon,分布式键值存储系统,用于共享配置、服务注册和查找),而是采用定期持久化到 ETCD。每次写 ETCD 的 CSN 要加上一个 backup_step(100 万),一旦 GTM 出现故障,CSN 从 ETCD 读取出来的值保证单调递增。当前 GTM 只完成 CSN++,可以支持 200MB/s 请求。GTM 处理获取 CSN 消息和 CSN++的消息,TCP 协议栈消耗 CPU 会非常严重,采用用户态协议栈提高 GTM 单节点的处理能力。GTM 关键数据结构和线程,如图 5-7 所示。

1. 单节点的事务

单节点事务设计,如图 5-8 所示。

关键设计如下:

(1) GTM 只维护一个 CSN++,快照(snapshot)只包含 CSN。

(2) DN(Data Node,数据节点)本地维护事务 ID(唯一标识符),维护 ID 到 CSN

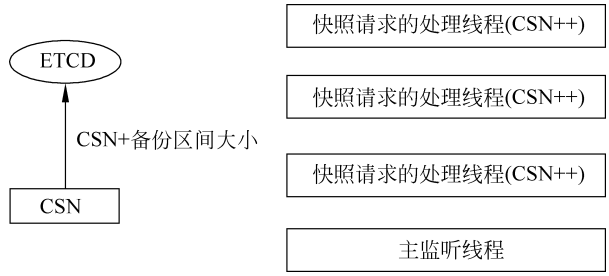


图 5-7　GTM 关键数据结构和线程

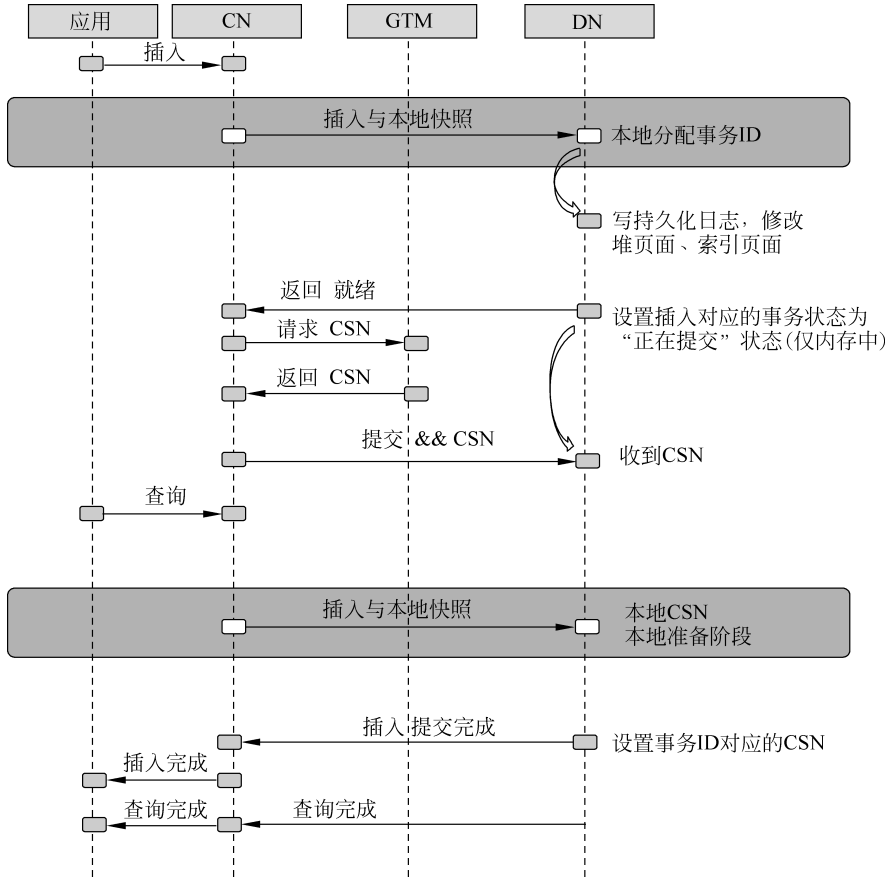


图 5-8　单节点事务设计

的映射(CSN_LOG)。

(3) DN 本地垃圾回收的过程中回填 CSN。

(4) 单分片读事务使用本地快照：

① 获取本地最新的 CSN 和准备阶段事务号；

② 如果 CSN 状态为“提交中”则进行等待；

③ 如果 row.CSN＜ localsnapshot.csn || xid in prepared_xid list 可见，否则不可见。

2. 跨节点事务

跨节点事务设计，如图 5-9 所示。

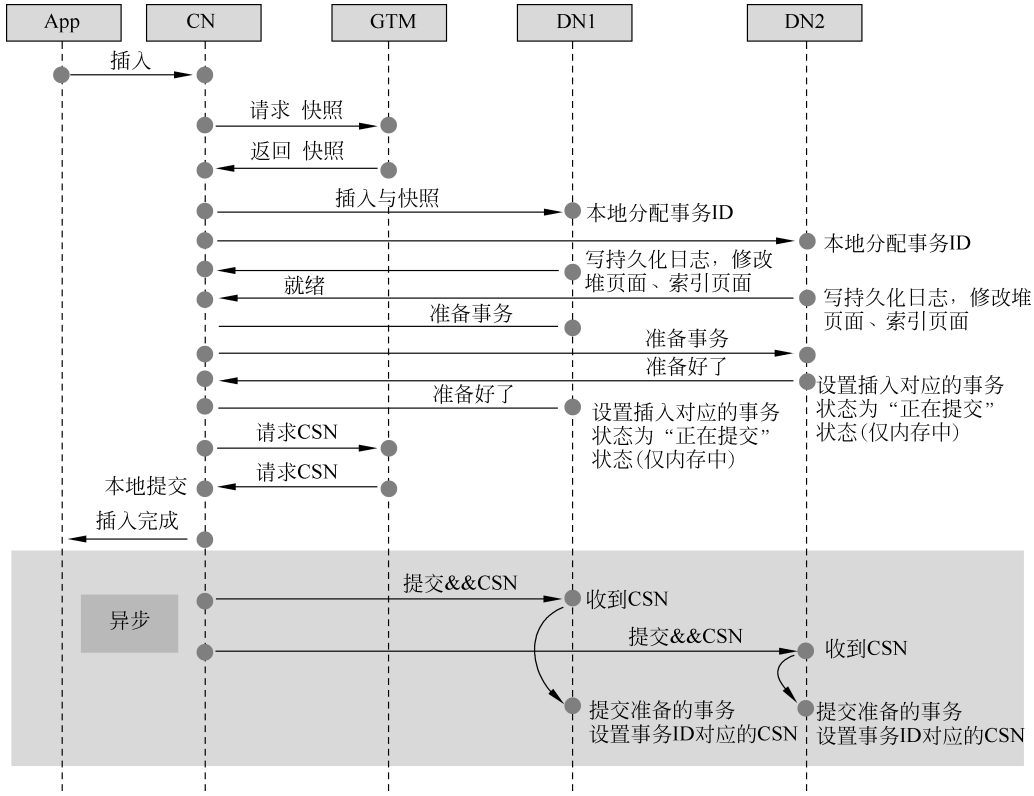


图 5-9　跨节点事务设计

关键设计如下：

(1) 第二阶段事务提交改为异步方式，只同步做两阶段提交的准备阶段。

(2) DN 上行级别可见性判断：

① DN 处于准备状态的事务依赖对应 CN 上的事务是否提交，如果已经提交，且 CSN 比 snapshot.CSN 小，就可见。

② 对 DN 上处于准备状态(prepared)的事务，CN 上的事务不处于提交状态，则必

须判断是否是残留状态，如果是则进行回滚。

```
TRY_AGAIN:
IF row.xact_status is prepared
{
       notify clean_pending_prepared_xact
       IF CN.xact_status is committed && CN.xact.CSN < snapshot.CSN
           return visible
       goto try_again;
}
Else
{
         if(row.xact is committed and row.CSN < snapshot.CSN)
               return visible
         else
               return not_visible
}
```

5.3.3　可插拔存储引擎架构

面向 OLTP 不同的时延要求，需要的存储引擎技术是不同的。例如在银行的风控场景里，对时延的要求是非常苛刻，传统的行存储引擎的时延很难满足业务要求。因此 GaussDB 设计支持了可插拔存储引擎架构，可以同时支持传统行存储引擎和内存引擎。内存引擎采用记录（record）的组织方式，Masstree 无锁化索引设计，提高系统并发能力和降低了事务的时延。行存储引擎可以支持不同的 MVCC（多版本）实现机制，包括 append-only 形式的 MVCC 实现机制和 in-place update 的 MVCC 实现机制。整个数据库中存储引擎、SQL 引擎都是解耦的，可以快速添加（演进）新的存储引擎和 SQL 引擎。

5.4　GaussDB 200 OLAP 数据库架构

OLAP 是数据仓库系统的主要应用，支持复杂的分析操作，侧重决策支持，并且提供直观易懂的查询结果。

5.4.1 设计思想与目标客户

OLAP 数据分析场景有 5 个关键的需求：

(1) 数据量大，从几百万亿字节(TB)到千万亿字节(PB)是很正常的，容量可扩展。

(2) 复杂查询多，计算复杂度高，必须分布式并行计算。

(3) SQL 自动优化能力要强，自动调优诉求强烈。

(4) 数据入库速度要快，入库几百兆字节、万亿字节的数据很常见。

(5) 故障恢复 RTO 尽可能短，数据不丢失，即 RPO=0。

银行数据仓库、安全行业的同行同住分析、证券行业的数据挖掘分析都对集群的规模和并行计算能力有很高的要求。GaussDB OLAP 针对这几个关键需求设计，支持了列存储引擎、自适应压缩，大大降低了存储空间，基于 share nothing 架构，线性扩展，解决了千万亿字节(PB)级数据存储问题。

通过分布式优化器和分布式执行器，构筑了分布式并行技术能力。数据入库采用并行加载技术，大大提高入库效率。

5.4.2 面向数据分析的高效存储和计算架构

GaussDB 200 OLAP 数据库采用列存储引擎提高存储的压缩比和面向列的计算能力，而向量化执行相对于传统的执行模式改变是对于一次一元组的模型修改为一次一批元组，且按照列运算，这种看似简单的修改却带来巨大的性能提升。

(1) 一次一元组模型函数调用次数较大，每一条元组都会根据执行树的形态遍历执行树，面对 OLAP 场景，一次一元组模型巨量的函数调用次数使开销较大，而一次一批元组的执行模式则大大减小遍历执行节点的开销。

(2) 一次一批元组的数据运载方式为某些表达式计算的 SIMD(Single-Instruction, Multiple-Data stream processing，单指令流多数据流)化提供了机会，SIMD 化能带来性能的提升。

(3) 一次一批元组的数据运载方式天然对接列存储，列存储引擎能够很方便地在底层扫描节点装填向量化的列数据。CPU 的指令缓存(cache)和数据缓存的命中率大大提高。

GaussDB OLAP 高效存储和计算架构，如图 5-10 所示。

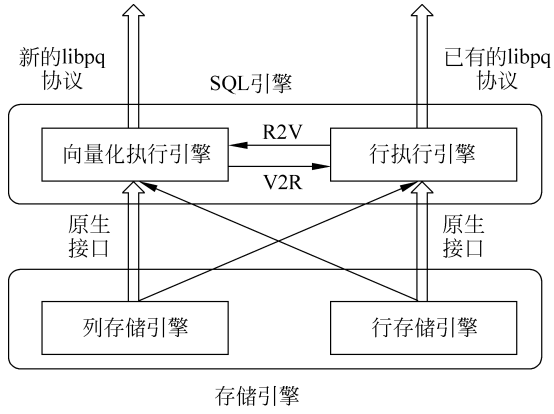


图 5-10　GaussDB OLAP 高效存储和计算架构

5.4.3　分布式并行计算架构

分布式并行计算架构核心实现如下：

(1) 通过分布式优化器，根据代价估算、AI 数据分析，产生最优的分布式执行计划。

好的执行计划和差的执行计划在运行性能上可能会有很大的差距。优化器生成执行计划的过程如图 5-11 所示。

图 5-11　优化器生成执行计划的过程

GaussDB 200 OLAP 数据库优化器支持 CBO(Cost Based Optimization，基于代价的查询优化)和 ABO(基于机器学习的查询优化)，根据代价和 AI 学习，会自动选择

SMP(Symmetric Multi-Processing,对称多处理结构)、Join order、group 算法、index 等,GaussDB OLAP 优化器整体架构如图 5-12 所示。

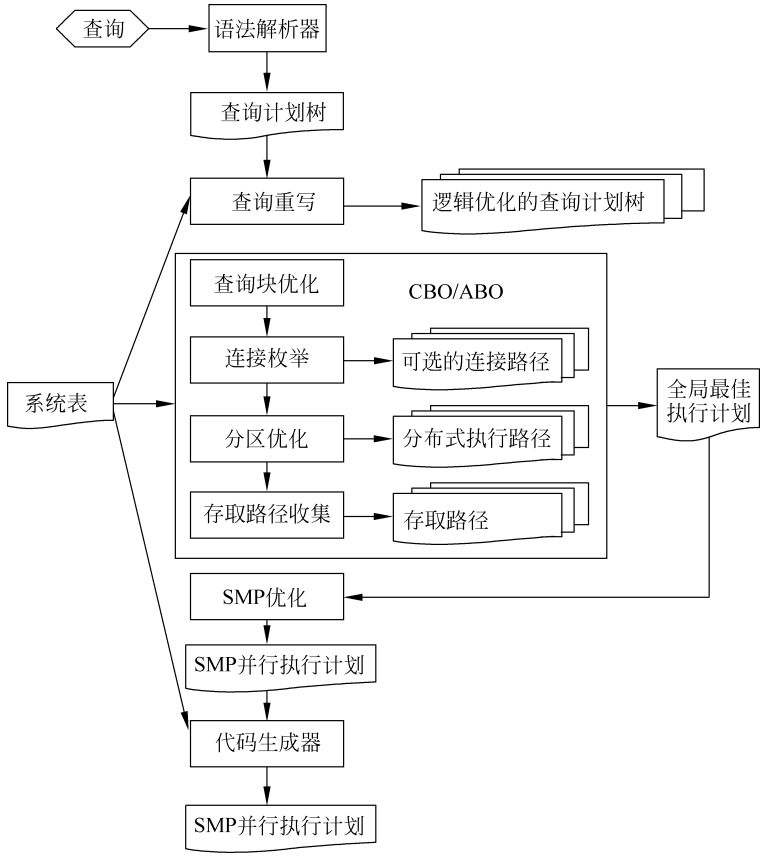


图 5-12　GaussDB OLAP 优化器架构

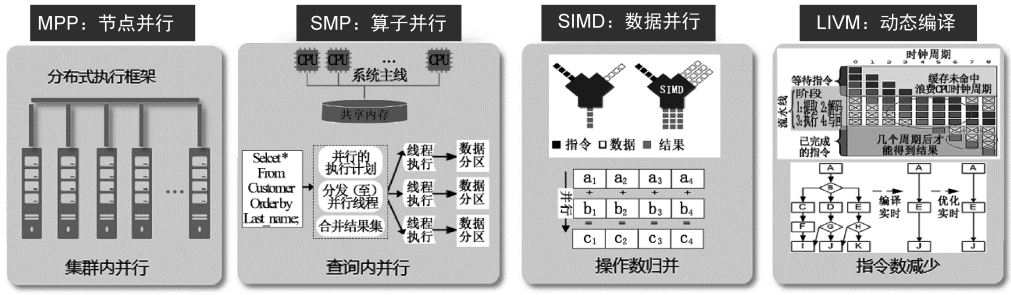
(2) 通过 LLVM(Low Level Virtual Machine,一个编译器框架)编译执行、SIMD 单指令多数据执行、算子并行执行和节点并行执行技术,提高复杂查询的性能。

复杂查询性能提升方式如图 5-13 所示。

5.4.4　并行数据加载

通过并行重分布(Redistribute Streaming)算子技术,让各个 DN 都参与数据导入,充分利用各个设备的计算能力及网络带宽。并行数据加载的关键技术如下:

(1) GDS:数据源服务进程。



核心技术：
- 分布式执行框架，VPP用户态TCP协议，支持>1000 PC Server并行参与计算。

核心技术：
- 多线程并行算法，实现核心算子内并行执行。
- ARM64众核支持，NUMA架构优化。

核心技术：
- SIMD + 向量化引擎，一个指令执行一批数据的操作。
- 支持x86、ARM指令。

核心技术：
- LLVM编译执行，将热点函数预编译成机器码，减少SQL执行指令数，提升性能。

图 5-13　复杂查询性能提升方式

（2）重分布：从 GDS 读取数据，计算哈希（Hash）重新分发数据。

（3）协调节点：根据数据源和数据节点个数，产生并行重分布的计划，把数据源和数据节点分配好。

并行数据加载方式，如图 5-14 所示。

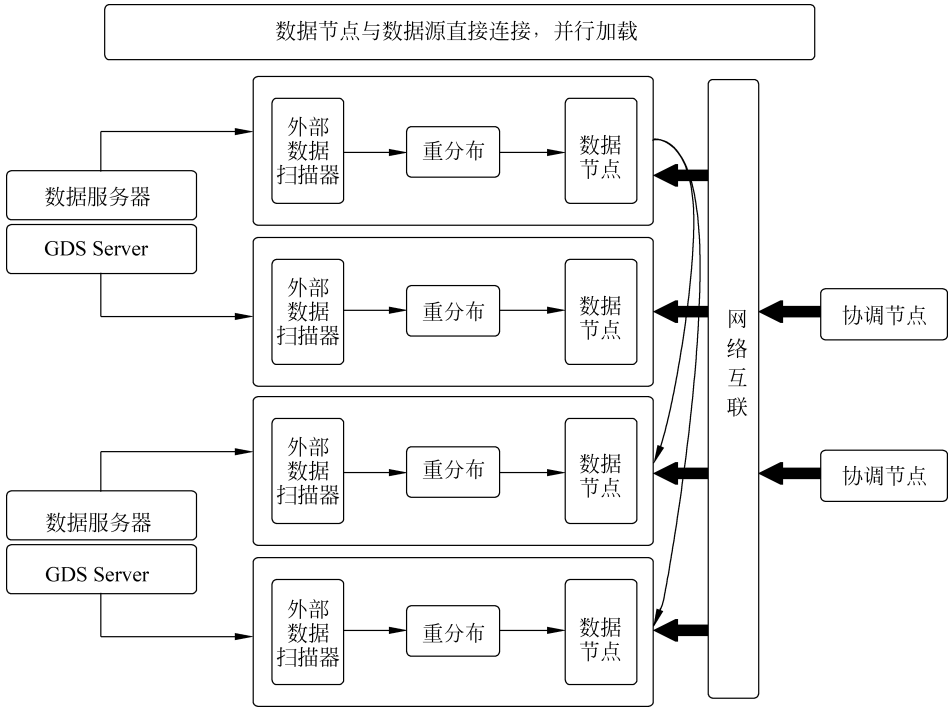


图 5-14　并行数据加载

5.5 GaussDB 云数据库架构

云数据库系统的主要目的是提供数据库系统服务的基础设施，以实现对计算机资源的共享。本节所讲述的 GaussDB 云数据库架构设计的内容，目前处于研发阶段，对应产品尚未向客户发布。

5.5.1 设计思想与目标客户

从数据存放的位置来看，云数据库系统可以分成三大类：

（1）公有云数据库系统服务：该类数据库系统服务主要面向中小型企业的数据库需求。针对中小型企业提供公有云数据库系统服务，可以大幅降低这类公司的运营成本，比如构建数据中心或者机房、构建服务器、运维服务器、运维数据库系统的成本等，同时也使得这类使用公有云数据库系统服务的公司，可以更加专注在业务领域，而无须花费太多的精力在基础设施的构建上。

（2）私有云数据库系统服务：该类数据库服务主要面向大中型企业的数据库服务需求。这类云数据库系统的构建，通常需要在公司内部购买大量的设备，同时构筑相关的 PaaS 层、SaaS 层，其中数据库服务是非常关键的一类服务。该类服务使得公司内部的各个部门的信息新系统可以共享相关资源，同时实现数据共享，并降低整体的维护成本，最终降低总体拥有成本。

（3）混合云数据库系统服务：这类数据库服务同时包含公有云数据库系统服务和私有云数据库系统服务两类。至于哪部分数据库系统服务选择公有云服务，哪部分数据库系统服务选择私有云服务，主要从降低系统的总体拥有成本（Total Cost of Ownership，TCO）上考虑，包括构建成本、运维成本、折旧费用等。

应该选择哪种云数据库服务，主要从如下 3 个层面权衡。

（1）成本。当企业的系统规模不是很大，通常情况下租用公有云数据库系统服务的成本会低于在企业内部构建数据库系统服务的成本，但是当系统规模扩大到一定程度，在企业内部构建私有云的成本会比购买公有云服务的成本低。当前规模稍大的物联网企业，如今日头条、美团等均是构建企业内部的云服务。

(2) 差异化竞争力。如果企业对外竞争力构筑在数据库系统服务的基础上,那么企业也将构筑自己的私有云数据库系统服务。

(3) 数据的隐私度及价值。如果数据是企业的重要资产和核心竞争力,那么这类企业大多会采用基于私有云的数据库系统服务,以更好地保护数据及个人隐私。

通过上述的分析,中小型企业通常在成本、竞争力构筑、数据隐私保护这几方面权衡后,更大概率地选择公有云数据库系统服务,而大中型企业则更大概率地选择私有云数据库系统服务或者混合云数据库系统服务,其中成本因素会占据比较高的比重。GaussDB 云数据系统的目标客户主要是大中型企业。

大中型企业对云数据库系统的需求与中小型企业对云数据库系统服务的需求有较多的不同之处,具体分析如下:

(1) 具有更加高效的资源整合和利用能力。对于大中型企业,通常会管辖多个部门,这些独立部门有独立的、不同的业务,为此每个部门均针对各自不同的业务,配备不同的数据库系统,并由此提供数据库系统服务;另外,对于大中型企业,为了更好地服务各个业务部门,通常会组建平台部门,并为业务部门提供更好的数据库系统服务支持。为了提升整个企业对计算机资源、数据库系统资源的利用率,实现资源的共享和整合非常有必要,将进一步降低整体成本。

(2) 对数据库系统的规格,特别是 SLA(Service Level Agreements,服务水平协议)有更加严格的要求。大中型企业通常服务的客户数量大,业务重要,因此对数据库服务请求的吞吐量、每个请求的响应时延、容量扩展速度、计算扩展速度均有更加严格的要求。

(3) 大中型企业内部多个部门之间的业务并不是完全独立的,通常各自系统之间存在着一定的耦合关系。这类耦合关系,通常表现在数据库表模式之间的耦合关系、数据对象之间的一致性问题、数据之间流动关联关系(Extract Transform Load-workflow,ETL)。例如某银行,需要通过数据中台维护各个部门之间数据表模式的一致性。

(4) 具有对新应用的快速迭代和快速开发需求。为了能够加速新应用的开发,在云场景下对数据库系统的克隆、回溯、合并等能力提出了新的需求。特别是在大数据规模环境下,如何高效地实现数据库数据的克隆、回溯等能力是极其重要的。

5.5.2　弹性伸缩的多租户数据库架构

为了能够适应各类大中型企业对云数据库系统的需求,GaussDB 云数据库系统提

供了更强的存储资源、计算资源之间的组合能力。其主要目的是实现存储资源的独立扩容和缩容能力、计算资源的独立扩容和缩容能力，以及存储资源与计算资源在弹性扩缩容环境下的自由组合能力。从本质而言，GaussDB 云数据库系统提供多租户(Multi-tenant)和扩缩容(Elasticity)的组合能力。

1. 多租户存储计算共享架构

单个应用服务独立部署转向共享服务，对企业内部数据库系统的运维产生较大的变革，并有效降低其运维成本。

如图 5-15 所示，数据库系统从孤立的独立部署转向计算与存储共享的部署形态，在实现计算与存储共享的同时实现存储资源的独立扩缩容，以及计算资源独立的扩缩容。当云部署的数据库系统能够提供独立的存储、计算扩缩容能力后，数据系统需要被迁移的概率将被大幅度降低，由此可以提升数据库系统的业务连续性(Business Continuity)，系统比较容易实现在运行过程中存储资源的扩缩容，以及计算资源的扩缩容。

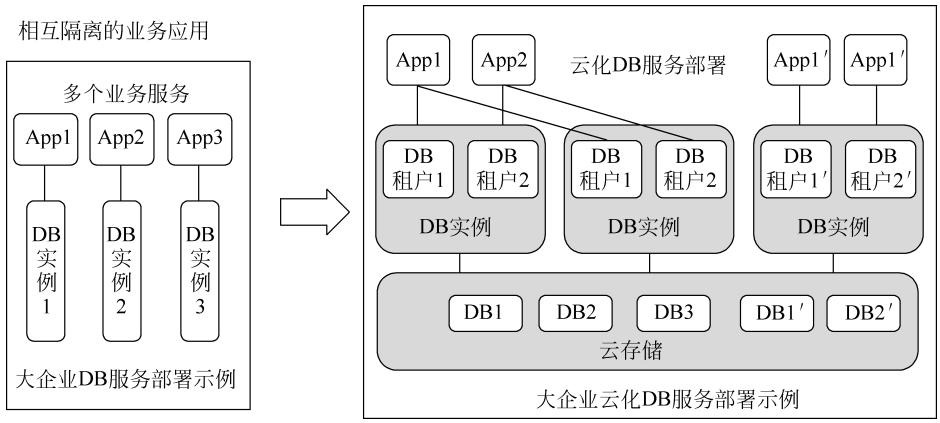


图 5-15　多租户数据库系统部署形态

2. 三层逻辑架构实现存储、计算独立扩缩容

为了有效实现云数据库系统在存储资源、计算资源的独立扩缩容，需要实现计算与存储的解耦，以及各自的扩缩容能力。

如图 5-16 所示，为了实现 GaussDB 云数据库系统在存储和计算方面的弹性，现将整个数据库系统分为 3 层，分别是弹性存储层、弹性事务处理层及无状态 SQL 执行

层。GaussDB 云数据可以在事务处理层实现横向扩展，以保证满足大中型企业对数据库系统的不同需求。无状态的 SQL 执行层，可以实现对不同客户端连接请求数进行扩展的能力。

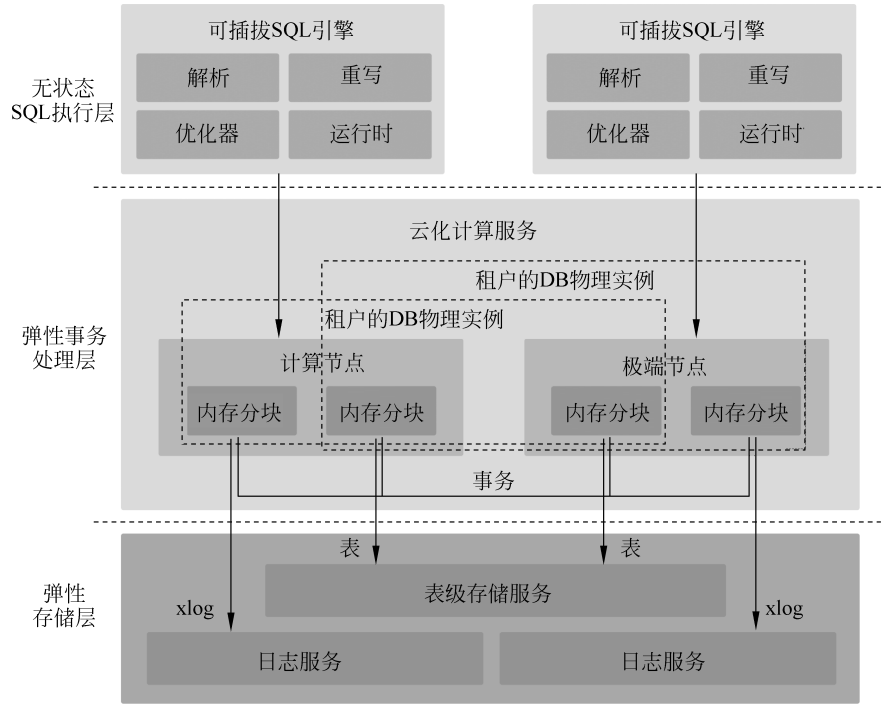


图 5-16　GaussDB 云数据库系统的分层架构

GaussDB 虽然实现了在数据库系统 3 个层次上的不同可扩展能力，但是不要以为这些组件是部署在不同的物理机器上。相反的，为了更好地提供性能，这 3 个层次的组件通常在部署的时候，具有很强的相关性，需要尽可能地联合部署（尽量部署在一台物理机上或一个交换机内），以降低网络时延带来的开销。

3. 云数据库的克隆与复制支持

将企业的数据库系统搬到云系统之上，可以提供更加便利的数据库系统管理功能，以满足企业对业务的测试、新业务的构建等不同需求，加速业务上线的速度。

由于云数据库系统实现多个数据库系统之间数据的共享（即在一个存储池中，存储大量的数据库）。因此，可以实现对这些数据库高效的复制、克隆等功能。比如，某公司可能需要基于现有数据库系统的当前数据，开发一个新的应用。传统的做法是，

为了测试应开发的应用不影响到现有的线上应用，公司通常会构建一个新的数据库系统，并从当前线上系统导出一份最新的数据，将这份新的数据导入另外一个数据库系统中（比如刚创建的数据库系统实例），并在该数据库系统开发、测试新的应用。

当这些数据库系统共同部署在云数据库系统中时，可以实现数据库系统的克隆（包括数据与系统）和复制（仅数据），比如使用 COW 机制（对于持久化存储的 Copy-on-Write 机制）可以实现对于数据库数据的快速克隆（仅克隆了元数据，数据库数据并未复制）。通过 COW 机制，构建在克隆数据库上的业务可以直接修改克隆的数据库系统中的数据。

如图 5-17 所示，云数据库系统可以对生产数据库系统进行克隆、复制等操作，对于克隆、复制出来的数据库系统可以用于非生产系统，并用于开发、测试流程或是参与到基准测试中。需要说明的是，用户非生产系统的数据库系统保持了和生产系统当前一致的数据，同时生产系统中更新的一部分数据也可以实时同步到非生产数据库系统中，进而保持这两部分数据之间的一致性。

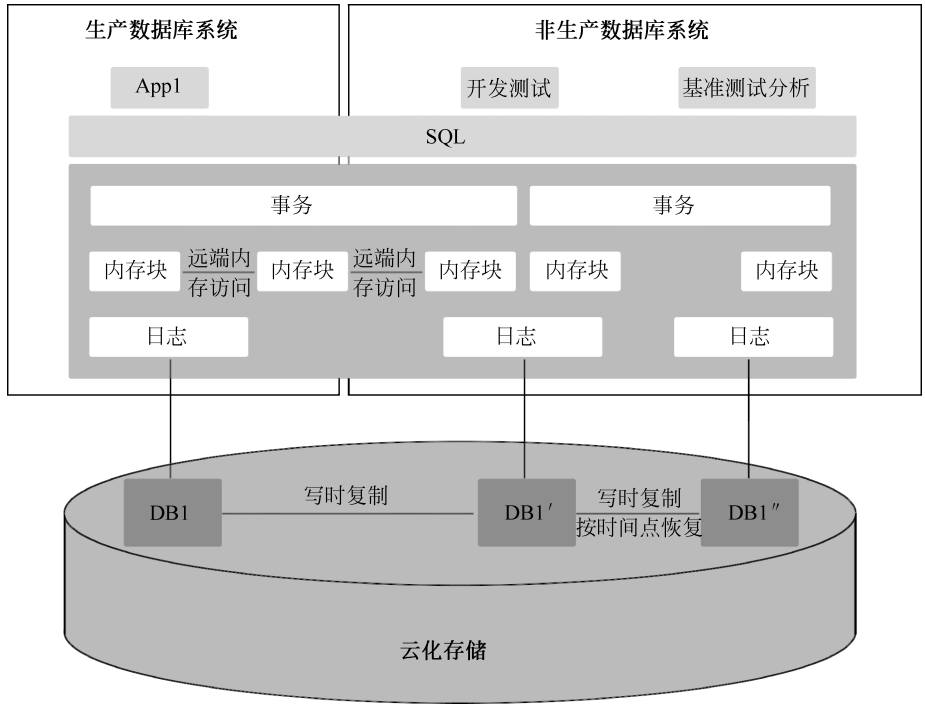


图 5-17　GaussDB 云数据库系统的数据库克隆与复制

通过上述分析，GaussDB 云数据库系统通过分层，实现了在存储层和计算层的弹性，以及这两者的任意组合，能够较好地适应大中型企业对云数据库系统的需求。另外，GaussDB 云数据库系统在此基础上又进一步实现了对现有数据库系统的高效克隆、复制，以满足中大型企业提升业务演进的速度和节奏。

5.6　GaussDB 多模数据库架构

从字面意思来理解，多模数据库系统主要用于实现对多种模型数据的管理与处理。它包括 3 个层面的内容：

（1）多模数据的存储：一个统一的多模数据库系统需要提供多种数据模型，包括关系、时序、流、图、空间等的存储能力。

（2）多模数据的处理：一个统一的多模数据库系统需要提供多种数据库模型，包括关系、时序、流、图、空间等的处理能力。

（3）多模数据之间的相关转换：大多数情况下，客户的数据产生源只有一个，即数据产生源的数据模型是单一的，但是后续处理中可能需要使用多种数据库模型来表征物理世界，进而进行数据处理，或者需要通过多种模型之间相互协作来完成单一任务，因此不同模型之间的数据转换也是极为重要的。

5.6.1　设计思想与目标客户

多模数据库系统的设计与实现，主要是为了简化客户对数据管理、数据处理的复杂度，以及降低整体系统运维的复杂度。为此，在数据库系统之上提供统一的多模数据管理、处理能力，以及统一运维能力是多模数据库系统核心设计思想。经过近两年的设计与开发，我们总结出客户的需求，客户可以分成如下两大类，而不同的类别的客户将影响到整个多模数据库系统的架构。

（1）侧重多模数据一致性的客户：这类客户通常有比较单一的数据产生源，并以关系数据为主，重要关键性业务是强调数据之间的一致性，如银行类客户、政府类客户。在构建多模数据库系统时，需要重点考虑多模数据之间的一致性，以及多模数据之间的融合处理。

(2) 侧重多模极致性能的客户：这类客户的需求通常无法通过简单的多模数据融合来达成。在他们苛刻的条件下，通常需要极致地优化性能，才能满足他们的需求。

5.6.2 面向数据强一致的多模数据库系统架构

GaussDB 用户除能使用关系数据库外，还有使用图数据库、时序等多模引擎的能力。如公共安全场景下，用户会将 MPPDB(Massively Parallel Processing DataBase，大规模并行处理数据库)的数据，导出到图数据库中，使用图引擎提供的图遍历算法，查找同航班、同乘火车等关系。在类似应用场景下，存在数据转换性能低，使用多套系统维护和开发成本高，数据导出安全性差等问题。引入多模数据库统一框架[Multi-Model Database (MMDB) Uniform Framework]，为用户提供关系数据库、图数据库、时序数据库等多模数据库统一数据访问和维护接口，减少运维和应用开发人员的学习和使用成本，提升数据使用安全性(数据无须在多个系统之间进行切换，减少了数据在网络上暴露的时间)，如图 5-18 所示。

1. 系统逻辑架构

多模数据库统一框架基于 GaussDB 开发，通过类似领养(Linked)机制，快速扩展图、时序数据库引擎，对外提供统一的 DML、DDL、DCL、Utilities、GUI 访问接口。运维和应用开发人员可以将扩展的多模数据库与 GaussDB 无缝衔接起来，当成一套系统，统一管理与运维通过统一接口使用扩展引擎提供的能力，减少对新的数据库引擎的学习和使用成本。具体介绍如下。

(1) 扩展引擎(Extension Engine)包括图引擎、时序引擎、空间引擎，扩展采用统一机制、模块化设计，并提供类似领养机制，具有扩展快速、对原系统无影响的优点。

(2) 统一(Uniform)DML 提供关系数据库 SQL、图数据库图遍历语言(Gremlin)、时序函数和操作等多语言的数据操作能力。用户可以使用统一的 ODBC、JDBC、GUI 接口访问 MPPDB 及扩展引擎。

(3) 统一 DDL、DCL、Utilities 均使用存储过程，为各扩展引擎维护专属的虚拟系统表(Pseudo Catalog)，减少对 MPPDB 的影响和依赖。

(4) 统一 DDL 为扩展引擎提供统一数据定义能力，包括扩展引擎创建、删除，扩展引擎对象的创建、销毁[如图 5-18 所示图引擎的图(graph)、顶点(vertex)、边(edge)的创建和删除]等 DDL 能力。

(5) 统一 DCL 为扩展引擎提供统一数据控制能力，包括统一权限(Grant、Revoke)管

理，性能统计（Analyze）能力。

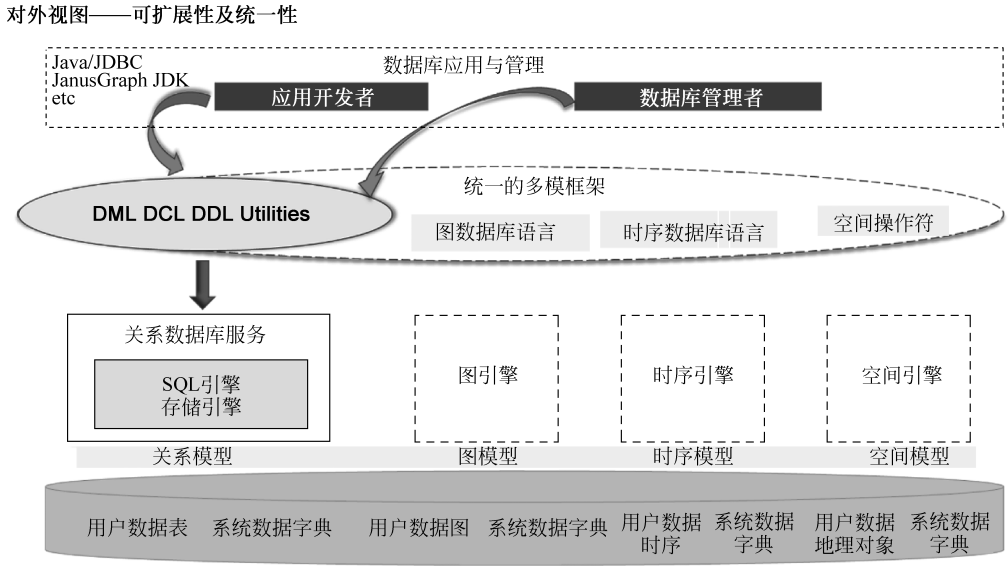


图 5-18　多模数据库逻辑架构图[Multi-Model Database（MMDB）Uniform Framework]

（6）统一 Utilities 提供备份恢复、安装卸载、集群管理等功能。

（7）统一 GUI 在高斯 Data Studio IDE 基础上，扩展了对图数据库、时序数据库的支持，提供扩展引擎的基本数据访问接口及管理接口（备份、恢复等）。设计时尽量保持 Data Studio 原有系统设计及显示结构，减少对原有 Data Studio 的改动量。

在图 5-18 的多模数据库系统的逻辑架构中，除了统一的多模框架外，该系统架构使用了统一的数据存储，即关系型存储。据统计，当前大量客户的数据产生源主要包括两大类：①关系型交易数据系统；②传感器（周期性地产生比较规则的数据）。分布式关系数据库系统实现数据的统一存储与处理，可以大幅度简化客户的数据处理，最终实现数据的强一致。

为了简化用户对数据的管理与处理，我们在数据统一存储（即关系型存储）的基础上提供了多类数据处理引擎，包括图引擎、时序引擎、空间引擎等，不仅可以提高对多类数据模型的处理效率，同时也提供了多类数据处理引擎的处理语言。比如，对于图引擎，提供了 Gremlin 图处理语言的支持；对于时序引擎，提供了业界标准的时序处理语言。

多模数据库系统中多模数据模型的任意组合。为了适应不同用户对不同类型数据处理的需求，GaussDB 多模数据库系统提供了多种模型之间的任意组合。在整体架构上，将引擎的元数据独立出来，以实现任意时刻的启动和关闭新的多模引擎。

2. 系统物理架构

多模数据库是处理包含图、时序等多种数据模型的统一的数据库。图 5-19 给出了多模数据库的物理设计架构图。多模数据库提供统一的 DDL 和 DCL 管理，用户可以方便地把外部引擎交给多模数据库进行管理。

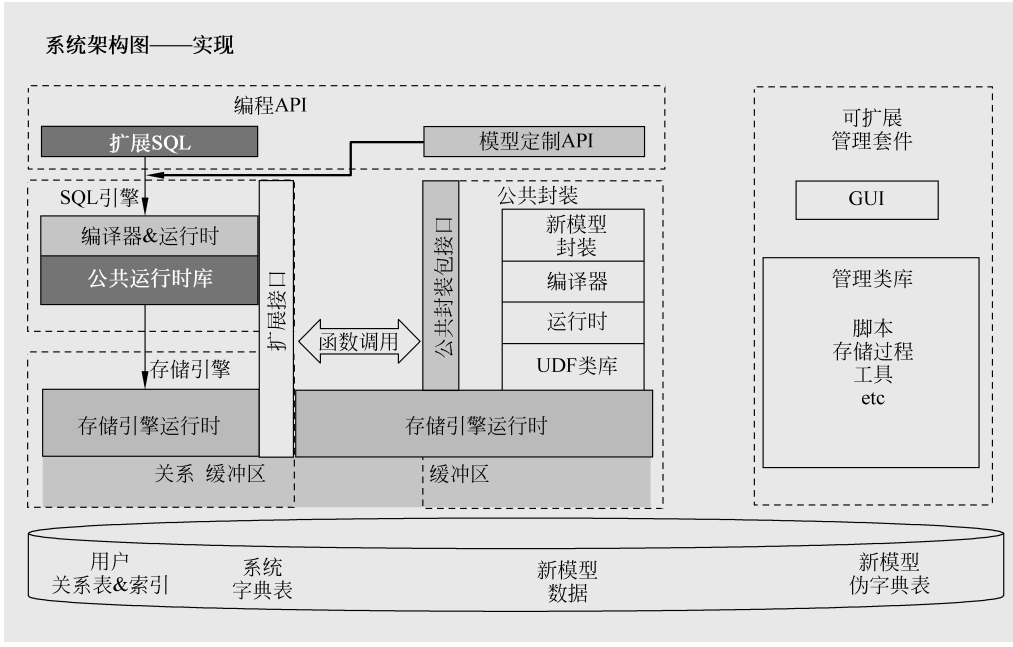


图 5-19　GaussDB 多模数据库的物理设计架构图

多模数据库 DML 采用 UDF(User Defined Function，用户自定义函数)的方式，提供统一的 GUI、ODBC、JDBC 等外部接口，输入相应的 UDF 进行对外部数据的查询分析。多模数据库接收到查询请求后，发送给对应的外部引擎执行，并将执行结果返回，借助 GaussDB 原有的方式呈现给用户。

多模数据库的系统表采用虚拟系统表(Pseudo Catalog)的方式管理。虚拟系统表都是用户表。这样，用户可以方便地添加和删除多模的能力，对 MPPDB 的影响减到最小。

多模数据库在 GaussDB 基础上进行设计。GaussDB 引入多模框架后，需要在 GaussDB 内部进行扩展，用来适配多模数据的执行和管理流程。这里的扩展指的是 GaussDB 内部针对多模引擎所做的适配。它既可以是功能上的，包括多模数据对象和

关系数据对象的相互依赖关系，对异常处理、事务管理所做的适配，还有针对多模数据的执行流程在 GaussDB 内部所做的适配工作；又可以是性能上的，例如优化器等组件上提供对多模引擎的支持。

公共模块(Common Envelope)介于这些扩展和外部引擎之间，关键组件——公共模块封装(Common Envelope Wrapper)打包提供了 GaussDB 扩展针对不同引擎的具体实现。也可以把这部分内容叫作外部引擎封装(Foreign Engine Wrapper)。即针对不同的引擎，可以通过外部引擎封装打包不同的实现过程。

此外多模数据库还提供其他统一框架管理，包括连接管理、轻量解析(Shallow Parse)和多模缓存管理等。

5.6.3　面向极致性能的多模数据库系统架构

面向极致性能的场景，如极端的互联网场景，上述多模数据系统可能无法满足需求。如果需要处理的数据量，或者需要处理的响应时间包含有极致的要求，统一的关系存储可能无法满足要求。在这类业务场景下，通常需要面向特性数据模型的原生数据存取模型，进而加速数据的存取与处理，面向极致性能的多模数据库系统架构如图 5-20 所示。

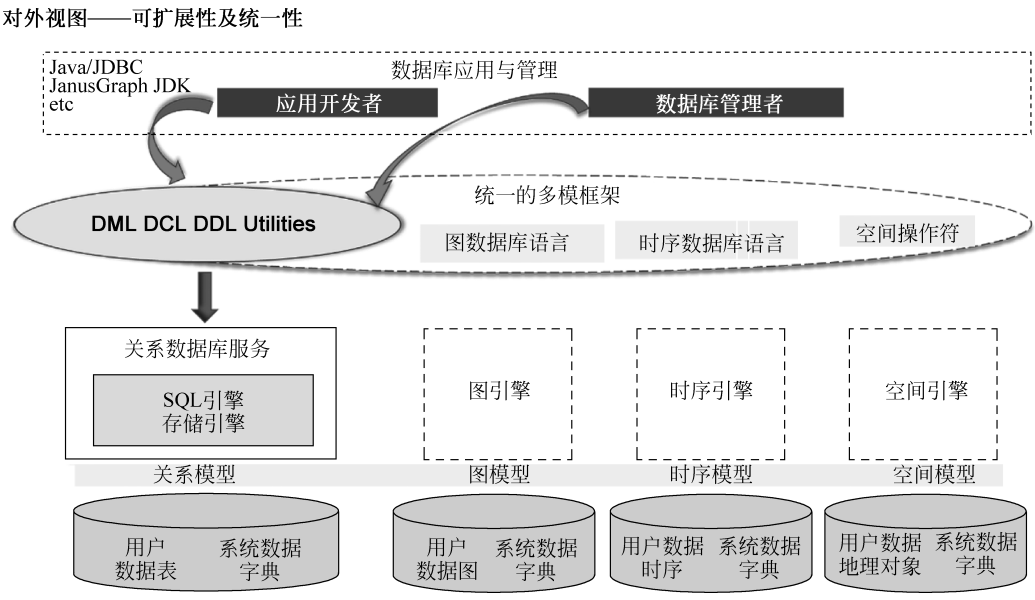


图 5-20　面向极致性能的多模数据库系统架构

5.7 小结

GaussDB 从华为公司内部开始应用，现在已经发展为多个数据库产品系列，包括了 GaussDB 100 OLTP 数据库、GaussDB 200 OLAP 数据库、GaussDB 云数据库、多模数据库等，形成了种类丰富、技术先进的数据库系列。随着处理器、存储介质等硬件的持续发展和软件技术的日益更新，新型应用不断产生，GaussDB 数据库也将提供更加丰富的产品。

习题

(1)(多选)下列哪些措施，有助于建立数据库生态？(　　)

A. 通过云化，降低数据库维护成本

B. 通过 AI 算法替代 DBA 进行数据库优化

C. 进行周边产品的对接与认证

(2) 大中型企业对云数据库系统的需求有哪些特点？

(3) GaussDB 多模数据库分为哪两种类型？它们在架构上有什么区别？

(4) GaussDB 的关键技术架构有哪些？