

达梦技术丛书

DM8 分布计算集群



前言

概述

本文档主要介绍 DM 分布计算集群的系统架构、系统原理、系统特性、基本概念、关键技术以及如何搭建 DM 分布计算集群并使用等。

读者对象





本文档主要适用于 DM 数据库的：

- 开发工程师
- 测试工程师
- 技术支持工程师
- 数据库管理员

通用约定

在本文档中可能出现下列标志，它们所代表的含义如下：

表 0.1 标志含义

| 标志 | 说明 |
|--|--------------------------|
|  警告： | 表示可能导致系统损坏、数据丢失或不可预知的结果。 |
|  注意： | 表示可能导致性能降低、服务不可用。 |
|  小窍门： | 可以帮助您解决某个问题或节省您的时间。 |
|  说明： | 表示正文的附加信息，是对正文的强调和补充。 |

DM8 分布计算集群

在本文档中可能出现下列格式，它们所代表的含义如下：

表 0.2 格式含义

| 格式 | 说明 |
|-------------------------|--|
| HarmonyOS Sans SC Light | 表示正文。 |
| Poppins Light | 表示代码或者屏幕显示内容。 |
| 粗体 | 表示命令行中的关键字（命令中保持不变、必须照输的部分）或者正文中强调的内容。标题、警告、注意、小窍门、说明等内容均采用粗体。 |
| <> | 语法符号中，表示一个语法对象。 |
| ::= | 语法符号中，表示定义符，用来定义一个语法对象。定义符左边为语法对象右边为相应的语法描述。 |
| | 语法符号中，表示或者符，限定的语法选项在实际语句中只能出现一个。 |
| { } | 语法符号中，大括号内的语法选项在实际的语句中可以出现 0...N 次（N 为大于 0 的自然数），但是大括号本身不能出现在语句中。 |
| [] | 语法符号中，中括号内的语法选项在实际的语句中可以出现 0...1 次，但是中括号本身不能出现在语句中。 |
| 关键字 | 关键字在 DM_SQL 语言中具有特殊意义，在 SQL 语法描述中，关键字以大写形式出现。但在实际书写 SQL 语句时，关键字既可以大写也可以小写。 |

访问相关文档

如果您安装了 DM 数据库，可在安装目录的“\doc”子目录中找到 DM 数据库的各种手册与技术丛书。

目录

| | | |
|-----|--------------------|----|
| 1 | 引言..... | 1 |
| 2 | DMDPC 概述..... | 3 |
| 2.1 | 系统架构 | 3 |
| 2.2 | 系统原理 | 8 |
| 2.3 | 系统特性 | 11 |
| 3 | 基本概念和技术指标..... | 14 |
| 3.1 | 基本概念 | 14 |
| 3.2 | 主要技术指标..... | 16 |
| 4 | DMDPC 使用须知..... | 20 |
| 4.1 | 软硬件环境..... | 20 |
| 4.2 | 系统规范 | 20 |
| 4.3 | 暂不支持功能..... | 21 |
| 5 | DMDPC 的关键技术..... | 24 |
| 5.1 | 元数据服务..... | 24 |
| 5.2 | 并行线程管理..... | 25 |
| 5.3 | 数据存储 | 25 |
| 5.4 | 数据交换与数据迭代操作符 | 25 |
| 5.5 | DMDPC 的计划生成..... | 27 |
| 5.6 | 子计划分割与调度..... | 34 |
| 5.7 | 生产者、消费者并行执行模型..... | 37 |

DM8 分布计算集群

| | | |
|----------|--------------------------------|------------|
| 5.8 | 链路通讯 | 38 |
| 5.9 | 计算与存储分离 | 38 |
| 5.10 | 动态增删节点 | 39 |
| 5.11 | 分布式事务一致性 | 41 |
| 5.12 | 分布式事务的数据可见性 | 44 |
| 5.13 | 自动选举主库 | 46 |
| 5.14 | RAFT 归档 | 47 |
| 5.15 | 自动同步日志 | 48 |
| 5.16 | C_SEQNO/C_LSN 维护 | 48 |
| 5.17 | 数据页刷盘和检查点推进 | 49 |
| 5.18 | 自动故障处理 | 49 |
| 5.19 | 故障恢复 | 50 |
| 5.20 | 多副本影子库 | 51 |
| 6 | DMDPC 配置 | 53 |
| 6.1 | 相关参数 | 53 |
| 6.2 | 配置方法 | 70 |
| 7 | DMDPC 集群部署 | 104 |
| 7.1 | 命令行工具部署 DMDPC (单机架构) | 104 |
| 7.2 | 命令行工具部署 DMDPC (BP 多副本架构) | 111 |
| 7.3 | 命令行工具部署 DMDPC (MP 多副本架构) | 126 |
| 7.4 | 图形化部署 DMDPC | 139 |
| 7.5 | 命令行工具搭建多副本影子库 | 146 |
| 8 | DMDPC 集群的管理与使用 | 149 |

DM8 分布计算集群

| | | |
|-----------|-------------------------|------------|
| 8.1 | 集群信息查看 | 149 |
| 8.2 | 动态增删节点 | 150 |
| 8.3 | 管理 DMDPC 表空间 | 166 |
| 8.4 | 管理 DMDPC 表空间组 | 167 |
| 8.5 | 管理用户 | 170 |
| 8.6 | 管理 DMDPC 表 | 172 |
| 8.7 | 数据迁移 | 187 |
| 8.8 | 备份与还原 | 198 |
| 8.9 | 管理表的读写属性 | 235 |
| 8.10 | 灰度升级与引流 | 236 |
| 8.11 | 多副本手动切换主备 | 239 |
| 8.12 | SP 缓存表行数 | 241 |
| 8.13 | 本地登录 | 241 |
| 9 | 快速装载工具 | 246 |
| 10 | 查询分析 | 248 |
| 10.1 | 查看子计划 | 248 |
| 10.2 | 查看物理计划生成阶段的计划 | 250 |
| 10.3 | 查看正在执行的语句信息 | 253 |
| 11 | SQL 调优 | 255 |
| 11.1 | DMDPC 数据分发方式提示 | 255 |
| 11.2 | HINT 实例 | 259 |
| 12 | 相关系统表和动态视图 | 269 |

DM8 分布计算集群

| | | |
|-------|---------------------------|-----|
| 12.1 | DPC_NETWORK..... | 269 |
| 12.2 | DPC_HOST | 269 |
| 12.3 | DPC_HOST_ADDR | 269 |
| 12.4 | DPC_FOLDER | 270 |
| 12.5 | DPC_BP_DOMAIN | 270 |
| 12.6 | DPC_DOMAIN_FOLDER..... | 270 |
| 12.7 | DPC_FOLDER_INSTANCE | 271 |
| 12.8 | DPC_BP_GROUP | 271 |
| 12.9 | DPC_BP_RAFT | 271 |
| 12.10 | DPC_INSTANCE..... | 272 |
| 12.11 | DPC_NET_CONF | 273 |
| 12.12 | DPC_TABLESPACE | 274 |
| 12.13 | DPC HTS..... | 274 |
| 12.14 | V\$DPC_STASK_THRD | 275 |
| 12.15 | V\$RLOG_RAFT_INFO | 278 |
| 12.16 | V\$SQL_NODE_HISTORY | 281 |
| 12.17 | V\$DPC_MP_CFG | 282 |
| 12.18 | V\$DPC_QC_HISTORY | 283 |
| 12.19 | V\$ARCH_STATUS | 283 |
| 12.20 | V\$DPC_EGTS_MP_INFO | 284 |
| 12.21 | V\$DPC_ENET_HISTORY | 284 |
| 12.22 | V\$ARCH_SYS..... | 285 |
| 12.23 | V\$DM_ARCH_INI..... | 286 |
| 12.24 | V\$DPC_ESESS_STMTS..... | 288 |

DM8 分布计算集群

| | | |
|-------|------------------------------|-----|
| 12.25 | V\$DPC_EXA_INFO | 288 |
| 12.26 | V\$DPC_ROT_INFO | 289 |
| 12.27 | V\$DPC_ESESS..... | 289 |
| 12.28 | V\$DPC_EDCT_RAFT | 290 |
| 12.29 | V\$DPC_EDCT_INSTANCE | 290 |
| 12.30 | V\$EHTD_GRP | 291 |
| 12.31 | V\$TABLE_ROW_CNT_CACHE | 292 |
| 12.32 | V\$DPC_TS_MOVE | 292 |
| 12.33 | V\$DPC_QC_SCHED_HISTORY..... | 293 |
| 12.34 | V\$GLOBAL_RAFT_INFO | 294 |

1 引言

达梦分布计算集群英文全称 DM Distributed Processing Cluster，简称 DMDPC。

DMDPC 是基于达梦数据库管理系统研发的一款同时支持在线分析处理和在线事务处理的新型分布式数据库系统。它既具备传统单机数据库的绝大部分功能，又提供了分布式计算集群才拥有的高可用、高扩展、高性能、高吞吐量和对用户透明等高级特性。

DMDPC 关注和解决的是大数据、计算与存储分离、高可用、支持全部的 SQL 标准、拥有完整的事务处理能力和集群规模能够动态伸缩的业务场景。在这些场景中，用户经常会遇到以下问题：

✚ 大量的复杂查询操作要求优化器能够生成优良的执行计划，并且执行引擎能够充分利用多机器、多核的硬件资源；

✚ 某些行业对数据一致性和多副本备份容灾有较高要求，同时希望维护成本足够低和故障恢复时间足够短；

✚ 用户的业务规模有峰值，要求所需的机器资源能够灵活地添加和删除。

为了解决上述问题，DM 提供了全新的、一站式的分布式数据库解决方案 DMDPC。

本文档主要介绍 DMDPC 的系统架构、系统原理，系统特性，关键技术，并举例说明搭建过程和管理方法。

本手册可以帮助用户：

- 了解 DMDPC 相关概念
- 了解 DMDPC 如何通过 RAFT 协议保证高可用
- 了解 DMDPC 的动态扩展
- 了解 DMDPC 计划生成、子任务划分和通过动态性能视图来监控各子任务、各工作线程的执行情况

DM8 分布计算集群

- 了解如何对 DMDPC 集群进行部署和使用

2 DMDPC 概述

本章重点介绍 DMDPC 的系统架构、系统原理和系统特性。

2.1 系统架构

DMDPC 架构旨在提供具有分布式特性的可扩展、高性能数据库解决方案，以满足具有高并发、大规模数据存储、业务快速扩张等特征的用户业务对数据库的要求。

2.1.1 DMDPC 架构

一个完整的 DMDPC 架构由计划生成节点 SP、数据存储节点 BP 和元数据服务器节点 MP 三部分组成。SP 对外提供分布式数据库服务，用户可以登录到任意一个 SP 节点，获得完整的数据库服务；BP 负责存储数据，执行 SP 的调度指令并将执行结果返回给 SP；MP 负责存储元数据并向 SP、BP 提供元数据服务。

SP 节点不存储数据，配置成单机即可。MP 和 BP 节点既可以配置成单机，也可以配置成多副本系统。其中每一个多副本系统中只有一个作为主节点，其余节点均作为备份节点。

下图是一个典型的 DMDPC 架构图。左侧为单机 DMDPC，右侧为配有多副本系统的 DMDPC。

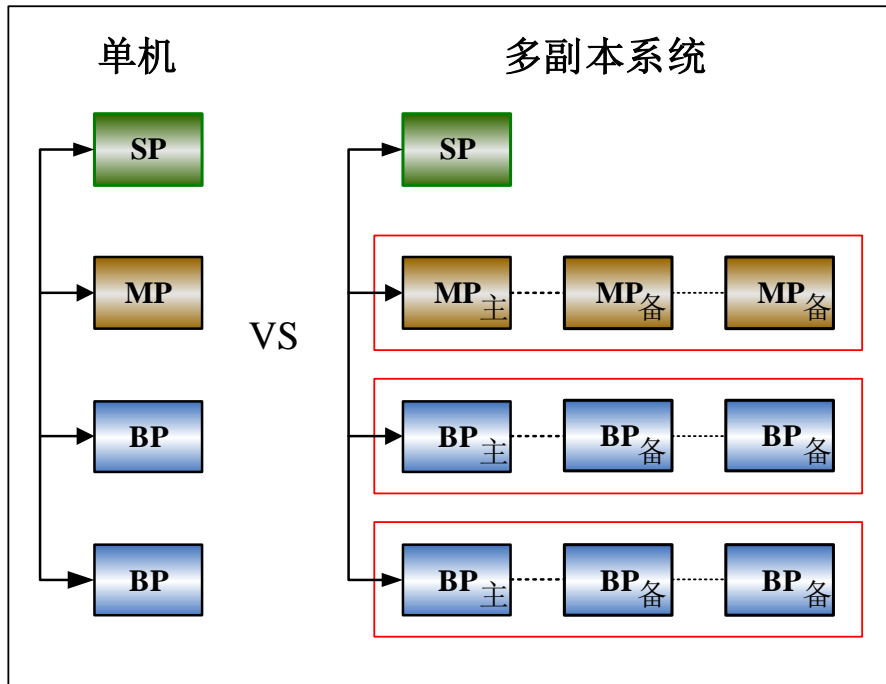


图 2.1 典型的 DMDPC 架构

2.1.1.1 计划生成节点

计划生成节点，英文全称为 SQL Processor，简称为 SP。

SP 为 DMDPC 集群中对外提供数据库服务的节点，负责接收用户请求并生成计划、划分子计划、按照一定规则计算并行度并调度各个子计划，并最终将执行结果返回给用户。对于一次客户端请求任务来说，客户端连接的 SP 负责生成、划分并调度计划，其它的 SP 和 BP 负责执行计划。

SP 的实现是在已有的成熟达梦单机数据库处理框架的基础上新增了分布式计算处理。

因此 SP 具备以下特征：

- 全部的 SQL 标准支持：包括复杂关联查询、存储过程、视图、序列等某些分布式数据库难以支持的特性。
- SP 节点自身无状态：每个 SP 节点上不存储任何数据字典信息和用户数据，一个集群中可以存在多个 SP 节点。连接上任何一个 SP 节点都可以获得完整的数据库服务。

DM8 分布计算集群

- SP 具备子计划的执行能力：因为 SP 和 BP 是由同一套代码编译出来的，只是不同的启动参数决定了担任不同的角色，所以 SP 和 BP 一样具有操作符的执行能力。例如：从资源均衡利用和计算存储分离角度来考虑，DMDPC 将部分子计划安排在 SP 上执行。

- 支持动态增删节点。

2.1.1.2 数据存储节点

数据存储节点，英文全称为 Backend Processor，简称为 BP。

BP 为 DMDPC 集群中数据实际存储的节点，负责存储数据和接收 SP 的子任务调度指令，执行子任务，并返回结果给 SP。

一个 DMDPC 集群可配置多个 BP 节点同时提供服务，且可以随着用户业务量变化动态增删 BP 节点。为了保障 BP 节点能够持续提供服务，每一个 BP 节点又可以配置成一个 BP 多副本系统。

2.1.1.3 元数据服务器节点

元数据服务器节点，英文全称为 Metadata Processor，简称为 MP。

MP 为 DMDPC 集群中提供元数据服务（即字典信息服务）的节点。所有 DDL 请求都会经过 SP 转发给 MP 执行，元数据信息全部存储在 MP。

一个 DMDPC 集群只能配置一个 MP 节点提供服务。为了保障 MP 节点能持续提供服务，MP 节点可以配置成一个 MP 多副本系统。

2.1.2 多副本系统

在现实环境中，DMDPC 运行过程中有可能会碰到各种故障情况，比如系统掉电、硬件故障（如磁盘损坏）、自然灾害（地震、火灾）等意外情况，因此需要对 BP 或 MP 采用多副本系统架构进行存储，以保障 DMDPC 的数据安全和高可用性，避免出现数据损坏和丢失，

DM8 分布计算集群

并且可以快速恢复数据库服务，满足用户不间断提供数据库服务的要求。

DM 多副本系统由 N 个节点实例组成，N 必须是大于 1 的奇数。只有配置了 RAFT 归档的实例才能加入多副本系统。

目前一个多副本系统最多支持部署 9 个节点实例。同一个 RAFT 组中的所有节点（三个或三个以上）共同构成一个多副本系统。实例之间通过 XMAL 模块进行 TCP 消息通讯。各个节点实例之间基于 RAFT 协议选举出一个领导者作为主库，其他实例作为备库（也就是副本）角色运行。主库会自动向备库同步日志，备库接收并重新应用日志，从而达到主备库之间数据保持一致的目的。

2.1.2.1 BP 多副本架构

如下图 2.2 展示了三个 BP 域的集群架构，每个 BP 域中都包含多个 BP。同一个 RAFT 组的多个 BP 保存了同样数据的多个副本。例如：BP1、BP1'和 BP1''三者内容完全相同。同一个 RAFT 组中的所有 BP 节点共同构成一个 BP 多副本系统。同一个 RAFT 组中的多个副本中只有一个作为主节点对外提供服务，其余节点均作为备份节点。当主节点发生故障后，系统会从备份节点中（BP1', BP1''）重新选举出新的主节点对外提供服务。

在 BP 多副本系统中，可通过 V\$ARCH_STATUS 和 V\$RLOG_RAFT_INFO 来查看整个系统中主备环境的运行状况。

DM8 分布计算集群

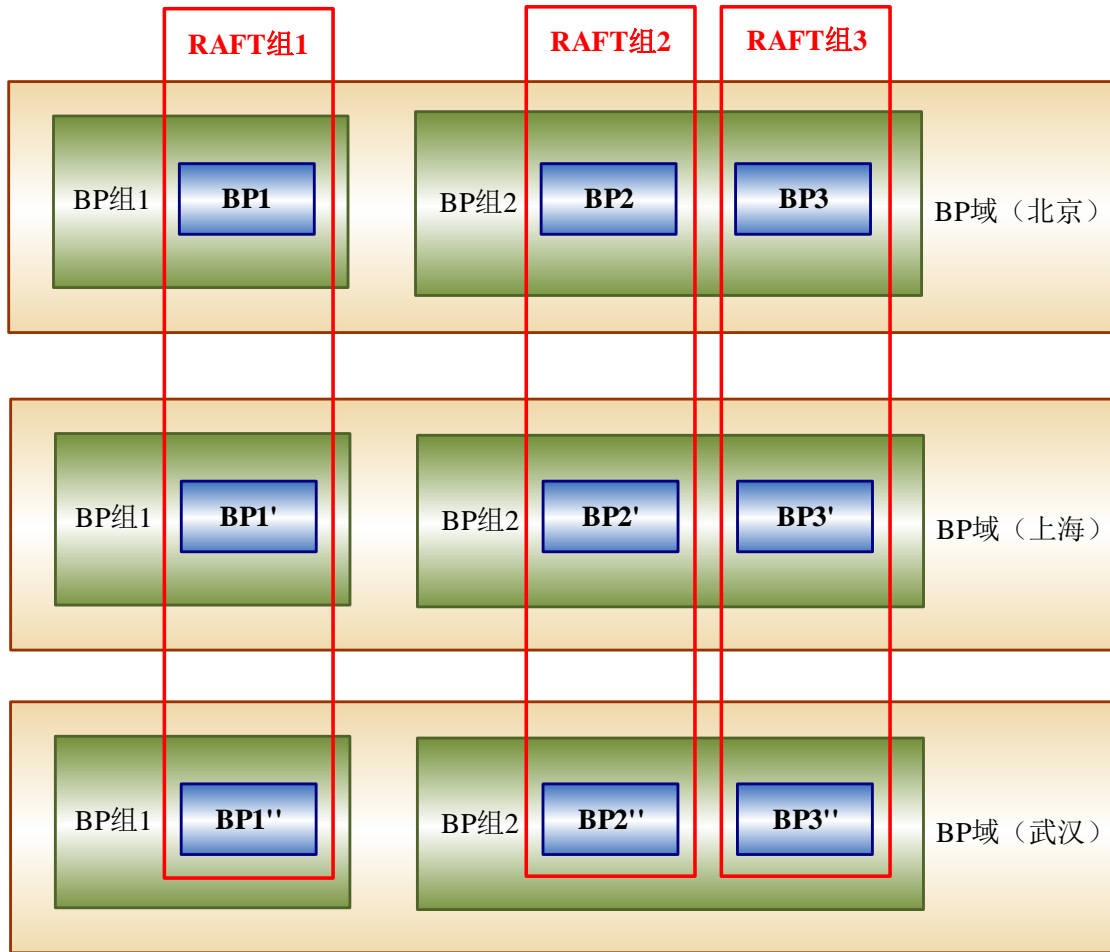


图 2.2 BP 多副本示意图

2.1.2.2 MP 多副本架构

如下图 2.3 展示了三个 MP 域的集群架构，每个 MP 域中最多只包含 1 个 MP。同一个 RAFT 组的多个 MP 保存了同样数据的多个副本。例如：MP1、MP1'和 MP1''三者内容完全相同。同一个 RAFT 中的三个 MP 节点共同构成一个 MP 多副本系统。同一个 RAFT 组中的多个副本中只有一个作为主节点对外提供服务，其余节点均作为备份节点。当主节点发生故障后，系统会从备份节点中（MP1'， MP1''）重新选举出新的主节点对外提供服务。

在 MP 多副本系统中，可通过 V\$ARCH_STATUS 和 V\$RLOG_RAFT_INFO 来查看整个系统中主备环境的运行状况。



图 2.3 MP 多副本示意图

2.2 系统原理

在 DMDPC 中，数据根据用户指定的分布规则分布在不同的 BP 上。DMDPC 的核心在于对用户请求的并行执行。下面对 DML（查询、插入、删除、更新）和 DDL 操作流程分别进行详细说明。

2.2.1 DML 流程

DML 流程分为两种情况：一般流程和优化流程。优化流程是指在实际执行时，优化器会综合多种条件，对符合优化的细节进行优化之后形成的流程。EXPLAIN 查看执行计划时，包含 `mpp_opt_flag(1)` 的即为优化流程下的执行计划，包含 `mpp_opt_flag(0)` 的即为一般流程下的执行计划。

2.2.1.1 查询

DMDPC 架构中各节点相互配合，完成用户的查询请求。下图展示了 DMDPC 架构的查询处理流程。

DM8 分布计算集群

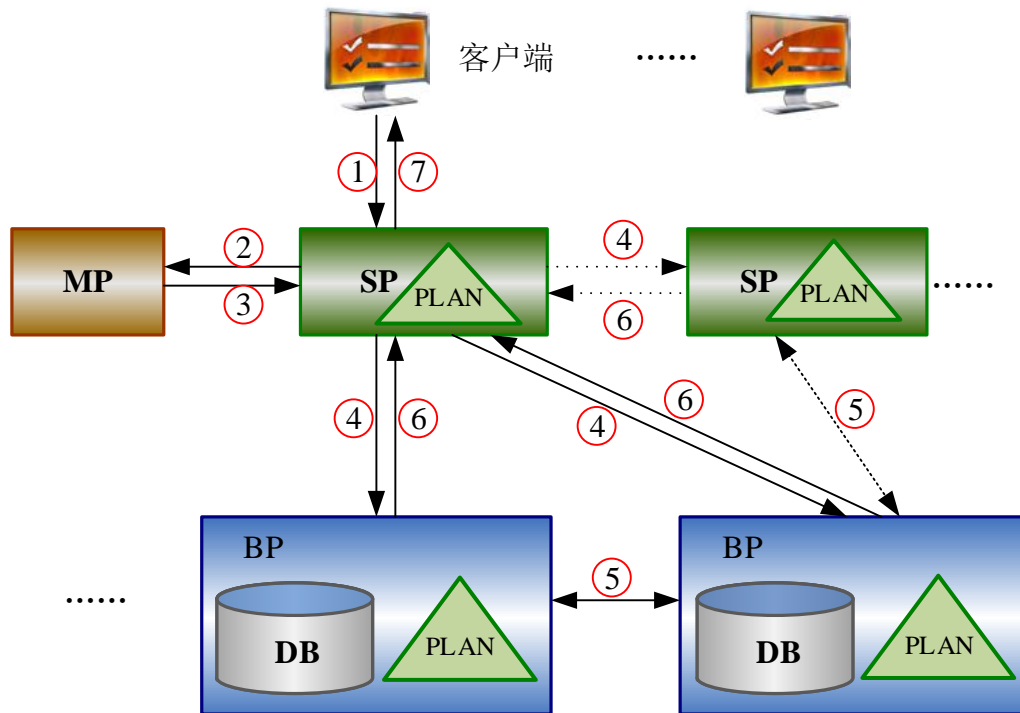


图 2.4 DMDPC 架构的查询执行流程

下面以客户端的查询 SELECT 操作为例,介绍并行查询的流程。如上图箭头序号所示。

查询请求的处理流程如下:

1. 客户端发送请求给 SP;
2. SP 生成执行计划,同时 SP 向 MP 申请获取字典信息;
3. MP 返回字典信息给 SP;
4. SP 将生成的执行计划中的一个或多个子计划发送给此次查询相关的 BP;
5. 相关 BP 接收并执行子计划。根据实际需要,不同 BP 间、同一 BP 不同线程间可能存在数据交换;

存在数据交换;

6. 相关 BP 在子计划执行完成时将成功与否信息和/或请求调度信息返回 SP;
7. SP 向客户端返回最终查询结果。

2.2.1.2 插入

插入分为两种情况：值插入和查询插入。下面分别进行说明：

1. 值插入的流程

SP 端的插入操作符计算出待插入数据所属的 BP 站点, 然后以站点为单位逐一发送插入操作符和待插入数据给 BP。BP 端负责插入。

2. 查询插入的流程

查询插入分为查询和插入两部分。

查询部分的执行过程同前文介绍的查询流程一样。

插入分为一般流程和优化流程。一般流程：由 SP 计算记录所属的 BP 站点, 然后以站点为单位发送待插入数据。优化流程：查询得到的结果直接发送给待插入表所在 BP, 如果待插入表为分区表, 那么数据在经过和表分布相同的分发后发送到各个 BP 上。

2.2.1.3 更新

更新分为一般流程和优化流程。

一般流程：SP 上查询得到了待更新的记录, 计算每条记录所属的 BP 站点, 而后以站点为单位发送更新前后的记录。

优化流程：查询得到的待更新记录按照更新对象的分布方式进行分发, 各个 BP 站点直接进行更新, 节省一次 SP 中转。可优化的情况下, 支持并发更新数据行时冲突自动重试, 一般流程不支持, 直接报错处理。

2.2.1.4 删除

删除分为一般流程和优化流程。

DM8 分布计算集群

一般流程：SP 上计算待删除记录的所属 BP 站点，以站点为单位发送待删除记录。

优化流程：待删除记录按照删除表对象的分布方式进行分发，各个 BP 站点直接删除，节省一次 SP 中转开销。可优化的情况下，支持并发删除数据行时冲突自动重试，一般流程不支持，直接报错处理。

2.2.2 DDL 流程

DMDPC 中处理 DDL 请求的流程如下：

1. 客户端发送 DDL 请求给 SP；
2. SP 在经过初步分析后，判断出是 DDL 请求，转发给 MP，等待 MP 响应；
3. MP 接收到 SP 转发的 DDL 请求后，根据具体类型，更改系统表；如果有 B 树创建、删除等用户数据操作，转发给相应 BP 完成；所有步骤完成后回复 SP；
4. SP 根据 MP 的反馈结果向客户端报告成功或失败。

2.3 系统特性

DMDPC 的主要特点包括：高可用、高扩展、高性能、高吞吐量和透明性。

2.3.1 高可用

达梦基于 RAFT 协议实现了一套全新的达梦多副本（DM Multiple Copy）系统架构。与传统的数据守护集群架构相比，此方案不再依赖守护进程和监视器，具有节点自动选主、自动故障处理、自动故障恢复的特点，在发生故障后可以快速恢复数据库服务，满足用户不间断提供数据库服务的要求。并且少数副本出现故障或者网络延迟不会影响整个系统的正常运行，因此也更能够适应分布式集群两地三中心的部署需求。

2.3.2 高扩展

DMDPC 集群提供达梦数据库高扩展性解决方案。根据用户业务量的变化，用户可以对集群规模进行扩展和缩小。详见 [5.10 动态增删节点](#)。

2.3.3 高性能

DMDPC 架构对 OLAP 和 OLTP 型场景都很适用。

查询的执行计划被拆分为一系列子任务，这些子计划被分散到多个 BP、SP 上执行以有效利用硬件资源；同时执行框架上采取了基于生产者、消费者的并行执行模型。不同的子任务允许有不同的并行度，同一个子任务在不同 BP 上的并行度也可以不同，并行度设置的灵活性能大大地提升线程资源的利用效率。

另外，通过将业务的不同表、或者同一表的不同分区拆分到多个 BP，甚至于多个主机，在面对高并发的 OLTP 型应用，集群可以极大地提升 IO 能力，分摊并发压力。

在多副本系统中，主备库之间的日志同步采用异步通信方式，主库同步日志时不需要等待备库刷盘或重演完成，备库也以异步消息通知主库自己的日志刷盘进度，消除了主备库之间的消息同步等待时间。

2.3.4 高吞吐量

与单节点数据库管理系统处理用户请求时的性能瓶颈相比，DMDPC 集群中，多个 BP 节点可以充分利用多台物理主机的处理能力，支撑更多的用户连接请求，提供更高的吞吐量。

DMDPC 集群中包含多个 BP 数据库实例，BP 数据库实例访问独立的处理器、内存。数据库实例之间通过 XMAL 模块交换数据，每个 BP 数据库实例都可以接收并处理用户的各种数据库请求。

多个 BP 节点同时提供数据库服务，有效提升集群的整体事务处理能力。

2.3.5 透明性

DMDPC 架构逻辑对用户透明。透明性是指 DMDPC 上任意一个 SP 提供的功能与普通单机数据库基本无异。用户登录 DMDPC 的任意一个 SP 节点,即可获取完整的数据库服务。

3 基本概念和技术指标

3.1 基本概念

1. 计划生成节点 (SQL Processor, SP)

对外响应数据库请求，生成并调度计划。

2. 数据存储节点 (Backend Processor, BP)

存储实际的数据，接收并执行发来的子计划。

3. 元数据节点 (Metadata Processor, MP)

提供元数据服务。

4. BS 模式

BP 节点有两种启动模式：BP 模式和 BS 模式。

通常 BP 节点是以 BP 模式启动并运行，仅作为后台数据存储节点。

如果 BP 节点以 BS 模式启动运行，用户可直接登录 BP 进行操作，此时的 BP 同时具有前端 SP 节点的功能也具有 BP 的功能，生成分布式计划，并和 MP/其他 BP 甚至辅助 SP 进行数据交互。如果查询或修改的数据都在直连的 BP 本地，则类似于单节点，执行时无网络交互的代价，这相对于先连接到 SP，再查询或修改 BP 的模式，可获得性能上的提升。

5. 查询调度总单元 (Query Coordinator, QC)

在 SP 上，执行计划生成后拆分得到了一组子计划，执行期间由 QC 对执行计划的执行进行调度控制。

6. 查询调度子单元 (Sub Query Coordinator, SQC)

在 BP 上，为处理同一计划中的一个或者多个子计划，采用 SQC 来进行协调，为每一个子计划生成一组线程执行、协调。SQC 可以认为是 QC 在每个 BP 上的调度助手，由 SQC 向 QC 汇报某一个子任务的完成情况。

7. 粒度迭代操作符 (Granule Iterator, GI)

为充分利用 CPU 的多核特性，并发访问多个分区或者一个分区内的多个不同数据页，GI 操作符从 GIU 链表摘除任务，获取到的 GIU 信息被设置到叶子操作符中。

为充分利用 CPU 的多核特性，并发访问多个分区或者一个分区内的多个不同数据页，GI 操作符负责控制每路工作线程要访问的数据信息。

8. 子计划 (Sub Plan, SPLAN)，子任务 (Sub Task, STASK)

SP 所生成的计划按照通讯操作符 ERECV/ESEND 进行划分后得到的一系列小的计划单元。在计划生成阶段，这些小的计划单元被称为子计划。每个子计划可以有自己独立的并行度。在执行阶段，子计划被称为子任务。执行时，SP 将子任务分发给相关的 BP，这些子任务由 SQC 统一调度执行。

子计划和子任务就是同一对象在不同阶段的名称。

9. 分区组 (partition Group, PG)

分区组用于定义分区的类型、分区数量和分区字段类型等信息。创建分区表时，引用相同分区组的分区表将拥有同样的分区方式，优化器可以使用这些信息构建优化的执行计划。

10. 大规模并行处理 (Masive Parallel Processing, MPP)

集群间多个主机的实例并行协作完成一个查询计划，提升查询性能。

11. 本地并行处理 (Local Parallel Processing, LPQ)

同一实例内部多个线程并行协作完成一个查询计划，提升查询性能。

12. RAFT 组

拥有相同数据的一个或多个节点共同构成一个 RAFT 组。RAFT 组中的节点个数为奇数。多副本系统基于 RAFT 算法，管理组内的节点，并始终维护一个主节点，保持 RAFT 组内的数据一致性。当 RAFT 组中含有三个或三个以上节点时，RAFT 组中的所有节点（三个或三个以上）共同构成一个多副本系统。当 RAFT 组中只有 1 个节点时，则该节点称为单副本系统。

DM8 分布计算集群

对于 BP 来说，一个 DMDPC 架构可以配置一个或多个 RAFT 组。一个 BP 组可以包含一个或多个 RAFT 组。RAFT 组的创建不依赖于 BP 组，一个 RAFT 组既可隶属于多个 BP 组，又可独立存在，不加入任何 BP 组。在 BP 单副本系统中，一个 RAFT 组最多只能包含 1 个 BP。

对于 MP 来说，一个 DMDPC 架构最多只能配置一个 RAFT 组。在 MP 单副本系统中，一个 RAFT 组最多只能包含 1 个 MP。

对于 SP 来说，SP 是对用户开放的用于计算的实例进程，本身不包含数据，为了方便统一管理，配置时也会把 SP 注册到 RAFT 组中。

13. BP 域和 MP 域

一个实例或一组实例所在的位置。DMDPC 中域分为两种：MP 域和 BP 域。例如：为了保证 MP 数据的安全，不至于同时被毁坏，现将两个 MP 分开部署，分别部署在 MP 域 1（北京）和 MP 域 2（上海）上。一个 RAFT 组在每个域上至少设有一个成员，从而使每一个域都拥有一份完整的集群数据。

一个 MP 域最多只能包含一个 MP 实例，一个 DMDPC 中的所有 MP 实例都属于同一个的 RAFT 组。

一个 BP 域包含多个 BP 实例，这些 BP 实例属于不同的 RAFT 组。

14. 容错域

容错域是多个 MP、BP 节点在同一个物理区域内的集合（例如机架/机房/城市/地区等），它可以是 DMDPC 系统中所有 MP 和 BP 节点的全集、子集或空集。

容错域内可以同时存在 MP 和 BP，不允许 SP；每个容错域内包含 0-N 个不同 RAFT 组内的节点；同一个 RAFT 组内的不同节点不能包含在同一个域内。

在实际业务中，为了考虑数据的安全性，至少要求在一个物理区域内部署的节点数不能超过半数，否则该物理区域发生故障（例如机器）时，其他副本将无法接管。

15. BP 组

BP 组是个逻辑概念，它是 BP RAFT 组的集合，可作为 DPC 系统中所有 BP RAFT 组

DM8 分布计算集群

集合的全集或子集，甚至可以为空集。

一个 BP 组由一个或多个 BP RAFT 实例组成，这些实例可以来自不同的 BP 域，也可以来自同一 BP 域。各个 BP 组所包含的 RAFT 组互不相关。BP 组只是用来简化用户建表操作中对存储 BP 的指定，实际使用中也可不创建任何 BP 组。指定 BP 组创建的分区表的数据，其数据按子表为单位分布在不同的组内 BP 上。一个子表的数据只能属于一个 BP，一个 BP 可以管理多个子表。其具体分布方式按分区类型以及分区数量的不同由系统确定。

16. SP 组

SP 组是个逻辑概念，它是 SP RAFT 组的集合，可作为 DPC 系统中所有 SP RAFT 组集合的全集或子集，甚至可以为空集。

一个 SP 组由一个或多个 SP RAFT 组成。SP 组的作用是将不同用户、不同应用使用的计算节点进行隔离，以便更好地管理硬件资源。

SP 组的存在会影响计划生成阶段所使用的计算节点，具体参考 [5.9 计算与存储分离](#)。

17. 表空间

在 DM 数据库中，表空间由一个或者多个数据文件组成。DM 数据库中的所有对象在逻辑上都存放在表空间中，而物理上都存储在所属表空间的数据文件中。

18. 表空间组

表空间组是一组位于相同 RAFT 或者不同 RAFT 上的表空间集合，用于用户建表或者指定用户默认的存储位置。表空间组中的表空间可动态扩展和收缩。且能同时包含非 HUGE 表的普通表空间和 HUGE 表专用的大表空间。

19. 日志包 (Rlog Package, RLOG_PKG)

保存物理事务产生的 Redo 日志的数据单元。

20. 日志包的全局包序号 (Global Sequence Number, G_SEQNO)

全局包序号由多副本系统中的主备库共同维护，具有全局唯一、连续、递增的特性。

21. 日志包的本地包序号 (Local Sequence Number, L_SEQNO)

DM8 分布计算集群

本地包序号由多副本系统中的主备库各自在本地维护，具有本地唯一、连续、递增的特性。

22. 已刷盘的最大全局包序号 (File Sequence Number, F_SEQNO)

已写入联机日志文件的最大 G_SEQNO，在日志包刷盘时修改维护。

23. 已刷盘的最大 LSN 全局包序号 (File Log Sequence Number, F_LSN)

已写入联机日志文件的最大 LSN，在日志包刷盘时修改维护。

24. RAFT 协议

一种分布式数据一致性协议。RAFT 协议包括领导者选举、日志同步、日志安全（提交到多数节点的日志不会丢失）和角色转换等关键算法。

一个多副本集群会包含若干节点，在任何时刻，每一个节点都处于一种角色（领导者、跟随者、候选者）。在不同的场景中，角色可发生转换。正常情况下，一个多副本系统中只有一个领导者和若干跟随者。发起选举时，发起者（跟随者或候选者）会转换为候选者，选举结束时推选出一个领导者和若干跟随者，这时候候选者会相应地转换成领导者或跟随者。

25. 领导者 (LEADER)

RAFT 协议中的领导者。

26. 跟随者 (FOLLOWER)

RAFT 协议中的跟随者。

27. 候选者 (CANDIDATE)

RAFT 协议中的候选者，发起选举时会切换为此角色。

28. 学习者 (LEARNER)

RAFT 协议中的学习者，动态增加节点时的临时角色，不参与选举和日志推进。当学习者成功加入多副本集群，成为一名正式成员后，才可以转换为 CANDIDATE、FOLLOWER 或 LEADER 角色。

29. 领导者任期号 (Leader TermID, L_TERM_ID)

RAFT 协议中当前领导者的任期号，在每次发起选举时递增。

30. 已刷盘的任期号 (TERM_ID)

已写入联机日志文件中的最后一个日志包的任期号，此任期号小于等于 L_TERM_ID。

31. 已提交的包序号 (Commit Seqno, C_SEQNO)

已提交到联机日志文件中的最大全局包序号。

32. 已提交的 LSN (Commit LSN, C_LSN)

已提交到联机日志文件中的最大全局包序号的 MAX_LSN。

33. XMAL 系统

XMAL 系统是基于 TCP/IP 协议实现的一种内部通信机制，具有可靠、灵活、高效的特性。XMAL 专门用于 DMDPC 分布式架构中。DM 通过 XMAL 系统实现 REDO 日志传输，以及其他一些实例间的消息通信。

DMDPC 使用 XMAL 系统管理各部件节点网络信息，建立节点间的通信机制。用户只需要将 SP 和 BP 节点的 IP 地址和端口等信息注册到 MP 上即可，使得增删节点的部署能够做到快速且便捷。XMAL 系统负责自动建立节点间的 TCP 通讯链路，每个节点对应一个站点编号，采用消息盒子(XBOX)的方式进行通信，只要指定站点编号和消息盒子编号即可实现节点之间的点对点消息收发。XBOX 是用来存放和管理消息的一块内存区域。

XMAL 相关的 INI 参数有 DPC_SYNC_TOTAL、XBOX_MEMORY_TARGET、XBOX_DUMP_THRESHOLD、XBOX_DUMP_PATH 和 STMT_XBOX_REUSE 等。

3.2 主要技术指标


1. BP 上的最大存储空间为 PB 级；
2. 最大 SP 组和 BP 组总数为 1024；
3. 最大的 RAFT 组数（包括 MP/SP/BP）为 1024；
4. 最大的实例数（包括 MP/SP/BP）为 4096；
5. 最大副本数为 9；
6. 最大表空间组个数 65534；
7. 一个表空间组中的最大表空间个数 1536；
8. 每个容错域至多包含 4096 个实例节点；
9. 其它指标和普通单机数据库一样。

4 DMDPC 使用须知


4.1 软硬件环境


部署 DMDPC 集群所用到的硬件和软件环境。

■ 硬件环境

 主机一台或多台。用于部署 MP、SP 和 BP。内存大小要求：至少 2GB。测试环境中可以将上述服务部署到同一主机，生产环境下推荐每个主机只部署一种类型服务器。

■ 软件环境

 操作系统。Linux、Unix、Windows 等。

 达梦数据库软件。安装好 DM 数据库软件之后，将拥有配置和管理 DMDPC 所需的所有软件：dmserver、dminit、Disql 等。这些软件位于安装目录的 bin、tool 等子目录中。

4.2 系统规范

在 DMDPC 系统中，有如下规范及约束：

1. 集群中 SP、BP、MP 都有自己的实例名，这些实例名要求全局唯一；
2. 一个 DMDPC 集群可以有多个 SP 和多个 BP，但只能有一个 MP；
3. DMDPC 中数据的分布以分区为单位，同一分区必须位于同一 BP，同一表的不同分区可以位于不同 BP，一张分区表的所有分区可以只落在部分 BP；非分区表只包含一个分区，因此非分区表只落在一个 BP 上。分区表包含一个或多个分区，因此分区表可以落在一个或多个 BP 上；DMDPC 下，分区列同时也是分布列；
4. 在 DMDPC 中分区和分布统一为一个概念；在 DM MPP 中，分区和分布是两个不同的概念；
5. 实例在初始化时需指定 DPC_MODE 以决定其角色（SP、BP 或 MP），随后启动时

需要附上 DPC_MODE 参数。实例角色在初始化后不能更改。

4.3 暂不支持功能

和达梦单机实例、DM MPP 架构相比，DMDPC 目前尚有如下功能不支持：可更新视图的 CHECK 约束/虚拟列上的 CHECK 约束/DOMAIN 的 CHECK 约束、数组表、外部表、涉及临时表的查询建表、自动统计信息搜集功能、DBMS_WORKLOAD_REPOSITORY、DBMS_STATS 包中

COLUMN_STATS_SHOW/INDEX_STATS_SHOW/TABLE_STATS_SHOW 统计信息显示过程、DBMS_SPACE 包、DBMS_RLS 包、临时表的 DML 操作回滚、堆表的全局索引。

不支持将包方法、类方法和用户自定义函数作为函数索引的列。

DMDPC 仅支持多级分区中的二级分区。二级分区中的分区组合方式和单机一样，主表和子表的总个数也不能超过 16384。

因为 DMDPC 不支持外部表，所以下面三个专门修改外部表的操作均不支持：

```
<修改表定义子句> ::=
MODIFY PATH <外部表文件路径> |
DEFAULT DIRECTORY <目录名> |
LOCATION ('<文件名>')
```

不支持修改水平分区表的操作：

```
<修改表定义子句> ::=
EXCHANGE < SUBPARTITION > <分区名> WITH TABLE [<模式名.>]<表名> |
MOVE SUBPARTITION <子分区名> TABLESPACE <表空间名>
```

不支持修改 HUGE 表的操作：WITH DELTA 和 HUGE 表跨节点交换分区 EXCHANGE PARTITION。

不支持创建二级索引时在 STORAGE 子句中指定表空间。如果已经指定则直接忽略。系

DM8 分布计算集群

统自动将表聚集索引所在的表空间作为表二级索引的表空间。二级索引表空间必须和表聚集索引表空间一致。

不支持创建 SET NULL 或 SET DEFAULT 约束检查规则的引用约束。引用约束必须包含分布列，即引用表中的引用列和被引用表中的被引用列都必需包含分布列，且分布情况完全相同。

不支持在 SP 节点执行 ALTER DATABASE MOUNT 操作。

不支持备份还原相关的系统函数，位于《DM8_SQL 语言使用手册》附录 3 系统存储过程和函数中 3. 备份恢复管理；仅支持删除归档的系统函数 SF_ARCHIVELOG_DELETE_BEFORE_TIME() 和 SF_ARCHIVELOG_DELETE_BEFORE_LSN()，位于《DM8_SQL 语言使用手册》附录 3 系统存储过程和函数中 9. 日志与检查点管理。

不支持表级备份，不支持 DPC 下 DDL_CLONE 备份；不支持联机还原，不支持还原恢复到指定 LSN，不支持 DDL_CLONE 还原及恢复。

不支持表空间级备份还原。不支持执行 LINUX 操作系统下，恢复被删除文件的过程 SP_TABLESPACE_PREPARE_RECOVER 和 SP_TABLESPACE_RECOVER。

不支持空间索引、位图索引、全局函数索引。

不支持表列的半透明加密。

不支持有 DPC 全局索引的分区表 truncate 子表。

不支持游标类型的参数。

不支持 new 数组对象。

不支持高级日志表 advanced log。

不允许设置索引的默认表空间，二级索引的表空间必须和表聚集所以表空间相同。

关闭两阶段提交后不支持 XA 接口功能。

不支持在 ALTER TABLESPACE 中使用<增加 HUGE 路径子句>。

DM8 分布计算集群

对于包含大字段列或自定义字段列的水平分区表不支持 ENABLE ROW MOVEMENT 参数(可以指定, 但是无效), 即不允许更新后数据发生跨分区的移动。

仅 BS 模式支持自引用约束, 其他模式均不支持。

5 DMDPC 的关键技术

5.1 元数据服务

元数据服务又称字典信息服务。

在 DMDPC 中，为了简化 DDL 的处理，使用专门的元数据服务器 MP 负责管理所有的字典对象元数据信息。所有 SP 和 BP 都是通过网络请求从 MP 获取字典对象的。

SP、BP 和 MP 的字典模块的构架基本一致，都拥有字典缓存；不同的是，SP 和 BP 不再进行字典数据的存储，统一交给 MP 来存储。SP 和 BP 的字典模块相当于是 MP 字典模块的客户端。

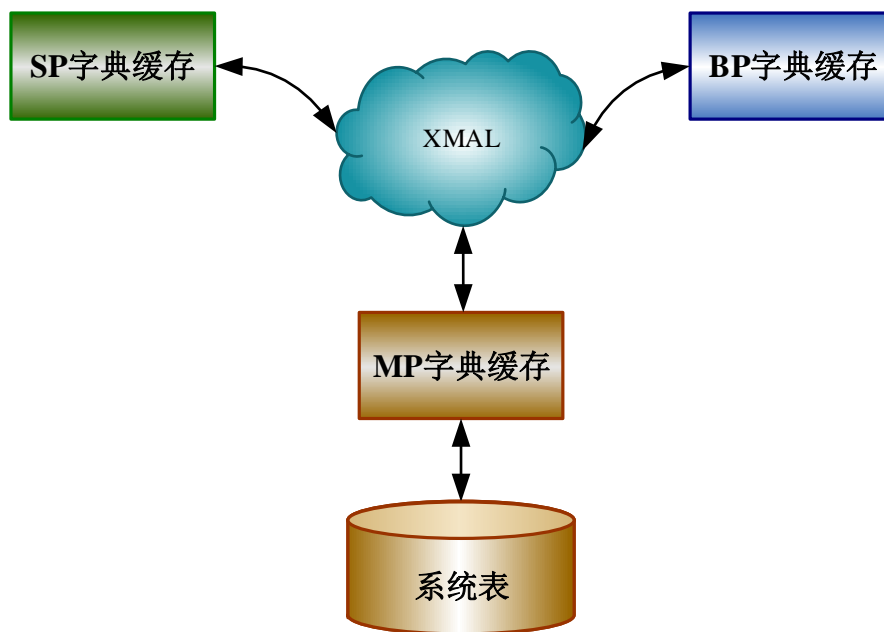


图 5.1 元数据服务整体示意图

对 SP 和 BP 而言，它们获取表、视图、索引等字典对象的步骤完全一致，现以 SP 为例进行说明。

SP 获取字典对象的步骤如下：

1. SP 在自己的字典缓存中查找是否有该对象的缓存，如果有则直接使用该对象，否则执行步骤 2；

DM8 分布计算集群

2. SP 向 MP 发出请求，获取某个特定 ID 的字典对象；
3. MP 接收请求后从系统表中加载对应的字典对象，然后返回 SP；
4. SP 解包 MP 返回的字典对象并添加到自己的缓存中。

5.2 并行线程管理

并行线程英文全称为 Parallel Thread，简称为 PTHD。

DMDPC 环境中会有大量的并行计算。简单的、固定个数的并行线程已经无法满足需求。于是 DMDPC 提供了一个可扩展的并行线程池（PTHD POOL），来供应和管理并行线程。并行线程池中的线程数量可按需分配。当服务器负载增大时，服务器会自动增加池内并行线程的数量；当负载减少时，会自动释放一些并行线程以减少资源消耗。

5.3 数据存储

DMDPC 的数据存储在各个 BP 中。

用户在创建表对象时需要指定用户表空间。如果建表时未显式指定用户表空间，系统则随机选择用户表空间。如果没有创建任何用户表空间则建表会报错。

表空间的元数据信息由 MP 进行管理。所有本地表空间元信息都记录在 MP 的控制文件中。

在 DMDPC 中，MP 包含 SYSTEM、MAIN、ROLL 和 TEMP 表空间；SP 包含 TEMP 表空间；BP 包含 SYSTEM、ROLL、TEMP 和用户表空间。

5.4 数据交换与数据迭代操作符

为了在不同的实例之间或同一实例不同的线程之间进行数据交换，DMDPC 引入了两个高效的数据交换操作符：发送操作符 ESEND 和接收操作符 ERECV。为了控制工作线程获取表数据时的粒度，DMDPC 引入了 GI 操作符。下面对这三种操作符分别加以介绍。

DM8 分布计算集群

● 发送操作符 ESEND

ESEND 将孩子操作符的数据按照某种分发规则发送给特定 ERECV 操作符。

ESEND 操作符发送的内容为当前子计划的执行结果，有时也会包含一些附加信息。附加信息包含：如是否为最后一批数据、错误码、请求调度父层子任务等。ESEND 共有 6 种数据分发方式：BY HASH、BY RANGE、BY LIST、BY N_DEST、DIRECT 和 BROADCAST。下面分别详细介绍：

1. BY HASH：按照发送列的 HASH 取值和并行度，决定发送给 ERECV 操作符中的第几个 worker 线程。例如：ERECV 所在并行度为 8，发送 key 为 C1，某条数据的 $\text{HASH}(C1) \% 8 = 5$ ，那么这条记录将发送给 ERECV 操作中第 5 个 worker 线程。

2. BY RANGE：按照发送列的范围判断发送给 ERECV 操作符的第几个 worker 线程，每个 worker 线程事先都约定了各自的接收数据范围。

3. BY LIST：按照发送列的值判断发送给 ERECV 操作符的第几个 worker 线程，每个 worker 线程事先都约定了各自的接收数据值。

4. BY N_DEST：类似于 BY HASH，都是按照 $\text{HASH}(\text{key}) \% N$ 决定发送给哪一个接收者线程。区别在于 BY HASH 中的 N 取值为 HASH 水平分区子表个数，通常用在 Partition Wise Join 连接的一侧，另一侧是 HASH 分区表。而 BY N_DEST 方式下，连接的两侧按照约定的目标线程个数进行数据划分，N 的取值由并行度决定，一般是成对出现在连接的两侧孩子中。

5. DIRECT：直接发送给某个接收线程。如果接收线程数为 1，那么所有的数据全部发给了同一个线程；如果接收线程个数大于 1，则 ESEND 逐一发送给每个接收线程。

6. BROADCAST：每一批数据都向所有接收线程发送。

● 接收操作符 ERECV

ERECV 操作符用于接收某个 ESEND 操作符的输出结果，并将接收的结果向上层操作符传递。

● 数据迭代操作符 GI

DM8 分布计算集群

GI(Granule Iterator)操作符用于控制工作线程如何获取表中的数据，访问的粒度可以是单个子表，也可以是子表中的部分页。当数据来源是分区表且存在分区列条件过滤时，GI 会剔除不满足条件的分区，即进行分区裁剪优化。

GI 的数据访问粒度分为 RANDOM、PART_UNIT (LV2_PART_UNIT) 和 USE_SQC_NO (LV2_SQC_NO)。RANDOM 表示粒度可以任意划分，既能以子表为单位也可以是子表中部分页为单位；后面几种粒度的访问策略都是以单个子表为单位。PART_UNIT 和 USE_SQC_NO 的区别在于后者通常出现在分区智能连接 (PWJ)，工作线程访问哪一张子表已经事先确定不能随意挑选。

PART_UNIT 和 LV2_PART_UNIT 的区别是，后者针对二级分区表，且叶子子表分布方式一致，此时单个线程会将中间主表下所有相同序号的叶子子表作为一个整体进行全部扫描。例如：二级分区表 T1，中间层有 P1、P2、P3 三个子表，每个中间层表 Pi 有 S1~S4 个叶子子表，如果采用了 LV2_PART_UNIT 的粒度策略，t1.p1.s1、t1.p2.s1 和 t1.p3.s1 会交由同一个工作线程扫描处理。LV2_PART_UNIT 的出现通常是因为连接、分组条件只用到了二级分区列。

USE_SQC_NO 和 LV2_SQC_NO 的区别与 PART_UNIT 和 LV2_PART_UNIT 的区别一样，也是一个线程扫描所有中间主表下相同序号的叶子子表。

5.5 DMDPC 的计划生成

DMDPC 的执行计划是在单机数据库执行计划的基础上插入了数据交换操作符 ESEND/ERECV 后形成的。同 DM MPP 架构类似，数据交换操作符的插入只发生在特定的操作符中。例如：连接、分组、排序等。

在处理每一个单机操作符时，优化器会考虑各种可能的数据分发方式。例如：哈希内连接，既可以左侧广播右侧无任何操作，也可以两侧同时按照连接列进行分布，当然如果连接两侧数据已经按照连接条件分布好了，那么可以直接连接不用任何数据交换操作符。分组、

DM8 分布计算集群

排序等也是一样，优化器穷举各种可能的路径，根据代价模型依次计算代价值，最后挑选出最小代价。

用户可以通过 EXPLAIN 命令查看执行计划，通过 10053 TRACE 信息（需设置 10053 trace event）观察到计划的优化过程。

以下是一个例子，从中我们可以观察到连接、分组和排序是如何挑选分布式计划路径的。

一，使用 EXPLAIN 查看语句的执行计划：

```

explain

select TOP 10

l_orderkey,

sum(l_extendedprice*(1-l_discount)) as revenue,

o_orderdate,

o_shippriority

from customer, orders, lineitem

where c_mktsegment = 'BUILDING'

and c_custkey = o_custkey

and l_orderkey = o_orderkey

and o_orderdate < date'1995-03-15'

and l_shipdate > date'1995-03-15'

group by l_orderkey, o_orderdate, o_shippriority

order by revenue desc, o_orderdate;
  
```

得到的计划如下：

```

1  #NSET2: [579065, 10, 116]

2  #ERECV: [579065, 10, 116]; stask_no(-1), l_stask_no(3), n_key(0), recv_in_turn(0)

3  #ESEND: [579065, 10, 116]; stask_no(3), type(DIRECT), sites(2:1), sql_invoke(0),
  
```

DM8 分布计算集群

```

table(-)
4      #PRJT2: [579065, 10, 116]; exp_num(4), is_atom(FALSE)
5      #ERECV: [579065, 10, 116]; stask_no(3), l_stask_no(2), n_key(2),
recv_in_turn(0)
6      #ESEND: [579065, 10, 116]; stask_no(2), type(DIRECT), sites(2:19, 3:19, 4:19, 5:19),
sql_invoke(0), table(-)
7      #SORT3: [579065, 10, 116]; key_num(2), is_distinct(FALSE), top_flag(1),
is_adaptive(0)
8      #HAGR2: [341666, 15221235, 116]; grp_num(3), sfun_num(1);
slave_empty(0) keys(DMTEMPVIEW_16778491.TMPCOLO, DMTEMPVIEW_16778491.TMPCOLI,
DMTEMPVIEW_16778491.TMPCOL2)
9      #PRJT2: [103926, 15221235, 116]; exp_num(4), is_atom(FALSE)
10     #HASH2 INNER JOIN: [103926, 15221235, 116]; KEY_NUM(1);
KEY(ORDERS.O_ORDERKEY=LINEITEM.L_ORDERKEY) KEY_NULL_EQU(0)
11     #ERECV: [19597, 6997382, 80]; stask_no(2), l_stask_no(0), n_key(0),
recv_in_turn(0)
12     #ESEND: [19597, 6997382, 80]; stask_no(0), type(HASH), sites(2:19,
3:19, 4:19, 5:19), sql_invoke(0), table(LINEITEM)
13     #HASH2 INNER JOIN: [19597, 6997382, 80]; KEY_NUM(1);
KEY(CUSTOMER.C_CUSTKEY=ORDERS.O_CUSTKEY) KEY_NULL_EQU(0)
14     #GI: [710, 1540563, 52]; policy(USE_SQC_NO), gi_unit[0..0]
15     #HFLKUP2: [710, 1540563, 52]; (CUSTOMER)
16     #SLCT2: [710, 1540563, 52]; CUSTOMER.C_MKTSEGMENT =
'BUILDING'

```

DM8 分布计算集群

```

17          #HFSEK2: [710, 1540563, 52]; (CUSTOMER),
scan_type(EQU) ['BUILDING', 'BUILDING']

18          #ERECV: [16149, 37598752, 28]; stask_no(0), l_stask_no(1),
n_key(0), recv_in_turn(0)

19          #ESEND: [16149, 37598752, 28]; stask_no(1), type(HASH),
sites(2:64, 3:64, 4:64, 5:64), sql_invoke(0), table(CUSTOMER)

20          #GI: [16149, 37598752, 28]; policy(RANDOM), gi_unit[0..0]

21          #HFLKUP2: [16149, 37598752, 28]; (ORDERS)

22          #SLCT2: [16149, 37598752, 28]; ORDERS.O_ORDERDATE
< 1995-03-15

23          #HFSEK2: [16149, 37598752, 28]; (ORDERS),
scan_type(L) (null, 1995-03-15)

24          #GI: [71905, 167431981, 36]; policy(USE_SQC_NO), gi_unit[0..0]

25          #HFLKUP2: [71905, 167431981, 36]; (LINEITEM)

26          #SLCT2: [71905, 167431981, 36]; LINEITEM.L_SHIPDATE > 1995-03-15

27          #HFSEK2: [71905, 167431981, 36]; (LINEITEM),
scan_type(G) (1995-03-15, max)

```

简要说明 RECV/SEND/GI 操作符后的部分字段含义。以序号 2、3、24 的操作符为例。

```
2  #ERECV: [579065, 10, 116]; stask_no(-1), l_stask_no(3), n_key(0), recv_in_turn(0)
```

该 RECV 操作符所在的子任务序号是 -1（可以理解为最根层的子任务），左孩子任务序号为 3，RECV 的 n_key 为 0，即不需要进行归并排序。

```
3  #ESEND: [579065, 10, 116]; stask_no(3), type(DIRECT), sites(2:1), sql_invoke(0),
table(-)
```

DM8 分布计算集群

该 SEND 操作符所在的子任务序号是 3，数据发送方式是 DIRECT，当前子任务会被分配到 RAFT_ID 为 2 的实例执行，并行度为 1。

```
24          #GI: [71905, 167431981, 36]; policy(USE_SQC_NO), gi_unit[0..0]
```

该 GI 的数据访问粒度是 USE_SQC_NO，意味着以子表为单位扫描，控制所在子任务的第 0 个 SCN/SEK 操作符。

由上可看出，要想了解每一个子任务会在哪些站点执行，并行度是多少，只要观察子任务的根操作符 sites(M:N) 字段即可，子任务的根操作符可以是 SEND、SPOOL 和 HEAP TAB。sites(1:16,2:16,3:16) 表示该子计划会在 RAFT_ID 为 1、2、3 的站点执行，每个站点上并行度为 16。

二，使用 10053 trace 文件查看执行计划生成。

下面截取 10053 trace 文件中和 DMDPC 计划生成相关的执行计划。

```
*** PHASE F STARTED...

<<处理哈希内连接

*** probe best distribute method for hash inner join[0x7f38a42db178], restricts[0x18], self cost
10791.797

<<各种通讯代价

left motion cost: broadcast = 1909.954, dis = 477.488, gather = 477.488

right motion cost: broadcast = 100135.803, dis = 25033.951, gather = 25033.951

<<各种分发方式的路径探测

> try L NO R NO(401), not available

> try L BROADCAST(403), cost 122237.445, n_parallel 64, best*

> try R BROADCAST(404), cost 6408691.754, n_parallel 64

> try L DIS R DIS(405), cost 25511.830, n_parallel 64, best*

> try R DIS ONLY(407), cost 25034.341, n_parallel 64, best*
```

DM8 分布计算集群

> try L GAT R GAT(408), cost 36303.237, n_parallel 0

*** probe best distribute method for hash inner join[0x7f38a42da8b8], restricts[0x18], self cost
19362.614

<<各种通讯代价

left motion cost: broadcast = 103365.639, dis = 25841.410, gather = 25841.410

right motion cost: broadcast = 143708.022, dis = 35927.006, gather = 35927.006

<<各种分发方式的路径探测

> try L NO R NO(401), not available

> try L BROAD THD DIS(402), cost 6615401.259, n_parallel 64, best*

> try L BROADCAST(403), cost 6615401.259, n_parallel 64

> try R BROADCAST(404), cost 9197313.800, n_parallel 64

> try L DIS R DIS(405), cost 61768.806, n_parallel 64, best*

> try L DIS ONLY(406), cost 25841.800, n_parallel 64, best*

> try L GAT R GAT(408), cost 81131.029, n_parallel 0

<<处理哈希分组

*** probe best distribute method for group[0x7f38a42d8d38], restricts[0x1f], self cost
944515.546

left motion cost: broadcast = 167297.602, dis = 41824.401, gather = 41824.401

self motion cost: broadcast = 167297.602, dis = 41824.401, gather = 41824.401

> try X ONLY(410), cost 944515.546, n_parallel 0, best*

> try X DIS X(411), cost 49267.428, n_parallel 64, best*

> try X GAT X(412), cost 990029.460, n_parallel 64

> try GAT X(413), cost 986339.946, n_parallel 0

DM8 分布计算集群

```
> try DIS X(414), cost 45577.914, n_parallel 64, best*
```

<<处理排序

```
*** probe best distribute method for order[0x7f38a42d8458], restricts[0x45], self cost
943452.726
```

<<各种通讯代价

```
left motion cost: broadcast = 167297.602, dis = 41824.401, gather = 41824.401
```

```
self motion cost: broadcast = 0.028, dis = 0.007, gather = 0.007
```

<<各种分发方式的路径探测

```
> try X ONLY(410), not available
```

```
> try GAT X(413), cost 985277.126, n_parallel 0, best*
```

```
> try X MERGE(417), cost 12171.229, n_parallel 0, best*
```

<<经过 DMDPC 阶段后的最终计划, ESEND/ERECV 是新增的

```
*** BEST PLAN FOR THIS STATEMENT AFTER PHF ***
```

```
ERECV[0x7f38a457a708] (cost: 2007335.29974, rows: 10);
```

```
ESEND[0x7f38a457a0e8] send_type(DIRECT) sites(2:1) (cost: 2007335.29974, rows: 10);
```

```
project[0x7f38a42d7a28] (cost: 2007335.29974, rows: 10);
```

```
ERECV[0x7f38a45797d8] (cost: 2007335.29974, rows: 10);
```

```
ESEND[0x7f38a45791b8] send_type(DIRECT) sites(2:64, 3:64, 4:64, 5:64) (cost:
2007335.29974, rows: 10);
```

```
order[0x7f38a42d8458] (cost: 2007335.29974, rows: 10);
```

```
group[0x7f38a42d8d38] (cost: 1063882.57389, rows: 60491120);
```

```
ERECV[0x7f38a4578738] (cost: 119367.02828, rows: 60491120);
```

```
ESEND[0x7f38a4578118] send_type(N_DEST) sites(2:19, 3:19, 4:19, 5:19) (cost:
119367.02828, rows: 60491120);
```

DM8 分布计算集群

```

project(DMTEMPVIEW_16778402) [0x7f38a42d9768] (cost: 119367.02828,
rows: 60491120);

hash inner join[0x7f38a42da8b8] (cost: 119367.02828, rows: 60491120);

ERECV[0x7f38a4577000] (cost: 28098.98457, rows: 54193363);

ESEND[0x7f38a45769e0] send_type(HASH) sites(2:19, 3:19, 4:19, 5:19)
(cost: 28098.98457, rows: 54193363);

select[0x7f38a42dec20] (cost: 28098.98457, rows: 54193363);
(ORDERS.O_ORDERDATE < 1995-03-15)

hash inner join[0x7f38a42db178] (cost: 28098.98457, rows:
54193363);

base table[0x7f38a42dc5d0] (CUSTOMER, INDEX33556384,
EQU SEARCH) (cost: 710.93447, rows: 1540563); (CUSTOMER.C_MKTSEGMENT = 'BUILDING')

ERECV[0x7f38a4575a50] (cost: 16596.25267, rows:
149999999);

ESEND[0x7f38a4575430] send_type(HASH) sites(2:64, 3:64,
4:64, 5:64) (cost: 16596.25267, rows: 149999999);

base table[0x7f38a42dd2d8] (ORDERS, INDEX33555914,
FULL SEARCH) (cost: 16596.25267, rows: 149999999);

base table[0x7f38a42ddf18] (LINEITEM, INDEX33555456, G SEARCH)
(cost: 71905.42958, rows: 167431981); (LINEITEM.L_SHIPDATE > 1995-03-15)

```

5.6 子计划分割与调度

子计划分割以 ESEND/ERECV 操作符为界限分割原始计划得到一系列的子计划。每个子计划之间相互独立执行，有各自的并行度，采用不同的线程。

DM8 分布计算集群

子计划一般以 ESEND 操作符为根节点，以 ERECV 为叶子操作符节点。一个子计划最多包含一个 ESEND 操作符，可以没有 ERECV 操作符，也可以有一个或多个 ERECV 操作符。

相邻的一对 ESEND/ERECV 操作符所在的子计划被称为父子关系子计划，因为 ESEND 所在的子计划发送的数据是要依靠 ERECV 所在的父亲子计划来接收的，二者之间存在着依赖关系，可以称为 Parent SPLAN 和 Child SPLAN。一个 SPLAN 最多只有一个 Parent SPLAN，但是可以有零个、一个或多个 Child SPLAN。

父子关系子计划是一个相对的概念，某个子计划 A 相对于子计划 B 是其父亲子计划，但是相对于子计划 C，有可能是其孩子子计划。

下面是一个完整的子计划划分例子：

```
create table t1(id int, c2 int) partition by hash(id) partitions 3;
create table t2(id int, d2 int) partition by range(id) (
partition p1 values less than (10),
partition p2 values less than (20),
partition p3 values less than (30));
```

假定 T1, T2 数据分布如下图 5.2 所示：

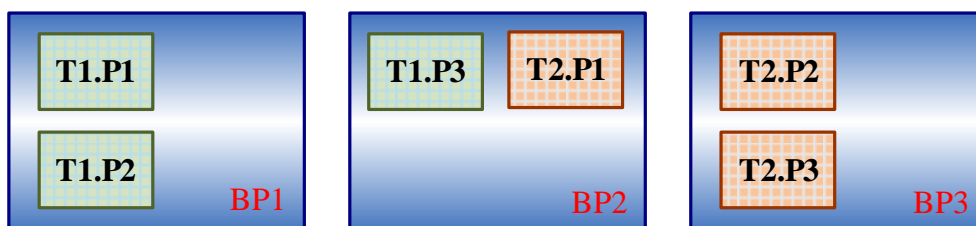


图 5.2 T1 和 T2 表的数据分布

查询 Q1:

```
select count(*) from t1, t2 where c2 = d2 and t1.id = 10 and t2.id between 15 and 25;
```

假设 t1.id=10 位于 P1 分区，执行时计划可以是这样的：

DM8 分布计算集群

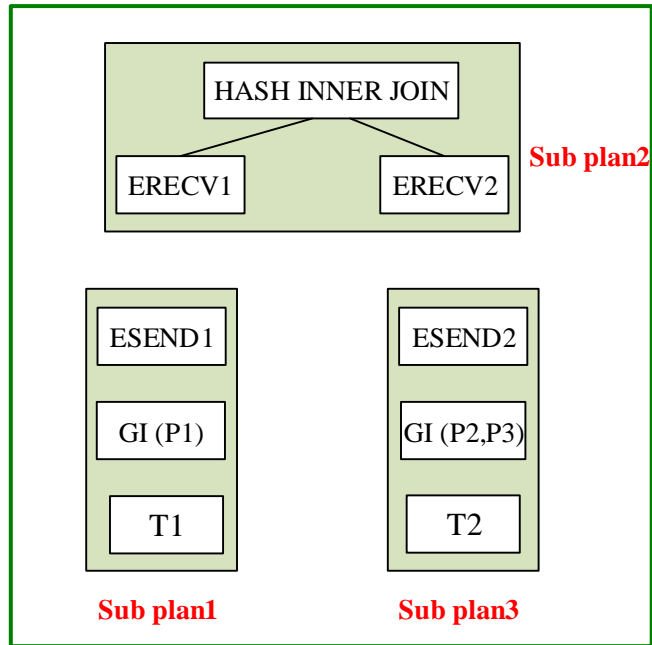


图 5.3 Q1 语句的执行计划

调度顺序是从无依赖的子任务开始，大体按照中序遍历顺序。当 ESEND 操作符第一次输出数据时，SQC 通知 QC 调度它所依赖的父层子任务。

上述计划的执行顺序是：

第一步，QC 通知调度执行 Sub plan1 和 Sub plan3。Sub plan1 附加信息：Sub plan1 的并行度=1，需要访问 T1.P1 分区，执行的 BP 节点是 BP1；Sub plan3 附加信息：t2.p2, t2.p3@BP3，并行度=2；

第二步，ESEND1 第一次对外输出数据时，向 QC 发送请求调度执行 Sub plan2。Sub plan2 附加信息：Sub plan2 的并行度=4，需要访问 T2.P2 和 T2.P3 分区，执行的 BP 节点是 BP3；ESEND2 发送数据后由于 ERECV2 没有接收数据（此时 sub plan2 忙于执行 ERECV1 和 HASH 表构造），ESEND2 也会暂时挂起；

第三步，Sub plan1 完成后，sub plan2 和 sub plan3 协同工作直至 sub plan3 和 sub plan2 分别完成。

ESEND1 和 ESEND2 的分发方式是 BY N_DEST(4)；

ESEND1_pl: 1->4，1 个 sender 线程发送给 4 个 recver 线程；

DM8 分布计算集群

ESEND2_p2: 2->4, 2 个 sender 线程发送给 4 个 recver 线程;

为了解决 ESEND 发送过快而 ERECV 来不及处理的情况, DMDPC 引入了流量控制管理, 详细信息参考 [5.8 链路通讯](#)。

5.7 生产者、消费者并行执行模型

DM MPP 并行执行模型和本地并行处理 LPQ 并行执行模型, 均属于一种对称计划设计, 即在单机计划基础上插入了数据交换操作符, 但是计划仍然是一个整体。

DMDPC 使用的生产者、消费者并发执行模型。该模型将父子任务分别视为消费者子任务、生产者子任务。

生产者子任务的角色是生产数据, 只要有数据的存放空间, 生产动作可以一直进行下去直至生产结束。

消费者子任务的角色是消费数据, 只要有新的数据进来, 消费动作可以一直进行下去。

当生产者、消费者子任务被同时调度起来后, 数据就可以在二者之间形成类似流水线上的协同操作, 两组线程各司其职, 互相配合极大提升查询性能。我们称互相协作的一组消费者、生产者子任务为形成了一组流水。

该执行模型弱化了实例间和实例内部线程的数据交换区分, 统一用 ESEND、ERECV 处理, 简化了此前的 DM MPP, LPQ 的两层并行执行模型。

如上一小节中的计划, Sub Plan1 和 Sub Plan2 被同时调度后形成一组流水, 当 Sub Plan1 执行完成后, Sub Plan2 和 Sub Plan3 协同工作, 形成新的流水。在实际调度执行中, Sub Plan1 和 Sub Plan3 可能被同时调度, 这样可以提高并行效率。Sub Plan3 发送的数据被缓存在 Sub Plan2 的数据接收缓冲区内, 在需要时, Sub Plan2 可以直接从本地的数据缓冲区获取到 Sub Plan3 发送的数据而无需调度并等待 Sub Plan3 的执行。

不同的子任务允许有不同的并行度, 同一个子任务在不同 BP 上的并行度也可以不同, 并行度设置的灵活性能大大地提升线程资源的利用效率。

用户也可以通过查询 `V$DPC_STASK_THRD` 来了解子任务的执行情况。

5.8 链路通讯

系统使用 XMAL 系统管理各部件节点网络信息，建立节点间的通讯机制。用户只需要将 SP 和 BP 节点的 IP 地址和端口等信息注册到 MP 上即可，使得增删节点的部署能够做到快速且便捷。

单个查询的各子计划按照数据驱动进行调度，尽管存在调度的先后顺序，但大多数情况会并行执行。数据由下往上传输，可能造成上层子计划的数据来不及处理而堆积。例如：客户端执行查询后并不把所有数据都取完，而 BP 节点还在不断往 SP 节点发送数据，或者连接查询子计划需要左孩子全部取完数据后，才会取右节点数据，此时右节点也可能产生数据堆积。

为了解决上述数据堆积问题，DMDPC 内引入一种限流机制，当参数 `MPP_MOTION_SYNC` 不为 0 时开启。数据发送的目的地是消息盒子。在消息盒子接收端增加一种由信号量控制的授权机制，对接收的数据量进行控制。发送端发送数据前，需要向目标消息盒子发送请求，获取发送授权。如果消息盒子的资源不足，表示已经堆积，不再授权后续的发送。为了提高授权的效率，每次请求一批资源集中进行授权。

5.9 计算与存储分离

为了克服传统数据库系统中将计算和存储耦合在一起导致的瓶颈问题，DMDPC 采用了计算与存储分离的架构。将数据存储存储在 BP 上，并将数据获取相关的子计划放在 BP 上执行。将中间层计算相关的子任务放在一个或多个 SP 上执行。

当连接和分组、去重等操作需要用到多个计算节点时（例如：连接时两侧分发 `L_DIS_R_DIS` 或者两阶段聚合分组 `X_DIS_X`），优化器会从 DMDPC 集群选择多个 SP 节点，具体规则如下：

DM8 分布计算集群

1. 假设应用登录的 SP 节点 S1 所属 SP 组为 G1，选择 G1 下所有的 SP 节点；
2. 若集群中未定义任何 SP 组，选择和当前计划涉及的 BP 节点在同一台机器上的 SP 节点；
3. 如果按上一条规则选择的 SP 节点数过少（不足 4 个或者远低于 BP 节点个数），选择集群中所有 SP 节点，最多 16 个。

5.10 动态增删节点

在 DMDPC 使用过程中，可对节点进行动态增删。动态增删分为两种情况：一是横向增删 SP 或 BP 节点。二是纵向增删 MP 或 BP 多副本系统中的副本节点。

5.10.1 横向增删

横向增删是指增删 DMDPC 系统中的 SP 单机节点或 BP 单机节点的操作，以此来扩大和缩小集群规模，灵活应对业务变化。MP 不支持横向增删。单机 DMDPC 或多副本 DMDPC 中均支持横向增删。

横向增删既可以登录到 SP 上执行，也可以登录到 MP 上执行。

如果现有主机有可用的硬件资源，可以添加新的 SP、BP 节点到当前集群。新节点直接部署到已有的主机上。下图展示了一个 DMDPC 集群含有 2 个 SP、1 个 MP 和 2 个 BP，现通过横向增加 1 个 SP，增加 1 个 BP，将其改变成为一个含有 3 个 SP、1 个 MP 和 3 个 BP 的集群。

DM8 分布计算集群

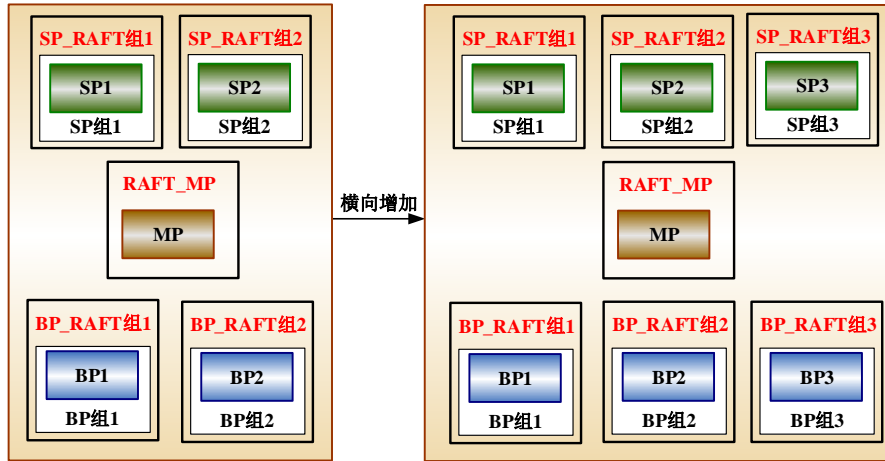


图 5.4 横向增加节点示意图

添加新的 BP 节点后，后续数据存储便可以利用这些 BP。由于 BP 上并不持有元数据信息，因此新加入的 BP 节点可以马上提供服务。新加入的 SP 节点可以用于响应更多的客户请求，分担子计划的执行。

支持动态增删的内容如下：

一 支持删除 SP 组、BP 组、RAFT 组、BP 节点或 SP 节点。删除节点时需满足下列要求：

1. 只有正常退出的 SP 节点才能删除；
2. 没有数据的 RAFT 组内节点才能删除。如果想删除包含数据的 BP 节点，必须先将此节点上的数据迁移到其他分区，具体见 [8.7 数据迁移](#)。
3. 删除 RAFT 组必须先删除组内包含的所有节点。

二 支持修改 SP 和 BP 节点中的部分配置：ap_port、inst_port 和 ip_addr。

5.10.2 纵向增删

纵向增删是指对 MP 多副本系统或 BP 多副本系统中的副本节点进行增删。因为 SP 不支持多副本，所以 SP 不支持纵向增删。多副本系统内的节点数必须是大于等于 3 的奇数，因此一次性增删节点的个数必须为偶数，并且增删之后该 RAFT 组内节点总数依然要大于等于 3。

不支持对多副本系统中的主节点进行增删。

纵向增删 MP 副本节点，需要登录到 MP 主节点上执行。纵向增删 BP 多副本节点，需

DM8 分布计算集群

要登录到以 BS 模式运行的 BP 主节点上执行。

下图展示了一个 DMDPC 集群含有 2 个 BP 节点，其中一个 BP 配置成 3 副本，另外一个 BP 配置成 5 副本。通过纵向增删节点之后，一个 BP 变成了 7 副本，另外一个 BP 变成了 3 副本。

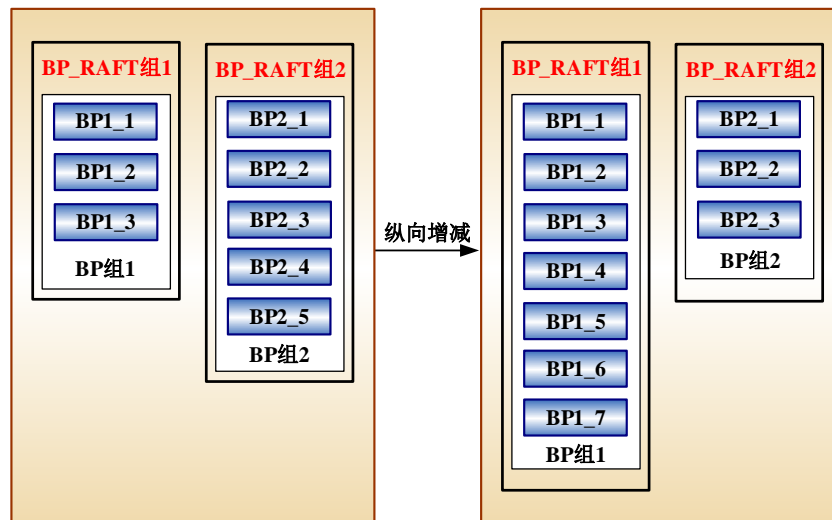


图 5.5 纵向增删节点示意图

5.11 分布式事务一致性

在分布式系统架构下，不同的服务器之间通过网络远程协作而完成的事务，称为分布式事务。

DMDPC 通过两阶段提交技术来保证多个 BP 之间的分布式事务一致性。两阶段提交的参与者为 SP 和 BP。SP 作为全局事务的协调者，统一处理全局事务。BP 作为参与者，是被 SP 调度并执行事务的节点。SP 根据 BP 的响应来决定是否真正的执行并提交事务。所有参与者 BP 要么一起提交要么一起回滚，始终保持事务一致性状态。

DMDPC 是否采用两阶段提交由 INI 参数 DPC_2PC 来控制。

两阶段分为预提交阶段和提交阶段。

5.11.1 第一阶段 预提交

1. 客户端向 SP 发起事务提交请求。
2. SP 向所有参与者 BP 广播预提交命令。询问是否可以进行事务 COMMIT，并等待 BP 的响应。
3. BP 接收到预提交命令之后，将相关操作生成回滚日志写入回滚段并响应 SP。

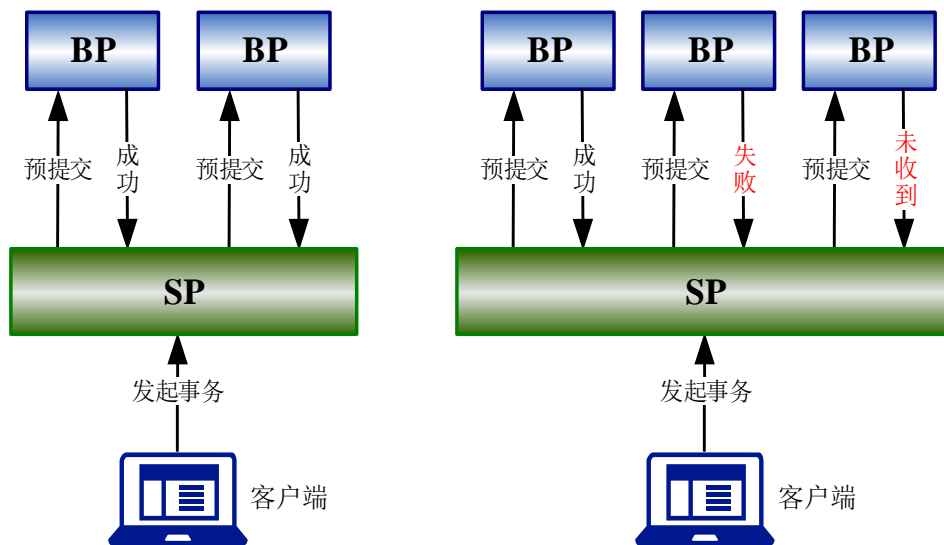


图 5.6 预提交流程（左成功/右失败）

4. 如果 SP 收到所有参与者 BP 的预提交成功的消息，则进入第二阶段进行提交。否则，当存在 BP 没有响应导致 SP 无法确定该 BP 是否执行了预提交时，SP 先尝试通知存活 BP 执行回滚。若有 BP 回滚成功，SP 就可以直接响应“事务提交失败”给客户端；否则 SP 只能响应“未知的提交结果”给客户端。客户端不论是收到“事务提交失败”还是“未知的提交结果”，都表示事务执行失败。

DM8 分布计算集群

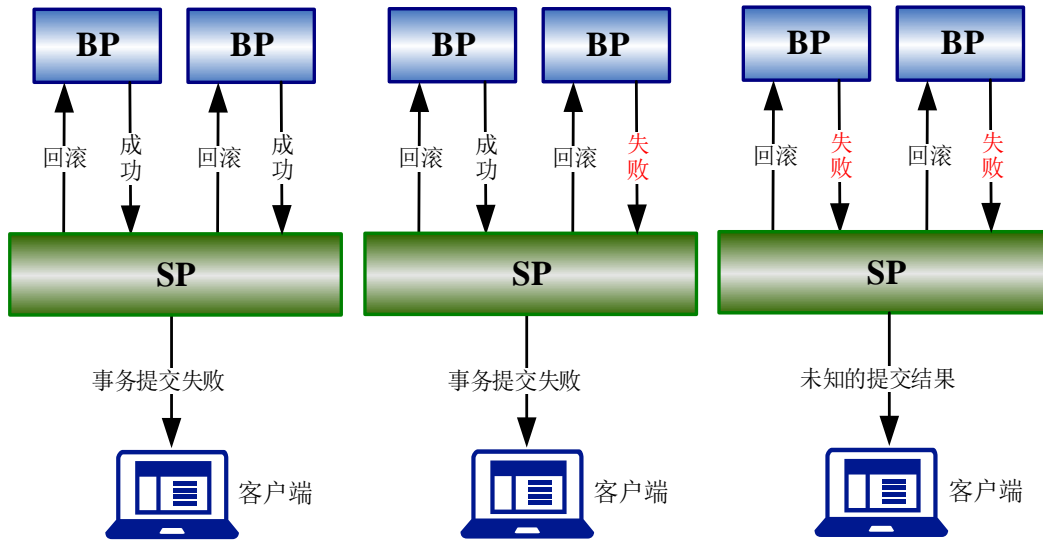


图 5.7 回滚流程

5.11.2 第二阶段 提交

如果 SP 收到所有参与者 BP 的预提交成功的消息，那么说明可以进入提交阶段。

1. SP 向所有参与者 BP 广播 COMMIT 命令。
2. BP 接收到提交命令之后，执行二阶段的提交任务，释放事务资源，并将 COMMIT

成功结果反馈给 SP。

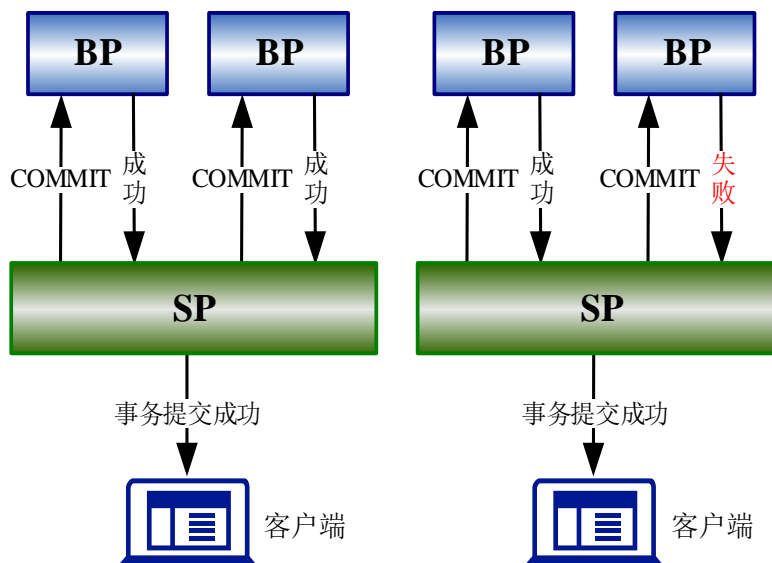


图 5.8 提交流程

3. SP 收到所有参与者 BP 的 COMMIT 成功消息之后，直接响应客户端事务提交成功。

DM8 分布计算集群

二阶段提交时若遇到 BP 与 SP 通信中断，SP 会将提交任务交由异步任务进行处理并立即响应客户端事务提交成功。待故障 BP 与 SP 重新建立连接后，异步任务会自动重新执行二阶段提交，直到提交成功。客户端收到事务提交成功的消息，表示事务执行成功完成。

5.12 分布式事务的数据可见性

在分布式集群环境下，事务通常会跨多个 BP 节点，不同 BP 节点修改数据后执行提交操作的时间差将无法保证各节点上事务的隔离性。为此，DM 借助全局时钟系统（GTS）来帮助 BP 节点对数据的数据的可见性进行判断，进而保证各节点上事务的隔离性。

全局时钟值由 MP 节点统一管理。BP 节点执行事务内的每条语句、执行预提交、提交操作时均会向 MP 申请当前的全局时钟值。

5.12.1 CA

CA 是 BP 用来存放事务信息的数组，CA 是一个全局变量。CA 中登记的事务信息如下：

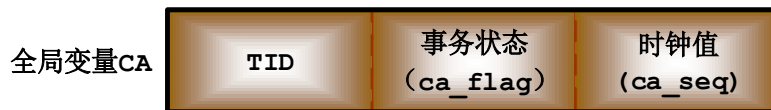


图 5.9 CA 数组结构图

BP 节点执行事务的执行预提交、提交操作时均会向 MP 申请当前的全局时钟值。然后，BP 将执行操作时的事务号 TID、事务状态和时钟值登记在数组 CA 中。事务状态分为预提交和提交两种，分别表示一阶段预提交和二阶段提交。

事务的 TID 是全局唯一且不变的。当事务从预提交状态进入提交状态，CA 中的事务状态和时钟值会进行同步更新。

5.12.2 MCA

MCA 是 MP 用来存放事务信息的数组。MCA 中登记的事务信息如下：



图 5.10 MCA 数组结构图

当事务执行提交操作向 MP 申请全局时钟值时，MP 会将该事务的 TID 和当前时钟值登记在 MCA 中。MCA 中登记的事务均为已经执行过提交操作的事务。

5.12.3 事务信息登记流程

1. 当事务对数据进行修改时，日志记录中会登记修改该数据的事务号 TID。
2. 当事务执行一阶段预提交时，CA 中会登记 TID、预提交状态和预提交时的时钟值。
3. 当事务二阶段提交时，MCA 会登记 TID 和提交时的时钟值，CA 会将第 2 步中登记的信息更新为最终的提交状态和提交时的时钟值。

整体流程如下：

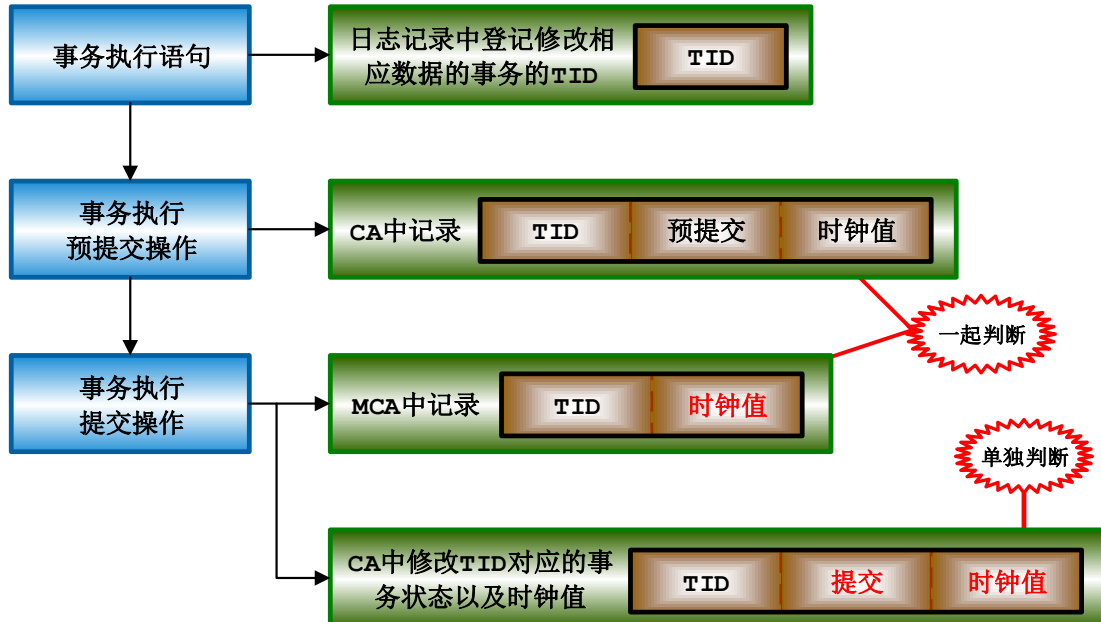


图 5.11 事务信息登记流程图

5.12.4 数据可见性判断

事务对数据进行操作时，BP 都会向 MP 申请当前的全局时钟值 `cur_seq`。只要是当前事务操作时钟值 `cur_seq` 之前的已提交事务修改的数据，对当前事务都是可见的。

BP 节点判断某一数据的可见性时，首先找到日志记录中登记的修改该数据的事务 TID，通过 TID 在 CA 中查询到对应的事务状态 `ca_flag` 和时钟值 `ca_seq`，然后将 `ca_seq` 与 `cur_seq` 进行比较，最后通过比较结果和 `ca_flag` 来综合判断出数据的可见性。

具体可见性判断原则如下：

1. 在 CA 中判断

如果 `ca_seq ≤ cur_seq`，且 `ca_flag` 为“提交”，说明生成该数据的事务在 `cur_seq` 之前就已提交，则该数据对当前事务可见。

如果 `ca_seq ≤ cur_seq`，但 `ca_flag` 为“预提交”，说明生成该数据的事务在 `cur_seq` 之前已经成功预提交。那么需要结合 MCA 来进一步判断该事务是否已提交，进入下一步 MCA 中继续判断。

2. 在 MCA 中判断

如果 MCA 中找到了对应的 TID 和时钟值 `mca_seq`，且 `mca_seq ≤ cur_seq`，则该数据对当前事务可见。

3. 其他情况均为不可见。

5.13 自动选举主库

多副本系统中，基于 RAFT 协议的选举规则，结合达梦 REDO 日志包的特点，制定了一套适合达梦多副本系统的选举规则，各节点实例根据此规则自动选举出领导者（Leader）和跟随者（Follower），其中 Leader 角色的实例会自动切换为主库模式，Follower 角色的实例会自动切换为备库模式。

DM8 分布计算集群

选举是集群废黜旧 Leader 产生新 Leader 的过程，其作用是使集群在旧 Leader 故障后及时产生新 Leader。Leader 选举成功后，会定时发送心跳消息到其他节点，如果某个节点在选举超时的间隔内没有收到 Leader 的消息就会认为其故障，并会发起新的选举。

每次发起选举时，节点会切换为 Candidate 角色，并将自己的任期号加 1，从而废黜旧 Leader，然后向其余节点发送投票请求。选举过程通过投票完成，每个任期每个节点有且仅有一张选票，每个节点只会投给与自己相比符合选举要求的节点（有效日志比自己多）。发起选举的节点会直接投给自己，其余节点先收到的投票请求先处理，符合要求就会直接投出选票，不会考虑之后的投票请求。当一个节点收到超过半数（包括自己）的选票，就会成为这个新任期的 Leader。

被选举为 Leader 的数据库实例，会自动将自己切换到主库模式和 Open 状态，其他实例则会自动将自己切换到备库模式和 Open 状态，然后整个多副本系统继续正常对外提供服务。

5.14 RAFT 归档

多副本系统中的主库通过 RAFT 归档方式向备库同步数据。

与本地归档写入保存在磁盘中的日志文件不同，RAFT 归档将主库产生的 Redo 日志通过 XMAL 模块传递到备库，RAFT 归档是多副本系统的实现基础，RAFT 归档只在主库生效，一个主库可以配置 2~8 个 RAFT 备库，归档目标个数必须是偶数（确保总的实例个数是奇数）。

RAFT 归档的执行流程是主库在 Redo 日志（RLOG_PKG）写入联机日志文件前，将 Redo 日志发送到备库，并且不需要等待备库的响应消息，主库继续往下正常执行。备库收到 Redo 日志（RLOG_PKG）后，将日志包加入日志重演任务系统，在日志包写入本地日志文件后，发送日志刷盘消息给主库，主库根据此消息确定是否需要推进 C_SEQNO 和 C_LSN。

5.15 自动同步日志

在多副本系统中，主库通过 XMAL 模块自动将日志包发送给备库，备库在日志刷盘完成后发送刷盘消息给主库，主库在收到多数备库的刷盘消息后，向前推进已提交到的日志包序号和 LSN。

日志的发送与接收，详细介绍如下：

日志发送

主库产生的日志，以日志包为单位发送给备库重演。

每个日志包上会记录日志产生时的任期号以及一个全局递增的日志包序号 `G_SEQNO`，通过任期号和 `G_SEQNO` 可以唯一确定一个日志包。

主库每产生一个新的日志包后，就会立即并行、异步地将日志包发送给其余备库节点。如果日志发送失败或者备库校验日志失败，则将对对应备库节点置为失效状态，并对其启动故障处理流程。如果主库发现自己过时（其他节点选举出来新主库），则停止发送日志并将自己切换为备库（Follower）模式。

日志接收

多副本系统中，备库收到日志包后会进行有效性校验。先校验是否是当前任期内的主库发送的日志，如果发现日志包来自旧主库，会通知其已经过时，再校验日志是否连续，如果发现日志缺失或者失效，会通知主库校验失败，由主库后续启动异步恢复流程。

备库会对收到的日志包进行排序、缓存，逐个交给日志重演系统进行重演，避免日志包乱序重演导致校验失败。日志重演系统会重构、拆分收到的日志包，采用并行方式重演日志，在日志包刷盘成功后会发送消息通知主库自己的刷盘信息。

5.16 C_SEQNO/C_LSN 维护

在多副本系统中，`C_SEQNO` 是已经提交的日志包序号，`C_LSN` 是已经提交的

DM8 分布计算集群

C_SEQNO 日志包中的最大 LSN。日志包只有在超过半数（包括自己）节点刷盘之后才可以被提交。C_SEQNO 与 C_LSN 由主库推进，备库跟随主库调整。

主库会记录所有节点（包括自己）已刷盘完成的日志包序号和 LSN，当某一日志包在超过半数（包括自己）节点刷盘后，主库会主动将此日志包的包序号和最大 LSN 设为 C_SEQNO 和 C_LSN。主库会在日志包和心跳消息中附加当前的 C_SEQNO 和 C_LSN 发送给备库，备库收到后，根据此信息推进自己本地的 C_SEQNO 和 C_LSN 值。

备库在日志刷盘完成后，会将已刷盘的日志包的 G_SEQNO 和对应的 LSN 发给主库供主库推进 C_SEQNO 和 C_LSN 使用。

特别地，主库和备库均不会将 C_SEQNO 和 C_LSN 推到超过自己已刷盘完成的最大的日志包序号和 LSN，另外处于异步恢复中的备库不参与 C_SEQNO 和 C_LSN 的推进。

5.17 数据页刷盘和检查点推进

在多副本系统中，由于 RAFT 协议中允许对未提交的日志进行截断，因此需要对数据页刷盘和检查点推进做一定的条件限制。

要求数据页上的 LSN 必须小于或等于 C_LSN，也就是所有修改数据页产生的 Redo 日志已经提交到多数节点之后，才允许将数据页写入本地磁盘。

同样的，检查点 LSN 最多只允许推进到 C_LSN 位置，而不是本地已刷盘的最大 LSN。检查点推进过程中，可能碰到某个数据页上又有新的修改而无法刷盘的情况，此时检查点只能推进到修改此页的最小 LSN 的前一个位置，这个过程中需要将检查点从目标位置回调。

5.18 自动故障处理

在多副本系统中，在主库发生故障时，根据达梦多副本的选举规则可以安全的从活动备库中选出新主库，保证已提交的数据不会丢失；在少数备库发生故障或者出现网络延迟时，不会影响主库的正常运行。主备库的故障处理如下：

1. 主库故障处理

当出现硬件故障（掉电、存储损坏等）原因导致主库无法启动，或者是主库内部网卡故障导致主库短期不能恢复正常的情况下，剩余活动节点会自动启动选举流程，选出新的主库，其他节点仍然作为备库运行，选举完成后，多副本集群仍可对外提供服务。

选举新主库的前提条件是活动节点个数需超过配置的总节点个数的一半。也就是要保证多数节点处于活动状态，才可以正常发起选举，如果多数节点都发生故障，则暂时无法启动选举流程。

2. 备库故障处理

备库出现故障时，主库发送心跳或者日志包失败后，会将其归档状态失效，不再向故障备库同步数据。

如果只是少数备库故障，则不会影响到主库正常的日志提交动作，多副本系统仍然能够正常运行，如果多数备库都发生故障，则主库新产生的日志无法提交到多数节点，主库上的事务提交动作可能会被挂起。

5.19 故障恢复

在多副本系统中，在故障实例恢复时，主库可以自动发起异步的数据同步，最终可将备库数据同步到和主库完全一致。

主备库的故障恢复如下：

1.主库故障恢复

主库故障恢复时，多副本系统中如果已经有新主库，则会自动将老主库切换为备库重新加入多副本系统，并自动向其发起异步数据同步，直到主备库数据完全同步一致。

主库故障恢复时，如果多副本系统中没有活动主库，在多数节点都处于活动状态的情况下，则会重新发起选举，如果老主库再次选举成功，则直接将其切换为 Open 状态即可，否则仍然是作为备库重新加入多副本系统。

DM8 分布计算集群

需要注意的是，主库故障前，本地日志文件中可能已经写入有未提交成功的日志，在作为备库重新加入多副本系统时，如果这些未提交的日志在当前主库上没有找到，则需要对这些日志进行截断处理。

2. 备库故障恢复

备库故障恢复后，多数情况下仍然是作为跟随者重新加入多副本系统，除非重新加入时没有活动主库才会尝试选举为新主库。

如果发起选举流程并且选举成功，则会自动切换为主库模式及 Open 状态。

如果仍然是作为备库重新加入多副本系统，则当前主库会自动向其发起异步恢复流程，直到和主库数据完全一致后，将其归档状态设置为有效状态，然后再次回到正常的数据同步方式并参与主库的日志提交流程。

主库对备库进行异步恢复失败后，再次对其进行异步恢复的时间间隔通常为 30 秒，但是针对以下三种情况：主备库 DB_MAGIC 不一致、主备库 PMNT_MAGIC 不一致或者主库归档缺失（主库上备库故障开始位置对应的归档文件被删除），仅当对备库进行还原恢复后，主库才能成功对其进行异步恢复，因此在以上三种情况下，为避免主库重复对备库进行异步恢复造成过多的内存消耗，主库对备库进行异步恢复的时间间隔被延长为 30 分钟。

如果在备库故障期间，主库发生过多次切换，并且备库本地有未提交成功的日志，则备库异步恢复时，需要先对本地多写入的日志进行截断处理。

5.20 多副本影子库

在一个多副本集群中，可以将所有节点划分为两种：RAFT 库和影子库。RAFT 库为多副本集群中正常修改数据文件，有完整数据，可以正常提供服务的节点；多副本影子库，又称为影子库（SHADOW 库），为多副本集群中的特殊节点，不修改数据文件，没有数据，但正常参与选举和事务提交，仅提供 V\$ 动态视图查询的节点。RAFT 库是必选项。影子库为可选项，当用户存储资源不足的时候，可以部署少量影子库来代替 RAFT 库。

DM8 分布计算集群

一个多副本集群中，影子库的数量不能超过整个集群节点数量的一半。不支持对影子库节点执行动态增加和删除。

可以通过查询 `V$RLOG_RAFT_INFO` 中 `RAFT_SHADOW` 字段判断当前库是否为影子库，值为 0 表示是 RAFT 库；值为 1 表示是影子库。

可以通过配置 `SHADOW_CHECK_INTERVAL` 参数，定期清理影子库的本地归档文件。

影子库搭建的具体内容请参考 [7.5 命令行工具搭建多副本影子库](#)。

6 DMDPC 配置

6.1 相关参数

6.1.1 DM.INI

DM.INI 是 DM 数据库实例的配置文件，通过配置该文件可以设置 DM 数据库服务器的各种功能和性能选项。

6.1.1.1 DMDPC 相关的参数

DM.INI 中的配置项多达数百个，涉及 DM 数据库运行各个方面的设置，这里我们只介绍与 DMDPC 相关的几个配置项，读者若对其他配置项感兴趣可以参看《DM8 系统管理员手册》的相关章节。

表 6.1 DM.INI DMDPC 相关配置项

| 参数 | 说明 | 属性 | 缺省值 |
|---------------|--|----|----------|
| INSTANCE_NAME | 实例名称。推荐以角色开头，例如：SP_1, BP_2 | 静态 | DMSERVER |
| PORT_NUM | 数据库服务器监听端口，SP/BP 本地设置的 PORT_NUM 如果和自身在 MP 上注册的不一致，按照在 MP 上实际注册的为准，启动时会自动覆盖 | 静态 | 5236 |
| AP_PORT_NUM | 分布式环境下协同工作的监听端口，SP/BP 本地设置的 AP_PORT_NUM 如果和自身在 MP 上注册的 AP_PORT_NUM 不一致，按 | 静态 | 6000 |

DM8 分布计算集群

| | | | |
|--------------|--|--------|-------|
| | 照在 MP 上实际注册的为准，启动时会自动覆盖 | | |
| DPC_OPT_FLAG | <p>执行计划优化策略。取值 1、2、4、8、16、32、64、128、256、512、1024、2048、4096 和 8192，或上述任意取值的和。</p> <p>1: 表示忽略 UNIONALL，将分支的内容直接发给 UNIONALL 之上的 ERECV；</p> <p>2: 表示根据 HSCN 的使用情况，对 HTAB 的分布信息调整，可以减少 HSCN 之后的数据分量；</p> <p>4: 表示在通讯符之前将 afun 转换成两个，可以减少通讯量；</p> <p>8: 表示按照实际执行的 BP 个数调整子任务并行度；</p> <p>16: 根据利用统计信息最大、最小值计算相同表不同别名的选择率。例如: $tl.cl + 10 > tl.cl ==> tl.cl < \max(tl.cl) + 10$;</p> <p>32: 同时开启 SP、MP 和 BP 节点的计划缓存的功能，减少事务密集型应用中计划发送带来的耗时。非 32 的情况下，只有 SP 具有计划缓存功能；</p> <p>64: 多列 primary key 用于连接时的行数估算调整优化；</p> <p>128: 根据估算的行数对连接、分组等使用到</p> | 动态，会话级 | 16383 |

DM8 分布计算集群

| | | | |
|-------------------|---|---------|---|
| | <p>的哈希表大小进行缩放优化;</p> <p>256: 哈希连接时, 如果连接左边数据来自广播, 启用共享哈希表优化减少哈希表构造开销;</p> <p>512: huge 表考虑 hfsek 与 hfscn+slct 的代价;</p> <p>1024: 计划只从几个子任务开始, 而不是所有的叶子节点;</p> <p>2048: DISTINCT 尝试下放至集合运算的右侧的子任务先做;</p> <p>4096: afun/sagr+sort+dis 的 sort 下放到 dis 下面, 变成 afun/sagr+dis (归并信息) +sort。sort 放到 dis 下面后, 每一路可先做排序, 如果子任务并行度比 sagr/afun 的并行度大时能较快的完成排序;</p> <p>8192: esend 发送时对 link_id 进行优化。优化后, 消息可以多条链路同时发生, 而非一条链路串行</p> | | |
| ALTER_MODE_STATUS | <p>在多副本系统中使用。</p> <p>是否允许手工修改服务器的模式、状态和 OGUID。1: 允许; 0: 不允许。建议设置为 0</p> | 动态, 系统级 | 1 |

DM8 分布计算集群

| | | | |
|-------------|--|---------|---|
| DPC_2PC | <p>DMDPC 系统是否采用两阶段提交策略及是否使用优化方案。</p> <p>0: 关闭两阶段提交, 没有故障的情况下直接提交效率更高, 但是出现故障就无法保证事务一致性了;</p> <p>1: 启用两阶段提交;</p> <p>2 或 3: 启用两阶段提交, 启用 trxid 缓存, 启用 snap_seq 缓存方案优化;</p> <p>4 或 5: 启用两阶段提交, 启用 trxid 缓存, 启用 snap_seq 本地优化;</p> <p>6 或 7: 启用两阶段提交, 启用 trxid 缓存, 启用 snap_seq 缓存, 启用 snap_seq 本地优化</p> | 手动 | 1 |
| SWITCH_CONN | <p>JDBC 客户端是否进行连接切换, 执行过程中发现计划只和一个 BP 相关时, 客户端是否切换到该 BP 执行。</p> <p>0: 否; 1: 是。</p> <p>此参数需要和 JDBC 驱动/dm_svc.conf 配合使用, 只用于 BS 模式</p> | 动态, 系统级 | 0 |
| TRC_LOG | <p>TRACE 日志开关。需和 TRC_LOG_MODULE 配合使用。</p> <p>0: 表示关闭;</p> <p>非 0: 为功能组合值, 组合规则为:</p> | 动态, 系统级 | 0 |

DM8 分布计算集群

| | | | |
|-----------------------|--|---------------|------------|
| | <p>switch_mod*4 + asyn_flag*2 + 1。其中，<switch_mode>日志文件切换方式： 0 不切换；1 按文件记录行数，满 10000 行切换；2 按文件大小进行切换，满 128M 切换；3 按时间间隔进行切换，每小时切换。</p> <p><asyn_flag>日志文件是否采用异步方式： 0 否，采用同步方式；1 是，采用异步方式。</p> <p>TRACE 日志位于安装目录下的 /log 中，名称为 dmtrc_instname.log（不含切换方式）或 dmtrc_instname_日期时间.log（含有切换方式）</p> | | |
| <p>TRC_LOG_MODULE</p> | <p>当打开 TRC_LOG 开关时，用于配置 TRACE 日志中记录的内容。该参数为字符类型，最大长度 128。</p> <p>书写格式为：TRC_LOG_MODULE=服务器记录项，DMDPC 记录项。</p> <p>服务器记录项和 DMDPC 记录项设置方法相同。缺省表示不记录该模块中的任何内容。</p> <p>具体记录项和代号：内存 MEM (0),任务线程 TSK (1), 站点信息 SITE (2), 事务 TRX (3), 通信消息 MSG (4), 会话 SESS (5), 计划 PLN_GEN (6), HUGE 文件系统 HFS (7), 0~7 全部记录(ALL)。多个记录项之间使用冒号(:)分割。</p> | <p>动态，系统级</p> | <p>ALL</p> |

DM8 分布计算集群

| | | | |
|---------------------------|---|----------------|----------|
| | <p>例如:</p> <p>1) 记录 0~7 项所有的信息, 需设置</p> <p><code>trc_log_module = ALL</code></p> <p>2) 只服务器模块记录 HFS 和 TRX, DMDPC 模块不记录, 需设置 <code>trc_log_module=3:7,</code></p> <p>3) 只 DMDPC 模块记录 MSG, 服务器模块不记录, 需设置 <code>trc_log_module=.4</code></p> <p>4) 若服务器记录 HFS 和 TRX, DMDPC 记录 MSG, 需设置 <code>trc_log_module=3:7,4</code></p> | | |
| <p>DPC_LOG_INTERVAL</p> | <p>MP 定时通知各节点生成相同时间点日志的时间间隔。取值范围 0~1440 (24 小时)。以分钟为单位。0 表示关闭定时通知功能。</p> <p>此参数打开后, 可支持使用归档恢复到指定时间点功能, 在对系统性能有要求的情况下, 可以调大或关闭此参数值</p> | <p>动态, 系统级</p> | <p>0</p> |
| <p>SQC_GI_NUM_PER_TAB</p> | <p>每个表拆分的扫描单元个数。取值范围 1~1024。</p> <p>扫描表对象时, 每个表可以被拆分成多个单元以提升并行性能。参数值表示拆分的单元个数, 每个单元可以独立用于工作线程的扫描任务。例如: 需要扫描一个子表且子表数据量较大, 为充分利用工作线程资源, 可以</p> | <p>动态, 会话级</p> | <p>1</p> |

DM8 分布计算集群

| | | | |
|---------------------|---|--------|----|
| | 调大此参数以使用分区内并行 | | |
| HP_TAB_COUNT_PER_BP | <p>创建一级、二级哈希分区时，如果分区数指定为 DEFAULT 而不是具体数值，系统会根据本参数值决定每个 BP RAFT 上创建的哈希子表个数。</p> <p>取值范围 1~1024</p> | 动态，会话级 | 1 |
| DPC_SYNC_STEP | <p>打开 DMDPC 限流（MPP_MOTION_SYNC 非 0）时使用。</p> <p>表示检查接收端数据堆积的步长，即每发送 DPC_SYNC_STEP 次 BDTA 后检查一次是否堆积。取值范围 1~5000</p> | 动态，会话级 | 16 |
| DPC_SYNC_TOTAL | <p>打开 DMDPC 限流（MPP_MOTION_SYNC 非 0）时使用。XBOX 上所有来源路数（并行线程数）的总限流资源数。取值范围 0~10000。0 表示不限总资源数</p> | 动态，会话级 | 0 |
| XBOX_DUMP_THRESHOLD | <p>XBOX 邮件内存阈值，超过该值则保存到文件中。取值范围 0~409600。单位 MB。和 XBOX_DUMP_PATH 一起设置才生效</p> | 动态，系统级 | 0 |
| XBOX_DUMP_PATH | <p>XBOX 邮件保存到临时文件的目录。</p> <p>和 XBOX_DUMP_THRESHOLD 一起设置才生效</p> | 手动 | 空串 |
| STMT_XBOX_REUSE | <p>DMDPC 会话的语句是否重用 box_id。</p> <p>0：否，每个语句都请求分配一个 box_id；</p> | 动态，会话级 | 1 |

DM8 分布计算集群

| | | | |
|--------------------------|--|--------|------|
| | 1: 是 | | |
| XBOX_SHORT_MSG_SIZE | 在配置了 XBOX_DUMP_THRESHOLD 后，即使内存达到阈值的情况下，小于此长度的短消息也不会保存到文件，防止大量的文件读写。取值范围 0~32768。单位 MB | 系统，会话级 | 1024 |
| DPC_TRX_TIMEOUT | 用于 DMDPC 中，表示事务等待超时时间。取值范围 0~604800。单位秒 | 动态，会话级 | 10 |
| DPC_GUP_SESS_TIMEOUT | 用于 DMDPC 中，表示 SP 升级时事务、会话超时断开时间。取值范围 60~600。单位秒 | 动态，会话级 | 180 |
| DPC_TABLESPACE_BALANCE | 当在 DPC 下创建分区表，子表个数少于所选的表空间个数时，是否启用随机选择起始表空间以使得表空间负载均衡。0: 不启用；1: 启用 | 动态，系统级 | 0 |
| DPC_DCT_REFRESH_POLICY | MP 广播实例字典信息优化开关。 0: 表示关闭，MP 会定时广播实例字典信息； 1: 表示打开，MP 只会在每次修改实例字典信息时进行广播 | 动态，系统级 | 1 |
| ENET_SESS_CHECK_INTERVAL | 当 BP/SP 处于 MP 无效服务状态时，BP/SP 断开会话连接的时间间隔阈值。如果 BP/SP 处于 MP 无效服务状态的时间超过该阈值，则会主动断开所有会话连接。取值范围 0~4294967294，单位 S。0 表示关闭 | 动态，系统级 | 10 |

DM8 分布计算集群

| | | | |
|-----------------------|---|------------|------|
| ETHD_FLAG | 是否启用 ESESS 线程池，取值范围 0~1。0 表示关闭；1 表示打开 | 静态 | 0 |
| EHTD_THREAD_NUM | 每个 ESESS 线程池初始化的线程数量。取值范围 1~10000 | 静态 | 8 |
| ETHD_GRP_NUM | 初始化的 ESESS 线程池数量。取值范围 1~64 | 静态 | 8 |
| RLOG_RAFT_NEED_WAIT | 主库是否进行日志包发送等待控制，仅 RAFT 主库生效。 0：主库不进行日志包发送控制；1：任意备库存在日志堆积时，根据 RLOG_RAFT_WAIT_TIME 暂缓日志包发送；2：存在日志堆积的备库个数超过备库总数的半数时，根据 RLOG_RAFT_WAIT_TIME 暂缓日志包发送；3：任意归档状态有效的备库存在日志堆积，根据 RLOG_RAFT_WAIT_TIME 暂缓日志包发送；4：存在日志堆积的归档状态有效的备库个数超过备库总数的半数时，根据 RLOG_RAFT_WAIT_TIME 暂缓日志包发送 | 动态， 系统级 | 3 |
| RLOG_RAFT_WAIT_TIME | 备库出现堆积后，RAFT 主库延迟发送等待时间，取值范围 0~3600000，单位 MS。仅 RAFT 主库生效 | 动态， 系统级 | 1000 |
| SHADOW_CHECK_INTERVAL | 用于设置清理影子库的本地归档文件的时间 | 动态，系 | 60 |

DM8 分布计算集群

| | | | |
|-----------------------|--|---------------|-----------|
| | <p>间隔，取值范围 0~3600，单位 s。如果设为 0，则关闭自动清理功能。此参数仅对影子库有用</p> | <p>统级</p> | |
| <p>GROUP_OPT_FLAG</p> | <p>分组项优化参数开关。</p> <p>0：不优化；</p> <p>1：非 MYSQL 兼容模式下（即 COMPATIBLE_MODE 不等于 4），支持查询项不是 GROUP BY 表达式；</p> <p>2：外层分组项下放到内层派生表中提前分组优化；</p> <p>4：表示对于多级分区，并行下允许尝试不生成多个 AGR；</p> <p>8：进行哈希分组时，依赖分组项列中的核心项分组列来分组；</p> <p>16：位图索引覆盖简单分组查询中的所有列时，允许使用位图索引对查询进行优化；</p> <p>32：进行哈希分组时，对 TOP 查询进行优化，提前返回指定数量的分组；</p> <p>64：DMDPC 下，对无分组项且含有 DISTINCT 集函数的查询，允许对集函数参数先进行一次分发处理。</p> <p>支持使用上述有效值的组合值，如 3 表示同时进行 1 和 2 的优化</p> | <p>动态，会话级</p> | <p>52</p> |

DM8 分布计算集群

| | | | |
|---------------------|--|---------|--------|
| STAT_CACHE_FLAG | <p>SP 是否缓存表行数。</p> <p>0: 不缓存;</p> <p>1: SP 缓存表行数, 而不是每次都向 MP 请求表行数, 从本地缓存中获取表行数以降低 MP 的性能瓶颈。和 STAT_CACHE_CAPACITY 搭配使用</p> | 静态 | 1 |
| STAT_CACHE_CAPACITY | <p>当 STAT_CACHE_FLAG=1 时使用。SP 缓存表行数的缓存容量。取值范围 100~100000000, 单位缓存表的个数</p> | 动态, 系统级 | 1000 |
| TSMV_RAFIL_SIZE | <p>指定表空间迁移的临时归档文件的最大空间。取值范围 2~1024。单位 MB</p> | 静态 | 64 |
| TSMV_FILE_COPY_PLL | <p>表空间迁移时物理文件拷贝并行度。取值范围 1~128</p> | 动态, 系统级 | 1 |
| TSMV_WAIT_TIMEOUT | <p>表空间迁移上锁等待时间。取值范围 0~12000, 单位 S</p> | 动态, 系统级 | 120 |
| TSMV_SEND_BUF_SIZE | <p>表空间迁移发送文件缓冲区大小。单位 MB, 取值范围 1~64。值越小, 迁移越慢, 但迁移中对并发业务影响越小; 值越大迁移速度越快, 但迁移中对并发业务影响越大, 可结合实际情况设置</p> | 动态, 系统级 | 4 |
| MAX_ESESSIONS | <p>指定分支 ESESSION 个数上限, 取值范围 0~650000, 0 表示无限制。</p> <p>ESESSION 为 SP 和其他节点之间的内部会</p> | 动态, 系统级 | 100000 |

DM8 分布计算集群

| | | | |
|--|--|--|--|
| | <p>话, SP 上的 ESESSION 为主 ESESSION, 其他节点上的 ESESSION 为分支 ESESSION</p> | | |
|--|--|--|--|



注意:

DMDPC 环境下不允许修改 TS_MAX_ID 和 TS_FIL_MAX_ID 参数。

6.1.1.2 自动覆盖不一致的参数

部分 DM.INI 参数, 当 SP/BP 本地设置的 DM.INI 参数值和自身在 MP 上注册的参数值不一致时, 按照在 MP 上实际注册值为准, 系统启动时会将 SP/BP 本地的参数自动覆盖掉。

如果 MP 为多副本架构, 则以 MP 主节点为准。

会被覆盖的 DM.INI 参数为: 一是包括和 DMDPC 相关的 PORT_NUM、AP_PORT_NUM; 以及二是 DM 数据库通用的参数。目前有以下这些: LENGTH_IN_CHAR、PARALLEL_POLICY、USE_NEW_HASH、COMPRESS_MODE、LIST_TABLE、LIST TABLE BRANCH、LIST TABLE NON BRANCH、COMPATIBLE_MODE、COUNT_64BIT、BLANK_PAD_MODE、ALTER_TABLE_OPT、DECIMAL_FIX_STORAGE、ENABLE_HUGE_SECIND、PK_WITH_CLUSTER、DATETIME_FMT_MODE、SLCT_ERR_PROCESS_FLAG、FORCE_CERTIFICATE_ENCRYPTION、CAST_VARCHAR_MODE、SPACE_COMPARE_MODE、JSON_MODE、DPC_2PC、MPP_MOTION_SYNC、BDTA_SIZE、USE_PLN_POOL、MAX_SESSION_STATEMENT、MAX_EP_SITES、DPC_SYNC_STEP、PC_SYNC_TOTAL、STMT_XBOX_REUSE、HUGE_ENABLE_DEL_UPD、ENABLE_INJECT_HINT、ENABLE_FLASHBACK、UNDO_RETENTION、ALLOWED_CLIENT_VERSIONS、SELECT_LOCK_MODE、ENABLE_CS_CVT、HFI_HP_MODE、NUMBER_MODE、VIEW_ACCESS_MODE、TRUNC_CHECK_MODE、PL SQL STRIP 等。

DM8 分布计算集群

当试图修改上述参数的内存值时，DM服务器会拒绝并报错。

6.1.1.3 从主库同步参数

在主备环境中，当备库 DM.INI 参数与主库不一致时，可在备库执行 INT SF_SYNC_INI(LEVEL INT)函数来同步主库上的 DM.INI 参数。LEVEL 为同步级别，取值 0 和 1。0 表示同步所属环境下必须同步的参数；1 表示同步所有可以同步的参数。

例 在备库上执行 SF_SYNC_INI 函数，备库同步主库上所有可以同步的 INI 参数。

```
SELECT SF_SYNC_INI (1);
```

SF_SYNC_INI 函数的详细用法请参考《DM8 SQL 语言使用手册》。

6.1.2 DMINIT 工具

通过 DMINIT 工具指定初始化参数。和 DMDPC 集群搭建相关的参数如下表 6.2 所示。

使用 DMINIT 工具初始化时，如果同一个 INI 参数在 DM.INI 和 DMINIT 中的值不同，那么以 DMINIT 工具中指定的值为准。

表 6.2 DMINIT 工具配置参数

| 参数 | 说明 | 缺省值 |
|---------------|---|-------------|
| PATH | 初始数据库存放的路径 | 无。不指定时会提示输入 |
| INSTANCE_NAME | 实例名 | DMSERVER |
| PORT_NUM | 数据库服务器监听端口号，取值范围 1024~65534 | 5236 |
| AP_PORT_NUM | 分布式环境下协同工作的监听端口号，取值范围 1024~65534 | 6000 |
| DPC_MODE | 指定 DMDPC 集群中的实例角色。 0: 无; 1/MP: MP; 2/BP: BP; 3/SP: SP。 | 0 |

DM8 分布计算集群

| |
|--|
| 使用 DMINIT 搭建环境时, DPC_MODE 参数值既可以使用数字 0/1/2/3, 也可以使用 MP/BP/SP 字符串代替, 二者作用等价 |
|--|

只要 DMINIT 初始化库时, 指定了 DPC_MODE 参数值, 那么在 DMSERVER 启动时, 启动参数 DPC_MODE 值必须和 DMINIT 保持一致。否则, 可能导致服务器起不来。DPC_MODE 在 DMINIT 和 DMSERVER 中的取值对应关系如表 6.3 所示。

表 6.3 DMINIT 和 DMSERVER 中 DPC_MODE 参数值对应关系

| DMINIT 的 DPC_MODE | DMSERVER 的 DPC_MODE |
|-------------------|---------------------|
| 0 | 0 |
| 1 或 MP | 1 或 MP |
| 2 或 BP | 2 或 BP |
| | 4 或 BS |
| 3 或 SP | 3 或 SP |

6.1.3 MP.INI

DMDPC 集群中除 MP 自身外, SP、BP 都存在和 MP 通讯的可能, 因此在 SP、BP 和 MP 初始化的 PATH 相同目录下需要用 MP.INI 来写明 MP 的 IP 地址和端口号。同一集群中 SP、BP 的 MP.INI 内容完全一致。

表 6.4 MP.INI 中的参数

| 参数名 | 说明 |
|-------------------|--|
| MP_HOST | mp 实例的 IP 地址 |
| MP_PORT | mp 实例的监听端口号 |
| MP_RECONN_TIMEOUT | 连接 MP 失败时, 尝试重连的超时时间。在设定的时间段内, 如果与 MP 连接 |

DM8 分布计算集群

| | |
|--|--|
| | 断开，会不断的尝试连接 MP。单位为秒。取值范围 0~4294967294，缺省为 4294967294 |
|--|--|

例 一个完整的 MP.INI 文件内容如下：

```
mp_host = 223.254.30.136
mp_port = 9000      #与 MP、BP 和 SP 上的 ap_port_num 不冲突的端口号
```

6.1.4 DMARCH.INI

DMARCH.INI 是 DM 数据库实例的归档配置文件，用于配置数据库本地归档和 RAFT 归档。这里仅对多副本系统相关配置项进行介绍。若读者对其他配置项感兴趣，可参考《DM8 系统管理员手册》的相关章节。

表 6.5 DMARCH.INI 多副本相关配置项

| 项目 | 项目意义 | 字段 | 字段意义 |
|-------|------|--------------------|---|
| 全局配置项 | | XMAL_HB_INTERVAL | 节点间通信检测间隔。单位 s，取值范围 1~600。该参数仅在多副本集群中使用，多副本各实例需配置相同 |
| | | RAFT_HB_INTERVAL | 主库广播心跳消息的间隔时间。单位 ms，取值范围 5~1800。该参数仅在多副本集群中使用，多副本各实例需配置相同 |
| | | RAFT_VOTE_INTERVAL | 选举超时时间。单位 ms，取值范围 10~60000。这个配置值需要保证至少是 RAFT_HB_INTERVAL 的 2 倍大小，确保在选举超时时间内 |

DM8 分布计算集群

| | | | |
|-----------------|---------|-------------------|--|
| | | | <p>能够收到主库的心跳消息，避免误判主库故障发起无效选举。建议各实例配置为不同的值，以便能够快速选举出主库，配置的值越小，被选举为主库的优先级越高。该参数仅在多副本集群中使用</p> |
| | | RAFT_SELF_ID | <p>多副本节点编号，用于标识多副本 RAFT 组中不同的节点。取值范围 0~31</p> |
| | | RAFT_LEARNER | <p>用于标识自己是否为 LEARNER 角色。1: 是，0: 否。 参数缺省为 0，非 LEARNER 节点可不配置此参数</p> |
| | | RAFT_PKG_INTERVAL | <p>专用于多副本环境中，设置备库接收主库日志包的超时时间。当备库超过该时间还未接收到主库日志包，则备库主动断开与主库连接。 单位 S，取值范围 60~36000。缺省为不配置该参数。可选参数</p> |
| [ARCHIVE_RAFTI] | RAFT 归档 | ARCH_TYPE | <p>归档类型。ARCH_TYPE 支持配置为 RAFT 和 LEARNER。其中 LEARNER 归档仅在动态增删节点时使用</p> |

DM8 分布计算集群

| | | | |
|------------------|--------|---------------------|--|
| | | ARCH_DEST | 归档目标实例名 |
| | | ARCH_DEST_ID | 归档目标多副本节点编号，用于标识多副本 RAFT 组中不同的节点。 取值范围 0~31 |
| [ARCHIVE_LOCAL1] | 本地归档配置 | ARCH_TYPE | 归档类型 |
| | | ARCH_DEST | 归档路径 |
| | | ARCH_FILE_SIZE | 单个归档文件大小，单位 MB，取值范围 64~2048，缺省为 1024MB，即 1G |
| | | ARCH_SPACE_LIMIT | 归档文件空间限制，单位 MB，取值范围 1024~4294967294，缺省为 0，表示无空间限制 |
| | | ARCH_FLUSH_BUF_SIZE | 归档合并刷盘缓存大小，单位 MB，取值范围 0~128，缺省为 0，表示不使用归档合并刷盘 |
| | | ARCH_HANG_FLAG | 本地归档写入失败时系统是否挂起。取值 0 或 1。0 不挂起；1 挂起。 缺省为 1。第一路本地归档系统内固定设为 1，设 0 实际也不起作用 |

说明：

1. ARCH_TYPE 表示 Redo 日志归档类型，多副本系统中需要使用到 RAFT 归档和本地归档，配置 RAFT 归档时，必须配置至少一个本地归档；

2. XMAL_PORT 是多副本用于选举和消息通信的专用端口号。为了防止端口通信冲突，

DM8 分布计算集群

XMAL_PORT 配置项必须不同于 dm.ini 中 PORT_NUM 配置;

3. 由于多副本系统总的节点个数必须为奇数, 因此 RAFT 归档目标个数要配置为偶数个 (除去本机后剩余的节点个数), 也就是最少要配置 2 个, 最多允许配置 8 个;

4. 可以通过 RAFT_VOTE_INTERVAL 设置选举优先级, RAFT_VOTE_INTERVAL 配置值越小的节点实例, 被选举为新主库的优先级越高。

6.1.5 DMSERVER 启动参数

DMSERVER 程序指定启动参数, 表示实例以何种模式工作。相关参数如下表 6.6 所示。

表 6.6 DMSERVER 启动参数

| 参数 | 说明 | 缺省值 |
|----------|------------------|---|
| PATH | 数据库 dm.ini 存放的路径 | dm.ini 绝对路径或者 dmserver 当前目录的 dm.ini |
| DPC_MODE | 实例运行模式 | 指定 DPC 中的实例角色, 0: 无角色, 缺省为 0、1: MP、2: BP、3: SP、4: BS, 取值 1/2/3/4 时也可以分别用 MP/BP/SP/BS 代替。其中 BS 模式表示直连 BP 模式, 详见 3.1 基本概念 中的 BS 模式说明。 DMSERVER 启动时, DPC_MODE 在 DMINIT 和 DMSEVER 中的取值须按照对应关系正确对应, 对应关系请参考 DMINIT 工具介绍 |

6.2 配置方法

本节对 DMDPC 进行配置的多种过程进行详细介绍。

6.2.1 SP_CREATE_DPC_BP_GROUP

定义:

```
SP_CREATE_DPC_BP_GROUP(  
    GROUP_NAME    VARCHAR(128),  
    DESC          VARCHAR(256)  
)
```

功能说明:

向 MP 注册一个新的 BP 组。

参数说明:

GROUP_NAME: 待注册 BP 组名称;

DESC: 描述信息。

举例说明:

注册名字为 BG_1 的 BP 组，描述信息为“bp group1”。

```
SP_CREATE_DPC_BP_GROUP('BG_1', 'bp group1');
```

6.2.2 SP_DROP_DPC_BP_GROUP

定义:

```
SP_DROP_DPC_BP_GROUP(  
    NAME          VARCHAR(128)  
)
```

功能说明:

删除一个 BP 组。

参数说明:

DM8 分布计算集群

NAME: 待移除 BP 组名称, 必须是已经注册过的 BP 组。

举例说明:

删除名字为 BG_1 的 BP 组。

```
SP_DROP_DPC_BP_GROUP('BG_1');
```

6.2.3 SP_CREATE_DPC_SP_GROUP

定义:

```
SP_CREATE_DPC_SP_GROUP(
    GROUP_NAME    VARCHAR(128),
    DESC          VARCHAR(256)
)
```

功能说明:

向 MP 注册一个新的 SP 组。

参数说明:

GROUP_NAME: 待注册 SP 组名称;

DESC: 描述信息。

举例说明:

注册名字为 SPG_1 的 SP 组, 描述信息为“sp group1”。

```
SP_CREATE_DPC_SP_GROUP('SPG_1', 'sp group1');
```

6.2.4 SP_DROP_DPC_SP_GROUP

定义:

```
SP_DROP_DPC_SP_GROUP(
    NAME          varchar(128)
```

)

功能说明:

删除一个 SP 组。

参数说明:

NAME: 待移除 SP 组名称, 必须是已经注册过的 SP 组。

举例说明:

删除名字为 SPG_1 的 SP 组。

```
SP_DROP_DPC_SP_GROUP('SPG_1');
```

6.2.5 SP_CREATE_DPC_RAFT

定义:

```
SP_CREATE_DPC_RAFT (
    DPC_MODE          VARCHAR(8),
    RAFT_NAME         VARCHAR(128)
)
```

功能说明:

向 MP 注册一个新的 RAFT 组。

参数说明:

DPC_MODE: 待注册 RAFT 组类型, 取值只能为 BP 或者 SP。一个 SP 类型的 RAFT 组中最多只能加入一个 SP 节点;

RAFT_NAME: 待注册 RAFT 组名称。

举例说明:

注册名字为 RAFT_1 的 BP RAFT 组, 隶属于 BG_1。

```
SP_CREATE_DPC_RAFT('BP', 'RAFT_1');
```

6.2.6 SP_DROP_DPC_BP_RAFT/SP_DROP_DPC_RAFT

定义:

```
SP_DROP_DPC_BP_RAFT(  
    NAME      VARCHAR(128)  
)
```

或

```
SP_DROP_DPC_RAFT(  
    NAME      VARCHAR(128)  
)
```

功能说明:

删除一个 RAFT 组。

参数说明:

NAME: 待删除 RAFT 组名称。

举例说明:

删除名字为 RAFT_1 的 RAFT 组。

```
SP_DROP_DPC_BP_RAFT('RAFT_1');
```

或

```
SP_DROP_DPC_RAFT('RAFT_1');
```

6.2.7 SP_CREATE_DPC_INSTANCE

SP_CREATE_DPC_INSTANCE 过程有两个。

1.内外网地址相同情景

定义:

DM8 分布计算集群

```

SP_CREATE_DPC_INSTANCE (
    RAFT_NAME    VARCHAR(128),
    NAME         VARCHAR(128),
    DPC_MODE     VARCHAR(8),
    AP_PORT      INT,
    INST_PORT    INT,
    IP_ADDR      VARCHAR(256),
    SYS_MODE     VARCHAR(32),
    STATUS       INT,
    DESC         VARCHAR(256)
)
    
```

功能说明:

注册一个 BP/SP/MP 实例。

参数说明:

RAFT_NAME: RAFT 名称, 和 SP_CREATE_DPC_RAFT 中保持一致。如果是 MP, 该值可写成'MP_RAFT', 也可缺省为空串"或 NULL, 因为 MP 多副本系统只有一个 RAFT 组, 所以 RAFT 组名固定为 MP_RAFT, 缺省值也为 MP_RAFT。如果 BP 或 SP, 则该值不能缺省;

NAME: 待注册的实例名称, 和 DM.INI 中 INSTANCE_NAME 保持一致;

DPC_MODE: 注册的实例类型, 取值: BP、SP 或 MP。需要和 RAFT_NAME 所属类型一致;

AP_PORT: AP_PORT 端口号, 和 DM.INI 中 AP_PORT_NUM 保持对应。和 DPC_INSTANCE 表中 XMAL_PORT 值对应。SP/BP 本地设置的 AP_PORT_NUM 如果和自身在 MP 上注册的 AP_PORT_NUM 不一致, 按照在 MP 上实际注册的为准, 启动时会自

DM8 分布计算集群

动覆盖;

INST_PORT: 实例端口号, 和 DM.INI 中 PORT_NUM 保持对应。和 DPC_INSTANCE 表中 INST_PORT 值对应。SP/BP 本地设置的 PORT_NUM 如果和自身在 MP 上注册的 PORT_NUM 不一致, 按照在 MP 上实际注册的为准, 启动时会自动覆盖;

IP_ADDR: IP 地址;

SYS_MODE: 主备模式, 取值有 NORMAL, PRIMARY, STANDBY;

STATUS: 实例状态, 1: 有效; 0: 无效; 2: 表示 SP 为正常退出状态, 可动态删除此 SP 实例;

DESC: 描述信息。

举例说明:

在 RAFT_1 组注册一个新的 BP 节点 BP_1。

```
SP_CREATE_DPC_INSTANCE('RAFT_1', 'BP_1', 'BP', 1822, 5238, '192.168.1.76', 'NORMAL', 1, 'BP instance');
```

2.内外网地址不同或相同的情景

定义:

```
SP_CREATE_DPC_INSTANCE (
    RAFT_NAME    VARCHAR(128),
    NAME        VARCHAR(128),
    DPC_MODE    VARCHAR(8),
    AP_PORT     INT,
    INST_PORT   INT,
    IP_INTER    VARCHAR(256),
    IP_EXTER    VARCHAR(256),
    SYS MODE    VARCHAR(32),
```

DM8 分布计算集群

```

STATUS          INT,

DESC            VARCHAR(256)

)
    
```

功能说明:

注册一个 BP/SP/MP 实例。

参数说明:

RAFT_NAME: RAFT 名称, 和 SP_CREATE_DPC_RAFT 中保持一致。如果是 MP, 该值可写成'MP_RAFT', 也可缺省为空串"或 NULL, 因为 MP 多副本系统只有一个 RAFT 组, 所以 RAFT 组名固定为 MP_RAFT, 缺省值也为 MP_RAFT。如果 BP 或 SP, 则该值不能缺省。

NAME: 待注册的实例名称, 和 DM.INI 中 INSTANCE_NAME 保持一致;

DPC_MODE: 注册的实例类型, 取值: BP、SP 或 MP。需要和 RAFT_NAME 所属类型一致;

AP_PORT: AP_PORT 端口号, 和 DM.INI 中 AP_PORT_NUM 保持对应。和 DPC_INSTANCE 表中 XMAL_PORT 值对应。SP/BP 本地设置的 AP_PORT_NUM 如果和自身在 MP 上注册的 AP_PORT_NUM 不一致, 按照在 MP 上实际注册的为准, 启动时会自动覆盖;

INST_PORT: 实例端口号, 和 DM.INI 中 PORT_NUM 保持对应。和 DPC_INSTANCE 表中 INST_PORT 值对应。SP/BP 本地设置的 PORT_NUM 如果和自身在 MP 上注册的 PORT_NUM 不一致, 按照在 MP 上实际注册的为准, 启动时会自动覆盖;

IP_INTER: 实例的内网 IP 地址;

IP_EXTER: 实例的外网 IP 地址; 可传空串, 表示内外网地址相同;

SYS_MODE: 主备模式, 取值有 NORMAL, PRIMARY, STANDBY;

STATUS: 实例状态, 1: 有效; 0: 无效; 2: 表示 SP 为正常退出状态, 可动态删除

此 SP 实例;

DESC: 描述信息。

举例说明:

在 RAFT_1 组注册一个新的 BP 节点 BP_1。

```
SP_CREATE_DPC_INSTANCE('RAFT_1','BP_1','BP',1822,5238,'192.168.1.76','192.168.1.77','NORMAL',1,
'BP instance');
```

6.2.8 SP_DROP_DPC_INSTANCE

定义:

```
SP_DROP_DPC_INSTANCE (
    NAME          VARCHAR(128)
)
```

功能说明:

移除多副本中的一个 BP/SP/MP 实例。只剩最后一个实例则不能删除。

参数说明:

NAME: 待移除实例名称, 必须是已经注册过的实例名。

举例说明:

移除名字为 BP_1 的 BP 服务器。

```
SP_DROP_DPC_INSTANCE('BP_1');
```

6.2.9 SP_MODIFY_DPC_INSTANCE

定义:

```
SP_MODIFY_DPC_INSTANCE (
    NAME          VARCHAR(128),
```

DM8 分布计算集群

```

IP_INTER    VARCHAR(256),
IP_EXTER    VARCHAR(256),
AP_PORT     INT,
INST_PORT   INT
)
  
```

功能说明:

修改注册在 MP 上的实例信息。修改成功后需要重启全部环境来应用新参数。

SP/BP 本地设置的 AP_PORT_NUM/PORT_NUM 如果和自身在 MP 上注册的 AP_PORT_NUM/PORT_NUM 不一致, 按照在 MP 上实际注册的为准, 启动时会自动覆盖。

参数说明:

NAME: 待修改的实例名称, 非空;

IP_INTER: 修改为新的内网 IP 地址, 可为空串, 表示不修改;

IP_EXTER: 修改为新的外网 IP 地址, 可为空串, 表示不修改;

AP_PORT: AP_PORT 端口号, 和 DM.INI 中 AP_PORT_NUM 保持对应。和 DPC_INSTANCE 表中 XMAL_PORT 值对应。可为空串, 表示不修改。SP/BP 本地设置的 AP_PORT_NUM 如果和自身在 MP 上注册的 AP_PORT_NUM 不一致, 按照在 MP 上实际注册的为准, 启动时会自动覆盖;

INST_PORT: 实例端口号, 和 DM.INI 中 PORT_NUM 保持对应。和 DPC_INSTANCE 表中 INST_PORT 值对应。可为空串, 表示不修改。SP/BP 本地设置的 PORT_NUM 如果和自身在 MP 上注册的 PORT_NUM 不一致, 按照在 MP 上实际注册的为准, 启动时会自动覆盖。

举例说明:

修改 BP_1 实例的 IP 地址、ap 端口号和实例端口号。可同时修改多个字段, 也可以只修改部分字段值, 不需要修改的字段传入空串"值即可, 要求必须修改至少一个字段。

DM8 分布计算集群

```
SP_MODIFY_DPC_INSTANCE('BP_1', '192.168.1.76', '192.168.100.77', 2822, 5338);
```

6.2.10 SP_BP_GROUP_ADD_RAFT

定义:

```
SP_BP_GROUP_ADD_RAFT (  
    BP_GROUP_NAME    VARCHAR(128),  
    RAFT_NAME        VARCHAR(128)  
)
```

功能说明:

向 BP 组中添加一个 RAFT 组。

参数说明:

BP_GROUP_NAME: 指定添加 RAFT 组的 BP 组名;

RAFT_NAME: 待添加的 RAFT 组名。

举例说明:

向 BP 组 BG_1 中添加一个名为 RAFT_1 的 RAFT 组。

```
SP_BP_GROUP_ADD_RAFT('BG_1', 'RAFT_1');
```

6.2.11 SP_BP_GROUP_DEL_RAFT

定义:

```
SP_BP_GROUP_DEL_RAFT (  
    BP_GROUP_NAME    VARCHAR(128),  
    RAFT_NAME        VARCHAR(128)  
)
```

功能说明:

DM8 分布计算集群

从 BP 组中删除一个 RAFT 组。

参数说明:

BP_GROUP_NAME: 指定删除的 BP 组名;

RAFT_NAME: 待删除的 RAFT 组名。

举例说明:

从 BP 组 BG_1 中删除名为 RAFT_1 的 RAFT 组。

```
SP_BP_GROUP_DEL_RAFT('BG_1', 'RAFT_1');
```

6.2.12 SP_SP_GROUP_ADD_RAFT

定义:

```
SP_SP_GROUP_ADD_RAFT (  
    SP_GROUP_NAME    VARCHAR(128),  
    RAFT_NAME        VARCHAR(128)  
)
```

功能说明:

向 SP 组中添加一个 RAFT 组。

参数说明:

SP_GROUP_NAME: 指定添加 RAFT 组的 SP 组名;

RAFT_NAME: 待添加的 RAFT 组名。

举例说明:

向 SP 组 SPG_1 中添加一个名为 SP_RAFT_1 的 RAFT 组。

```
SP_SP_GROUP_ADD_RAFT('SPG_1', 'SP_RAFT_1');
```

如果传入的参数 sp_group_name 并不是 SP 组或者 raft_name 不是 SP RAFT 对象, 系统会因指定的对象不存在而报错。

6.2.13 SP_SP_GROUP_DEL_RAFT

定义:

```
SP_SP_GROUP_DEL_RAFT (  
    SP_GROUP_NAME    VARCHAR(128),  
    RAFT_NAME        VARCHAR(128)  
)
```

功能说明:

从 SP 组中删除一个 SP RAFT。

参数说明:

SP_GROUP_NAME: 指定删除的 SP 组名;

RAFT_NAME: 待删除的 RAFT 组名。

举例说明:

从 SP 组 SPG_1 中删除名为 SP_RAFT_1 的 RAFT 组。

```
SP_SP_GROUP_DEL_RAFT('SPG_1', 'SP_RAFT_1');
```

6.2.14 SP_SET_DPC_NET_CONF

定义:

```
SP_SET_DPC_NET_CONF(  
    PARA_NAME        varchar(128),  
    PARA_VALUE       varchar(128)  
);
```

功能说明:

修改注册在 MP 上的通信参数。

DM8 分布计算集群

参数说明:

PARAM_NAME: 待修改的参数名称, 非空;

PARAM_VALUE: 修改为新的参数值, 非空。

举例说明:

修改 MP 与 BP, MP 与 SP 通信时使用的参数信息, 调用时同步修改系统表

SYS.DPC_NET_CONF, 修改成功后需要重启全部环境来应用新参数。目前支持修改的参数为 TIMEOUT (通信超时, 单位为秒)、XLNK_NUM (链路数)、MONITOR_XLNK_TIME (监控链路消息收发用时, 单位为微秒)、CRC_CHECK (消息的 CRC 检验)、COMPRESS_LEVEL (消息压缩级别)。

```
SP_SET_DPC_NET_CONF('TIMEOUT','1000');
```

为了增加网络传输效率和减少带宽利用, 对 XLNK 消息统一增加 COMPRESS_LEVEL 参数, 配置在 DPC_NET_CONF 系统表中。修改成功后需要重启全部节点才能生效。登录 SP 或者 MP 执行系统过程设置 XLNK 消息压缩等级。

```
SP_SET_DPC_NET_CONF('COMPRESS_LEVEL',10);
```

6.2.15 SP_TS_GROUP_CREATE

定义:

```
SP_TS_GROUP_CREATE(
    NAME          VARCHAR(128),
    DESC          VARCHAR(256)
)
```

功能说明:

创建表空间组。

参数说明:

NAME: 表空间组名称。

DESC: 表空间描述。

举例说明:

```
SP_TS_GROUP_CREATE('TSG_1', 'tablespace group1');
```

6.2.16 SP_TS_GROUP_DROP

定义:

```
SP_TS_GROUP_DROP(  
    GROUP_NAME  VARCHAR(128)  
)
```

功能说明:

删除表空间组。

参数说明:

NAME: 表空间组名称。

举例说明:

```
SP_TS_GROUP_DROP('TSG_1');
```

6.2.17 SP_TS_GROUP_ADD_TS

定义:

```
SP_TS_GROUP_ADD_TS(  
    GROUP_NAME  VARCHAR(128),  
    TS_NAME     VARCHAR(128)  
)
```

功能说明:

表空间组添加表空间。

参数说明:

GROUP_NAME: 表空间组名称;

TS_NAME: 表空间名称。

举例说明:

```
SP_TS_GROUP_ADD_TS('TSG_I', 'TSI');
```

6.2.18 SP_TS_GROUP_REMOVE_TS

定义:

```
SP_TS_GROUP_REMOVE_TS(
    GROUP_NAME  VARCHAR(128),
    TS_NAME     VARCHAR(128)
)
```

功能说明:

表空间组移除表空间。

参数说明:

GROUP_NAME: 表空间组名称;

TS_NAME: 表空间名称。

举例说明:

```
SP_TS_GROUP_REMOVE_TS('TSG_I', 'TSI');
```

6.2.19 SP_RENAME_DPC_INSTANCE

定义:

```
SP_RENAME_DPC_INSTANCE(
```

DM8 分布计算集群

```

    OLD_NAME    VARCHAR(128),
    NEW_NAME    VARCHAR(128)
)

```

功能说明:

重命名实例。

参数说明:

OLD_NAME: 旧实例名;

NEW_NAME: 新实例名。

举例说明:

```
SP_RENAME_DPC_INSTANCE('BP1_A', 'BP_INSTANCE_A');
```

6.2.20 SP_ALTER_DPC_INSTANCE

定义:

```

SP_ALTER_DPC_INSTANCE(
    NAME          VARCHAR(128),
    SYS_MODE      VARCHAR(32),
    SYS_STATUS    INT,
    STATUS        INT
)

```

功能说明:

修改实例的系统模式、系统状态、实例状态信息。

参数说明:

NAME: 实例名;

SYS_MODE: 系统模式;

SYS_STATUS: 系统状态;

STATUS: 实例状态。

举例说明:

```
SP_ALTER_DPC_INSTANCE('BPI','STANDBY',4,1);
```

6.2.21 SP_DPC_MOVE_TS_OFFLINE

定义:

```
SP_DPC_MOVE_TS_OFFLINE(
    TS_NAME      VARCHAR(128),
    RAFT_NAME    VARCHAR(128)
);
```

功能说明:

脱机方式表空间迁移时, MP 上手动修改系统表信息。只在脱机表空间迁移过程中使用。

参数说明:

TS_NAME: 迁移的表空间名称;

RAFT_NAME: 目标 RAFT 名。

举例说明:

将表空间 TS_1 迁移到 RAFT_1 上。

```
SP_DPC_MOVE_TS_OFFLINE('TS_1','RAFT_1');
```

6.2.22 SP_SET_DPC_INST_AUX

定义:

```
SP_SET_DPC_INST_AUX(
    INST_NAME    VARCHAR(128),
```

```
AUX_FLAG      INT
);
```

功能说明:

设置 SP 实例是否作为辅助计算节点。缺省为是。

参数说明:

INST_NAME: SP 节点的实例名;

AUX_FLAG: 是否作为辅助计算节点。1 是; 0 否。

举例说明:

取消 SP_1 实例作为辅助计算节点。

```
SP_SET_DPC_INST_AUX('SP_1', 0);
```

6.2.23 SP_ADD_RAFT_LEARNER

定义:

```
SP_ADD_RAFT_LEARNER(
    DEST          VARCHAR,
    DEST_IP      VARCHAR,
    DEST_PORT     INT,
    DEST_ID       INT
);
```

功能说明:

增加 LEARNER 节点。需在当前多副本集群的主节点上执行。

参数说明:

DEST: LEARNER 节点实例名, 对应 ARCH_DEST 字段;

DEST_IP: LEARNER 节点 IP 地址, 对应 ARCH_DEST_IP 字段;

DM8 分布计算集群

DEST_PORT: LEARNER 节点通信 PORT, 对应 ARCH_DEST_PORT 字段;

DEST_ID: LEARNER 节点编号, 对应 ARCH_DEST_ID 字段。

返回值:

无

举例说明:

增加多副本集群的 LEARNER 节点 BP_04。

```
SP_ADD_RAFT_LEARNER('BP_04', '192.168.1.68', 10010, 3);
```

6.2.24 SP_DELETE_RAFT_LEARNER

定义:

```
SP_DELETE_RAFT_LEARNER(  
    DEST          VARCHAR  
)
```

功能说明:

删除 LEARNER 节点。需在当前多副本集群的主节点上执行。

使用细则如下:

1. 每次执行只允许删除 1 个节点;
2. 只允许在 LEARNER 主节点上执行, 其他节点不允许执行;
3. 仅多数 RAFT 节点正常的情况下, 才允许执行;
4. DMDPC 下的删除 LEARNER 的顺序为, 先执行 SP_DELETE_RAFT_LEARNER 从 RAFT 组中删除, 再使用 SP_DROP_DPC_INSTANCE 从系统表中删除节点;
5. 和 SP_ADD_RAFT_LEARNER 策略一样, 此函数仅会删除 RAFT 节点上的 LEARNER 归档配置, 对于 LEARNER 节点上的配置则仍然是以手工方式进行修改。

参数说明:

DM8 分布计算集群

DEST: LEARNER 节点实例名, 对应 ARCH_DEST 字段。

返回值:

无

举例说明:

删除多副本集群的 LEARNER 节点 BP_04。

```
SP_DELETE_RAFT_LEARNER('BP_04');
```

6.2.25 SP_ALTER_RAFT_NODE

定义:

```
SP_ALTER_RAFT_NODE(  
    NEW_NODE VARCHAR  
)
```

功能说明:

执行多副本集群成员变更。将当前多副本集群变更为仅含有 NEW_NODE 节点的集群。

不管是增加、删除还是替换, 都可以使用这个系统函数来完成。

需在当前多副本集群的主节点上执行。

参数说明:

NEW_NODE: 指定新配置的所有节点实例名字符串。可以是一个或多个实例名, 多个实例名之间用“/”符号分隔, 目前最多支持 9 个节点。

返回值:

无

举例说明:

将当前多副本集群变更为仅含有 BP_01、BP_02、BP_03、BP_04、BP_05 节点的集群。

DM8 分布计算集群

```
SP_ALTER_RAFT_NODE('BP_01/BP_02/BP_03/BP_04/BP_05');
```

6.2.26 SP_ADD_RAFT_NODE

```
SP_ADD_RAFT_NODE(  
    NEW_NODE VARCHAR  
)
```

功能说明:

增加多副本集群节点。如果集群变更还包含删除或替换节点，则不建议使用这个系统函数。

需在当前多副本集群的主节点上执行。

参数说明:

NEW_NODE: 指定新增的所有节点实例名。实例名之间用“/”符号分隔。加入新增节点后，新集群的节点总数最多支持 9 个节点。

返回值:

无

举例说明:

增加多副本集群的节点 BP_04、BP_05。

```
SP_ADD_RAFT_NODE('BP_04/BP_05');
```

6.2.27 SP_DELETE_RAFT_NODE

```
SP_DELETE_RAFT_NODE(  
    DELETE_NODE VARCHAR  
)
```

功能说明:

DM8 分布计算集群

删除多副本集群节点。如果集群变更还包含增加或替换节点，则不建议使用这个系统函数。

需在当前多副本集群的主节点上执行。

参数说明：

DELETE_NODE：指定要删除的所有节点实例名。各实例名以“/”符号分隔。

返回值：

无

举例说明：

删除多副本集群的节点 BP_04、BP_05。

```
SP_DELETE_RAFT_NODE('BP_04/BP_05');
```

6.2.28 SP_REPLACE_RAFT_NODE

定义：

```
SP_REPLACE_RAFT_NODE(
    OLD_NODE    VARCHAR,
    NEW_NODE    VARCHAR
)
```

功能说明：

替换多副本集群的节点。即同时包含新增和删除节点的情况下，使用该系统函数来完成。

需在当前多副本集群的主节点上执行。

参数说明：

OLD_NODE：指定要删除的所有节点实例名。各实例名先“/”符号分隔，OLD_NODE 必须是旧配置下的多副本节点；

NEW_NODE：指定要增加的所有节点实例名。各实例名以“/”符号分隔，NEW_NODE

必须是 LEARNER 节点。

返回值:

无

举例说明:

例 1 将 BP_02/BP_03 替换为 BP_04/BP_05。

```
SP_REPLACE_RAFT_NODE('BP_02/BP_03','BP_04/BP_05');
```

例 2 将 BP_02/BP_03 替换为 BP_04/BP_05，再新增 BP_06/BP_07。

```
SP_REPLACE_RAFT_NODE('BP_02/BP_03','BP_04/BP_05/BP_06/BP_07');
```

6.2.29 SP_DPC_DUMP_INST

定义:

```
SP_DPC_DUMP_INST (
    FILE_PATH          VARCHAR(256)
)
```

功能说明:

将 DPC_INSTANCE 的配置以追加的形式，转储到文本文件，以便于在 MP 故障时查看。

文件路径不存在时报错。在 MP/BP/BS/SP 均能执行。

参数说明:

FILE_PATH: 导出的文本文件路径。

举例说明:

```
SP_DPC_DUMP_INST('s:\DPC\C\dpc_instance.md');
```

6.2.30 SP_SET_SP_UPGRADE

定义:

```
SP_SET_SP_UPGRADE(
    INST_NAME    VARCHAR(128)
)

```

功能说明:

将一个正常状态的 SP 标记为升级状态。非正常状态的 SP 无效。

只能在 SP 上运行；BP 和 MP 上无法运行。只能升级 SP。

参数说明:

INST_NAME: 待升级的 SP 的实例名称。

返回值:

无

举例说明:

```
SP_SET_SP_UPGRADE('SP2');
```

6.2.31 SP_RESET_SP_UPGRADE

定义:

```
SP_RESET_SP_UPGRADE(
    INST_NAME    VARCHAR(128)
)

```

功能说明:

将一个升级状态的 SP 恢复正常；非升级状态的 SP 无效。

只能在 SP 或 MP 上运行；BP 上无法运行。只能恢复 SP。

参数说明:

INST_NAME: 待恢复的 SP 的实例名称。

返回值:

无

举例说明:

```
SP_RESET_SP_UPGRADE('SP2');
```

6.2.32 SP_DPC_REBANLANCE_SESSION

定义:

```
SP_DPC_REBANLANCE_SESSION(
    BALANCE_FLAG    INT
)
```

功能说明:

收集当前 SP 可见的所有 SP 的会话数，并产生或清空所有节点各自的平衡方案。

参数说明:

BALANCE_FLAG: 取 1 表示产生节点各自的平衡方案; 取 0 表示清空各个节点的方案;

不论取 0 还是取 1，都会收集可见的所有 SP 的会话数。

返回值:

无

举例说明:

```
SP_DPC_REBANLANCE_SESSION(1);
```

6.2.33 SP_RAFT_SUSPEND_THREAD

定义:

```
SP_RAFT_SUSPEND_THREAD()
```

功能说明:

多副本环境中，主库挂起工作线程。直连主库执行。

举例说明:

主库挂起自己的工作线程。

```
SP_RAFT_SUSPEND_THREAD();
```

6.2.34 SP_RAFT_SWITCHOVER

定义:

```
SP_RAFT_SWITCHOVER()
```

功能说明:

多副本环境中，将备库手动切换为主库。直连备库执行。

举例说明:

备库手动切换为主库流程:

1.连接主库执行挂起

```
SP_RAFT_SUSPEND_THREAD();
```

2.连接备库执行手动切换

```
SP_RAFT_SWITCHOVER();
```

6.2.35 SP_RAFT_RESUME_THREAD

定义:

```
SP_RAFT_RESUME_THREAD(  
    INST_NAME  VARCHAR(128)  
)
```

功能说明:

多副本环境中，唤醒指定多副本组内的节点的工作线程，直连任意一个备库执行。

参数说明:

INST_NAME: 待唤醒的实例名称。

举例说明:

唤醒 BPO1 的工作线程。

```
SP_RAFT_RESUME_THREAD('BPO1');
```

6.2.36 SP_TS_DROP_INVALID

定义:

```
SP_TS_DROP_INVALID()
```

功能说明:

删除本地节点有，但是 MP 中没有注册的表空间。

只能在 BP 或 BS 节点执行该系统函数。

参数说明:

无。

举例说明:

```
SP_TS_DROP_INVALID;
```

6.2.37 SP_CLEAR_TAB_ROW_CNT_CACHE

定义:

```
SP_CLEAR_TAB_ROW_CNT_CACHE (
    RAFT_ID INT,
    TAB_ID INT
)
```

功能说明:

清除所有或指定节点上的全部或部分表行数缓存。

参数说明:

RAFT_ID: 站点号。

TAB_ID: 表 ID。

如果表 ID 和站点号都为 NULL，就清除所有站点上的全部缓存；提供表 ID，但是站点号为 NULL，就清除所有站点上的表 ID；如果表 ID 为 NULL，提供站点号，就清除设置站点上的所有缓存。

举例说明:

清理所有 SP 站点上的全部表行数缓存:

```
SP_CLEAR_TAB_ROW_CNT_CACHE(NULL,NULL);
```

6.2.38 SP_GET_ALL_TS_BY_TSGROUP

定义:

```
SP_GET_ALL_TS_BY_TSGROUP (
    TGNAME VARCHAR(128)
)
```

功能说明:

显示表空间组内的所有表空间信息。显示的列信息和系统表 DPC_INSTANCE 相同。

参数说明:

TGNAME: 表空间组名。

举例说明:

显示表空间组 TSG_1 内的所有表空间信息:

```
SP_GET_ALL_TS_BY_TSGROUP('TSG_1');
```

6.2.39 SP_SET_RAFT_ASYNC_INTERVAL

定义:

```
SP_SET_RAFT_ASYNC_INTERVAL(  
  
    ARCH_DEST VARCHAR,  
  
    ASYNC_INTERVAL INT  
  
)
```

功能说明:

重置一个 RAFT 备库或 LEARNER 备库的异步恢复时间间隔。

仅在多副本环境主库上执行有效。

对于 BS 主库，需要以 LOCAL 方式登录执行。

参数说明:

ARCH_DEST: 待重置异步恢复时间间隔的节点实例名。

ASYNC_INTERVAL: 异步恢复时间间隔，单位：s(秒)，取值范围 3~86400。

返回值:

无

举例说明:

在主库上执行，设置 RAFT 备库 BP_2 的异步恢复时间间隔为 30S。

```
SP_SET_RAFT_ASYNC_INTERVAL('BP_2', 30);
```

6.2.40 SP_GET_RAFT_ASYNC_INTERVAL

定义:

```
SP_GET_RAFT_ASYNC_INTERVAL(  
  
    ARCH_DEST    VARCHAR
```

)

功能说明:

获取一个 RAFT 备库或 LEARNER 备库的异步恢复时间间隔。

仅在多副本环境主库上执行有效。

对于 BS 主库，需要以 LOCAL 方式登录执行。

参数说明:

ARCH_DEST: 待重置异步恢复时间间隔的节点实例名。

返回值:

异步恢复时间间隔。

举例说明:

在主库上执行，获取 RAFT 备库 BP_2 的异步恢复时间间隔。

```
SP_GET_RAFT_ASYNC_INTERVAL('BP_2');
```

6.2.41 SP_CREATE_FAULT_DOMAIN

定义:

```
SP_CREATE_FAULT_DOMAIN(
    NAME          varchar(128),
    DESC          varchar(256)
)
```

功能说明:

创建一个容错域，创建后同步字典信息。可在 MP、BS、SP 上执行，BP 报错。

参数说明:

NAME: 容错域名称，与已有相同名称时报错。

DM8 分布计算集群

DESC: 容错域描述, 可以为空。

返回值:

无

举例说明:

创建一个名为 FDOM_1 的容错域。

```
SP_CREATE_FAULT_DOMAIN('FDOM_1', 'shanghai');
```

6.2.42 SP_DROP_FAULT_DOMAIN

定义:

```
SP_DROP_FAULT_DOMAIN(  
  
    NAME            varchar(128),  
  
    FORCE            INT  
  
)
```

功能说明:

删除一个容错域, 删除后同步字典信息。可在 MP、BS、SP 上执行, BP 报错。

参数说明:

NAME: 容错域名称, 原本就没有该容错域时报错。

FORCE: 是否强制删除, 0 为不强制删除: 该容错域不为空时报错; 1 为强制删除: 将移出容错域内所有实例, 然后删除容错域。

返回值:

无

举例说明:

删除一个名为 FDOM_1 的容错域。

```
SP_DROP_FAULT_DOMAIN('FDOM_1', 0);
```

6.2.43 SP_MODIFY_FAULT_DOMAIN

定义:

```
SP_MODIFY_FAULT_DOMAIN(  
  
    DOMAIN_ID      INT,  
  
    NAME           varchar(128),  
  
    DESC          varchar(128)  
  
)
```

功能说明:

修改一个容错域，修改后同步字典信息。可在 MP、BS、SP 上执行，BP 报错。

参数说明:

DOMAIN_ID: 容错域 ID，指定要修改哪一个容错域，id 不存在时报错。

NAME: 修改后的容错域名称，与已有相同名称时报错，为 NULL 表示不修改。

DESC: 修改后的容错域描述，为 NULL 时表示不修改。

返回值:

无

举例说明:

修改一个名为 FDOM_1 的容错域。

```
SP_MODIFY_FAULT_DOMAIN(1, 'FDOM_2', 'shanghai123'); --同时修改 0 号域的名称和描述
```

```
SP_MODIFY_FAULT_DOMAIN(1, NULL, ""); --不修改名称，同时描述清空
```

```
SP_MODIFY_FAULT_DOMAIN(1, 'FDOM_3', NULL); --修改名称，描述不修改
```

6.2.44 SP_FAULT_DOMAIN_MV_INST

定义:

DM8 分布计算集群

```
SP_FAULT_DOMAIN_MV_INST(
    DOMAIN_NAME    varchar(128),
    INST_NAME      varchar(128)
)
```

功能说明:

将实例移入或移出某个容错域，调整后同步字典信息。可在 MP、BS、SP 上执行，BP 报错。

参数说明:

DOMAIN_NAME: 容错域名称，容错域不存在时报错；取 NULL 时将尝试实例移出其在当前容错域；容错域不为 NULL 且存在时，需要容错域中无相同 RAFT 组时才能成功移入，否则将报错。

INST_NAME: 实例名称，实例不存在时报错。仅允许 MP,BP,BS 实例加入，SP 实例会报错。

返回值:

无

举例说明:

将实例 MPI 移入容错域 FDOM_1。将实例 BP3 移出当前容错域。

```
SP_FAULT_DOMAIN_MV_INST('FDOM_1', 'MPI'); --将 MPI 移入 FDOM_1
```

```
SP_FAULT_DOMAIN_MV_INST(NULL, 'BP3'); --将 BP3 移出它所在的当前容错域
```

7 DMDPC 集群部署

一个最小的 DMDPC 集群包含一个 BP、一个 SP 和一个 MP。可以在部署完毕后随时增添新的 BP、SP 节点。至少需要两台 BP 才方便看出子任务的各种计划形态和调度。

Windows 和 Linux 环境下部署 DMDPC 集群并无区别，目前支持借助命令行工具部署和 DEM 图形化界面方式部署两种方式。

7.1 命令行工具部署 DMDPC（单机架构）

DMDPC 集群规划部署 1 个 SP，2 个 BP 和 1 个 MP。其中，BP 采用单机模式，未配置为多副本系统。

IP 和端口分配如下表所示，仅用作示例，需按实际 IP 进行设置。

表 7.1 集群规划示意

| RAFT 组名 | 角色 | 实例名称 | IP | PORT _NUM | AP_POR T_NUM | 路径 |
|---------------------------|----|------|----------------|--------------|-----------------|-------------------------|
| RAFT_SP1 | SP | SP1 | 223.254.30.136 | 5236 | 6000 | d:\dpc_data_mac\sp1 |
| RAFT_1 | BP | BP1 | 223.254.30.136 | 5237 | 6001 | d:\dpc_data_mac\bp1 |
| RAFT_2 | BP | BP2 | 223.254.30.136 | 5238 | 6002 | d:\dpc_data_mac\bp 2 |
| 缺省为 NULL 或者 MP_RAFT | MP | MP | 223.254.30.136 | 5239 | 6003 | d:\dpc_data_mac\mp |

以下章节就以在同一台 Windows 主机为例介绍如何部署。

7.1.1 初始化数据库实例

初始化 4 个数据库实例，实例名称分别为 SP1、BP1、BP2 和 MP。

```
dminit path=d:\dpc_data_mac\sp1 instance_name=SP1 port_num=5236
ap_port_num=6000 dpc_mode=SP
dminit path=d:\dpc_data_mac\bp1 instance_name=BP1 port_num=5237
ap_port_num=6001 dpc_mode=BP
dminit path=d:\dpc_data_mac\bp2 instance_name=BP2 port_num=5238
ap_port_num=6002 dpc_mode=BP
dminit path=d:\dpc_data_mac\mp instance_name=MP port_num=5239
ap_port_num=6003 dpc_mode=MP
```

7.1.2 为 SP、BP 和 MP 配置 MP.INI 文件

为 SP、BP 和 MP 实例配置 MP.INI 文件。

MP.INI 文件内容如下：

```
mp_host = 223.254.30.136
mp_port = 9000      #与 MP、BP 和 SP 上的 ap_port_num 不冲突的端口号
```

将 MP.INI 文件内容分别写入 SP（SP1）、BP（BP1、BP2）和 MP 中。打开 cmd 命令行

工具执行：

```
//为 SP1 配置 MP.INI 文件
echo mp_host = 223.254.30.136 >d:\dpc_data_mac\sp1\DAMENG\mp.ini
echo mp_port = 9000    #与 MP、BP 和 SP 上的 ap_port_num 不冲突的端口号 >>
d:\dpc_data_mac\sp1\DAMENG\mp.ini
//为 BP1 配置 MP.INI 文件
```

DM8 分布计算集群

```

echo mp_host = 223.254.30.136 > d:\dpc_data_mac\bp1\DAMENG\mp.ini

echo mp_port = 9000    #与 MP、BP 和 SP 上的 ap_port_num 不冲突的端口号>>

d:\dpc_data_mac\bp1\DAMENG\mp.ini

//为 BP2 配置 MP.INI 文件

echo mp_host = 223.254.30.136 > d:\dpc_data_mac\bp2\DAMENG\mp.ini

echo mp_port = 9000    #与 MP、BP 和 SP 上的 ap_port_num 不冲突的端口号>>

d:\dpc_data_mac\bp2\DAMENG\mp.ini

//为 MP 配置 MP.INI 文件

echo mp_host = 223.254.30.136 >d:\dpc_data_mac\mp\DAMENG\mp.ini

echo mp_port = 9000    #与 MP、BP 和 SP 上的 ap_port_num 不冲突的端口号>>

d:\dpc_data_mac\mp\DAMENG\mp.ini
  
```

7.1.3 启动 MP

启动 MP。

```
dmserver d:\dpc_data_mac\mp\DAMENG\dm.ini dpc_mode=MP
```

DMDPC 运行过程中，MP 需要始终处于开启状态。

7.1.4 将 MP、SP 和 BP 加入集群

增加 1 个 MP、1 个 SP 和 2 个 BP 节点。只有在注册当前登录 MP 节点后，才可以注册其余节点。后续增加 MP、SP 节点和 BP 节点无先后之分。

```
//搭建 DMDPC 过程中加入 MP、SP 和 BP，必须登录 MP 进行操作
```

```
Disql SYSDBA/SYSDBA@223.254.30.136:5239
```

```
//增加 MP 节点
```

DM8 分布计算集群

```

//注册当前 MP 实例，MP 的 RAFT 组名可以指定为 NULL 或者'MP_RAFT'

SP_CREATE_DPC_INSTANCE(NULL,'MP','MP',6003,5239, '223.254.30.136',
'223.254.30.136','NORMAL',1,'MP instance');

//增加两个 BP 节点: BP1 和 BP2

//注册 RAFT 组，名为 RAFT_1

SP_CREATE_DPC_RAFT('BP','RAFT_1');

//在 RAFT_1 组内注册 BP 实例 BP1

SP_CREATE_DPC_INSTANCE('RAFT_1','BP1','BP',6001,5237, '223.254.30.136',
'223.254.30.136','NORMAL',1,'BP instance');

//注册 RAFT_2

SP_CREATE_DPC_RAFT('BP', 'RAFT_2');

//在 RAFT_2 内注册 BP 实例 BP2

SP_CREATE_DPC_INSTANCE('RAFT_2','BP2','BP',6002,5238, '223.254.30.136', '223.254.30.136',
'NORMAL', 1, 'BP instance');

//注册一个 BP 组，名为 BG_1

SP_CREATE_DPC_BP_GROUP('BG_1', 'bp group1');

//往 BP 组中添加 RAFT 组

SP_BP_GROUP_ADD_RAFT('BG_1', 'RAFT_1');

SP_BP_GROUP_ADD_RAFT('BG_1', 'RAFT_2');

//增加 SP 节点: SPI

```

DM8 分布计算集群

```
//增加 SP, 也要注册 RAFT 组

SP_CREATE_DPC_RAFT('SP', 'RAFT_SPI');

//在 RAFT_SPI 内注册 SP 实例 SPI

SP_CREATE_DPC_INSTANCE('RAFT_SPI','SPI','SP',6000,5236, '223.254.30.136',
'223.254.30.136','NORMAL', 2, 'SP instance');

//注册一个容错域: FDOM_1 (可选)

SP_CREATE_FAULT_DOMAIN ('FDOM_1','shanghai');

//往容错域中添加实例

SP_FAULT_DOMAIN_MV_INST('FDOM_1','MP');

SP_FAULT_DOMAIN_MV_INST('FDOM_1','BP1');

SP_FAULT_DOMAIN_MV_INST('FDOM_1','BP2');
```



将 SP 和 BP 节点加入 DMDPC 集群中的步骤必须在 SP、BP 第一次启动前完成！
注意：成！

7.1.5 检查注册是否成功

查询系统表，检查上一步骤的注册是否成功。能查到相关信息表示注册成功。

```
select * from DPC_BP_GROUP;

//查询结果如下:

行号      ID          NAME DESCRIPTION
-----
1         1          BG_1 bp group

select * from DPC_BP_RAFT;
```

DM8 分布计算集群

//查询结果如下:

| 行号 | RAFT_ID | GROUP_ID | DPC_MODE | NAME | IS_VALID | INFO1 | INFO2 | INFO3 |
|----|---------|----------|----------|------------|----------|-------|-------|-------|
| 1 | 0 | -1 | MP | MP_RAFT_1 | 1 | NULL | NULL | |
| 2 | 1 | -1 | BP | RAFT_1_1 | 1 | NULL | NULL | |
| 3 | 2 | -1 | BP | RAFT_2_1 | 1 | NULL | NULL | |
| 4 | 3 | -1 | SP | RAFT_SPI_1 | 1 | NULL | NULL | |

select * from DPC_INSTANCE;

//查询结果如下:

| 行号 | RAFT_ID | INST_ID | NAME | DPC_MODE | XML_PORT | INST_PORT | IP_INTERNAL | SYS_MODE | SYS_STATUS | STATUS | DESCRIPTION | IP_EXTERNAL | INFO1 | INFO2 | INFO3 |
|--------|---------|---------|-------------|----------|----------------|-----------|----------------|----------|------------|--------|-------------|-------------|-------|-------|-------|
| 1 | 0 | 4096 | MP | MP | 6003 | 5239 | 223.254.30.136 | | | | | | | | |
| NORMAL | 6 | 1 | MP instance | | 223.254.30.136 | 65536 | | | | | | | NULL | | |

DM8 分布计算集群

| | | | | | | |
|--------|---|------|-------------|----------------|-------|--------------|
| NULL | | | | | | |
| 2 | 1 | 4097 | BP1 BP | 6001 | 5237 | 192.168.1.27 |
| NORMAL | 6 | 1 | BP instance | 223.254.30.136 | 65536 | NULL |
| NULL | | | | | | |
| 3 | 2 | 4098 | BP2 BP | 6002 | 5238 | 192.168.1.27 |
| NORMAL | 6 | 1 | BP instance | 223.254.30.136 | 65536 | NULL |
| NULL | | | | | | |
| 4 | 3 | 4099 | SP1 SP | 6000 | 5236 | 192.168.1.27 |
| NORMAL | 6 | 2 | SP instance | 223.254.30.136 | NULL | NULL |
| NULL | | | | | | |

7.1.6 启动 SP 和 BP

启动 SP 和 BP 没有先后之分。

启动 SP。

```
dmserver d:\dpc_data_mac\sp1\DAMENG\dm.ini dpc_mode=SP
```

启动 BP。

```
dmserver d:\dpc_data_mac\bp1\DAMENG\dm.ini dpc_mode=BP
```

```
dmserver d:\dpc_data_mac\bp2\DAMENG\dm.ini dpc_mode=BP
```

至此，DMDPC 集群搭建完毕。

MP、SP、BP 全部正常启动，且均处于 OPEN 状态，才是一个正常的 DMDPC 系统！

当 BP 处于下述情况时，表示无可用的 BP，系统异常，报错“无效的系统状态”。

无可用的 BP 的情况为：



注意：1.BP 处于非 OPEN 状态；

2.BP 尚未成功启动；

3.BP 宕机。

7.1.7 连接 SP

DMDPC 搭建完成后，用户只需要连接对外提供服务的 SP，即可获得完整的数据库服务。

验证环境搭建是否成功，可以在 SP 上执行查询 `V$instance`，看是否所有 RAFT 组中实例都能够查到。

只有出于监控目的进行 `V$` 视图的查询，或出于维护需要时，才允许直连 MP 和 BP，否则搭建完成后，不允许再连接 MP 和 BP。切勿在 MP 和 BP 上执行大量 SQL 语句，否则可能会导致系统崩溃！

7.1.8 退出 DMDPC

使用 `exit` 命令退出 DMDPC 需要按照正确的顺序有序进行。

第一步 退出 SP；

第二步 退出 BP；

第三步 退出 MP。



注意：

如果没有按正常顺序退出，可能会导致剩下的机器直接宕机。此时，只能对其他的机器采用强杀。这种服务器异常退出的情况，当再次重启的时候，服务器需要做大量的 REDO 日志，因此重启的时间会稍长一些，其它没有影响。

7.2 命令行工具部署 DMDPC（BP 多副本架构）

DMDPC 集群规划部署 1 个 SP，2 个 BP 和 1 个 MP。MP 采用单机模式，BP 采用多副本模式（每个 BP 配置成 3 副本）。

IP 和端口分配如下表所示，仅用作示例，需按实际 IP 进行设置。

DM8 分布计算集群

表 7.2 集群规划示意

| RAFT 组名 | 角 色 | 实例 名称 | IP | PORT _NU M | AP_POR T_NUM | 路径 |
|------------------------------|--------|----------|----------------|------------------|-----------------|--------------------------|
| RAFT_SPI | SP | SPI | 223.254.30.136 | 5236 | 6000 | d:\dpc_data_mac\spi |
| 缺省为 NULL 或者 MP_RAFT | MP | MP | 223.254.30.136 | 5237 | 6001 | d:\dpc_data_mac\mp |
| RAFT_1 | BP | BP11 | 223.254.30.136 | 5238 | 6002 | d:\dpc_data_mac\bp11 |
| | BP | BP12 | 223.254.30.136 | 5239 | 6003 | d:\dpc_data_mac\bp12 |
| | BP | BP13 | 223.254.30.136 | 5240 | 6004 | d:\dpc_data_mac\bp13 |
| RAFT_2 | BP | BP21 | 223.254.30.136 | 5241 | 6005 | d:\dpc_data_mac\bp21 |
| | BP | BP22 | 223.254.30.136 | 5242 | 6006 | d:\dpc_data_mac\bp2 2 |
| | BP | BP23 | 223.254.30.136 | 5243 | 6007 | d:\dpc_data_mac\bp2 3 |

以下章节就以在同一台 Windows 主机为例介绍如何部署。

7.2.1 初始化数据库实例

初始化 8 个实例，分别为 SP、MP 和 6 个 BP 角色。

```
dmnit path=d:\dpc_data_mac\spi instance_name=SPI port_num=5236
ap_port_num=6000 dpc_mode=SP
```

DM8 分布计算集群

```

dminit path=d:\dpc_data_mac\mp instance_name=MP port_num=5237

ap_port_num=6001 dpc_mode=MP

//初始化 RAFT_1 组中的 BP 实例

dminit path=d:\dpc_data_mac\bp11 instance_name=BP11 port_num=5238

ap_port_num=6002 dpc_mode=BP

dminit path=d:\dpc_data_mac\bp12 instance_name=BP12 port_num=5239

ap_port_num=6003 dpc_mode=BP

dminit path=d:\dpc_data_mac\bp13 instance_name=BP13 port_num=5240

ap_port_num=6004 dpc_mode=BP

//初始化 RAFT_2 组中的 BP 实例

dminit path=d:\dpc_data_mac\bp21 instance_name=BP21 port_num=5241

ap_port_num=6005 dpc_mode=BP

dminit path=d:\dpc_data_mac\bp22 instance_name=BP22 port_num=5242

ap_port_num=6006 dpc_mode=BP

dminit path=d:\dpc_data_mac\bp23 instance_name=BP23 port_num=5243

ap_port_num=6007 dpc_mode=BP

```

7.2.2 为 SP、BP 和 MP 配置 MP.INI 文件

为 SP、BP 和 MP 实例配置 MP.INI 文件。

MP.INI 文件内容如下：

```

mp_host = 223.254.30.136

mp_port = 9000      #与 MP、BP 和 SP 上的 ap_port_num 不冲突的端口号

```

DM8 分布计算集群

将 MP.INI 文件内容分别写入 SP (SP1)、两个 RAFT 组内的所有 BP (BP11、BP12、BP13、BP21、BP22、BP23) 和 MP 中。打开 cmd 命令行工具执行：

```
//向 SP 写入 MP.INI

echo mp_host = 223.254.30.136 > d:\dpc_data_mac\sp1\DAMENG\mp.ini

echo mp_port = 9000      #与 MP、BP 和 SP 上的 ap_port_num 不冲突的端口号 >>

d:\dpc_data_mac\sp1\DAMENG\mp.ini

//向两个 RAFT 组内的所有 BP 写入 MP.INI, 不同 BP 需要切换到不同的路径下, 这里以 BP11 的路径为例 (或者直接拷贝 SP 上的 MP.INI 到所有 BP 的库目录下)

echo mp_host = 223.254.30.136 > d:\dpc_data_mac\bp11\DAMENG\mp.ini

echo mp_port = 9000      #与 MP、BP 和 SP 上的 ap_port_num 不冲突的端口号 >>

d:\dpc_data_mac\bp11\DAMENG\mp.ini

//为 MP 配置 MP.INI 文件

echo mp_host = 223.254.30.136 > d:\dpc_data_mac\mp\DAMENG\mp.ini

echo mp_port = 9000      #与 MP、BP 和 SP 上的 ap_port_num 不冲突的端口号 >>

d:\dpc_data_mac\mp\DAMENG\mp.ini
```

7.2.3 启动 MP

启动 MP。

```
dmserver d:\dpc_data_mac\mp\DAMENG\dm.ini dpc_mode=MP
```

DMDPC 运行过程中，MP 需要始终处于开启状态。

7.2.4 将 MP、SP 和 BP 加入集群

增加 1 个 MP、1 个 SP 和 6 个 BP 节点。只有在注册当前登录 MP 节点后，才可以注册其余节点。后续增加 MP、SP 节点和 BP 节点无先后之分。

DM8 分布计算集群

```

//搭建 DMDPC 过程中加入 MP、SP 和 BP，必须登录 MP 进行操作

Disql SYSDBA/SYSDBA@223.254.30.136:5237

//增加 MP 节点

//注册当前 MP 实例，MP 的 RAFT 组名可以指定为 NULL 或者'MP_RAFT'

SP_CREATE_DPC_INSTANCE(NULL,'MP','MP',6001,5237, '223.254.30.136',
'223.254.30.136','NORMAL',1,'MP instance');

//增加 BP 节点

//注册 RAFT 组，名为 RAFT_1

SP_CREATE_DPC_RAFT('BP', 'RAFT_1');

//在 RAFT_1 组内注册 BP 实例 BPI1、BPI2、BPI3

//所有实例均以 STANDBY 模式、INVALID 状态注册进 MP，实例启动后，在 RAFT 组内选举出有效主库后，
会自动更新 MP 中的模式、状态信息。

SP_CREATE_DPC_INSTANCE('RAFT_1', 'BPI1', 'BP', 6002, 5238, '223.254.30.136',
'223.254.30.136','STANDBY', 0, 'BP instance');

SP_CREATE_DPC_INSTANCE('RAFT_1', 'BPI2', 'BP', 6003, 5239, '223.254.30.136',
'223.254.30.136','STANDBY', 0, 'BP instance');

SP_CREATE_DPC_INSTANCE('RAFT_1', 'BPI3', 'BP', 6004, 5240, '223.254.30.136',
'223.254.30.136','STANDBY', 0, 'BP instance');

//注册 RAFT 组，名为 RAFT_2

SP_CREATE_DPC_RAFT('BP', 'RAFT_2');

//在 RAFT_2 内注册 BP 实例 BP21、BP22、BP23
  
```

DM8 分布计算集群

```

SP_CREATE_DPC_INSTANCE('RAFT_2', 'BP21', 'BP', 6005, 5241, '223.254.30.136',
'223.254.30.136', 'STANDBY', 0, 'BP instance');

SP_CREATE_DPC_INSTANCE('RAFT_2', 'BP22', 'BP', 6006, 5242, '223.254.30.136',
'223.254.30.136', 'STANDBY', 0, 'BP instance');

SP_CREATE_DPC_INSTANCE('RAFT_2', 'BP23', 'BP', 6007, 5243, '223.254.30.136',
'223.254.30.136', 'STANDBY', 0, 'BP instance');

//注册一个 BP 组， 名为 BG_1

SP_CREATE_DPC_BP_GROUP('BG_1', 'bp group1');

//在 BP 组内添加 RAFT 组

SP_BP_GROUP_ADD_RAFT('BG_1', 'RAFT_1');

SP_BP_GROUP_ADD_RAFT('BG_1', 'RAFT_2');

//增加 SP 节点

//增加 SP， 也要注册 RAFT 组， 参数 1 为空串

SP_CREATE_DPC_RAFT('SP', 'RAFT_SPI');

//在 RAFT_SPI 内注册 SP 实例 SPI

SP_CREATE_DPC_INSTANCE('RAFT_SPI', 'SPI', 'SP', 6000, 5236, '223.254.30.136',
'223.254.30.136', 'NORMAL', 2, 'SP instance');

//注册三个容错域: FDOM_1 FDOM_2 FDOM_3

SP_CREATE_FAULT_DOMAIN ('FDOM_1', 'shanghai_1');

SP_CREATE_FAULT_DOMAIN ('FDOM_2', 'shanghai_2');

SP_CREATE_FAULT_DOMAIN ('FDOM_3', 'shanghai_3');
  
```

DM8 分布计算集群

```
//往容错域中添加实例

SP_FAULT_DOMAIN_MV_INST('FDM_1','MP');

SP_FAULT_DOMAIN_MV_INST('FDM_1','BP11');

SP_FAULT_DOMAIN_MV_INST('FDM_1','BP21');

SP_FAULT_DOMAIN_MV_INST('FDM_2','BP12');

SP_FAULT_DOMAIN_MV_INST('FDM_2','BP22');

SP_FAULT_DOMAIN_MV_INST('FDM_3','BP13');

SP_FAULT_DOMAIN_MV_INST('FDM_3','BP23');
```



将 SP 和 BP 节点加入 DMDPC 集群中的步骤必须在 SP、BP 第一次启动前完成！
注意：成！

7.2.5 检查注册是否成功

查询系统表，检查上一步骤的注册是否成功。能查到相关信息表示注册成功。

```
select * from DPC_BP_GROUP;

//查询结果如下:

行号      ID          NAME DESCRIPTION
-----
1         1          BG_1 bp group1

select * from DPC_BP_RAFT;

//查询结果如下:

行号      RAFT_ID     GROUP_ID    DPC_MODE NAME    IS_VALID    INFO1    INFO2
INFO3
```

DM8 分布计算集群

```

-----
-----
1      0      -1      MP      MP_RAFT  1      NULL      NULL
NULL
2      1      -1      BP      RAFT_1   1      NULL      NULL
NULL
3      2      -1      BP      RAFT_2   1      NULL      NULL
NULL
4      3      -1      SP      RAFT_SPI 1      NULL      NULL
NULL

```

select * from DPC_INSTANCE;

//查询结果如下:

```

行号      RAFT_ID      INST_ID      NAME DPC_MODE XMAL_PORT      INST_PORT
IP_INTERNAL      SYS_MODE SYS_STATUS      STATUS      DESCRIPTION IP_EXTERNAL      INFO1
INFO2      INFO3

```

```

-----
-----
-----
1      0      4096      MP  MP      6001      5237      223.254.30.136
NORMAL  6      1      MP instance 223.254.30.136  65536      NULL
NULL
2      1      4097      BP|| BP      6002      5238      223.254.30.136
STANDBY 6      0      BP instance 223.254.30.136  65536      NULL

```

DM8 分布计算集群

| | | | | | | |
|---------|---|------|-------------|----------------|--------|----------------|
| NULL | | | | | | |
| 3 | 1 | 4098 | BP12 BP | 6003 | 5239 | 223.254.30.136 |
| STANDBY | 6 | 0 | BP instance | 223.254.30.136 | 131072 | NULL |
| NULL | | | | | | |
| 4 | 1 | 4099 | BP13 BP | 6004 | 5240 | 223.254.30.136 |
| STANDBY | 6 | 0 | BP instance | 223.254.30.136 | 196608 | NULL |
| NULL | | | | | | |
| 5 | 2 | 4100 | BP21 BP | 6005 | 5241 | 223.254.30.136 |
| STANDBY | 6 | 0 | BP instance | 223.254.30.136 | 65536 | NULL |
| NULL | | | | | | |
| 6 | 2 | 4101 | BP22 BP | 6006 | 5242 | 223.254.30.136 |
| STANDBY | 6 | 0 | BP instance | 223.254.30.136 | 131072 | NULL |
| NULL | | | | | | |
| 7 | 2 | 4102 | BP23 BP | 6007 | 5243 | 223.254.30.136 |
| STANDBY | 6 | 0 | BP instance | 223.254.30.136 | 196608 | NULL |
| NULL | | | | | | |
| 8 | 3 | 4103 | SP1 SP | 6000 | 5236 | 223.254.30.136 |
| NORMAL | 6 | 2 | SP instance | 223.254.30.136 | NULL | NULL |
| NULL | | | | | | |

7.2.6 配置 BP11

1. 配置 BP11 的 dm.ini

```
ARCH_INI          = 1          #打开归档配置
```

2. 配置 BP11 的本地归档 dmarch.ini。此处先只配本地归档，避免提前发起 RAFT 选举

DM8 分布计算集群

```
[ARCHIVE_LOCAL1]
```

```
ARCH_TYPE          = LOCAL          #本地归档类型

ARCH_DEST          = d:\dpc_data_mac\bp11\DAMENG\arch    #本地归档文件路径

ARCH_FILE_SIZE     = 128              #本地单个归档文件最大值,单位 MB

ARCH_SPACE_LIMIT   = 0                #本地归档文件总大小,0 表示无限制
```

3. 正常启动 BP11 到 Open 状态

```
dmserver d:\dpc_data_mac\bp11\DAMENG\dm.ini dpc_mode=bp
```

7.2.7 准备 BP RAFT 组内数据

RAFT_1 和 RAFT_2 组内的 BP 数据准备方式完全相同,只是路径不同,这里对组 RAFT_1 的数据准备步骤进行说明,RAFT_2 组参考这个步骤执行即可。注意,RAFT_2 组和 RAFT_1 组要各自按照此小节介绍的方式准备数据,RAFT_2 组不能直接使用 RAFT_1 组的备份集执行还原恢复。

7.2.7.1 退出 BP11 并进行脱机备份

首先,退出 BP11。

其次,对 BP11 进行脱机备份。

```
dmrman CTLSTMT="BACKUP DATABASE 'd:\dpc_data_mac\bp11\DAMENG\dm.ini' FULL TO
BACKUP_01 BACKUPSET 'd:\dpc_data_mac\bp11\BACKUP_01'" USE_AP=2
```

1. 将 BP11 的备份文件脱机还原到 BP12

```
//还原数据库:
```

```
dmrman CTLSTMT="RESTORE DATABASE 'd:\dpc_data_mac\bp12\DAMENG\dm.ini' FROM
BACKUPSET 'd:\dpc_data_mac\bp11\BACKUP_01'" USE_AP=2
```

```
//因为脱机备份没有产生任何 REDO 日志,所以此处省略恢复数据库的操作步骤
```

DM8 分布计算集群

```
//更新数据库:

dmrman CTLSTMT="RECOVER DATABASE 'd:\dpc_data_mac\bp12\DAMENG\dm.ini' UPDATE
DB_MAGIC" USE_AP=2
```

2. 将 BP11 的备份文件脱机还原到 BP13

```
//还原数据库:

dmrman CTLSTMT="RESTORE DATABASE 'd:\dpc_data_mac\bp13\DAMENG\dm.ini' FROM
BACKUPSET 'd:\dpc_data_mac\bp11\BACKUP_01'" USE_AP=2

//因为脱机备份没有产生任何 REDO 日志，所以此处省略恢复数据库的操作步骤

//更新数据库:

dmrman CTLSTMT="RECOVER DATABASE 'd:\dpc_data_mac\bp13\DAMENG\dm.ini' UPDATE
DB_MAGIC" USE_AP=2
```

7.2.7.2 配置 BP dm.ini

RAFT_1 组各实例 (BP11、BP12、BP13) 和 RAFT_2 组各实例 (BP21、BP22、BP23) 中 dm.ini 文件的配置方法完全相同。

下面以 BP11 的 dm.ini 为例进行介绍。dm.ini 文件配置如下：

```
ARCH_INI          = 1      #打开归档配置

ALTER_MODE_STATUS = 0      #不允许用户直接通过 SQL 语句修改服务器模式
```

7.2.7.3 配置 BP RAFT 归档文件

修改 DMARCH.INI，配置 RAFT 归档与本地归档。RAFT_1 组配置如下，RAFT_2 组配置可参考此步骤进行配置。

除了本地归档外，其他归档配置项中的 ARCH_DEST 表示实例是 Primary 模式时，需要同步归档数据的目标实例名。

DM8 分布计算集群

1. 配置 BP11 归档文件 DMARCH.INI

当前实例为 BP11，需要向 BP12、BP13 同步数据，因此 RAFT 归档中的 ARCH_DEST 分别配置为 BP12、BP13。

DMARCH.INI 文件内容如下：

```

XNAL_HB_INTERVAL      = 5           #节点通信检测间隔

RAFT_HB_INTERVAL      = 150          #选举心跳间隔

RAFT_VOTE_INTERVAL    = 1500         #选举超时时间,三个库设置不同以尽快选出主库

RAFT_SELF_ID          = 1           #多副本自身节点 ID

[ARCHIVE_RAFT1]

ARCH_TYPE              = RAFT        #RAFT 归档

ARCH_DEST              = BP12        #归档目标实例名

ARCH_DEST_ID           = 2           #归档目标多副本节点 ID

[ARCHIVE_RAFT2]

ARCH_TYPE              = RAFT        #RAFT 归档

ARCH_DEST              = BP13        #归档目标实例名

ARCH_DEST_ID           = 3           #归档目标多副本节点 ID

[ARCHIVE_LOCAL1]

ARCH_TYPE              = LOCAL       #本地归档类型

ARCH_DEST              = d:\dpc_data_mac\bp11\DAMENG\arch  #本地归档文件路径

ARCH_FILE_SIZE         = 128         #本地单个归档文件最大值,单位 MB

ARCH_SPACE_LIMIT       = 0           #本地归档文件总大小,0 表示无限制

```

DM8 分布计算集群

2.配置 BP12 归档配置文件 DMARCH.INI

当前实例为 BP12, 系统配置完成后, 可能在各种故障处理中, BP12 可能选为 Leader。而原来的 Leader 会切换为 Follower。此时 BP12 需要向 BP11 和 BP13 同步数据, 因此 RAFT 归档的 ARCH_DEST 分别配置为 BP11 和 BP13。

```

XNAL_HB_INTERVAL      = 5           #节点通信检测间隔

RAFT_HB_INTERVAL      = 150          #选举心跳间隔

RAFT_VOTE_INTERVAL    = 2000         #选举超时时间

RAFT_SELF_ID          = 2           #多副本自身节点 ID

[ARCHIVE_RAFT1]

ARCH_TYPE              = RAFT        #RAFT 归档

ARCH_DEST              = BP11        #归档目标实例名

ARCH_DEST_ID           = 1           #归档目标多副本节点 ID

[ARCHIVE_RAFT2]

ARCH_TYPE              = RAFT        #RAFT 归档

ARCH_DEST              = BP13        #归档目标实例名

ARCH_DEST_ID           = 3           #归档目标多副本节点 ID

[ARCHIVE_LOCAL1]

ARCH_TYPE              = LOCAL       #本地归档类型

ARCH_DEST              = d:\dpc_data_mac\bp12\DAMENG\arch  #本地归档文件路径

ARCH_FILE_SIZE         = 128         #本地单个归档文件最大值,单位 MB

ARCH_SPACE_LIMIT       = 0           #本地归档文件总大小,0 表示无限制
  
```

DM8 分布计算集群

3.配置 BP13 归档配置文件 DMARCH.INI

当前实例为 BP13,系统配置完成后,可能在各种故障处理中,BP13 可能选为组内 Leader。而原来的 Leader 会切换为 Follower。此时 BP13 需要向 BP11 和 BP12 同步数据,因此 RAFT 归档的 ARCH_DEST 分别配置为 BP11 和 BP12。

```

XNAL_HB_INTERVAL      = 5           #节点通信检测间隔

RAFT_HB_INTERVAL      = 150          #选举心跳间隔

RAFT_VOTE_INTERVAL    = 2500         #选举超时时间

RAFT_SELF_ID          = 3           #多副本自身节点 ID

[ARCHIVE_RAFT1]

ARCH_TYPE              = RAFT        #RAFT 归档

ARCH_DEST              = BP11        #归档目标实例名

ARCH_DEST_ID           = 1          #多副本归档目标节点 ID

[ARCHIVE_RAFT2]

ARCH_TYPE              = RAFT        #RAFT 归档

ARCH_DEST              = BP12        #归档目标实例名

ARCH_DEST_ID           = 2          #多副本归档目标节点 ID

[ARCHIVE_LOCAL1]

ARCH_TYPE              = LOCAL       #本地归档类型

ARCH_DEST              = d:\dpc_data_mac\bp13\DAMENG\arch  #本地归档文件路径

ARCH_FILE_SIZE         = 128         #本地单个归档文件最大值,单位 MB

ARCH_SPACE_LIMIT       = 0          #本地归档文件总大小,0 表示无限制
  
```

DM8 分布计算集群

全部修改完成后，RAFT_1 组中的三个 BP 即配置完成。

RAFT_2 组中的三个 BP 配置方式和 RAFT_1 完全相同，只是需要对路径、实例名等进行区分，具体可参考 7.2.5~7.2.7 小节进行配置。

7.2.8 启动 SP 和 RAFT 组内所有 BP

启动 SP 和 BP 没有先后之分。

启动 SP。

```
dmserver d:\dpc_data_mac\sp1\DAMENG\dm.ini dpc_mode=SP
```

启动 BP。此处以 RAFT_1 组的启动方式为例，三个 BP 的启动顺序没有要求，RAFT_2 组配置完成后，也按照相同方式启动即可。

```
dmserver d:\dpc_data_mac\bp11\DAMENG\dm.ini dpc_mode=BP MOUNT
```

```
dmserver d:\dpc_data_mac\bp12\DAMENG\dm.ini dpc_mode=BP MOUNT
```

```
dmserver d:\dpc_data_mac\bp13\DAMENG\dm.ini dpc_mode=BP MOUNT
```

启动完成后，RAFT 组内的 BP 会自动选出主库，待主库选举完成后，整个 DMDPC 系统即可正常运行。

至此，DMDPC 集群搭建完毕。

MP、SP、BP 全部正常启动，且均处于 OPEN 状态，才是一个正常的 DMDPC 系统！

当 BP 处于下述情况时，表示无可用 BP，系统异常，报错“无效的系统状态”。

无可用 BP 的情况为：

1. BP 处于非 OPEN 状态；



注意：

2. BP 尚未成功启动；

3. BP 全部宕机；

4. BP 如果是多副本环境，确认是否所有 BP 组都已经选举出主库，并且主库

也已切换为 Primary 模式和 Open 状态。

7.2.9 连接 SP

DMDPC 搭建完成后，用户只需要连接对外提供服务的 SP，即可获得完整的数据库服务。

只有出于监控目的进行 V\$ 视图的查询，或出于维护需要时，才允许直连 MP 和 BP，否则不允许再连接 MP 和 BP。切勿在 MP 和 BP 上执行大量 SQL 语句，否则可能会导致系统崩溃。

7.2.10 退出 DMDPC

退出副本架构的 DMDPC 需要按正确顺序使用指定命令退出各节点。

第一步 使用 exit 命令退出所有 SP；

第二步 使用 exit all 命令依次退出每个 RAFT 组。在 BP 的每个 RAFT 组内任一有效节点输入 exit all 命令或者对其发出信号 SIGUSR1（如 kill -10）即可对整个 RAFT 组执行协同退出，由有效 Leader 协同其余有效节点正常退出，无效节点不会参与协同退出。如果执行 exit all 命令的是无效节点，则会转为只在此节点进行退出（exit），多数情况下，无效节点的退出会由于无法推进 C_LSN 强制 HALT；

第三步 退出未参与 exit all 命令的 BP。使用 exit 命令依次退出所有剩余的无效 BP；

第四步 使用 exit 命令退出 MP。

如果没有按正常顺序退出，可能会导致剩下的机器直接宕机。此时，只能对其



注意：

他的机器采用强杀。这种服务器异常退出的情况，当再次重启的时候，服务器需要做大量的 REDO 日志，因此重启的时间会稍长一些，其它没有影响。

7.3 命令行工具部署 DMDPC（MP 多副本架构）

DMDPC 集群规划部署 1 个 SP，1 个 BP 和 1 个 MP。BP 采用单机模式，MP 采用多副本模式（配置成 3 副本）。

DM8 分布计算集群

IP 和端口分配如下表所示，仅用作示例，需按实际 IP 进行设置。

表 7.3 集群规划示意

| RAFT 组名 | 角色 | 实例名称 | IP | PORT_N UM | AP_PORT _NUM | 路径 |
|----------|----|------|---------------|--------------|-----------------|--------------------|
| SPRAFT_1 | SP | SPI | 192.168.1.118 | 5236 | 6000 | e:\mpraft_data\spl |
| BPRAFT_1 | BP | BPI | 192.168.1.118 | 5336 | 6100 | e:\mpraft_data\bp1 |
| 缺省为 | MP | MP1 | 192.168.1.118 | 5436 | 6200 | e:\mpraft_data\mp1 |
| NULL 或者 | MP | MP2 | 192.168.1.118 | 5437 | 6201 | e:\mpraft_data\mp2 |
| MP_RAFT | MP | MP3 | 192.168.1.118 | 5438 | 6202 | e:\mpraft_data\mp3 |

以下章节就以在同一台 Windows 主机为例介绍如何部署。

7.3.1 初始化数据库实例

初始化 5 个实例，分别为 SP、BP 和 3 个 MP 角色。

```

dminit path=e:\mpraft_data\spl instance_name=SPI port_num=5236
ap_port_num=6000 dpc_mode=SP AUTO_OVERWRITE=1
dminit path=e:\mpraft_data\bp1 instance_name=BPI port_num=5336
ap_port_num=6100 dpc_mode=BP AUTO_OVERWRITE=1
dminit path=e:\mpraft_data\mp1 instance_name=MP1 port_num=5436
ap_port_num=6200 dpc_mode=MP AUTO_OVERWRITE=1
dminit path=e:\mpraft_data\mp2 instance_name=MP2 port_num=5437
ap_port_num=6201 dpc_mode=MP AUTO_OVERWRITE=1
dminit path=e:\mpraft_data\mp3 instance_name=MP3 port_num=5438
ap_port_num=6202 dpc_mode=MP AUTO_OVERWRITE=1
    
```

7.3.2 为 SP、BP 和 MP 配置 MP.INI 文件

为 SP、BP 和 MP 实例配置 MP.INI 文件。

MP.INI 文件内容如下：

```
# mp_port 端口号必须与 MP、BP 和 SP 上的 ap_port_num 端口号不冲突

[MP1]

mp_host = 192.168.1.118

mp_port = 10622

[MP2]

mp_host = 192.168.1.118

mp_port = 10623

[MP3]

mp_host = 192.168.1.118

mp_port = 10624
```

将 MP.INI 文件内容分别写入 SP (SP1)、BP (BP1) 和 MP (MP1、MP2、MP3) 的库目录 DAMENG 下。例如：SP 的 MP.INI 位于 e:\mpraft_data\sp1\DAMENG\mp.ini。

7.3.3 配置并启动 MPI

7.3.3.1 配置 MPI 的 dmarch.ini

配置 MPI 的本地归档 dmarch.ini。

```
[ARCHIVE_LOCAL1]

ARCH_TYPE           = LOCAL                #本地归档类型

ARCH_DEST           = e:\mpraft_data\mpi\DAMENG\arch  #本地归档文件路径

ARCH_FILE_SIZE      = 128                  #本地单个归档文件最大值,单位 MB
```

DM8 分布计算集群

```
ARCH_SPACE_LIMIT = 0 #本地归档文件总大小,0 表示无限制
```

7.3.3.2 配置 MPI 的 dm.ini

打开 MPI 归档配置。

```
ARCH_INI = 1
```

7.3.3.3 启动 MPI

```
dmserver e:\mpraft_data\mpi\DAMENG\dm.ini dpc_mode=MP
```

7.3.4 将 MP、SP 和 BP 加入集群

增加 3 个 MP、1 个 SP 和 1 个 BP 节点。只有在注册当前登录 MP 节点后，才可以注册其余节点。后续增加 MP、SP 节点和 BP 节点无先后之分。

```
//搭建 DMDPC 过程中加入 MP、SP 和 BP，必须登录 MP 进行操作

Disql SYSDBA/SYSDBA@192.168.1.118:5436

//增加 MP 节点

//注册当前 MP 实例，MP 的 RAFT 组名可以指定为'MP_RAFT'或缺省 NULL

SP_CREATE_DPC_INSTANCE(NULL, 'MP1', 'MP', 6200, 5436, '192.168.1.118', 'STANDBY', 0, 'MP
instance');

SP_CREATE_DPC_INSTANCE(NULL, 'MP2', 'MP', 6201, 5437, '192.168.1.118', 'STANDBY', 0, 'MP
instance');

SP_CREATE_DPC_INSTANCE(NULL, 'MP3', 'MP', 6202, 5438, '192.168.1.118', 'STANDBY', 0, 'MP
instance');
```

DM8 分布计算集群

```

//增加 BP 节点

//注册 RAFT 组, 名为 BPRAFT_1

SP_CREATE_DPC_RAFT('BP', 'BPRAFT_1');

//在 BPRAFT_1 组内注册 BP 实例 BP1

SP_CREATE_DPC_INSTANCE('BPRAFT_1', 'BP1', 'BP', 6100, 5336, '192.168.1.118', '', 'NORMAL', 1, 'BP
instance');

//注册一个 BP 组, 名为 BG_1

SP_CREATE_DPC_BP_GROUP('BG_1', 'bp group1');

//在 BP 组内添加 RAFT 组

SP_BP_GROUP_ADD_RAFT('BG_1', 'BPRAFT_1');

//增加 SP 节点

//增加 SP, 也要注册 RAFT 组

SP_CREATE_DPC_RAFT('SP', 'SPRAFT_1');

//在 SPRAFT_1 内注册 SP 实例 SPI

SP_CREATE_DPC_INSTANCE('SPRAFT_1', 'SPI', 'SP', 6000, 5236, '192.168.1.118', '', 'NORMAL', 1, 'SP
instance');

//注册三个容错域: FDOM_1 FDOM_2 FDOM_3

SP_CREATE_FAULT_DOMAIN ('FDOM_1', 'shanghai_1');

SP_CREATE_FAULT_DOMAIN ('FDOM_2', 'shanghai_2');

SP_CREATE_FAULT_DOMAIN ('FDOM_3', 'shanghai_3');

//往容错域中添加实例
  
```

DM8 分布计算集群

```

SP_FAULT_DOMAIN_MV_INST('FDM_1', 'MP1');

SP_FAULT_DOMAIN_MV_INST('FDM_1', 'BP1');

SP_FAULT_DOMAIN_MV_INST('FDM_2', 'MP2');

SP_FAULT_DOMAIN_MV_INST('FDM_3', 'MP3');
  
```

7.3.5 检查注册是否成功

查询系统表，检查上一步骤的注册是否成功。能查到相关信息表示注册成功。

```
select * from DPC_BP_GROUP;
```

//查询结果如下:

| 行号 | ID | NAME | DESCRIPTION |
|----|----|------|-------------|
| 1 | 0 | BG_1 | bp group1 |

```
select * from DPC_BP_RAFT;
```

//查询结果如下:

| 行号 | RAFT_ID | GROUP_ID | DPC_MODE | NAME | IS_VALID | INFO1 | INFO2 |
|----|---------|----------|----------|-----------|----------|-------|-------|
| 1 | 0 | -1 | MP | MP_RAFT_1 | NULL | NULL | NULL |
| 2 | 1 | -1 | BP | BPRAFT_11 | NULL | NULL | NULL |
| 3 | 2 | -1 | SP | SPRAFT_11 | NULL | NULL | NULL |

DM8 分布计算集群

NULL

```
select * from DPC_INSTANCE;
```

//查询结果如下:

| 行号 | RAFT_ID | INST_ID | NAME | DPC_MODE | XMAL_PORT | INST_PORT | IP_INTERNAL | SYS_MODE | SYS_STATUS | STATUS | DESCRIPTION | IP_EXTERNAL | INFO1 | INFO2 | INFO3 |
|---------|---------|---------|-------------|----------|---------------|-----------|---------------|----------|------------|--------|-------------|-------------|-------|-------|-------|
| 1 | 0 | 4096 | MPI | MP | 6200 | 5436 | 192.168.1.118 | | | | | | | | |
| STANDBY | 6 | 0 | MP instance | | 192.168.1.118 | 65536 | | | | | | NULL | | | NULL |
| 2 | 0 | 4097 | MP2 | MP | 6201 | 5437 | 192.168.1.118 | | | | | | | | |
| STANDBY | 6 | 0 | MP instance | | 192.168.1.118 | 131072 | | | | | | NULL | | | NULL |
| 3 | 0 | 4098 | MP3 | MP | 6202 | 5438 | 192.168.1.118 | | | | | | | | |
| STANDBY | 6 | 0 | MP instance | | 192.168.1.118 | 196608 | | | | | | NULL | | | |
| 4 | 1 | 4099 | BP1 | BP | 6100 | 5336 | 192.168.1.118 | | | | | | | | |
| NORMAL | 6 | 1 | BP instance | | 192.168.1.118 | 65536 | | | | | | NULL | | | NULL |
| 5 | 2 | 4100 | SP1 | SP | 6000 | 5236 | 192.168.1.118 | | | | | | | | |
| NORMAL | 6 | 1 | SP instance | | 192.168.1.118 | NULL | | | | | | NULL | | | NULL |

7.3.6 准备 MP RAFT 组内数据

7.3.6.1 退出 MPI 并进行脱机备份

首先，退出 MPI。

其次，对 MPI 进行脱机备份。

```
dmrman CTLSTMT="BACKUP DATABASE 'e:\mpraft_data\mp1\DAMENG\dm.ini' FULL TO  
BACKUP_01 BACKUPSET 'e:\mpraft_data\mp1\BACKUP_01'" USE_AP=2
```

7.3.6.2 将 MPI 的备份文件脱机还原到 MP2 及 MP3

将 MPI 的备份文件 BACKUP_01 脱机还原到 MP2 及 MP3。

此处不需要重启 MP2，MP3。如果用户有特殊需要，必须要启动 MP2，MP3，则一定要以 MOUNT 方式启动，否则可能导致 MP2，MP3 生成日志，引发三个节点日志内容不匹配。

```
//还原到 MP2  
  
dmrman CTLSTMT="RESTORE DATABASE 'e:\mpraft_data\mp2\DAMENG\dm.ini' FROM  
BACKUPSET 'e:\mpraft_data\mp1\BACKUP_01'" USE_AP=2  
  
dmrman CTLSTMT="RECOVER DATABASE 'e:\mpraft_data\mp2\DAMENG\dm.ini' UPDATE  
DB_MAGIC" USE_AP=2  
  
//还原到 MP3  
  
dmrman CTLSTMT="RESTORE DATABASE 'e:\mpraft_data\mp3\DAMENG\dm.ini' FROM  
BACKUPSET 'e:\mpraft_data\mp1\BACKUP_01'" USE_AP=2  
  
dmrman CTLSTMT="RECOVER DATABASE 'e:\mpraft_data\mp3\DAMENG\dm.ini' UPDATE  
DB_MAGIC" USE_AP=2
```

7.3.6.3 配置 MP dm.ini

MP 的 RAFT 组各实例（MP1、MP2、MP3）中 dm.ini 文件的配置方法完全相同。

下面以 MP1 的 dm.ini 为例进行介绍。dm.ini 文件配置如下：

```
ARCH_INI          = 1          #打开归档配置
ALTER_MODE_STATUS = 0          #不允许用户直接通过 SQL 语句修改服务器模式
```

7.3.6.4 配置 MP RAFT 归档文件

修改 DMARCH.INI，配置 RAFT 归档与本地归档。

除了本地归档外，其他归档配置项中的 ARCH_DEST 表示实例是 Primary 模式时，需要同步归档数据的目标实例名。

1. 配置 MP1 归档文件 DMARCH.INI

当前实例为 MP1，需要向 MP2、MP3 同步数据，因此 RAFT 归档中的 ARCH_DEST 分别配置为 MP2、MP3。

DMARCH.INI 文件内容如下：

```
XMAL_HB_INTERVAL = 5          #节点通信检测间隔
RAFT_HB_INTERVAL = 150         #选举心跳间隔
RAFT_VOTE_INTERVAL = 1500      #选举超时时间,三个库设置不同以尽快选出主库
RAFT_SELF_ID     = 1           #多副本自身节点 ID

[ARCHIVE_RAFT1]
ARCH_TYPE        = RAFT        #RAFT 归档
ARCH_DEST        = MP2        #归档目标实例名
ARCH_DEST_ID     = 2          #归档目标多副本节点 ID
```

DM8 分布计算集群

```
[ARCHIVE_RAFT2]

ARCH_TYPE          = RAFT          #RAFT 归档

ARCH_DEST          = MP3          #归档目标实例名

ARCH_DEST_ID       = 3           #归档目标多副本节点 ID

[ARCHIVE_LOCAL1]

ARCH_TYPE          = LOCAL        #本地归档类型

ARCH_DEST          = e:\mpraft_data\mpi\DAMENG\arch #本地归档文件路径

ARCH_FILE_SIZE     = 128         #本地单个归档文件最大值,单位 MB

ARCH_SPACE_LIMIT   = 0          #本地归档文件总大小,0 表示无限制
```

2. 配置 MP2 归档文件 DMARCH.INI

```
XMAL_HB_INTERVAL  = 5           #节点通信检测间隔

RAFT_HB_INTERVAL  = 150         #选举心跳间隔

RAFT_VOTE_INTERVAL = 2000       #选举超时时间

RAFT_SELF_ID      = 2           #多副本自身节点 ID

[ARCHIVE_RAFT1]

ARCH_TYPE          = RAFT        #RAFT 归档

ARCH_DEST          = MPI         #归档目标实例名

ARCH_DEST_ID       = 1          #归档目标多副本节点 ID

[ARCHIVE_RAFT2]

ARCH_TYPE          = RAFT        #RAFT 归档
```

DM8 分布计算集群

| | | |
|------------------|----------------------------------|--------------------|
| ARCH_DEST | = MP3 | #归档目标实例名 |
| ARCH_DEST_ID | = 3 | #归档目标多副本节点 ID |
| [ARCHIVE_LOCAL1] | | |
| ARCH_TYPE | = LOCAL | #本地归档类型 |
| ARCH_DEST | = e:\mpraft_data\mp2\DAMENG\arch | #本地归档文件路径 |
| ARCH_FILE_SIZE | = 128 | #本地单个归档文件最大值,单位 MB |
| ARCH_SPACE_LIMIT | = 0 | #本地归档文件总大小,0 表示无限制 |

3. 配置 MP3 归档文件 DMARCH.INI

| | | |
|--------------------|--------|---------------|
| XMAL_HB_INTERVAL | = 5 | #节点通信检测间隔 |
| RAFT_HB_INTERVAL | = 150 | #选举心跳间隔 |
| RAFT_VOTE_INTERVAL | = 2500 | #选举超时时间 |
| RAFT_SELF_ID | = 3 | #多副本自身节点 ID |
| [ARCHIVE_RAFT1] | | |
| ARCH_TYPE | = RAFT | #RAFT 归档 |
| ARCH_DEST | = MP1 | #归档目标实例名 |
| ARCH_DEST_ID | = 1 | #归档目标多副本节点 ID |
| [ARCHIVE_RAFT2] | | |
| ARCH_TYPE | = RAFT | #RAFT 归档 |
| ARCH_DEST | = MP2 | #归档目标实例名 |
| ARCH_DEST_ID | = 2 | #归档目标多副本节点 ID |

DM8 分布计算集群

| | | |
|------------------|----------------------------------|--------------------|
| [ARCHIVE_LOCAL1] | | |
| ARCH_TYPE | = LOCAL | #本地归档类型 |
| ARCH_DEST | = e:\mpraft_data\mp3\DAMENG\arch | #本地归档文件路径 |
| ARCH_FILE_SIZE | = 128 | #本地单个归档文件最大值,单位 MB |
| ARCH_SPACE_LIMIT | = 0 | #本地归档文件总大小,0 表示无限制 |

7.3.7 启动所有 MP

启动 MP。

```
dmserver e:\mpraft_data\mp1\DAMENG\dm.ini dpc_mode=MP MOUNT
dmserver e:\mpraft_data\mp2\DAMENG\dm.ini dpc_mode=MP MOUNT
dmserver e:\mpraft_data\mp3\DAMENG\dm.ini dpc_mode=MP MOUNT
```

启动完成后，RAFT 组内的 MP 会自动选出主库，待主库选举完成后，整个 MP 系统即可正常运行。

DMDPC 运行过程中，MP 需要始终处于开启状态。



搭建时需要严格按照搭建步骤执行并确保配置正确，否则可能会导致集群状态

注意：不正确无法选主等问题。

7.3.8 启动 SP 和 BP

启动 SP 和 BP 没有先后之分。

启动 SP。

```
dmserver e:\mpraft_data\sp1\DAMENG\dm.ini dpc_mode=SP
```

启动 BP。

```
dmserver e:\mpraft_data\bp1\DAMENG\dm.ini dpc_mode=BP
```

至此，DMDPC 集群搭建完毕。

DM8 分布计算集群

MP、SP、BP 全部正常启动，且均处于 OPEN 状态，才是一个正常的 DMDPC 系统!

7.3.9 连接 SP

DMDPC 搭建完成后，用户只需要连接对外提供服务的 SP，即可获得完整的数据库服务。

验证环境搭建是否成功，可以在 SP 上执行查询 V\$instance，看是否所有 RAFT 组中实例都能够查到。

只有出于监控目的进行 V\$视图的查询，或出于维护需要时，才允许直连 MP 和 BP，否则搭建完成后，不允许再连接 MP 和 BP。切勿在 MP 和 BP 上执行大量 SQL 语句，否则可能会导致系统崩溃!

7.3.10 退出 DMDPC

退出副本架构的 DMDPC 需要按正确顺序使用指定命令退出各节点。

第一步 使用 exit 命令退出所有 SP;

第二步 使用 exit 命令退出所有 BP;

第三步 使用 exit all 命令依次退出每个 RAFT 组。在 MP 的每个 RAFT 组内任一有效节点输入 exit all 命令或者对其发出信号 SIGUSR1 (如 kill -10) 即可对整个 RAFT 组执行协同退出，由有效 Leader 协同其余有效节点正常退出，无效节点不会参与协同退出。如果执行 exit all 命令的是无效节点，则会转为只在此节点进行退出 (exit)，多数情况下，无效节点的退出会由于无法推进 C_LSN 强制 HALT;

第四步 退出未参与 exit all 命令的 MP。使用 exit 命令依次退出所有剩余的无效 MP。

如果没有按正常顺序退出，可能会导致剩下的机器直接宕机。此时，只能对其



注意:

他的机器采用强杀。这种服务器异常退出的情况，当再次重启的时候，服务器需要做大量的 REDO 日志，因此重启的时间会稍长一些，其它没有影响。

7.4 图形化部署 DMDPC

DM WEB 版数据库管理工具 (DEM) 提供 DMDPC 的图形化部署与管理功能。关于 DEM 工具的启动, 请查看 DEM 相关文档。DEM 启动后, 通过浏览器访问。

DEM 对 DMDPC 的部署与管理包括集群部署和监控。集群部署在“客户端工具”栏点击“部署集群”可以打开, 集群监控在“监控及告警”栏的“数据库”监控中添加对集群的监控即可打开, 也可以在集群部署完成后通过“打开监控”按钮, 自动添加对所部署的集群的监控。

本章主要对 DMDPC 的部署和监控进行介绍。

7.4.1 DMDPC 图形化部署

在 DEM 的“客户端工具”栏, 选择部署集群工具, 打开新建集群部署的对话框, 如下图:

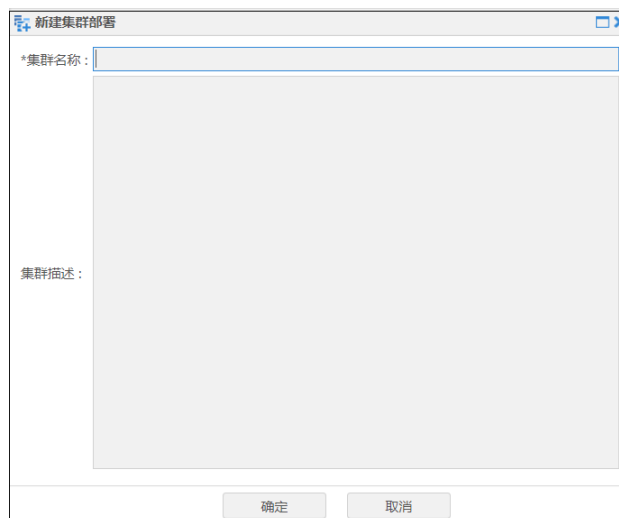


图 7.1 新建集群部署界面

给定集群名称, 点击“确定”以后, 在 DEM 右边面板打开部署面板, 进入“选择部署集群类型”界面:

DM8 分布计算集群

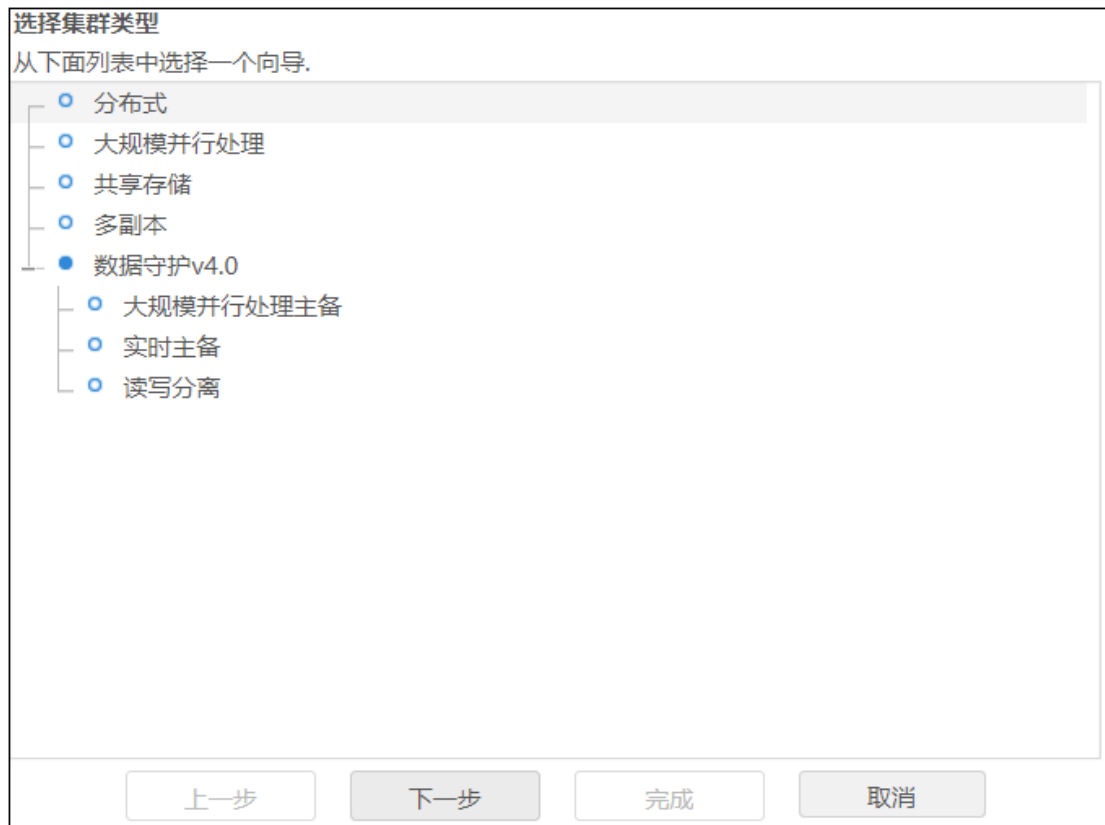
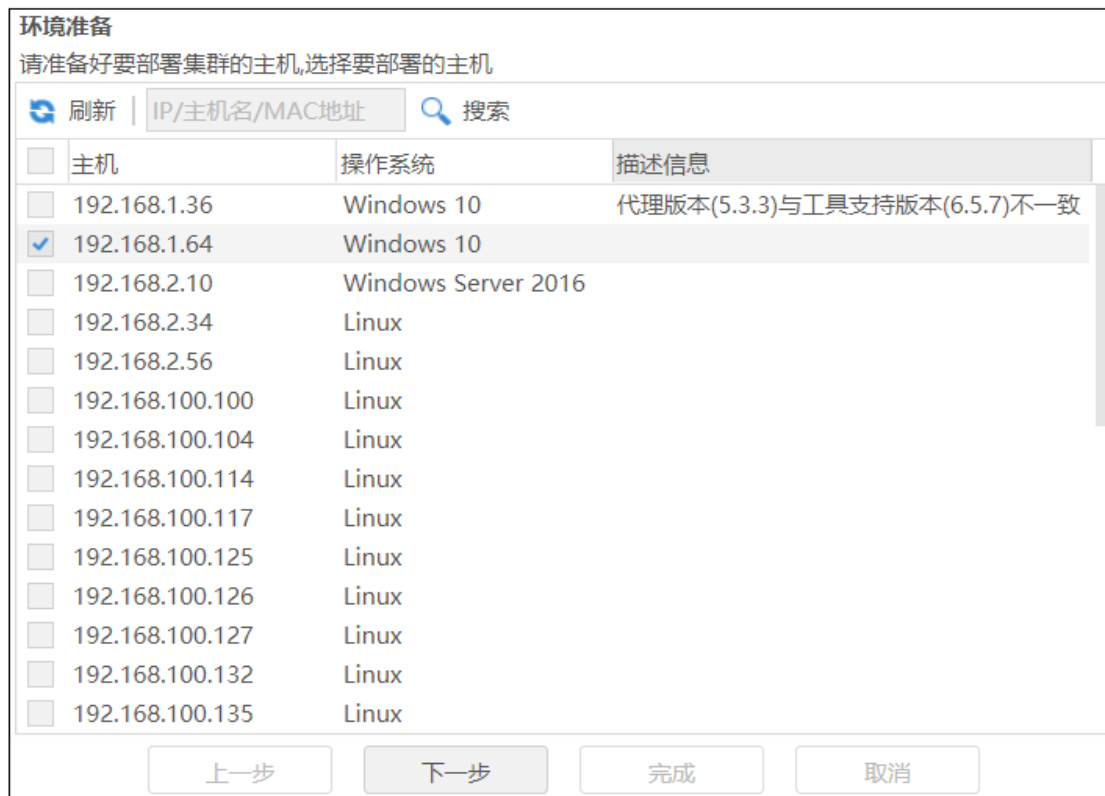


图 7.2 选择部署集群类型

选择分布式，进入“环境准备”界面：



DM8 分布计算集群

图 7.3 环境准备

选择要部署的主机，点击“下一步”，进入“实例规划”界面：

实例规划
配置实例的主机分布,以及划分端口

部署名称：

参数配置
 添加实例
 快速添加实例
 删除实例
 创建混合表空间
 独立执行码
 注册服务

| 主机 | 工作目录 | 类型 | 实例名 | 影子库 | 内部连接IP | PORT_NUM | AP_PORT_NUM | MP_PORT | RAFT组 |
|--------------|-------------|----|-------|-----|--------------|----------|-------------|---------|----------|
| 192.168.1.64 | g:\DMDeploy | SP | SP1 | | 192.168.1.64 | 5230 | 1630 | | RAFT_SP1 |
| | | SP | SP2 | | 192.168.1.64 | 5231 | 1631 | | RAFT_SP2 |
| | | MP | MP_A | | 192.168.1.64 | 5220 | 1620 | 9000 | |
| | | BP | BP1_A | | 192.168.1.64 | 5240 | 1640 | | RAFT_1 |
| | | BP | BP1_B | | 192.168.1.64 | 5241 | 1641 | | RAFT_2 |

图 7.4 实例规划

配置好实例的分布，以及规划好端口后，点击“下一步”，进入“BP 组和 BP 域配置”界面：

BP组和BP域配置
配置BP组和BP域相关信息

BP组：

| RAFT组 | BP组 |
|---------------|------|
| RAFT_1[BP1_A] | BG_1 |
| RAFT_2[BP1_B] | BG_1 |

BP域：

| BP | BP域 |
|-------|-----|
| BP1_A | |
| BP1_B | |

图 7.5 BP 组和 BP 域配置

在该页面可以配置 BP 组和 BP 域信息，默认只有一个 BP 组，没有 BP 域，可根据实际情况进行配置，配置完成后，点击“下一步”，进入“初始化参数配置”界面：

DM8 分布计算集群

初始化参数配置
配置初始化(dminit)相关参数

| 主机 | 实例类型 | 实例名 | 初始化库参数 |
|--------------|------|------|--|
| 192.168.1.64 | SP | SP1 | DB_NAME=DAMENG, EXTENT_SIZE=16, PAGE_SIZE=8, LOG_SIZE=256, TIME_ZONE=+08:00, PAGE_CHECK=0, CHARSET=0, CASE_SENSITIVE=1, BLANK_PAD_MODE=0, LENGTH_IN_CHAR=0, USE_NEW_HASH=1 |
| 192.168.1.64 | SP | SP2 | DB_NAME=DAMENG, EXTENT_SIZE=16, PAGE_SIZE=8, LOG_SIZE=256, TIME_ZONE=+08:00, PAGE_CHECK=0, CHARSET=0, CASE_SENSITIVE=1, BLANK_PAD_MODE=0, LENGTH_IN_CHAR=0, USE_NEW_HASH=1 |
| 192.168.1.64 | MP | MP_A | DB_NAME=DAMENG, EXTENT_SIZE=16, PAGE_SIZE=8, LOG_SIZE=256, TIME_ZONE=+08:00, PAGE_CHECK=0, CHARSET=0, CASE_SENSITIVE=1, BLANK_PAD_MODE=0, LENGTH_IN_CHAR=0, USE_NEW_HASH=1 |

[编辑](#)

SP1初始化库参数配置

数据库名: DAMENG

簇大小: 16 页 页大小: 8 K

日志文件大小: 256 (M:256~2048) 时区设置: +08:00 (-12:59~+14:00)

页面检查: 不启用 字符集: GB18030

USBKEY-PIN: (长度: 1~48) key文件: [浏览](#)

字符串比较大写敏感 空格填充模式 VARCHAR类型以字符为单位

改进的字符串HASH算法 四权分立

启用全库加密 加密算法:

单独配置口令(留空表示使用默认口令) 使用同一口令

SYSDBA: 口令: 确认口令: 默认口令: (SYSDBA)

SYSAUDITOR: (SYSAUDITOR)

自定义库初始化参数: [配置](#)

同步修改所有实例 [应用到其它实例](#)

[上一步](#) [下一步](#) [完成](#) [取消](#)

图 7.6 初始化参数配置

在该页面配置初始化参数信息，目前，每个实例的初始化参数保持一致，配置完成后，点击“下一步”，进入“dm.ini 配置”界面：

dm.ini配置
配置dm.ini相关参数

| 实例类型 | 实例名 | 参数配置详情 |
|------|---------------|--|
| SP | SP1 | INSTANCE_NAME=SP1, PORT_NUM=5230 |
| SP | SP2 | INSTANCE_NAME=SP2, PORT_NUM=5231 |
| MP | MP_A | INSTANCE_NAME=MP_A, PORT_NUM=5220, ARCH_INI=1 |
| BP | BP1_A(RAFT_1) | INSTANCE_NAME=BP1_A, PORT_NUM=5240, ARCH_INI=1 |
| BP | BP1_B(RAFT_2) | INSTANCE_NAME=BP1_B, PORT_NUM=5241, ARCH_INI=1 |

[编辑](#)

自定义参数配置: [配置...](#)

同步修改其它实例 [应用到其它实例](#)

[上一步](#) [下一步](#) [完成](#) [取消](#)

图 7.7 dm.ini 配置

在该页面配置各实例的 dm.ini，默认为初始化生成的默认值，如需修改，则通过自定义参数配置，以“参数名=参数值”且多个参数用逗号分开方式配置，配置完成后，点击“下一步”，进入“dmarch.ini 配置”界面：

DM8 分布计算集群

dmarch.ini配置
MP/BP实例配置归档信息

ARCH_RESERVE_TIME: (M)

| 实例类型 | 实例名 | 参数配置详情 |
|------|---------------|---|
| BP | BP1_A(RAFT_1) | ARCH_DEST=g:\DMDeploy\dmdpc\BP1_A\DAMENG\arch, ARCH_FILE_SIZE=128, ARCH_SPACE_LIMIT=0, ARCH_FLUSH=0 |
| BP | BP1_B(RAFT_2) | ARCH_DEST=g:\DMDeploy\dmdpc\BP1_B\DAMENG\arch, ARCH_FILE_SIZE=128, ARCH_SPACE_LIMIT=0, ARCH_FLUSH=0 |
| MP | MP_A | ARCH_DEST=g:\DMDeploy\dmdpc\MP_A\DAMENG\arch, ARCH_FILE_SIZE=128, ARCH_SPACE_LIMIT=0, ARCH_FLUSH=0 |

[编辑](#)

ARCH_DEST: 使用默认值

ARCH_FILE_SIZE: (M:64~2048)

ARCH_SPACE_LIMIT: (M:1024~4294967294,0表示无限制)

ARCH_FLUSH_BUF_SIZE: (M:0~128,0表示不使用归档合并刷盘)

图 7.8 dmarch.ini 配置

在该页面配置 BP 和 MP 的 dmarch.ini，默认为初始化生成的默认值，如需修改，则通过自定义参数配置，以“参数名=参数值”且多个参数用逗号分开方式配置，配置完成后，点击“下一步”，进入“上传服务器文件”界面，请选择上传的服务器文件：

上传服务器文件
选择要上传的服务器文件或者使用主机上已存在服务器执行码目录

各节点将使用同一个达梦数据库服务器文件
 各节点将单独配置达梦数据库服务器文件
 各节点已部署达梦数据库服务器文件

| 应用主机 | 文件名 | 操作 |
|-----------------|---|-----|
| 1 所有为windows的主机 | dm8_20230112_x86_win_64_sec_8.1.2.192_pack1.iso | + - |

使用ssh通讯加密

图 7.9 上传服务器文件

上传完服务器文件后，点击“下一步”，进入“详情总览”界面。列出将要部署的集群环境的所有配置信息：

DM8 分布计算集群

详情总览

详情总览

部署类型: 分布式
部署名称: dmdpc

【配置环境说明】

| HOST_NAME | IP | EP LIST | SYSTEM | COMMENTS |
|------------|--------------|---|------------|--|
| guoyanfeng | 192.168.1.64 | SP:SP1 SP:SP2 MP:MP_A BP:BP1_A BP:BP1_B | Windows 10 | OuterIP:192.168.1.64 InnerIP:192.168.1.64 |

【端口规划】

| INSTANCE_NAME | PORT_NUM | AP_PORT_NUM |
|---------------|----------|-------------|
| [SP]SP1 | 5230 | 1630 |
| [SP]SP2 | 5231 | 1631 |
| [MP]MP_A | 5220 | 1620 |
| [BP]BP1_A | 5240 | 1640 |
| [BP]BP1_B | 5241 | 1641 |

【MP, BP, SP初始化配置】
初始化参数

- 数据库名 : DAMENG
- 库大小 : 1G

上一步
保存并执行
完成
取消

图 7.10 详情总览

点击“下一步”，开始执行部署任务。执行过程如下：

DM8 分布计算集群

创建并执行任务
正在执行任务
进度:任务总数:32,已完成:12,出错:0,取消:0,剩余:20,开始时间:2023-01-30 09:52:55

| 任务 | 状态 | 消息 | 耗时 |
|------------------------------------|---------|---------|----------|
| ✓ 上传服务器文件到主机192.168.1.64 | 成功 | | 15秒174毫秒 |
| ✓ 配置实例[SP]SP1(192.168.1.64:5230) | | | |
| ✓ 数据库初始化 | 成功 | | 406毫秒 |
| ✓ 配置mp.ini | 成功 | | 15毫秒 |
| ✓ 配置dm.ini | 成功 | | 16毫秒 |
| 启动实例 | 等待执行... | | 0毫秒 |
| ✓ 配置实例[SP]SP2(192.168.1.64:5231) | | | |
| ✓ 数据库初始化 | 成功 | | 390毫秒 |
| ✓ 配置mp.ini | 成功 | | 16毫秒 |
| ✓ 配置dm.ini | 成功 | | 78毫秒 |
| 启动实例 | 等待执行... | | 0毫秒 |
| ✓ 配置实例[MP]MP_A(192.168.1.64:5220) | | | |
| ▶ 数据库初始化 | 正在执行... | 执行初始化命令 | |
| 配置mp.ini | 等待执行... | | 0毫秒 |
| 配置dmarch.ini | 等待执行... | | 0毫秒 |
| 配置dm.ini | 等待执行... | | 0毫秒 |
| 启动实例 | 等待执行... | | 0毫秒 |
| 注册MP,BP,SP | 等待执行... | | 0毫秒 |
| ✓ 配置实例[BP]BP1_A(192.168.1.64:5240) | | | |
| ▶ 数据库初始化 | 正在执行... | 执行初始化命令 | |
| 配置mp.ini | 等待执行... | | 0毫秒 |
| 配置dmarch.ini | 等待执行... | | 0毫秒 |

刷新 查看部署信息 停止所有任务 回滚所有任务 重做失败任务

上一步 下一步 重新配置 取消

图 7.11 执行部署任务-正在部署

部署任务执行完成后显示如下:

创建并执行任务
任务执行完成!
进度:任务总数:32,已完成:32,出错:0,取消:0,剩余:0,开始时间:2023-01-30 09:52:55,结束时间:2023-01-30 09:53:35,耗时:40秒551毫秒

| 任务 | 状态 | 消息 | 耗时 |
|------------------------------------|----|----|----------|
| ✓ 上传服务器文件到主机192.168.1.64 | 成功 | | 15秒174毫秒 |
| ✓ 配置实例[SP]SP1(192.168.1.64:5230) | | | |
| ✓ 数据库初始化 | 成功 | | 406毫秒 |
| ✓ 配置mp.ini | 成功 | | 15毫秒 |
| ✓ 配置dm.ini | 成功 | | 16毫秒 |
| ✓ 启动实例 | 成功 | | 5秒767毫秒 |
| ✓ 配置实例[SP]SP2(192.168.1.64:5231) | | | |
| ✓ 数据库初始化 | 成功 | | 390毫秒 |
| ✓ 配置mp.ini | 成功 | | 16毫秒 |
| ✓ 配置dm.ini | 成功 | | 78毫秒 |
| ✓ 启动实例 | 成功 | | 5秒767毫秒 |
| ✓ 配置实例[MP]MP_A(192.168.1.64:5220) | | | |
| ✓ 数据库初始化 | 成功 | | 3秒656毫秒 |
| ✓ 配置mp.ini | 成功 | | 16毫秒 |
| ✓ 配置dmarch.ini | 成功 | | 32毫秒 |
| ✓ 配置dm.ini | 成功 | | 31毫秒 |
| ✓ 启动实例 | 成功 | | 9秒126毫秒 |
| ✓ 注册MP,BP,SP | 成功 | | 62毫秒 |
| ✓ 配置实例[BP]BP1_A(192.168.1.64:5240) | | | |
| ✓ 数据库初始化 | 成功 | | 2秒751毫秒 |
| ✓ 配置mp.ini | 成功 | | 15毫秒 |

刷新 查看部署信息 停止所有任务 回滚所有任务 重做失败任务

上一步 下一步 重新配置 添加到监控

图 7.12 执行部署任务-部署完成

DM8 分布计算集群

至此，基于 DMDPC 搭建完成。

点击“添加到监控”可以把该集群添加到数据库监控中，并且可以直接打开数据库监控界面。

7.4.2 DMDPC 图形化监控

在 DEM 的“监控与告警”栏，双击“数据库”，在 DEM 右边面板打开数据库监控面板。

| 数据库 | 连接 | 健康度 | 运行时间 | 会话 | 每秒事务 | CPU | 内存 | 交换区 | 磁盘读 | 磁盘写 | 网络读 | 网络写 | 告警 | 操作 |
|------------------------|----|-----|---------|--------|------|-----|----------|-----|---------|--------|----------|-----------|----|-----|
| ▼ DPC - dmdpc | | | | 站点数: 5 | | | 会话数: 5/7 | | 重启次数: 1 | | | | | |
| [SP] 192.168.1.64:5231 | 成功 | 92 | 11.56分钟 | 1/2 | 5 | 0% | 2.72% | 0% | 0Bps | 0Bps | 1.36KBps | 14.49KBps | 0 | ... |
| [SP] 192.168.1.64:5230 | 成功 | 92 | 11.55分钟 | 1/2 | 1 | 0% | 2.85% | 0% | 0Bps | 0Bps | 750Bps | 13.57KBps | 0 | ... |
| [MP] 192.168.1.64:5220 | 成功 | 95 | 11.82分钟 | 1/1 | 10 | 0% | 4.24% | 0% | 0Bps | 96KBps | 2.35KBps | 16.81KBps | 0 | ... |
| [BP] 192.168.1.64:5241 | 成功 | 95 | 11.64分钟 | 1/1 | 1 | 0% | 5.24% | 0% | 0Bps | 0Bps | 750Bps | 13.57KBps | 0 | ... |
| [BP] 192.168.1.64:5240 | 成功 | 95 | 11.68分钟 | 1/1 | 4 | 0% | 5.42% | 0% | 0Bps | 16KBps | 1.72KBps | 15.74KBps | 0 | ... |

图 7.13 监控面板

DMDPC 监控把 DMDPC 的每个节点当成数据库实例来监控，其支持的监控项与普通数据库实例相同，详细参见 DEM 相关文档。

7.5 命令行工具搭建多副本影子库

影子库的搭建和现有的 RAFT 库搭建基本相同，只有备份还原的步骤有差异，其他配置方法完全相同，除备份还原步骤外，其他步骤都可以参考 [7.2 命令行工具部署 DMDPC \(BP 副本架构\)](#) 和 [7.3 命令行工具部署 DMDPC \(MP 副本架构\)](#) 小节。

7.5.1 影子库的备份还原

影子库的备份还原有两种方式，分别说明如下：

1. 可以对 RAFT 主库做联机或脱机备份，备份时指定 SHADOW 关键字，然后按照正常的还原恢复步骤还原出一个影子库。

例 以脱机备份为例。

```
//备份数据库
```

DM8 分布计算集群

```

dmrman CTLSTMT="BACKUP DATABASE 'd:\dpc_data_mac\bp11\DAMENG\dm.ini' FULL
SHADOW TO BACKUP_02 BACKUPSET 'd:\dpc_data_mac\bp11\BACKUP_02'" USE_AP=2

//还原数据库:

dmrman CTLSTMT="RESTORE DATABASE 'd:\dpc_data_mac\bp12\DAMENG\dm.ini' FROM
BACKUPSET 'd:\dpc_data_mac\bp11\BACKUP_02'" USE_AP=2

//因为脱机备份没有产生任何 REDO 日志，所以此处省略恢复数据库的操作步骤

//更新数据库:

dmrman CTLSTMT="RECOVER DATABASE 'd:\dpc_data_mac\bp12\DAMENG\dm.ini' UPDATE
DB_MAGIC" USE_AP=2
  
```

2. 可以对 RAFT 主库做联机或脱机备份，备份时不指定 SHADOW 关键字，在还原时指定 TO SHADOW 关键字，还原出一个影子库。

例 以脱机备份为例。

```

//备份数据库

dmrman CTLSTMT="BACKUP DATABASE 'd:\dpc_data_mac\bp11\DAMENG\dm.ini' FULL TO
BACKUP_02 BACKUPSET 'd:\dpc_data_mac\bp11\BACKUP_02'" USE_AP=2

//还原数据库:

dmrman CTLSTMT="RESTORE DATABASE 'd:\dpc_data_mac\bp12\DAMENG\dm.ini' TO
SHADOW FROM BACKUPSET 'd:\dpc_data_mac\bp11\BACKUP_02'" USE_AP=2

//因为脱机备份没有产生任何 REDO 日志，所以此处省略恢复数据库的操作步骤

//更新数据库:

dmrman CTLSTMT="RECOVER DATABASE 'd:\dpc_data_mac\bp12\DAMENG\dm.ini' UPDATE
DB_MAGIC" USE_AP=2
  
```

7.5.2 影子库与 RAFT 库之间的转换

RAFT 库和影子库之间的转换方式说明如下：

1. 将 RAFT 库转换为影子库

- 1) 选择集群中某个 RAFT 库执行单独退出，不能是协同退出方式。
- 2) 在相同机器上新初始化一个影子库，保留 IP/PORT、实例名等信息不变，这样不需要改其他 RAFT 库的配置。

3) 使用 SHADOW 方式备份主库、还原影子库。

4) 启动影子库，预期可以正常跟随主库同步日志。

2. 将影子库转换为 RAFT 库

1) 退出影子库。

2) 使用正常备份还原方式，备份主库，还原一个新的 RAFT 目标库。

3) 新的 RAFT 库使用和影子库相同的 IP/PORT、实例名等配置信息，这样不需要改其他 RAFT 库的配置。

4) 启动新的 RAFT 库，预期可以正常跟随主库同步日志。

8 DMDPC 集群的管理与使用

8.1 集群信息查看

集群对外提供服务的只有 SP 节点，BP 和 MP 在集群搭建完毕后一般不需登录。连接 SP，通过查询相关系统表和动态视图可了解集群信息。具体的系统表和视图请参考 [12 相关系统表和动态视图](#)。

例 1 查看当前节点的 DPC_MODE 模式信息。0: 无; 1: MP; 2: BP; 3: SP; 4: BS。

```
select * from V$dm_ini where para_name='DPC_MODE'and  
sf_get_ep_seqno(rowid)=SF_GET_SELF_EP_SEQNO;
```

例 2 连接 SP，查询集群中 BP GROUP 的相关信息。

```
select * from DPC_BP_GROUP;
```

例 3 获取会话所在实例的 RAFT 序号 RAFT_ID。其中，MP 只有一个 RAFT，因此 RAFT_ID 定为 0。

```
SELECT SF_GET_SELF_EP_SEQNO();
```

例 4 根据 ROWID 获取本行数据所在的实例 RAFT_ID。

```
SELECT SF_GET_EP_SEQNO(ROWID); //此语句查询的是 SYSDUAL 系统表，都认为是从 MP 获取的  
数据，因此返回 MP 的 RAFT_ID 为 0
```

例 5 获取当前会话连接的实例名。

```
SELECT NAME FROM V$INSTANCE WHERE SF_GET_EP_SEQNO(ROWID) =  
SF_GET_SELF_EP_SEQNO();
```

例 6 获取 DMDPC 系统内所有实例的会话信息。

```
SELECT * FROM V$SESSIONS;
```

例 7 获取当前连接的实例上的所有会话信息。

DM8 分布计算集群

```
SELECT * FROM V$SESSIONS WHERE SF_GET_EP_SEQNO(ROWID) = SF_GET_SELF_EP_SEQNO();
```

例 8 获取表 TEST 在每个实例上的数据行数。

```
CALL SP_GET_EP_COUNT('SYSDBA','TEST');
```

例 9 获取表 TEST 已使用的页数。

```
SELECT TABLE_USED_PAGES('SYSDBA','TEST');
```

8.2 动态增删节点

本章详细介绍动态增删节点的操作步骤。更多的理论介绍请参考 [5.10 动态增删节点](#)。

8.2.1 横向增删

横向增删支持动态增删 BP 单机节点或 SP 单机节点。

横向增删既可以登录到 SP 上执行，也可以登录到 MP 上执行。

单机 DMDPC 或多副本 DMDPC 中均支持横向增删。下面以单机 DMDPC 为例，介绍 BP、SP 单机节点的增删操作。

8.2.1.1 BP 节点（单机架构）

8.2.1.1.1 增加 BP

动态添加 BP 节点的流程：第一步，注册一个新的 RAFT 组；第二步，在 RAFT 组中注册实例；第三步，注册一个新的 BP 组并添加 RAFT 组；第四步，将 MP.INI 文件拷贝到 BP 的库目录 DAMENG 下。

如果使用已经注册的 RAFT 组和 BP 组，则前两步可以省略，在第三步中直接使用已有的 RAFT 组即可。

使用的过程如下：

```
//第一步 创建一个新的 RAFT 组，名称为 RAFT_1
```

DM8 分布计算集群

```

SP_CREATE_DPC_RAFT('BP', 'RAFT_1');

//第二步 注册一个 BP 实例，名称为 BP_1。此处的实例 BP_1 是提前初始化好的

SP_CREATE_DPC_INSTANCE('RAFT_1', 'BP_1', 'BP', 6001, 5237, '223.254.30.136', ",NORMAL", 1, 'BP
instance');

//第三步 创建一个新的 BP 组，名称为 BG_1

SP_CREATE_DPC_BP_GROUP('BG_1', 'bp group1');

//在 BP 组内添加 RAFT 组

SP_BP_GROUP_ADD_RAFT('BG_1', 'RAFT_1');

//第四步 将 MP.INI 文件拷贝到 BP 的库目录 DAMENG 下
  
```

8.2.1.1.2 删除 BP

同样，当业务规模缩小时，用户也可以从集群中移除 BP。删除 BP 节点的流程：第一步，移除实例；第二步，删除 RAFT 组，必须保证 RAFT 组内的 BP 实例已全部移除，才能删掉 RAFT；第三步，删除 BP 组。

使用的过程如下：

```

//第一步 移除 BP 实例

SP_DROP_DPC_INSTANCE('BP_1');

//第二步 删除 RAFT 组

SP_DROP_DPC_BP_RAFT('RAFT_1');

//第三步 删除 BP 组

SP_DROP_DPC_BP_GROUP('BG_1');
  
```

8.2.1.2 SP 节点（单机架构）

8.2.1.2.1 增加 SP

动态添加 SP 节点的流程：第一步，注册一个新的 RAFT 组；第二步，在 RAFT 组中注册实例；第三步，将 MP.INI 文件拷贝到 SP 的库目录 DAMENG 下。

如果使用已经注册的 RAFT 组，则第一步可以省略，在第二步中直接使用已有的 RAFT 组即可。

一个 SP 类型的 RAFT 组最多只能加入一个 SP 节点。

使用的过程如下：

```
//第一步，为 SP 创建一个新的 RAFT 组，名称为 RAFT_SPI  
  
SP_CREATE_DPC_RAFT('SP', 'RAFT_SPI');  
  
//第二步，在 RAFT_SPI 内注册 SP 实例，名称为 SP_1。此处的实例 SP_1 是提前初始化好的  
  
SP_CREATE_DPC_INSTANCE('RAFT_SPI', 'SP_1', 'SP', 6000, 5236, '223.254.30.136', '', 'NORMAL', 2, 'SP  
instance');  
  
//第三步 将 MP.INI 文件拷贝到 SP 的库目录 DAMENG 下
```

8.2.1.2.2 删除 SP

动态移除 SP 的流程：第一步，移除实例；第二步，移除 RAFT 组，必须保证 RAFT 组内的 SP 实例已经移除，才能删掉 RAFT。

使用的过程如下：

```
//第一步 移除 SP 实例  
  
SP_DROP_DPC_INSTANCE('SP_1');  
  
//第二步 删除 RAFT 组  
  
SP_DROP_DPC_BP_RAFT('RAFT_SPI');
```

8.2.2 纵向增删

纵向增删支持动态增删 MP 和 BP 节点。

纵向增删 MP 副本节点，需要登录到 MP 主节点上执行。纵向增删 BP 多副本节点，需要登录到以 BS 模式运行的 BP 主节点上执行。

仅多副本系统支持纵向增删。下面以多副本 DMDPC 为例，介绍 BP 多副本系统、MP 多副本系统的增删操作。

8.2.2.1 BP 节点（多副本架构）

8.2.2.1.1 增加 BP

多副本架构下动态添加 BP 节点的流程与非多副本架构存在差异。在多副本架构下对增删节点操作存在部分限制，说明如下：

1. 增加或删除后的多副本集群中节点需要为奇数。
2. 暂时不允许删除主节点，若要删除主节点，则需要执行手动主库切换动作，将待删除的节点变为备库。

下面介绍多副本集群下增删节点操作流程：

多副本架构增加 BP 节点流程如下：

第一步，初始化节点。

初始化 BP 节点，并将 BP 节点注册对应 RAFT 组中。

```
//初始化 BP4, BP5 节点  
  
dminit path=e:\dpc_data\bp4 instance_name=BP4 port_num=5242 ap_port_num=6606  
  
dpc_mode=BP  
  
dminit path=e:\dpc_data\bp5 instance_name=BP5 port_num=5243 ap_port_num=6607  
  
dpc_mode=BP
```

DM8 分布计算集群

```
//将新初始化的节点注册进待新增节点的 RAFT_1 组中

SP_CREATE_DPC_INSTANCE('RAFT_1', 'BP4', 'BP', 6606, 5242, '192.168.1.68', 'STANDBY', 0, 'BP
instance');

SP_CREATE_DPC_INSTANCE('RAFT_1', 'BP5', 'BP', 6607, 5243, '192.168.1.68', 'STANDBY', 0, 'BP
instance');
```

第二步，修改配置。

1. 打开 BP4, BP5 节点 dm.ini 归档选项 ARCH_INI = 1。
2. 配置 dmarch.ini 文件。

BP4 的归档配置如下：

```
XMAL_HB_INTERVAL      = 5           #节点通信检测间隔
RAFT_HB_INTERVAL      = 150          #选举心跳间隔
RAFT_VOTE_INTERVAL    = 1500         #选举超时时间,三个库设置不同以尽快选出主库
XMAL_IP                = 192.168.1.68 #本地通信 IP
XMAL_PORT              = 6606         #本地通信端口
RAFT_SELF_ID          = 4            #指定自身编号
RAFT_LEARNER           = 1           #指定自己是 Learner 节点

[ARCHIVE_RAFT1]
ARCH_TYPE              = RAFT         #RAFT 归档
ARCH_DEST              = BP1          #归档目标实例名
ARCH_DEST_IP           = 192.168.1.68 #归档目标 IP
ARCH_DEST_PORT         = 10008        #归档目标通信端口
ARCH_DEST_ID           = 1           #指定归档目标节点编号
```

DM8 分布计算集群

```
[ARCHIVE_RAFT2]

ARCH_TYPE           = RAFT           #RAFT 归档
ARCH_DEST           = BP2             #归档目标实例名
ARCH_DEST_IP        = 192.168.1.68 #归档目标 IP
ARCH_DEST_PORT      = 10009        #归档目标通信端口
ARCH_DEST_ID        = 2             #指定归档目标节点编号

[ARCHIVE_RAFT3]

ARCH_TYPE           = RAFT           #RAFT 归档
ARCH_DEST           = BP3             #归档目标实例名
ARCH_DEST_IP        = 192.168.1.68 #归档目标 IP
ARCH_DEST_PORT      = 10010        #归档目标通信端口
ARCH_DEST_ID        = 3             #指定归档目标节点编号

[ARCHIVE_RAFT4]

ARCH_TYPE           = LEARNER        #RAFT 归档
ARCH_DEST           = BP5             #归档目标实例名
ARCH_DEST_IP        = 192.168.1.68 #归档目标 IP
ARCH_DEST_PORT      = 6607          #归档目标通信端口
ARCH_DEST_ID        = 5             #指定归档目标节点编号

[ARCHIVE_LOCAL1]

ARCH_TYPE           = LOCAL          #本地归档类型
ARCH_DEST           = E:\RAFT_DB\ARCH\BP4 #本地归档文件路径
```

DM8 分布计算集群

| | | |
|------------------|-------|---------------------|
| ARCH_FILE_SIZE | = 128 | #本地单个归档文件最大值, 单位 MB |
| ARCH_SPACE_LIMIT | = 0 | #本地归档文件总大小, 0 表示无限制 |

BP5 的归档配置如下:

| | | |
|--------------------|----------------|-------------------------|
| XMAL_HB_INTERVAL | = 5 | #节点通信检测间隔 |
| RAFT_HB_INTERVAL | = 150 | #选举心跳间隔 |
| RAFT_VOTE_INTERVAL | = 1500 | #选举超时时间, 三个库设置不同以尽快选出主库 |
| XMAL_IP | = 192.168.1.68 | #本地通信 IP |
| XMAL_PORT | = 6607 | #本地通信端口 |
| RAFT_SELF_ID | = 5 | #指定自身编号 |
| RAFT_LEARNER | = 1 | #指定自己是 Learner 节点 |
| [ARCHIVE_RAFT1] | | |
| ARCH_TYPE | = RAFT | #RAFT 归档 |
| ARCH_DEST | = BP1 | #归档目标实例名 |
| ARCH_DEST_IP | = 192.168.1.68 | #归档目标 IP |
| ARCH_DEST_PORT | = 10008 | #归档目标通信端口 |
| ARCH_DEST_ID | = 1 | #指定归档目标节点编号 |
| [ARCHIVE_RAFT2] | | |
| ARCH_TYPE | = RAFT | #RAFT 归档 |
| ARCH_DEST | = BP2 | #归档目标实例名 |
| ARCH_DEST_IP | = 192.168.1.68 | #归档目标 IP |
| ARCH_DEST_PORT | = 10009 | #归档目标通信端口 |
| ARCH_DEST_ID | = 2 | #指定归档目标节点编号 |

DM8 分布计算集群

```
[ARCHIVE_RAFT3]

ARCH_TYPE          = RAFT          #RAFT 归档

ARCH_DEST          = BP3            #归档目标实例名

ARCH_DEST_IP       = 192.168.1.68   #归档目标 IP

ARCH_DEST_PORT     = 10010         #归档目标通信端口

ARCH_DEST_ID       = 3             #指定归档目标节点编号

[ARCHIVE_RAFT4]

ARCH_TYPE          = LEARNER       #RAFT 归档

ARCH_DEST          = BP4            #归档目标实例名

ARCH_DEST_IP       = 192.168.1.68   #归档目标 IP

ARCH_DEST_PORT     = 6606         #归档目标通信端口

ARCH_DEST_ID       = 4             #指定归档目标节点编号

[ARCHIVE_LOCAL1]

ARCH_TYPE          = LOCAL         #本地归档类型

ARCH_DEST          = E:\RAFT_DB\ARCH\BP5 #本地归档文件路径

ARCH_FILE_SIZE     = 128           #本地单个归档文件最大值，单位 MB

ARCH_SPACE_LIMIT   = 0            #本地归档文件总大小，0 表示无限制
```

3. 配置 mp.ini 与本集群的其他节点 mp.ini 文件内容一致。

第三步，准备节点数据。

1. 启动 DMAP，连接 BP1，联机备份 RAFT_1 组中主节点 BP1。

```
SQL>BACKUP DATABASE BACKUPSET 'E:\dpc\bpl\DAMENG\bak';
```

DM8 分布计算集群

得到 BP1 的备份集 E:\dpc\bp1\DAMENG\bak_BP1。

2. 还原 BP4, BP5 节点。

//还原 BP4。其中 BAK_MAGIC 可通过 DMRMAN 工具 SHOW 命令查看备份集得知。

```
RMAN>SHOW BACKUPSET 'E:\dpc\bp1\DAMENG\bak_BP1';
```

//假设 BAK_MAGIC 为 1903999294。

```
RMAN>RESTORE DATABASE 'E:\dpc\bp4\DAMENG\dm.ini' FROM BACKUPSET
```

```
'E:\dpc\bp1\DAMENG\bak_BP1' USE BAK_MAGIC 1903999294;
```

```
RMAN>RECOVER DATABASE 'E:\dpc\bp4\DAMENG\dm.ini' FROM BACKUPSET
```

```
'E:\dpc\bp1\DAMENG\bak_BP1' USE BAK_MAGIC 1903999294 UNTIL END_LSN;
```

```
RMAN>RECOVER DATABASE 'E:\dpc\bp4\DAMENG\dm.ini' UPDATE DB_MAGIC;
```

//还原 BP5

```
RMAN>RESTORE DATABASE 'E:\dpc\bp5\DAMENG\dm.ini' FROM BACKUPSET
```

```
'E:\dpc\bp1\DAMENG\bak_BP1' USE BAK_MAGIC 1903999294;
```

```
RMAN>RECOVER DATABASE 'E:\dpc\bp5\DAMENG\dm.ini' FROM BACKUPSET
```

```
'E:\dpc\bp1\DAMENG\bak_BP1' USE BAK_MAGIC 1903999294 UNTIL END_LSN;
```

```
RMAN>RECOVER DATABASE 'E:\dpc\bp5\DAMENG\dm.ini' UPDATE DB_MAGIC;
```

第四步，启动节点，执行增加节点操作。

1. 启动 BP4, BP5。

2. 连接 RAFT_1 组中主库 BP1，执行增加节点流程。

//将 BP4, BP5 以 leaner 节点的方式加入集群，从而在不影响集群使用条件下向新节点同步日志

```
SP_ADD_RAFT_LEARNER('BP4','192.168.1.68',6606,4);
```

```
SP_ADD_RAFT_LEARNER('BP5','192.168.1.68',6607,5);
```

//将新节点加入集群

```
SP_ALTER_RAFT_NODE('BP1/BP2/BP3/BP4/BP5');
```

8.2.2.1.2 删除 BP

执行节点删除操作如下：

1. 连接 RAFT_1 中主节点 BP1，执行删除节点操作。

```
//执行  
SP_ALTER_RAFT_NODE('BP1/BP2/BP3');  
  
或  
SP_DELETE_RAFT_NODE('BP4/BP5');
```

2. 连接 MP 移除 BP 实例。

```
//移除 BP4, BP5 实例  
SP_DROP_DPC_INSTANCE('BP4');  
  
SP_DROP_DPC_INSTANCE('BP5');
```

8.2.2.2 MP 节点（多副本架构）

在多副本架构下对增删 MP 节点操作存在部分限制，说明如下：

1. 由于 MP.INI 暂不支持动态修改，因此若执行 MP 的动态增删操作时，需要额外手动修改集群内其他节点的 MP.INI 文件。
2. 增加或删除后的多副本集群中节点需要为奇数。
3. 暂时不允许删除主节点。若要删除主节点，则需要执行手动主库切换动作，将待删除的节点变为备库，才可删除。

下面介绍多副本集群下增删节点操作流程。

8.2.2.2.1 增加 MP

多副本架构增加 MP 节点流程如下：

DM8 分布计算集群

第一步，初始化节点。

初始化 MP 节点，并将 MP 节点注册对应 RAFT 组中。

```
//初始化 MP4, MP5 节点

dminit path=e:\dpc_data\mp4 instance_name=MP4 port_num=5242 ap_port_num=6606

dpc_mode=MP

dminit path=e:\dpc_data\mp5 instance_name=MP5 port_num=5243 ap_port_num=6607

dpc_mode=MP

//将新初始化的节点注册仅待新增节点的 MP_RAFT 组中

SP_CREATE_DPC_INSTANCE(", 'MP4', 'MP', 6606, 5242, '192.168.1.68', 'STANDBY', 0, 'MP instance');

SP_CREATE_DPC_INSTANCE(", 'MP5', 'MP', 6607, 5243, '192.168.1.68', 'STANDBY', 0, 'MP instance');
```

第二步，修改配置。

1. 打开 MP4, MP5 节点 dm.ini 归档选项 ARCH_INI = 1。
2. 配置 dmarch.ini 文件。

MP4 的归档配置如下：

```
XMAL_HB_INTERVAL      = 5           #节点通信检测间隔

RAFT_HB_INTERVAL      = 150          #选举心跳间隔

RAFT_VOTE_INTERVAL    = 1500         #选举超时时间,三个库设置不同以尽快选出主库

RAFT_SELF_ID          = 4            #指定自身编号

RAFT_LEARNER          = 1            #指定自己是 Learner 节点

[ARCHIVE_RAFT1]

ARCH_TYPE              = RAFT         #RAFT 归档

ARCH_DEST              = MP1         #归档目标实例名

ARCH_DEST_ID          = 1            #指定归档目标节点编号
```

DM8 分布计算集群

```
[ARCHIVE_RAFT2]

ARCH_TYPE          = RAFT          #RAFT 归档

ARCH_DEST          = MP2          #归档目标实例名

ARCH_DEST_ID       = 2            #指定归档目标节点编号

[ARCHIVE_RAFT3]

ARCH_TYPE          = RAFT          #RAFT 归档

ARCH_DEST          = MP3          #归档目标实例名

ARCH_DEST_ID       = 3            #指定归档目标节点编号

[ARCHIVE_RAFT4]

ARCH_TYPE          = LEARNER       #RAFT 归档

ARCH_DEST          = MP5          #归档目标实例名

ARCH_DEST_ID       = 5            #指定归档目标节点编号

[ARCHIVE_LOCAL1]

ARCH_TYPE          = LOCAL         #本地归档类型

ARCH_DEST          = E:\RAFT_DB\ARCH\MP4 #本地归档文件路径

ARCH_FILE_SIZE     = 128           #本地单个归档文件最大值,单位 MB

ARCH_SPACE_LIMIT   = 0            #本地归档文件总大小,0 表示无限制
```

MP5 的归档配置如下:

```
XMAL_HB_INTERVAL   = 5            #节点通信检测间隔
```

DM8 分布计算集群

| | | |
|--------------------|-----------|------------------------|
| RAFT_HB_INTERVAL | = 150 | #选举心跳间隔 |
| RAFT_VOTE_INTERVAL | = 1500 | #选举超时时间,三个库设置不同以尽快选出主库 |
| RAFT_SELF_ID | = 5 | #指定自身编号 |
| RAFT_LEARNER | = 1 | #指定自己是 Learner 节点 |
| [ARCHIVE_RAFT1] | | |
| ARCH_TYPE | = RAFT | #RAFT 归档 |
| ARCH_DEST | = MP1 | #归档目标实例名 |
| ARCH_DEST_ID | = 1 | #指定归档目标节点编号 |
| [ARCHIVE_RAFT2] | | |
| ARCH_TYPE | = RAFT | #RAFT 归档 |
| ARCH_DEST | = MP2 | #归档目标实例名 |
| ARCH_DEST_ID | = 2 | #指定归档目标节点编号 |
| [ARCHIVE_RAFT3] | | |
| ARCH_TYPE | = RAFT | #RAFT 归档 |
| ARCH_DEST | = MP3 | #归档目标实例名 |
| ARCH_DEST_ID | = 3 | #指定归档目标节点编号 |
| [ARCHIVE_RAFT4] | | |
| ARCH_TYPE | = LEARNER | #RAFT 归档 |
| ARCH_DEST | = MP4 | #归档目标实例名 |
| ARCH_DEST_ID | = 4 | #指定归档目标节点编号 |

DM8 分布计算集群

```
[ARCHIVE_LOCAL1]

ARCH_TYPE          = LOCAL          #本地归档类型

ARCH_DEST          = E:\RAFT_DB\ARCH\MP5 #本地归档文件路径

ARCH_FILE_SIZE     = 128              #本地单个归档文件最大值,单位 MB

ARCH_SPACE_LIMIT   = 0                #本地归档文件总大小,0 表示无限制
```

3. 配置 MP4, MP5 的 mp.ini, 并将本集群所有节点的 mp.ini 文件内容修改到一致。

```
# mp_port 端口号必须与 MP、BP 和 SP 上的 ap_port_num 端口号不冲突

[MP1]

mp_host = 192.168.1.68

mp_port = 10622

[MP2]

mp_host = 192.168.1.68

mp_port = 10623

[MP3]

mp_host = 192.168.1.68

mp_port = 10624

[MP4]

mp_host = 192.168.1.68

mp_port = 10625

[MP5]

mp_host = 192.168.1.68

mp_port = 10626
```

第三步, 准备节点数据。

DM8 分布计算集群

1. 启动 DMAP，连接 MPI，联机备份主节点 MPI。

```
SQL>BACKUP DATABASE BACKUPSET 'E:\dpc\mpi\DAMENG\bak';
```

得到 MPI 的备份集 E:\dpc\mpi\DAMENG\bak_MPI。

2. 还原 MP4，MP5 节点。

//还原 MP4。BAK_MAGIC 值可通过 DMRMAN 工具的 SHOW 命令查看备份集得知。

```
RMAN>SHOW BACKUPSET 'E:\dpc\mpi\DAMENG\bak_MPI';
```

//BAK_MAGIC 为 1903999294。

```
RMAN>RESTORE DATABASE 'E:\dpc\mp4\DAMENG\dm.ini' FROM BACKUPSET
```

```
'E:\dpc\mpi\DAMENG\bak_MPI' USE BAK_MAGIC 1903999294;
```

```
RMAN>RECOVER DATABASE 'E:\dpc\mp4\DAMENG\dm.ini' FROM BACKUPSET
```

```
'E:\dpc\mpi\DAMENG\bak_MPI' USE BAK_MAGIC 1903999294 UNTIL END_LSN;
```

```
RMAN>RECOVER DATABASE 'E:\dpc\mp4\DAMENG\dm.ini' UPDATE DB_MAGIC;
```

//还原 MP5

```
RMAN>RESTORE DATABASE 'E:\dpc\mp5\DAMENG\dm.ini' FROM BACKUPSET
```

```
'E:\dpc\mpi\DAMENG\bak_MPI' USE BAK_MAGIC 1903999294;
```

```
RMAN>RECOVER DATABASE 'E:\dpc\mp5\DAMENG\dm.ini' FROM BACKUPSET
```

```
'E:\dpc\mpi\DAMENG\bak_MPI' USE BAK_MAGIC 1903999294 UNTIL END_LSN;
```

```
RMAN>RECOVER DATABASE 'E:\dpc\mp5\DAMENG\dm.ini' UPDATE DB_MAGIC;
```

第四步，启动节点，执行增加节点操作。

1. 启动 MP4，MP5。
2. 连接主库 MPI，执行增加节点流程。

//将 MP4，MP5 以 learner 节点的方式加入集群，从而在不影响集群使用条件下向新节点同步日志

```
SP_ADD_RAFT_LEARNER('MP4',192.168.1.68',10624,4);
```

```
SP_ADD_RAFT_LEARNER('MP5',192.168.1.68',10625,5);
```

DM8 分布计算集群

```
//将新节点加入集群

SP_ALTER_RAFT_NODE('MP1/MP2/MP3/MP4/MP5');
```

8.2.2.2.2 删除 MP

执行节点删除操作如下：

1. 连接 RAFT_1 中主节点 MP1，执行删除节点操作。

```
//执行

SP_ALTER_RAFT_NODE('MP1/MP2/MP3');

或

SP_DELETE_RAFT_NODE('MP4/MP5');
```

2. 连接 MP 删除 MP 实例。

```
//移除 MP4, MP5 实例

SP_DROP_DPC_INSTANCE('MP4');

SP_DROP_DPC_INSTANCE('MP5');
```

3. 修改所有节点的 mp.ini 文件。

```
# mp_port 端口号必须与 MP、BP 和 SP 上的 ap_port_num 端口号不冲突

[MP1]

mp_host = 192.168.1.68

mp_port = 10622

[MP2]

mp_host = 192.168.1.68

mp_port = 10623

[MP3]

mp_host = 192.168.1.68
```

```
mp_port = 10624
```

8.3 管理 DMDPC 表空间

DMDPC 支持表空间的功能，包括表空间的创建，建表时指定表空间，创建分区组时指定表空间，基于表空间实现的节点间数据迁移从而实现数据重分布等功能。

DMDPC 创建表空间语法和单机类似，不同的地方有两点：一是 DMDPC 增加了一个指定表空间所在 RAFT 组和 BP 组的<STORAGE 子句>；二是在 DMDPC 中增加了一个<HUGE 路径子句>，和<数据文件子句>搭配使用，可创建一个混合表空间，可以同时存储 HUGE 表和非 HUGE 表。

语法如下：

```
CREATE TABLESPACE <表空间名> <数据文件子句>[<数据页缓冲池子句>][<存储加密子句>][<HUGE 路径子句>][<STORAGE 子句>]
```

<STORAGE 子句> ::=

```
STORAGE (ON <RAFT 组名>)|
```

```
STORAGE (ON <BP 组名>)
```

<HUGE 路径子句> ::= WITH HUGE PATH <文件路径>

其他语法请参考《DM8_SQL 语言使用手册》中管理表空间章节

<HUGE 路径子句>和<数据文件子句>同时使用，可创建一个混合表空间。所谓的混合表空间就是一次创建两个同名的表空间：一个常规(非 HUGE)表空间和一个 HUGE 表空间。常规表空间位于<数据文件子句>指定的路径下，HUGE 表空间位于<HUGE 路径子句>指定的路径下。因为两个表空间同名，建表时指定表空间的用法又一样，因此对用户来说仿佛就是一个表空间，且既能存储 HUGE 表又能存储非 HUGE 表，因此又称为一个混合表空间。

<STORAGE 子句> RAFT 组名或 BP 组名必须是已存在的组名。<表空间名>、<RAFT 组名>、<BP 组名>三者不能同名。

DM8 分布计算集群

<RAFT 组名>和<BP 组名>需要分别单独指定。若只指定了 BP 组名，RAFT 组则是从 BP 组内的所有 RAFT 组中随机选定一个。若只指定了 RAFT，所以 BP 组不需要再选。如果二者均不指定，则从现有的 RAFT 组中随机挑选一个作为表空间的存储位置。

例 分别创建存储在 RAFT_1 的非 HUGE 普通表空间 TS_1 和存储在 RAFT_2 上的混合表空间 TS_2。

```
CREATE TABLESPACE TS_1 DATAFILE 'TS_1.DBF' SIZE 128 STORAGE( ON RAFT_1);  
  
CREATE TABLESPACE TS_2 DATAFILE 'TS_2.DBF' SIZE 128 WITH HUGE PATH 'e:\mpraft_data\hts'  
STORAGE( ON RAFT_2);
```

8.4 管理 DMDPC 表空间组

表空间组是一组位于相同 RAFT 或者不同 RAFT 上的表空间集合，用于用户建表或者指定用户默认的存储位置。表空间组中的表空间可动态扩展和收缩。且能同时包含普通表空间和混合表空间。

表空间组对象存储在系统表 SYSTSGROUPS 中，表空间组通过系统过程进行管理。详细的系统过程请参考 [6.2 配置方法](#)。

8.4.1 创建表空间组

定义：

```
SP_TS_GROUP_CREATE(  
  
    ts_group_name      varchar(128),  
  
    desc               varchar(256)  
  
)
```

功能说明：

创建一个表空间组。

参数说明:

ts_group_name: 表空间组名称;

desc: 描述信息。

举例说明:

创建一个表空间组 TSG_1。

```
SP_TS_GROUP_CREATE('TSG_1', 'tablespace group1');
```

8.4.2 删除表空间组

定义:

```
SP_TS_GROUP_DROP(  
  
    ts_group_name      varchar(128)  
  
)
```

功能说明:

删除一个表空间组。

参数说明:

ts_group_name: 表空间组名称。

举例说明:

删除表空间组 TSG_1。

```
SP_TS_GROUP_DROP('TSG_1');
```

8.4.3 向表空间组中添加表空间

定义:

```
SP_TS_GROUP_ADD_TS(  
  
    ts_group_name      varchar(128),
```

DM8 分布计算集群

```
ts_name          varchar(256)
)
)
```

功能说明:

向表空间组中添加一个表空间。表空间组内只能添加用户创建的表空间，不能添加系统表空间。表空间组内可同时添加普通表空间和带 HUGE 路径的表空间。实际使用时会根据建表的 HUGE 属性自动选择。

参数说明:

ts_group_name: 表空间组名称;

ts_name: 表空间名称。

举例说明:

向表空间组 TSG_1 中添加表空间 TS_1。

```
SP_TS_GROUP_ADD_TS('TSG_1','TS_1');
```

8.4.4 删除表空间组中的表空间

定义:

```
SP_TS_GROUP_REMOVE_TS(
    ts_group_name    varchar(128),
    ts_name          varchar(256)
)
)
```

功能说明:

删除表空间组中的一个表空间。

参数说明:

ts_group_name: 表空间组名称;

ts_name: 表空间名称。

举例说明:

删除表空间组 TSG_1 中的表空间 TS_1。

```
SP_TS_GROUP_REMOVE_TS('TSG_1', 'TS_1');
```

8.5 管理用户

用户可以指定和修改默认表空间/表空间组。

8.5.1 创建用户

语法格式

```
CREATE USER <用户名> IDENTIFIED <身份验证模式> [PASSWORD_POLICY <口令策略>][<锁定子句>][<存储加密密钥>][<空间限制子句>][<只读标志>][<资源限制子句>][<允许 IP 子句>][<禁止 IP 子句>][<允许时间子句>][<禁止时间子句>][<TABLESPACE 子句>]
<TABLESPACE 子句> ::= DEFAULT TABLESPACE <表空间名> |
                        DEFAULT TABLESPACE GROUP <表空间组名>
```

<TABLESPACE 子句>用于指定用户的默认表空间或表空间组。

<表空间名>默认表空间名。<表空间名>缺省或者设置为 NULL 表示未设置默认用户表空间。

<表空间组名>默认表空间组名。不能和<表空间名>重名。<表空间组名>缺省或者设置为 NULL 表示未设置用户默认表空间组。

例 创建一个指定默认表空间组的用户。

```
CREATE USER USER_1 IDENTIFIED BY 123456789 DEFAULT TABLESPACE GROUP TSG_1;
```

8.5.2 修改用户

语法格式

DM8 分布计算集群

```
ALTER USER <用户名> [[IDENTIFIED <身份验证模式>] [PASSWORD_POLICY <口令策略>] [<锁定子句>]
[<存储加密密钥>] [<空间限制子句>] [<只读标志>][<资源限制子句>][<允许 IP 子句>][<禁止 IP 子
句>][<允许时间子句>][<禁止时间子句>][<TABLESPACE 子句>]
<TABLESPACE 子句> ::=DEFAULT TABLESPACE <表空间名> |
DEFAULT TABLESPACE GROUP <表空间组名>
```

<TABLESPACE 子句>用于修改用户的默认表空间或表空间组。如果<表空间名>或<表空间组名>设置的是 NULL，表示清除之前设置的用户默认表空间或者表空间组。

例 修改用户的默认表空间组。

```
ALTER USER USER_1 DEFAULT TABLESPACE GROUP TSG_1;
```

8.6 管理 DMDPC 表

在 DMDPC 环境中数据存储在各个 BP，用户创建表之前必须先创建用户表空间，如果没有创建任何用户表空间则建表会报错。

有两种指定存储位置的方式：一，建表时指定表空间或表空间组，该方式既可以创建分区表又可以创建非分区表；二，基于分区组创建表，该方式先创建分区组，在分区组中指定表空间名，然后基于分区组创建分区表。方式二只能创建分区表。如果应用场景中存在多个分区方式相同且分区列之间有关联关系的分区表，推荐使用方式二。

8.6.1 建表时指定表空间或表空间组

DMDPC 创建表语法和单机类似。不同的是：指定 HASH 分区个数 <分区数> 时，可以使用关键字 DEFAULT 代替具体的分区数字，此时分区个数为建表语句涉及的 BP RAFT 个数和 DM.INI 参数 hp_tab_count_per_bp 的乘积，但是上限不能超过当前系统允许的最大子表个数。

在 DMDPC 中使用 [ON <表空间名>] 或 [ON <表空间组名>] 来指定表空间名。其他语法请参考《DM8_SQL 语言使用手册》中定义数据库基表章节。

语法如下：

.....

```
<STORAGE 子句> ::= STORAGE(<STORAGE 项> {,<STORAGE 项>})
```

```
<STORAGE 项> ::=
```

```
[INITIAL <初始簇数目>] |
```

```
[NEXT <下次分配簇数目>] |
```

```
[MINEXTENTS <最小保留簇数目>] |
```

```
[ON <表空间名>] |
```

DM8 分布计算集群

```

[ON <表空间组名>] |
[FILLFACTOR <填充比例>]|
[BRANCH <BRANCH 数>]|
[BRANCH (<BRANCH 数>, <NOBRANCH 数>)]|
[NOBRANCH]|
[<CLUSTERBTR>]|
[WITH COUNTER]|
[WITHOUT COUNTER] |
[USING LONG ROW]
  
```

.....

[ON <表空间名>]或[ON <表空间组名>]用来指定存储的表空间名或表空间组名。非 HUGE 表既可以存储在非 HUGE 普通表空间也可以存储在混合表空间中;但 HUGE 表只能使用混合表空间进行存储。对包含引用约束的引用表和被引用表,要求二者的对应分区存储在同一个表空间中。

建表时系统选择存储表空间的规则为:

- 1.允许显示指定表空间/表空间组的方式建表;
- 2.如果未显示指定存储选项,则先使用用户默认的表空间/表空间组;
- 3.如果用户未指定默认表空间/组,再从系统的用户表空间中挑选;
- 4.如果未创建任何用户表空间会报错,提示用户需要创建表空间;
- 5.创建水平分区表时,如果参数 DPC_TABLESPACE_BALANCE=1,且未显式指定存储选项,或者仅指定了分区主表存储于某表空间组,那么系统会计算将要创建的单层的叶子子表数 N_1 和可用表空间数 N_2 。当 $N_1 < N_2$ 时,系统会从众多可用表空间中随机选择起始表空间,其目的是确保表空间的存储均匀。哈希分区使用了 Partitions default 的情况不在此列;
6. 向间隔分区表中插入超出了指定分区范围的数据,将会自动新增分区,新增的分区

DM8 分布计算集群

只会落在建表时已有的表空间上，不会落在建表后新增的表空间上。因为在建表时，每个子分区（包括模板）的存储表空间已经固化，后面新增分区时只会按照上述固化的表空间遍历挑选。

例 1 将表 test1 的 P1 分区存储在表空间 TS_1 上，P2 分区存储在表空间 TS_2 上，P3 分区存储在表空间 TS_3 上。

```
create table test1(c1 int, c2 int) PARTITION BY RANGE(c1) (  
PARTITION p1 VALUES LESS THAN (10) storage (on TS_1),  
PARTITION p2 VALUES LESS THAN (20) storage (on TS_2),  
PARTITION p3 VALUES EQU OR LESS THAN(MAXVALUE) storage (on TS_3));
```

例 2 将表 test2 中的 P1 分区存储在表空间 TS_1 上，P2 分区存储在表空间组 TSG_1 上。P3 未明确指定表空间或表空间组，缺省存储在表空间组 TSG_2 上。

```
create table test2(c1 int, c2 int) PARTITION BY RANGE(c1) (  
PARTITION p1 VALUES LESS THAN (10) storage (on TS_1),  
PARTITION p2 VALUES LESS THAN (20) storage (on TSG_1),  
PARTITION p3 VALUES EQU OR LESS THAN(MAXVALUE)  
) storage (on TSG_2);
```

8.6.2 基于分区组创建分区表

8.6.2.1 分区组简介

分区组（Partition Group, PG）是 DM8 为了简化水平分区表建表操作而提供的一项新功能。分区组是将水平分区表创建时使用的分区规则提取成一个独立的逻辑概念。使用分区组创建水平分区表的步骤是先创建分区组，然后再基于分区组创建分区表。如果创建水

DM8 分布计算集群

平分区表的语句中未使用分区组，那么还是按照常规的方式处理，将分区规则直接写在建表语句中。使用分区组建表既可以简化建表操作，又可以提升事务处理的性能。

不同分区表可以使用同一个分区组，每个分区表只允许属于一个分区组。

使用分区组后，隶属相同分区组的对应分区始终在同一表空间中，基于分区列的表连接处理可以减少跨机事务和连接时数据交换的开销。因此，对于需要进行连接操作的两张表，推荐使用相同的分区组 PG。

图 8.1 中预先定义了分区组 PG1、PG2、PG3 和 PG4，其中 PG1 有三个分区 P1、P2、P3 分别存储于表空间 TS1、TS2 和 TS3。PG2 定义了三个分区 P1、P2、P3 全部存储于 TS1。PG3 有二个分区 P1、P2 分别存储于 TS2 和 TS3 中。PG4 只有一个分区 P1 存储于 TS2 中。

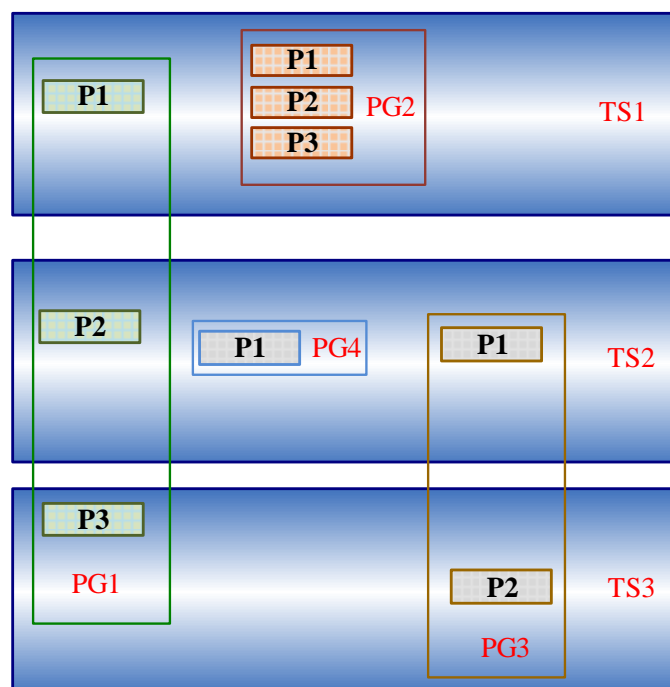


图 8.1 分区组

同一分区组可以跨多个表空间。例如：PG1 和 PG3 都横跨了几个表空间。

隶属于同一分区组的多个表，分区组中定义的分区可以保证这些表的相同分区会分布在相同的表空间上。例如：若 T1 和 T2 表都使用了 PG1 分区组，则 PG1 的分区可以保证 T1 和 T2 表的相同分区会分布在相同的表空间上。若 T3 和 T4 表都使用了 PG2 分区组，则它们所

对应的分区分布情况也相同。

8.6.2.2 分区组定义语句

目前，DMDPC 系统仅支持一级水平分区和二级水平分区。

语法如下：

```

CREATE PARTITION GROUP <组名> <PARTITION 子句> [<DMDPC_存储位置子句>]

<PARTITION 子句>::= PARTITION BY <PARTITION 项>

<PARTITION 项>::=

    RANGE (<分区列类型>{,<分区列类型>}) [INTERVAL <间隔表达式>] [<SUBPARTITION 子句>{,<
SUBPARTITION 子句>}](<RANGE 分区项> {,<RANGE 分区项>}) |

    HASH (<分区列类型>{,<分区列类型>}) [<SUBPARTITION 子句>{,< SUBPARTITION 子
句>}]PARTITIONS <分区数> [<STORAGE HASH 子句>]|

    HASH(<分区列类型>{,<分区列类型>}) [<SUBPARTITION 子句>{,< SUBPARTITION 子句>}]

(<HASH 分区项> {,<HASH 分区项>})|

    LIST(<分区列类型>)[<SUBPARTITION 子句>{,< SUBPARTITION 子句>}](<LIST 分区项> {,<LIST 分
区项>})

<RANGE 分区项>::= PARTITION <分区名> VALUES [EQU OR] LESS THAN (< <常量表达式>|<日期函数表
达式>|MAXVALUE >{,< <常量表达式>|<日期函数表达式>|MAXVALUE >}) [<DMDPC_STORAGE 子
句>][<子分区描述项>]

<日期函数表达式>::= <to_date 函数表达式> | <to_datetime 函数表达式> | <to_timestamp 函数表
达式>

<HASH 分区项>::= PARTITION <分区名> [<DMDPC_STORAGE 子句>][<子分区描述项>]

<LIST 分区项>::= PARTITION <分区名> VALUES (DEFAULT|<表达式>{,<表达式>})

 [<DMDPC_STORAGE 子句>][<子分区描述项>]
  
```

DM8 分布计算集群

<子分区描述项>::=

(<RANGE 子分区描述项> {, <RANGE 子分区描述项> }) |

(<HASH 子分区描述项> {, <HASH 子分区描述项> }) |

SUBPARTITIONS <分区数> [<DMDPC_存储位置子句>] |

(<LIST 子分区描述项> {, <LIST 子分区描述项> })

<RANGE 子分区描述项>::= <RANGE 子分区项> [<子分区描述项>]

<HASH 子分区描述项>::= <HASH 子分区项> [<子分区描述项>]

<LIST 子分区描述项>::= <LIST 子分区项> [<子分区描述项>]

<RANGE 子分区项>::=

SUBPARTITION <分区名> VALUES [EQU OR] LESS THAN (<常量表达式> | <日期函数表达式> | MAXVALUE) {, <常量表达式> | <日期函数表达式> | MAXVALUE }) [<DMDPC_STORAGE 子句>]

<HASH 子分区项>::= SUBPARTITION <分区名> [<DMDPC_STORAGE 子句>]

<LIST 子分区项>::= SUBPARTITION <分区名> VALUES (DEFAULT | <<表达式>, { <表达式> })) [<DMDPC_STORAGE 子句>]

<SUBPARTITION 子句> ::= <RANGE 子分区模板项> | <HASH 子分区模板项> | <LIST 子分区模板项>

<RANGE 子分区模板项> ::= SUBPARTITION BY RANGE (<列名> {, <列名> }) [SUBPARTITION TEMPLATE (<RANGE 子分区项> {, <RANGE 子分区项> })]

<HASH 子分区模板项> ::=

SUBPARTITION BY HASH (<列名> {, <列名> }) SUBPARTITION TEMPLATE SUBPARTITIONS <分区数> [<STORAGE HASH 子句>] |

SUBPARTITION BY HASH (<列名> {, <列名> }) SUBPARTITION TEMPLATE (<HASH 子分区项> {, <HASH 子分区项> })

<LIST 子分区模板项> ::= SUBPARTITION BY LIST (<列名> {, <列名> }) [SUBPARTITION TEMPLATE (<LIST 子分区项> {, <LIST 子分区项> })]

DM8 分布计算集群

`<STORAGE HASH 子句> ::= STORE IN (<DMDPC_存储位置> {, <DMDPC_存储位置> })`
`<DMDPC_ STORAGE 子句> ::= STORAGE (ON <DMDPC_存储位置>)`
`<DMDPC_存储位置子句> ::= STORAGE (<DMDPC_存储位置链表> | <DMDPC_存储选项链表>)`
`<DMDPC_存储位置链表> ::= ON <DMDPC_存储位置> {, <DMDPC_存储位置> }`
`<DMDPC_存储选项链表> ::= <DMDPC_存储选项> {, <DMDPC_存储选项> }`
`<DMDPC_存储选项> ::= <DMDPC_HASHMAP 选项> | <DMDPC_存储位置链表选项>`
`<DMDPC_HASHMAP 选项> ::= HASHPARTMAP (<哈希分区表定位方式>)`
`<哈希分区表定位方式> ::= 0||2`
`<DMDPC_存储位置链表选项> ::= ON (<DMDPC_存储位置> {, <DMDPC_存储位置> })`
`<DMDPC_存储位置> ::= <表空间名> | <表空间组名>`

其他语法请参见《DM8 SQL 语言使用手册》3.6.1.4 节说明

`<DMDPC_ STORAGE 子句>`用于指定子表的存储位置。

`<DMDPC_存储位置子句>` 用于指定主表或子表的存储位置。分区组创建时，如果未指定子表的存储位置，子表会从父层主表处继承；如果主表、子表都未指定，系统会从集群可用的 BP 实例中随机挑选。如果`<DMDPC_存储位置子句>`没有指定分区组中分区的存储表空间时，系统会在保证存储负载均衡的前提下优先考虑混合表空间。即当已有的混合表空间所在的 BP 不足以覆盖集群中大部分的 BP 实例时，系统会添加未被覆盖的 BP 实例中的普通表空间。除非确信分区组不会用于创建 HUGE 分区表，否则建议用户在分区组定义脚本中显式指定混合表空间的存储方式。

`<STORAGE HASH 子句>`中指定的`<DMDPC_存储位置>`数量少于分区个数时，可以按照从前到后的顺序，循环使用现有的`<DMDPC_存储位置>`。

`<分区列类型>`必须满足已有的分区类型约束，详见《DM8 SQL 语言使用手册》中对“分区列类型”的描述。

`<哈希分区表定位方式>`用于指定哈希分区表的数据定位方式。取值 0 表示采用 DM 原

DM8 分布计算集群

有的数据定位方式；取值 1 表示采用新数据定位方式，此时可以添加分区，默认取值 1；取值 2 为兼顾数据分布均衡和允许添加分区的数据定位方式。当分区表根表为 RANGE 或 LIST 分区表时也可以指定<HASHPARTMAP 定位方式>，用来影响下层 HASH 分区子表。

例 1 创建一个分区列类型是整型的范围分区组 PG1，指定每个分区的存储表空间。

```
create partition group PG1 partition by range(int)
(
  partition p1 values less than(10) storage (on TS_2),
  partition p2 values less than(40) storage (on TS_1),
  partition p3 values less than(80) storage (on TS_2));
```

例 2 创建分区列类型是 varchar(100)的 HASH 分区组，32 个子表存储位置均匀分布在 TS_1, TS_2 上。

```
create partition group PG2 partition by hash(varchar(100)) partitions 32 storage (on
TS_1,TS_2);
```

例 3 使用<STORAGE HASH 子句>创建一个 HASH 分区组，3 个子表存储位置分别分布在 TS_1、TS_2、TS_3 上。

```
create partition group PG3 partition by hash(varchar(1)) partitions 3 store in (TS_1,TS_2,TS_3);
```

例 4 创建一个二级分区组，两级分区方式都为范围分区。

```
create partition group PG4
partition by range(int)
subpartition by range(int)
(
  partition p1 values less than(0) storage(on TS_1) (subpartition p11 values less than(1),
  subpartition p12 values less than(10)),
```

DM8 分布计算集群

```
partition p2 values less than(10) (subpartition p21 values less than(0) storage (on TS_2),
subpartition p22 values less than(10)) ,
partition p3 values less than(100) (subpartition p31 values less than(50) , subpartition p32
values less than(60))
) storage (on TS_1);
```

例 5 创建一个二级分区组，一级分区为 HASH，二级分区为 RANGE。所有叶子表的存储位置将从 TS_1, TS_2 中挑选。

```
create partition group PG5
partition by hash(int)
subpartition by range(int)
(
partition "part_1" (subpartition p11 values less than(1), subpartition p12 values less
than(10)) ,
partition "part_2" (subpartition p21 values less than(0) , subpartition p22 values less
than(10)) ,
partition "part_3" (subpartition p31 values less than(50) , subpartition p32 values less
than(60))
) storage (on TS_1,TS_2);
```

使用分区组 PG5 创建一个分区表 T_XQ64。

```
create table T_XQ64 (c int, c2 int, c3 int) using partition group PG5 by (c) SUBPARTITION by
(c3);
```

查看每个子表的存储位置。

```
SQL>select(select x.name from sysobjects x where x.id = y.pid) tname, groupid as ts_id,
(select name from dpc_tablespace where ts_id = groupid) as ts_name
```

DM8 分布计算集群

```
from sysobjects y, sysindexes z where z.id = y.id and pid in (select id from sysobjects where
name like 'T_XQ64%' and subtype$='UTAB' and name not like '%AUX') and subtype$='INDEX'
and schid = (select id from sysobjects where type$='SCH' and name = 'SYSDBA') order by
tname asc;
```

//查询结果如下:

| 行号 | TNAME | TS_ID | TS_NAME |
|----|-------------------|-------|---------|
| 1 | T_XQ64 | 7 | TS_1 |
| 2 | T_XQ64_part_1 | 8 | TS_2 |
| 3 | T_XQ64_part_1_P11 | 7 | TS_1 |
| 4 | T_XQ64_part_1_P12 | 8 | TS_2 |
| 5 | T_XQ64_part_2 | 7 | TS_1 |
| 6 | T_XQ64_part_2_P21 | 7 | TS_1 |
| 7 | T_XQ64_part_2_P22 | 8 | TS_2 |
| 8 | T_XQ64_part_3 | 8 | TS_2 |
| 9 | T_XQ64_part_3_P31 | 7 | TS_1 |
| 10 | T_XQ64_part_3_P32 | 8 | TS_2 |

8.6.2.3 查看分区组

查看分区组定义有两种方式：一是通过 PARTGROUPDEF() 函数查看；二是通过 SYSOBJECTS 和 SYSTEXTS 相互配合查看。

一 通过 PARTGROUPDEF() 函数查看。

PARTITION GROUP 作为一个模式对象，可以通过 PARTGROUPDEF 查看定义信息。

```
select PARTGROUPDEF('SYSDBA','PG2');
```

DM8 分布计算集群

```

行号      PARTGROUPDEF('SYSDBA','PG2')
-----
1          CREATE PARTITION GROUP PG2 PARTITION BY  HASH(VARCHAR (100)) PARTITIONS 32
STORAGE (ON TS_1,TS_2);
    
```

二 通过 SYSOBJECTS 和 SYSTEXTS 相互配合查看。

首先，通过查询 SYSOBJECTS 获取当前系统中的所有 PARTITION GROUP 信息。得到 PGI 的 ID 为 3283。

```

select * from sysobjects where      subtype$='PGRP';

SQL> select * from sysobjects where      subtype$='PGRP';

//查询结果如下:

行号      NAME      ID          SCHID      TYPE$  SUBTYPE$ PID          VERSION
CRTDATE  INFO1      INFO2      INFO3      INFO4      INFO5
INFO6      INFO7      INFO8      VALID
-----
1          PG1      3283      150994945  SCHOBJ PGRP      NULL          0
2021-05-07 13:54:41.000000
          0          NULL      NULL      NULL      NULL          NULL
NULL      NULL      NULL      NULL      Y
2          PG2      3284      150994945  SCHOBJ PGRP      NULL          0
2021-05-07 13:54:49.000000
          0          NULL      NULL      NULL      NULL          NULL
NULL      NULL      NULL      NULL      Y
    
```

然后，查询 SYSTEXTS 系统表，得到 PGI 分区组定义。

DM8 分布计算集群

```
SQL> select * from systexts where id =3283;
```

| 行号 | ID | SEQNO | TXT |
|----|------|-------|--|
| 1 | 3283 | 0 | CREATE PARTITION GROUP PGI PARTITION BY RANGE(INT)(PARTITION P1 VALUES LESS THAN(10) STORAGE(ON TS_2), PARTITION P2 VALUES LESS THAN(40) STORAGE(ON TS_1), PARTITION P3 VALUES LESS THAN(80) STORAGE(ON TS_2)) |

用户在创建表对象时应该优先考虑借助分区组，选择最佳分区列和分区方式，将分区方式定义成分区组，这会为多表查询和事务处理带来性能提升。

8.6.2.4 使用分区组创建分区表

使用分区组创建表时，分区列的个数和数据类型必须和分区组中定义保持完全一致。

语法如下：

```
CREATE TABLE <表名定义> ( <列定义> {, <列定义>} [ <表级约束定义> {, <表级约束定义>} ] ) <分区组子句> [ <STORAGE 子句> ] [ <ROW MOVEMENT 子句> ]
```

<表名定义> ::= [<模式名> .] <表名>

<分区组子句> ::= USING PARTITION GROUP <分区组名> BY (<列名> {, <列名>}) [SUBPARTITION BY (<列名> {, <列名>})]

<列定义>、<表级约束定义>和<STORAGE 子句> ::= 参见《DM8 SQL 语言使用手册》3.6.1.1 节说明

<ROW MOVEMENT 子句> ::= 参见《DM8 SQL 语言使用手册》3.6.1.4 节说明

例 1 基于分区组 PGI 创建范围分区表。

```
create table DPC_T1(c int) using partition group PGI by(c);
```

例 2 因为数据类型没有完全一致会在创建时报错。

```
create table DPC_T1_ERR(c bigint) using partition group PGI by(c);
```

DM8 分布计算集群

//查询结果报错:

```
[-3804]:列[C]类型与分区组要求的不一致
```

例 3 基于分区组 PG2 创建哈希分区。

```
create table DPC_T2(c varchar(100)) using partition group PG2 by(c);
```

例 4 基于分区组 PG5 创建二级分区。

```
create table DPC_T3 (c int, c2 int, c3 int) using partition group PG5 by (c) SUBPARTITION by (c3);
```

例 5 使用分区 PG6 创建分区表，同时指定表空间。

```
create table DPC_T4(c varchar(1)) using partition group PG6 by(c) STORAGE (on ts_01);
```

8.6.2.5 删除分区组

语法如下:

```
DROP PARTITION GROUP <分区组名> [FORCE]
```

分区组被使用后无法直接删除，此时可指定 FORCE 选项强制删除分区组，基于该分区组创建的表对象不会被删除，但是表的 DDL 属性会发生变化，不再依附于分区组。

例 PGI 被表 DPC_T1 依赖，直接删除时失败。

```
drop partition group PGI;
```

//查询结果报错:

```
[-3805]:分区组被依赖，不能被删除.
```

查看表 DPC_T1 定义，发现 PGI 被表 DPC_T1 依赖，所以无法删除。

```
select tabledef('SYSDBA','DPC_T1');
```

//查询结果如下:

```
行号      TABLEDEF('SYSDBA','DPC_T1')
```

DM8 分布计算集群

```
1 CREATE TABLE "SYSDBA"."DPC_T1"
("C" INT)
```

```
USING PARTITION GROUP "SYSDBA"."PG1" BY ("C");
```

强制删除分区组 PG1 后，再次查看表定义，表 DPC_T1 不再依赖 PG1。

```
SQL> drop partition group PG1 force;
```

```
SQL> select tabledef('SYSDBA','DPC_T1');
```

//查询结果如下:

```
行号      TABLEDEF('SYSDBA','DPC_T1')
```

```
1 CREATE TABLE "SYSDBA"."DPC_T1"
```

```
(
```

```
"C" INT)
```

```
PARTITION BY RANGE("C")
```

```
(
```

```
PARTITION "P1" VALUES LESS THAN(10) STORAGE(ON "TS_2", CLUSTERBTR) ,
```

```
PARTITION "P2" VALUES LESS THAN(40) STORAGE(ON "TS_1", CLUSTERBTR) ,
```

```
PARTITION "P3" VALUES LESS THAN(80) STORAGE(ON "TS_2", CLUSTERBTR)
```

```
) STORAGE(ON "TS_1", CLUSTERBTR) ;
```

8.6.2.6 使用分区组的好处

分区组是一个逻辑概念，将相同的分区和分布选项定义为一个模板，然后基于这个模板创建分区表。它的引入能带来如下好处：

- 1.简化用户创建水平分区表的操作。

DM8 分布计算集群

假设在一个应用中有多张表的分区方式一样，如果没有分区组，那么用户必须为每一张表反复指定相同的分区方式和存储表空间。

例 下面用为使用分区组和使用了分区组的两种情况进行对比，来说明分区组的更加简洁的操作性。

- 未使用分区组的建表情况

```
create table M5_A(c1 int not null, c2 char(1000)) partition by range(c1)
(
partition p1 values less than(10) storage (on TS_1),
partition p2 values less than(40) storage (on TS_2),
partition p3 values less than(80) storage (on TS_3));

create table M5_B(c1 int not null, c2 char(1000)) partition by range(c1)
(
partition p1 values less than(10) storage (on TS_1),
partition p2 values less than(40) storage (on TS_2),
partition p3 values less than(80) storage (on TS_3));
```

- 使用分区组的建表情况

但是创建分区组 PGI 后，创建分区表的时候，只要指定分区组即可。

```
create partition group PGI partition by range(int)
(
partition p1 values less than(10) storage (on TS_1),
partition p2 values less than(40) storage (on TS_2),
partition p3 values less than(80) storage (on TS_3));
```

使用分区组 PGI 进行建表。大大简化了创建分区表的方式。

DM8 分布计算集群

```
create table M5_A(c1 int not null, c2 char(1000)) using partition group PGI by(c1);  
create table M5_B(c1 int not null, c2 char(1000)) using partition group PGI by(c1);
```

2.确保相同分区方式两张表的对应分区始终在同一 BP，减少跨机事务和连接时数据交换开销，提升事务处理的性能。

连接中一种常见的优化 Partition Wise Join（简称 PWJ），即当连接两侧不需要任何数据交换操作而可以直接在每个工作线程里进行连接。PWJ 优化的使用前提是两边分区方式一致且对应分区的 BP 位置也一致。借助于分区组，优化器可以直接判断出两张表连接时是否适用于 PWJ，否则优化器需要在比较完两张表的分区方式后，遍历表的每一个分区，逐一判断是否存储的 BP 一致。

8.7 数据迁移

DMDPC 进行节点扩容和缩容（增删节点）之后，为了能够充分利用系统内的节点资源，用户需要手动执行数据迁移，将数据在节点间进行重新均衡分布，以提高系统的资源利用率和整体性能。

DMDPC 提供库迁移、表迁移和表空间迁移三种方案。推荐用户使用表空间迁移。原因如下：

库级迁移是先将单个节点库停机，再将节点上的库拷贝到另外的节点上。库级迁移缺点是需要停机，对系统业务影响比较大。

表级迁移是指将单个表或者分区表的单个分区从一个表空间迁移另一个表空间。表级迁移缺点是粒度比较小，当存在大量表需要迁移时比较繁琐，且在表迁移期间不支持对表的读写操作，影响用户业务。

表空间迁移是将整个表空间从一个 RAFT 组迁移到另外一个 RAFT 组。表空间迁移分为联机 and 脱机两种方式。相比于库迁移和表迁移，表空间联机迁移效率更高，对系统正常运行影响更小。因此，表空间联机迁移是 DMDPC 系统用于实现数据均衡的最常用、最重要的技

术手段。

8.7.1 表空间迁移

在 DM 数据库中，表空间由一个或者多个数据文件组成。DM 数据库中的所有对象在逻辑上都存放在表空间中，而物理上都存储在所属表空间的数据文件中。

DMDPC 支持在 BP 上创建用户表空间，BP 和表空间的关系为一对多的关系，即一个 BP 上可以创建多个表空间，但一个表空间只能创建一个 BP 上。一个分区表的多个子表可以指定分布在不同的表空间上，从而将数据分布到多个 BP 上进行均衡存储。传统的单表只能指定一个表空间存储，因此无法分布到多个 BP 上。在一个 DMDPC 系统中，每个 BP 节点上必须至少创建一个表空间才会被用来存储用户数据，因此 BP 节点扩容后需要在其上创建表空间。

节点扩容或缩容后，整个系统中原有的数据均衡被打破。此时，我们只需要将分区表的部分分区所属的表空间迁移到新增 BP 节点上，将分区数据迁移到新增 BP 节点的目的，从而使系统数据再次分布均衡。

按照表空间为最小单位进行数据迁移的方法，和传统意义上的按照表对象的分布方式，根据表中每一行数据重新计算目标节点进行重分布的方式有着本质上的区别。表空间迁移可以同时多个存储在表空间上的表进行均衡。

表空间的迁移分为联机 and 脱机两种方式。联机方式不用停机即可操作，推荐用户使用联机方式迁移。

DMDPC 环境支持表空间迁移，但不允许将表空间向存在故障节点的 RAFT 组迁移。

8.7.1.1 联机方式

表空间联机迁移为在线进行，不需要停机。利用修改表空间 SQL，一键式拷贝表空间物理文件，且迁移期间允许用户读写操作，并将期间写操作新产生的日志不断进行重演达到数

DM8 分布计算集群

据同步，平滑实现表空间的跨 BP 节点转移，达到数据的均衡分布。

利用修改表空间语法实现联机表空间移动。目前 DMDPC 仅支持部分表空间修改语法。

语法如下：

```
ALTER TABLESPACE <表空间名> [ONLINE | OFFLINE|<增加数据文件子句>|<修改文件大小子句>|<修改文件自动扩展子句>|<表空间移动子句>]
```

```
<表空间移动子句> ::= MOVE TO <RAFT 组名> [[NOT] READ ONLY]
```

其他语法请参考《DM8_SQL 语言使用手册》中管理表空间章节

ONLINE | OFFLINE 表示表空间的状态。ONLINE 为联机状态，ONLINE 时才允许用户访问该表空间中的数据；OFFLINE 为脱机状态，OFFLINE 时不允许访问该表空间中的数据。副本架构的 DMDPC 不支持 OFFLINE，非副本架构支持 OFFLINE。

READ ONLY 只读和 NOT READ ONLY 读写方式区别：只读方式在表空间迁移的过程中，用户只能对表空间包含的表进行读操作，不能进行写操作。读写方式在迁移的过程中，用户大部分时间，比如在文件拷贝的过程中仍然能对表空间进行写操作，只在最后切换节点的短时间内写操作会被阻塞。读写方式对迁移过程中新修改的数据，通过重演日志的方式完成修改，这样对系统的影响更小，对用户更友好。如果不指定读写的属性，默认为只读方式。

例 1 使用 READ ONLY 只读方式，将表空间 TS_01 迁移到 RAFT_2 上。

```
alter tablespace ts_01 move to raft_2;  
alter tablespace ts_01 move to raft_2 read only;
```

例 2 使用 NOT READ ONLY 读写方式，将表空间 TS_01 迁移到 RAFT_2 上。

```
alter tablespace ts_01 move to raft_2 not read only;
```

8.7.1.2 脱机方式

所有节点正常退出的情况下，可以直接手动拷贝表空间文件到目标 BP 上，然后通过辅助步骤修改字典、控制文件以及更新系统表。

DM8 分布计算集群

步骤如下:

1. 用户确保需要移动的表空间所在 BP 是正常退出的。
2. 手动拷贝源表空间所有文件到目标 RAFT 的库目录中。
3. 利用 dmctlcvt 工具, 转换源库 dm.ctl->dm.txt。首先将 dm.txt 中的表空间项 (TS_ID、TS_NAME) 另存在他处备用; 其次将 TS_ID、TS_NAME 从 dm.txt 中删除后保存; 再次将删除后的 dm.txt 转换为新的 dm.ctl, 并替换掉源库 dm.ctl。
4. 利用 dmctlcvt 工具, 转换目标库 dm.ctl->dm.txt 得到对应表空间的文本, 添加步骤 3 中另存的表空间项 (TS_ID、TS_NAME), 再将 dm.txt 转换为 dm.ctl 并替换掉目标库 dm.ctl。
5. 单独启动 MP, 直连 MP 执行 SP_DPC_MOVE_TS_OFFLINE(TS_NAME, TO_RAFT_NAME)。
6. 比较目标库和源库的联机日志 CUR LSN 大小, 确保目标库联机日志 CUR LSN 更大。如果目标库 CUR LSN 大则跳过此步骤。否则将目标库的 CUR LSN 修改为源库 CUR LSN+1。
7. 启动所有节点, 验证迁移后数据是否正确。

例 表空间 TS1 从 RAFT_1 脱机方式迁移到 RAFT_2 的全过程。假设 RAFT_1 存在 BP1_A 节点上, RAFT_2 存在 BP2_A 节点上。

1. 确保需要移动的表空间所在 BP 是正常退出的。
2. 手动拷贝源库 RAFT_1 中表空间 TS1 的所有数据文件到目标库 RAFT_2 的库目录中。
3. 使用 dmctlcvt 工具转换 dm.ctl 为文本文件 dm.txt。

```
dmctlcvt.exe type=1 src=D:\data\dmdpc_dem_new\dmdpc\BP1_A\DAMENG\dm.ctl  
dest=D:\data\dmdpc_dem_new\dmdpc\BP1_A\DAMENG\dm.txt
```

首先, 将 dm.txt 中的表空间项 (TS_ID、TS_NAME 等内容) (下面内容) 另存在它处备用。其次, 将 TS_ID、TS_NAME 从 dm.txt 中删除后保存。另外需要记录一下

DM8 分布计算集群

next_ts_id=16, 如果目标库next_ts_id小于16, 则修改目标库中的next_ts_id值为16。

```
#=====
# table space name
ts_name=TS1
# table space ID
ts_id=7
# table space status
ts_state=0
# table space cache
ts_cache=NORMAL
# DSC node number
ts_nth=0
# DSC optimized node number
ts_opt_node=0
# table space create time
ts_create_time=DATETIME '2022-2-22 14:26:36'
# table space modify time
ts_modify_time=DATETIME '2022-2-22 14:26:36'
# table space encrypt flag
ts_encrypt_flag=0
# table space copy num
ts_copy_num=0
# table space region size flag
ts_size_flag=0
```

DM8 分布计算集群

```
# table space region huge size flag

ts_huge_size_flag=0

#-----

# file path

fil_path=D:\data\dmdpc_dem_new\dmdpc\BPI_A\DAMENG\ts1.dbf

# mirror path

mirror_path=

# file id

fil_id=0

# whether the file is auto extend

autoextend=1

# file create time

fil_create_time=DATETIME '2022-2-22 14:26:36'

# file modify time

fil_modify_time=DATETIME '2022-2-22 14:26:36'

# the max size of file

fil_max_size=0

# next size of file

fil_next_size=0
```

再次，将删除后的 dm.txt 转换为新的 dm.ctl，并替换掉源库 dm.ctl。

```
dmctlcv.exe type=2 src=D:\data\dmdpc_dem_new\dmdpc\BPI_A\DAMENG\dm.txt
dest=D:\data\dmdpc_dem_new\dmdpc\BPI_A\DAMENG\dm.ctl
```

DM8 分布计算集群

4. 将目标库 dm.ctl 转换为文本文件 dm.txt。

```
dmctlcv.exe type=1 src=D:\data\dmdpc_dem_new\dmdpc\BP2_A\DAMENG\dm.ctl
dest=D:\data\dmdpc_dem_new\dmdpc\BP2_A\DAMENG\dm.txt
```

首先，在目标库的 dm.txt 中添加步骤 3 中另存的表空间项 (TS_ID、TS_NAME)，注意需要修改 fil_path 路径为目标库路径，另外根据需要修改 next_ts_id 的值。

```
#=====
# table space name
ts_name=TS1
# table space ID
ts_id=7
# table space status
ts_state=0
# table space cache
ts_cache=NORMAL
# DSC node number
ts_nth=0
# DSC optimized node number
ts_opt_node=0
# table space create time
ts_create_time=DATETIME '2022-2-22 14:26:36'
# table space modify time
ts_modify_time=DATETIME '2022-2-22 14:26:36'
# table space encrypt flag
ts_encrypt_flag=0
```

DM8 分布计算集群

```
# table space copy num
ts_copy_num=0

# table space region size flag
ts_size_flag=0

# table space region huge size flag
ts_huge_size_flag=0

#-----

# file path
fil_path=D:\data\dmdpc_dem_new\dmdpc\BP2_A\DAMENG\ts1.dbf

# mirror path
mirror_path=

# file id
fil_id=0

# whether the file is auto extend
autoextend=1

# file create time
fil_create_time=DATETIME '2022-2-22 14:26:36'

# file modify time
fil_modify_time=DATETIME '2022-2-22 14:26:36'

# the max size of file
fil_max_size=0

# next size of file
```

DM8 分布计算集群

```
fil_next_size=0
```

其次，再将 dm.txt 转换为 dm.ctl 并替换掉目标库 dm.ctl。

```
dmctlcv.exe type=2 src=D:\data\dmdpc_dem_new\dmdpc\BP2_A\DAMENG\dm.txt
dest=D:\data\dmdpc_dem_new\dmdpc\BP2_A\DAMENG\dm.ctl
```

5. 单独启动 MP，直连 MP 执行 SP_DPC_MOVE_TS_OFFLINE 过程。

```
SP_DPC_MOVE_TS_OFFLINE('TSI',RAFT_2');
```

6. 比较目标库和源库的联机日志 CUR LSN 大小，确保目标库联机日志 CUR LSN 更大。

首先，用 dmmdf 查看源库的联机日志中的 CUR LSN（取 CLSN 和 C_LSN 中的最大值），此处源库为 BP1_A。

```
dmmdf TYPE=2 FILE=D:\data\dmdpc_dem_new\dmdpc\BP1_A\DAMENG\DAMENG01.log
```

记录下该值，例如为 621563。

用 dmmdf 查看目标库的联机日志中的 CUR LSN（取 CLSN 和 C_LSN 中的最大值），此处为 BP2。

```
dmmdf TYPE=2 FILE=D:\data\dmdpc_dem_new\dmdpc\BP2_A\DAMENG\DAMENG01.log
```

记录下该值，例如为 620000。可见，620000 小于 621563，因此需要将目标库的 CUR LSN 修改为大于源库的 CUR LSN。

温馨提示：如果目标库 BP2_A 是新初始化的库且未启动过，在步骤 12 前需要先以 OPEN 状态正常启动 BP2_A。

其次，以 MOUNT 方式启动目标库，修改目标库的 CUR LSN 大于源库的联机日志中的 CUR LSN。修改为源库的 CUR LSN+1。

```
SP_ADJUST_CUR_LSN(621564);
```

如果目标库 BP2_A 的 CUR LSN 本来就大于源库的 CUR LSN，则跳过这一步。

再次，再将目标库 ALTER 为 OPEN 模式。

```
alter database open;
```

最后，正常退出目标库 BP2_A、MP。

7. 重新依次启动 MP、BP、SP，脱机方式迁移表空间过程结束。验证迁移后数据是否正确。

8.7.2 表迁移

表迁移分为两种情况：一是基于分区的迁移，专门用于分区子表；二是基于分区组的迁移，专门用于整表迁移。

8.7.2.1 基于分区的迁移

基于分区的迁移方式将分区表中的一个子分区从一个表空间迁移到另一个表空间。如果使用基于分区的迁移方式，创建表时不允许使用分区组，并且分区迁移方式不支持整表迁移。

语法如下：

```
Alter TABLE [<模式名>.]<表名> MOVE PARTITION <分区名> TO <表空间名> [FAST]
```

<表名>分区表的主表名。只有未使用分区组创建的主表才支持使用基于分区的迁移。

<分区名>待迁移分区的名字。

<表空间名>目标表空间名。必须是实际存在的表空间名。普通表可以迁移到普通表空间或混合表空间，HUGE 表只能迁移到混合表空间。

[FAST]是否使用快速方式迁移，缺省使用普通方式迁移。FAST 方式通过数据的转换和数据的封装直接对 B 树进行操作，省去了普通插入方式下各个操作符之间的跳转，提升了迁移的效率，但对于约束的检查等由用户保证，系统将不处理有约束冲突的数据。普通方式迁移选择查询插入方式迁移数据，可以保证数据的正确性和约束的有效性，效率比前者要低。HUGE 表不支持 FAST 方式。

例 将 test22 表的 PAR5 子分区从表空间 TS_1 快速迁移到表空间 TS_2，然后再迁移到 TS_1 表空间。

DM8 分布计算集群

```

//数据准备

create table test22(c1 varchar(100),c2 float,c3 double, c4 int, c5 bigint , c6 int)

partition by range(c4, c5, c6) (

PARTITION PAR1 values less than(10,10, 10) storage(on TS_1),

PARTITION PAR2 values equ or less than(10,10, 100) storage(on TS_2),

PARTITION PAR3 values less than(10,10,170) storage(on TS_1),

PARTITION PAR4 values less than(10,10, 330) storage(on TS_2),

partition par5 values less than(maxvalue,maxvalue,maxvalue) storage(on TS_1));

insert into test22 select level,level,level,level,level,level from dual connect by level < 1000;

commit;

//将 PAR5 子分区从表空间 TS_1 快速迁移到表空间 TS_2

alter table test22 move partition PAR5 to TS_2 FAST;

//将 PAR5 子分区从表空间 TS_2 迁移到表空间 TS_1

alter table test22 move partition PAR5 to TS_1;
  
```

8.7.2.2 基于分区组的迁移

基于分区组的迁移方式是将整个分区表从一个分区组迁移到另外一个分区组。通过改变分区组的方式达到改变 RAFT 组的目的。基于分区组方式的迁移仅支持整表迁移。

语法如下：

```
Alter TABLE [<模式名>.]<表名> MOVE TO <pg 名> [FAST]
```

<表名>待迁移分区表的主表名。必须是基于分区组创建的分区表才支持使用基于分区组的迁移。

<pg 名>目标分区组的名字。原表的分区组与目标<pg 名>除了存储位置不同,其他(分区方式,分区个数等)都必须相同。

DM8 分布计算集群

[FAST]是否使用快速方式迁移，缺省使用普通方式迁移。HUGE 表不支持 FAST 方式。

例 将 test1 表从 PG1 快速迁移到 PG2 分区。

```
//数据准备
create partition group pg1 partition by hash(BIGINT) partitions 8 storage (on TS_1,TS_2);
create partition group pg2 partition by hash(BIGINT) partitions 8 storage (on TS_2,TS_1);
create table test1(c1 varchar(100),c2 float,c3 double, c4 int, c5 bigint , c6 int) using partition
group pg1 by (c5);

insert into test1 select level,level,level,level,level,level from dual connect by level < 10000;

commit;

//将 test1 表从当前组 (PG1) 快速迁移到 PG2 组

alter table test1 move to pg2 fast;

//将 test1 表从当前组 (PG2) 迁移到 PG1 组

alter table test1 move to pg1;
```

8.7.3 库迁移

所有节点正常退出的情况下，可以直接手动拷贝整个数据库到目标 BP 上，然后通过辅助步骤修改字典、控制文件以及更新系统表。

因为停机会影响到正常的业务，所以不推荐用户使用此种方式。

8.8 备份与还原

8.8.1 概述

DMDPC 集群中，SP 因为不存储数据，不需要备份与还原。需要备份与还原的节点为 MP 和 BP 节点。如果 MP 或 BP 为多副本集群，则参与备份的只有主节点，参加还原的为所有 MP 和 BP 节点。

8.8.1.1 完整的备份与还原流程

目前，DMDPC 仅支持单库、整个集群和归档的备份与还原。

针对不同的备份对象，一个完整的备份与还原过程包含的步骤不同。下面分别进行介绍。

8.8.1.1.1 数据备份与还原

8.8.1.1.1.1 单库

一个完整的单库的备份与还原过程包括：备份—还原。

- 备份

备份时在联机备份和脱机备份中二选一即可。

单库的联机备份专用于 MP 和 BP 节点。如果是单副本则直连 MP 或 BP 执行；如果是多副本只需直连主 MP 或主 BP 执行。

单库的脱机备份，可用于 MP 或 BP 节点。需要使用 DMRMAN 工具对某一个 BP 或 MP 节点单独执行。

表 8.1 单库的备份操作

| 备份粒度 | 状态 | 备份（二选一） |
|-----------|----|---------|
| 单库（MP、BP） | 联机 | ✓ |
| 单库（MP、BP） | 脱机 | ✓ |

- 还原

还原只支持脱机还原。既可使用联机备份集也可使用脱机备份集。需要使用 DMRMAN 工具对所有的 BP 或 MP 节点分别单独执行。为了保证数据一致性，多副本中的所有节点使用同一份备份集。

还原分为数据还原和数据恢复两步。数据恢复又细分为恢复一致性和更新 DB_MAGIC

DM8 分布计算集群

两步。

表 8.2 单库的还原操作

| 备份粒度 | 状态 | 还原 | | |
|------------|----|------|-------|-------------|
| | | 数据还原 | 数据恢复 | |
| | | | 恢复一致性 | 更新 DB_MAGIC |
| 单库 (MP、BP) | 脱机 | ✓ | ✓ | ✓ |

8.8.1.1.1.2 整个集群

一个完整的整个集群的备份与还原过程包括：备份—还原。

- 备份

备份时在联机备份和脱机备份中二选一即可。

整个集群的联机备份，只需连接一个 SP 执行联机备份语句，即可完成整个集群中的所有 MP 和 BP 节点的备份。整个集群的脱机备份，对于单副本集群，需要使用 DMRMAN 工具对所有 BP 和 MP 节点分别单独执行备份，对于多副本集群，仅需要备份每个 RAFT 组中的主节点。

表 8.3 整个集群的备份操作

| 备份粒度 | 状态 | 备份 (二选一) |
|------|----|----------|
| 整个集群 | 联机 | ✓ |
| 整个集群 | 脱机 | ✓ |

- 还原

整个集群的还原只支持脱机方式。既可使用联机备份集也可使用脱机备份集。需要使用 DMRMAN 工具对所有 BP 和 MP 节点分别单独执行还原。为了保证数据一致性，多副本中的所有节点使用同一份备份集。

DM8 分布计算集群

还原分为数据还原和数据恢复两步。数据恢复又细分为恢复一致性、更新 DPC_MAGIC、更新 DB_MAGIC 三步。

表 8.4 整个集群的还原操作

| 备份粒度 | 状态 | 还原 | | | |
|------|----|------|-------|-----------------|-------------|
| | | 数据还原 | 数据恢复 | | |
| | | | 恢复一致性 | 更新 DPC_MAGIC | 更新 DB_MAGIC |
| 整个集群 | 脱机 | ✓ | ✓ | ✓ | ✓ |

8.8.1.1.2 归档备份与还原

归档备份与还原过程包括：备份—还原。

归档备份时，用户可根据实际需要选择联机或脱机执行，二选一即可。归档还原只支持脱机还原。

8.8.1.2 魔数介绍

为了能够唯一标识一个 DMDPC 集群以及 DMDPC 集群中同一个批次的备份集，特意为 DMDPC 增加了 DPC_MAGIC 和 BAK_MAGIC 两个魔数。具体含义说明如下：

- **DPC_MAGIC** DMDPC 集群魔数，用于唯一标识一个 DMDPC 集群，同一个 DMDPC 集群内的 MP 和 BP 节点拥有相同的 DPC_MAGIC 值。DPC_MAGIC 在 MP 初始化时生成。BP 初始化完成后，首次启动时向 MP 获取 DPC_MAGIC 值并写入 BP 本地。

- **BAK_MAGIC** 备份集魔数，用于唯一标识 DMDPC 集群内同一批次的 MP 和 BP 的备份集。BAK_MAGIC 在对 DMDPC 节点执行脱机备份时由用户指定，并确保同一批次的备份集 BAK_MAGIC 相同，以便区分不同批次的备份集，避免备份集出现混用，脱机备份未指定时 BAK_MAGIC 为 0。而在连接 SP 执行集群联机备份时，BAK_MAGIC 由

DM8 分布计算集群

MP 自动生成并自动同步给所有 BP 节点，无需用户指定。可通过 DMRMAN 工具的 SHOW BACKUPSET 命令查看备份集的 BAK_MAGIC。使用备份集进行数据还原和数据恢复时指定的 BAK_MAGIC 需要与备份集的 BAK_MAGIC 一致，若备份集中 BAK_MAGIC 为 0，数据还原和数据恢复时可以不指定 BAK_MAGIC。

8.8.2 脱机备份

目前，DMDPC 集群仅支持脱机库级备份与脱机归档备份。下面分别进行介绍。

8.8.2.1 库级备份

脱机库级备份分为完全备份和增量备份两种。

8.8.2.1.1 完全备份

通过 DMRMAN 工具对 DMDPC 集群执行脱机库级完全备份。

由于 MP 和 BP 节点上都存放有数据，因此需要分别对 MP 和 BP 执行备份操作并生成相应的备份集。对于单副本架构的 DPC 集群，如果存在多个 BP 节点，则需要备份 MP 节点和所有的 BP 节点。对于多副本架构的 DPC 集群，仅需要备份每个 RAFT 组中的主节点（主 MP、主 BP），从节点不支持备份。开始执行 DMDPC 集群的脱机备份时，一定要确保 MP 和所有 BP 节点已经退出，否则无法保证还原出来的 MP 和 BP 节点的数据一致性。

对于脱机备份，MP 和 BP 备份集的 BAK_MAGIC 由用户指定并确保唯一，对同一次备份操作，MP 和 BP 需要指定成相同的 BAK_MAGIC 值。建议用户指定 BAK_MAGIC，因为如果不指定，则备份集上的 BAK_MAGIC 为 0，还原时有可能会出现备份集混淆的情况。

使用 BAK_MAGIC 示例如下：

备份 MP：

```
RMAN>BACKUP DATABASE 'E:\dpc\mp\DAMENG\dm.ini' BACKUPSET
```

DM8 分布计算集群

```
'E:\dpc\mp\DAMENG\bak\mp_back' USE BAK_MAGIC 132256;
```

备份 BP:

```
RMAN>BACKUP DATABASE 'E:\dpc\bp\DAMENG\dm.ini' BACKUPSET
```

```
'E:\dpc\bp\DAMENG\bak\bp_back' USE BAK_MAGIC 132256;
```

对于非多副本架构的 DPC 集群备份（含 1 个 MP 节点，2 个 BP 节点，1 个 SP 节点）备份，示例如下：

备份 MP:

```
RMAN>BACKUP DATABASE 'E:\dpc\mp\DAMENG\dm.ini' BACKUPSET
```

```
'E:\dpc\mp\DAMENG\bak\mp_back' USE BAK_MAGIC 132256;
```

备份 BP1:

```
RMAN>BACKUP DATABASE 'E:\dpc\bp1\DAMENG\dm.ini' BACKUPSET
```

```
'E:\dpc\bp1\DAMENG\bak\bp1_back' USE BAK_MAGIC 132256;
```

备份 BP2:

```
RMAN>BACKUP DATABASE 'E:\dpc\bp2\DAMENG\dm.ini' BACKUPSET
```

```
'E:\dpc\bp2\DAMENG\bak\bp2_back' USE BAK_MAGIC 132256;
```

对于多副本架构的 DPC 集群（1MP，3 个同属一个 RAFT 组的 BP 节点，1 个 SP 节点）备份。

示例如下（MP 多副本类似，此处不再举例）：

备份 MP:

```
RMAN>BACKUP DATABASE 'E:\dpc\mp\DAMENG\dm.ini' BACKUPSET
```

```
'E:\dpc\mp\DAMENG\bak\mp_back' USE BAK_MAGIC 132256;
```

找到主库 BP1，备份主 BP1 即可：

```
RMAN>BACKUP DATABASE 'E:\dpc\bp1\DAMENG\dm.ini' BACKUPSET
```

DM8 分布计算集群

```
'E:\dpc\bp1\DAMENG\bak\bp1_back' USE BAK_MAGIC 132256;
```

8.8.2.1.2 增量备份

通过 DMRMAN 工具对 DMDPC 集群执行脱机增量备份。执行增量备份时同样也需要指定 BAK_MAGIC，需要分别对 MP 和 BP 执行备份操作并生成相应的备份集。除需额外指定 BAK_MAGIC 以外与普通单节点的增量备份语法一致。

在执行增量备份时为了确保 DMDPC 集群各节点数据一致，需要注意以下事项：

1. 执行增量备份时，需要保证 DMDPC 集群内各节点均正常退出。
2. 对 MP 和 BP 执行同一批次的增量备份操作时，要求 MP 和所有 BP 节点都有可用的基备份集，且这些基备份集是在同一批次的备份操作中产生的（基备份集的 BAK_MAGIC 相同）。
3. 对 MP 和 BP 执行同一批次的增量备份操作时，需要指定相同的 BAK_MAGIC 值。
4. 执行不同批次的增量备份操作时，需要指定不同的 BAK_MAGIC 值，避免出现备份集混用的情况，由用户确保 BAK_MAGIC 的唯一性。

一次 DMDPC 集群的增量备份操作示例如下：

第一步，准备基备份集。若基备份集已存在，可以跳过这一步。

MP:

```
RMAN>BACKUP DATABASE 'E:\dpc\mp\DAMENG\dm.ini' BACKUPSET  
'E:\dpc\mp\DAMENG\bak\mp_back' USE BAK_MAGIC 132256;
```

BP:

```
RMAN>BACKUP DATABASE 'E:\dpc\bp\DAMENG\dm.ini' BACKUPSET  
'E:\dpc\bp\DAMENG\bak\bp_back' USE BAK_MAGIC 132256;
```

在对集群执行一次增量备份时，各节点使用的基备份集需要确保是在同一批次的备份操作中生成的，所有基备份集的 BAK_MAGIC 相同，以确保还原后各节点数据一致。

DM8 分布计算集群

第二步,在所有节点均有可用的同一批次的基备份集的情况下,开始执行脱机增量备份。

执行增量备份时,同样需要指定 BAK_MAGIC,同一批次的增量备份操作中,MP 和所有 BP 节点必须指定相同的 BAK_MAGIC 值。

注意:本示例中使用的是指定基备份搜索目录(WITH BACKUPDIR)的方式进行增量备份,需要用户确保在这些目录下可以搜索到最近一个批次的所有节点的基备份集,如果是通过指定基备份集(BASE ON BACKUPSET)的方式执行增量备份,则需要用户确保各节点指定的基备份集是同一个批次产生的(即所有基备份集的 BAK_MAGIC 相同)。

MP:

```
RMAN>BACKUP DATABASE 'E:\dpc\mp\DAMENG\dm.ini' INCREMENT WITH BACKUPDIR  
'E:\dpc\mp\DAMENG\bak\mp_back' BACKUPSET  
'E:\dpc\mp\DAMENG\bak\mp_back_increment' USE BAK_MAGIC 112233;
```

BP:

```
RMAN>BACKUP DATABASE 'E:\dpc\bp\DAMENG\dm.ini' INCREMENT WITH BACKUPDIR  
'E:\dpc\bp\DAMENG\bak\bp_back' BACKUPSET  
'E:\dpc\bp\DAMENG\bak\bp_back_increment' USE BAK_MAGIC 112233;
```

8.8.2.2 归档备份

通过 DMRMAN 工具对 DMDPC 集群执行脱机归档备份与普通单机归档备份流程一致。

由于 MP 和 BP 节点上都存放有数据,因此需要分别对 MP 和 BP 执行备份操作并生成相应的备份集。对于单副本架构的 DMDPC 集群,如果存在多个 BP 节点,则需要备份 MP 节点和所有的 BP 节点。对于多副本架构的 DMDPC 集群,仅需要备份每个 RAFT 组中的主节点(主 MP、主 BP),从节点不支持备份。

脱机归档备份时,无需指定 BAK_MAGIC。

8.8.3 联机备份

目前，DMDPC 集群仅支持联机库级备份与联机归档备份。联机库级备份分为完全备份和增量备份两种，暂不支持 DDL_CLONE 备份。

DMDPC 的联机备份语法和普通单机没有差别，只是在执行流程上增加了 SP、MP 和 BP 之间的协同处理。

只要连接任意一个 SP 执行备份操作，即可完成对整个 DMDPC 系统的联机备份。直连 MP 备份、直连 BP 备份是特殊情况下使用，下文会详细介绍。

8.8.3.1 直连 SP 备份

8.8.3.1.1 整个集群备份

8.8.3.1.1.1 完全备份

直接在 SP 端执行联机备份语句，即可完成 MP 和所有 BP 节点的备份，执行成功后，会在 MP 和所有 BP 节点本地生成各自的备份集，SP 本地不会有备份集产生。如果是 DMDPC 多副本环境，则只有主 MP 和主 BP 会参与备份并在本地生成备份集。

各备份集上的 BAK_MAGIC 值由 MP 自动生成并同步至各 BP 节点，不需要再由用户指定，每执行一次联机备份，MP 会分配一个不同的 BAK_MAGIC，用于标识不同批次的联机备份集，还原时必须使用同一批次产生的 MP 和 BP 备份集。

为了保证 MP 和各 BP 节点备份下来的数据处于事务一致状态，在备份流程中会暂停 MP 的提交请求，这个暂停动作理论上会很快完成，但为了避免影响上层业务，可以考虑在系统空闲时执行联机备份。

联机库级备份的注意事项：

1. 联机备份时，若指定了备份集名称，则在 MP 和 BP 端生成备份集时会在备份集名称之后各自添加实例名后缀，避免 MP 和 BP 部署在同一台机器上，在同一个路径下生成备份

DM8 分布计算集群

集时发生名称冲突。

2. 联机备份开始前如果有 MP 或 BP 节点故障，则不允许执行备份。

3. 联机备份过程中如果有 MP 或 BP 节点故障，则备份失败。

4. 支持直接连接 MP 和 BP 节点执行联机备份，但是不建议使用。因为 DMDPC 下各节点的 LSN 值不统一，无法保证各节点事务一致，通过这种方式可能引发数据不一致的风险。

5. 联机备份过程中如果有增删 BP 节点的动作（非副本节点增删），则新增节点不会参与备份，若备份过程中节点被删除，则会导致备份失败。

例 连接 SP 执行库级联机备份。

```
SQL>BACKUP DATABASE;
```

8.8.3.1.1.2 增量备份

DMDPC 的联机增量备份（可支持差异增量备份和累积增量备份），需要遵守和普通单机增量备份相同的规则要求。例如：MP 和 BP 各节点本地必须已经存在有可用的基备份集，具体规则可参考《DM8 备份与还原》手册，此处不再详细说明。

直接在 SP 端执行联机增量备份语句，即可完成 MP 和所有 BP 节点的增量备份，执行成功后，会在 MP 和所有 BP 节点本地生成各自的增量备份集，SP 本地不会有增量备份集产生。如果是 DPC 多副本环境，则只有主 MP 和主 BP 会参与备份并在本地生成增量备份集。

同样的，增量备份集上也有 BAK_MAGIC 标识，由 MP 自动生成并同步至各 BP 节点。

每执行一次增量备份，MP 会分配一个不同的 BAK_MAGIC，用于标识不同批次的增量备份集，还原时必须使用同一批次产生的 MP 和 BP 的增量备份集。

为了保证 MP 和各 BP 节点备份下来的数据处于事务一致状态，在备份流程中会暂停 MP 的提交请求，这个暂停动作理论上会很快完成，但为了避免影响上层业务，可以考虑在系统空闲时执行联机增量备份。

执行联机增量备份时，不要求所有节点一定拥有同一批次的基备份集，只要能够生成同

DM8 分布计算集群

一批次的增量备份集，并使用同一批次的增量备份集对各节点执行还原，就一定能够将各节点恢复到事务一致状态。

联机完全备份的注意事项，对联机增量备份也同样适用，此处不再重复说明，具体请参考 [8.8.3.1.1.1 完全备份](#)。

执行增量备份集指定基备份集路径后，会在指定的基备份集路径上拼接每个节点自己的实例名作为真正的基备份集路径搜索基备份集。如指定了基备份集路径为 '/home/backupset/dm_bak'，对于实例名为 MP_INSTANCE 的 MP 节点，会在 '/home/backupset/dm_bak_MP_INSTANCE' 下搜索自己的基备份集。因此执行增量备份时，需要确保执行增量备份的节点指定的基备份路径下对应的基备份集均存在。

例 连接 SP 执行库级联机增量备份。

```
SQL>BACKUP DATABASE INCREMENT WITH BACKUPDIR '/home/backupset/dm_bak' BACKUPSET  
'/home/backupset/db_increment_bak';
```

8.8.3.1.2 归档备份

DMDPC 集群归档联机备份执行方式同库级一样，也是直接在 SP 端执行联机备份语句，即可完成 MP 和所有 BP 节点的备份。执行成功后，会在 MP 和所有 BP 节点本地生成各自的备份集，SP 本地不会有备份集产生。如果是 DMDPC 多副本环境，则只有主 MP 和主 BP 会参与备份并在本地生成备份集。与库级备份不同的是，归档备份集中不包含备份集上的 BAK_MAGIC 值。

联机归档备份的注意事项：

1. 联机备份时，若指定了备份集名称，则在 MP 和 BP 端生成备份集时会在备份集名称之后各自添加实例名后缀，避免 MP 和 BP 部署在同一台机器上，在同一个路径下生成备份集时发生名称冲突。

2. 联机备份开始前所有的主 MP 和主 BP 须全部正常。如果有 MP 或 BP 节点故障，则

不允许执行备份。

3. 联机备份过程中，如果有主 MP 或主 BP 节点故障，则备份失败。

4. 不允许直接连接 MP 或 BP 节点执行联机归档备份，因为无法保证各节点的事务一致性。

5. 联机备份过程中如果有增删 BP 节点的动作（非副本节点增删），则新增节点不会参与备份，若备份过程中节点被删除，则会导致备份失败。

例 连接 SP 执行归档备份。

```
SQL>BACKUP ARCHIVELOG;
```

8.8.3.2 直连 MP 备份

8.8.3.2.1 单库备份

专门用于 MP 多副本集群中。当 MP 多副本集群中的某一个非主 MP 节点故障且无法重新启动时，连接 SP 执行集群备份还原的开销较大，此时可通过直连主 MP 执行备份还原的方式来恢复故障的 MP 节点。具体操作分为：一直连主 MP 进行备份，获取备份集；二使用 DMRMAN 工具和备份集对故障 MP 进行还原；三重启故障 MP，此时集群内主 MP 会自动对故障 MP 还原到数据一致状态。

DM8 分布计算集群

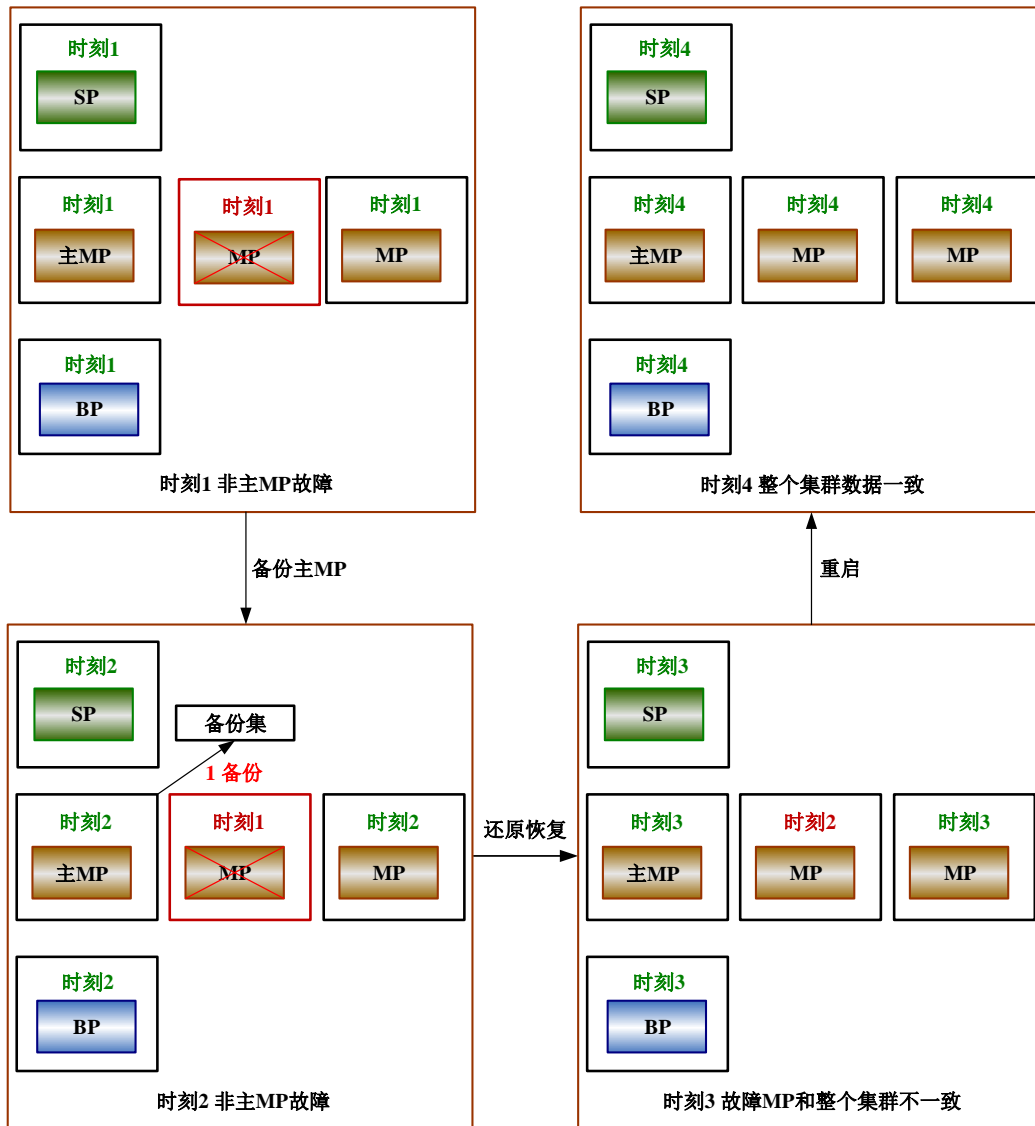


图 8.2 直连主 MP 执行备份还原

需要注意的是，单副本环境下，直连 MP 执行联机备份，然后进行还原操作，重启之后，因为没有多副本集群中的主节点自动恢复故障节点的数据一致性的功能，所以无法保证集群数据一致性。因此，单副本集群环境建议使用连接 SP 执行备份，用同一批备份集还原集群，才可保证数据的一致性。

例 直连主 MP 执行备份操作。

```
SQL>BACKUP DATABASE;
```

直连主 MP 执行备份生成的备份集中没有 BAK_MAGIC 信息，因此用这个备份集进行

DM8 分布计算集群

数据还原和数据恢复时也无需指定 BAK_MAGIC。

若需要将一套 DMDPC 集群的 MP 生成的备份集还原到另外一套 DMDPC 集群的 MP 节点上，则需要保证新 MP 节点的所有初始化参数与老 MP 节点完全相同。同时还原恢复后启动后，新 MP 的注册信息已被还原，因此还需要修改新 MP 中注册信息。

例 将 MP_1 还原到 MP_2 上后，修改 MP_2 上的注册信息。

```
//重命名 MP_1 为 MP_2
SQL>SP_RENAME_DPC_INSTANCE('MP_1','MP_2');

//更新 MP_2 端口信息
SQL>SP_MODIFY_DPC_INSTANCE('MP_2','192.168.1.68','192.168.1.68', 6004,5239);
```

8.3.3.2 归档备份

直连主 MP 备份归档成功后，只会在当前 MP 节点本地生成 MP 的备份集。归档备份的注意事项请参考 [8.8.3.1.2 归档备份](#)。

例 连接 MP 执行归档备份。

```
SQL>BACKUP ARCHIVELOG;
```

8.8.3.3 直连 BP 备份

专门用于 BP 多副本集群中。当 BP 多副本集群中的某一个非主 BP 节点故障时，连接 SP 执行集群备份还原的开销较大，也可通过直连主 BP 执行备份还原的方式恢复故障的 BP 节点。具体操作分为：一直连主 BP 进行备份，获取备份集；二使用 DMRMAN 工具和备份集对故障 BP 进行还原；三重启故障 BP。重启之后，主 BP 会自动对故障 BP 进行异步恢复。

需要注意的是，单副本环境下，直连 BP 模式节点执行联机备份，然后进行还原操作无法保证集群数据一致性。因此，单副本集群环境建议使用连接 SP 执行备份，用同一批备份集还原整个集群，才可保证数据的一致性。

下面对不同模式（BP 模式和 BS 模式）启动的 BP 节点备份分别进行介绍。

8.8.3.3.1 直连 BP 模式节点

8.8.3.3.1.1 单库备份

直连 BP 模式节点执行备份生成的备份集中没有 BAK_MAGIC 信息，因此用这个备份集进行数据还原和数据恢复时也无需指定 BAK_MAGIC。

例 直连 BP 模式节点执行备份操作。

```
SQL>BACKUP DATABASE;
```

若需要将生成的备份集还原到一个新的 BP 模式节点上，则需要保证新 BP 模式节点的所有初始化参数与老 BP 模式节点完全相同。同时还原后启动后，新 BP 模式节点的注册信息已被还原，因此还需要修改新 BP 模式节点中注册信息。

例 BP_1 和 BP_2 均以 BP 模式启动。将 BP_1 还原到 BP_2 上后，修改 BP_2 上的注册信息。操作登录 MP 执行。

```
//重命名 BP_1 为 BP_2  
SQL>SP_RENAME_DPC_INSTANCE('BP_1','BP_2');  
  
//更新 MP_2 端口信息  
SQL>SP_MODIFY_DPC_INSTANCE('BP_2','192.168.1.69','192.168.1.69', 6014,5249);
```

8.8.3.3.1.2 归档备份

直连 BP 备份归档成功后，只会在当前 BP 节点本地生成 BP 的备份集。归档备份的注意事项请参考 [8.8.3.1.2 归档备份](#)。

例 连接 BP 执行归档备份。

```
SQL>BACKUP ARCHIVELOG;
```

8.8.3.3.2 直连 BS 模式节点

目前直连 BS 节点进行备份有两种登录方式: 一是以非 LOCAL 方式登录; 二是以 LOCAL 方式登录。下面分别介绍两种登录方式的备份方法:

8.8.3.3.2.1 非 LOCAL 方式

以非 LOCAL 方式登录 BS 节点执行备份。分为库级备份和归档备份。

8.8.3.3.2.1.1 单库备份

非 LOCAL 方式登录 BS 节点执行备份时, BS 节点会以 SP 方式执行备份语句, 执行备份语句时, 相当于连接 SP 对整个 DMDPC 集群进行一次备份, 对各个主节点生成一批备份集, 此时会生成一个 BAK_MAGIC 值, 这批备份集的 BAK_MAGIC 值相同, 以用户进行还原时使用同一批备份集还原集群, 从而使整个集群恢复一致。当使用这批备份集还原节点时, 数据还原和数据恢复时需要指定 BAK_MAGIC 进行还原, 可以通过 show backupset 命令查看备份集 BAK_MAGIC 值。非 LOCAL 方式登录备份时, 若以非 USE_AP=2 的方式备份, 需要确保集群中各个主节点的 dmap 服务启动。

例 非 LOCAL 方式登录 BS 备份, 等效于对 DMDPC 集群整体执行一次备份。

```
./disql SYSDBA/SYSDBA@192.168.100.168:5600
```

```
SQL>BACKUP DATABASE;
```

8.8.3.3.2.1.2 归档备份

非 LOCAL 方式登录 BS 备份, 等效于对 DMDPC 集群整体执行一次归档备份。具体用法和 [8.8.3.1.2 归档备份](#) 一样。

8.8.3.3.2.2 LOCAL 方式

8.8.3.3.2.2.1 单库备份

以 LOCAL 方式登录 BS 节点执行备份，会以 BP 的方式执行备份语句，等效于直连 BP 备份，仅会备份自身节点，这种情况下备份集中不会生成 BAK_MAGIC，默认为 0，因此还原时无需指定 BAK_MAGIC。

例 LOCAL 方式登录 BS 备份，仅会备份自身 BS 节点。

```
./disql SYSDBA/SYSDBA@192.168.100.168:5600#{mpp_type=local}  
  
SQL>BACKUP DATABASE;
```

需要注意的是，单副本环境下，直连 BS 执行 LOCAL 方式登录执行联机备份，然后进行还原操作无法保证集群数据一致性。因此，单副本集群环境建议使用连接 SP 执行备份，或者连接 BS 非 LOCAL 方式备份，然后用同一批备份集还原集群，才可保证数据的一致性。

8.8.3.3.2.2.2 归档备份

直连 BP 备份归档成功后，只会在当前 BP 节点本地生成 BP 的备份集。这和非 LOCAL 方式下登录 BP 进行归档备份完全一样。

8.8.4 还原

还原是备份的逆过程。DMRMAN 工具通过使用 MP 和 BP 的备份集，依次对目标库的 MP 和 BP 执行还原。

还原时，如果只还原 DMDPC 集群中的 MP 或者某个 BP 节点，则无法保证 MP 和 BP 节点的数据一致性，原则上是需要对整个 DMDPC 集群内的所有 MP 和 BP 节点全部执行还原。

如果存在 MP 或 BP 多副本，则必须要对所有副本节点全部执行还原。

8.8.4.1 还原须知

本节说明对完全备份集和增量备份集均适用。

8.8.4.1.1 BAK_MAGIC 用法

如果备份时未指定 BAK_MAGIC, 数据还原、数据恢复时也不需要指定 BAK_MAGIC, 但这种方式无法识别备份集混用的情况。

如果备份时指定了 BAK_MAGIC, 数据还原、数据恢复时需要指定相同的 BAK_MAGIC, 以确保使用同一组备份集进行还原, 还原、恢复时会校验备份集中 BAK_MAGIC 与指定 BAK_MAGIC 是否一致, 可以通过 DMRMAN 工具的 SHOW BACKUPSET 命令查看备份集的 BAK_MAGIC 字段值。

8.8.4.1.2 DPC_MAGIC 用法

如果是对整个 DMDPC 集群进行还原, 则还需要增加更新 DPC_MAGIC 这一步, 以区分不同的 DMDPC 集群, DPC_MAGIC 也由用户指定并确保唯一, MP 和 BP 需要更新为相同的 DPC_MAGIC 值, 更新 DPC_MAGIC 的动作必须要放在更新 DB_MAGIC 之前。

8.8.4.1.3 恢复一致性

恢复一致性在单库还原和整个集群还原中用法略有不同, 下面详细介绍。

8.8.4.1.3.1 指定备份集恢复

■ 单副本

对于单副本的 DMDPC 环境, 无论是使用联机备份集还是脱机备份集, 都可以将各节点恢复到备份结束时的位置, 和普通单机的恢复一致性完全相同。

■ 多副本

DM8 分布计算集群

对于多副本架构的 DMDPC 环境，在常规恢复一致性时的基础上，建议指定 UNTIL END_LSN 选项，可保证所有节点数据恢复到一致性状态。

在多副本架构中在使用联机备份集恢复时可能存在以下几种情况：

1. 各节点（主 MP 或主 BP）生成联机备份集时，各备份集备份结束时的 END_LSN 位置可以保证数据是一致的，但主 MP 或主 BP 的 C_LSN（已写入多数副本节点的 LSN）不一定推进到了 END_LSN 位置，备份集中记录的 C_LSN 有可能小于 END_LSN。

2. 使用联机备份集恢复时，默认只恢复到 C_LSN 位置，此时不能保证各节点恢复出来的数据是一致的，因此新增了一个 UNTIL END_LSN 关键字，允许在 C_LSN 小于 END_LSN 的情况下，强制恢复到 END_LSN 位置。可通过 SHOW BACKUPSET 命令查看 C_LSN 和 END_LSN 信息。

对于多副本架构的 DMDPC 环境，在使用脱机备份集恢复时存在以下几种情况：

1. 如果脱机备份集是在各节点正常退出的情况下生成的，UNTIL END_LSN 也可不指定，不指定 UNTIL END_LSN 也可以将各节点恢复到数据一致状态。

2. 如果脱机备份集是在异常退出的情况下生成的，则必须使用 UNTIL END_LSN 方式才可以将各节点恢复到最新状态（包括所有副本节点），否则不能保证各节点的数据是一致的。

8.8.4.1.3.2 指定归档恢复

和普通单节点相比，不论是单副本还是多副本的 DMDPC 集群恢复，均需要保证将所有 MP 和 BP 节点恢复到一个事务一致（数据一致）的状态，因此在指定归档恢复时，采用常规的恢复到最新还不行，还需要对 DMDPC 进行一些额外的操作。

DMDPC 归档恢复操作：

1. 支持 UNTIL LSN 恢复方式，但是不建议用户使用这种方式。因为这种方式无法通过 LSN 判断出各节点是否处于一致状态，具体为 DMDPC 下各节点的 LSN 值不统一，通过 UNTIL LSN 的方式恢复无法保证集群恢复到数据一致状态，可能引发数据不一致的风险。

DM8 分布计算集群

2. 建议使用 UNTIL TIME 方式恢复，将各节点恢复到同一时间点，确保将各节点数据恢复到一致状态，此方式必须和 DPC_LOG_INTERVAL 参数配合使用，否则无法保证各节点的数据一致性。

3. 如果不指定 UNTIL TIME，直接将各节点恢复到最新状态，则需要用户自己保证各节点恢复后的数据一致性。

指定 UNTIL TIME 方式恢复时，如果在日志中没有找到相匹配的时间点，则会一直恢复到最新，可通过日志中记录的恢复结束信息来确认各节点是否恢复到了相同的时间点。

8.8.4.2 单库的还原

专门用于直连 MP 备份或直连 BP 备份之后，对 MP 或 BP 的还原。

单库还原具体包括数据还原和数据恢复两步。数据恢复又细分为恢复一致性和更新 DB_MAGIC 两步。

其中恢复一致性有两种方式：指定备份集恢复和指定归档恢复。

下面按照恢复一致性的两种方式进行分类，分别说明单库还原的操作步骤。

8.8.4.2.1 指定备份集恢复

本节的恢复一致性操作采用指定备份集恢复方式。

示例如下：

1. 使用完全备份集还原

在准备好完全备份集的基础上，对 DMDPC 集群执行一次完全还原操作。

还原单副本 MP：

```
//数据还原  
RMAN>RESTORE DATABASE 'E:\dpc2\mp\DAMENG\dm.ini' FROM BACKUPSET  
'E:\dpc\mp\DAMENG\bak\mp_back' USE BAK_MAGIC 132256;
```

DM8 分布计算集群

```
//使用备份集恢复

RMAN>RECOVER DATABASE 'E:\dpc2\mp\DAMENG\dm.ini' FROM BACKUPSET

'E:\dpc\mp\DAMENG\bak\mp_back' USE BAK_MAGIC 132256;

//更新 DB_MAGIC

RMAN>RECOVER DATABASE 'E:\dpc2\mp\DAMENG\dm.ini' UPDATE DB_MAGIC;
```

还原单副本 BP:

```
//数据还原

RMAN>RESTORE DATABASE 'E:\dpc2\bp\DAMENG\dm.ini' FROM BACKUPSET

'E:\dpc\bp\DAMENG\bak\bp_back' USE BAK_MAGIC 132256;

//使用备份集恢复

RMAN>RECOVER DATABASE 'E:\dpc2\bp\DAMENG\dm.ini' FROM BACKUPSET

'E:\dpc\bp\DAMENG\bak\bp_back' USE BAK_MAGIC 132256 UNTIL END_LSN;

//更新 DB_MAGIC

RMAN>RECOVER DATABASE 'E:\dpc2\bp\DAMENG\dm.ini' UPDATE DB_MAGIC;
```

2. 使用增量备份集还原

增量还原与增量备份相对应。执行增量还原时，指定的 BAK_MAGIC 为增量备份集的 BAK_MAGIC 而不是基备份集的 BAK_MAGIC。为了确保还原出来的 DMDPC 集群各节点数据一致，在一次 DMDPC 增量还原操作中，要求 MP 和所有 BP 节点指定相同的 BAK_MAGIC 值执行还原操作。

在准备好增量备份集的基础上，对 DMDPC 集群执行一次增量还原操作示例如下：

还原 MP:

```
//数据还原

RMAN>RESTORE DATABASE 'E:\dpc\mp\DAMENG\dm.ini' FROM BACKUPSET

'E:\dpc\mp\DAMENG\bak\mp_back_increment' WITH BACKUPDIR
```

DM8 分布计算集群

```
'E:\dpc\mp\DAMENG\bak\mp_back' USE BAK_MAGIC 112233;

//使用备份集恢复

RMAN>RECOVER DATABASE 'E:\dpc\mp\DAMENG\dm.ini' FROM BACKUPSET

'E:\dpc\mp\DAMENG\bak\mp_back_increment' USE BAK_MAGIC 112233;

//更新 DB_MAGIC

RMAN>RECOVER DATABASE 'E:\dpc\mp\DAMENG\dm.ini' UPDATE DB_MAGIC;
```

还原 BP:

```
//数据还原

RMAN>RESTORE DATABASE 'E:\dpc\bp\DAMENG\dm.ini' FROM BACKUPSET

'E:\dpc\bp\DAMENG\bak\bp_back_increment' WITH BACKUPDIR

'E:\dpc\bp\DAMENG\bak\bp_back' USE BAK_MAGIC 112233;

//使用备份集恢复

RMAN>RECOVER DATABASE 'E:\dpc\bp\DAMENG\dm.ini' FROM BACKUPSET

'E:\dpc\bp\DAMENG\bak\bp_back_increment' USE BAK_MAGIC 112233 UNTIL END_LSN;

//更新 DB_MAGIC

RMAN>RECOVER DATABASE 'E:\dpc\bp\DAMENG\dm.ini' UPDATE DB_MAGIC;
```

8.8.4.2.2 指定归档恢复

本节的恢复一致性操作采用指定归档恢复方式。

本节和在准备好归档的前提下，对 DMDPC 集群执行一次归档还原操作。

示例如下：

还原 MP:

```
//数据还原

RMAN>RESTORE DATABASE 'E:\dpc\mp\DAMENG\dm.ini' FROM BACKUPSET
```

DM8 分布计算集群

```
'E:\dpc\mp\DAMENG\bak\db_full_bak_for_recover_arch' USE BAK_MAGIC 112233;

//使用归档恢复

RMAN>RECOVER DATABASE 'E:\dpc\mp\DAMENG\dm.ini' WITH ARCHIVEDIR

'E:\dpc\mp\DAMENG\arch' UNTIL TIME '2021-10-10 10:56:40';

//更新 DB_MAGIC

RMAN>RECOVER DATABASE 'E:\dpc\mp\DAMENG\dm.ini' update DB_MAGIC;
```

还原 BP:

```
//数据还原

RMAN>RESTORE DATABASE 'E:\dpc\bp\DAMENG\dm.ini' FROM BACKUPSET

'E:\dpc\bp\DAMENG\bak\db_full_bak_for_recover_arch' USE BAK_MAGIC 112233;

//使用归档恢复

RMAN>RECOVER DATABASE 'E:\dpc\bp\DAMENG\dm.ini' WITH ARCHIVEDIR

'E:\dpc\bp\DAMENG\arch' UNTIL TIME '2021-10-10 10:56:40';

//更新 DB_MAGIC

RMAN>RECOVER DATABASE 'E:\dpc\bp\DAMENG\dm.ini' update DB_MAGIC;
```

8.8.4.3 整套集群的还原

DMDPC 整套集群的还原操作，是指将一套 DMDPC 集群的备份集还原到另一套 DMDPC 集群上，从而实现整套 DMDPC 的还原。

整个集群还原同样包括数据还原和数据恢复两步。数据恢复又细分为恢复一致性、更新 DPC_MAGIC、更新 DB_MAGIC 三步。更新 DPC_MAGIC 这一步是整套集群还原中特有的一步，为了标识所有节点均属于同一个集群。

其中恢复一致性有两种方式：指定备份集恢复和指定归档恢复。

目前整套 DMDPC 的还原主要有以下几种方式：

DM8 分布计算集群

1. 单副本 DMDPC 集群-->单副本 DMDPC 集群
2. 单副本 DMDPC 集群-->多副本 DMDPC 集群
3. 多副本 DMDPC 集群-->单副本 DMDPC 集群
4. 多副本 DMDPC 集群-->多副本 DMDPC 集群

下面按照恢复一致性的两种方式进行分类，分别说明整个还原的操作步骤。

8.8.4.3.1 指定备份集恢复

本节的恢复一致性操作采用指定备份集恢复方式。

8.8.4.3.1.1 单副本集群还原到单副本集群

对于单副本集群之间的还原，需要确保备份时产生备份集的集群与目标需要确保两套 DMDPC 集群的配置是同构的，即 RAFT 组数是一致的，BP 节点数目等于还原的目标集群库的 BP 节点数目，从而确保还原后的集群数据是完整的。

另外，由于还原后的 DMDPC 集群中 MP 上的注册信息仍然是旧集群 MP 的注册信息，因此还需要更新还原后的 MP 上注册信息。

将一套 DMDPC 集群备份并还原到另外一套 DPC 集群(DPC 集群 1 到 DPC 集群 2)。

DPC 集群 1 (MP: MP_1; BP: BP_11、BP_12; SP: SP1)

DPC 集群 2 (MP: MP_2; BP: BP_21、BP_22; SP: SP2)

流程如下：

- 1) 还原 MP_2、BP_21、BP_22。

为了保证还原后集群的数据一致，一定要确保还原使用的备份集是同一批次的备份集 (可通过 DMRMAN 中 SHOW BACKUPSET 命令查看备份集上的 BAK_MAGIC 字段是否一致，以确认使用的备份集是否是同一批)。另外在执行 UPDATE DB_MAGIC 前，需要更新 DPC_MAGIC，使新集群和旧集群的 DPC_MAGIC 不同，以便通过 DPC_MAGIC 和旧集

DM8 分布计算集群

群进行区分，避免造成混淆。

- 2) 正常退出 DPC 集群 2。
- 3) 使用 MP_1 的备份集还原 MP_2。

```
//数据还原
RMAN>RESTORE DATABASE 'E:\dpc2\mp_2\DAMENG\dm.ini' FROM BACKUPSET
'E:\dpc1\mp_1\DAMENG\bak\mp_1_back' USE BAK_MAGIC 132256;

//使用备份集恢复
RMAN>RECOVER DATABASE 'E:\dpc2\mp_2\DAMENG\dm.ini' FROM BACKUPSET
'E:\dpc1\mp_1\DAMENG\bak\mp_1_back' USE BAK_MAGIC 132256;

//更新 DPC_MAGIC
RMAN>RECOVER DATABASE 'E:\dpc2\mp_2\DAMENG\dm.ini' UPDATE DPC_MAGIC 123456;

//更新 DB_MAGIC
RMAN>RECOVER DATABASE 'E:\dpc2\mp_2\DAMENG\dm.ini' UPDATE DB_MAGIC;
```

使用 BP_11 的备份集还原 BP_21。

```
//数据还原
RMAN>RESTORE DATABASE 'E:\dpc2\bp_21\DAMENG\dm.ini' FROM BACKUPSET
'E:\dpc1\bp_11\DAMENG\bak\bp_11_back' USE BAK_MAGIC 132256;

//使用备份集恢复
RMAN>RECOVER DATABASE 'E:\dpc2\bp_21\DAMENG\dm.ini' FROM BACKUPSET
'E:\dpc1\bp_11\DAMENG\bak\bp_11_back' USE BAK_MAGIC 132256;

//更新 DPC_MAGIC
RMAN>RECOVER DATABASE 'E:\dpc2\bp_21\DAMENG\dm.ini' UPDATE DPC_MAGIC 123456;

RMAN>RECOVER DATABASE 'E:\dpc2\bp_21\DAMENG\dm.ini' UPDATE DB_MAGIC;
```

使用 BP_12 的备份集还原 BP_22。

DM8 分布计算集群

```
//数据还原
RMAN>RESTORE DATABASE 'E:\dpc2\bp_22\DAMENG\dm.ini' FROM BACKUPSET
'E:\dpc1\bp_12\DAMENG\bak\bp_12_back' USE BAK_MAGIC 132256;

//使用备份集恢复
RMAN>RECOVER DATABASE 'E:\dpc2\bp_22\DAMENG\dm.ini' FROM BACKUPSET
'E:\dpc1\bp_12\DAMENG\bak\bp_12_back' USE BAK_MAGIC 132256;

//更新 DPC_MAGIC
RMAN>RECOVER DATABASE 'E:\dpc2\bp_22\DAMENG\dm.ini' UPDATE DPC_MAGIC 123456;

//更新 DB_MAGIC
RMAN>RECOVER DATABASE 'E:\dpc2\bp_22\DAMENG\dm.ini' UPDATE DB_MAGIC;
```

4) 启动 MP。

通过 DlsqI 连接 MP，更新 MP 上注册信息。由于 BP11 及 BP12 的 RAFT_ID 已被表对象使用，因此无法在 MP 注册信息表中直接删除 BP11，BP12 实例。这里通过重命名和更新 IP 的操作来更新 MP 上的注册信息。

```
//重命名 MP_1 为 MP_2
SQL>SP_RENAME_DPC_INSTANCE('MP_1','MP_2');

//更新 MP_2 端口信息
SQL>SP_MODIFY_DPC_INSTANCE('MP_2','192.168.1.68','192.168.1.68', 6004, 5239);

//重命名 BP_11 实例为 BP_21,
SQL>SP_RENAME_DPC_INSTANCE('BP_11','BP_21');

//更新 BP_21 端口信息
SQL>SP_MODIFY_DPC_INSTANCE('BP_21','192.168.1.68','192.168.1.68', 6005, 5240);

//重命名 BP_12 为 BP_22
SQL>SP_RENAME_DPC_INSTANCE('BP_12','BP_22');
```

DM8 分布计算集群

```
//更新 BP_22 端口信息

SQL>SP_MODIFY_DPC_INSTANCE('BP_22', '192.168.1.68', '192.168.1.68', 6007, 5241);

//重命名 SPI 为 SP2

SQL>SP_RENAME_DPC_INSTANCE('SPI', 'SP2');

//更新 SP2 注册信息

SQL>SP_MODIFY_DPC_INSTANCE('SP2', '192.168.1.68', '192.168.1.68', 6008, 5242);
```

5) 启动 BP_21、BP_22、SP2。至此 DPC 集群备份与还原全部完成。

可通过连接 SP2 查看表 DPC_INSTANCE 或建表查询等操作确认还原后集群是否正常。

8.8.4.3.1.2 单副本集群还原到多副本集群

单副本集群还原到多副本集群，同样要求即 RAFT 组数是一致的，以确保还原后的集群数据是完整的。

另外，由于还原后的 DMDPC 集群中 MP 上的注册信息仍然是旧集群 MP 的注册信息，因此也需要更新还原后的 MP 上注册信息。

将一套单副本 DMDPC 集群环境还原到另一套多副本 DMDPC 集群环境（DPC 集群 1 到 DPC 集群 2）。

DPC 集群 1 包括 RAFT_MPI(MP_1)、RAFT_BP11(BP_1)、RAFT_BP12(BP_2) 和 RAFT_SPI(SPI)。

DPC 集群 2 包括 RAFT_MP2(MP_21、MP_22、MP_23)、RAFT_BP21(BP_11、BP_12、BP_13)、RAFT_BP22(BP_21、BP_22、BP_23) 和 RAFT_SP2(SP2)。

流程如下：

1) 还原 MP_21、MP_22、MP_23。

语句如下：

```
//还原 MP_21
```

DM8 分布计算集群

```

RMAN>RESTORE DATABASE 'E:\dpc2\mp_21\DAMENG\dm.ini' FROM BACKUPSET
'E:\dpc1\mp_1\DAMENG\bak\mp_1_back' USE BAK_MAGIC 132256;

RMAN>RECOVER DATABASE 'E:\dpc2\mp_21\DAMENG\dm.ini' FROM BACKUPSET
'E:\dpc1\mp_1\DAMENG\bak\mp_1_back' USE BAK_MAGIC 132256;

RMAN>RECOVER DATABASE 'E:\dpc2\mp_21\DAMENG\dm.ini' UPDATE DPC_MAGIC 123456;

RMAN> RECOVER DATABASE 'E:\dpc2\mp_21\DAMENG\dm.ini' UPDATE DB_MAGIC;

//还原 MP_22

RMAN>RESTORE DATABASE 'E:\dpc2\mp_22\DAMENG\dm.ini' FROM BACKUPSET
'E:\dpc1\mp_1\DAMENG\bak\mp_1_back' USE BAK_MAGIC 132256;

RMAN>RECOVER DATABASE 'E:\dpc2\mp_22\DAMENG\dm.ini' FROM BACKUPSET
'E:\dpc1\mp_1\DAMENG\bak\mp_1_back' USE BAK_MAGIC 132256;

RMAN>RECOVER DATABASE 'E:\dpc2\mp_22\DAMENG\dm.ini' UPDATE DPC_MAGIC 123456;

RMAN>RECOVER DATABASE 'E:\dpc2\mp_22\DAMENG\dm.ini' UPDATE DB_MAGIC;

//还原 MP_23

RMAN>RESTORE DATABASE 'E:\dpc2\mp_23\DAMENG\dm.ini' FROM BACKUPSET
'E:\dpc1\mp_1\DAMENG\bak\mp_1_back' USE BAK_MAGIC 132256;

RMAN>RECOVER DATABASE 'E:\dpc2\mp_23\DAMENG\dm.ini' FROM BACKUPSET
'E:\dpc1\mp_1\DAMENG\bak\mp_1_back' USE BAK_MAGIC 132256;

RMAN>RECOVER DATABASE 'E:\dpc2\mp_23\DAMENG\dm.ini' UPDATE DPC_MAGIC 123456;

RMAN>RECOVER DATABASE 'E:\dpc2\mp_23\DAMENG\dm.ini' UPDATE DB_MAGIC;
  
```

2) 使用 BP_1 的备份集还原 BP_11、BP_12、BP_13。

其中还原 BP11 的语句如下，还原 BP12，BP13 的语句与还原 BP11 的语句基本一致。（单副本还原到多副本不需要指定 MODE）

```
RMAN>RESTORE DATABASE 'E:\dpc2\bp_11\DAMENG\dm.ini' FROM BACKUPSET
```

DM8 分布计算集群

```
'E:\dpc1\bp_1\DAMENG\bak\bp_1_back' USE BAK_MAGIC 132256;

RMAN>RECOVER DATABASE 'E:\dpc2\bp_11\DAMENG\dm.ini' FROM BACKUPSET

'E:\dpc1\bp_1\DAMENG\bak\bp_1_back' USE BAK_MAGIC 132256;

RMAN>RECOVER DATABASE 'E:\dpc2\bp_11\DAMENG\dm.ini' UPDATE DPC_MAGIC 123456;

RMAN>RECOVER DATABASE 'E:\dpc2\bp_11\DAMENG\dm.ini' UPDATE DB_MAGIC;
```

3) 使用 BP_2 的备份集还原 BP_21、BP_22、BP_23, 这一步与使用 BP_1 的备份集还原 BP_11、BP_12、BP_13 的流程一致, 这里不再赘述。

4) 修改 MP_21 归档文件: 删除 RAFT 归档, 仅保留本地归档。使 MP_21 启动不会进入 MOUNT 状态, 确保 MP_21 启动后可以正常修改注册信息。修改后 MP_21 的 dmarch.ini 文件如下:

```
[ARCHIVE_LOCAL1]

ARCH_TYPE          = LOCAL          #本地归档类型

ARCH_DEST          = e:\dpc_2\mp_21\DAMENG\arch #本地归档文件路径

ARCH_FILE_SIZE     = 128             #本地单个归档文件最大值,单位 MB

ARCH_SPACE_LIMIT   = 0              #本地归档文件总大小, 0 表示无限制
```

5) 启动 MP_21, 使用 DIsql 连接 MP_21, 修改 MP_21 中注册信息。目前还不支持重命名 RAFT 组, 因此仅修改 RAFT 组下节点信息。

```
//1.将实例 MP_1 重命名为 MP_21

SQL>SP_RENAME_DPC_INSTANCE('MP_1','MP_21');

//2.修改 MP_21 的端口信息

SQL>SP_MODIFY_DPC_INSTANCE('MP_21','192.168.1.66','192.168.1.80','9457','8457');

//3.修改 MP_21 的状态信息,以便 MP_21 对应的 RAFT 组中可以加入 MP_22,MP_23

SP_ALTER_DPC_INSTANCE('MP_21','STANDBY',4,1);

//4.将实例 MP_22, MP_23 加入 MP_21 所在的 RAFT 组 (若忘记 MP_21 所在的 RAFT 组, 可通过查询
```

DM8 分布计算集群

```

DPC_INSTANCE 表得知所在的 RAFT 组名)

SP_CREATE_DPC_INSTANCE(NULL,'MP_22','MP',6002,5238,'192.168.1.68', 'STANDBY',1,'MP instance');

SP_CREATE_DPC_INSTANCE(NULL,'MP_23','MP',6003,5239,'192.168.1.68', 'STANDBY',1,'MP instance');

//5.将实例 BP_1 重命名为 BP_11

SQL>SP_RENAME_DPC_INSTANCE('BP_1','BP_11');

//6.修改端口信息

SQL>SP_MODIFY_DPC_INSTANCE('BP_11','192.168.1.68','192.168.1.68',7530,7430);

//7.修改 BP_11 模式状态信息，以便 BP_11 对应 raft 组中可以加入 BP_12,BP_13

SQL>SP_ALTER_DPC_INSTANCE('BP_11','STANDBY',4,0);

//加入 BP_12,BP_13

SQL>SP_CREATE_DPC_INSTANCE('RAFT_1','BP_12','BP',7531,7431,'192.168.1.68', 'standby',0,'BP
instance');

SQL>SP_CREATE_DPC_INSTANCE('RAFT_1','BP_13','BP',7532,7432,'192.168.1.68', 'standby',0,'BP
instance');

//8.将 BP_2 重命名为 BP_21, 并修改端口及模式状态信息，然后在 BP_21 所在的 RAFT 组中加入 BP_22、
BP_23

SQL>SP_RENAME_DPC_INSTANCE('BP_2','BP_21');

SQL>SP_MODIFY_DPC_INSTANCE('BP_21','192.168.1.68','192.168.1.68',7630,7830);

SQL>SP_ALTER_DPC_INSTANCE('BP_21','STANDBY',4,0);

SQL>SP_CREATE_DPC_INSTANCE('RAFT_2','BP_22','BP',7631,7831,'192.168.1.68', 'standby',0,'BP
instance');

SQL>SP_CREATE_DPC_INSTANCE('RAFT_2','BP_23','BP',7632,7832,'192.168.1.68', 'standby',0,'BP
instance');

//9.重命名 SP 并修改端口信息

```

DM8 分布计算集群

```
SQL>SP_RENAME_DPC_INSTANCE('SP1';SP2');
```

```
SQL>SP_MODIFY_DPC_INSTANCE('SP2','192.168.1.68','192.168.1.68',7529,7429);
```

6) 正常退出 MP_21, 脱机备份 MP_21, 还原到 MP_22、MP_23 上, 使在 MP_21 上修改的注册信息同步到 MP_22、MP_23。

```
//备份 MP_21
```

```
dmrman CTLSTMT="BACKUP DATABASE 'e:\dpc_2\mp_21\DAMENG\dm.ini' FULL TO BACKUP_01  
BACKUPSET 'e:\dpc_2\mp_21\BACKUP_01'" USE_AP=2
```

```
//还原到 MP_22
```

```
dmrman CTLSTMT="RESTORE DATABASE 'e:\dpc_2\mp_22\DAMENG\dm.ini' FROM BACKUPSET  
'e:\dpc_2\mp_21\BACKUP_01'" USE_AP=2
```

```
dmrman CTLSTMT="RECOVER DATABASE 'e:\dpc_2\mp_22\DAMENG\dm.ini' FROM BACKUPSET  
'e:\dpc_2\mp_21\BACKUP_01'" USE_AP=2
```

```
dmrman CTLSTMT="RECOVER DATABASE 'e:\dpc_2\mp_22\DAMENG\dm.ini' UPDATE  
DB_MAGIC" USE_AP=2
```

```
//还原到 MP_23
```

```
dmrman CTLSTMT="RESTORE DATABASE 'e:\dpc_2\mp_23\DAMENG\dm.ini' FROM BACKUPSET  
'e:\dpc_2\mp_21\BACKUP_01'" USE_AP=2
```

```
dmrman CTLSTMT="RECOVER DATABASE 'e:\dpc_2\mp_23\DAMENG\dm.ini' FROM BACKUPSET  
'e:\dpc_2\mp_21\BACKUP_01'" USE_AP=2
```

```
dmrman CTLSTMT="RECOVER DATABASE 'e:\dpc_2\mp_23\DAMENG\dm.ini' UPDATE  
DB_MAGIC" USE_AP=2
```

7) 重新为 MP_21 配置好 RAFT 归档, 配置好后启动集群, 单副本还原到多副本动作完成。

```
XMLAL_HB_INTERVAL = 5 #节点通信检测间隔
```

DM8 分布计算集群

| | | |
|--------------------|------------------------------|------------------------|
| RAFT_HB_INTERVAL | = 150 | #选举心跳间隔 |
| RAFT_VOTE_INTERVAL | = 1500 | #选举超时时间，三个库设置不同以尽快选出主库 |
| [ARCHIVE_RAFT1] | | |
| ARCH_TYPE | = RAFT | #RAFT 归档 |
| ARCH_DEST | = mp_22 | #归档目标实例名 |
| [ARCHIVE_RAFT2] | | |
| ARCH_TYPE | = RAFT | #RAFT 归档 |
| ARCH_DEST | = mp_23 | #归档目标实例名 |
| [ARCHIVE_LOCAL1] | | |
| ARCH_TYPE | = LOCAL | #本地归档类型 |
| ARCH_DEST | = e:\dpc_2\mp_21\DAMENG\arch | #本地归档文件路径 |
| ARCH_FILE_SIZE | = 128 | #本地单个归档文件最大值,单位 MB |
| ARCH_SPACE_LIMIT | = 0 | #本地归档文件总大小,0 表示无限制 |

8.8.4.3.1.3 多副本集群还原到单副本集群

多副本集群还原到单副本集群同样确保两套 DMDPC 集群的配置是同构的，即 RAFT 组数是一致的。

将一套多副本架构的 DMDPC 集群还原到另外一套多副本架构 DMDPC 集群。

流程如下：

DPC 集群 1 还原到 DPC 集群 2。

DPC 集群 1 包括 RAFT_MPI (MP_11、 MP_12、 MP_13)、RAFT_BP11(BP_11、BP_12、BP_13)、RAFT_BP12(BP_14、BP_15、BP_16)和 RAFT_SPI(SP1)。

DPC 集群 2 包括 RAFT_MP2 (MP_2)、RAFT_BP21(BP_21)、RAFT_BP22(BP_22)和 RAFT_SP2(SP2)。

DM8 分布计算集群

1) 还原 MP、BP。

为了保证还原后集群的数据一致,一定要确保还原使用的备份集是同一批产生的备份集(可通过 DMRMAN 中 SHOW BACKUPSET 命令查看备份集上的 BAK_MAGIC 字段是否一致,以确认使用的备份集是否是同一批)。另外在执行 UPDATE DB_MAGIC 前,需要更新 DPC_MAGIC,以便和旧集群进行区分。

还原 MP_2 时需要注意的是在 MP 执行 restore 阶段,注意指定 mode 为 normal 模式,以便启动后可以修改注册信息。

```
RMAN>RESTORE DATABASE 'E:\dpc2\mp_2\DAMENG\dm.ini' FROM BACKUPSET
'E:\dpc1\mp_11\DAMENG\bak\mp_11_back' USE BAK_MAGIC 132256 MODE 'NORMAL';
RMAN>RECOVER DATABASE 'E:\dpc2\mp_2\DAMENG\dm.ini' FROM BACKUPSET
'E:\dpc1\mp_11\DAMENG\bak\mp_11_back' USE BAK_MAGIC 132256;
RMAN>RECOVER DATABASE 'E:\dpc2\mp_2\DAMENG\dm.ini' UPDATE DPC_MAGIC 123456;
RMAN>RECOVER DATABASE 'E:\dpc2\mp_2\DAMENG\dm.ini' UPDATE DB_MAGIC;
```

还原 BP_21, BP_22 的语句与还原 MP_2 基本一致,使用 RAFT_BP11 组中节点的备份集还原 BP_21,使用 RAFT_BP12 组中节点备份集还原 BP_22。

此处需要指定 MODE 为 NORMAL。

2) 启动修改 MP_2, 修改 MP_2 上注册信息。

```
//1.将实例 MP_11 重命名为 MP_2
SQL>SP_RENAME_DPC_INSTANCE('MP_11','MP_2');

//2.修改 MP_2 的端口信息
SQL>SP_MODIFY_DPC_INSTANCE('MP_2','192.168.1.68','192.168.1.68', 7630, 7830);

//3.删除 MP_12、MP_13
SQL>SP_DROP_DPC_INSTANCE('MP_12');
SQL>SP_DROP_DPC_INSTANCE('MP_13');
```

DM8 分布计算集群

```
//4.修改 MP_2 模式状态为 NORMAL、OPEN、VALID
SQL>SP_ALTER_DPC_INSTANCE('MP_2','NORMAL',4,1);

//5.使用同样的方式将 BP_11 重命名为 BP_21，删除 BP_12,BP_13，修改 BP_21 模式状态

//6.使用同样的方式将 BP_14 重命名为 BP_22，删除 BP_15、BP_15，修改 BP_22 模式状态

//7.重命名 SPI 为 SP2，修改 SP2 上端口信息
```

3) 重启 MP_2、BP_21、BP_22、SP2，集群还原完成。

8.8.4.3.1.4 多副本集群还原到多副本集群

多副本集群还原到多副本集群同样需要确保两套 DMDPC 集群的配置是同构的，即 RAFT 组数是一致的。

将一套多副本架构的 DMDPC 集群还原到另外一套多副本架构 DMDPC 集群流程如下：

(DMDPC 集群 1 到 DPC 集群 2)。

DPC 集群 1 包括 RAFT_MP1 (MP_11、 MP_12、 MP_13)、RAFT_BP11(BP_11、BP_12、BP_13)、RAFT_BP12(BP_14、BP_15、BP_16)和 RAFT_SPI(SPI)。

DPC 集群 2 包括 RAFT_MP2 (MP_21、MP_22、MP_23)、RAFT_BP21(BP_21、BP_22、BP_23)、RAFT_BP22(BP_24、BP_25、BP_26)和 RAFT_SP2(SP2)。

流程如下：

1) 还原 MP、BP。

为了保证还原后集群的数据一致，一定要确保还原使用的备份集是同一批产生的备份集 (可通过 DMRMAN 中 SHOW BACKUPSET 命令查看备份集上的 BAK_MAGIC 字段是否一致，以确认使用的备份集是否是同一批)。另外在执行 UPDATE DB_MAGIC 前，需要更新 DPC_MAGIC，以便和旧集群进行区分。

还原 MP_21、MP_22、MP_23。

以还原 MP_21 为例：

DM8 分布计算集群

```

RMAN>RESTORE DATABASE 'E:\dpc2\mp_21\DAMENG\dm.ini' FROM BACKUPSET
'E:\dpc1\mp_11\DAMENG\bak\mp_11_back' USE BAK_MAGIC 132256 MODE 'NORMAL';

RMAN>RECOVER DATABASE 'E:\dpc2\mp_21\DAMENG\dm.ini' FROM BACKUPSET
'E:\dpc1\mp_11\DAMENG\bak\mp_11_back' USE BAK_MAGIC 132256;

RMAN>RECOVER DATABASE 'E:\dpc2\mp_21\DAMENG\dm.ini' UPDATE DPC_MAGIC 123456;

RMAN>RECOVER DATABASE 'E:\dpc2\mp_21\DAMENG\dm.ini' UPDATE DB_MAGIC;
    
```

还原 MP_22、MP_23 的流程与 MP_21 基本一致，需要注意的是在 MP 执行 restore 阶段，注意指定 mode 为 normal 模式，以便启动后可以修改注册信息。

还原 BP_21、BP_22、BP_23:

多副本还原到多副本，还原 BP 时无需指定 MODE 字段。

以还原 BP_21 为例:

```

RMAN>RESTORE DATABASE 'E:\dpc2\bp_21\DAMENG\dm.ini' FROM BACKUPSET
'E:\dpc1\bp_11\DAMENG\bak\bp_11_back' USE BAK_MAGIC 132256;

RMAN>RECOVER DATABASE 'E:\dpc2\bp_21\DAMENG\dm.ini' FROM BACKUPSET
'E:\dpc1\bp_11\DAMENG\bak\bp_11_back' USE BAK_MAGIC 132256;

RMAN>RECOVER DATABASE 'E:\dpc2\bp_21\DAMENG\dm.ini' UPDATE DPC_MAGIC 123456;

RMAN>RECOVER DATABASE 'E:\dpc2\bp_21\DAMENG\dm.ini' UPDATE DB_MAGIC;
    
```

还原 BP_22、BP_23 的流程与 BP_21 基本一致。

- 2) 修改 MP_21 的归档文件，删除 RAFT 归档，确保启动后是 OPEN 状态。
- 3) 启动 MP_21，通过 DISQL 连接 MP_21，更新 MP_21 上注册信息。

这里以 MP_21 为例。MP_22、MP_23、BP_21、BP_22、BP_23、BP_24、BP_25、BP_26 和 SP2 的更新注册信息流程与 MP_21 基本一致，均为执行重命名和更新端口信息两步。

```
//将实例 MP_11 重命名为 MP_21
```

DM8 分布计算集群

```
SQL>SP_RENAME_DPC_INSTANCE('MP_11','MP_21');

//更新 MP_21 的端口信息


SQL>SP_MODIFY_DPC_INSTANCE('MP_21','192.168.1.68','192.168.1.68', 7630, 7830);

//下面分别继续执行其他节点的重命名和更新端口号

.....
```


如果多副本 RAFT 内节点进行了改变（扩容或者缩容）。例如：集群 1 中 MP 为 MP1、MP2 和 MP3，集群 2 中 MP 为 MP21、MP22、MP23、MP4、MP5。那么新增或删除的节点请按照下面的情况进行处理。

首先，对于新增或者删除的节点注册信息。

 对于新增的节点只需要使用 SP_CREATE_DPC_INSTANCE 按照实际配置注册进对应的 RAFT 组这一步即可，无需重命名和修改端口信息。

例 增加一个 MP4，只需要执行如下步骤：

```
SP_CREATE_DPC_INSTANCE(NULL, 'MP4', 'MP', 6203, 5439, '192.168.1.118', '', 'STANDBY', 0, 'MP
instance');
```

 对于需要删除的节点也仅需使用 SP_DROP_DPC_INSTANCE 删除对应节点的注册信息这一步即可。

其次，完成更新注册信息后，可在系统表 DPC_INSTANCE 中查询到。注册信息包括实例名和 IP：PORT 信息。注册信息和目标 DPC 环境配置完全一致表示注册成功。

4) 备份 MP_21，还原到 MP_22、MP_23。使在 MP_21 上修改的注册信息同步到 MP_22、MP_23。

5) 重新为 MP_21 配置好 RAFT 归档，归档文件与删除 RAFT 归档前的配置一致即可。

6) 启动 MP (MP_21、MP_22、MP_23)、BP (BP_21、BP_22、BP_23、BP_24、BP_25、BP_26) 和 SP (SP2)。至此，多副本集群还原结束。

8.8.4.3.2 指定归档恢复

本节的还原内容和 [8.8.4.3.1 指定备份集恢复](#) 中的还原内容基本一致。唯一不同的是恢复一致性这一步，采用的是指定归档恢复。

例 指定归档修复的方式完成恢复一致性。

使用归档恢复 MP。

```
RMAN>RECOVER      DATABASE      'E:\dpc\mp\DAMENG\dm.ini'      WITH      ARCHIVEDIR  
'E:\dpc\mp\DAMENG\arch' UNTIL TIME '2021-10-10 10:56:40';
```

使用归档恢复 BP。

```
RMAN>RECOVER      DATABASE      'E:\dpc\bp\DAMENG\dm.ini'      WITH      ARCHIVEDIR  
'E:\dpc\bp\DAMENG\arch' UNTIL TIME '2021-10-10 10:56:40';
```

8.8.4.4 归档还原

归档还原就是将备份集中的归档日志文件重新拷贝到指定归档目录中。通过 DMRMAN 工具可实现对 DMDPC 集群实现归档还原。备份集可以是脱机归档备份集，也可以是联机归档备份集。

对 DMDPC 中节点归档还原方式与单节点的归档还原方式一致。

例 使用 DMRMAN 工具执行归档还原，下面两种方式二选一即可。

指定还原的目标归档日志目录：

```
RMAN> RESTORE ARCHIVE LOG FROM BACKUPSET '/home/dm_bak/arch_all_for_restore' TO  
ARCHIVEDIR'/opt/dmdbms/data/DAMENG_FOR_RESTORE/arch_dest' OVERWRITE 2;
```

指定还原目标库的 dm.ini 文件路径：

```
RMAN> RESTORE ARCHIVE LOG FROM BACKUPSET '/home/dm_bak/arch_all_for_restore' TO  
DATABASE '/opt/dmdbms/data/DAMENG_FOR_RESTORE/dm.ini' OVERWRITE 2;
```

8.9 管理表的读写属性

8.9.1 修改表的属性

DMDPC 支持将数据库表设置为 READ ONLY 或 READ WRITE 属性。READ ONLY 属性修改不支持分区表和 HUGE 表。

语法如下：

```
ALTER TABLE table_name {READ ONLY | READ WRITE};
```

READ ONLY 修改为只读，具备 READ ONLY 属性的表又称为只读表，不支持对只读表修改数据。READ WRITE 修改为可读可写。

将表 T（假定表名为 T）从 READ WRITE 修改为 READ ONLY 的同时，对于以 BS 模式启动的 DMDPC 集群，系统会在集群所有节点上隐式创建一个与该表同构的临时表 T\$ROT（包括二级索引），并在随后的查询中用 T\$ROT 替换表 T 的查询；以 BP 模式启动的集群则仅在用户连接的 SP 节点上创建同构临时表。将表从 READ ONLY 修改为 READ WRITE 的同时，系统会隐式删除同构的临时表（包括二级索引）。

但是在下面这种情况下，DMDPC 系统不会在查询中将表 T 替换为 T\$ROT：如果影响列数据类型的服务器参数值（如参数 DECIMAL_FIX_STORAGE）在修改表 T 为只读时和创建表 T 时不同，那么修改表后服务器并不会将 T 和 T\$ROT 进行关联，从而也不会将查询中的表 T 替换为 T\$ROT。

8.9.2 使用只读表的好处

在分布式环境中，将数据库表的属性修改为只读表时，系统会自动在相关节点上创建一个同构临时表，并将只读表的数据装载进同构临时表。如此以来，只读表中的数据将以同构临时表的形式全部存放在 SP 上。

使用只读表好处有两点：一是将访问各个站点目标表的分布式访问变成对同构临时表的

DM8 分布计算集群

本地访问，减少通讯代价，提高查询效率；二是减少事务涉及的 BP，减少分布式事务提交时的交互。

```
//第一步，准备数据库表。

CREATE TABLE T1(C1 INT);

INSERT INTO T1 VALUES(1);

COMMIT;

//第二步，将数据库表 T1 修改为只读表（READ ONLY 属性）。

ALTER TABLE T1 READ ONLY;

//第三步，查询数据库表。此时使用的查询对象为同构临时表，可提高查询效率。

SELECT * FROM T1;

//第四步，可以观察 EXPLAIN 所展示的表对象名称看出有无替换成临时表 T1$ROT。

EXPLAIN SELECT * FROM T1;
```

8.10 灰度升级与引流

8.10.1 灰度升级

在 DMDPC 使用过程中，可以对 SP 进行灰度升级：在不停止集群服务的情况下，轮流更新升级 SP 的代码版本，升级后 SP 能正常对外服务。升级后可以执行引流，使得在 SP 上的会话达到负载均衡的效果。

8.10.1.1 灰度升级介绍

灰度升级支持的内容介绍如下：

1. 支持用系统函数 `SP_SET_SP_UPGRADE()` 将 SP 标记为升级状态。不能对 BP、MP 标记升级。

DM8 分布计算集群

2. 支持用系统函数 SP_RESET_SP_UPGRADE() 将升级中的 SP 恢复正常。不能对 BP、MP 执行恢复。
3. 升级状态的 SP 不接受新连接，不允许被选为计算节点。
4. 存在 SP 进行升级时，禁用在 SP 上执行的事件触发器，时间触发器以及调度。MP 上执行的触发器不进行禁用。
5. 升级状态的 SP 的会话平滑释放。具体为在配置有服务名的前提下，事务提交或回滚时，客户端会主动断开并尝试切换到其他正常服务的 SP；未配置服务名时，会话仅断开，不切换。长事务或长空闲会话会在服务器检测超时后自动断开，超时时间由 INI 参数 DPC_GUP_SESS_TIMEOUT 进行配置。
6. 升级状态的 SP 在所有会话结束后，SP 服务器自行退出。

SP 灰度升级过程中的注意事项：

1. 升级状态中的 SP，对于已存在的 ESESSION 连接支持其它节点广播的系统函数或 V\$ 视图查询请求，对于新连接则不支持。
2. 升级状态中的 SP，支持 MP 在 DDL 期间广播的字典封锁及计划淘汰等请求。
3. 长事务与空闲会话达到 DPC_GUP_SESS_TIMEOUT 配置的超时值，逐渐退出。
4. 升级状态中 SP 的当前连接数目会话数目前无法查看。
5. 服务器重启前，需要由用户先登录 MP 或可用的 SP 执行系统函数使升级节点恢复正常。

8.10.1.2 灰度升级操作过程

灰度升级过程共分为六步。下面详细介绍：

第一步 配置服务名，可选择跳过该阶段。

第二步 用户对目标 SP 进行标记升级。

例 升级登录任意正常 SP，升级 SPI。

DM8 分布计算集群

```
SP_SET_SP_UPGRADE('SPI');
```

第三步 等待该节点退出后。未配置服务名时，该阶段事务结束时，会话仅断开而不是切换。

第四步 更换执行码。

第五步 手动对该节点进行恢复。

例 登录任意正常状态 SP 或 MP，恢复 SPI。

```
SP_RESET_SP_UPGRADE('SPI');
```

第六步 最后重启节点。

例如重启 SPI。

8.10.2 引流

引流是为了解决 DMDPC 使用过程中（例如灰度升级导致的）SP 上会话数差距较大的问题而提出的一种负载均衡方案。

8.10.2.1 引流介绍

当一个 SP 上会话数较多时，引流会在会话的事务执行提交或回滚操作后，对会话做一次切换，将会话连接到其他会话数低的 SP 节点上，从而达到整体的负载均衡效果。

引流支持的内容介绍如下：

1. 支持在任意 SP 上执行系统函数 `SP_DPC_REBANLANCE_SESSION(1)`，收集当前可见的所有 SP 的会话数，并产生所有节点各自的平衡方案。

2. 支持在任意 SP 上执行系统函数 `SP_DPC_REBANLANCE_SESSION(0)`，收集当前可见的所有 SP 的会话数，并对所有 SP 节点上的平衡方案清空，清空时关闭引流。

SP 引流的注意事项。下面正在执行灰度升级的 SPI 为例进行说明：

1. 在 SPI 上灰度升级时，SPI 无法执行引流；若已正在引流，引流也会暂停执行。

DM8 分布计算集群

2. 在 SP1 上灰度升级时，其他 SP 执行引流时 SP1 对他们是不可见的，若其他已正在向 SP1 切换，此次切换跳过。

3. 在进行中的引流任务未结束时，不能再次引流覆盖未完成的平衡方案，需要先清除，因此 SP_DPC_REBANLANCE_SESSION(1)不能连续执行。

4. 引流的清除操作 SP_DPC_REBANLANCE_SESSION(0)可以连续执行。

8.10.2.2 引流开启与关闭

引流过程共分为两步。下面详细介绍：

一 引流开启。

例 在任意 SP 上执行系统函数，收集当前可见的所有 SP 的会话数，并产生平衡方案，开始引流。

```
SP_DPC_REBANLANCE_SESSION(1);
```

引流开启后，当 SP 上会话较多时，每当会话的事务提交或回滚，会话均会自动进行一次切换，切换到其他相对空闲的 SP 站点上，直到负载均衡。例如：DMDPC 系统共三个 SP，SP1 有 15 个会话，SP2 有 0 个会话，SP3 有 0 个会话。开启引流之后，当 SP1 的第 1-5 个会话的事务结束，系统会将会话切换给 SP2，第 6-10 个会话的事务结束后，系统会将会话切换给 SP3，直到负载均衡。

二 引流关闭。

例 在任意 SP 上执行系统函数，收集当前可见的所有 SP 的会话数，并清除平衡方案，结束引流。

```
SP_DPC_REBANLANCE_SESSION(0);
```

8.11 多副本手动切换主备

目前多副本集群是通过自动选举的方式选出主备的，为了方便集群使用，同时提供了手

DM8 分布计算集群

动切换主库的方式。

备库在执行切换前,会对自身是否具有切换条件进行检查,下面列出了不具备切换的情况:

1. 不是多副本环境,不允许切换。
2. 备库选举角色不为 FOLLOWER,不允许切换。
3. 备库模式不为 STANDBY 或状态不为 OPEN,不允许切换。
4. 不存在有效的主库,不允许切换。
5. 备库自身归档无效,不允许切换。
6. 系统前一次节点变更还未完成(比如正在执行增删节点,尚未完成),不允许切换。

手动切换的前提条件和可能遇到问题的解决办法:

1. 所有操作是直连 MP 或 BP 节点执行切换。其中,对于 BS 主库节点需要 LOCAL 方式登录执行。

2. 执行手动切换需要严格按照执行流程执行。

3. 若执行 SP_RAFT_SUSPEND_THREAD 时失败,可由用户根据返回值决定是否重试。

4. 若执行 SP_RAFT_SWITCHOVER 执行失败,可登录任意一个备库,执行 SP_RAFT_RESUME_THREAD 唤醒主库挂起的工作线程。若所有备库都故障,可以等到有活动备库再执行此操作,或者将主库重启。

```
SP_RAFT_RESUME_THREAD('BP_1'); //唤醒 BP_1 工作线程
```

手动切换流程如下:

1.Disql 登录主库(对于 BS 主库节点需要通过 LOCAL 方式登录),挂起主库工作线程。

```
SP_RAFT_SUSPEND_THREAD();
```

2.Disql 登录备库,执行 SP_RAFT_SWITCHOVER(),将备库切换为主库

```
SP_RAFT_SWITCHOVER();
```

8.12 SP 缓存表行数

提供 SP 可在本地缓存表行数的功能。开启本功能后，SP 每次执行 SQL 语句时，可先从本地缓存中获取表行数，本地没有再向 MP 请求表行数，可大大降低 MP 的性能瓶颈。关闭本功能，每次执行 SQL 语句时均需向 MP 请求表行数。

使用 INI 参数 `STAT_CACHE_FLAG=1`，开启 SP 在本地缓存表行数的功能。此参数值在各站点可以不一致。`STAT_CACHE_FLAG=1` 需和 `STAT_CACHE_CAPACITY` 搭配使用，由 `STAT_CACHE_CAPACITY` 设置 SP 缓存表行数的缓存容量。

如果需要清理本地缓存的表行数，可使用 `SP_CLEAR_TAB_ROW_CNT_CACHE` (站点号, 表 id)，清除所有或指定节点上的全部或部分缓存。

使用 `V$TABLE_ROW_CNT_CACHE` 可查看各节点中的缓存情况, 包含表 id, 访问次数, 表行数。

8.13 本地登录

本地登录 (即登录时设置 `mpp_type=local`) 是一种仅允许访问自身节点与 MP 节点的一种登录模式。只要自身节点存活且存在可用 MP，就允许本地登录。本地登录仅支持访问和修改本地数据，不支持访问和修改其他节点数据。

8.13.1 DDL 操作

本地模式登录 BP/MP 时，仅允许对自身节点进行库级备份这一种 DDL 操作。SP 不支持任何 DDL。以本地模式登录，执行备份，只会备份本节点。该情况下备份集中 `BAK_MAGIC=0`，数据还原和数据恢复时不需要指定 `BAK_MAGIC` 即还原成功。

例 本地登录直连 BP 备份本节点，备份集中 `BAK_MAGIC=0`，数据还原和数据恢复时不需要指定 `BAK_MAGIC`。

```
./disql SYSDBA/SYSDBA@192.168.100.121:5273#"{mpp_type=local}"
```

DM8 分布计算集群

```
disql V8

SQL> BACKUP DATABASE;

操作已执行

RMAN> show backupset

'/ssd/test/BP3_D/DAMENG/bak/DB_DAMENG_FULL_20220707_171446_555778';

bak_magic:          0

RMAN>  RESTORE  DATABASE  '/ssd/test/BP1_C/DAMENG/dm.ini'  FROM  BACKUPSET

'/ssd/test/BP3_D/DAMENG/bak/DB_DAMENG_FULL_20220707_171446_555778';

RMAN>  RECOVER  DATABASE  '/ssd/test/BP1_C/DAMENG/dm.ini'  FROM  BACKUPSET

'/ssd/test/BP3_D/DAMENG/bak/DB_DAMENG_FULL_20220707_171446_555778';

RMAN> RECOVER DATABASE '/ssd/test/BP1_C/DAMENG/dm.ini' UPDATE DB_MAGIC;
```

8.13.2 DML 操作

本地登录支持 DML 操作访问或修改本地数据。不支持跨节点操作。例如：不支持数据更新操作中，旧值和新值位于不同的节点。

以本地模式登录 MP、BP 和 SP 主库时，允许执行的 DML 操作有增加、删除、修改、查询、提交、回滚。查询、插入、更新、删除操作均支持带有子查询。备库仅支持提交和回滚。

不同表对象的查询，插入，修改、删除在不同节点上的限制各不相同。下面分别介绍：

8.13.2.1 用户表

本小节的用户表是指除了临时表、DUAL 表、V\$ 动态视图、系统表之外的表。

MP、SP 不支持操作用户表。

BP 支持操作非分区表，但是表和表所属表空间均需位于本节点上。BP 支持操作分区表，

但是仅支持操作本节点上的数据。例如：查询其他节点数据将返回空集。

8.13.2.2 动态视图和系统表

以本地模式登录 MP、BP 或 SP，支持查询自身 V\$ 视图，但视图中无法获得其他节点的 V\$ 动态视图信息。

MP 支持查询系统表，BP 和 SP 不支持。

8.13.2.4 临时表

MP、BP、SP 均支持对临时表 DML 操作（查询，插入，修改、删除、提交、回滚）。

8.13.3 系统过程或函数处理

系统过程或函数处理。在本地登录 MP、BP 或 SP 时：

1. SP 上执行的函数：仅在 MP 与本 SP 节点执行的函数将被允许，否则将报错。
2. 只直连 MP 执行的函数：支持重建系统视图函数 SP_CREATE_SYSTEM_VIEWS、

通信参数 SP_SET_DPC_NET_CONF、以及动态增删节点等相关函数。具体支持的函数如下：

SP_CREATE_SYSTEM_VIEWS

SP_SET_DPC_NET_CONF

SP_CREATE_DPC_系列

SP_DROP_DPC_系列

SP_MODIFY_DPC_系列

SP_BP_GROUP_系列

3. 可直连 MP 或 BP 执行的函数，在本地登录时，均允许执行。具体支持的函数如下：

SP_RAFT_SUSPEND_THREAD

```
SP_RAFT_SWITCHOVER  
  
SP_RAFT_RESUME_THREAD  
  
SP_ADD_RAFT_LEARNER  
  
SP_ALTER_RAFT_NODE  
  
SP_ADD_RAFT_NODE  
  
SP_DELETE_RAFT_NODE  
  
SP_REPLACE_RAFT_NODE  
  
SP_DELETE_RAFT_LEARNER  
  
CHECKPOINT  
  
SP_SET_PARA_VALUE
```

8.14 容错域的使用

容错域可以添加 MP 或 BP，禁止添加 SP。RAFT 协议中的 4 种角色，以及影子节点也可以正常加入。同个容错域可以同时包含 MP、BP。容错域的 DOMAIN_ID 为主键，且为当前最小空闲 id；容错域的名称 name 唯一；DESCRIPTION 不唯一。

每个容错域内包含 0-N 个不同 RAFT 组内的节点，同一个 RAFT 组内的不同节点不能包含在同一个域内。因此每个容错域的容纳个数和 RAFT 个数有关：每个容错域能容纳 4096 个节点。在容错域新增实例、以及 MP 主实例启动时，会检查容错域是否有重复 raft 组。

不同容错域不能位于同一个物理区域。

例 1 新增系统表 DPC_FAULT_DOMAIN。新建容错域 FDOM_1。

```
SP_CREATE_FAULT_DOMAIN('FDOM_1','shanghai');
```

例 2 将 MPI, BP1 实例移入容错域，不论 MPI,BP1 现在属于哪个容错域。

```
SP_FAULT_DOMAIN_MV_INST('FDOM_1','MPI');
```

```
SP_FAULT_DOMAIN_MV_INST('FDOM_1','BP1');
```

DM8 分布计算集群

例 3 查看实例与容错域对应关系。

```
SELECT RAFT_ID, NAME, FAULT_DOMAIN from V$DPC_EDCT_INSTANCE;
```

例 4 将 BP1 移出容错域 FDOM_1, 并使其不属于任何一个容错域。

```
SP_FAULT_DOMAIN_MV_INST(NULL, 'BP1');
```

例 5 删除容错域 FDOM_1, 并移出 FDOM_1 内所有的实例, 使它们不属于任何容错域。

```
SP_DROP_FAULT_DOMAIN('FDOM_1', 1);
```

9 快速装载工具

达梦快速装载工具 dmfldr (DM Fast Loader) 是 DM 提供的快速数据装载命令行工具。

用户通过使用 dmfldr 工具能够把按照一定格式排序的文本数据以简单、快速、高效的方式载入到 DM 数据库中，或把 DM 数据库中的数据按照一定格式写入文本文件。

达梦快速装载工具具备向多个子表并发同时导入的特性，装载效率很高。

当用户使用 dmfldr 向 DMDPC 系统装载数据时，需要指定 dmfldr 的连接串为连接 SP 服务器的连接串。由于 SP 并不存放用户关系表真实数据，数据实际存储于 BP 站点，因此 dmfldr 实际发送数据是需要发往 BP 站点。然而 SP 服务器并不知晓装载表存储在哪些 BP 站点，表的 BP 站点信息由 MP 服务器管理，因此 SP 需要向 MP 请求查询装载表的 BP 信息，然后将 BP 信息转发给 dmfldr 客户端，dmfldr 客户端解析消息后可获取到装载表的 BP 信息，直接连接到各个 BP 站点服务器，将数据发往各个 BP 站点。

dmfldr 在 DMDPC 环境下的装载流程为：

1. 用户提出装载需求，使用 dmfldr 向 DMDPC 系统装载数据；
2. dmfldr 向 SP 发送请求，请求获取装载表的 BP 服务器站点信息；
3. SP 向 MP 发送请求，请求获取装载表的 BP 服务器站点信息；
4. MP 根据 SP 的请求，在 MP 站点上查询系统表，收集待装载表所涉及的所有 BP 站点信息，并将这些 BP 站点信息全部返回给 SP；
5. SP 将 BP 站点信息返回给 dmfldr；
6. dmfldr 得到表的 BP 站点信息后，首先与各个 BP 分别建立连接，然后将所有数据发往各自 BP 站点，再分别进行装载。

在 DMDPC 架构中，dmfldr 装载流程如下图所示。

DM8 分布计算集群

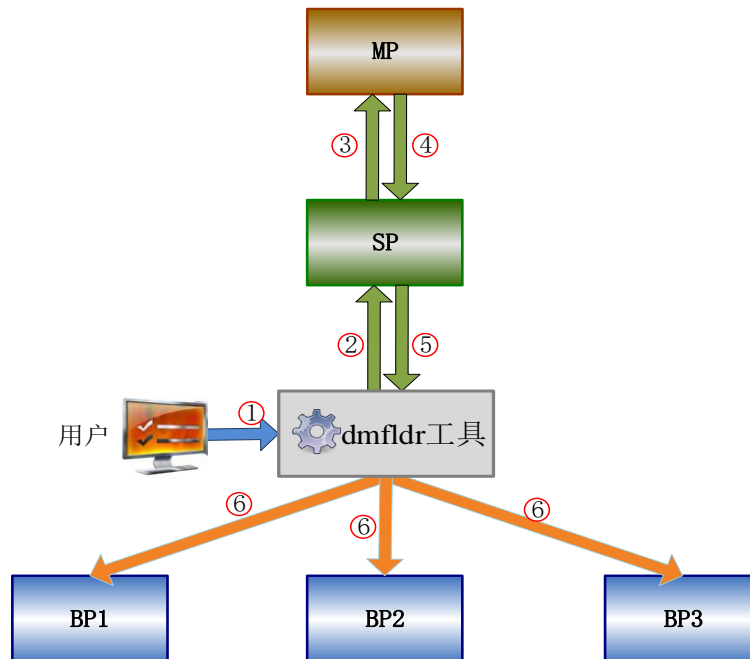


图 9.1 dmfldr 装载框架示意图

例 在 Windows 环境下，向 SP 在 192.168.0.40 的数据库进行数据装载。

```
dmfldr.exe SYSDBA/SYSDBA@192.168.0.40:5238 control='d:\tl.ctf'
```

关于 dmfldr 工具的具体使用方法和控制文件 control 的详细介绍请参考手册《DM8_dmfldr 使用手册》。

10 查询分析

在执行形态和执行调度上，DMDPC 和 DM MPP 相比差别很大。DMDPC 执行计划中增加了子计划的概念。为此 DMDPC 新增了 V\$DPC_STASK_THRD 动态视图，来展示每个子任务、每个线程的执行统计信息，便于用户对性能进行分析。

10.1 查看子计划

子计划是使用一对 ESEND/ERECV 操作符来划分的。整个执行计划可通过 EXPLAIN 获取，子计划为整体计划中的一部分。

举例：

```
create table M1(c1 int not null, c2 char(1000)) partition by hash(c1) (
partition p1 storage (on TS_01),
partition p2 storage (on TS_02),
partition p3 storage (on TS_01),
partition p4 storage (on TS_02),
partition p5 storage (on TS_01),
partition p6 storage (on TS_02)
);
insert into M1 select level, 'aaa' || mod(level, 10) || repeat('x', 20) from dual connect by level <=
50;
commit;
```

通过 EXPLAIN 命令查看 SQL 语句的整体执行计划，其中就使用到了子计划。

```
Explain
select /*+ use_hash(m1, sysobjects) xparallel(#1 8) parallel(sysobjects 8) stat(m1 IM) */ co
```

DM8 分布计算集群

```

unt(*) from m1 where c1 in (select mod(id, 10000) from sysobjects);

1  #NSET2: [170, 1, 4]

2  #PRJT2: [170, 1, 4]; exp_num(1), is_atom(FALSE)

3  #AAGR2: [170, 1, 4]; grp_num(0), sfun_num(1) slave_empty(0)

4  #ERECV: [170, 1, 4]; stask_no(-1), l_stask_no(1), n_key(0), recv_in_turn(0)

5  #ESEND: [170, 1, 4]; stask_no(1), type(DIRECT), sites(1:8,2:8), sql_invoke(0),
pwj_opt(0), table(-)

6  #AAGR2: [170, 1, 4]; grp_num(0), sfun_num(1) slave_empty(0)

7  #HASH RIGHT SEMI JOIN2: [170, 312, 4]; n_keys(1)

KEY(DMTEMPVIEW_16778238.colname=cast(M1.C1 as BIGINT)) KEY_NULL_EQU(0);

INFO_BITS(0x1)

8  #ERECV: [1, 831, 4]; stask_no(1), l_stask_no(0), n_key(0), recv_in_turn(0)

9  #ESEND: [1, 831, 4]; stask_no(0), type(BROADCAST), sites(0:8),
sql_invoke(0), pwj_opt(0), table(-)

10 #GI: [1, 831, 4]; policy(RANDOM), gi_unit[0..0]

11 #PRJT2: [1, 831, 4]; exp_num(1), is_atom(FALSE)

12 #SSCN: [1, 831, 4]; SYSINDEXIDSYSOBJECTS(SYSOBJECTS as
SYSOBJECTS)

13 #GI: [105, 1000000, 4]; policy(RANDOM), gi_unit[0..0]

14 #CSCN2: [105, 1000000, 4]; INDEX33562504(M1)
  
```

执行 SQL 语句，得到结果：

```

SQL>select /*+ use_hash(m1, sysobjects) xparallel(#1 8) parallel(sysobjects 8) stat(m1 1M)
*/ count(*) from m1 where c1 in (select mod(id, 20)+10 from sysobjects);
  
```

DM8 分布计算集群

//查询结果如下:

```
行号      COUNT(*)
```

```
1         20
```

已用时间: 139.984(毫秒). 执行号:16777624.

10.2 查看物理计划生成阶段的计划

DM 提供一个展示物理计划生成阶段的计划（包含子计划拆分细节）的存储过程。

定义:

```
sp_set_dbg_show (
    mode_name      varchar(256),
    v int
)
```

功能说明:

开启或者关闭物理计划生成阶段（PHD）的计划展示。

参数说明:

mode_name: 阶段的名称，取值 PHD。PHD 表示物理计划生成阶段；

v: 开启或者关闭。1: 开启； 0: 关闭。

举例说明:

使用 `sp_set_dbg_show('PHD',1)` 开启 PHD 阶段计划展示。服务器控制台上子任务独

立展示如下:

```
====> 3 stasks, RT <====
```

```
>>> #0: data_source: TAB, n_recv:0, SINGLE PARTITION, sites_source:SELF, parent stask: #1
```

DM8 分布计算集群

```

11 #SEND: [1, 4616, 4]; stask_no(0), type(BROADCAST), n_key(0)

12 #GI: [1, 4616, 4]; policy(RANDOM), gi_unit[0..0]

13 #PRJT2: [1, 4616, 4]; exp_num(1), is_atom(FALSE)

14 #SSCN: [1, 4616, 4]; SYSINDEXIDSYSOBJECTS(SYSOBJECTS)

MP(id=0): workers 8, scan operator 1

>>>scan operator#0: partition IDs[0:0], no DPP

>>>> #1: children[#0], data_source: HP_TAB|RECV, n_recv:1, MULTI SITES, sites_source:SELF,
parent stask: #2

7 #SEND: [171, 1, 4]; stask_no(1), type(DIRECT), n_key(0)

8 #AAGR2: [171, 1, 4]; grp_num(0), sfun_num(1)

9 #HASH RIGHT SEMI JOIN2: [171, 4808, 4]; key_num(1)

10 #RECV: [1, 4616, 4]; stask_no(1), l_stask_no(0), n_key(0), nth_recv(0),
recv_in_turn(0)

15 #GI: [105, 1000000, 4]; policy(RANDOM), gi_unit[0..0]

16 #CSCN2: [105, 1000000, 4]; INDEX33559033(MI)

BP2(id=3): workers 8, scan operator 1

>>>scan operator#0: partition IDs[0:4578, 2:4580, 4:4582], no DPP

BPI(id=2): workers 8, scan operator 1

>>>scan operator#0: partition IDs[1:4579, 3:4581, 5:4583], no DPP

>>>> #2: children[#1], data_source: RECV, n_recv:1, PQ forbidden, SINGLE SITE,
sites_source:C_BP, ROOT task

3 #SEND: [171, 1, 4]; stask_no(2), type(DIRECT), n_key(0)

```

DM8 分布计算集群

```

4    #PRJT2: [171, 1, 4]; exp_num(1), is_atom(FALSE)
5    #AAGR2: [171, 1, 4]; grp_num(0), sfun_num(1)
6    #RECV: [171, 1, 4]; stask_no(2), l_stask_no(1), n_key(0), nth_rcv(0),
rcv_in_turn(0)

BP2(id=3): workers 1, scan operator 0
    
```

这种展示方案更贴近于 SQL 实际调度，包括子任务依赖关系，调度的站点和并行度信息一目了然。

也可以通过 DEM 或者 MANAGER 工具进行图形化的计划展示。下面以 DEM 为例介绍。

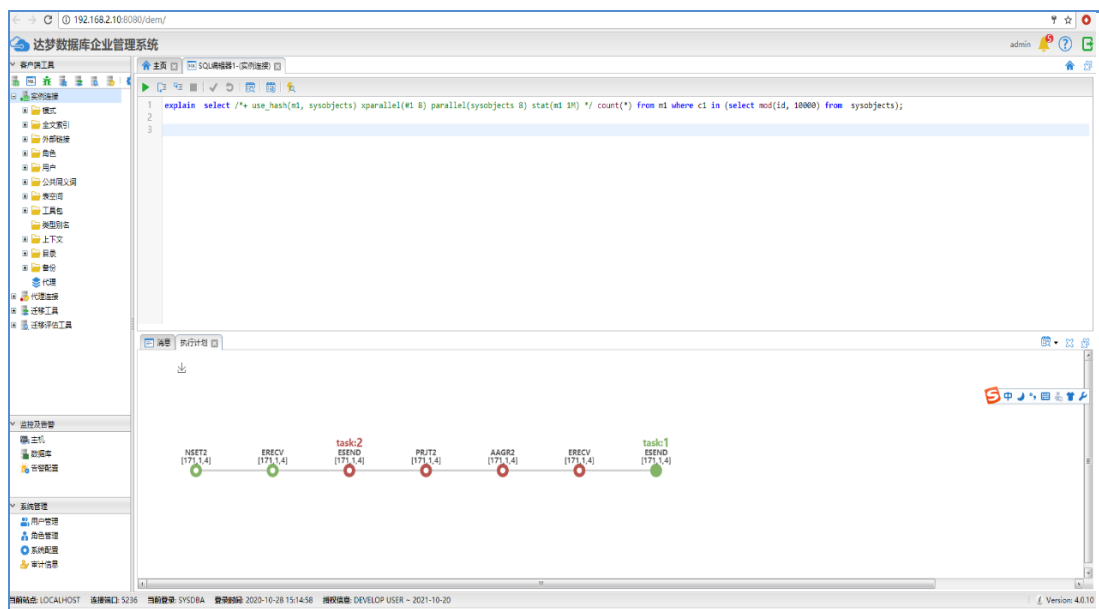


图 10.1 通过 DEM 查看执行计划

在 DEM 的计划展示中可以点击任意操作符图标进行展开或收缩，方便对局部计划细节进行关注。例如：点击 task1 的节点，所展示的计划如下：

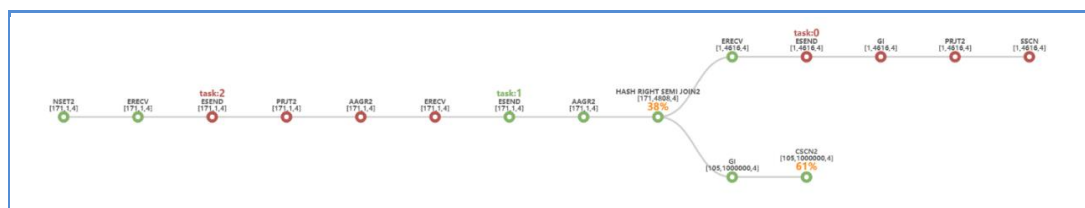


图 10.2 展开某个操作符

10.3 查看正在执行的语句信息

使用之前需开启监控。监控开启方法是设置 INI 参数 ENABLE_MONITOR 大于 0。

1. 查看监控是否开启。

```
select SF_GET_PARA_STRING_VALUE(1, 'INSTANCE_NAME') as ins_name,* from v$dm_ini
where para_name = 'ENABLE_MONITOR';
```

如果未开启，可以执行下列语句动态更改。例如：设置 ENABLE_MONITOR=1。

```
alter system set 'ENABLE_MONITOR'=1 MEMORY;
```

2. 获取正在执行语句的执行号。

- 1) 通过 V\$SESSIONS 找到执行的会话 ID 等信息；

2) 根据上一步会话 ID 信息找到对应的 QC、执行号。如果返回多个 QC 记录，可以加入 is_over = 'N' 排除已完成 SQL 或者根据执行号随时间递增的特性来筛选得到正确的执行号。

例 以某个执行时间耗时较长的语句 Q1 为例。

```
//Q1
select /*+ parallel(8) */ count(*) from (select * from j2 x union all select id, pid, id, pid from
sys.sysobjects x) t1 join j2 t2 on t1.d2 = t2.d2;
```

用下列查询获取 Q1 执行号。

```
with v_sql_1 as (select sess_id, sess_seq from v$sessions
where SQL_TEXT like 'select /*+ parallel(8) */ count(*) from %')
select x.* from v$dpc_qc_history x,v_sql_1 y where x.sess_id = y.sess_id and x.sess_seq
= y.sess_seq;
```

返回的部分结果：

| EXEC_ID | MPP_EXEC_ID | SQC_EXEC_ID | N_STASK | IDU_FLAG | N_SITES | N_HTAB |
|---------|-------------|-------------|---------|----------|---------|--------|
|---------|-------------|-------------|---------|----------|---------|--------|

DM8 分布计算集群

| N_SPL | N_DPP | IS_OVER | | | | | |
|----------|----------|---------|---|---|---|---|--|
| 12582912 | 13313984 | 1 | 2 | - | 3 | 0 | |
| 0 | 0 | N | | | | | |

3. 根据找到的执行号 12582912, 在 V\$DPC_STASK_THRD 视图中获取执行的动态信息, 观察某个子任务发送、存储行数的变化。

```
select * from v$dpc_stask_thrd where exec_id = 12582912;
```

其中, V\$DPC_QC_HISTORY 和 V\$DPC_STASK_THRD 是在内存中维持特定数量的信息。某些陈旧的已完成记录或者高并发压力下尚未执行完成的记录都存在被淘汰覆盖的可能。

11 SQL 调优

DM 查询优化器采用基于代价的方法。在估计代价时，主要以统计信息或者数据分布为依据。通常情况下，估计的代价都是准确的。但在一些估算代价的依据极度缺乏的情况下，例如：缺少统计信息、或统计信息陈旧、或抽样数据不能很好地反映数据分布时，优化器选择的执行计划不是最优的，甚至可能是很差的执行计划。

然而 DBA 对于数据分布是很清楚的，并且知道 SQL 语句按照哪种方法执行会最快。在面临上述优化器选择的执行计划不是最优的情况下，DBA 可以主动进行人工干预，指示优化器按照指定的方法去选择 SQL 的执行计划。

这种人工干预优化器的方法称为 HINT。优化器可根据 DBA 的 HINT 提示来生成指定的执行计划。如果优化器无法根据给定的 HINT 生成相应的执行计划，那么将忽略该 HINT。

HINT 的常见格式如下所示：

```
SELECT /*+ HINT1 [HINT2]*/ 列名 FROM 表名 WHERE _CLAUSE ;  
  
UPDATE 表名 /*+ HINT1 [HINT2]*/ SET 列名 =变量 WHERE _CLAUSE ;  
  
DELETE FROM 表名 /*+ HINT1 [HINT2]*/ WHERE _CLAUSE ;
```

需要注意的是：如果 HINT 的语法没有写对或指定的值不正确，DM 并不会报错，而是直接忽略 HINT 继续执行。

11.1 DMDPC 数据分发方式提示

DMDPC 环境下提供了一种对指定的连接、分组、排序、去重操作符数据分发方式进行人工干预的优化器提示。该优化器提示被采纳的前提是指定的分发路径有效，因为代价原因没有被优化器选中。

语法：

```
/*+DPC(分发方式探测序号 分发方式字符串)*/
```

DM8 分布计算集群

其中分发方式探测序号可以在 10053 trace event 生成的 TRACE 文件中查看到，以 nth_try 标示；分发方式请参考 [11.1.1 二元操作符](#)和 [11.1.2 一元操作符](#)。

例 在 SQL 语句中使用 DMDPC 数据分发方式提示。

```
select /*+ DPC(1 x_merge) */ para_name from v$dm_ini order by para_name;
```

11.1.1 二元操作符

对于连接和集合运算二元操作符，可供选择的分发方式字符串参见表 11.1。

表 11.1 二元操作符的分发方式字符串

| 分发方式字符串 | 含义 |
|---------------|---|
| L_NO_R_NO | 二元操作符两侧都不添加任何数据分发操作符 |
| L_BRO | 二元操作符左侧增加广播操作 |
| R_BRO | 二元操作符右侧增加广播操作 |
| L_DIS_R_DIS | 分别对二元操作符两侧数据进行分发 |
| L_DIS | 对二元操作符左侧数据进行分发 |
| R_DIS | 对二元操作符右侧数据进行分发 |
| L_GAT_R_GAT | 二元操作符两侧数据汇总到一个线程 |
| ONE_GAT_ONLY | 二元操作符一侧为指令，另一侧为非指令。非指令的一侧将数据发送到一个 SP 站点执行 |
| ONE_PART_ONLY | 二元操作符一侧为普通表，另一侧将数据发送到此普通表所在的站点执行 |

例 下面一段 trace 片段展示了采用优化器提示中的二元操作符分发方式的情况。

```
//连接片段，正在试探第 5 个需要分发的操作符 hash left semi join(anti)。
*** probe best distribute method for hash left semi join(anti) (000000005089E500), DPC
motion nth_try[5], restricts[0x12], self cost 1.000000
```

DM8 分布计算集群

```

left  motion cost: broadcast = 0.000011, dis = 0.000011, gather = 0.000011

right motion cost: broadcast = 0.000023, dis = 0.000011, gather = 0.000011

> try      R_BRO(404), cost 0.062523, n_parallel 2, n_sites 0, best*

> try L_DIS_R_DIS(405), cost 0.062523, n_parallel 2, n_sites 0, best*

> try L_GAT_R_GAT(408), cost 1.000023, n_parallel 1, n_sites 0
    
```

11.1.2 一元操作符

对于分组 (group)、排序 (order)、去重 (distinct) 和分析函数 (analytic function) 一元操作符，可供选择的分发方式字符串参见表 11.2。

下表中的 X 代表 group、order、distinct 或 analytic function。例如：当按照 group 进行分发时，x 即表示 group。

表 11.2 一元操作符的分发方式字符串

| 分发方式字符串 | 含义 |
|-------------|---|
| X | 直接进行 X |
| X_DIS_X | 本地先 X，在数据分发后进行二次 X |
| DIS_X | 数据分发后进行 X |
| DIS_SORT_X | 数据分发后先进行排序，接下来执行 X |
| X_GAT_X | 本地先 X，数据汇总到一个线程后进行二次 X |
| GAT_X | 数据汇总到一个线程后进行 X |
| GAT_SORT_X | 数据汇总到一个线程后排序，接下来执行 X |
| X_MERGE | 每个线程先分别排序，汇总后由最终的线程进行归并排序 |
| DIS_X_GAT_X | 对无分组项且含有 DISTINCT 的集函数，先按集函数参数进行数据分发，然后再对分发的数据进行 X_GAT_X。仅当 INI 参数 GROUP_OPT_FLAG 取值包 |

DM8 分布计算集群

| | |
|------------------|---|
| | 含 64 时考虑该方式 |
| DIST_DIS_X_GAT_X | 和 DIS_X_GAT_X 类似。区别为，在对集函数参数进行数据分发前先做一次去重处理。仅当 INI 参数 GROUP_OPT_FLAG 取值包含 64 时考虑该方式 |

例 下面一段 trace 展示了采用优化器提示中的二元操作符分发方式的情况。

```
//分组 group 片段，正要探测第 2 个需要分发的 group。
*** probe best distribute method for group(000000000070EIF8), DPC motion nth_try[2],
restricts[0xd], self cost 1.000000

left motion cost: broadcast = 236.980132, dis = 236.980132, gather = 236.980132

self motion cost: broadcast = 0.000006, dis = 0.000006, gather = 0.000006

> try          x(420), cost 1.100000, n_parallel 0, n_sites 0, best*
> try    X_GAT_X(424), cost 3.000006, n_parallel 0, n_sites 0
> try    GAT_X(425), cost 239.980132, n_parallel 0, n_sites 0

//排序 order 片段，正要探测第 7 个需要分发的 order。
*** probe best distribute method for order(0000000050897128), DPC motion nth_try[7],
restricts[0x45], self cost 1.000000

left motion cost: broadcast = 0.000011, dis = 0.000011, gather = 0.000011

self motion cost: broadcast = 0.000011, dis = 0.000011, gather = 0.000011

> try          x(420), cost 1.100000, n_parallel 0, n_sites 0, best*
> try    GAT_X(425), cost 3.000011, n_parallel 0, n_sites 0
> try    X_MERGE(427), cost 12.000011, n_parallel 0, n_sites 0

//排序去重 distinct 片段，正要探测第 1 个需要分发的 distinct。
```

DM8 分布计算集群

```

*** probe best distribute method for distinct(00000000007112E8), DPC motion nth_try[1],
restricts[0x1f], self cost 21.828696

    left  motion cost: broadcast = 3.638299, dis = 1.819149, gather = 1.819149

    self  motion cost: broadcast = 0.036383, dis = 0.018191, gather = 0.018191

> try          X(420), not available

> try    X_DIS_X(421), cost 114.344539, n_parallel 1, n_sites 1, best*

> try    DIS_X(422), cost 2464.784487, n_parallel 1, n_sites 1

> try    X_GAT_X(424), cost 14.150826, n_parallel 0, n_sites 0, best*

> try    GAT_X(425), cost 26.647845, n_parallel 0, n_sites 0

//分析函数 analytic function 片段，正要探测第 6 个需要分发的 analytic function。

*** probe best distribute method for analytic function(0000000050899AC0), DPC motion
nth_try[6], restricts[0x181], self cost 1.000000

    left  motion cost: broadcast = 0.000011, dis = 0.000011, gather = 0.000011

    self  motion cost: broadcast = 0.000011, dis = 0.000011, gather = 0.000011

> try          X(420), not available

> try    DIS_SORT_X(423), cost 20.000112, n_parallel 1, n_sites 1, best*

> try    GAT_SORT_X(426), cost 4.000011, n_parallel 0, n_sites 0
  
```

11.2 HINT 实例

对上述部分 HINT 的用法和效果进行举例说明。

11.2.1 数据准备

数据准备。

DM8 分布计算集群

```
create table t1(c1 int, c2 int) partition by hash(c1) partitions 3;
create table t2(d1 int, d2 int) partition by hash(d1) partitions 4;
```

表 t1, t2 分区方式不同, 分别分布于 raft1, raft2 两个实例上。

11.2.2 带条件查询

本节优化的对象为带 WHERE 条件的查询语句。

例 1 采用对左侧数据进行广播方式通常适用连接两边数据量相差较大的场合。

```
explain select /*+ dpc(t1_bro) */ * from t1, t2 where c1 = d1;
```

结果如下:

```
1  #NSET2: [1, 1, 16]
2  #ERECV: [1, 1, 16]; stask_no(-1), l_stask_no(1), n_key(0), recv_in_turn(0)
3  #ESEND: [1, 1, 16]; stask_no(1), type(DIRECT), sites(2:1,1:1), sql_invoke(0), pwj_opt(0),
table(-)
4  #PRJT2: [1, 1, 16]; exp_num(4), is_atom(FALSE)
5  #HASH2 INNER JOIN: [1, 1, 16]; KEY_NUM(1); KEY(T1.C1=T2.D1) KEY_NULL_EQU(0)
6  #ERECV: [1, 1, 8]; stask_no(1), l_stask_no(0), n_key(0), recv_in_turn(0)
7  #ESEND: [1, 1, 8]; stask_no(0), type(BROADCAST), sites(2:1,1:1), sql_invoke(0),
pwj_opt(0), table(-); INFO_BITS(0x4)
8  #GI: [1, 1, 8]; policy(RANDOM), gi_unit[0..0], scan_type[0](FULL)
9  #CSCN2: [1, 1, 8]; INDEX33651668(T1)
10 #GI: [1, 1, 8]; policy(RANDOM), gi_unit[0..0], scan_type[0](FULL)
11 #CSCN2: [1, 1, 8]; INDEX33651672(T2)
```

例 2 哈希连接一般选择小表一次建立哈希表, 对右侧进行广播发送不太常见, 但右侧广播可以保留左侧的分布特性, 某些场合下有用。

DM8 分布计算集群

```
explain select /*+ dpc(1 r_bro) */ * from t1, t2 where c1 = d1;
```

结果如下:

```

1  #NSET2: [1, 1, 16]
2  #ERECV: [1, 1, 16]; stask_no(-1), l_stask_no(0), n_key(0), recv_in_turn(0)
3  #ESEND: [1, 1, 16]; stask_no(0), type(DIRECT), sites(2:1:1), sql_invoke(0), pwj_opt(0),
table(-); INFO_BITS(0x4)
4  #PRJT2: [1, 1, 16]; exp_num(4), is_atom(FALSE)
5  #HASH2 INNER JOIN: [1, 1, 16]; KEY_NUM(1); KEY(T1.C1=T2.D1) KEY_NULL_EQU(0)
6  #GI: [1, 1, 8]; policy(RANDOM), gi_unit[0..0], scan_type[0](FULL)
7  #CSCN2: [1, 1, 8]; INDEX33651668(T1)
8  #ERECV: [1, 1, 8]; stask_no(0), l_stask_no(1), n_key(0), recv_in_turn(0)
9  #ESEND: [1, 1, 8]; stask_no(1), type(BROADCAST), sites(2:1:1), sql_invoke(0),
pwj_opt(0), table(-); INFO_BITS(0x4)
10 #GI: [1, 1, 8]; policy(RANDOM), gi_unit[0..0], scan_type[0](FULL)
11 #CSCN2: [1, 1, 8]; INDEX33651672(T2)

```

例 3 连接两侧分别加入分发操作, 可以使连接操作较好地利用并行特性, 是一种常见的分发策略。

```
explain select /*+ dpc(1 l_dis_r_dis) */ * from t1, t2 where c1 = d1;
```

结果如下:

```

1  #NSET2: [1, 1, 16]
2  #ERECV: [1, 1, 16]; stask_no(-1), l_stask_no(1), n_key(0), recv_in_turn(0)
3  #ESEND: [1, 1, 16]; stask_no(1), type(DIRECT), sites(3:1), sql_invoke(0), pwj_opt(0),
table(-); INFO_BITS(0x4)
4  #PRJT2: [1, 1, 16]; exp_num(4), is_atom(FALSE)

```

DM8 分布计算集群

```

5      #HASH2 INNER JOIN: [1, 1, 16]; KEY_NUM(1); KEY(T1.C1=T2.D1) KEY_NULL_EQU(0)
6      #ERECV: [1, 1, 8]; stask_no(1), l_stask_no(0), n_key(0), recv_in_turn(0)
7      #ESEND: [1, 1, 8]; stask_no(0), type(N_DEST), sites(2:1:1), sql_invoke(0),
pwj_opt(0), table(-), keys(T1.C1) ; INFO_BITS(0x4)
8      #GI: [1, 1, 8]; policy(RANDOM), gi_unit[0..0], scan_type[0](FULL)
9      #CSCN2: [1, 1, 8]; INDEX33651668(T1)
10     #ERECV: [1, 1, 8]; stask_no(1), l_stask_no(2), n_key(0), recv_in_turn(0)
11     #ESEND: [1, 1, 8]; stask_no(2), type(N_DEST), sites(2:1:1), sql_invoke(0),
pwj_opt(0), table(-), keys(T2.D1)
12     #GI: [1, 1, 8]; policy(RANDOM), gi_unit[0..0], scan_type[0](FULL)
13     #CSCN2: [1, 1, 8]; INDEX33651672(T2)

```

例 4 仅左侧分发，此处分发会按照右孩子的数据分布特征进行。保留了右孩子原始的分布特性，是一种部分分区连接优化。

```
explain select /*+ dpc(11_dis) */ * from t1, t2 where c1 = d1;
```

结果如下：

```

1  #NSET2: [1, 1, 16]
2  #ERECV: [1, 1, 16]; stask_no(-1), l_stask_no(1), n_key(0), recv_in_turn(0)
3  #ESEND: [1, 1, 16]; stask_no(1), type(DIRECT), sites(2:2:2), sql_invoke(0), pwj_opt(0),
table(-)
4  #PRJT2: [1, 1, 16]; exp_num(4), is_atom(FALSE)
5  #HASH2 INNER JOIN: [1, 1, 16]; KEY_NUM(1); KEY(T1.C1=T2.D1) KEY_NULL_EQU(0)
6  #ERECV: [1, 1, 8]; stask_no(1), l_stask_no(0), n_key(0), recv_in_turn(0)
7  #ESEND: [1, 1, 8]; stask_no(0), type(HASH), sites(2:1:1), sql_invoke(0),
pwj_opt(1), table(T2), keys(T1.C1) ; INFO_BITS(0x4)

```

DM8 分布计算集群

```

8          #GI: [1, 1, 8]; policy(RANDOM), gi_unit[0..0], scan_type[0](FULL)
9          #CSCN2: [1, 1, 8]; INDEX33651668(T1)
10         #GI: [1, 1, 8]; policy(USE_SQC_NO), gi_unit[0..0], scan_type[0](FULL)
11         #CSCN2: [1, 1, 8]; INDEX33651672(T2)
  
```

例 5 仅右侧分发，此处分发会按照左孩子的数据分布特征进行。保留了左孩子原始的分布特性，是一种部分分区智能连接优化。

```
explain select /*+ dpc(1 r_dis) */ * from t1, t2 where c1 = d1;
```

结果如下：

```

1  #NSET2: [1, 1, 16]
2  #ERECV: [1, 1, 16]; stask_no(-1), l_stask_no(0), n_key(0), recv_in_turn(0)
3  #ESEND: [1, 1, 16]; stask_no(0), type(DIRECT), sites(2:2,1:1), sql_invoke(0), pwj_opt(0),
table(-); INFO_BITS(0x4)
4  #PRJT2: [1, 1, 16]; exp_num(4), is_atom(FALSE)
5  #HASH2 INNER JOIN: [1, 1, 16]; KEY_NUM(1); KEY(T1.C1=T2.D1) KEY_NULL_EQU(0)
6  #GI: [1, 1, 8]; policy(USE_SQC_NO), gi_unit[0..0], scan_type[0](FULL)
7  #CSCN2: [1, 1, 8]; INDEX33651668(T1)
8  #ERECV: [1, 1, 8]; stask_no(0), l_stask_no(1), n_key(0), recv_in_turn(0)
9  #ESEND: [1, 1, 8]; stask_no(1), type(HASH), sites(2:1,1:1), sql_invoke(0),
pwj_opt(1), table(T1), keys(T2.D1); INFO_BITS(0x4)
10 #GI: [1, 1, 8]; policy(RANDOM), gi_unit[0..0], scan_type[0](FULL)
11 #CSCN2: [1, 1, 8]; INDEX33651672(T2)
  
```

例 6 两侧数据汇总到一个线程进行，通常适合两边都是小表的情况，汇总后数据仅在 SP 上。

```
explain select /*+ dpc(1 l_gat_r_gat) */ * from t1, t2 where c1 = d1;
```

DM8 分布计算集群

结果如下:

```

1  #NSET2: [1, 1, 16]
2  #ERECV: [1, 1, 16]; stask_no(-1), l_stask_no(1), n_key(0), recv_in_turn(0)
3  #ESEND: [1, 1, 16]; stask_no(1), type(DIRECT), sites(3:1), sql_invoke(0), pwj_opt(0),
table(-); INFO_BITS(0x4)
4  #PRJT2: [1, 1, 16]; exp_num(4), is_atom(FALSE)
5  #HASH2 INNER JOIN: [1, 1, 16]; KEY_NUM(1); KEY(T1.C1=T2.D1) KEY_NULL_EQU(0)
6  #ERECV: [1, 1, 8]; stask_no(1), l_stask_no(0), n_key(0), recv_in_turn(0)
7  #ESEND: [1, 1, 8]; stask_no(0), type(DIRECT), sites(2:1,1:1), sql_invoke(0),
pwj_opt(0), table(-); INFO_BITS(0x4)
8  #GI: [1, 1, 8]; policy(RANDOM), gi_unit[0..0], scan_type[0](FULL)
9  #CSCN2: [1, 1, 8]; INDEX33651668(T1)
10 #ERECV: [1, 1, 8]; stask_no(1), l_stask_no(2), n_key(0), recv_in_turn(0)
11 #ESEND: [1, 1, 8]; stask_no(2), type(DIRECT), sites(2:1,1:1), sql_invoke(0),
pwj_opt(0), table(-)
12 #GI: [1, 1, 8]; policy(RANDOM), gi_unit[0..0], scan_type[0](FULL)
13 #CSCN2: [1, 1, 8]; INDEX33651672(T2)
  
```

11.2.3 分组查询

本节优化的对象为分组查询语句。

例 1 适合本地分组后数量能减少较多的场合。

```

explain select /*+ dpc(1 x_dis_x) stat(t1 IG) parallel(4) */ count(*) from t1 group by
c2;
  
```

结果如下:

DM8 分布计算集群

```

1  #NSET2: [169968, 10000000, 4]

2  #ERECV: [169968, 10000000, 4]; stask_no(-1), l_stask_no(1), n_key(0),
recv_in_turn(0)

3  #ESEND: [169968, 10000000, 4]; stask_no(1), type(DIRECT), sites(3:4), sql_invoke(0),
pwj_opt(0), table(-)

4  #PRJT2: [169968, 10000000, 4]; exp_num(1), is_atom(FALSE)

5  #HAGR2: [169968, 10000000, 4]; grp_num(1), sfun_num(1), distinct_flag[0];
slave_empty(0) keys(T1.C2)

6  #ERECV: [169968, 10000000, 4]; stask_no(1), l_stask_no(0), n_key(0),
recv_in_turn(0)

7  #ESEND: [169968, 10000000, 4]; stask_no(0), type(N_DEST), sites(2:4,1:4),
sql_invoke(0), pwj_opt(0), table(-), keys(T1.C2) ; INFO_BITS(0x4)

8  #HAGR2: [169968, 10000000, 4]; grp_num(1), sfun_num(1),
distinct_flag[0]; slave_empty(0) keys(T1.C2)

9  #GI: [105723, 1000000000, 4]; policy(RANDOM), gi_unit[0..0],
scan_type[0](FULL)

10 #CSCN2: [105723, 1000000000, 4]; INDEX33651668(T1)
  
```

例 2 dis_x 适合本地分组并能不能就减少太多数据的场合。

```
explain select /*+ dpc(1 dis_x) */ count(*) from t1 group by c2;
```

结果如下：

```

1  #NSET2: [1, 1, 4]

2  #PRJT2: [1, 1, 4]; exp_num(1), is_atom(FALSE)

3  #HAGR2: [1, 1, 4]; grp_num(1), sfun_num(1), distinct_flag[0]; slave_empty(0)
keys(T1.C2)
  
```

DM8 分布计算集群

```

4      #ERECV: [1, 1, 4]; stask_no(-1), l_stask_no(0), n_key(0), recv_in_turn(0)
5      #ESEND: [1, 1, 4]; stask_no(0), type(N_DEST), sites(2:1:1), sql_invoke(0),
pwj_opt(0), table(-), keys(T1.C2); INFO_BITS(0x4)
6      #GI: [1, 1, 4]; policy(RANDOM), gi_unit[0..0], scan_type[0](FULL)
7      #CSCN2: [1, 1, 4]; INDEX33651668(T1)

```

例 3 适合本地分组后数据量较少的情况。

```

explain select /*+ dpc(1x_gat_x) stat(t1 IG) parallel(4) */ count(*) from t1 group by
c2;

```

结果如下：

```

1  #NSET2: [169968, 10000000, 4]
2  #PRJT2: [169968, 10000000, 4]; exp_num(1), is_atom(FALSE)
3  #HAGR2: [169968, 10000000, 4]; grp_num(1), sfun_num(1), distinct_flag[0];
slave_empty(0) keys(T1.C2)
4  #ERECV: [169968, 10000000, 4]; stask_no(-1), l_stask_no(0), n_key(0),
recv_in_turn(0)
5  #ESEND: [169968, 10000000, 4]; stask_no(0), type(DIRECT), sites(2:4:1:4),
sql_invoke(0), pwj_opt(0), table(-); INFO_BITS(0x4)
6  #HAGR2: [169968, 10000000, 4]; grp_num(1), sfun_num(1), distinct_flag[0];
slave_empty(0) keys(T1.C2)
7  #GI: [105723, 1000000000, 4]; policy(RANDOM), gi_unit[0..0],
scan_type[0](FULL)
8  #CSCN2: [105723, 1000000000, 4]; INDEX33651668(T1)

```

例 4 适合数据量较少，全部汇总到一个线程后再进行分组。

```

explain select /*+ dpc(1 gat_x) stat(t1 IG) parallel(4) */ count(*) from t1 group by c2;

```

DM8 分布计算集群

结果如下:

```

1  #NSET2: [169968, 10000000, 4]
2  #PRJT2: [169968, 10000000, 4]; exp_num(1), is_atom(FALSE)
3  #HAGR2: [169968, 10000000, 4]; grp_num(1), sfun_num(1), distinct_flag[0];
slave_empty(0) keys(T1.C2)
4  #ERECV: [105723, 1000000000, 4]; stask_no(-1), l_stask_no(0), n_key(0),
recv_in_turn(0)
5  #ESEND: [105723, 1000000000, 4]; stask_no(0), type(DIRECT), sites(2:4,1:4),
sql_invoke(0), pwj_opt(0), table(-); INFO_BITS(0x4)
6  #GI: [105723, 1000000000, 4]; policy(RANDOM), gi_unit[0..0],
scan_type[0](FULL)
7  #CSCN2: [105723, 1000000000, 4]; INDEX33651668(T1)
  
```

11.2.4 排序查询

本节优化的对象为分组排序语句。

例 1 适合数据量较少，全部汇总到一个线程后再进行排序。

```
explain select /*+ dpc(1 gat_x) */ * from t1 order by c2;
```

结果如下:

```

1  #NSET2: [1, 1, 16]
2  #PRJT2: [1, 1, 16]; exp_num(3), is_atom(FALSE)
3  #SORT3: [1, 1, 16]; key_num(1), is_distinct(FALSE), top_flag(0), is_adaptive(0)
4  #ERECV: [1, 1, 16]; stask_no(-1), l_stask_no(0), n_key(0), recv_in_turn(0)
5  #ESEND: [1, 1, 16]; stask_no(0), type(DIRECT), sites(2:1,1:1), sql_invoke(0),
pwj_opt(0), table(-); INFO_BITS(0x4)
  
```

DM8 分布计算集群

```
6      #GI: [1, 1, 16]; policy(RANDOM), gi_unit[0..0], scan_type[0](FULL)
7      #CSCN2: [1, 1, 16]; INDEX33651668(T1)
```

例 2 适合数据量较多，每路数据各自排序后汇总到一个线程后进行最终的归并排序。

```
explain select /*+ dpc(1 x_merge) */ * from t1 order by c2;
```

结果如下：

```
1  #NSET2: [1, 1, 16]
2  #PRJT2: [1, 1, 16]; exp_num(3), is_atom(FALSE)
3  #ERECV: [1, 1, 16]; stask_no(-1), l_stask_no(0), n_key(1), recv_in_turn(0)
4  #ESEND: [1, 1, 16]; stask_no(0), type(DIRECT), sites(2:1:1), sql_invoke(0),
   pwj_opt(0), table(-); INFO_BITS(0x24)
5  #SORT3: [1, 1, 16]; key_num(1), is_distinct(FALSE), top_flag(0), is_adaptive(0)
6  #GI: [1, 1, 16]; policy(RANDOM), gi_unit[0..0], scan_type[0](FULL)
7  #CSCN2: [1, 1, 16]; INDEX33651668(T1)
```

12 相关系统表和动态视图

下面这些系统表和动态视图，只有 SP 和 MP 上才能查询到。其中，在 SP 上查询到的是整个集群所有站点的信息，在 MP 上查询到的只有 MP 站点自身的信息。

12.1 DPC_NETWORK

记录网段信息。

| 序号 | 列名 | 列类型 | 含义 |
|----|-------------|--------------|----------------|
| 1 | ID | INT | 网段号 |
| 2 | NAME | VARCHAR(128) | 网段名称 |
| 3 | PREFIX | VARCHAR(64) | 前缀，如 192.168.0 |
| 4 | DESCRIPTION | VARCHAR(256) | 描述信息 |

12.2 DPC_HOST

记录主机信息。

| 序号 | 列名 | 列类型 | 含义 |
|----|-------------|--------------|------|
| 1 | ID | INT | 主机号 |
| 2 | NAME | VARCHAR(128) | 主机名 |
| 3 | DESCRIPTION | VARCHAR(256) | 描述信息 |

12.3 DPC_HOST_ADDR

记录主机上多块网卡的地址信息。

| 序号 | 列名 | 列类型 | 含义 |
|----|----|-----|-----|
| 1 | ID | INT | 主机号 |

DM8 分布计算集群

| | | | |
|---|----------|-------------|-------|
| 2 | NET_ID | INT | 网段号 |
| 3 | NET_ADDR | VARCHAR(64) | IP 地址 |

12.4 DPC_FOLDER

记录 BP 域相关的主机的目录信息。

| 序号 | 列名 | 列类型 | 含义 |
|----|-------------|--------------|------------|
| 1 | ID | INT | 目录号 |
| 2 | DOMAIN_ID | INT | 域号 |
| 3 | NAME | VARCHAR(128) | 目录名 |
| 4 | HOST_ID | INT | 主机号 |
| 5 | PATH | VARCHAR(256) | 路径 |
| 6 | DESCRIPTION | VARCHAR(256) | 描述信息 |
| 7 | XMAL_PORT | INT | XMAL 通信端口号 |

12.5 DPC_BP_DOMAIN

记录 BP 域信息。

| 序号 | 列名 | 列类型 | 含义 |
|----|-------------|--------------|-------|
| 1 | ID | INTEGER | BP 域号 |
| 2 | NAME | VARCHAR(128) | 域名 |
| 3 | DESCRIPTION | VARCHAR(256) | 描述信息 |

12.6 DPC_DOMAIN_FOLDER

记录每个域内包含的所有目录信息。

| 序号 | 列名 | 列类型 | 含义 |
|----|----|-----|----|
|----|----|-----|----|

DM8 分布计算集群

| | | | |
|---|--------------|-----|------------|
| 1 | DOMAIN_ID | INT | 域号 |
| 2 | FOLDER_ID | INT | 目录号 |
| 3 | BP_XMAL_PORT | INT | XMAL 通信端口号 |

12.7 DPC_FOLDER_INSTANCE

记录每个目录下的所有实例号信息。

| 序号 | 列名 | 列类型 | 含义 |
|----|-------------|-----|-----|
| 1 | FOLDER_ID | INT | 目录号 |
| 2 | INSTANCE_ID | INT | 实例号 |

12.8 DPC_BP_GROUP

记录系统中注册的每个 BP 组信息。

| 序号 | 列名 | 列类型 | 含义 |
|----|-------------|-----------------|---------------------|
| 1 | ID | INTEGER | BP 组号 |
| 2 | NAME | VARCHAR(128) | BP 组名 |
| 3 | DESCRIPTION | VARCHAR(256) | 描述信息 |
| 4 | RAFT_NUM | INTEGER | 包含 RAFT 组的个数 |
| 5 | RAFT_INFO | VARBINARY(2048) | 包含的所有 RAFT 组的 Id 信息 |
| 6 | INFO1 | INTEGER | 预留字段 |
| 7 | INFO2 | BIGINT | 预留字段 |
| 8 | INFO3 | VARBINARY(1024) | 预留字段 |

12.9 DPC_BP_RAFT

记录系统中注册的每个 RAFT 组信息。

DM8 分布计算集群

| 序号 | 列名 | 列类型 | 含义 |
|----|----------|-----------------|-----------------------------|
| 1 | RAFT_ID | INT | RAFT 组号 |
| 2 | GROUP_ID | INT | 本字段废弃，全部记为-1，无意义 |
| 3 | DPC_MODE | VARCHAR(8) | DMDPC 模式 |
| 4 | NAME | VARCHAR(128) | RAFT 组名 |
| 5 | IS_VALID | INT | BP 节点是否有效（节点是否可用，0 无效，1 有效） |
| 6 | INFO1 | INT | 预留字段 |
| 7 | INFO2 | BIGINT | 预留字段 |
| 8 | INFO3 | VARBINARY(1024) | 预留字段 |

12.10 DPC_INSTANCE

记录系统中所有实例对象信息。

| 序号 | 列名 | 列类型 | 含义 |
|----|-------------|--------------|--|
| 1 | RAFT_ID | INT | 系统内部 RAFT 组号。取值范围：0~1023 |
| 2 | INST_ID | INT | 系统内部实例号。取值范围：4096~8191 |
| 3 | NAME | VARCHAR(128) | 实例名 |
| 4 | DPC_MODE | VARCHAR(8) | DMDPC 模式。取值：MP，SP 或 BP |
| 5 | XMAL_PORT | INT | XMAL 通信端口号。用户通过 SP_CREATE_DPC_INSTANCE 中 AP_PORT 注册时配置，取值范围：1024~65534 |
| 6 | INST_PORT | INT | 实例端口。用户通过 SP_CREATE_DPC_INSTANCE 中 INST_PORT 注册时配置，取值范围：1024~65534 |
| 7 | IP_INTERNAL | VARCHAR(512) | 实例内网 IP 地址 |

DM8 分布计算集群

| | | | |
|----|-------------|-----------------|--|
| 8 | SYS_MODE | VARCHAR(32) | 系统模式 |
| 9 | SYS_STATUS | INT | 系统状态。1: 启动; 2: 启动, redo 完成; 3: MOUNT; 4: 打开; 5: 挂起; 6: 关闭 |
| 10 | STATUS | INT | 实例状态。1: 有效; 0: 无效; 2: SP 正常退出标记; 4: SP 灰度升级。仅对 SP 有效 |
| 11 | DESCRIPTION | VARCHAR(256) | 描述信息 |
| 12 | IP_EXTERNAL | VARCHAR(512) | 实例外网 IP 地址 |
| 13 | INFO1 | INT | 预留字段。其中, 这个字段的高 2 字节用来保存实例对 应的容错域 id, 用户可以执行 select info1>>16 fdom from DPC_INSTANCE;来查看结果 |
| 14 | INFO2 | BIGINT | 预留字段 |
| 15 | INFO3 | VARBINARY(1024) | 预留字段 |

12.11 DPC_NET_CONF

记录连接配置参数。

| 序号 | 列 | 数据类型 | 说明 |
|----|-----------|--------------|---|
| 1 | PARA_NAME | VARCHAR(128) | 参数名。 COMPRESS_LEVEL 消息压缩级别, 取值范围 0~10。0 表示不 压缩; 1 表示压缩速度最快但是压缩效率最低; 10 表示压缩速 度最慢但是压缩率最高, 默认 0; CRC_CHECK 消息的 CRC 检验; MONITOR_XLNK_TIME 监控链路消息收发用时, 单位微秒; |

DM8 分布计算集群

| | | | |
|---|------------|--------------|------------------------------------|
| | | | TIMEOUT 通信超时，单位为秒； XLNK_NUM 链路数 |
| 2 | PARA_VALUE | VARCHAR(128) | 参数值 |

12.12 DPC_TABLESPACE

记录 DMDPC 系统的用户表空间。

| 序号 | 列 | 数据类型 | 说明 |
|----|---------|-----------------|---------------------|
| 1 | TS_ID | INT | 全局表空间 ID，全局唯一 |
| 2 | RAFT_ID | INT | RAFT 组 ID |
| 3 | STATE | INT | 表空间状态。0：联机状态；1：脱机状态 |
| 4 | NAME | VARCHAR(256) | 表空间名，主键 |
| 5 | INFO1 | INT | 预留字段 |
| 6 | INFO2 | BIGINT | 预留字段 |
| 7 | INFO3 | VARBINARY(1024) | 预留字段 |

12.13 DPC HTS

记录 DMDPC 系统的混合表空间。

| 序号 | 列 | 数据类型 | 说明 |
|----|---------|--------------|---------------------|
| 1 | TS_ID | INT | 全局表空间 ID，全局唯一 |
| 2 | RAFT_ID | INT | RAFT 组 ID |
| 3 | STATE | INT | 表空间状态。0：联机状态；1：脱机状态 |
| 4 | NAME | VARCHAR(256) | 表空间名，主键 |
| 5 | INFO1 | INT | 预留字段 |

DM8 分布计算集群

| | | | |
|---|-------|-----------------|------|
| 6 | INFO2 | BIGINT | 预留字段 |
| 7 | INFO3 | VARBINARY(1024) | 预留字段 |

12.14 DPC_FAULT_DOMAIN

记录系统中所有容错域对象信息。

| 序号 | 列名 | 列类型 | 含义 |
|----|-------------|-----------------|-----------------------|
| 1 | DOMAIN_ID | INT | 系统内部容错域号。取值范围: 1~4096 |
| 2 | NAME | VARCHAR(128) | 容错域名 |
| 3 | DESCRIPTION | VARCHAR(256) | 容错域描述 |
| 4 | INFO1 | INT | 预留字段 |
| 5 | INFO2 | BIGINT | 预留字段 |
| 6 | INFO3 | VARBINARY(1024) | 预留字段 |

12.15 V\$DPC_STASK_THRD

查看每个子任务、每个线程的执行统计信息。当 DM.INI 参数 ENABLE_MONITOR=1 时有效。

| 序号 | 列 | 数据类型 | 说明 |
|----|-------------|---------|----------|
| 1 | EXEC_ID | BIGINT | 执行号 |
| 2 | MPP_EXEC_ID | BIGINT | 内部的表示执行号 |
| 3 | STASK_NO | INTEGER | 子任务序号 |
| 4 | THRD_NO | INTEGER | 工作线程序号 |
| 5 | PLAN_NO | INTEGER | 子计划序号 |

DM8 分布计算集群

| | | | |
|----|----------------|---------------|--------------------------------|
| 6 | START_TIME | DATETIME(0) | 当前线程的开始工作时间 |
| 7 | TIME_USED | BIGINT | 当前线程执行子任务的耗时，单位毫秒 |
| 8 | FIRST_ROW_USED | BIGINT | 返回第一行数据的耗时，单位毫秒 |
| 9 | N_ROWS_SEND | BIGINT | 子任务输出的行数，对于 HTAB 操作符为存储的行数 |
| 10 | N_BYTES_SEND | BIGINT | 子任务输出的数据大小，字节为单位 |
| 11 | N_ROWS_RECV | VARCHAR(1024) | 如果子任务有 ERECV 操作符，展示接收的行数 |
| 12 | P_WAIT_TIMES | VARCHAR(1024) | 打开限流时，请求资源操作的等待次数。未打开限流时，此值无意义 |
| 13 | IS_OVER | CHAR(1) | 工作线程执行是否结束。Y 是；N 否 |
| 14 | THRD_ID | BIGINT | 工作线程 ID |
| 15 | SESS_ID | BIGINT | 会话号 |

例 查看执行号 16777624 的执行情况。

```
select * from v$dpc_stask_thrd where exec_id = 16777624;
```

由于 V\$dpc_stask_thrd 记录的是原始信息，我们也可以在它的基础上定义视图方便查看。

例 创建一个视图 v_eet，将 V\$DPC_STASK_THRD 中的信息，通过 v_eet 展示出来。

```
create or replace view v_eet as select y.name name,
                                     exec_id,
                                     mpp_exec_id,
                                     stask_no,
                                     thrd_no,
                                     cast(start_time as varchar(30)) as START_TIME,
                                     time_used as TIME_USED_IN_MS,
                                     first_row_used as FIRST_ROW_IN_MS,
```

DM8 分布计算集群

```

n_rows_send as SENT_ROWS,

case when n_bytes_send > 1024 * 1024 * 1024 then

cast(n_bytes_send / 1024.0 / 1024 / 1024 as decimal(38,3)) || 'G'

when n_bytes_send > 1024 * 1024 then

cast(n_bytes_send / 1024.0 / 1024 as decimal(38,3)) || 'M'

when n_bytes_send = 0 then 0

else cast(n_bytes_send / 1024.0 as

decimal(38,3)) || 'K' end as SENT,

n_rows_recv as RECV_ROWS,

is_over

from SYS.V$DPC_STASK_THRD x, SYS.DPC_BP_RAFT y

where SF_GET_EP_SEQNO(x.rowid) = y.raft_id

order by stask_no, name, thrd_no;

```

通过 v_eeet 查看执行事务号为 16777624 的执行语句的执行时间。

```
select * from v_eeet where exec_id = 16777624;
```

查看结果:

```
SQL> select * from v_eeet where exec_id = 16777624;
```

| 行号 | NAME | EXEC_ID | MPP_EXEC_ID | STASK_NO | THRD_NO | START_TIME | TIME_USED_IN_MS | FIRST_ROW_IN_MS | SENT_ROWS | SENT | RECV_ROWS |
|----|------|----------|-------------|----------|---------|-------------------------|-----------------|-----------------|-----------|---------|-----------|
| 1 | SP2 | 16777624 | 17542759 | -1 | 0 | 2020-10-28 14:49:04.33 | 78 | 78 | 0 | 0 | 1 |
| 2 | MP | 16777624 | 17542759 | 0 | 0 | 2020-10-28 14:49:04.377 | 0 | 1 | 1326 | 10.400K | 0 |
| 3 | MP | 16777624 | 17542759 | 0 | 1 | 2020-10-28 14:49:04.377 | 0 | 1 | 1100 | 8.635K | 0 |
| 4 | MP | 16777624 | 17542759 | 0 | 2 | 2020-10-28 14:49:04.377 | 16 | 1 | 1078 | 8.463K | 0 |
| 5 | MP | 16777624 | 17542759 | 0 | 3 | 2020-10-28 14:49:04.377 | 0 | 1 | 1096 | 8.604K | 0 |
| 6 | MP | 16777624 | 17542759 | 0 | 4 | 2020-10-28 14:49:04.377 | 0 | 1 | 1010 | 7.932K | 0 |
| 7 | MP | 16777624 | 17542759 | 0 | 5 | 2020-10-28 14:49:04.377 | 0 | 1 | 1348 | 10.572K | 0 |
| 8 | MP | 16777624 | 17542759 | 0 | 6 | 2020-10-28 14:49:04.377 | 0 | 1 | 1092 | 8.572K | 0 |
| 9 | MP | 16777624 | 17542759 | 0 | 7 | 2020-10-28 14:49:04.377 | 0 | 1 | 1200 | 9.416K | 0 |
| 10 | BP1 | 16777624 | 17542759 | 1 | 0 | 2020-10-28 14:49:04.361 | 31 | 31 | 1 | 0.029K | 4625 |
| 11 | BP1 | 16777624 | 17542759 | 1 | 1 | 2020-10-28 14:49:04.361 | 31 | 31 | 1 | 0.029K | 0 |
| 12 | BP1 | 16777624 | 17542759 | 1 | 2 | 2020-10-28 14:49:04.361 | 31 | 31 | 1 | 0.029K | 0 |
| 13 | BP1 | 16777624 | 17542759 | 1 | 3 | 2020-10-28 14:49:04.361 | 31 | 31 | 1 | 0.029K | 0 |
| 14 | BP1 | 16777624 | 17542759 | 1 | 4 | 2020-10-28 14:49:04.361 | 31 | 31 | 1 | 0.029K | 0 |
| 15 | BP1 | 16777624 | 17542759 | 1 | 5 | 2020-10-28 14:49:04.361 | 31 | 31 | 1 | 0.029K | 0 |
| 16 | BP1 | 16777624 | 17542759 | 1 | 6 | 2020-10-28 14:49:04.361 | 31 | 31 | 1 | 0.029K | 0 |
| 17 | BP1 | 16777624 | 17542759 | 1 | 7 | 2020-10-28 14:49:04.361 | 31 | 31 | 1 | 0.029K | 0 |
| 18 | BP2 | 16777624 | 17542759 | 1 | 0 | 2020-10-28 14:49:04.377 | 32 | 32 | 1 | 0.029K | 4625 |
| 19 | BP2 | 16777624 | 17542759 | 1 | 1 | 2020-10-28 14:49:04.377 | 16 | 16 | 1 | 0.029K | 0 |
| 20 | BP2 | 16777624 | 17542759 | 1 | 2 | 2020-10-28 14:49:04.377 | 32 | 16 | 1 | 0.029K | 0 |
| 21 | BP2 | 16777624 | 17542759 | 1 | 3 | 2020-10-28 14:49:04.377 | 16 | 16 | 1 | 0.029K | 0 |
| 22 | BP2 | 16777624 | 17542759 | 1 | 4 | 2020-10-28 14:49:04.377 | 16 | 16 | 1 | 0.029K | 0 |
| 23 | BP2 | 16777624 | 17542759 | 1 | 5 | 2020-10-28 14:49:04.377 | 16 | 16 | 1 | 0.029K | 0 |
| 24 | BP2 | 16777624 | 17542759 | 1 | 6 | 2020-10-28 14:49:04.377 | 32 | 16 | 1 | 0.029K | 0 |
| 25 | BP2 | 16777624 | 17542759 | 1 | 7 | 2020-10-28 14:49:04.377 | 32 | 16 | 1 | 0.029K | 0 |
| 26 | BP2 | 16777624 | 17542759 | 2 | 0 | 2020-10-28 14:49:04.392 | 16 | 16 | 1 | 0.029K | 16 |

26 rows got

DM8 分布计算集群

图 12.1 执行语句的统计信息

从上图观察同一个 stask_no 的不同 thrd_no 线程的执行时间 (TIME_USED_IN_MS), 可以大概看出工作线程间的负载是否均衡, 哪一个子任务耗时最久等信息。

通过观察 SENT_ROWS、SENT 和 RECV_ROWS 字段信息, 可以看到每个线程发送、接收的数据量, 分析出是否数据分布不均匀。比如某个子任务并行度为 2、THRD_NO=0 的线程接收了 1000000 记录, THRD_NO=1 的线程接收了 500 条记录, 差别这么大的情况下我们可以继续分析数据来源, 为何会存在这么大的不均衡。

例 执行事务号 16777624 对应的绿色区域为最根层子任务 (stask_no 等于-1), 安排了一个工作线程, 耗时 78 毫秒。

红色区域为子任务 0, 共安排了 8 个线程调度执行, 耗时最长的为 2 号线程用时 16ms, 其余都是 0ms。由于子任务 0 工作量不大, 所以这点差距可以忽略。

假设某个子任务工作量很大的前提下, 不同线程完成的时间差距很大, 这时可以先观察 START_TIME, 查看线程是否同时被调度? 接收、发送行数是否均衡?

通过调整数据分布和工作线程个数, 或者改变数据分发方式, 我们可以对数据负载做出一定程度的均衡。

12.16 V\$RLOG_RAFT_INFO

查询当前节点的选举状态和日志提交信息。配置成多副本系统的情况下, V\$RLOG_RAFT_INFO 的值才是真实有效的, 非多副本系统情况下, V\$RLOG_RAFT_INFO 的值无实际意义。

| 序号 | 列 | 数据类型 | 说明 |
|----|---------|--------------|-----------------|
| 1 | TERM_ID | INTERGER | 当前节点所处的任期号 |
| 2 | STATE | VARCHAR(16) | 当前节点所处的角色 |
| 3 | LEADER | VARCHAR(256) | 当前的主库 (领导者) 实例名 |

DM8 分布计算集群

| | | | |
|----|-------------------|---------------|---|
| 4 | LOG_TERM_ID | INTEGER | 当前节点已经刷盘的任期号 |
| 5 | C_SEQ | BIGINT | 当前节点已经提交到的日志包序号 |
| 6 | C_LSN | BIGINT | 当前节点已经提交到的 LSN 值 |
| 7 | F_SEQ_ARR | VARCHAR(2048) | 所有实例已经刷盘完成的日志包序号数组，仅在主库上查询才有效 |
| 8 | F_LSN_ARR | VARCHAR(2048) | 所有实例已经刷盘完成的 LSN 数组，仅在主库上查询才有效 |
| 9 | H_SEQ | BIGINT | 主节点已经提交到的日志包序号 |
| 10 | H_LSN | BIGINT | 主节点已经提交到的 LSN 值 |
| 11 | NEED_WAIT | VARCHAR(8) | 备库是否达到日志堆积上限，仅在主库上查询才有效 |
| 12 | SELF_STATUS | VARCHAR(8) | 当前节点实例状态是否有效，仅在主库上查询才有效 |
| 13 | HP_FLAG | VARCHAR(8) | 当前节点日志是否堆积 |
| 14 | SWITCH_TIME | VARCHAR(50) | 当前节点上一次模式切换的时间 |
| 15 | ARCH_CHG_F LAG | VARCHAR(16) | 当前节点是否在执行节点变更 |
| 16 | ARCH_CHG_ STAT | VARCHAR(32) | 当前节点所处的变更状态 |
| 17 | DEST_ID_ARR | VARCHAR(2048) | 当前节点实例 ID 数组。与 F_SEQ_ARR 和 F_LSN_ARR 一一对应 |
| 18 | RAFT_SHADO W | INTERGER | 当前节点是否是影子库，0: RAFT 库; 1: 影子库 |

例 查询当前节点的所有选举状态及日志提交信息。

```
SQL> select * from v$rlog_raft_info;
```

```
//查询结果如下:
```

DM8 分布计算集群

```

行号          TERM_ID          STATE  LEADER LOG_TERM_ID C_SEQ          C_LSN
F_SEQ_ARR          F_LSN_ARR          H_SEQ          H_LSN
NEED_WAIT  SELF_STATUS  HP_FLAG  SWITCH_TIME  ARCH_CHG_FLAG  ARCH_CHG_STAT
DEST_ID_ARR RAFT_SHADOW
-----
1            1            LEADER BPI      1            2501          6935
(2501, 2501, 2501) (6935, 6935, 6935) 2501          6935          FALSE
VALID        FALSE        2022-01-21 09:36:53.685 FALSE C_NORMAL (1,2,3) 0

已用时间: 1.271(毫秒). 执行号:1.

```

当某一备库故障，主备启动后还未进行过正常同步，在主库上查询到对应库 F_SEQ_ARR 及 F_LSN_ARR 将为 0。此时备库处于失效状态。对于集群而言，只要存活节点数目超过半数整个集群就是正常的。

例 备库 RAFT_03 已失效，但在集群中仍有两个节点正常超过半数，但此时整个集群仍可对外正常工作。

```

SQL> select * from v$rlog_raft_info;

//查询结果如下:

行号          TERM_ID          STATE  LEADER LOG_TERM_ID C_SEQ          C_LSN
F_SEQ_ARR          F_LSN_ARR          H_SEQ          H_LSN
NEED_WAIT  SELF_STATUS  HP_FLAG  SWITCH_TIME  ARCH_CHG_FLAG  ARCH_CHG_STAT
DEST_ID_ARR RAFT_SHADOW
-----

```

DM8 分布计算集群

```

-----
-----
1          1          LEADER BPI      1          2501          6935
(2501, 2501, 0) (6935, 6935, 0) 2501          6935          FALSE      VALID
FALSE      2022-01-21 09:36:53.685 FALSE C_NORMAL (1,2,3) 0

已用时间: 1.271(毫秒). 执行号:1.

```

12.17 V\$SQL_NODE_HISTORY

通过该视图既可以查询 SQL 执行节点信息，包括 SQL 节点的类型、进入次数和使用时间等等；又可以查询所有执行的 SQL 节点执行情况，如哪些使用最频繁、耗时多少等。

当 INI 参数 ENABLE_MONITOR 和 MONITOR_SQL_EXEC 都开启时，才会记录 SQL 执行节点信息。

| 序号 | 列 | 数据类型 | 说明 |
|----|-----------------------|---------|---------|
| 1 | SEQ_NO | INTEGER | 序列号 |
| 2 | EXEC_ID | INTEGER | 执行 ID |
| 3 | NODE | BIGINT | 节点 ID |
| 4 | TYPE\$ | INTEGER | 节点类型 |
| 5 | BYTES_DYNAMIC_ALLOCED | BIGINT | 动态分配字节数 |
| 6 | BYTES_DYNAMIC_FREED | BIGINT | 动态释放字节数 |
| 7 | N_ENTER | INTEGER | 节点进入次数 |
| 8 | TIME_USED | INTEGER | 节点执行所耗时 |

DM8 分布计算集群

| | | | |
|----|-------------|---------|---------------------------------|
| 9 | PLN_OP_ID | INTEGER | MPP 模式下, 节点所属通讯操作符中的序号 |
| 10 | BYTES_SEND | INTEGER | 发送的字节数 |
| 11 | BYTES_RECV | INTEGER | 接收的字节数 |
| 12 | ROWS_SEND | INTEGER | 发送的行数 |
| 13 | ROWS_RECV | INTEGER | 接收的行数 |
| 14 | BDTA_SEND | INTEGER | 发送 BDTA 的次数 |
| 15 | BDTA_RECV | INTEGER | 接收 BDTA 的次数 |
| 16 | MAL_ID | BIGINT | 邮件标识号 |
| 17 | MPP_EXEC_ID | INTEGER | MPP 会话句柄上的执行序号, 同一个会话上的每个节点上值相同 |
| 18 | STASK_NO | INTEGER | 节点所在的子任务号 |
| 19 | THRD_NO | INTEGER | 节点所在的线程号 |

12.18 V\$DPC_MP_CFG

记录 RAFT 组内的 MP 实例信息。由于一个 RAFT 内最多 9 个实例, 因此最多 9 条记录。

| 序号 | 列 | 数据类型 | 说明 |
|----|----------------|--------------|---|
| 1 | INST_NAME | VARCHAR(128) | 实例名 |
| 2 | IP_ADDRESS | VARCHAR(128) | IP 地址 |
| 3 | MP_PORT | SMALLINT | MP_PORT |
| 4 | RECONN_TIMEOUT | INT | 连接 MP 失败时, 尝试重连的超时时间, 单位为秒。 -2 表示无限制 (即 MP_RECONN_TIMEOUT 取最大值)。该值由 MP.INI 中参数 MP_RECONN_TIMEOUT 来设置 |

12.19 V\$DPC_QC_HISTORY

记录查询调度总单元 QC 的历史信息。

| 序号 | 列 | 数据类型 | 说明 |
|----|-------------|------------|-------------------------------------|
| 1 | SESS_ID | BIGINT | 和 V\$SESSIONS 的 SESS_ID 字段匹配 |
| 2 | SESS_SEQ | INTEGER | 和 V\$SESSIONS 的 SESS_SEQ 字段匹配 |
| 3 | EID | BIGINT | ESESSION 的 ID |
| 4 | THRD_ID | BIGINT | 创建 QC 的线程 ID |
| 5 | STMT_ID | INTEGER | 句柄 ID |
| 6 | EXEC_ID | BIGINT | 执行 ID |
| 7 | MPP_EXEC_ID | BIGINT | 内部执行 ID |
| 8 | SQC_EXEC_ID | INTEGER | 内部执行 ID, 当函数调用包含 SQL 时, 每一行调用时取值会不同 |
| 9 | N_STASK | INTEGER | 子任务的个数 |
| 10 | IDU_FLAG | VARCHAR(6) | 增删改的优化标记 |
| 11 | N_BP | INTEGER | 计划涉及到的集群中节点个数, 可能包含 SP、BP、MP |
| 12 | N_HTAB | INTEGER | 子计划中 HTAB 操作符的个数 |
| 13 | N_SPL | INTEGER | 子计划中 SPOOL 操作符的个数 |
| 14 | N_DPP | INTEGER | 子计划中涉及的动态分区裁剪参数个数 |
| 15 | IS_OVER | CHAR(1) | 是否执行结束。Y 是; N 否 |

12.20 V\$ARCH_STATUS

查询归档状态信息, 归档状态是由主库记录和维持的, 此视图只在主库上查询有效, 备库上的查询结果没有实际意义。

DM8 分布计算集群

| 序号 | 列 | 数据类型 | 说明 |
|----|-------------|--------------|--|
| 1 | ARCH_TYPE | VARCHAR(256) | 归档类型 |
| 2 | ARCH_DEST | VARCHAR(256) | 归档目标，本地归档和 REMOTE 归档为归档路径，其他类型为归档目标实例名。 |
| 3 | ARCH_STATUS | VARCHAR(256) | 归档状态，Valid 为有效状态，Invalid 为无效状态，ASYNC_SEND 为异步恢复状态。 |
| 4 | ARCH_SRC | VARCHAR(256) | 对 REMOTE 归档为源实例名，对其他归档类型为本地实例名 |

12.21 V\$DPC_EGTS_MP_INFO

MP 上记录的各 SP 和 BP 的全局时钟系统 (GTS) 信息。

| 序号 | 列名 | 列类型 | 含义 |
|----|-----------------|-------------|------------------------------|
| 1 | RAFT_ID | INT | SP 和 BP 的 RAFT_ID |
| 2 | STAT | VARCHAR(20) | 状态 (OPEN/SHUTDOWN/CONN_LOST) |
| 3 | MIN_TID | BIGINT | 最小 TRXID |
| 4 | N_LONG | INT | 长事务个数 |
| 5 | LAST_CHECK_TICK | BIGINT | 上次检查的 TICK 计数 |

12.22 V\$DPC_ENET_HISTORY

获取 ENET 模块更新各个节点的设置历史。

| 序号 | 列名 | 列类型 | 含义 |
|----|--------------|-----|-------------|
| 1 | SELF_RAFT_ID | INT | 自身的 RAFT_ID |
| 2 | SELF_INST_ID | INT | 自身的 INST_ID |

DM8 分布计算集群

| | | | |
|---|--------------|--------------|--------------------|
| 3 | DEST_RAFT_ID | INT | 目标的 RAFT_ID |
| 4 | DEST_INST_ID | INT | 目标的 INST_ID |
| 5 | TYPE | VARCHAR(20) | 操作类型 (SET/RELEASE) |
| 6 | SITE_ARRD | VARBINARY(8) | 节点 XSITE 的内存地址 |
| 7 | DESC | VARCHAR(30) | 变更原因描述 |
| 8 | TIME | TIMESTAMP | 变更发生的时间 |

12.23 V\$ARCH_SYS

获取全局归档配置信息。

| 序号 | 列名 | 列类型 | 含义 |
|----|------------------------|-------------|-------------------------------|
| 1 | ARCH_COUNT | INT | 归档数目 |
| 2 | ARCH_CFG_CHANGING | INT | 归档变更标记 |
| 3 | ARCH_WAIT_APPLY | INT | 全局事务一致标记 |
| 4 | RAFT_SELF_ID | INT | 自身 RAFT 归档序号, 仅 RAFT 归档有效 |
| 5 | XMAL_IP | VARCHAR(64) | RAFT 归档通信 IP |
| 6 | XMAL_PORT | INT | RAFT 归档通信端口 |
| 7 | XMAL_HB_INTERVAL | INT | RAFT 归档 XMAL 链路心跳检测间隔 |
| 8 | RAFT_HB_INTERVAL | INT | RAFT 归档心跳检测间隔 |
| 9 | RAFT_VOTE_INTERVAL | INT | RAFT 归档选举超时间隔 |
| 10 | ARCH_RESERVE_TIME | INT | 归档保留时间。单位分钟 |
| 11 | ARCH_LOCAL_SHARE | INT | 远程归档是否共享其它节点本地归档路径。0: 否; 1: 是 |
| 12 | ARCH_LOCAL_SHARE_CHECK | INT | 是否检查远程归档路径和其它节点本地 |

DM8 分布计算集群

| | | | |
|--|--|--|-------------------|
| | | | 归档路径相等。0: 否; 1: 是 |
|--|--|--|-------------------|

12.24 V\$DM_ARCH_INI

归档参数信息。

| 序号 | 列 | 数据类型 | 说明 |
|----|------------------|---------------|--|
| 1 | ARCH_NAME | VARCHAR (128) | 归档名称 |
| 2 | ARCH_TYPE | VARCHAR (128) | 归档类型 |
| 3 | ARCH_DEST | VARCHAR (512) | 对于 LOCAL 归档, 表示归档路径; 对于 REMOTE 归档, 表示本节点归档要发送到的实例名; 对于其余类型的归档, 表示归档目标实例名 |
| 4 | ARCH_FILE_SIZE | INTEGER | 归档文件大小 |
| 5 | ARCH_SPACE_LIMIT | INTEGER | 归档文件的磁盘空间限制, 单位 MB |
| 6 | ARCH_HANG_FLAG | INTEGER | 如果本地归档时磁盘空间不够, 是否让服务器挂起。无实际意义, 对本地归档类型和远程归档类型显示为1, 其他归档类型显示为NULL |
| 7 | ARCH_TIMER_NAME | VARCHAR (128) | 对于异步归档, 表示定时器名称; 其余类型归档显示 NULL |
| 8 | ARCH_IS_VALID | CHAR(1) | 归档状态, 是否有效。0: 否; 1: 是 |
| 9 | ARCH_WAIT_APPLY | INTEGER | 在主备系统中, 主库发送归档日志给备库之前, 是否需要等待备库将上一批日志重演完成。 |

DM8 分布计算集群

| | | | |
|----|------------------------|--------------|--|
| | | | <p>0: 为高性能模式, 不用等待重演完成;</p> <p>1: 为数据一致模式, 需等待重演完成。</p> <p>如果为本地归档, 因不需要重演, 值定为 NULL</p> |
| 10 | ARCH_INCOMING_PATH | VARCHAR(256) | <p>对 REMOTE 归档, 表示远程节点发送过来的归档在本地的保存目录。其余类型归档显示 NULL</p> |
| 11 | ARCH_CURR_DEST | VARCHAR(256) | <p>当前归档目标实例名。如果备库是 DMDS 集群, 则归档目标是备库控制节点, 该字段表示当前被选定为归档目标的节点实例名。备库是单机的情况下, 就是指的备库实例名</p> |
| 12 | ARCH_FLUSH_BUF_SIZE | INTEGER | <p>归档刷盘日志包含并缓存大小。单位为 KB</p> |
| 13 | ARCH_RESERVE_TIME | INTEGER | <p>归档保留时间。单位分钟</p> |
| 14 | ARCH_LOCAL_SHARE | INTEGER | <p>远程归档是否共享其它节点本地归档路径。0: 否; 1: 是</p> |
| 15 | ARCH_LOCAL_SHARE_CHECK | INTEGER | <p>是否检查远程归档路径和其它节点本地归档路径相等。0: 否; 1: 是</p> |
| 16 | ARCH_SEND_DELAY | INTEGER | <p>源库到异步备库的归档延时发送时间。单位分钟。取值范围 0~1440。缺省值 0, 表示不启用归档延时发送功能</p> |
| 17 | ARCH_DEST_IP | VARCHAR(64) | <p>RAFT 归档目标 IP</p> |

DM8 分布计算集群

| | | | |
|----|----------------|---------|-----------------------------------|
| 18 | ARCH_DEST_PORT | INTEGER | RAFT 归档目标端口 |
| 19 | ARCH_DEST_ID | INTEGER | 归档目标多副本节点编号, 用于标识多副本 RAFT 组中不同的节点 |

12.25 V\$DPC_ESESS_STMTS

查询 DMDPC 系统内部所有 ESESSION 的所有语句句柄信息。ESESSION 为 SP 和其他节点之间的内部会话。该视图提供给达梦内部开发人员调试之用。

| 序号 | 列 | 数据类型 | 说明 |
|----|---------------|----------|-----------------|
| 1 | STMT_ID | INTERGER | 句柄 ID |
| 2 | SESS_ID | BIGINT | 会话 ID |
| 3 | ESESS_ID | BIGINT | ESESSION 的 ID |
| 4 | PLAN_NO | INTEGER | 计划序号 |
| 5 | SQC_EXEC_ID | BIGINT | 语句执行序号 |
| 6 | N_BOXID_REUSE | INTEGER | 当前句柄上已分配的 BOXID |
| 7 | FREE_FLAG | INTEGER | 空闲句柄标记 |

12.26 V\$DPC_EXA_INFO

查询当前连接的服务器正在自动处理的异步事务信息。

| 序号 | 列 | 数据类型 | 说明 |
|----|---------------|---------|----------------|
| 1 | TID | BIGINT | 异步处理事务 ID |
| 2 | STATUS | INTEGER | 事务处理状态 |
| 3 | BRANCH_STATUS | INTEGER | 重构异步事务在来源站点的状态 |
| 4 | CMTSEQ | BIGINT | 提交序号 |

DM8 分布计算集群

| | | | |
|---|------------|---------|-----------------|
| 5 | ACKID | BIGINT | 异步处理通信标识 |
| 6 | TICK | BIGINT | 开始异步处理的时间戳 |
| 7 | N_RAFT | INTEGER | 事务涉及的站点数 |
| 8 | RAFTID_ARR | VARCHAR | 事务涉及的站点 RAFT ID |
| 9 | STATUS_ARR | VARCHAR | 事务在各站点的分支事务状态 |

12.27 V\$DPC_ROT_INFO

查询集群各个节点已加载的只读表信息。此外，可以通过 SF_GET_EP_SEQNO(ROWID); 查看本视图数据所在实例的 RAFT_ID。

| 序号 | 列 | 数据类型 | 说明 |
|----|----------------|-------------|--------------|
| 1 | ROT_TID | INTEGER | 只读表 ID |
| 2 | ROT_VERSION | INTEGER | 只读表版本号 |
| 3 | TEMP_TID | INTEGER | 只读表对应的临时表 ID |
| 4 | TEMP_VERSION | INTEGER | 只读表对应的临时表版本号 |
| 5 | LOAD_TIME | DATETIME(0) | 加载时间 |
| 6 | LAST_USED_TIME | DATETIME(0) | 最后一次使用时间 |

12.28 V\$DPC_ESESS

查询 DMDPC 系统内部所有 ESESSION 的信息。ESESSION 为 SP 和其他节点之间的内部会话。

| 序号 | 列 | 数据类型 | 说明 |
|----|------------|---------|-----------------|
| 1 | EID | BIGINT | 会话 ID |
| 2 | SRC_SITEID | INTEGER | 对应 SP 的 RAFT_ID |

DM8 分布计算集群

| | | | |
|----|--------------|---------|-----------------------------|
| 3 | BP_CTL_BOXID | INTEGER | BP 端 ESESS 会话用于接收消息的消息盒子号 |
| 4 | TRX_BOXID | INTEGER | SP 上用于处理事务使用的消息盒子号 |
| 5 | SESS_ID | BIGINT | 对应会话的 ID |
| 6 | M_EID | BIGINT | 目前未使用（废弃字段），全是 NULL |
| 7 | N_SITE | INTEGER | 缓存的远程站点对象数 |
| 8 | N_TRX_SITE | INTEGER | 当前事务涉及的节点个数 |
| 9 | ETHD_GRP_ID | INTEGER | ESESS 对应的线程池编号，非 ethd 线程为-1 |
| 10 | THRD_ID | INTEGER | ESESS 对应的线程号，不与线程关联时为-1 |

12.29 V\$DPC_EDCT_RAFT

查询 DMDPC 系统各节点内存中记录的 RAFT 信息。MP 上的信息对应 DPC_BP_RAFT 系统表的内容。SP 和 BP 上的内容是 MP 发送过来的。

| 序号 | 列 | 数据类型 | 说明 |
|----|--------------|--------------|-----------------------------|
| 1 | RAFT_ID | INTEGER | RAFT 组号 |
| 2 | DPC_MODE | VARCHAR(8) | DMDPC 模式 |
| 3 | IS_VALID | INTEGER | BP 节点是否有效（节点是否可用，0 无效，1 有效） |
| 4 | NAME | VARCHAR(128) | RAFT 组名 |
| 5 | L_TERM_ID | BIGINT | 主库的任期号 |
| 6 | DISCARD_TICK | BIGINT | 废弃时的 tick 时间值 |

12.30 V\$DPC_EDCT_INSTANCE

查询 DMDPC 系统各节点内存中记录的实例信息。MP 上的信息对应 DPC_INSTANCE 系统表的内容。SP 和 BP 上的内容是 MP 发送过来的。

DM8 分布计算集群

| 序号 | 列 | 数据类型 | 说明 |
|----|--------------|--------------|---|
| 1 | RAFT_ID | INTEGER | 系统内部 RAFT 组号。取值范围：0~1023 |
| 2 | INST_ID | INTEGER | 系统内部实例号。取值范围：4096~8191 |
| 3 | NAME | VARCHAR(128) | 实例名 |
| 4 | DPC_MODE | VARCHAR(16) | DMDPC 模式。取值范围：MP, SP 或 BP |
| 5 | XMAL_PORT | INTEGER | XMAL 通信端口号。用户通过 SP_CREATE_DPC_INSTANCE 中 AP_PORT 注册时配置，取值范围：1024~6553 |
| 6 | INST_PORT | INTEGER | 实例端口。用户通过 SP_CREATE_DPC_INSTANCE 中 INST_PORT 注册时配置，取值范围：1024~65534 |
| 7 | IP_INTERNAL | VARCHAR(64) | 实例内网 IP 地址 |
| 8 | IP_EXTERNAL | VARCHAR(64) | 实例外网 IP 地址 |
| 9 | SYS_MODE | VARCHAR(16) | 系统模式 |
| 10 | SYS_STATUS | VARCHAR(16) | 系统状态 |
| 11 | STATUS | VARCHAR(16) | 实例状态 |
| 12 | AUX_SP | CHAR(1) | 是否辅助 SP |
| 13 | DISCARD_TICK | BIGINT | 废弃时的 tick 时间值 |
| 14 | DESC | VARCHAR(256) | 描述信息 |
| 15 | BS_FLAG | INTEGER | BS 启动标记。0：非 BS；1：BS |
| 16 | FAULT_DOMAIN | VARCHAR(256) | 实例对应的容错域 ID。NULL 表示实例不属于任何一个容错域 |

12.31 V\$ETHD_GRP

用于查看 DMDPC 系统各节点的 ESESS 线程池信息。

DM8 分布计算集群

| 序号 | 列 | 数据类型 | 说明 |
|----|--------------|---------|-----------------------|
| 1 | ID | INTEGER | 线程池 ID |
| 2 | N_USED_THRD | INTEGER | 当前所有正在使用线程的数量 |
| 3 | N_THRD | INTEGER | 线程池中线程总数量 |
| 4 | N_ESESS_TODO | INTEGER | 当前线程池中待处理的 ESESS 请求数量 |
| 5 | N_ESESS_WAIT | INTEGER | 线程池中锁等待的线程数量 |

12.32 V\$TABLE_ROW_CNT_CACHE

显示 DMDPC 系统中各节点中的缓存表行数情况。

| 序号 | 列 | 数据类型 | 说明 |
|----|------------|---------|------|
| 1 | TAB_ID | INTEGER | 表 ID |
| 2 | ROWCNT | BIGINT | 表行数 |
| 3 | ACCESS_NUM | INTEGER | 访问次数 |

12.33 V\$DPC_TS_MOVE

查询 DMDPC 系统当前表空间迁移进度等信息。

| 序号 | 列 | 数据类型 | 说明 |
|----|---------------|-----------|---------------------------|
| 1 | TS_ID | INTEGER | 表空间 ID |
| 2 | READ_ONLY | INTEGER | 只读迁移标记。1: 只读, 0: 读写 |
| 3 | SRC_RAFT_ID | INTEGER | 表空间迁移源 RAFT_ID |
| 4 | DEST_RAFT_ID | INTEGER | 表空间迁移目标 RAFT_ID |
| 5 | BP_TS_CRT_LSN | BIGINT | 目标 BP 主库创建 TS 后的 CUR_LSN。 |
| 6 | START_TIME | TIMESTAMP | 开始迁移时间 |

DM8 分布计算集群

| | | | |
|---|---------------|--------------|---|
| 7 | DURING_TIME | BIGINT | 从开始迁移的持续的时间(秒) |
| 8 | PROGRESS_TYPE | VARCHAR(128) | <p>当前进行的操作进度类型。PROGRESS_TYPE 为字符串类型,其值包含 2 部分:[当前进度类型:进度值][当前进度序号/迁移总进度值]。</p> <p>进度类型包括:字典封锁 (LOCK_DICT)、表空间清理 (PURG_TS)、执行检查点 (CHECKPOINT)、文件拷贝 (FILE_COPY)、日志重演 (APPLY_ARCH)。</p> <p>例如:</p> <p>[FILE_COPY:1442818560/2097152000(%68)][2/6]表示处于文件拷贝阶段,已经拷贝的字节数/总字节数(完成百分比),处于 6 阶段中的第 2 阶段;</p> <p>[APPLY_ARCH:1][3/6]表示处于重演日志阶段,第 1 次重演,处于 6 阶段中的第 3 阶段</p> |

12.34 V\$DPC_QC_SCHED_HISTORY

查询 DMDPC 任务调度历史信息。

| 序号 | 列 | 数据类型 | 说明 |
|----|-------------|------------|---------------------------|
| 1 | MPP_EXEC_ID | BIGINT | 执行号 |
| 2 | STMT_ID | INTEGER | 句柄号 |
| 3 | EID | BIGINT | DPC 全局会话 ID |
| 4 | STASK_NO | INTEGER | 子任务号 |
| 5 | STATUS | VARCHAR(8) | 调度状态,取值: NONE: 未调度 |

DM8 分布计算集群

| | | | |
|---|---------|--------|--------------------------------|
| | | | SCHDLED: 已调度 COMPLETE: 调度完成 |
| 6 | EXEC_ID | BIGINT | VM 执行号 |

12.35 V\$GLOBAL_RAFT_INFO

查询 DMDPC 系统全局多副本信息。仅会打印可正常通信的 MP/BP/BS 库的多副本信息；若节点异常，则不会打印该实例信息；或若 RAFT 组内无有效主库，则对应 RAFT 组中所有节点信息均不会打印。

| 序号 | 列 | 数据类型 | 说明 |
|----|---------------|--------------|---|
| 1 | INSTANCE_NAME | VARCHAR(128) | 实例名 |
| 2 | IS_SHADOW | VARCHAR(8) | 是否是影子库 |
| 3 | IS_LEARNER | VARCHAR(8) | 是否是学习者 |
| 4 | RAFT_STAT | VARCHAR(16) | 多副本选举角色 |
| 5 | TERM_ID | INTEGER | 任期号 |
| 6 | LOG_TERM_ID | INTEGER | 日志任期号 |
| 7 | SYS_MODE | VARCHAR(16) | 系统模式 |
| 8 | SYS_STATUS | VARCHAR(16) | 系统状态 |
| 9 | ARCH_STATUS | VARCHAR(16) | 归档状态。该字段结果在 DMDPC 下，连接主库或 SP 查询有效，连接备库查询无效；在 RAFT 下，连接主库查询有效，连接备库查询无效 |
| 10 | C_SEQNO | BIGINT | 已经提交到的日志包序号 |
| 11 | C_LSN | BIGINT | 节点已经提交到的 LSN 值 |

DM8 分布计算集群

| | | | |
|----|---------|--------|------------------|
| 12 | H_SEQNO | BIGINT | 主库已经提交到的日志包序号 |
| 13 | H_LSN | BIGINT | 主库节点已经提交到的 LSN 值 |
| 14 | F_SEQNO | BIGINT | 已经刷盘的最大日志包序号 |
| 15 | F_LSN | BIGINT | 已经刷盘到的最大 LSN 值 |

12.36 V\$DPC_EDCT_FAULT_DOMAIN

查询系统内存中容错域与实例的关系。

| 序号 | 列 | 数据类型 | 说明 |
|----|-------------|--------------|--------|
| 1 | DOMAIN_ID | INTEGER | 容错域 ID |
| 2 | DOMAIN_NAME | VARCHAR(128) | 容错域名 |
| 3 | DESCRIPTION | VARCHAR(256) | 容错域描述 |

DM8 分布计算集群

版本号: V8

发版日期: 2023 年 8 月