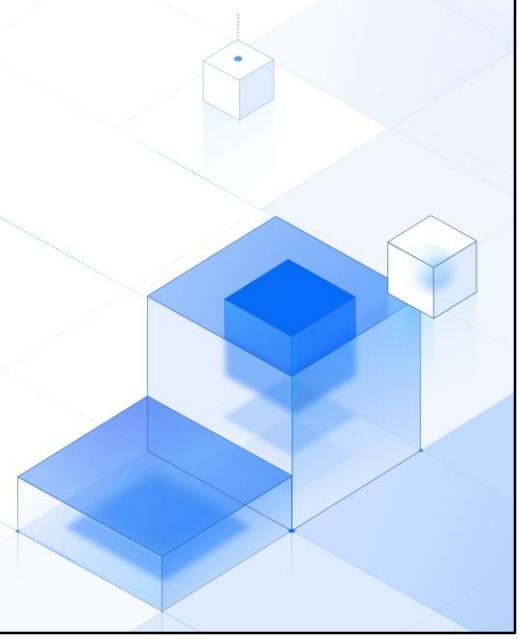
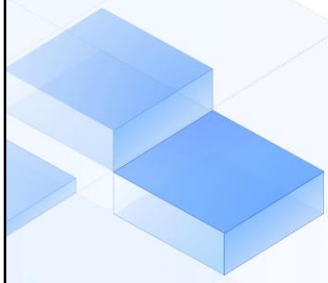
OCEANBASE

OBCP 认证培训

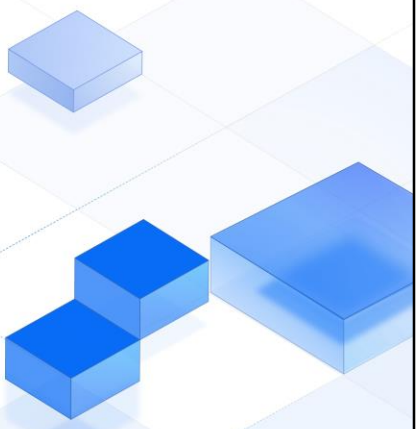


目录



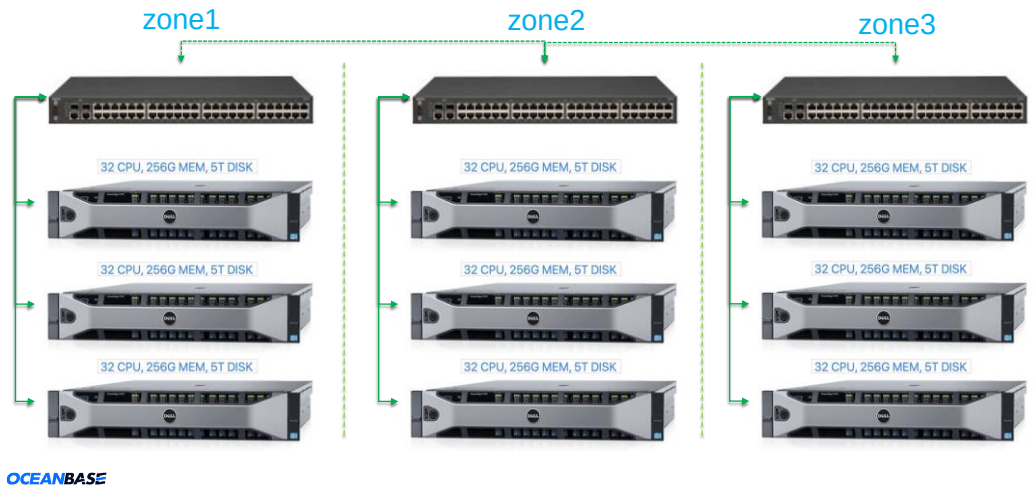
OCEANBASE

负载均衡策略



OCEANBASE

1.1 OB聚合资源的物理表示



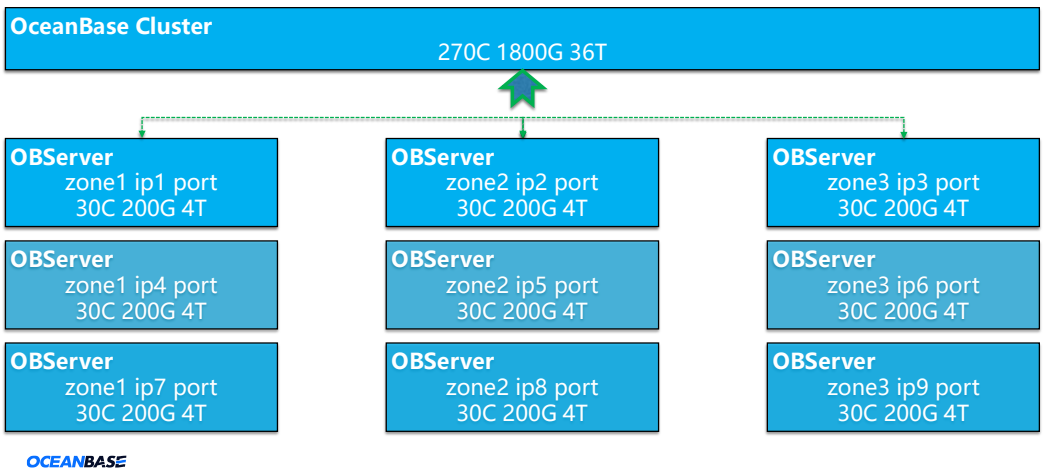
上图是9台服务器，每台资源能力是32 CPU, 256G MEM, 5T DISK,如果能聚合在一起，这个资源总和可以到288 CPU, 2304G MEM, 45T DISK。OceanBase做到这点主要靠每个节点上的数据库进程OBServer。

OBServer进程启动的时候指定了几个跟资源有关的参数。
cd /home/admin/node1/oceanbase &&
/home/admin/node1/oceanbase/bin/observer -i eth0 -P 2882 -p 2881 -z zone1 -d /home/admin/node1/oceanbase/store/obdemo -r '192.168.1.241:2882:2881;192.168.1.81:2882:2881;192.168.1.86:2882:2881' -c 20190423 -n obdemo -o "cpu_count=24,memory_limit=100G,datafile_size=200G,config_additional_dir=/data/data/1/obdemo/etc3;/data/log/log1/obdemo/etc2"

参数中cpu_count, memory_limit和datafile_size就是跟资源有关的。如果这些参数不指定，OBServer进程就会默认取主机逻辑CPU个数减去2(参数cpu_reserved默认值)、主机内存的80%(参数memory_limit_percentage默认

值)以及数据目录空间的90%(参数data_disk_usage_limit_percentage默认值)。

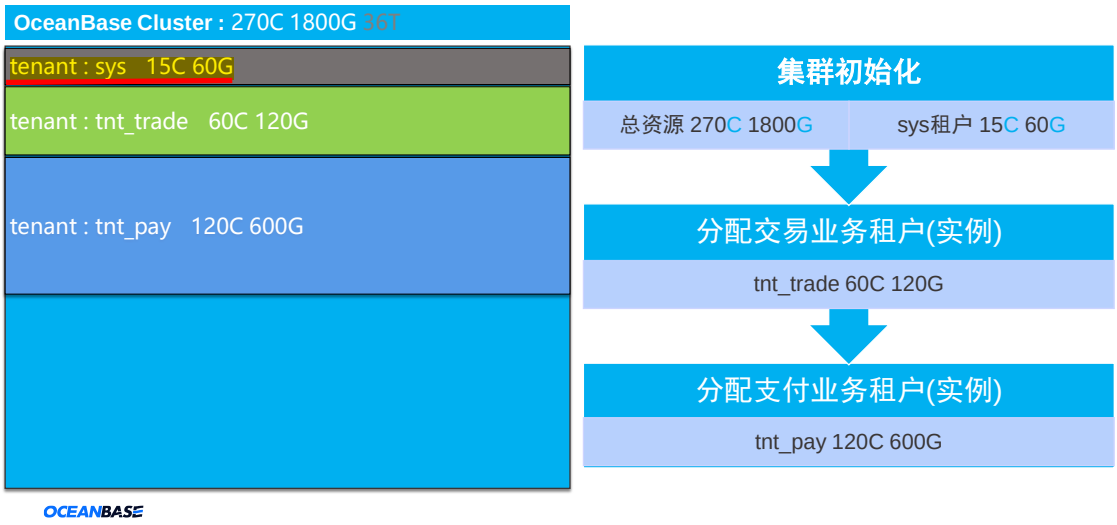
1.1 OB聚合资源的逻辑表示



OBServer启动成功后就拥有了主机的绝大部分资源。其中对内存和空间的占有是排它的，对CPU的占有是声明式的(因为除了主机OS，没有其他进程可以霸占CPU)。然后9个OBServer节点组建为一个OceanBase集群。这个集群就拥有了很大的资源(270C 1800G 36T)。

OceanBase集群对机器资源的管理是在内部的，这点不同于其他分布式数据库中间件产品的方案。OceanBase集群这种管理方式极大降低了运维成本和出错的概率。

1.1 OB资源的分配流程



OceanBase集群做了一个超级大的“资源蛋糕”，那么后面就面临一个分配的问题。通常不会把这么大的资源全部给某一个业务用。OceanBase集群实行的是按需分配。每个业务申请数据库的时候需要提供相应的业务数据量和访问量的评估，运维人员然后给出一个相应规格的评估，然后从OceanBase集群资源里切分出这个规格的资源，以租户的形式给到业务方使用。这个租户拿到手的感觉就是一个MySQL实例或者一个Oracle实例。如果是MySQL实例，业务可以在里面建库建表；如果是Oracle实例，业务可以在里面建多schema(即多用户)。

由于OceanBase集群自己也要工作，所以会保留少量资源用于内部运行，这个内部资源是一个内部租户sys。这个sys租户非常重要，是集群的管理核心所在。sys租户的管理员帐号也是整个集群的管理员帐号。从上图可以看出为两个业务分配了资源，其中交易业务租户(tnt_trade)分得60C120G,支付业务租户(tnt_pay)分得120C600G。注意，这里我略去了空间资源，因为OceanBase当前的版本不对空间的内部使用做隔离。
如果运维给的资源过小怎么办？这个不用担心，如果业务运行一段时间发现租户(实例)的性能瓶颈在资源，那么运维可以很方便的对租户资源进行扩容。当然前提是集群资源里还有足够的未分配的资源。同理，如果运维给的资源过大超出业务需求，就有点浪费。这个也可以随时对租户的资源进行相应的缩容。后

面会介绍资源的弹性伸缩原理。
上面租户资源在哪个机器上？这个就是OceanBase独特的地方，业务方不需要关心他的租户在哪个机器上。实际上这些资源有可能是在不同的机器上，而这一切都对业务透明。

不过运维人员是能看到租户资源在哪个机器上，并且还要关注这个业务租户资源的分布。因为深入的性能优化是需要了解租户资源的分布。

1.1 OB资源的分配流程

- 查看集群资源由各个节点的聚合情况

```
select zone,concat(svr_ip,':',svr_port) observer, cpu_total,cpu_assigned,cpu_assigned_percent,
round(mem_total/1024/1024/1024) mem_total_gb, round(mem_assigned/1024/1024/1024) mem_assigned_gb,
mem_assigned_percent, unit_Num,round(`load`,2) `load`, round(cpu_weight,2) cpu_weight,
round(memory_weight,2) mem_weight, leader_count
from __all_virtual_server_stat
order by zone,svr_ip
;
```

- 定义资源规格

运维人员先定义几个不同的资源规格。每个规格代表了一定的资源(包括CPU、Mem、Disk、session、IOPS）。当然实际上当前版本只对CPU和Mem资源进行限制。

```
create resource unit S2 max_cpu=20, min_cpu=20, max_memory='40G', min_memory='40G', max_iops=10000,
min_iops=1000, max_session_num=1000000, max_disk_size='1024G';

create resource unit S3 max_cpu=20, min_cpu=20, max_memory='100G', min_memory='100G', max_iops=10000,
min_iops=1000, max_session_num=1000000, max_disk_size='1024G';
;
```

OCEANBASE

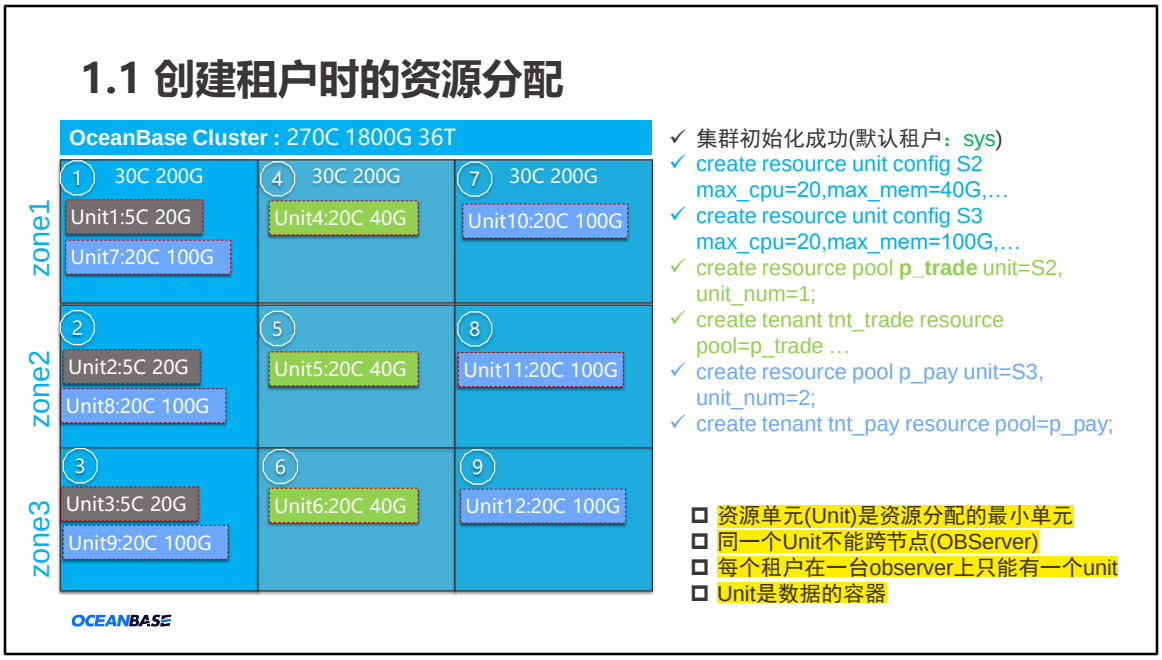
当前只有MIN_CPU，MAX_CPU，MIN_MEMORY，MAX_MEMORY产生实际作用，其他字段需要填写，但系统不做任何处理。其中MIN_CPU和MIN_MEMORY表示使用该资源配置的资源单元能够提供CPU/MEMORY的下限，以上述示例中的uc1资源配置为例，使用uc1的资源单元最低能够提供4个CPU和32G内存的服务能力。我们这里定义CPU_CAPACITY/MEM_CAPACITY为物理机的实际CPU/MEMORY值。根据定义可知某物理机上全部资源单元的MIN_CPU/MIN_MEMORY的加和值不能大于该物理机的CPU_CAPACITY/MEM_CAPACITY值。即需要满足：

Sum(min_cpu) <= CPU_CAPACITY;

Sum(min_memory) <= MEM_CAPACITY;

MAX_CPU/MAX_MEMORY的值需要对应大于等于MIN_CPU/MIN_MEMORY，MAX_CPU/MAX_MEMORY表示使用该资源配置的资源单元能够提供的CPU/MEMORY的上限，举例来说，某台物理机共有8个CPU，其上包含两个资源单元Ua，Ub；Ua和Ub的MIN_CPU/MAX_CPU分别为4和5，Ua上的实时负载使用2个CPU，由于物理机的CPU资源没有完全用尽，此时底层资源调度器可动态为Ub调高CPU服务能力，Ub的CPU服务能力上限由MAX_CPU决定(5个CPU)。MAX_CPU/MAX_MEMORY为资源配置的服务能力上限，通常称为超卖CPU和超卖MEMORY，MAX_CPU/MAX_MEMORY的加和值不能大于(CPU_CAPACITY/MEM_CPAPCITY)*resource_hard_limit，即需要满足：

Sum(max_cpu) <= CPU_CAPACITY * resource_hard_limit;
Sum(max_memory) <= MEM_CAPACITY * resource_hard_limit;
其中resource_hard_limit为系统可配置参数，称为资源超卖百分比。
我们可以通过查找__all_virtual_server_stat表的以下几个列来查询server的资源容量，其中：
cpu_capacity列对应CPU_CAPACITY;
mem_capacity列对应MEM_CAPACITY;
cpu_total列对应CPU_CAPACITY * resource_hard_limit;
mem_total列对应MEM_CAPACITY * resource_hard_limit.



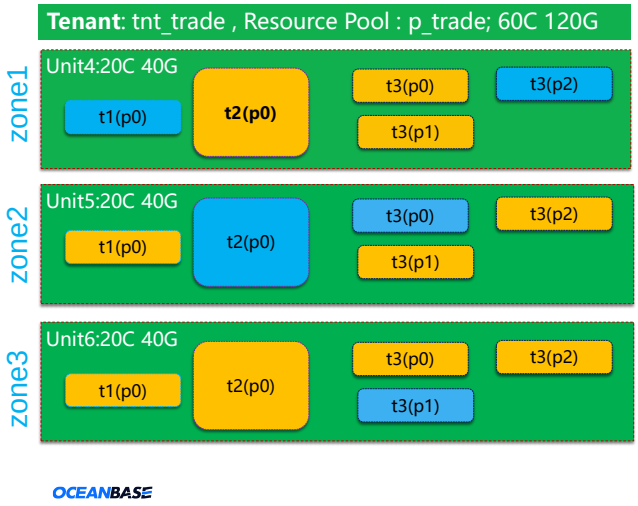
使用这两个资源单元规格分别创建两个资源池(Resource Pool)。每个Resource Pool实际是由三个资源单元(Resource Unit)组成。如下图。最后当Resource Pool创建成功后，还需要创建一个租户并关联这个资源池，租户就有了这个Resource Pool的使用权。每个Resource Pool只能关联到一个租户。

在这个过程里，Resource Unit从哪个节点上分配是有讲究的。OceanBase会考虑每个节点的资源分配率以及各个节点的负载等情况。其中有一点内存的分配策略会受参数resource_soft_limit和resource_hard_limit影响。

关于Resource Unit，它是资源分配的最小单位。同一个Unit不能跨越节点(OBServer)。上面交易租户(tnt_trade)在每个Zone里只有一个Unit，而支付租户(tnt_pay)在每个Zone里有两个Unit。在一个Zone里有多个Unit的时候，就能发挥多个节点的资源能力。换句话说并不一定是整个分布式数据库集群的机器都能为这个租户所用，这取决于Resource Pool里的unit_num的数量。

Resource Unit是数据的容器，当资源分配到位后，接下来就要看看数据在OceanBase里怎么分布。

1.1 创建租户分区表时的资源分配：租户有 1 个 unit

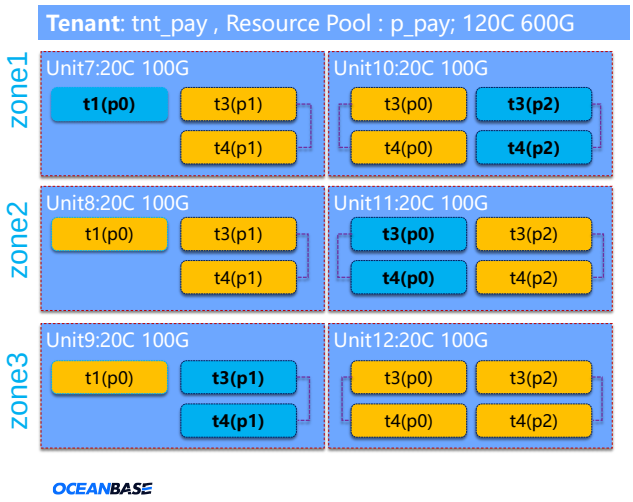


- ✓ 资源池及租户初始化成功(tnt_trade)
- ✓ create table t1(…);
- ✓ create table t2(…) primary_zone='zone2';
- ✓ create table t3(…) partition by hash （<column name>） partitions 3;

- 每个分区有三个副本，默认leader副本提供读写服务,follower副本不提供服务。
- 每个分区的三副本内容是一样的。

Unit是数据的容器，每个业务的数据只能在该业务租户相应的Resource Unit里分配出来。
如上图，是交易租户(tnt_trade)的Resource Pool，由3个Resource Unit构成。然后创建了2个普通表，1个分区表(3分区)。普通表t1只有一个分区(p0)，但是OceanBase会创建3个同样的分区，这三个称为分区(t1(p0))的三副本，三副本会分别位于三个Resource Unit里，并且必须是三个不同Zone的Resource Unit里。
三副本会有角色区分，其中蓝色底色的副本是leader副本，默认leader副本提供读写服务，另外两个follower副本不提供读写服务。所以，对于一份数据来说，其leader副本在哪个节点的Unit里，业务读写请求就落在哪个节点上。
leader副本默认位置由表的primary_zone属性控制。这个设置也可以继承自数据库以及租户(实例)的primary_zone设置。即使primary_zone设置为RANDOM，在只有三个Unit的情况下(即Resource Pool的unit_num=1的情况下)，每个leader副本会默认在第一个Zone里。除非是建表时明确指定它。
对于分区表t3而言，数据会在三个分区里，分别是t3(p0)、t3(p1)和t3(p2)。每个分区也有三副本。

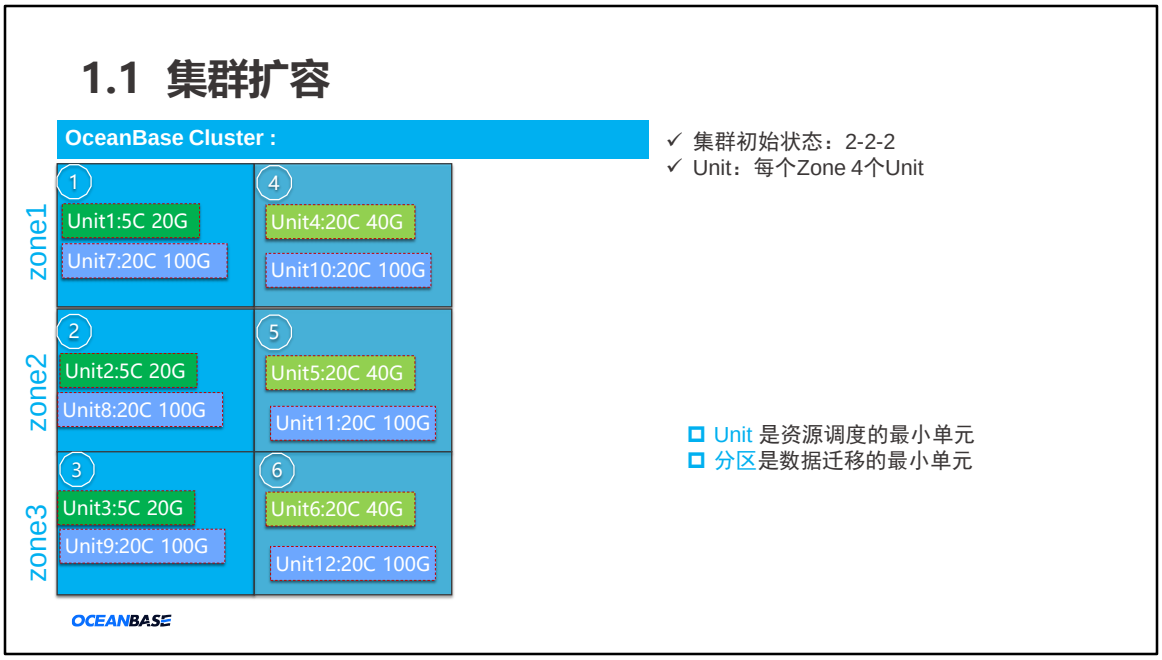
1.1创建租户分区表时的资源分配:租户有多个 Unit，表组



- ✓ 资源池及租户初始化成功(tnt_pay)
- ✓ create tablegroup tgorder partition by hash partitions 3;
- ✓ create table t1(…);
- ✓ create table t3(…) partition by hash(…) partitions 3 tablegroup=tgorder;
- ✓ create table t4(…) partition by hash(…) partitions 3 tablegroup=tgorder;

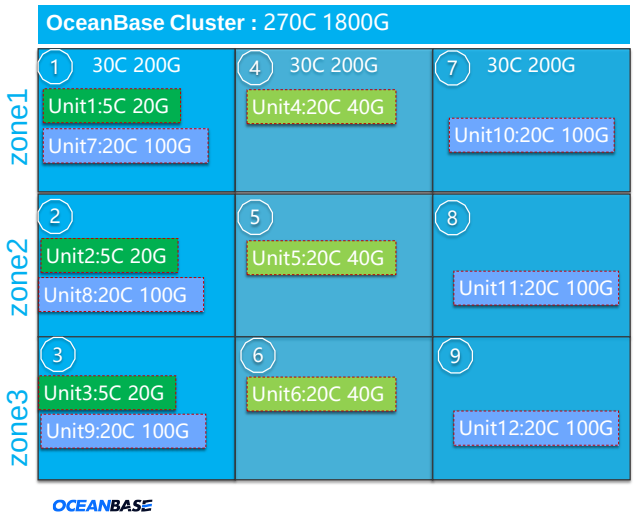
- 每个分区有三个副本，默认leader副本提供读写服务。
- 同一个分区不能跨Unit，同一个分区表不同分区可以跨Unit
- 同号分区组的分区会聚集在同一个Unit内部。

对于unit_num大于1的租户，每个分区在分配时会多一些选择，并且leader副本的位置也多一些选择。OceanBase会考虑各个unit的使用率和负载等。也就是说在建表创建分区的时候，OceanBase已经考虑到负载均衡了。
分区的分布以及leader的分布是随机的，这个对业务而言并不一定友好。在分布式数据库里，跨节点的请求时性能会有下降。
如果业务层面t3和t4关系很密切，经常做表连接查询和在同一个事务里。为了规避跨节点请求以及分布式事务，OceanBase提供了表分组(tablegroup)技术来干预有业务联系的多个表的分区分布特点。如上图示例，t3和t4都属于同一个表分组(tgorder)，则它们的同号分区属于一个分区组(partition group)，在稳定的状况下，同一个分区组的分区一定在同一个Unit内部(即在同一个节点内部)，并且它们的leader副本还要在同一个Unit内部。如图中的t3(p0)和t4(p0)有个虚线连接，就是指在同一个分区组里。



集群资源扩容就是加节点。因为每个节点(OBServer)代表了一定的资源能力。
如图是一个2-2-2的集群，总资源能力是180C1200G。当前有三个业务租户，每个Zone里有4个Resource Unit。目前分布基本平均。
集群的资源扩容就是为每个Zone增加相应的节点。
alter system add server '11.***.84.78:3882' zone 'zone1',
'11.***.84.79:3882' zone 'zone2', '11.***.84.83:3882' zone 'zone3';

1.1 OceanBase 的资源弹性伸缩与负载均衡：集群扩容



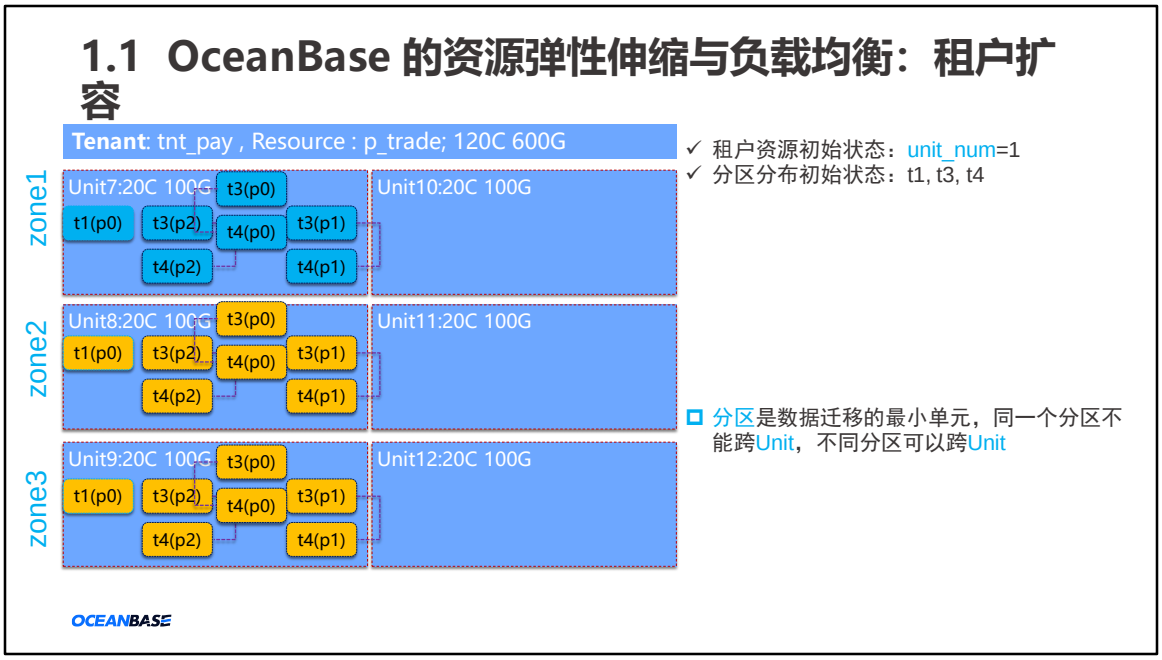
- ✓ 集群初始状态：2-2-2
- ✓ Unit：每个Zone 4个Unit
- ✓ 集群扩容：2-2-2 -> 3-3-3
- ✓ Unit增加后，分区迁移至新 Unit

- Unit 是资源调度的最小单元
- 分区是数据迁移的最小单元

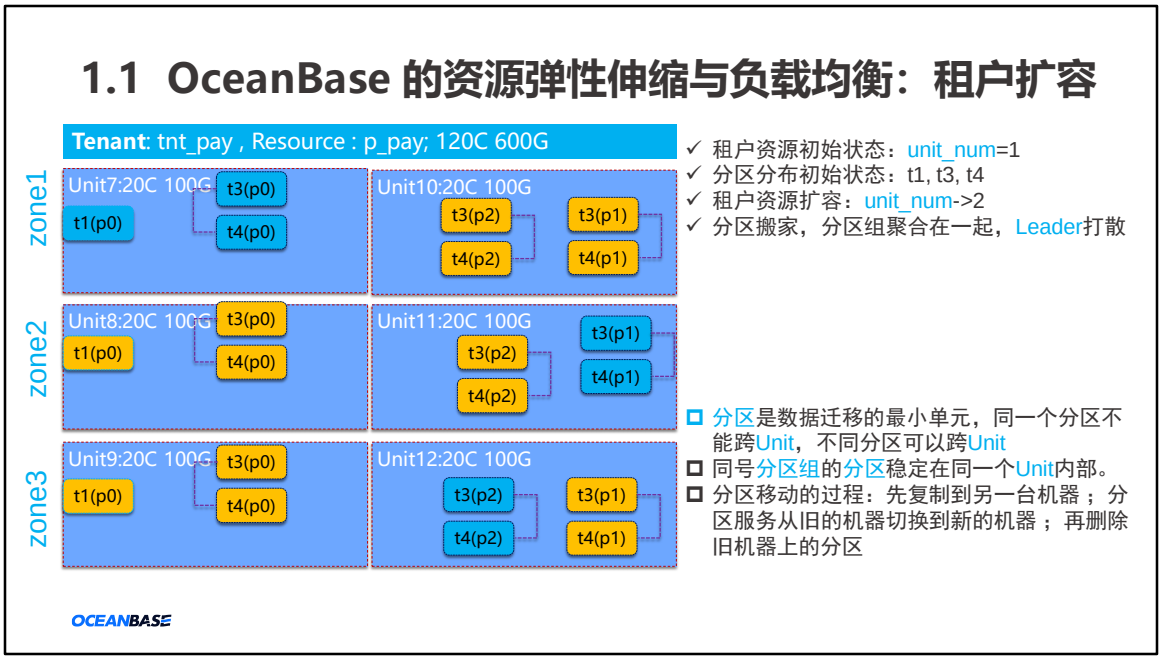
新节点加入后，由于资源利用率很低，整个集群负载就不均衡。OceanBase负载均衡的思路跟传统负载均衡产品不一样，OceanBase可以靠调整自己内部Unit的分布来间接改变各个节点的负载。因为Unit内部有数据(Partition)，其中leader副本集中的Unit就会有业务读写请求。我们可以看看发生Resource Unit迁移后的结果。

Resource Unit迁移的细节是在目标节点内部先创建Unit，然后再复制 Unit内部的Partition，最后做副本的角色切换(leader跟follower的切换)，最后下线多余的Partition和Unit。

集群资源扩容后，业务租户的Unit所在节点负载可能会下降一些。如果业务还是有性能问题，此时就要继续做业务租户的资源扩容。



租户资源的扩容是通过调整其Resource Unit的规格和数量来完成。比如说从规格S2升级到S3.
alter resource pool pool_mysql unit='S3';
或者不调整规格，而是增加Unit的数量。
alter resource pool pool_mysql unit_num=2;
上图是扩容前的租户的Resource Pool以及各个数据Partition的分布。所有leader副本默认在Zone1。



当扩容后，部分partition的位置发生变化，同时leader副本也出现在其他Zone里。

注意，在这个Partition迁移过程中，还有个特点就是同一个分区组内的Partition最终会迁移到同一个Unit内部。迁移期间有短暂时间可能出现分布在两个Unit里，可能导致有跨节点查询或者分布式事务，性能会短暂下跌一些然后很快恢复。这个是业务需要承受的。一般业务压力不大的时候是感知不到数据库性能这细小的变化。

1.1 OceanBase 的资源弹性伸缩与负载均衡相关参数

自动负载均衡相关参数

这个是通过参数enable_rebalance控制。同时为了控制负载均衡时Partition迁移的速度和影响，可以调整下面几个参数。

```
show parameters where name in
('enable_rebalance','migrate_concurrency','data_copy_concurrency','server_data_copy_out_concurrency','
server_data_copy_in_concurrency');
```

enable_rebalance: specifies whether the partition load-balancing is turned on
默认值：true
migrate_concurrency : set concurrency of migration
默认值：10
data_copy_concurrency:the maximum number of the data replication tasks
默认值：20
server_data_copy_in/out_concurrency: the maximum number of partitions allowed to migrate to/from the server
默认值：2

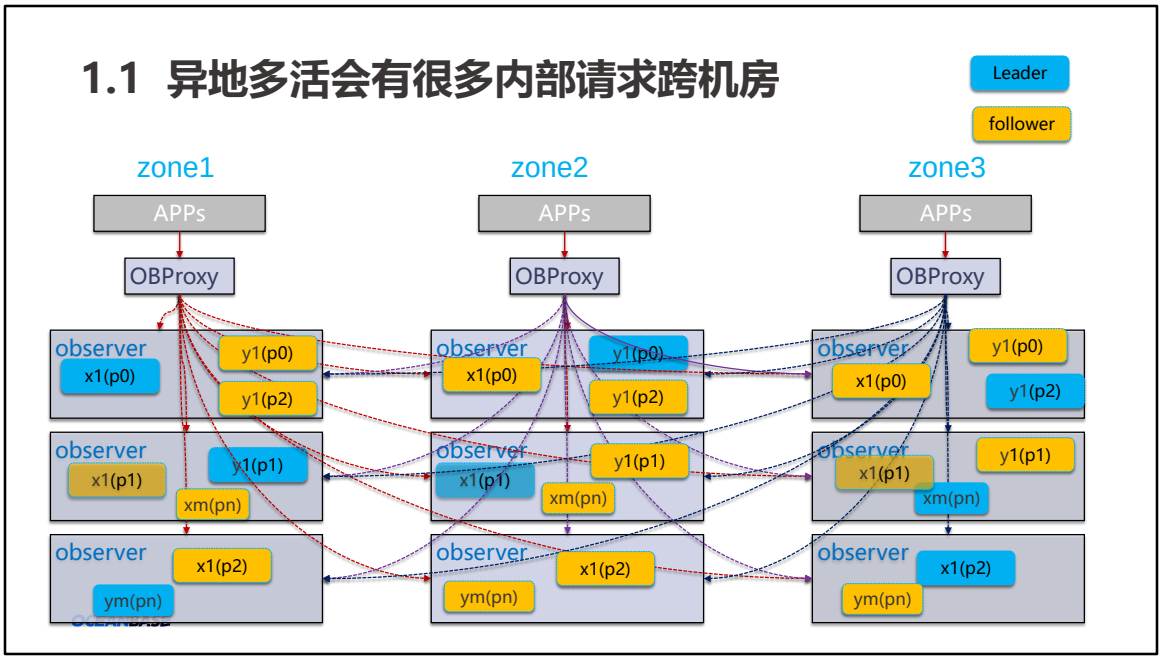
OCEANBASE

当集群资源扩容和租户资源扩容的时候，都会触发负载均衡机制。缩容同理。
在特殊的时期，比如说类似双11大促，为了防止负载均衡导致数据库性能抖动引起业务的雪崩，OceanBase还可以关闭自动负载均衡机制。
下面SQL可以查看业务租户内部所有leader副本的位置

```
select t5.tenant_name,
t4.database_name,t3.tablegroup_name,t1.table_id,t1.table_name,t2.partition_id, t2.role, t2.zone, concat(t2.svr_ip,':',t2.svr_port) observer,
round(t2.`data_size`/1024/1024) data_size_mb, t2.`row_count`
from __all_table t1 join gv$partition t2 on (t1.tenant_id=t2.tenant_id and t1.table_id=t2.table_id)
left join __all_tablegroup t3 on (t1.tenant_id=t3.tenant_id and t1.tablegroup_id=t3.tablegroup_id)
join __all_database t4 on (t1.tenant_Id=t4.tenant_id and t1.database_id=t4.database_id)
join __all_tenant t5 on (t1.tenant_id=t5.tenant_id)
where t5.tenant_id=1020 and t2.role=1
order by t5.tenant_name,
t4.database_name,t3.tablegroup_name,t2.partition_id;
```

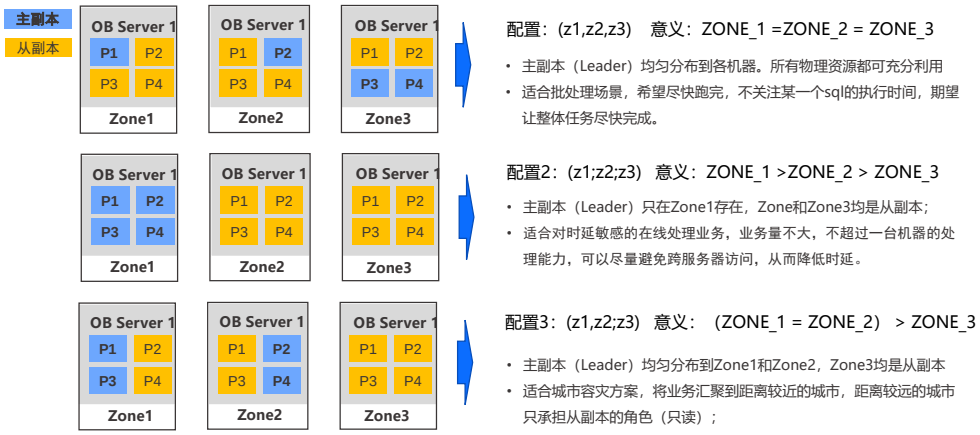
无论是自动负载均衡还是手动负载均衡，所有的Unit迁移和Partition迁移的事

件都可以在总控服务的事件日志表(__all_rootservice_event_history)里查询。



首先，分布式数据库数据分布特点通常是无规则的并且是业务无需感知的。当三机房部署时，三个Zone的业务都可以本地写数据库这个很容易做到，因为业务不关心也无从知道数据访问的内部路由。当数据的写入点在分布式数据库内部是无规则的，分布在所有机房时，会有很多内部请求跨机房，这个业务性能就很差。
所以OceanBase为业务提供一些策略用于干预leader副本分布位置。

1.1 通过Primary Zone设置优先级，适配不同业务



- 可指定ZONE提供Leader服务的优先级，如“ZONE_1,ZONE_2;ZONE_3”(ZONE_1 = ZONE_2 > ZONE_3)。
- 什么场景用什么规则。比如ZONE3在另外城市？比如ZONE3性能比较差？流程图模拟，
- 每个应用流量不同，每个服务器性能不同，如何评估。

1.1 小结

- OB 的资源分配流程是先定义资源规格，再通过资源池分配给各个租户
- 对于关系密切的表，可以通过表组（tablegroup）干预它们的分区分布，使同号分区在同一个 Unit 内部，避免跨节点请求时降低性能。
- Unit 负载均衡：集群扩容后或缩容后，Unit 可自动在不同的 observer 之间调度。
- Partition 负载均衡：租户扩容或缩容后，分区可在租户不同的 unit 内调整，使得 unit 单元的负载比较均衡。
- 管理员可以通过设置PrimaryZone影响主副本的分布。

OCEANBASE


感谢学习
OCEANBASE