快与稳的平衡
——分布式数据库研发运营

UCloud Robert

UDDB 介绍

UDDB（UCloud Distributed Database）是UCloud于2016年推出的公有云分布式关系型数据库服务。

UDDB全面兼容MySQL语法，支持数据水平切分、在线平滑扩容、透明读写分离，可有效解决业务在单机MySQL上遭遇的容量和性能问题。

分布式存储

弹性扩容

MySQL兼容

零维护

分布式事务

三大矛盾

高可靠和高可用

VS

快速响应

功能点多

VS

BUG&故障少

MySQL100%兼容

VS

技术的不确定性

快与稳的矛盾

快：
追求效率，快速推进

快是响应客户需求、解决客户问题的最好办法

快是应对不确定性的唯一办法

快是应对竞争的最好办法

VS

稳：
扎扎实实，稳健可信赖

存储产品需要的高可靠和高稳定性

2B业务面对客户时，需要高度的专业性，承诺的事情一定要做到

平衡

快和稳兼得，是做项目理想的境界。把握得好：
稳 能支持 快
快 能体现 稳

但现实是残酷的。把握不好：
快 会破坏 稳
稳 会拖累 快

对于创业型团队，在时间和资源有限的情况下，快和稳难以兼得，应该尽力去谋求快和稳的一个平衡：
有些问题的解决，需要侧重稳；
有些问题的解决，需要侧重快。

我们的实践

独立思考：UDDB架构选型

快速迭代，有序演进：计算节点构建

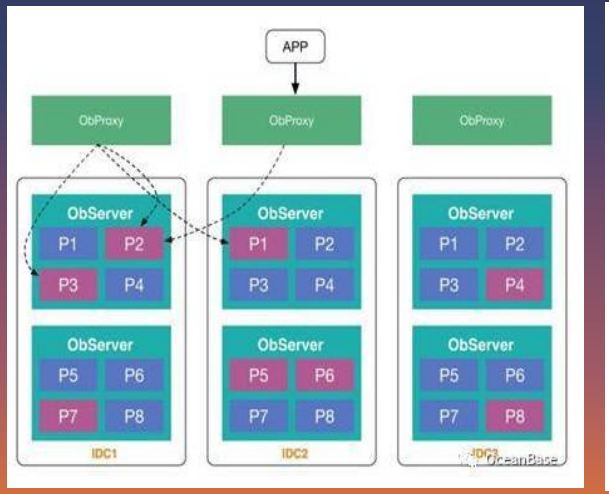
大系统小做：平滑扩容的数据迁移系统设计

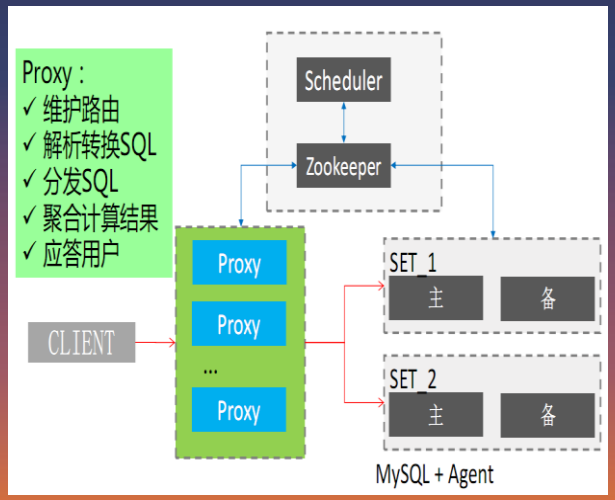
工具先行：现网运维如何兼顾稳定和效率

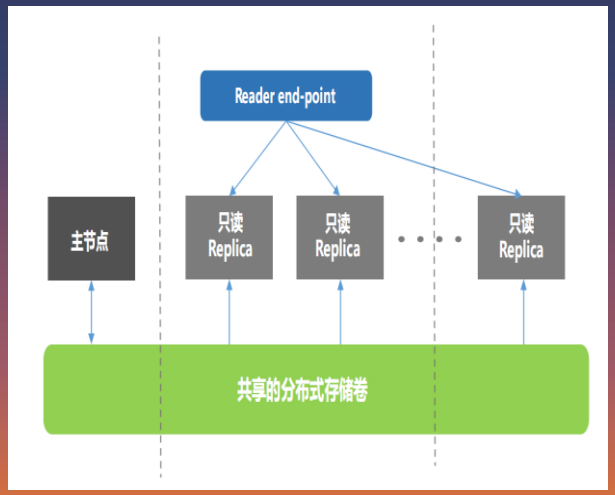
架构选型

架构选型

分布式数据库的三种典型架构：





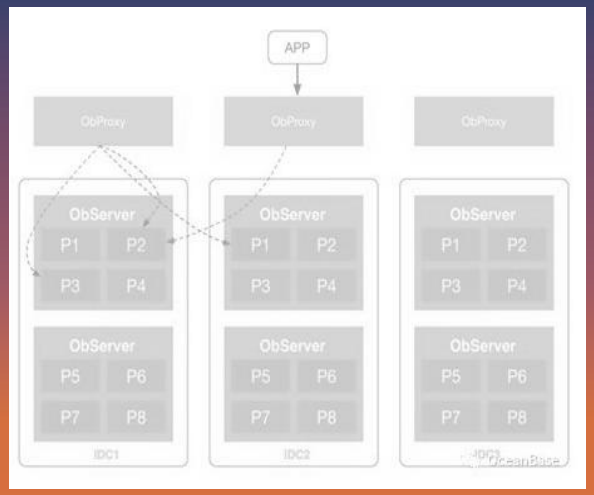


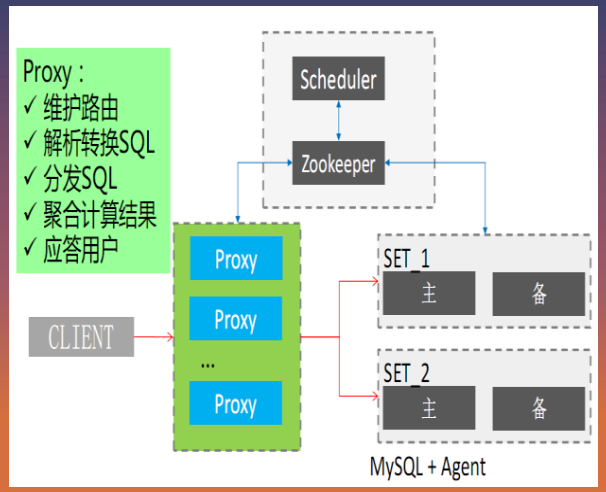
Shared-Nothing：
左：OceanBase（NewSQL)、
右：TDSQL(Proxy+MySQL)

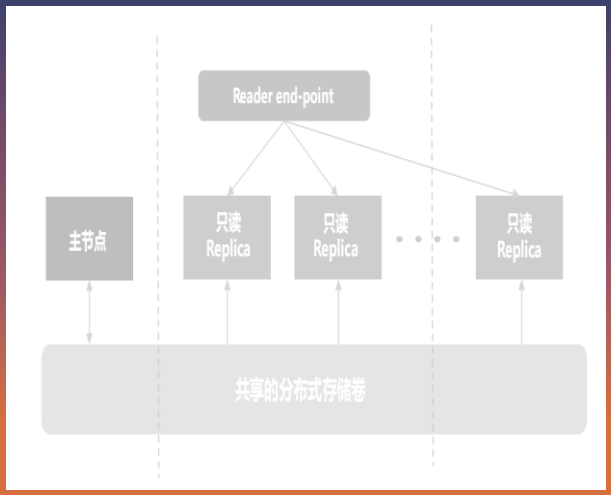
Shared-Disk:
AWS Aurora

架构选型

选择Proxy + MySQL架构：







冷静看待NewSQL

NewSQL能做的， Proxy + MySQL也能做：

考察点	NewSQL	Proxy+MySQL
架构	计算+存储双层架构	计算+存储双层架构
水平拆分	透明拆分（TIDB） 水平分区表（OceanBase）	水平分区表（OceanBase）
弹性扩容	添加节点 + 分片迁移	添加节点 + 自动化数据迁移
分布式事务	两阶段提交	两阶段提交
分布式join	分布式执行计划+操作符下推	分布式执行计划+操作符下推
三副本强一致	Raft、Paxos	Raft、Paxos
OLTP&OLAP混合	OLTP+简单的OLAP支持	OLTP+简单的OLAP支持

冷静看待NewSQL

NewSQL的优点：

1.每行代码都重新编写，抛弃老产品（MySQL/PG）的历史包袱（数据拆分、水平扩展、容灾等），做到更好的用户体验，性能问题也更能得以把控

2.新的架构，可以充分发挥新硬件的能力（比如大内存、读密集型SSD等）。

NewSQL的缺点：

1.现有NewSQL的架构，不可能成为分布式数据库（OLTP+OLAP）终极方案：

请写一条sql，统计出过去一年中，在淘宝上购买过服装的女性用户的平均年龄?

2.开发成本高、周期长、需要大量客户业务来打磨产品品质。

架构选型

采用MySQL作为存储节点:

1.MySQL可靠、稳定。 计算节点无状态，无需担心数据可靠性问题。

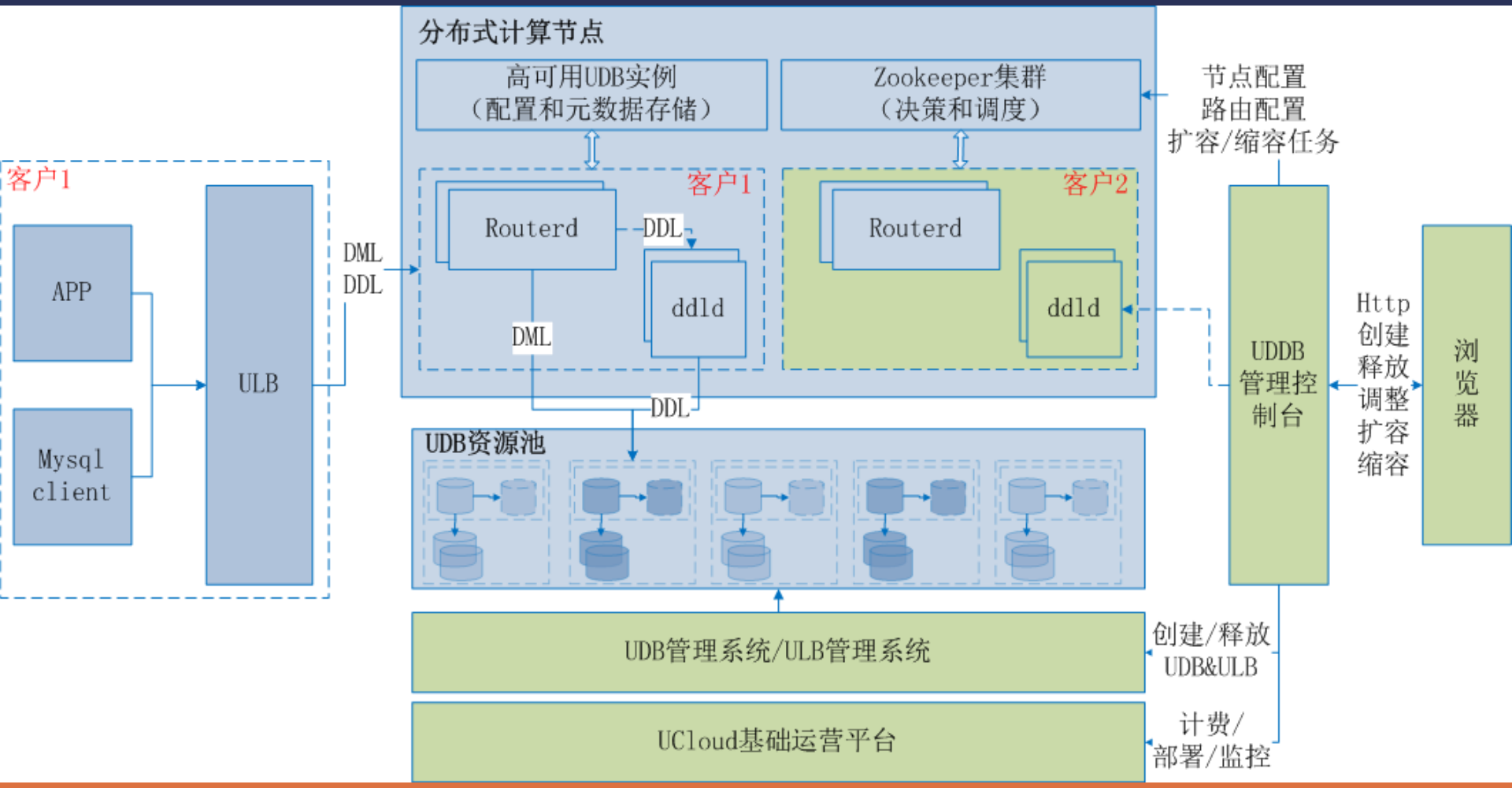
2.团队得以将精力集中在计算节点构建上。可以快速启动，快速迭代。

未来：

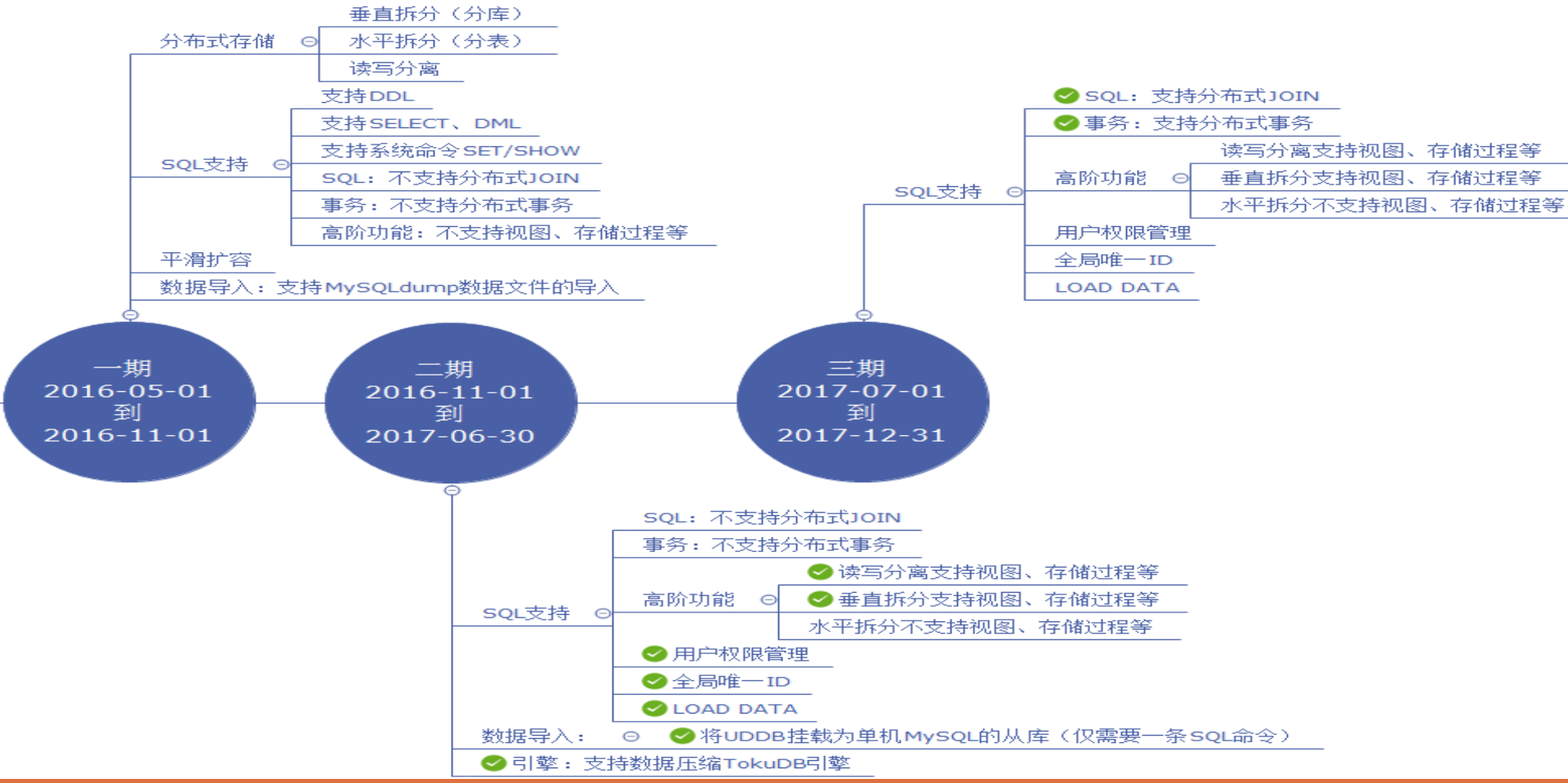
1. 相信MySQL（ MariaDB、Percona ）社区的力量。MySQL开源社区在单机数据库已取得巨大成功，面对分布式问题，未来必然也会有智慧和产品的输出。

2.拥抱社区，融入社区。围绕MySQL技术和源码，积极探索分布式的方案，积极进行沟通和分享。

系统架构

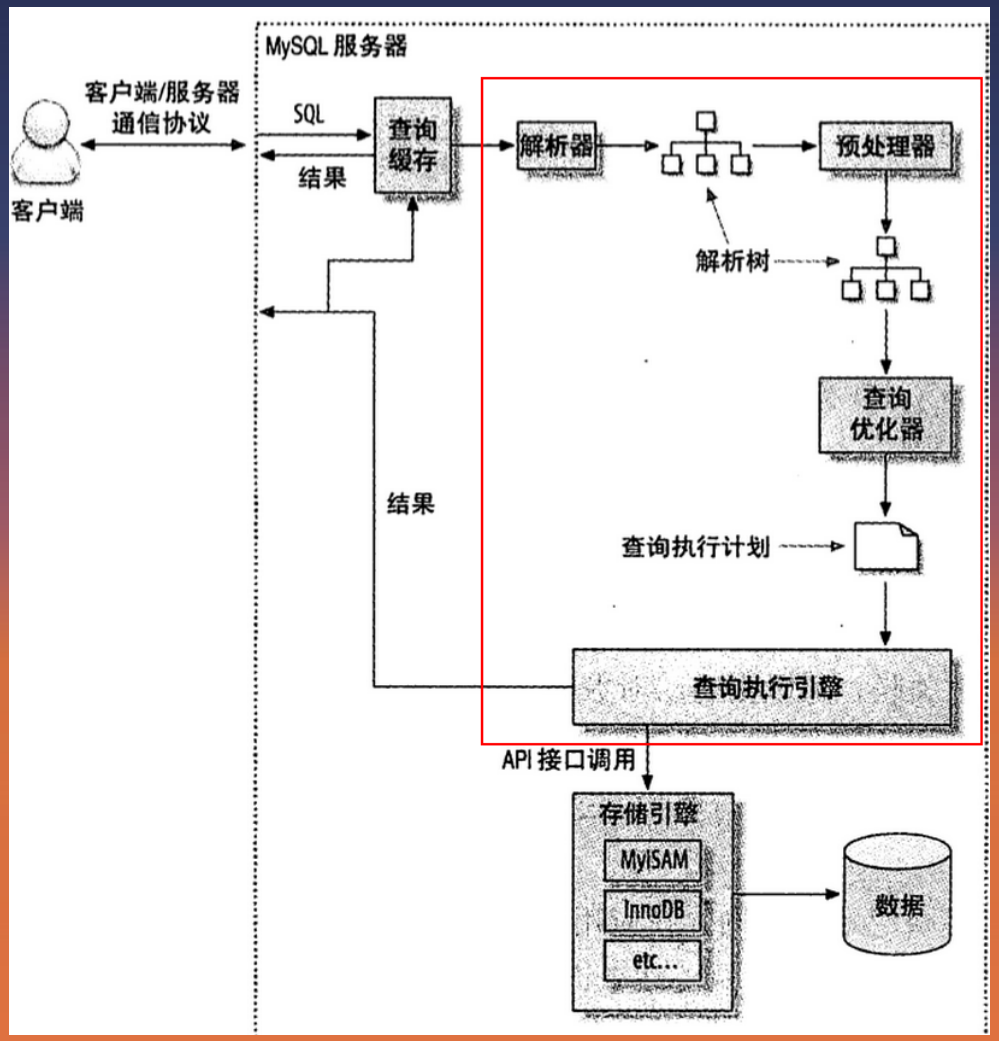

分布式计算节点
高可用UDB实例
（配置和元数据存储）
Zookeeper集群
（决策和调度）
节点配置
路由配置
扩容/缩容任务
客户1
APP
Mysql
client
ULB
DML
DDL
客户1
Routerd
DDL
ddld
DML
客户2
Routerd
ddld
DDL
UDB资源池
UDDB
管理控
制台
Http
创建
释放
调整
扩容
缩容
浏
览
器
UDB管理系统/ULB管理系统
创建/释放
UDB&ULB
UCloud基础运营平台
计费/
部署/监控

功能演进


一期
2016-05-01
到
2016-11-01
分布式存储
垂直拆分（分库）
水平拆分（分表）
读写分离
SQL支持
支持DDL
支持SELECT、DML
支持系统命令SET/SHOW
SQL：不支持分布式JOIN
事务：不支持分布式事务
高阶功能：不支持视图、存储过程等
平滑扩容
数据导入：支持MySQLdump数据文件的导入
二期
2016-11-01
到
2017-06-30
SQL支持
SQL：不支持分布式JOIN
事务：不支持分布式事务
高阶功能
读写分离支持视图、存储过程等
垂直拆分支持视图、存储过程等
水平拆分不支持视图、存储过程等
用户权限管理
全局唯一ID
LOAD DATA
数据导入：
将UDDB挂载为单机MySQL的从库（仅需要一条SQL命令）
引擎：支持数据压缩TokuDB引擎
三期
2017-07-01
到
2017-12-31
SQL支持
SQL：支持分布式JOIN
事务：支持分布式事务
高阶功能
读写分离支持视图、存储过程等
垂直拆分支持视图、存储过程等
水平拆分不支持视图、存储过程等
用户权限管理
全局唯一ID
LOAD DATA

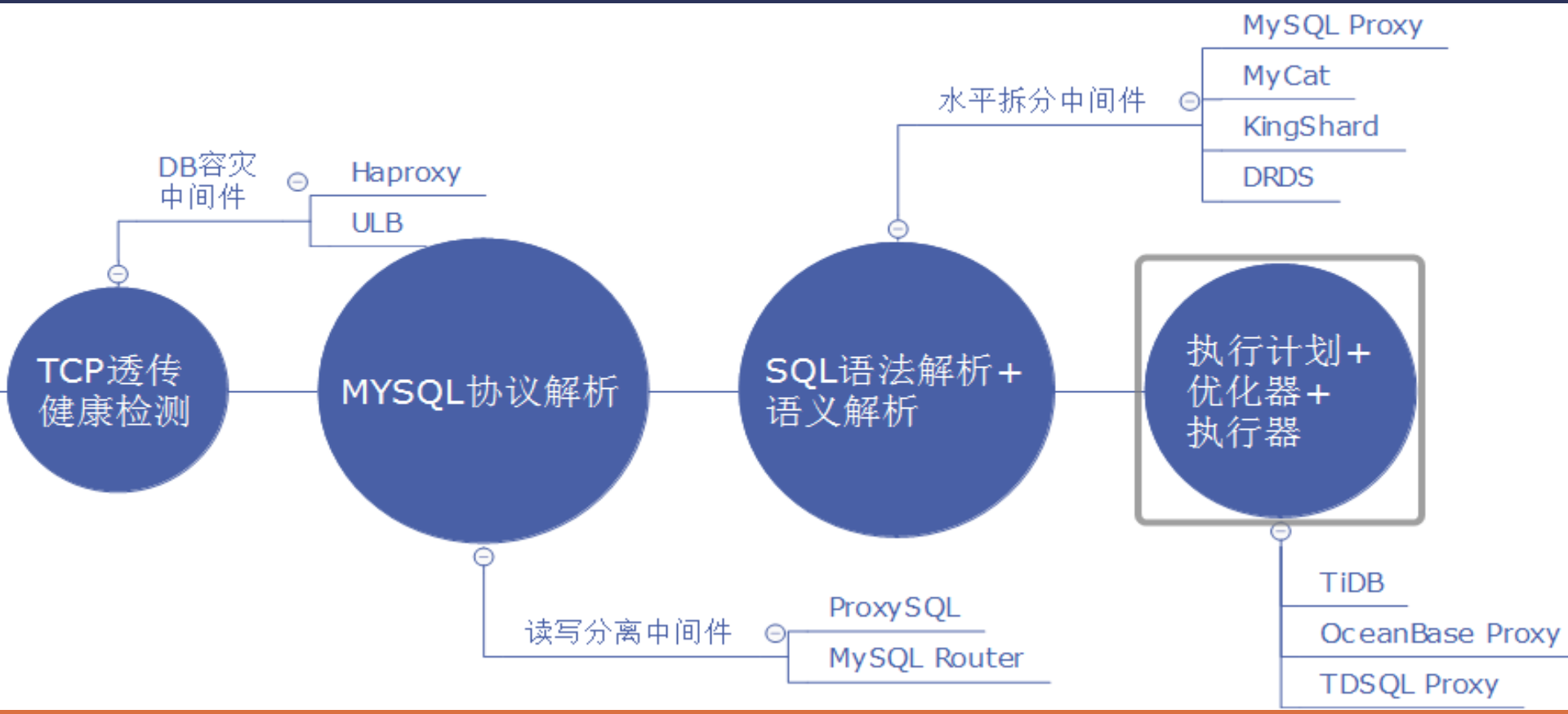
计算节点构建

需求



1. 分布式数据库的计算节点，必须具备SQL解析、计划优化、计划执行等所有功能。
2. MySQL SQL层代码：65W行，24MB

渐进式方案


TCP透传
健康检测
DB容灾
中间件
Haproxy
ULB
MYSQL协议解析
读写分离中间件
ProxySQL
MySQL Router
SQL语法解析+
语义解析
水平拆分中间件
MySQL Proxy
MyCat
KingShard
DRDS
执行计划+
优化器+
执行器
TiDB
OceanBase Proxy
TDSQL Proxy

开源中间件分析

开源中间件	功能	优点	缺陷	开源协议
MySQL	SQL解析	完整的MySQL语法解析能力 代码成熟高	语义分析模块无法复用	GPL
MyCat	SQL解析、水平拆分、读写分离	社区活跃	非独立的MySQL语法解析器	GPL
Atlas	SQL解析、水平拆分、读写分离	高性能	非独立的MySQL语法解析器	GPL
KingShard	SQL解析、水平拆分、读写分离	独立的SQL语法解析器 Go语言开发效率高		Apache2.0
Spider	水平拆分 MySQL 100%兼容	MySQL SQL层代码可以完全复用	性能太差、不成熟、没人维护	GPL
TiDB	水平拆分 分布式事务 分布式Join	独立的MySQL语法解析器，其最为完整 Go语言开发效率高	代码不成熟	GPL

基于：能力全面、有突出特长、无明显短板

的选择标准，选择KingShard作为原型

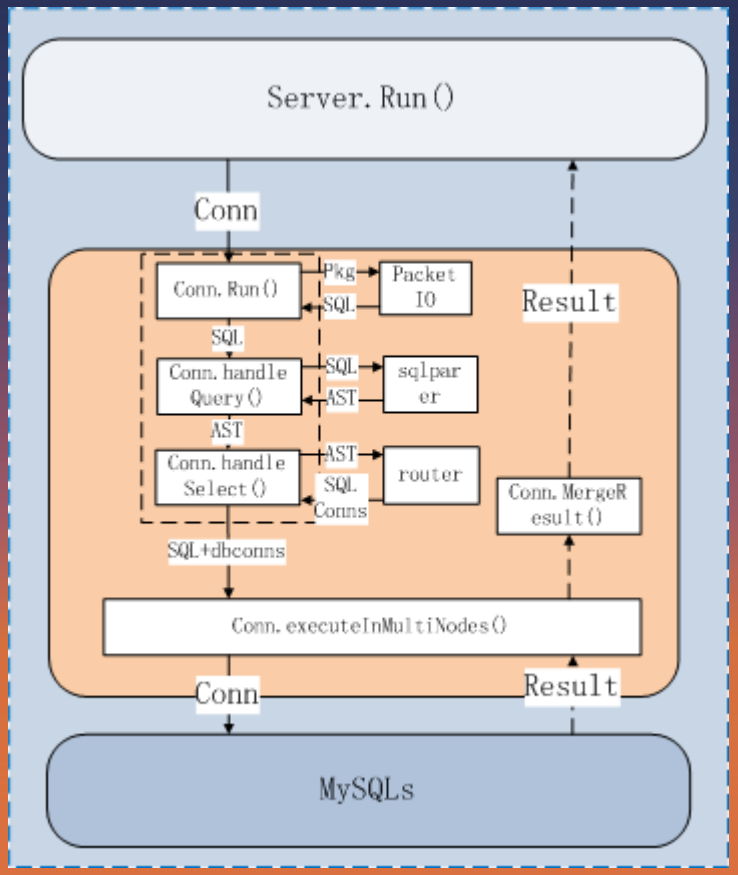
KingShard的问题

不支持DDL，分表规则靠配置

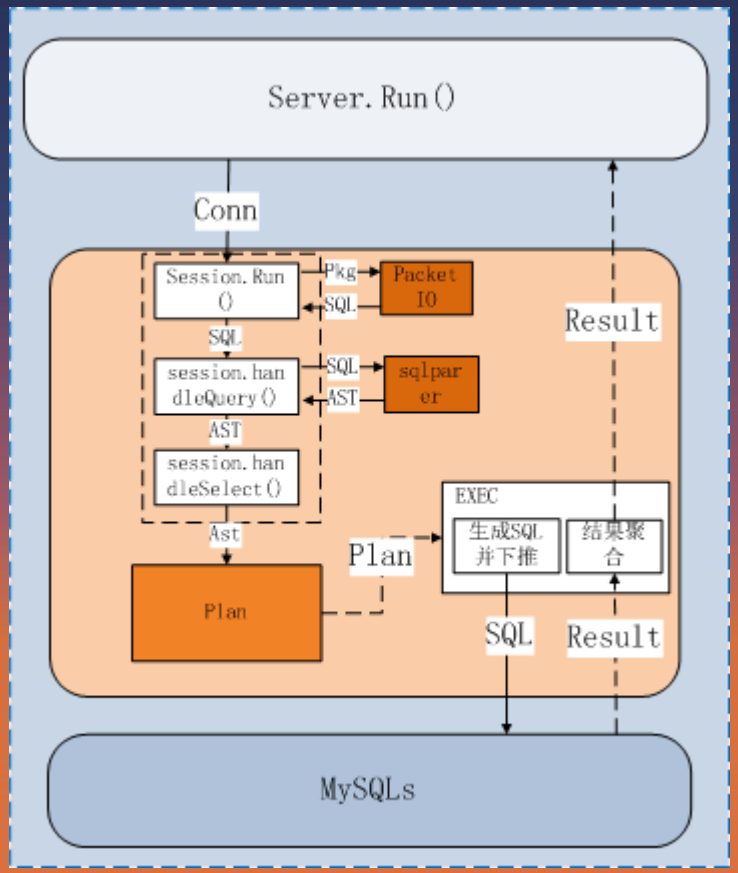
无分布式计划执行能力，只能支持简单的SQL路由

无法对系统进行平滑扩容

KingShard重构



KingShard内部架构



UDDB计算节点内部架构

KingShard重构

1. 参考数据库内核架构，设计了标准的语法解析、执行计划生成、执行计划优化、计划执行器模块。
2. 完善KingshardSQL解析模块；从零开始，开发了DDL解析和处理模块；最终SQL解析代码从不到3000行增加到1.7w行。
3. 最终代码为7.2w行。KingShard 3w行代码只复用了不到1/3：

   MySQL协议解析（mysql）：2255行
   MySQL连接池（backend）：2326行
   语法解析(sqlparser):2994行
   框架+语义解析（proxy）:2000+行

聚合SQL的支持

中间件的问题

聚合SQL：

Group By、Order By、Limit

SUM、MIN、MAX、COUNT、AVG

DISTINCT

大部分数据库中间件产品，

缺少对聚合SQL完善的支持：

```
mysql> select distinct id, avg(price) from tes
t_shard_hash where id>=1 group by concat(id,na
me) order by avg(price) limit 10;
ERROR 1105 (HY000): java.lang.IllegalArgumentE
xception: all columns in order by clause shoul
d be in the selected column list!com.alibaba.d
ruid.sql.ast.expr.SQLAggregateExpr@bdbe3ecc
```

MyCAT

```
mysql> select distinct id, avg(price) from test_s
hard_hash where id>=1 group by concat(id,name) or
der by avg(price) limit 10;
95ab74f37801000][10.144.240.13:3306][shard_db1]ER
R-CODE: [TDDL-4614][ERR_EXECUTE_ON_MYSQL] Error o
ccurs when execute on GROUP 'SHARD_DB1_1464601426
539NDBHSHARD_DB1_NFBN_0004_RDS': You have an erro
r in your SQL syntax; check the manual that corre
sponds to your MySQL server version for the right
syntax to use near 'distinct `test_shard_hash`.`i
d`,`test_shard_hash`.`name`),COUNT(`test_shard_ha
sh` at line 1 More: [http://middleware.alibaba-in
c.com/faq/faqByFaqCode.html?faqCode=TDDL-4614]
```

DRDS

UCLOUD

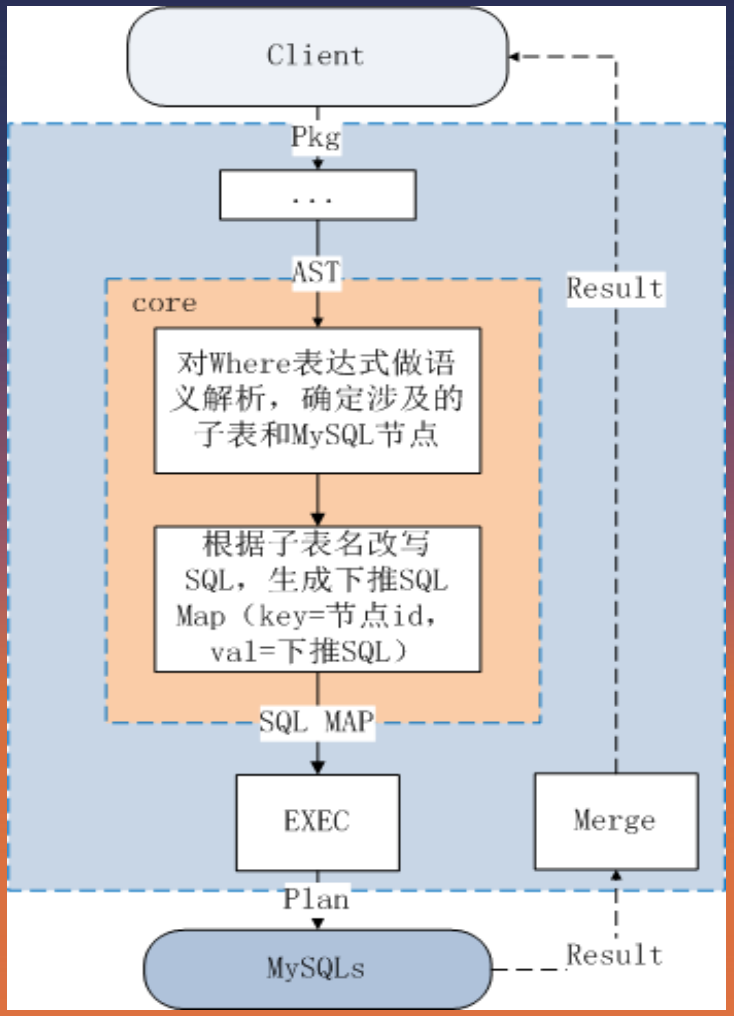
原因

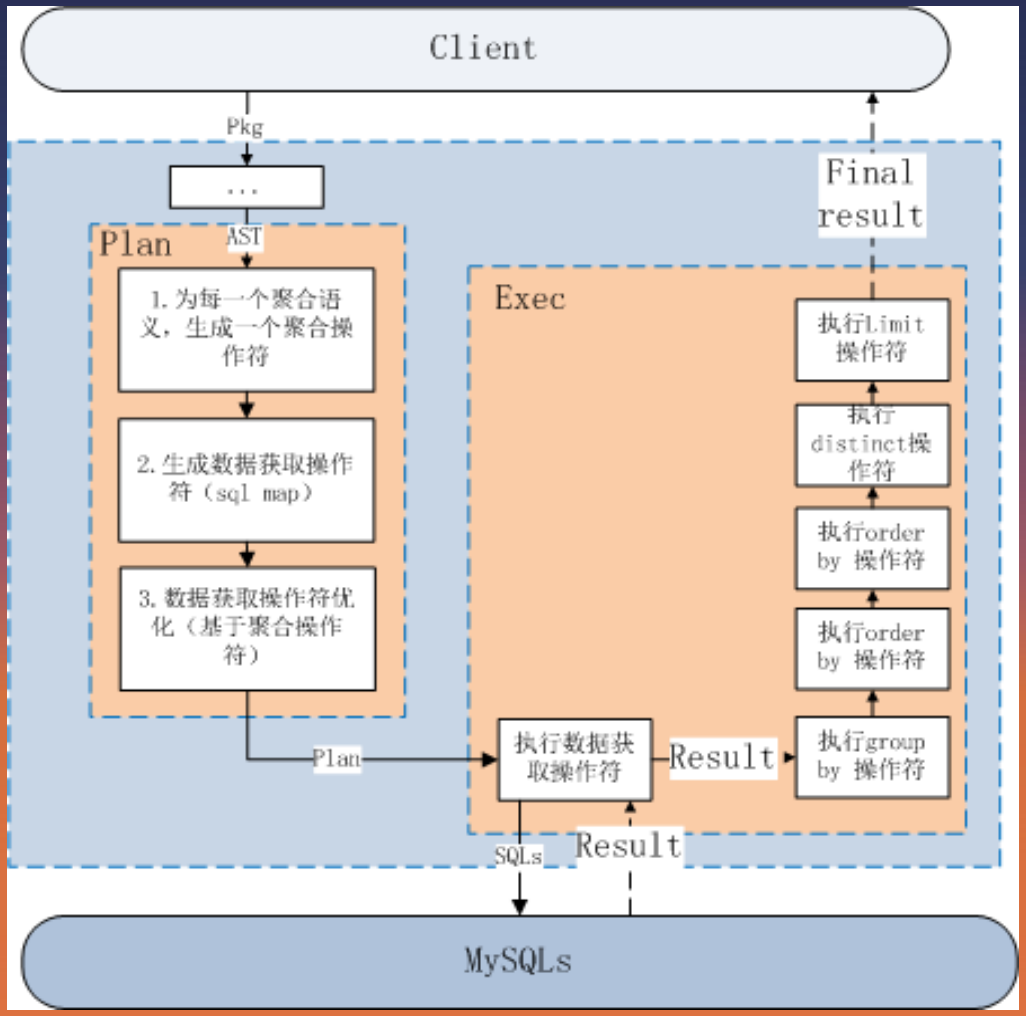
```
select distinct id, avg(price)
from test_shard_hash where id>=1
group by concat(id,name) order by avg(price) limit 10;
```

难点：

1. avg(price) 不能简单下推到MySQL， 需要转换为sum(price) + count(price) 下推，然后在计算节点重新计算
2. 计算节点需要对Group by 进行重新聚合，为此需要MySQL返回的结果集中包含id，name两个字段。但该SQL的查询表达式只含有id ，不含有name。为此需要在下推的SQL中，增加name字段

重构


Client
Pkg
...
AST
core
对Where表达式做语义解析，确定涉及的子表和MySQL节点
根据子表名改写SQL，生成下推SQL Map（key=节点id，val=下推SQL）
SQL MAP
EXEC
Plan
MySQLs
Result
Merge
Result


Client
Pkg
...
AST
Plan
1.为每一个聚合语义，生成一个聚合操作符
2.生成数据获取操作符（sql map）
3.数据获取操作符优化（基于聚合操作符）
Plan
执行数据获取操作符
Result
Exec
执行group by 操作符
执行order by 操作符
执行order by 操作符
执行distinct操作符
执行Limit操作符
Final result
SQLs
Result
MySQLs

开发效率的追求

```
CREATE TABLE `t2` (
  `id` int(11) NOT NULL,
  `price` int(11) NOT NULL,
  `name` varchar(64) DEFAULT NULL,
  `dt` varchar(32) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
PARTITION BY HASH(id)
PARTITIONS 8;

insert into t2(id,price,name,dt) values(1, 5, 'breakfast','20160830');
insert into t2(id,price,name,dt) values(2, 15, 'breakfast','20160831');
insert into t2(id,price,name,dt) values(3, 10, 'lunch','20160831');
insert into t2(id,price,name,dt) values(4, 8, 'motocar','20140101');
insert into t2(id,price,name,dt) values(5, 8, 'car','20150101');
insert into t2(id,price,name,dt) values(6, 20, 'car','20160101');

mysql> select name, min(price),max(price),avg(price) from t2 where id>=1 group by name order by max(price) desc limit 2;
+-----------+------------+------------+------------+
| name      | min(price) | max(price) | avg(price) |
+-----------+------------+------------+------------+
| car       |          8 |         20 |    14.0000 |
| breakfast |          5 |         15 |    10.0000 |
+-----------+------------+------------+------------+
2 rows in set (0.00 sec)

mysql> select distinct min(price) from t2 where id>=1 group by name order by max(price) desc ;
+------------+
| min(price) |
+------------+
|          5 |
|         10 |
|          8 |
+------------+
```

思路：

架构需要从长远进行考虑，开发应该追求效率

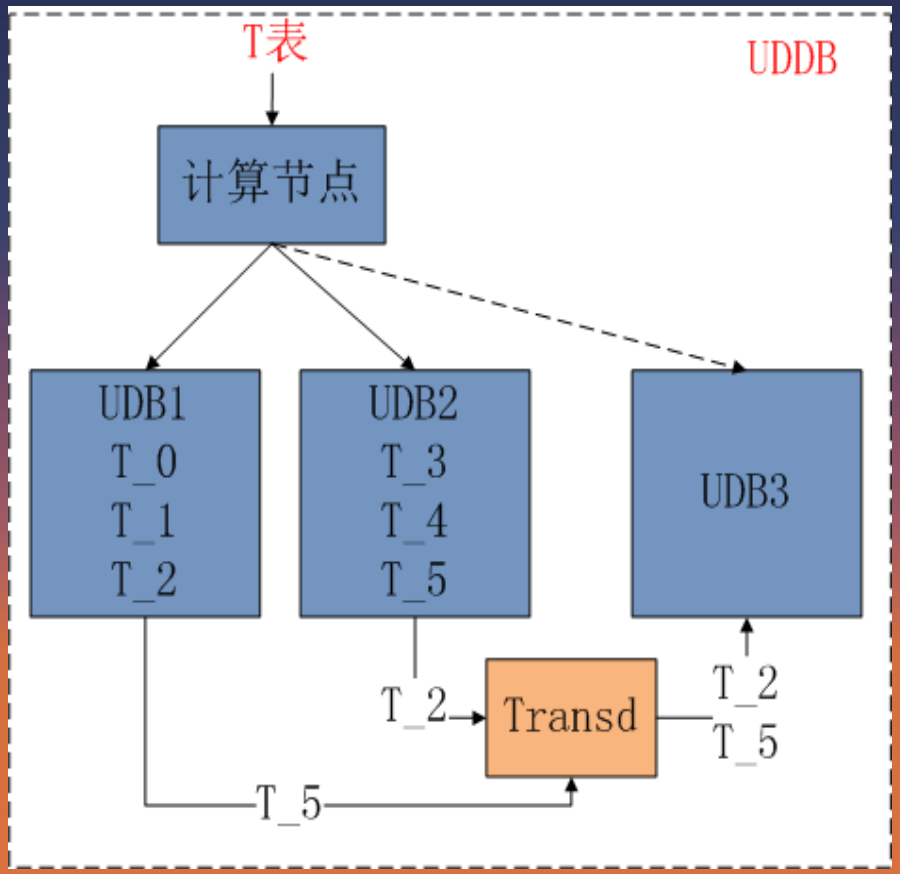
方法：基于用例的开发

1.精心构建两条测试SQL，首先将这两条SQL跑通

2.再结合自建或客户测试用例，做覆盖性测试

平滑扩容

流程



大表的分表方式：

采用Pre—Shard的方式，将大表划分为若干张子表，每个节点存储一部分子表。

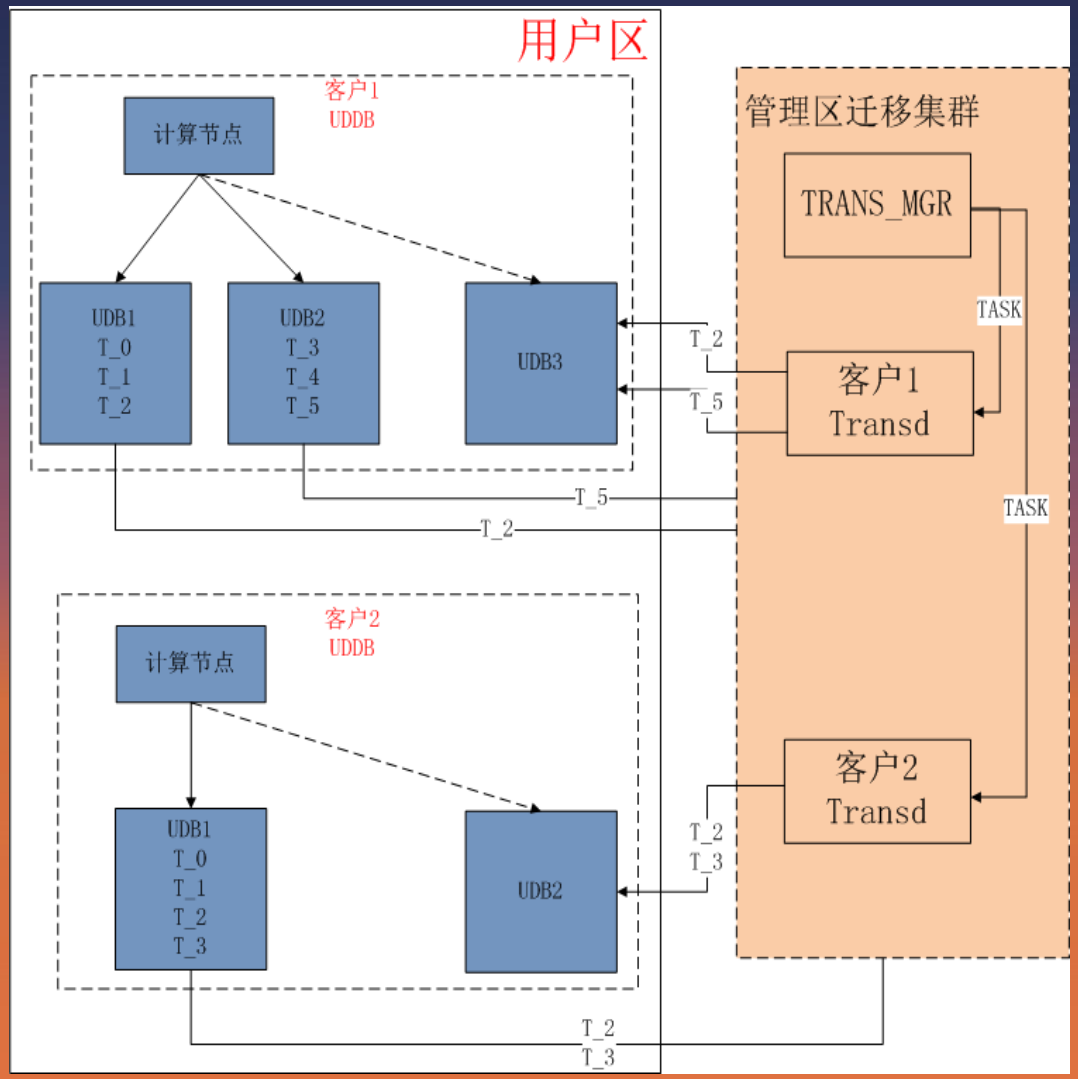
扩容流程：

1. 通过UCloud控制台添加新MySQL节点
2. 管理平台（控制台后端）发起数据迁移流程
3. 数据迁移流程将把老节点的部分子表， 迁完新节点，实现新老节点数据均衡。分布完成后重置计算节点的子表路由。

迁移原理：

- 采用全量dump + 增量同步 + 追平后重置计算节点路由的方式进行迁移。
- 需要一个第三方节点Transd， 执行以上三步操作。

问题



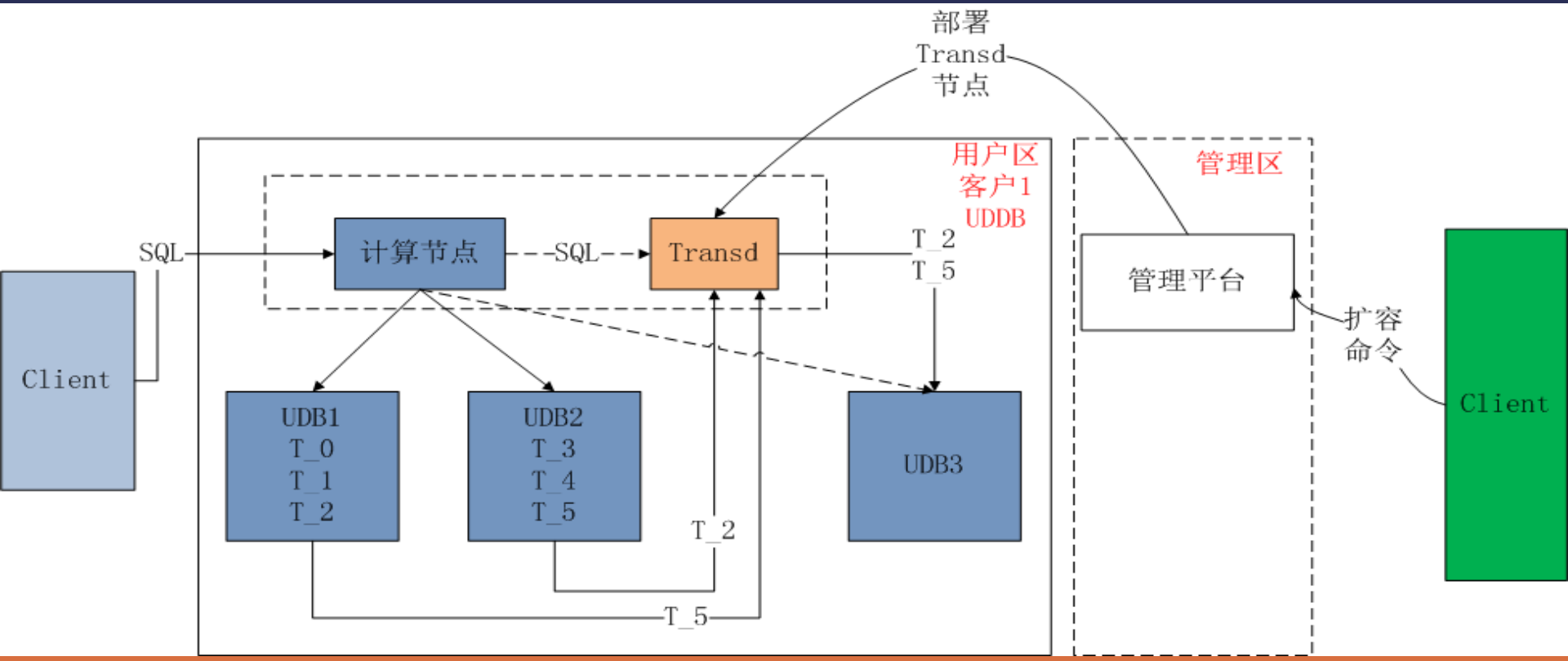
考虑一个机房内所有UDDB实例的数据迁移问题：

传统的方法，是在该机房部署一套数据迁移集群。由该集群管理机房内所有租户的UDDB实例的迁移任务。当客户通过控制台发起扩容操作时，即进行数据迁移。

问题：

1. 数据迁移集群的实现和运营复杂，涉及迁移任务的管理、资源调度等
2. 不够灵活。客户希望白天购买并添加存储节点，晚上低谷期进行数据迁移。

解决



优点：

1.化整为零。通过将迁移逻辑，下放到单个UDDB实例内部，极大简化了迁移任务管理和调度问题， 无需实现复杂的调度集群。

2.变自动化为半自动化，提供SQL给客户发起数据迁移操作，自主性和灵活性更好。

工具先行

需求

运营的两个核心要求：

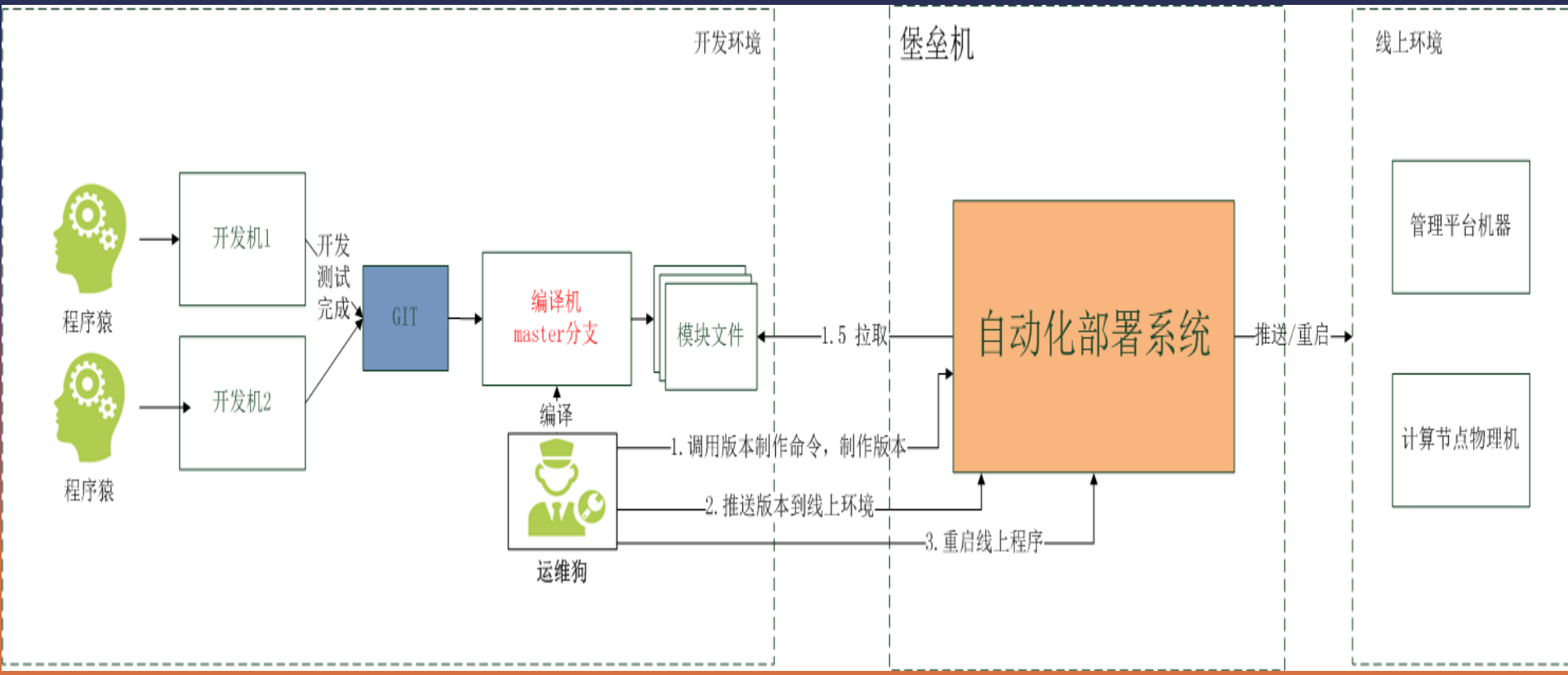
1.现网服务稳定

2.需求和问题快速响应

唯一解决之道：

自动化- 建设自动化的部署工具、监控平台、系统管理平台

自动化部署系统



1.粒度可控：

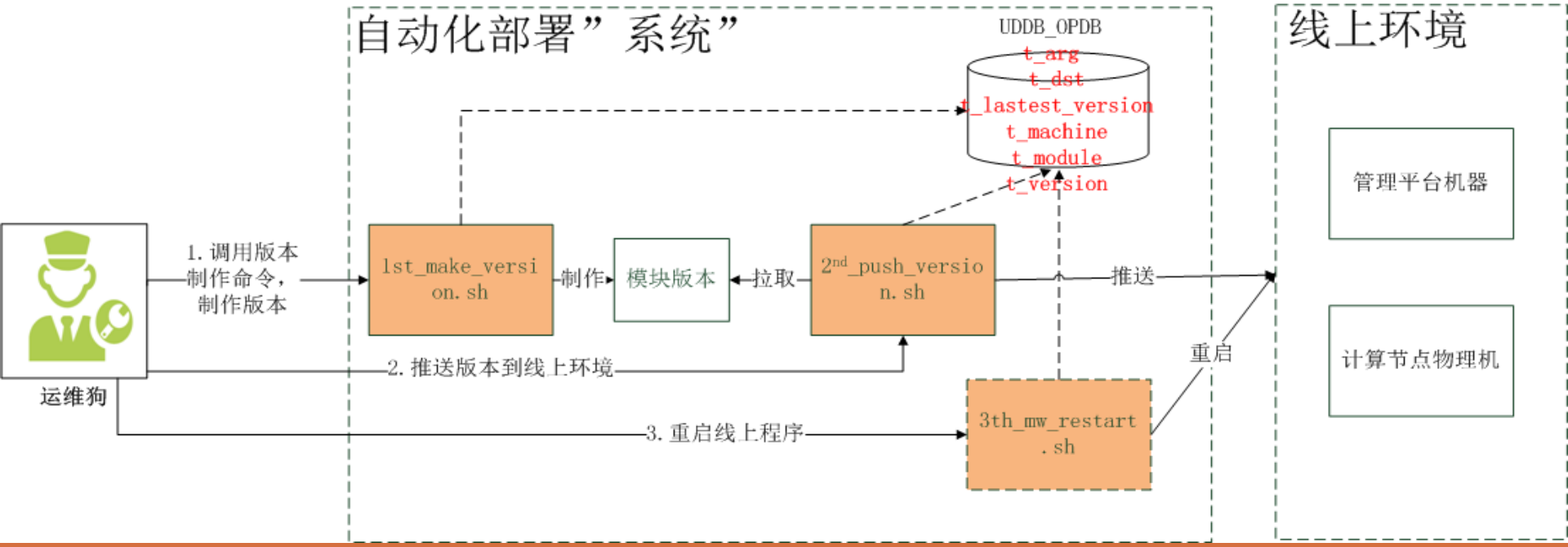
1.1可发布一个模块、某种类型的模块、某个机房全部模块

1.2提供针对一个机房、单台机器、单个UDDB实例的多种发布粒度

2.配置文件可配置：

基于配置文件模板，根据机房、机器的信息，自动生成配置文件

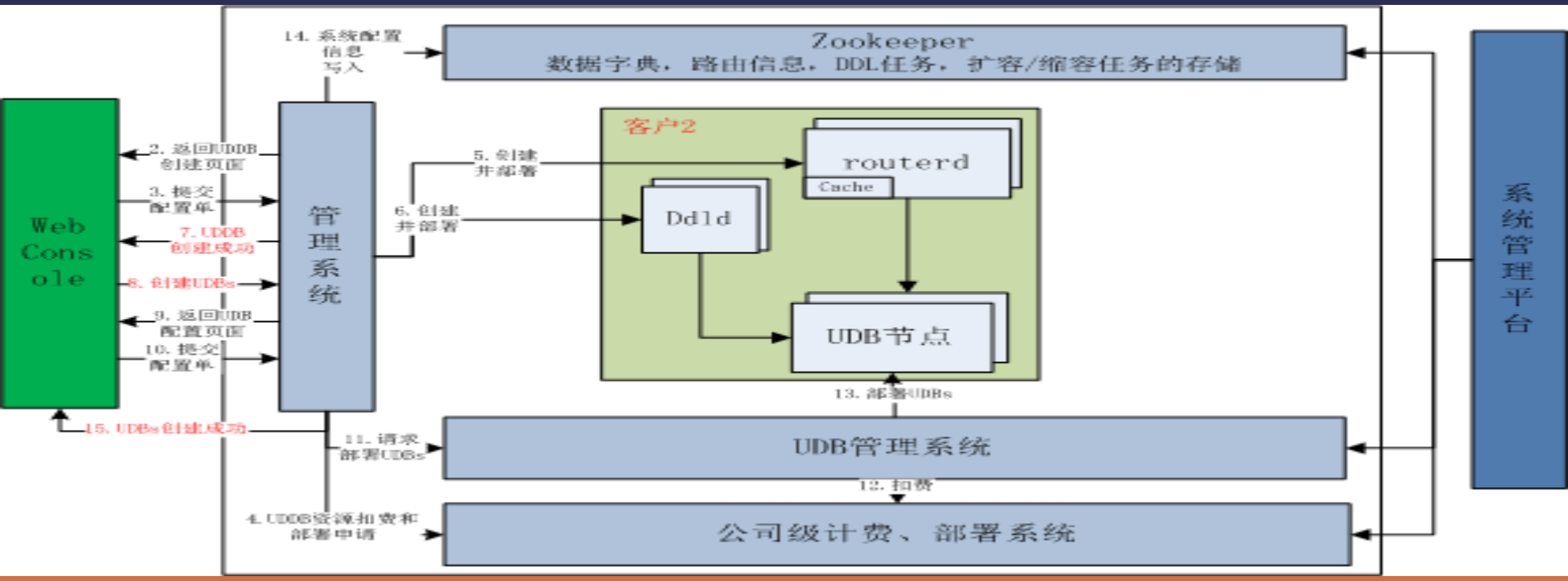
自动化部署“系统”



实现异常简单，甚至都不能称之为系统：

1. **三个脚本 + MySQL ，1周内即完成整套自动化部署工具的开发**
2. **核心在于数据库表的设计，合理的库表设计可以让逻辑异常简单；而数据库表的设计来源于对部署流程和需求的洞察。**
3. **做自动化工具并不难，难点在于如何又快又好，有力支撑线上运维**

系统管理平台



UDDB的创建、管理、删除，用户侧操作极简，但内部流程非常复杂。以创建为例，涉及到15个大步骤，N个小步骤，其中任何一个环节出错，即可能导致整个流程失败。

解决之道： 不花费高昂的成本去避免失败，而是把重点放在失败后能不能快速恢复。只要能及时进行恢复，客户也能够认可。**系统管理平台：** 创建等操作失败后，通过该平台，可以在无需客户任何干预情况下，将操作继续完成或回滚。

总结

•作为一家创业型云计算公司，无法跟巨头PK资源，只能PK智力、创造力和团队效率，靠人的主动性和技巧性存活、取胜。

•互联网的研发环境和节奏下，基础软件研发的技巧性在于：如何保证产品快且稳地向前迭代。以上是我们的一些实践和感悟，在此抛砖引玉，期待更多的交流和分享。

Thanks

Q&A