



Curve质量、监控与运维

D I G I T A L S A I L

秦 亦

网易数帆存储团队



目录

01

背景

为用户服务——质量、监控和运维

02

Curve质量控制

质量管理体系 | 测试方法论

03

Curve监控体系

总体架构 | 指标生成 | 后端采集 | 可视化展示

04

Curve运维体系

Curve运维特性 | 运维工具

Curve 是网易针对块存储、对象存储、云原生数据库、EC等多种场景自研的分布式存储系统：

- 高性能、低延迟
- 当前实现了高性能块存储，对接OpenStack和 K8s
- 网易内部线上无故障稳定运行近两年
- 已完整开源
 - [github主页](https://opencurve.github.io/)： <https://opencurve.github.io/>
 - [github代码仓库](https://github.com/opencurve/curve)： <https://github.com/opencurve/curve>

Tags	
v1.0.0	10 days ago 8b04e0e zip tar.gz Downloads
v1.1.0-beta	on 24 Sep 5d648c9 zip tar.gz Notes Downloads
v1.0.0-rc0	on 31 Aug b1b7845 zip tar.gz Downloads
v1.0.0-beta	on 17 Aug a9c30ec zip tar.gz Downloads
v0.1.4	on 7 Aug 9e347da zip tar.gz Notes Downloads

为用户服务



作为一个复杂的大型分布式存储系统，Curve 需要利用科学的方法论和专业的工具，在整个软件生命周期内更好地为用户服务：

- **质量**——向用户交付稳定可靠的软件；
- **监控**——直观地展示Curve运行状态；
- **运维**——保障Curve始终稳定高效运行。

质量

- ✓ 质量管理体系（设计、开发、review、CI）
- ✓ 测试方法论（单元测试、集成测试、系统测试）

监控

- ✓ 监控架构
- ✓ 指标采集、后端处理、可视化展示

运维

- ✓ 运维特性（易部署、易升级、自治）
- ✓ 运维工具（部署工具、管理工具）



目 录

01

背景

为用户服务——质量、监控和运维

02

Curve质量控制

质量管理体系 | 测试方法论

03

Curve监控体系

总体架构 | 指标生成 | 后端采集 | 可视化展示

04

Curve运维体系

Curve运维特性 | 运维工具

软件质量的定义是：软件与明确地和隐含地定义的需求相一致的程度。

为了确保最终交付的软件满足需求，必须将质量控制贯穿于设计、开发到测试的整个流程中。



Curve团队采用敏捷开发模式，负责人在制定迭代计划时，确认哪些任务需要设计文档：

- ❑ 小需求（改动小）将实现思路记录到任务管理系统中（JIRA），即可进行开发；
- ❑ 大需求（新模块、复杂功能）需要输出独立设计文档，并进行评审；对于功能或性能影响较大的功能，还需要进行POC验证；评审和验证通过后才能启动开发工作。



设计文档需要具备以下内容：

- 修订记录
- 审批记录
- 系统介绍
- 相关调研
- 架构
- 重要流程
- 关键算法
- 接口
- 数据库设计
- 非功能特性设计
- 参考文献

chunkserver 内存增长解决方案

- 修改记录
- 背景介绍
 - 问题描述
 - 原因阐释
 - 内存增长问题：
 - `bthread worker` 占满问题：
- 方案设计
 - 方案预期
 - 解决思路
 - 解决方案
 - 方案一：限制 `max_body_size` 和 `raft_max`
 - 方案二：`chunkserver` 慢启动
 - 方案三：限制内存大小，超过大小返回 E
 - 方案四：限制内存大小，超过大小不保留
 - 方案对比
- 方案细化
 - 启动流程
 - 当前启动流程
 - 修改后流程
 - 判断 `copyset` 加载完成
- 详细设计
 - `copyset` 加载
 - `worker` 的详细工作流程

代码编写规范



Curve代码编写规范遵循Google Style Guides (<https://google.github.io/styleguide/>)

Google C++ Style Guide

```
67  /**
68   * 同步模式读
69   * @param: buf为当前待读取的缓冲区
70   * @param: offset文件内的偏移
71   * @param: length为待读取的长度
72   * @param: mdsclient透传给底层, 在必要的时候与mds通信
73   * @return: 成功返回读取真实长度, -1为失败
74   */
75  int Read(char* buf, off_t offset, size_t length, MDSCClient* mdsclient);
```

```
127  int IOManager4File::Read(char* buf, off_t offset,
128    size_t length, MDSCClient* mdsclient) {
129    MetricHelper::IncrementUserRPSCount(fileMetric_, OpType::READ);
130    FlightIOGuard guard(this);
131
132    butil::IOBuf data;
133
134    IOTracker temp(this, &mc_, scheduler_, fileMetric_);
135    temp.SetUserDataTypes(UserDataTypes::IOBuffer);
136    temp.StartRead(&data, offset, length, mdsclient, this->GetFileInfo());
137
138    int rc = temp.Wait();
139
140    if (rc < 0) {
141        return rc;
142    } else {
143        size_t nc = data.CopyTo(buf, length);
144        return nc == length ? rc : -LIBCURVE_ERROR::FAILED;
145    }
146 }
```

[Namespaces](#) [The #define Guard](#) [Include What You Use](#) [Forward Declarations](#) [Inline Functions](#) [Namespaces](#)

[Named Namespaces and Static Variables](#) [Nonmember, Static Member, and Global Functions](#) [Local Variables](#)

[Structs](#) [Implicit Conversions](#) [Copyable and Movable Types](#) [Structs vs. Classes](#) [Structs vs. Classes](#) [Access Control](#) [Declaration Order](#)

[Write Short Functions](#) [Function Overloading](#) [Default Arguments](#) [Trailing Return Type Syntax](#)

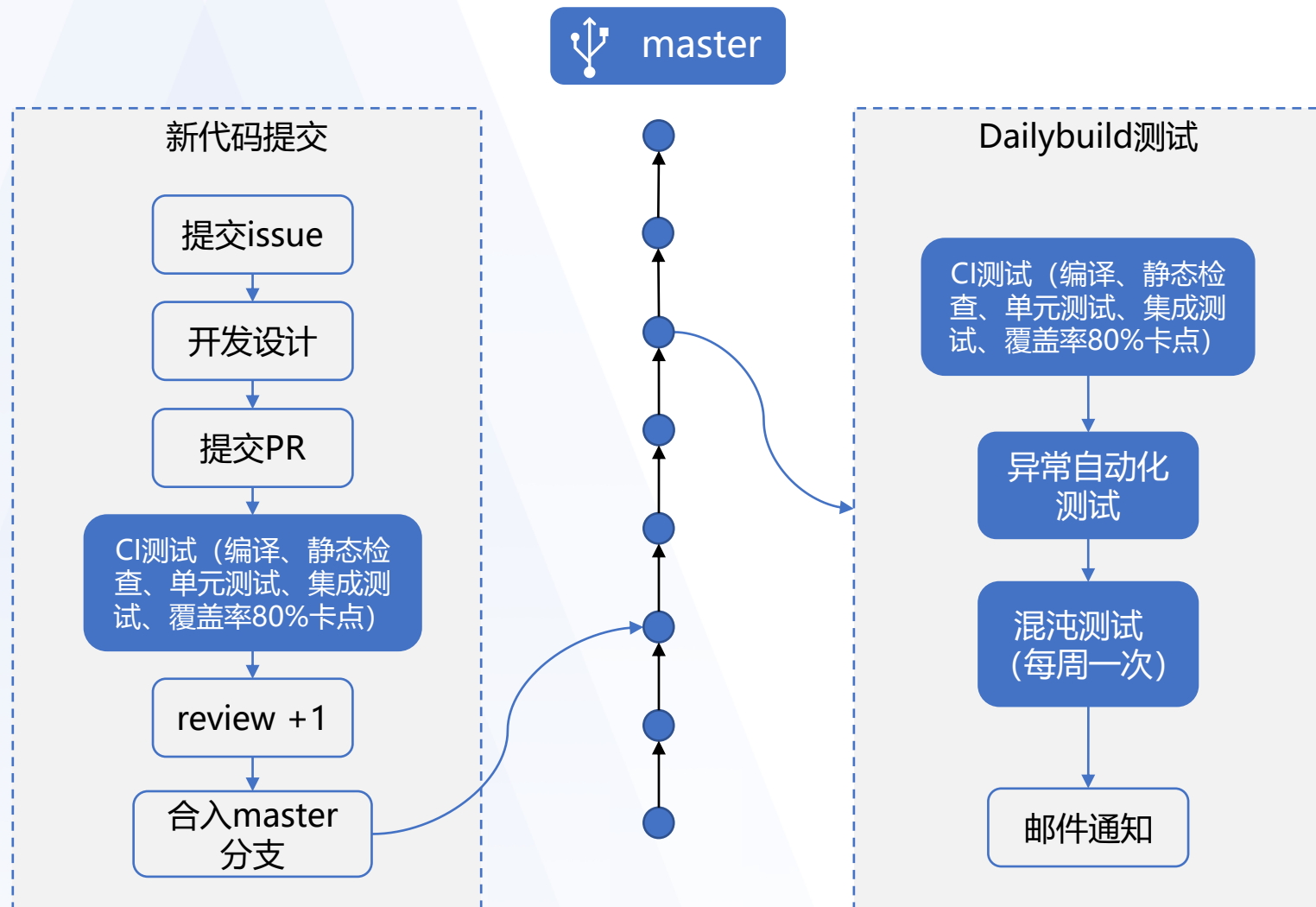
[Smart Pointers](#) [cpplint](#)

代码提交流程



Curve所有代码均在github托管。新代码需要通过CI测试和code review才能合入master分支，确保新合入代码的功能、正确性、规范性等都有基本保障；而每日运行的dailybuild测试在CI测试基础上增加了异常自动化测试和混沌测试，确保master分支代码的bug尽可能早地暴露出来。

通过这种流程，curve可以在一定程度上保证master分支的稳定性。

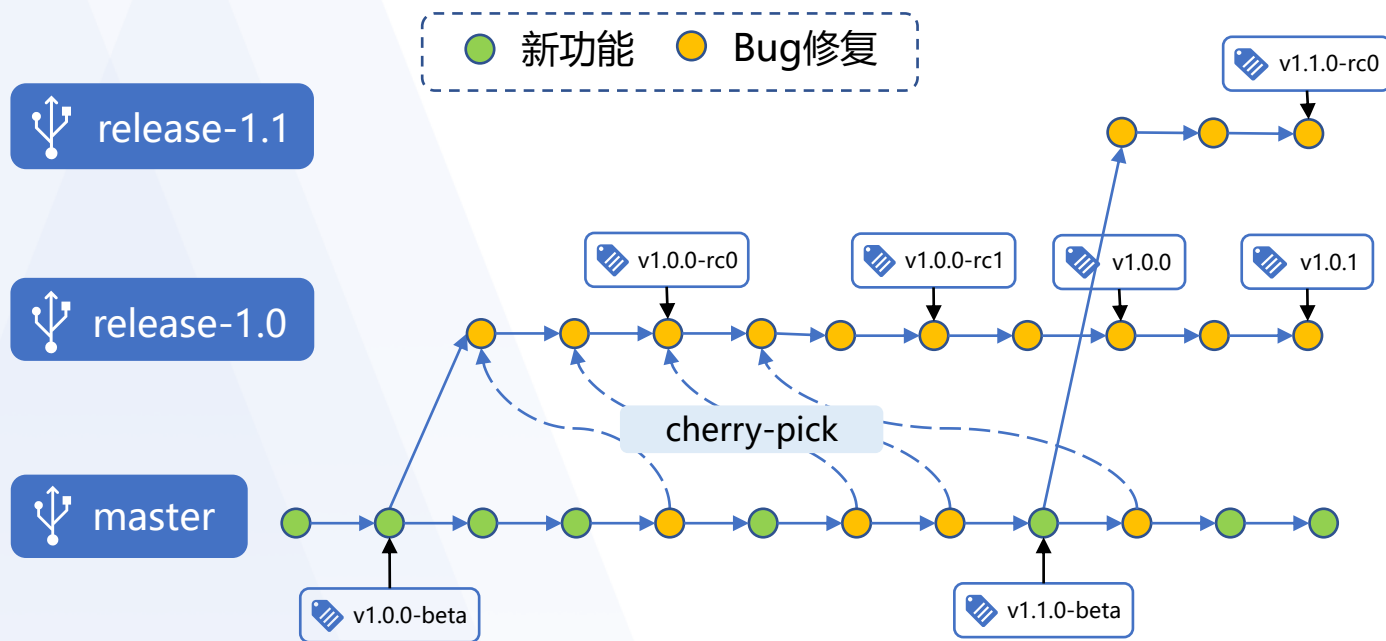


版本管理



Curve版本命名规则是**x.y.z{-后缀}**

- ❑ **x**为主版本号，每次发布大版本时递增；大版本一般半年发布一次。
- ❑ **y**为次版本号，每次发布小版本时递增；小版本一般1~2个月发布一次。
- ❑ **z**为修订号，修复一批bug后递增。
- ❑ **后缀**表示版本状态，beta表示测试版本，rc表示发布候选版本，空白表示正式版。



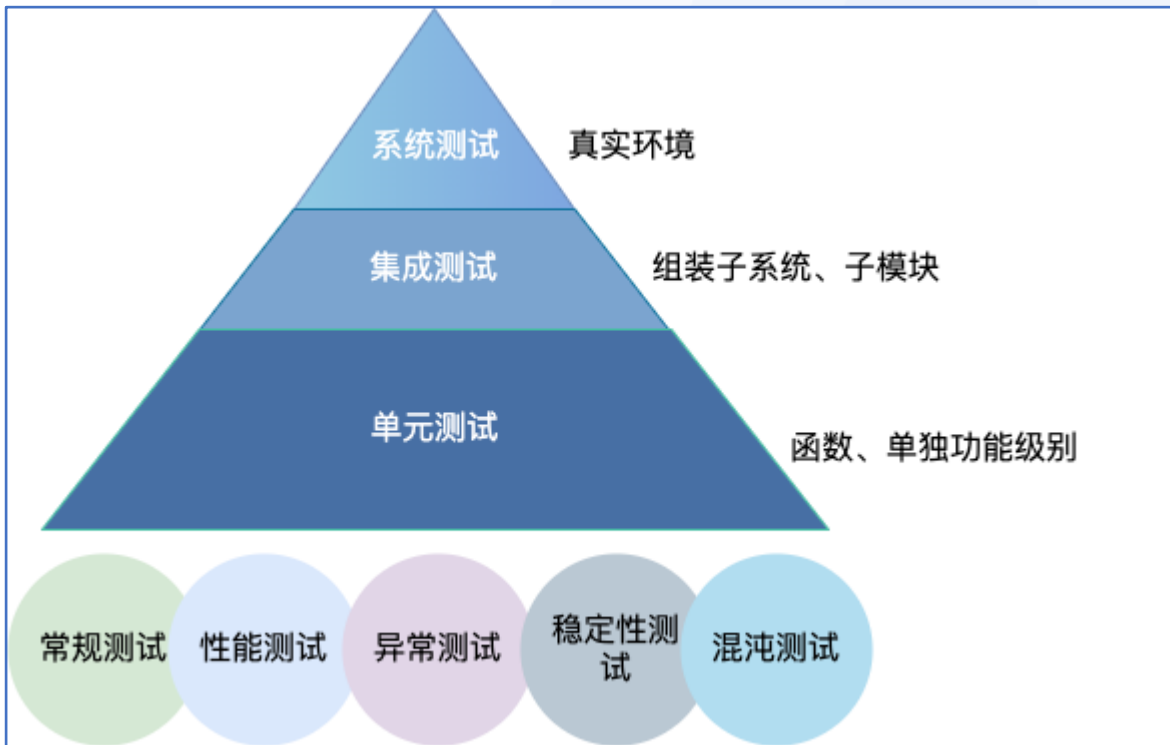
Curve所有功能开发均在master分支进行，而版本发布则在相应的release分支进行：

- ❑ 从master拉出一个新分支release-x.y，打beta版标签后，提交QA团队测试；
- ❑ beta版的bug修复代码先合入master分支，再cherry-pick到release-x.y分支；
- ❑ beta版bug修复完成后，打rc版标签（可能有多个rc版），上线到测试环境；
- ❑ 经bug修复和长时间运行测试后，若代码达到正式上线标准，则发布正式版。

测试方法论

从测试粒度看，测试可以分为单元测试、集成测试、系统测试；

从测试角度看，测试可以分为常规测试、性能测试、异常测试、稳定性测试、混沌测试，等等



- **单元测试**
1300+用例
行覆盖80%+, 分支覆盖70%+
- **集成测试**
Given When Then 设计方法
500+用例
- **异常测试**
40+自动化用例
- **混沌测试**
20轮自动化随机故障注入

单元测试



单元测试是软件开发的过程中最基本的测试，它用来对一个模块、一个函数或者一个类来进行正确性检验的测试工作。

curve通过lcov统计代码覆盖率，衡量单元测试的完备程度，如下图所示：

LCOV - code coverage report

Current view: [top level](#) - mds/heartbeat

Test: [coverage.info](#)

Date: 2020-12-01 15:05:06

	Hit	Total	Coverage
Lines:	439	465	94.4 %
Functions:	51	53	96.2 %
Branches:	169	218	77.5 %

Filename	Line Coverage ↕		Functions ↕		Branches ↕	
chunkserver_healthy_checker.cpp	<div><div></div></div>	92.4 %	73 / 79	100.0 %	9 / 9	86.1 %
chunkserver_healthy_checker.h	<div><div></div></div>	89.5 %	17 / 19	83.3 %	5 / 6	-
copyset_conf_generator.cpp	<div><div></div></div>	91.9 %	68 / 74	100.0 %	7 / 7	85.3 %
copyset_conf_generator.h	<div><div></div></div>	100.0 %	4 / 4	100.0 %	2 / 2	-

测试目的

单元测试后，有必要进行集成测试，发现并排除在模块连接中可能发生的上述问题，最终构成要求的软件子系统或系统。集成测试需要关注的主要是各模块连接起来后的问题：

- ❑ 穿越模块接口的数据是否会丢失；
- ❑ 子功能的组合是否可以达到预期的要求；
- ❑ 子模块之间是否会相互影响；
- ❑ 单个模块的误差积累是否会放大，从而达到不可接受的程度。

测试内容

❑ 功能测试

站在使用者的角度，对模块提供的功能进行完备的测试。

❑ 异常测试

制造或模拟系统异常(磁盘错误、网络错误、资源冲突等)、依赖服务异常、应用本身异常等非正常情况，测试软件的性能和稳定性是否符合预期。

❑ 规模测试

测试模块在一定规模下是否能够正常工作，是否会出现异常或者崩溃，

系统测试是对整个系统的测试，将硬件、软件、操作人员看作一个整体，检验它是否有不符合系统说明书的地方。它是一个黑盒测试，可以发现系统分析和设计中的错误。

Curve的系统测试一般是由QA来完成，包含：

- **常规测试**，主要是新增功能的手工测试；
- **性能测试**，将性能数据与基准对照，确定性能没有出现预期外的下降或提升；
- **稳定性测试**，在正常压力下运行足够长的时间；
- **异常测试**，在正常流程中注入一种软硬件异常；
- **混沌测试**，大压力多级故障（随机组合软硬件异常）。

在系统测试过程中，我们尽可能将所有用例自动化，其优点是：

- 大幅降低了测试回归成本，加快了测试进度；
- 可以对代码进行足够频繁的测试，有利于提高代码质量；
- 容易发现隐藏的问题，手工测试无法做到频繁触发
- 测试用例可以持续积累，成为代码质量的。

目前Curve的 **异常测试以及混沌测试** 均实现了自动化。

很多情况下，待测试场景会包含多个变化的参数，每个参数有若干个典型值；如果将用例覆盖所有可能的情况，总用例数将达到不可接受的程度。因此，需要通过组合测试的方法，尽量用较少的用例数量覆盖绝大多数情况：

□ 两因素组合测试

通过测试集覆盖任意两个变量的所有取值组合。理论上两因素组合测试最多可发现95%的缺陷，平均缺陷检出率也达到了86%，在用例数量和缺陷检测能力上达到了平衡。因此，一般测试用例应该保证两因素组合的100%覆盖。

□ 多因素组合测试

生成的测试集可以覆盖任意 t 个变量 ($t > 2$) 的所有取值组合。

□ 基于选择的覆盖

选择最常用的参数值作为基础组合，在此基础上每次改变一个参数，生成新用例。

我们可以依据Given-When-Then模式来编写具体的测试用例：

Given —— 测试上下文

When —— 执行一系列操作

Then —— 得到一系列可观察的后果，即需要检测的断言。

Given	When	Then
文件存在	CreateFile, type = INODE_PAGEFILE, fileLength < kMiniFileLength	返回kParaError, 不会创建文件
	CreateFile, type = INODE_PAGEFILE fileLength = kMiniFileLength	返回kFileExists 不会创建文件
	CreateFile, type = INODE_PAGEFILE fileLength > kMiniFileLength	返回kFileExists 不会创建文件
	CreateFile, type != INODE_PAGEFILE	返回kFileExists 不会创建文件

异常自动化测试实践



Curve使用Robotframework框架进行异常自动化测试，
相关代码见[curve/robot at opencurve/curve \(github.com\)](https://github.com/opencurve/curve/blob/master/robotframework)

Robotframework

- ❑ 支持python关键字，灵活定义测试
- ❑ 完善的测试报告
- ❑ 完美兼容Jenkins ci
- ❑ 丰富的第三方库 (ssh, paramiko, request等)

```
inject two chunkserver down/up
[Tags]    P1    base    first release    failover
${num}    evaluate    int(2)
${host}    test kill chunkserver num    ${num}
check vm iops
check data consistency
check copies consistency
sleep 5
test start chunkserver num    ${num}    ${host}
check vm iops
check data consistency
check copies consistency
[Teardown]    start host cs process    ${host}
```


用例设计原则

- ❑ 无需绑定特定环境，“随意拉起”
- ❑ 配置化（测试环境、测试负载定义）
- ❑ 控制用例时间（考虑一些折中方案）
- ❑ Case独立性
- ❑ Case通用性（兼顾curve、ceph等）
- ❑ Tag规范(优先级、版本、运行时间)
- ❑ 最大化覆盖率（打乱操作顺序、随机sleep)
- ❑ 精确性（checkpoint)
- ❑ 稳定性（避免环境因素、其他模块干扰)

CI测试与异常测试报表



Curve通过jenkins软件实现代码的持续集成（CI），下面分别是CI测试和异常测试的报表。

**构建 #535 (2020-12-1 19:28:43)**

启动构建

PR #167: fix copysset codec bug



No changes.





GitHub pull request #167 of commit 5ec8e77f682b45b1a79c4af09266617b1b875137, has merge conflicts.



Revision: 5ec8e77f682b45b1a79c4af09266617b1b875137

- detached

S	R	Job	Build #	Duration	Console
		opencurve-ci			
		curve_cpplint_job	build #444	(1 min 5 sec)	
		curve_untest_job	build #448	(21 min)	
		curve_cppcheck_job	build #239	(28 sec)	
		curve_robot_job	build #223	(7 min 2 sec)	

SUITE

Curve Failover Robot

Full Name:

Curve Failover Robot

Source:

/var/lib/jenkins/workspace/curve/curve_multijob/robot/curve_failover_robot.txt

Start / End / Elapsed:

20200719 06:37:25.229 / 20200719 09:38:15.505 / 03:00:50.276

Status:

41 critical test, 41 passed, 0 failed
41 test total, 41 passed, 0 failed

SETUP

init failover cluster

TEARDOWN

clean failover env

TEST

inject one chunkserver down/up

TEST

inject two chunkserver down/up

TEST

inject host all chunkserver down/up

TEST

inject restart one chunkserver

TEST

inject restart two chunkserver

TEST

inject kill diff host chunkserver

TEST

inject reboot nebd

TEST

inject stop and start vm

TEST

inject reboot vm

TEST

inject hang vm

TEST

inject kill one mds

TEST

inject kill two mds

TEST

inject round restart mds



目录

01

背景

为用户服务——质量、监控和运维

02

Curve质量控制

质量管理体系 | 测试方法论

03

Curve监控体系

总体架构 | 指标生成 | 后端采集 | 可视化展示

04

Curve运维体系

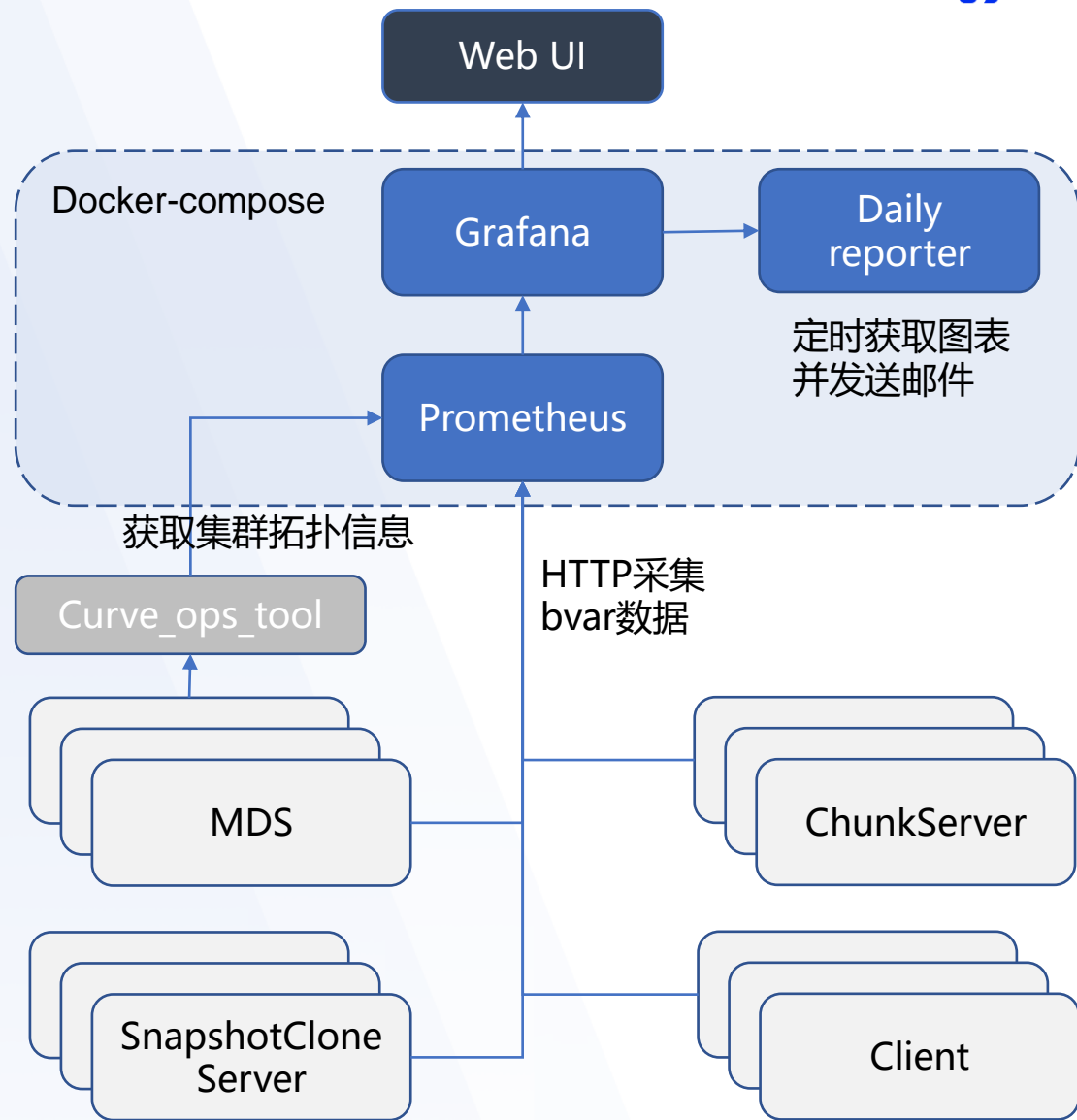
Curve运维特性 | 运维工具

Curve监控架构



Curve利用brpc内置的bvar组件生成监控指标，并使用部署在docker的三个组件进行监控指标的处理与展示：

- **Prometheus**——面向云原生应用程序的开源的监控&报警工具，curve利用它进行监控指标的采集与存储。
- **Daily reporter**——python脚本，定时从Grafana获取指定集群的图表，生成集群监控日报，并通过邮件发送。
- **Grafana**——开源的度量分析和可视化工具，curve利用它进行数据可视化展示。



- ❑ thread local存储，减少了cache bouncing，性能开销极小；
- ❑ 支持在 brpc server 服务的端口上以web portal的方式导出和查询监控指标：

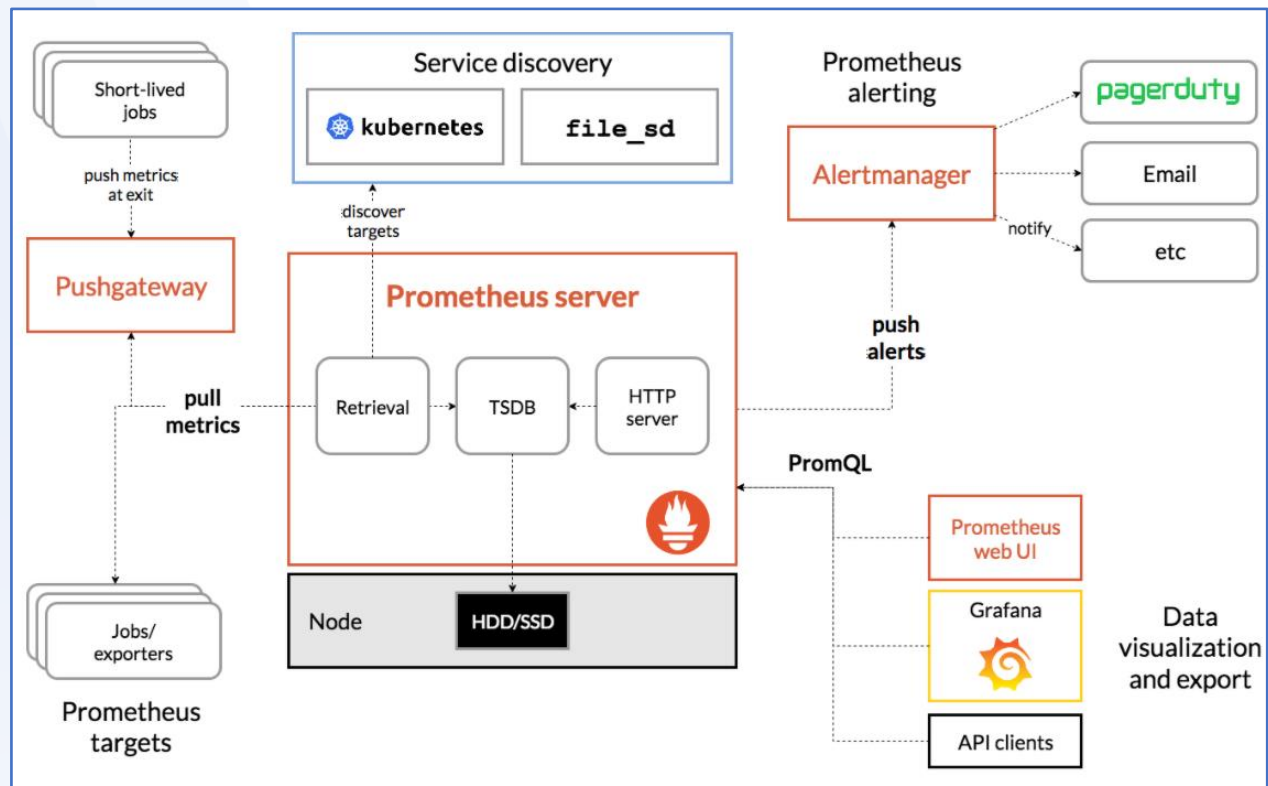
常用的bvar数据类型:

- 21/33

监控指标的采集与存储

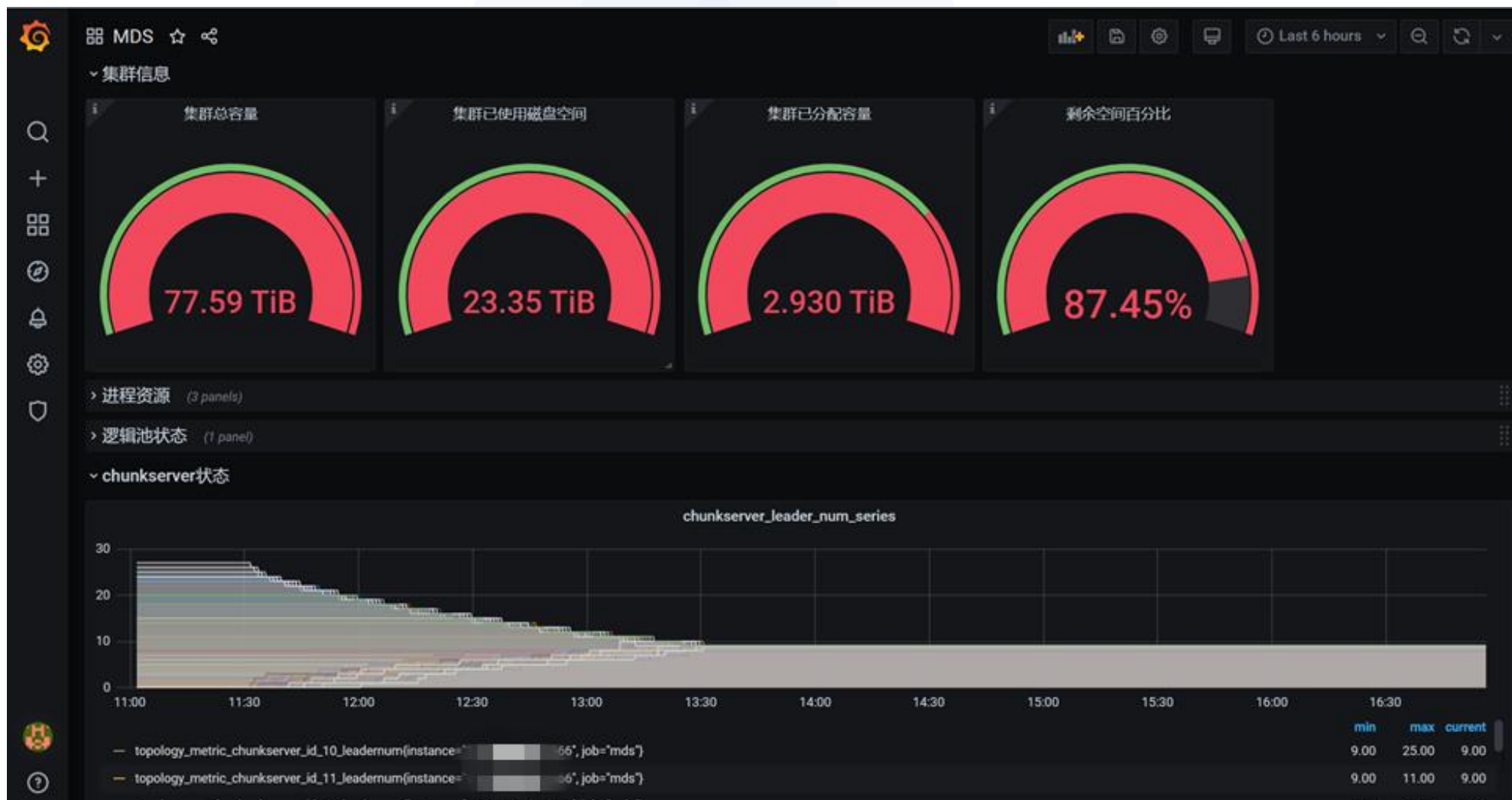
Curve使用开源的监控工具Prometheus采集监控指标，大致流程为：

1. 部署监控时，Curve根据集群信息生成配置文件，指定了Prometheus的监控目标（包括Client、MDS、ChunkServer、Etcd、物理节点等）。
2. Prometheus依据上述配置文件，发现相应服务。
3. Prometheus server以pull的方式，定期从Curve集群中MDS、ChunkServer、Client等组件的brpc Server拉取相应的监控指标，并存储在本地。

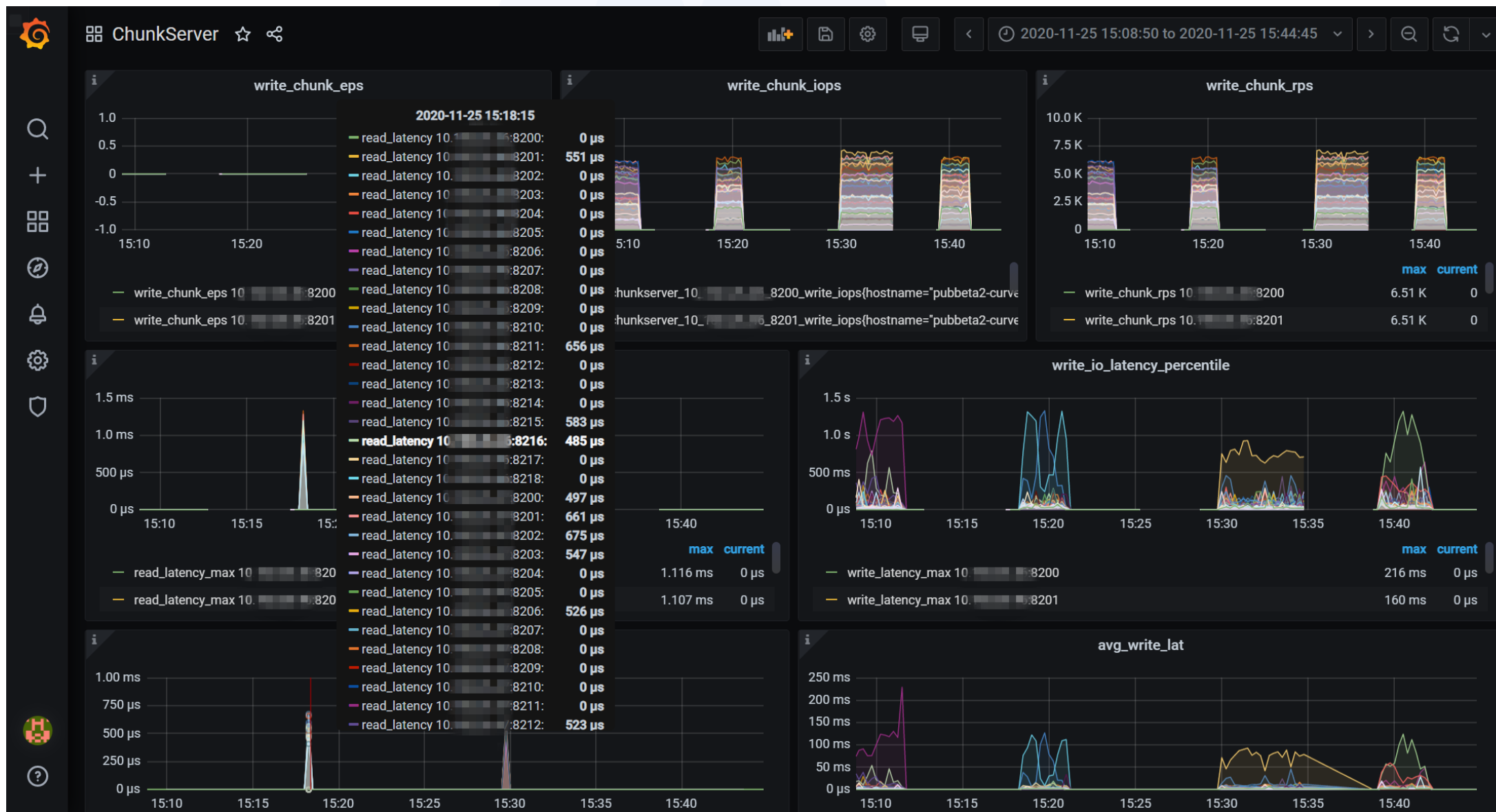


监控指标的可视化

Curve通过Grafana实现监控指标的可视化，其展示效果如下：



丰富的metric



每日报表



Curve每天通过daily reporter从Grafana获取图表，生成每日报表，并定时发送邮件。
用户无需登陆监控平台即可轻松掌握Curve每日运行状态。





目录

01

背景

为用户服务——质量、监控和运维

02

Curve质量控制

质量管理体系 | 测试方法论

03

Curve监控体系

总体架构 | 指标生成 | 后端采集 | 可视化展示

04

Curve运维体系

Curve运维特性 | 运维工具

易部署

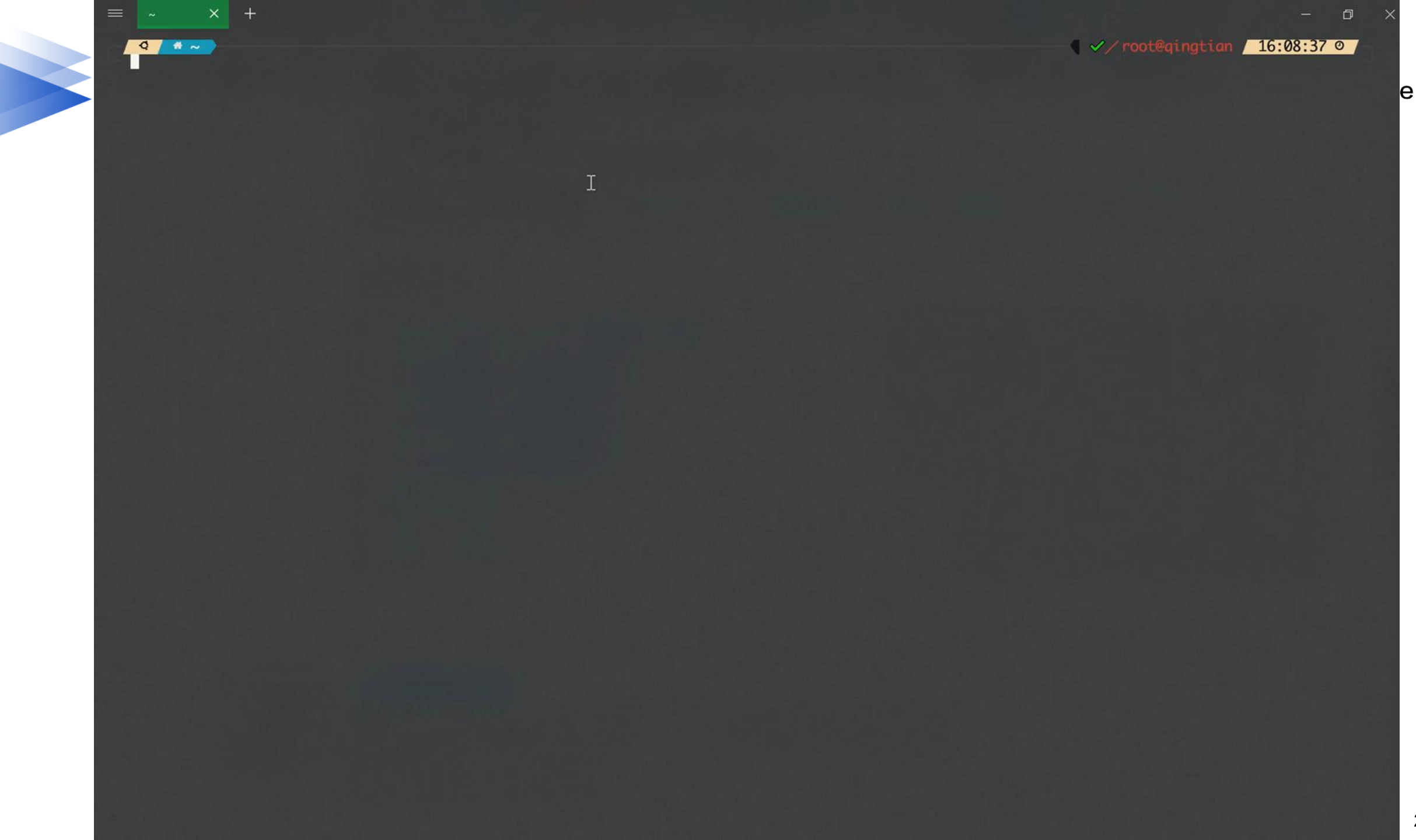
- 批量配置、批量部署
- 操作简单

易升级

- 客户端采用CS架构，升级只需重启服务，秒级影响
- MDS与ChunkServer支持滚动升级

自治

- 自动均衡
- 自动故障恢复



□ Client易升级

为避免Curve client升级影响QEMU，Curve Client采用了Client-Server架构，以支持热升级。

升级Curve Client只需重启NEBD Server，业务IO中断时间一般在5秒之内（右图为1.0版本实测结果）。

□ MDS易升级

自动化滚动升级——先升备再升主，确保升级过程中只发生一次主备切换。

□ ChunkServer易升级

自动化滚动升级——升级一个zone的所有ChunkServer后，等待集群恢复健康后，自动升级下一个zone的ChunkServer；以避免升级时一个copyset中多个ChunkServer离线，导致业务IO挂起。

重启NEBD Server



自动均衡



Curve可以在copyset层面自动均衡集群负载，
无需人工干预：

□ Copyset均衡

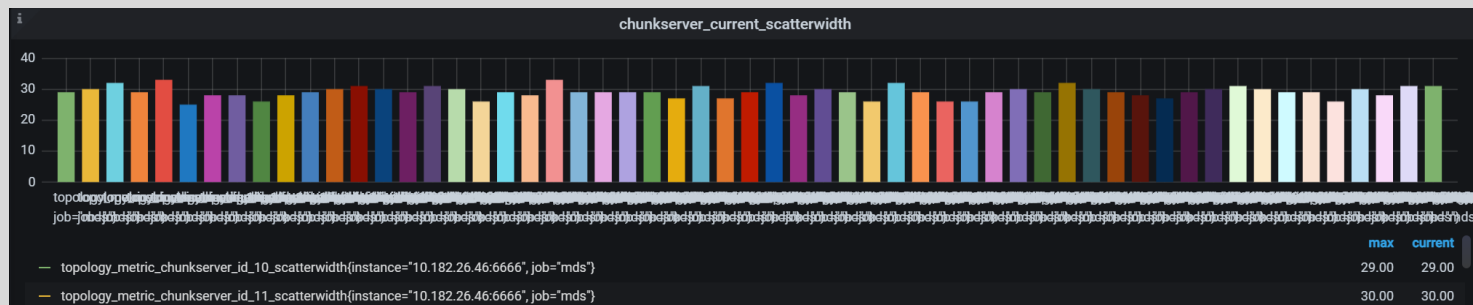
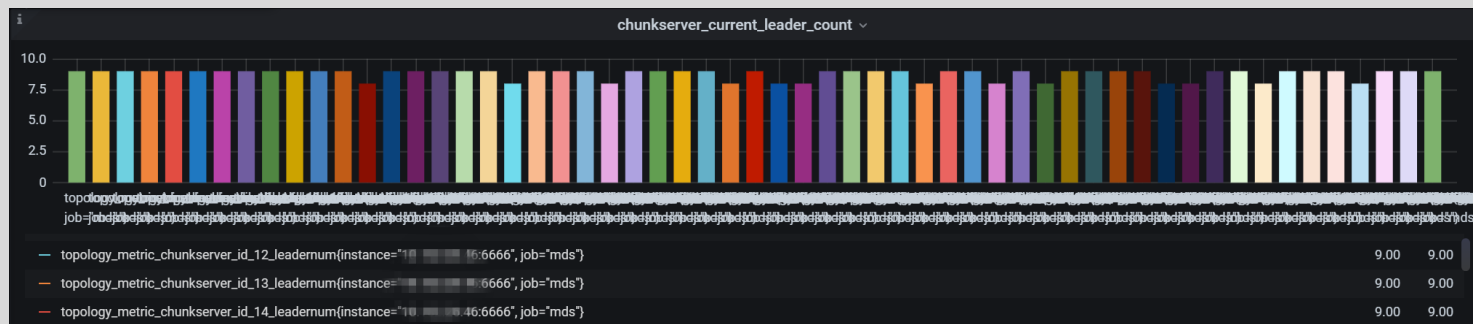
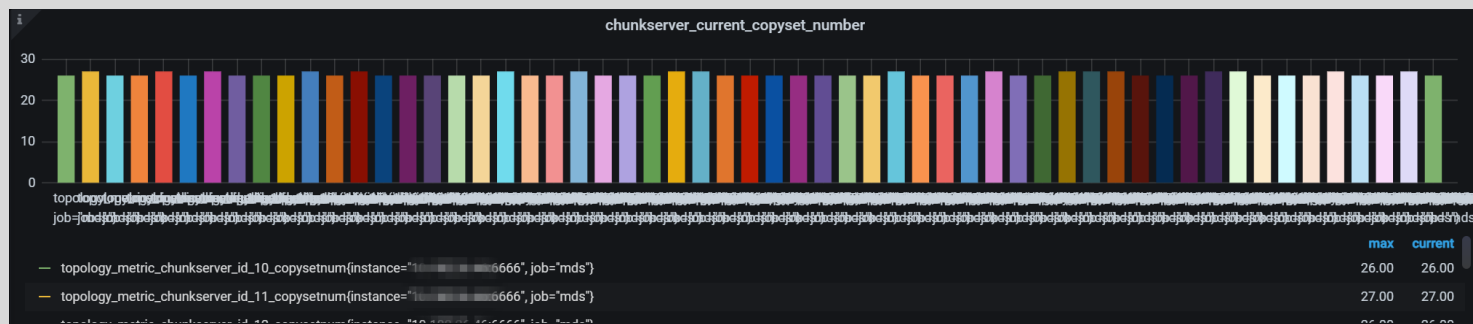
各ChunkServer上copyset数量均衡

□ Leader均衡

各ChunkServer上copyset leader数量均衡

□ Scatter-width（打散度）均衡

各ChunkServer上全部copyset，其副本分布的
ChunkServer总数量均衡。



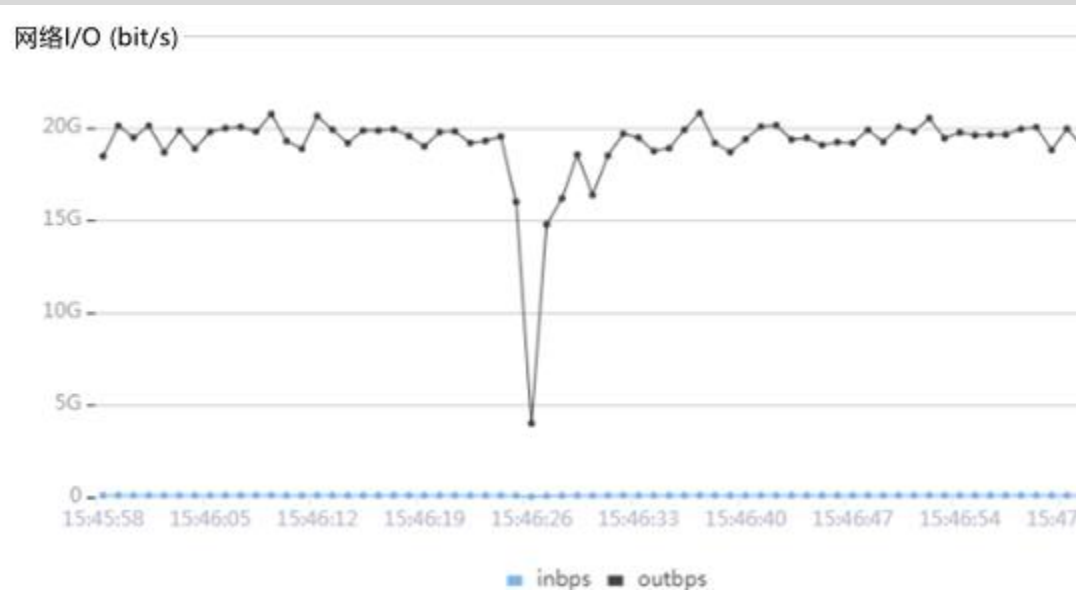
自动故障恢复



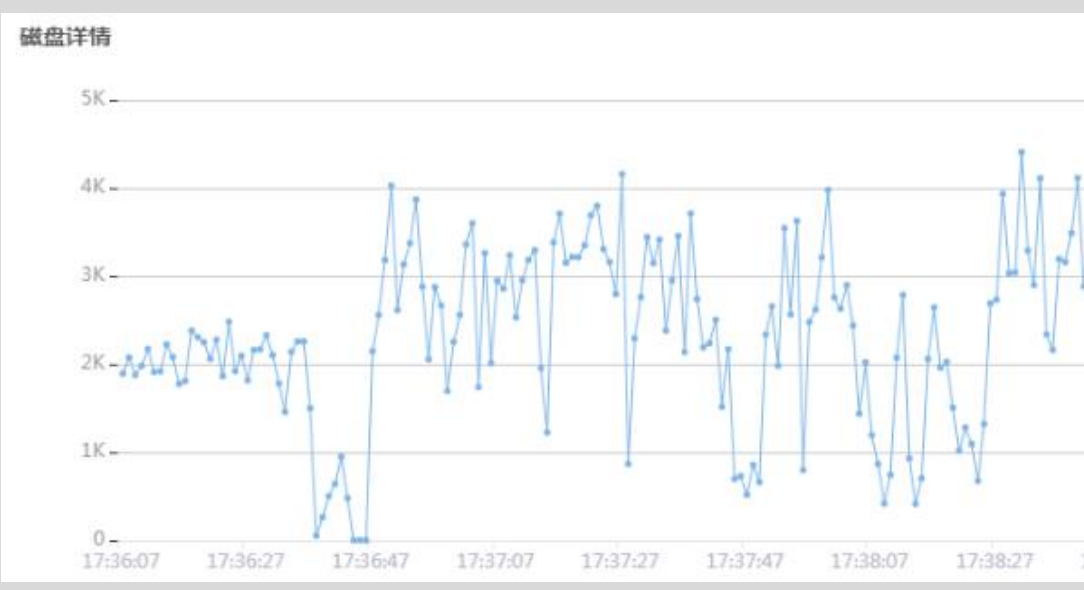
Curve可在多种软硬件故障场景（如单mds故障、单ChunkServer故障、硬盘故障、网络丢包等）实现自动恢复，保障存储服务高可用性。

- 多对多，恢复时间短
- 精确的流量控制，对io影响很小

Kill一个节点所有ChunkServer进程



网络丢包10%



Ansible

- ❑ 一键部署: `ansible-playbook -i server.ini deploy_curve.yml`
- ❑ 一键升级: `ansible-playbook -i server.ini rolling_update_curve.yml`

Curve_ops_tool

- ❑ 查询Curve状态
- ❑ 管理Curve文件
- ❑ 管理copyset

```
Usage: curve_ops_tool [Command] [OPTIONS...]
COMMANDS:
space : show curve all disk type space, inclu
status : show the total status of the cluster
chunkserver-status : show the chunkserver onl
mds-status : show the mds status
client-status : show the client status
etcd-status : show the etcd status
snapshot-clone-status : show the snapshot clo
copysets-status : check the health state of a
chunkserver-list : show curve chunkserver-lis
get : show the file info and the actual space
list : list the file info of files in the dir
seginfo : list the segments info of the file
delete : delete the file, to force delete, sh
clean-recycle : clean the RecycleBin
create : create file, file length unit is GB
chunk-location : query the location of the ch
check-consistency : check the consistency of
remove-peer : remove the peer from the copyse
transfer-leader : transfer the leader of the
reset-peer : reset the configuration of copyse
check-chunkserver : check the health state of
check-copyset : check the health state of one
check-server : check the health state of the
check-operator : check the operators
rapid-leader-schedule: rapid leader schedule
```

快照克隆工具snaptool

```
snapshot tool

positional arguments:
  {query-snapshot,query-clone-recover,sn
snapshot,clean-recover,clean-clone,creat
  query-snapshot      query snapshot b
  query-clone-recover

                        query clone and
snapshot-status      show the snapsho
clone-recover-status

                        query clone and
delete-snapshot      delete snapshot
cancel-snapshot      cancel snapshot
clean-recover        clean recover by
clean-clone          delete clone by
create-snapshot      create a snapsho
clone                do clone
recover              do recover
flatten              do flatten lazy
```


THANK YOU

D I G I T A L S A I L



扫码即可关注