Contents

0.1 TiDB 简介

TiDB 是 PingCAP 公司基于 Google Spanner / F1 论文实现的开源分布式 NewSQL 数据库。

TiDB 具备如下 NewSQL 核心特性 * SQL 支持（TiDB 是 MySQL 兼容的）* 水平线性弹性扩展 * 分布式事务 * 跨数据中心数据强一致性保证 * 故障自恢复的高可用

TiDB 的设计目标是 100% 的 OLTP 场景和 80% 的 OLAP 场景。

TiDB 对业务没有任何侵入性，能优雅的替换传统的数据库中间件、数据库分库分表等 Sharding 方案。同时它也让开发运维人员不用关注数据库 Scale 的细节问题，专注于业务开发，极大的提升研发的生产力。

0.2 TiDB 整体架构

要深入了解 TiDB 的水平扩展和高可用特点，首先需要了解 TiDB 的整体架构。

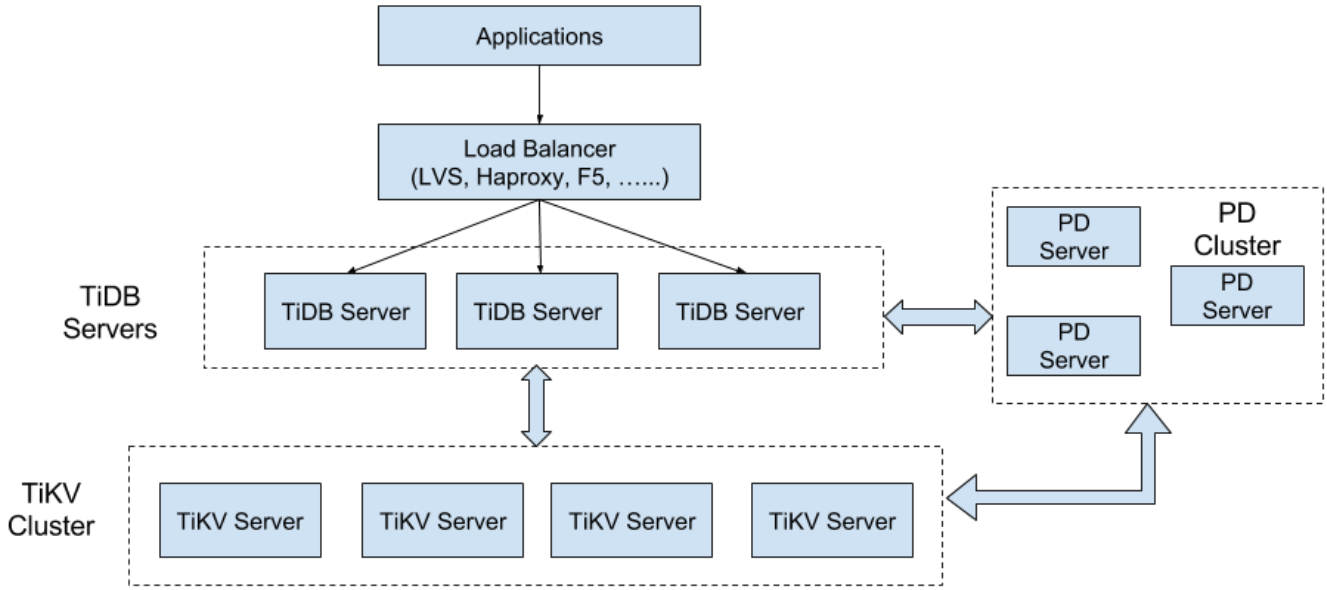

Figure 1: TiDB architecture

TiDB 集群主要分为三个组件：#### TiDB Server TiDB Server 负责接收 SQL 请求，处理 SQL 相关的逻辑，并通过 PD 找到存储计算所需数据的 TiKV 地址，与 TiKV 交互获取数据，最终返回结果。TiDB Server 是无状态的，其本身并不存储数据，只负责计算，可以无限水平扩展，可以通过负载均衡组件（如 LVS、HAProxy 或 F5）对外提供统一的接入地址。

0.2.0.1 PD Server

Placement Driver (简称 PD) 是整个集群的管理模块，其主要工作有三个：一是存储集群的原信息（某个 Key 存储在哪个 TiKV 节点）；二是对 TiKV 集群进行调度和负载均衡（如数据的迁移、Raft group leader 的迁移等）；三是分配全局唯一且递增的事务 ID

PD 是一个集群，需要部署奇数个节点，一般线上推荐至少部署 3 个节点。

0.2.0.2 TiKV Server

TiKV Server 负责存储数据，从外部看 TiKV 是一个分布式的提供事务的 Key-Value 存储引擎。存储数据的基本单位是 Region，每个 Region 负责存储一个 Key Range（从 StartKey 到 EndKey 的左闭右开区间）的数据，每个 TiKV 节点会负责多个 Region 。TiKV 使用 Raft 协议做复制，保持数据的一致性和容灾。副本以 Region 为单位进行管理，不同节点上的多个 Region 构成一个 Raft Group，互为副本。数据在多个 TiKV 之间的负载均衡由 PD 调度，这里也是以 Region 为单位进行调度。

0.2.1 水平扩展

无限水平扩展是 TiDB 的一大特点，这里说的水平扩展包括两方面：计算能力和存储能力。TiDB Server 负责处理 SQL 请求，随着业务的增长，可以简单的添加 TiDB Server 节点，提高整体的处理能力，提供更高的吞吐。TiKV 负责存储数据，随着数据量的增长，可以部署更多的 TiKV Server 节点解决数据 Scale 的问题。PD 会在 TiKV 节点之间以 Region 为单位做调度，将部分数据迁移到新加的节点上。所以在业务的早期，可以只部署少量的服务实例（推荐至少部署 3 个 TiKV，3 个 PD，2 个 TiDB），随着业务量的增长，按照需求添加 TiKV 或者 TiDB 实例。

0.2.2 高可用

高可用是 TiDB 的另一大特点，TiDB/TiKV/PD 这三个组件都能容忍部分实例失效，不影响整个集群的可用性。下面分别说明这三个组件的可用性、单个实例失效后的后果以及如何恢复。#### TiDB TiDB 是无状态的，推荐至少部署两个实例，前端通过负载均衡组件对外提供服务。当单个实例失效时，会影响正在这个实例上进行的 Session，从应用的角度看，会出现单次请求失败的情况，重新连接后即可继续获得服务。单个实例失效后，可以重启这个实例或者部署一个新的实例。

0.2.2.1 PD

PD 是一个集群，通过 Raft 协议保持数据的一致性，单个实例失效时，如果这个实例不是 Raft 的 leader，那么服务完全不受影响；如果这个实例是 Raft 的 leader，会重新选出新的 Raft leader，自动恢复服务。PD 在选举的过程中无法对外提供服务，这个时间大约是 3 秒钟。推荐至少部署三个 PD 实例，单个实例失效后，重启这个实例或者添加新的实例。

0.2.2.2 TiKV

TiKV 是一个集群，通过 Raft 协议保持数据的一致性（副本数量可配置，默认保存三副本），并通过 PD 做负载均衡调度。单个节点失效时，会影响这个节点上存储的所有 Region。对于 Region 中的 Leader 结点，会中断服务，等待重新选举；对于 Region 中的 Follower 节点，不会影响服务。当某个 TiKV 节点失效，并且在一段时间内（默认 10 分钟）无法恢复，PD 会将其上的数据迁移到其他的 TiKV 节点上。

0.2.2.3 更多资源

- PingCAP 团队技术博客
- 常用工具

0.3 部署建议

阅读本章前，请先阅读 TiDB 整体架构。

0.3.1 TiDB 集群各个组件的硬件消耗情况及推荐配置

组件	消耗硬件资源	推荐硬件配置
TiDB	CPU、内存	8+ 核/16G+ 内存
TiKV	CPU、内存、磁盘 IO	8+ 核/16G+ 内存/200G+ 硬盘（建议 SSD）
PD	CPU、内存、磁盘 IO	8+ 核/16G+ 内存/200G+ 硬盘（建议 SSD）

备注：* TiKV 硬盘大小建议不要超过 500G（防止硬盘损坏时，数据恢复耗时过长）

0.3.2 整体部署方案

组件	生产环境所需机器情况	测试环境所需机器情况
TiDB	至少 2 台，保证高可用，可按所需并发和吞吐，动态增加机器	可以 1 台
PD	必须 3 台，保证高可用	可以 1 台
TiKV	至少 3 台，保证高可用，可按所需计算资源和存储容量，动态增加机器	至少 3 台

备注：* TiDB 实例可以和任意一台 PD 部署在同一台机器，也可以单独部署。* PD 和 TiKV 实例，建议每个实例单独部署一个硬盘，避免 IO 冲突，影响性能。* TiDB 和 TiKV 实例，建议分开部署，以免竞争 CPU 资源，影响性能。

0.3.2.1 举例：生产环境部署（推荐至少 6 台机器）

机器 1	机器 2	机器 3	机器 4	机器 5	机器 6
TiKV1	TiKV2	TiKV3	PD1	PD2	PD3
-	-	-	TiDB1	TiDB2	-

其中 TiDB1 和 TiDB2 通过负载均衡组件对外统一提供 SQL 接口

0.3.2.2 举例：测试验证环境部署（推荐至少 4 台机器）

机器 1	机器 2	机器 3	机器 4
TiKV1	TiKV2	TiKV3	PD1
-	-	-	TiDB1

1 TiDB Binary 部署方案

1.1 概述

一个完整的 TiDB 集群包括 PD，TiKV 以及 TiDB。启动顺序依次是 PD，TiKV 以及 TiDB。

阅读本章前，请先确保阅读 TiDB 整体架构 及 部署建议

快速了解和试用 TiDB，推荐使用单节点方式快速部署。

功能性测试 TiDB，推荐使用功能性测试部署。

生产环境使用 TiDB，推荐使用多节点集群模式部署。

1.2 下载官方 Binary

1.2.1 Linux (CentOS 7+, Ubuntu 14.04+)

```
# 下载压缩包
wget http://download.pingcap.org/tidb-latest-linux-amd64.tar.gz
wget http://download.pingcap.org/tidb-latest-linux-amd64.sha256

# 检查文件完整性，返回 ok 则正确
sha256sum -c tidb-latest-linux-amd64.sha256

# 解开压缩包
tar -xzf tidb-latest-linux-amd64.tar.gz
cd tidb-latest-linux-amd64
```

1.2.1.1 CentOS 6（不推荐）

```
# 下载 CentOS6 压缩包
wget http://download.pingcap.org/tidb-latest-linux-amd64-centos6.tar.gz
wget http://download.pingcap.org/tidb-latest-linux-amd64-centos6.sha256

# 检查文件完整性，返回 ok 则正确
sha256sum -c tidb-latest-linux-amd64-centos6.sha256

# 解开压缩包
tar -xzf tidb-latest-linux-amd64-centos6.tar.gz
cd tidb-latest-linux-amd64-centos6
```

1.3 单节点方式快速部署

我们可以在单机上面，运行和测试 TiDB 集群，请按如下步骤**依次启动** PD，TiKV，TiDB：

1. 启动 PD.

```
./bin/pd-server --data-dir=pd
```

2. 启动 TiKV.

```
./bin/tikv-server --pd="127.0.0.1:2379" \
                  --store=tikv
```

3. 启动 TiDB.

```
./bin/tidb-server --store=tikv \
                  --path="127.0.0.1:2379"
```

4. 使用官方的 `mysql` 客户端连接 TiDB.

```
mysql -h 127.0.0.1 -P 4000 -u root -D test
```

1.4 多节点集群模式部署

在生产环境中，我们推荐多节点部署 TiDB 集群，首先请参考部署建议。

这里我们使用六个节点，部署三个 PD，三个 TiKV，以及一个 TiDB，各个节点以及所运行服务信息如下：

Name	Host IP	Services
node1	192.168.199.113	PD1, TiDB
node2	192.168.199.114	PD2
node3	192.168.199.115	PD3
node4	192.168.199.116	TiKV1
node5	192.168.199.117	TiKV2
node6	192.168.199.118	TiKV3

请按如下步骤**依次启动** PD 集群，TiKV 集群以及 TiDB：

1. 在 node1，node2，node3 启动 PD.

```
./bin/pd-server --name=pd1 \
                --data-dir=pd1 \
                --client-urls="http://192.168.199.113:2379" \
                --peer-urls="http://192.168.199.113:2380" \
                --initial-cluster="pd1=http://192.168.199.113:2380, \
                                   pd2=http://192.168.199.114:2380, \
                                   pd3=http://192.168.199.115:2380"

./bin/pd-server --name=pd2 \
                --data-dir=pd2 \
                --client-urls="http://192.168.199.114:2379" \
                --peer-urls="http://192.168.199.114:2380" \
                --initial-cluster="pd1=http://192.168.199.113:2380,\
                                   pd2=http://192.168.199.114:2380,\
                                   pd3=http://192.168.199.115:2380"

./bin/pd-server --name=pd3 \
                --data-dir=pd3 \
                --client-urls="http://192.168.199.115:2379" \
                --peer-urls="http://192.168.199.115:2380" \
                --initial-cluster="pd1=http://192.168.199.113:2380, \
```

```
                                    pd2=http://192.168.199.114:2380, \
                                    pd3=http://192.168.199.115:2380"
```

2. 在 node4，node5，node6 启动 TiKV.

```
./bin/tikv-server --pd="192.168.199.113:2379,192.168.199.114:2379,192.168.199.115:2379" \
                  --addr="192.168.199.116:20160" \
                  --store=tikv1

./bin/tikv-server --pd="192.168.199.113:2379,192.168.199.114:2379,192.168.199.115:2379" \
                  --addr="192.168.199.117:20160" \
                  --store=tikv2

./bin/tikv-server --pd="192.168.199.113:2379,192.168.199.114:2379,192.168.199.115:2379" \
                  --addr="192.168.199.118:20160" \
                  --store=tikv3
```

3. 在 node1 启动 TiDB.

```
./bin/tidb-server --store=tikv \
                  --path="192.168.199.113:2379,192.168.199.114:2379,192.168.199.115:2379"
```

4. 使用官方 `mysql` 客户端连接 TiDB.

```
mysql -h 192.168.199.113 -P 4000 -u root -D test
```

注意：在生产环境中启动 TiKV 时，建议使用 –config 参数指定配置文件路径，如果不设置这个参数，TiKV 不会读取配置文件。同样，在生产环境中部署 PD 时，也建议使用 –config 参数指定配置文件路径。

1.5 功能性测试部署

如果只是对 TiDB 进行测试，并且机器数量有限，我们可以只启动一台 PD 测试整个集群。

这里我们使用四个节点，部署一个 PD，三个 TiKV，以及一个 TiDB，各个节点以及所运行服务信息如下：

Name	Host IP	Services
node1	192.168.199.113	PD1, TiDB
node2	192.168.199.114	TiKV1
node3	192.168.199.115	TiKV2
node4	192.168.199.116	TiKV3

请按如下步骤**依次启动** PD 集群，TiKV 集群以及 TiDB：

1. 在 node1 启动 PD.

```
./bin/pd-server --name=pd1 \
                --data-dir=pd1 \
                --client-urls="http://192.168.199.113:2379" \
                --peer-urls="http://192.168.199.113:2380" \
                --initial-cluster="pd1=http://192.168.199.113:2380"
```

2. 在 node2，node3，node4 启动 TiKV.

```
./bin/tikv-server --pd="192.168.199.113:2379" \
                  --addr="192.168.199.114:20160" \
                  --store=tikv1

./bin/tikv-server --pd="192.168.199.113:2379" \
                  --addr="192.168.199.115:20160" \
                  --store=tikv2
```

```
./bin/tikv-server --pd="192.168.199.113:2379" \
                  --addr="192.168.199.116:20160" \
                  --store=tikv3
```

3. 在 node1 启动 TiDB.

```
./bin/tidb-server --store=tikv \
                  --path="192.168.199.113:2379"
```

4. 使用官方 mysql 客户端连接 TiDB.

```
mysql -h 192.168.199.113 -P 4000 -u root -D test
```

1.6 动态添加节点

1.6.1 PD

我们可以使用 join 参数，方便的将一个 PD 服务加入到现有的 PD 集群里面。假设现在我们有三个 PD 服务，详细信息如下：

Name	ClientUrls	PeerUrls
pd1	http://host1:2379	http://host1:2380
pd2	http://host2:2379	http://host2:2380
pd3	http://host3:2379	http://host3:2380

如果我们需要添加 pd4，只需要在 join 参数里面填入当前 PD 集群某一个 PD 服务的 ClientUrls 就可以了，如下：

```
./bin/pd-server --name=pd4 \
                --client-urls="http://host4:2379"
                --peer-urls="http://host4:2380"
                --join="http://host1:2379"
```

如果我们需要删除 pd4，可以通过 PD 的 HTTP API 来完成：

```
curl -X DELETE http://host1:2379/pd/api/v1/members/pd4
```

最后我们可以查看当前 PD 的所有节点来确定是否添加或者删除成功：

```
curl http://host1:2379/pd/api/v1/members
```

1.6.2 TiKV

动态新加入一个新的 TiKV 服务是非常容易的，我们可以直接启动一个 TiKV 服务，PD 会自动检测到，并开始做整个集群的 balance，将其他 TiKV 的数据移动到新加入的 TiKV 里面。

我们也能够显式的告诉 PD 去删除某个 TiKV。PD 会先把这个 TiKV 标记为正在下线的状态，然后把这个 TiKV 上的数据均匀地迁移到其他 TiKV 上面。当这个 TiKV 上的数据已经迁移完了，PD 会把这个 TiKV 标记为完成下线的状态，这时候就可以安全地把这个 TiKV 从集群中去掉。

假设我们要删除一个 store id 为 1 的 TiKV，可以调用 PD 的 HTTP API 来操作：

```
curl -X DELETE http://host:port/pd/api/v1/store/1
```

然后可以查看这个 TiKV 的当前状态：

```
curl http://host:port/pd/api/v1/store/1
```

如果这个 TiKV 正在下线，对应的 state=1，如果这个 TiKV 完成下线，对应的 state=2，否则 state=0。

更详细的 API 文档可以参考 PD APIv1。

1.6.3 TiDB

TiDB 是一个无状态的服务，这也就意味着我们能直接添加和删除 TiDB。需要注意的是如果我们在 TiDB 的服务的前面搭建了一个 proxy (譬如 HAProxy)，我们需要更新 proxy 的配置并重新载入。

2 TiDB Docker 部署方案

本篇将展示如何在多台主机上使用 Docker 部署一个 TiDB 集群。

阅读本章前，请先确保阅读 TiDB 整体架构 及 部署建议

2.1 环境准备

2.1.1 安装 Docker

Docker 可以方便地在 Linux / Mac OS / Windows 平台安装，安装方法请参考 Docker 官方文档

2.1.2 拉取 TiDB 的 Docker 镜像

部署 TiDB 集群主要包括 3 个服务组件:

- TiDB
- TiKV
- PD

对应的最新 Docker 镜像可以通过 Docker 官方镜像仓库 获取

```
docker pull pingcap/tidb:latest
docker pull pingcap/tikv:latest
docker pull pingcap/pd:latest
```

2.2 部署一个多节点集群

假设我们打算在 6 台主机上部署一个 TiDB 集群:

主机名	IP	部署服务	数据盘挂载
host1	192.168.1.101	PD1 & TiDB	/data
host2	192.168.1.102	PD2	/data
host3	192.168.1.103	PD3	/data
host4	192.168.1.104	TiKV1	/data
host5	192.168.1.105	TiKV2	/data
host6	192.168.1.106	TiKV3	/data

2.2.1 1. 启动 PD

登录 **host1** 执行 :

```
docker run -d --name pd1 \
  -p 2379:2379 \
  -p 2380:2380 \
  -v /etc/localtime:/etc/localtime:ro \
  -v /data:/data \
  pingcap/pd:latest \
  --name="pd1" \
```

```
  --data-dir="/data/pd1" \
  --client-urls="http://0.0.0.0:2379" \
  --advertise-client-urls="http://192.168.1.101:2379" \
  --peer-urls="http://0.0.0.0:2380" \
  --advertise-peer-urls="http://192.168.1.101:2380" \
  --initial-cluster="pd1=http://192.168.1.101:2380, \
                     pd2=http://192.168.1.102:2380, \
                     pd3=http://192.168.1.103:2380"
```

登录 **host2** 执行：

```
docker run -d --name pd2 \
  -p 2379:2379 \
  -p 2380:2380 \
  -v /etc/localtime:/etc/localtime:ro \
  -v /data:/data \
  pingcap/pd:latest \
  --name="pd2" \
  --data-dir="/data/pd2" \
  --client-urls="http://0.0.0.0:2379" \
  --advertise-client-urls="http://192.168.1.102:2379" \
  --peer-urls="http://0.0.0.0:2380" \
  --advertise-peer-urls="http://192.168.1.102:2380" \
  --initial-cluster="pd1=http://192.168.1.101:2380, \
                     pd2=http://192.168.1.102:2380, \
                     pd3=http://192.168.1.103:2380"
```

登录 **host3** 执行：

```
docker run -d --name pd3 \
  -p 2379:2379 \
  -p 2380:2380 \
  -v /etc/localtime:/etc/localtime:ro \
  -v /data:/data \
  pingcap/pd:latest \
  --name="pd3" \
  --data-dir="/data/pd3" \
  --client-urls="http://0.0.0.0:2379" \
  --advertise-client-urls="http://192.168.1.103:2379" \
  --peer-urls="http://0.0.0.0:2380" \
  --advertise-peer-urls="http://192.168.1.103:2380" \
  --initial-cluster="pd1=http://192.168.1.101:2380, \
                     pd2=http://192.168.1.102:2380, \
                     pd3=http://192.168.1.103:2380"
```

2.2.2 2. 启动 TiKV

登录 **host4** 执行：

```
docker run -d --name tikv1 \
  -p 20160:20160 \
  -v /etc/localtime:/etc/localtime:ro \
  -v /data:/data \
  pingcap/tikv:latest \
  --addr="0.0.0.0:20160" \
  --advertise-addr="192.168.1.104:20160" \
  --store="/data/tikv1" \
  --pd="192.168.1.101:2379,192.168.1.102:2379,192.168.1.103:2379"
```

登录 **host5** 执行：

```
docker run -d --name tikv2 \
  -p 20160:20160 \
  -v /etc/localtime:/etc/localtime:ro \
  -v /data:/data \
  pingcap/tikv:latest \
  --addr="0.0.0.0:20160" \
  --advertise-addr="192.168.1.105:20160" \
  --store="/data/tikv2" \
  --pd="192.168.1.101:2379,192.168.1.102:2379,192.168.1.103:2379"
```

登录 **host6** 执行：

```
docker run -d --name tikv3 \
  -p 20160:20160 \
  -v /etc/localtime:/etc/localtime:ro \
  -v /data:/data \
  pingcap/tikv:latest \
  --addr="0.0.0.0:20160" \
  --advertise-addr="192.168.1.106:20160" \
  --store="/data/tikv3" \
  --pd="192.168.1.101:2379,192.168.1.102:2379,192.168.1.103:2379"
```

2.2.3 3. 启动 TiDB

登录 **host1** 执行：

```
docker run -d --name tidb \
  -p 4000:4000 \
  -p 10080:10080 \
  -v /etc/localtime:/etc/localtime:ro \
  pingcap/tidb:latest \
  --store=tikv \
  --path="192.168.1.101:2379,192.168.1.102:2379,192.168.1.103:2379"
```

2.2.4 4. 使用 MySQL 标准客户端连接 TiDB 测试

登录 **host1** 并确保已安装 mysql 命令行客户端，执行：

```
$ mysql -h 127.0.0.1 -P 4000 -u root -D test
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| INFORMATION_SCHEMA |
| PERFORMANCE_SCHEMA |
| mysql              |
| test               |
+--------------------+
4 rows in set (0.00 sec)
```

2.2.5 如何自定义配置文件

TiKV 和 PD 可以通过指定配置文件的方式来加载更加丰富的启动参数，用于性能调优

假定配置文件在宿主机上的存放路径 `/path/to/config/pd.toml` 和 `/ path/to/config/tikv.toml`

启动 Docker 时需要调整相应的启动参数，以 tikv1 和 pd1 为例：

```
docker run -d --name tikv1 \
  -p 20160:20160 \
  -v /etc/localtime:/etc/localtime:ro \
  -v /data:/data \
  -v /path/to/config/tikv.toml:/tikv.toml:ro \
  pingcap/tikv:latest \
  --addr="0.0.0.0:20160" \
  --advertise-addr="192.168.1.104:20160" \
  --store="/data/tikv1" \
  --pd="192.168.1.101:2379,192.168.1.102:2379,192.168.1.103:2379" \
  --config="/tikv.toml"
```

```
docker run -d --name pd1 \
  -p 2379:2379 \
  -p 2380:2380 \
  -v /etc/localtime:/etc/localtime:ro \
  -v /data:/data \
  -v /path/to/config/pd.toml:/pd.toml:ro \
  pingcap/pd:latest \
  --name="pd1" \
  --data-dir="/data/pd1" \
  --client-urls="http://0.0.0.0:2379" \
  --advertise-client-urls="http://192.168.1.101:2379" \
  --peer-urls="http://0.0.0.0:2380" \
  --advertise-peer-urls="http://192.168.1.101:2380" \
  --initial-cluster="pd1=http://192.168.1.101:2380, \
                     pd2=http://192.168.1.102:2380, \
                     pd3=http://192.168.1.103:2380" \
  --config="/pd.toml"
```

3 参数解释

3.1 TiDB

3.1.1 –store

- 用来指定 TiDB 底层使用的存储引擎
- 默认: “goleveldb”
- 你可以选择 “memory”, “goleveldb”, “BoltDB” 或者 “TiKV”。前面三个是本地存储引擎，而 TiKV 是一个分布式存储引擎。
- 例如，如果我们可以通过 `tidb-server --store=memory` 来启动一个纯内存引擎的 TiDB。

3.1.2 –path

- 对于本地存储引擎 “goleveldb”, “BoltDB” 来说，path 指定的是实际的数据存放路径。
- 对于 “memory” 存储引擎来说，path 不用设置。
- 对于 “TiKV” 存储引擎来说，path 指定的是实际的 PD 地址。假设我们在 192.168.100.113:2379, 192.168.100.114:2379 和 192.168.100.115:2379 上面部署了 PD，那么 path 为 “192.168.100.113:2379,192.168.100.114:2379,192.168.100.115:2379”。
- 默认: “/tmp/tidb”

3.1.3 -L

- Log 级别
- 默认: “info”
- 我们能选择 debug, info, warn, error 或者 fatal.

3.1.4 –log-file

- Log 文件
- 默认: “”
- 如果没设置这个参数，log 会默认输出到 “stderr”，如果设置了，log 就会输出到对应的文件里面，在每天凌晨，log 会自动轮转使用一个新的文件，并且将以前的文件改名备份。

3.1.5 –host

- TiDB 服务监听 host。
- 默认: “0.0.0.0”
- TiDB 服务会监听这个 host。
- 0.0.0.0 默认会监听所有的网卡 address。如果有多块网卡，可以指定对外提供服务的网卡，譬如 192.168.100.113。

3.1.6 -P

- TiDB 服务监听端口。
- 默认: “4000”
- TiDB 服务将会使用这个端口接受 MySQL 客户端发过来的请求。

3.1.7 –status

- TiDB 服务状态监听端口。
- 默认: “10080”
- 这个端口是为了展示 TiDB 内部数据用的。包括 prometheus 统计 以及 pprof。
- Prometheus 统计可以通过 “http://host:status_port/metrics” 访问。
- Pprof 数据可以通过 “http://host:status_port/debug/pprof” 访问。

3.1.8 –lease

- Schema 的租约时间，单位：秒。
- 默认: “1”
- Schema 的 lease 主要用在 online schema changes 上面。这个值会影响到实际的 DDL 语句的执行时间。千万不要随便改动这个值，除非你能知道相关的内部机制。

3.1.9 –socket

- TiDB 服务使用 unix socket file 方式接受外部连接。
- 默认: “”
- 譬如我们可以使用 “/tmp/tidb.sock” 来打开 unix socket file。

3.1.10 –perfschema

- 使用 true/false 来打开或者关闭性能 schema。
- 默认: false
- 值可以是 (true) or (false)。性能 Schema 可以帮助我们在运行时检测内部的执行情况。可以通过 performance schema 获取更多信息。但需要注意，开启性能 Schema，会影响 TiDB 的性能。

3.1.11 –report-status

- 打开 (true) 或者关闭 (false) 服务状态监听端口.
- 默认: true
- 值可以为 (true) 或者 (false). (true) 表明我们开启状态监听端口。(false) 表明关闭。

3.1.12 –metrics-addr

- Prometheus Push Gateway 地址。
- 默认: “”
- 如果为空，TiDB 不会将统计信息推送给 Push Gateway。

3.1.13 –metrics-intervel

- 推送统计信息到 Prometheus Push Gateway 的时间间隔。
- 默认: 15s
- 设置为 0 表明不推送统计信息给 Push Gateway。

3.2 Placement Driver (PD)

3.2.1 -L

- Log 级别
- 默认: “info”
- 我们能选择 debug, info, warn, error 或者 fatal.

3.2.2 –log-file

- Log 文件
- 默认: “”
- 如果没设置这个参数，log 会默认输出到 “stderr”，如果设置了，log 就会输出到对应的文件里面，在每天凌晨，log 会自动轮转使用一个新的文件，并且将以前的文件改名备份。

3.2.3 –config

- 配置文件
- 默认: “”
- 如果你指定了配置文件，PD 会首先读取配置文件的配置。然后如果对应的配置在命令行参数里面也存在，PD 就会使用命令行参数的配置来覆盖配置文件里面的。

3.2.4 –name

- 当前 PD 的名字。
- 默认: “pd”
- 如果你需要启动多个 PD，一个要给 PD 使用不同的名字。

3.2.5 –data-dir

- PD 存储数据路径。
- 默认: “default.${name}”

3.2.6 –client-urls

- 处理客户端请求监听 URL 列表
- 默认: “http://127.0.0.1:2379”
- 如果部署一个集群，–client-urls 必须指定当前主机的 IP 地址，例如 “http://192.168.100.113:2379”，如果是运行在 docker 则需要指定为 “http://0.0.0.0:2379”。

3.2.7 –advertise-client-urls

- 对外客户端访问 URL 列表。
- 默认: ${client-urls}
- 在某些情况下，譬如 docker，或者 NAT 网络环境，客户端并不能通过 PD 自己监听的 client URLs 来访问到 PD，这时候，你就可以设置 advertise urls 来让客户端访问。
- 例如，docker 内部 IP 地址为 172.17.0.1，而宿主机的 IP 地址为 192.168.100.113 并且设置了端口映射 -p 2379:2379，那么可以设置为 –advertise-client-urls="http://192.168.100.113:2379"，客户端可以通过 http://192.168.100.113:2379 来找到这个服务。

3.2.8 –peer-urls

- 处理其他 PD 节点请求监听 URL 列表。
- default: "http://127.0.0.1:2380"
- 如果部署一个集群，–peer-urls 必须指定当前主机的 IP 地址，例如 "http://192.168.100.113:2380"，如果是运行在 docker 则需要指定为 "http://0.0.0.0:2380"。

3.2.9 –advertise-peer-urls

- 对外其他 PD 节点访问 URL 列表。
- 默认: ${peer-urls}
- 在某些情况下，譬如 docker，或者 NAT 网络环境，其他节点并不能通过 PD 自己监听的 peer URLs 来访问到 PD，这时候，你就可以设置 advertise urls 来让其他节点访问。
- 例如，docker 内部 IP 地址为 172.17.0.1，而宿主机的 IP 地址为 192.168.100.113 并且设置了端口映射 -p 2380:2380，那么可以设置为 –advertise-peer-urls="http://192.168.100.113:2380"，其他 PD 节点可以通过 http://192.168.100.113:2380 来找到这个服务。

3.2.10 –initial-cluster

- 初始化 PD 集群配置。
- 默认: "{name}=http://{advertise-peer-url}"
- 例如，如果 `name` 是 "pd", 并且 `advertise-peer-urls` 是 "http://192.168.100.113:2380", 那么 `initial-cluster` 就是

- 如果你需要启动三台 PD，那么 `initial-cluster` 可能就是

```
pd1=http://192.168.100.113:2380, pd2=http://192.168.100.114:2380,pd3=192.168.100.115:2380
```

3.2.11 –join

- 动态加入 PD 集群。
- 默认: ""
- 如果你想动态将一台 PD 加入集群，你可以使用 `--join="${advertise-client-urls}"`，`advertise-client-url` 是当前集群里面任意 PD 的 `advertise-client-url`，你也可以使用多个 PD 的，需要用逗号分隔。

3.3 TiKV

TiKV 在命令行参数上面支持一些可读性好的单位转换.

- 文件大小 (以 bytes 为单位) : KB, MB, GB, TB, PB (也可以全小写)。
- 时间 (以毫秒为单位) : ms, s, m, h。

3.3.1 -A, –addr

- TiKV 监听地址。
- 默认: “127.0.0.1:20160”
- 如果部署一个集群，–addr 必须指定当前主机的 IP 地址，例如 “http://192.168.100.113:20160”，如果是运行在 docker 则需要指定为 “http://0.0.0.0:20160”。

3.3.2 –advertise-addr

- TiKV 对外访问地址。
- 默认: ${addr}
- 在某些情况下，譬如 docker，或者 NAT 网络环境，客户端并不能通过 TiKV 自己监听的地址来访问到 TiKV，这时候，你就可以设置 advertise addr 来让客户端访问。
- 例如，docker 内部 IP 地址为 172.17.0.1，而宿主机的 IP 地址为 192.168.100.113 并且设置了端口映射 -p 20160: 20160，那么可以设置为 –advertise-addr=“192.168.100.113:20160”，客户端可以通过 192.168.100.113:20160 来找到这个服务。

3.3.3 -L, –Log

- Log 级别
- 默认: “info”
- 我们能选择 trace, debug, info, warn, error, 或者 off。

3.3.4 –log-file

- Log 文件
- 默认: “”
- 如果没设置这个参数，log 会默认输出到 “stderr”，如果设置了，log 就会输出到对应的文件里面，在每天凌晨，log 会自动轮转使用一个新的文件，并且将以前的文件改名备份。

3.3.5 -C, –config

- 配置文件
- 默认: “”
- 如果你指定了配置文件，TiKV 会首先读取配置文件的配置。然后如果对应的配置在命令行参数里面也存在，TiKV 就会使用命令行参数的配置来覆盖配置文件里面的。

3.3.6 -s, –store

- TiKV 数据存储路径。
- 默认: “/tmp/tikv/store”

3.3.7 –capacity

- TiKV 存储数据的容量。
- 默认: 0 (无限)
- PD 需要使用这个值来对整个集群做 balance 操作。(提示：你可以使用 10GB 来替代 1073741824，从而简化参数的传递)。

3.3.8 –pd

- PD 地址列表。
- 默认: “”
- TiKV 必须使用这个值连接 PD，才能正常工作。使用逗号来分隔多个 PD 地址，例如：“192.168.100.113:2379,192.168.100.114:2379,192.168.100.115:2379”。

4 TiDB 监控框架概述

TiDB 使用开源时序数据库 Prometheus 作为监控和性能指标信息存储方案，使用 Grafana 作为可视化组件进行展示。

Prometheus 是一个拥有多维度数据模型，灵活的查询语句的时序数据库。Prometheus 作为热门的开源项目，拥有活跃的社区及众多的成功案例。

Prometheus 提供了多个组件供用户使用。目前，我们使用 Prometheus Server，来收集和存储时间序列数据。Client 代码库，在程序中定制需要的 Metric 。Push GateWay 来接收 Client Push 上来的数据，统一供 Prometheus 主服务器抓取。以及 AlertManager 来实现报警机制。其结构如下图：

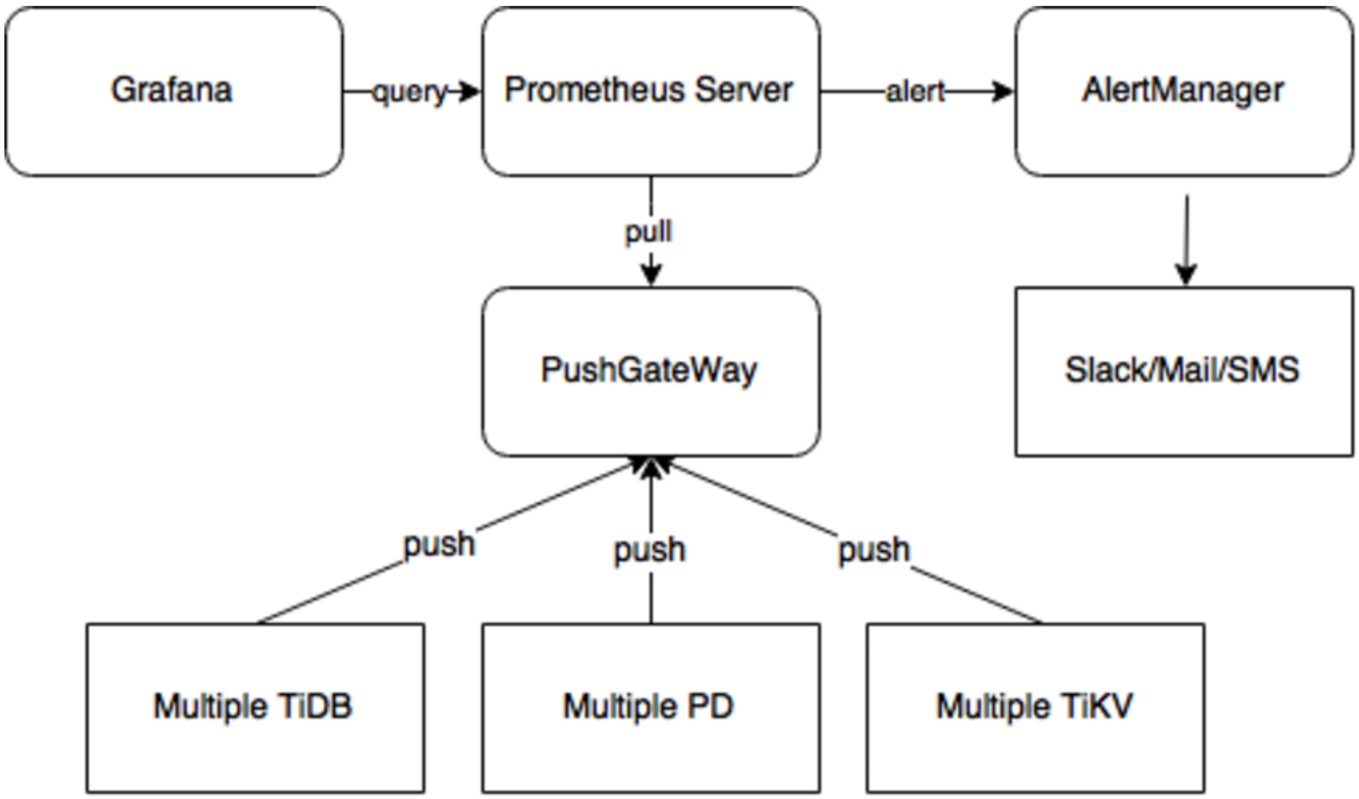

Figure 2: Prometheus in TiDB

Grafana 是一个开源的 metric 分析及可视化系统。我们使用 Grafana 来展示 TiDB 的各项性能指标。如下图所示:

5 TiDB 集群监控

TiDB 集群状态监控目前有两种接口，第一种是通过 HTTP 接口对外汇报组件的信息，我们称之为组件的状态接口；第二种是使用 prometheus 记录组件中各种操作的详细信息，我们称之为 Metrics 接口。

5.1 组件状态接口

这类接口可以获取组件的一些基本信息，并且可以作为 keepalive 监测接口。另外 PD 的接口可以看到整个 TiKV 集群的详细信息。

5.1.1 TiDB Server

TiDB 对外暴露的 HTTP 接口是 http://host:port/status，默认的端口号是 10080（可以通过 –status 参数设置），可以通过访问这个接口获取当前 TiDB Server 的状态，以及判断是否存活。返回结果是 **Json** 格式：

```
curl http://127.0.0.1:10080/status
{
```

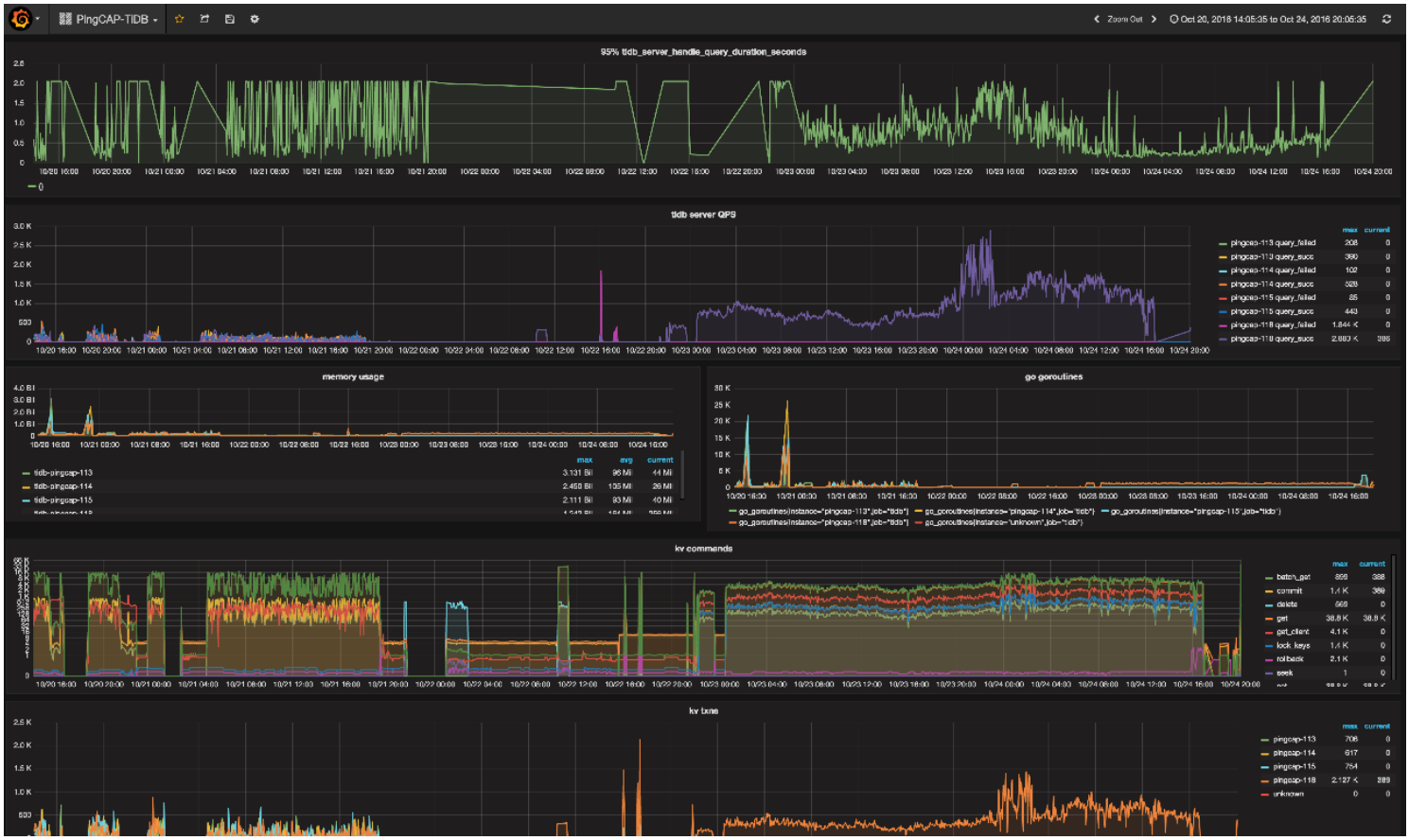

Figure 3: Grafana Screeshot

```
connections: 0,
version: "5.5.31-TiDB-1.0",
git_hash: "b99521846ff6f71f06e2d49a3f98fa1c1d93d91b"
}
```

- connection: 当前 TiDB Server 上的客户端连接数
- version: TiDB 版本号
- git_hash: TiDB 当前代码的 Git Hash

5.1.2 PD Server

PD API 地址：http://$host$:{port}/pd/api/v1/${api_name} 其中 port 默认为 2379，各类 api_name 详细信息参见 PD API Doc 通过这个接口可以获取当前所有 TiKV 的状态以及负载均衡信息。其中最重要也是最常用的接口获取 TiKV 集群所有节点状态的接口，下面以一个单个 TiKV 构成的集群为例，说明一些用户需要了解的信息：

```
curl http://127.0.0.1:2379/pd/api/v1/stores
{
  "count": 1,  TiKV 节点数量
  "stores": [  // TiKV 节点的列表
   // 下面列出的是这个集群中单个 TiKV 节点的信息
   {
      "store": {
        "id": 1,
        "address": "127.0.0.1:22161",
        "state": 0
      },
      "status": {
        "store_id": 1,                // 节点的 ID
        "capacity": 1968874332160,    // 存储总容量
        "available": 1264847716352,  // 存储剩余容量
        "region_count": 1,            // 该节点上存放的  Region 数量
        "sending_snap_count": 0,
        "receiving_snap_count": 0,
        "start_ts": "2016-10-24T19:54:00.110728339+08:00",             // 启动时间
        "last_heartbeat_ts": "2016-10-25T10:52:54.973669928+08:00",    // 最后一次心跳时间
        "total_region_count": 1,           // 总 Region 数量
        "leader_region_count": 1,        // Leader Region 数量
        "uptime": "14h58m54.862941589s"
      },
      "scores": [
        100,
        35
      ]
    }
  ]
}
```

5.2 Metrics 监控

这部分主要对整个集群的状态、性能做监控，通过 Prometheus+Grafana 展现 metrics 数据，在下面一节会介绍如何搭建监控系统。

5.2.1 TiDB Server

- query 处理时间，可以看到延迟和吞吐
- ddl 过程监控
- TiKV client 相关的监控

- PD client 相关的监控

5.2.2 PD Server

- 命令执行的总次数
- 某个命令执行失败的总次数
- 某个命令执行成功的耗时统计
- 某个命令执行失败的耗时统计
- 某个命令执行完成并返回结果的耗时统计

5.2.3 TiKV Server

- GC 监控
- 执行 KV 命令的总次数
- Scheduler 执行命令的耗时统计
- Raft propose 命令的总次数
- Raft 执行命令的耗时统计
- Raft 执行命令失败的总次数
- Raft 处理 ready 状态的总次数

5.3 使用 Prometheus+Grafana

5.3.1 部署架构

整个架构如下图所示，在 TiDB/PD/TiKV 三个组件的启动参数中添加 Prometheus Pushgateway 地址:

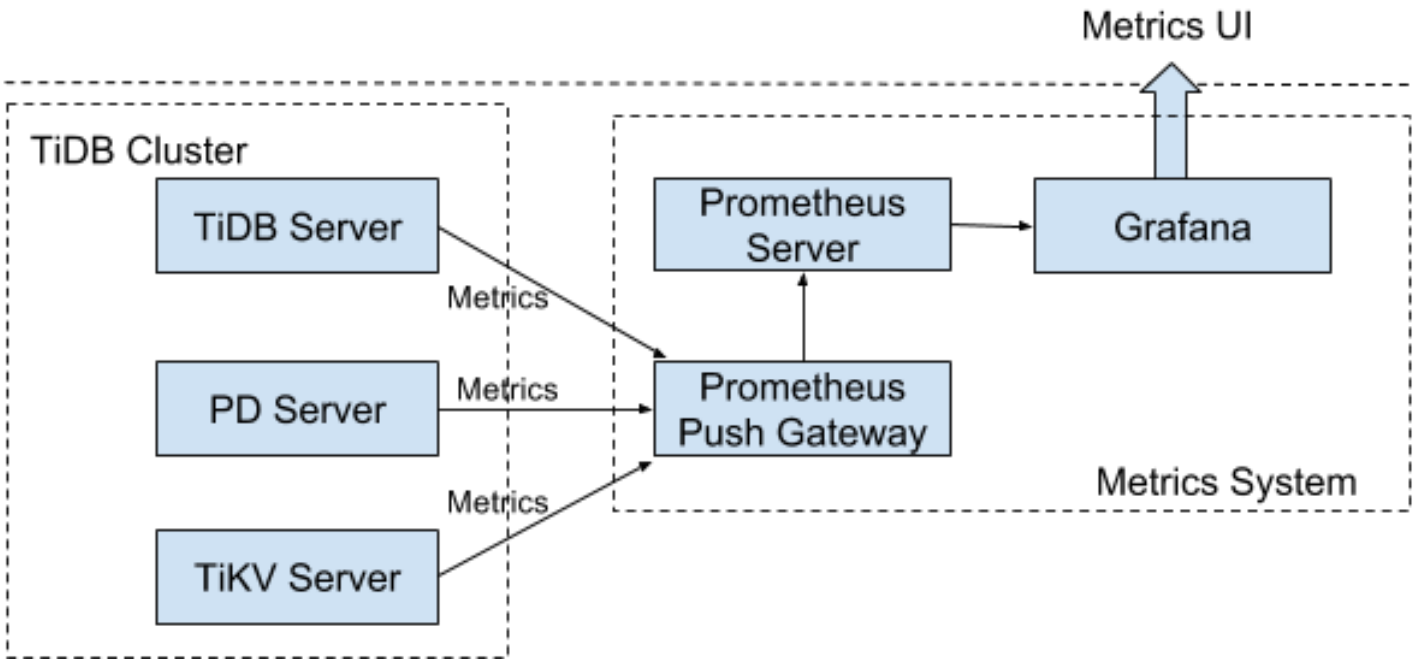

Figure 4: Deployment Architecture

5.3.2 搭建监控系统

Prometheus Push Gateway 参考：https://github.com/prometheus/pushgateway

Prometheus Server 参考：https://github.com/prometheus/prometheus#install

Grafana 参考：http://docs.grafana.org

5.3.3 配置

5.3.3.1 TiDB/PD/TiKV 配置

- TiDB 设置 –metrics-addr 和 –metrics-interval 两个参数，其中 metrics-addr 设为 Push Gateway 的地址，metrics-interval 为 push 的频率，单位为秒，默认值为 15
- PD 修改 toml 配置文件，填写 Push Gateway 的地址和推送频率

```
[metric]
# prometheus client push interval, set "0s" to disable prometheus.
interval = "15s"
# prometheus pushgateway address, leaves it empty will disable prometheus.
address = "host:port"
```

- TiKV 修改 toml 配置文件，填写 Push Gateway 的地址和推送频率，job 字段一般设为 “tikv”

```
[metric]
# the Prometheus client push interval. Setting the value to 0s stops Prometheus client from pushing.
interval = "15s"
# the Prometheus pushgateway address. Leaving it empty stops Prometheus client from pushing.
address = "host:port"
# the Prometheus client push job name. Note: A node id will automatically append, e.g., "tikv_1".
job = "tikv"
```

5.3.3.2 PushServer 配置

一般无需特殊配置，使用默认端口 9091 即可

- Prometheus 配置在 yaml 配置文件中添加 Push Gateway 地址：“`yaml scrape_configs:  # The job name is added as a labeljob=`’ to any timeseries scraped from this config.
- job_name: ‘TiDB’

6 Override the global default and scrape targets from this job every 5 seconds.

scrape_interval: 5s

honor_labels:true

static_configs:

- targets: [’host:port’] # 这里填写 pushgateway 地址 labels: group: ‘production’ “’ #### Grafana 配置

- 进入 Grafana Web 界面（默认地址: http://localhost:3000，默认账号: admin 密码: admin ）

点击 Grafana Logo -> 点击 Data Sources -> 点击 Add data source -> 填写 data source 信息（注: Type 选 Prometheus，Url 为 Prometheus 地址，其他根据实际情况填写 ）

- 导入 dashboard 配置文件

点击 Grafana Logo -> 点击 Data Sources -> 点击 Import -> 选择需要的 dashboard 配置文件上传 -> 选择对应的 data source

7 PD Control 使用说明

PD Control 是 PD 的命令行工具，用于获取集群状态信息和调整集群

7.1 源码编译

1. *Go* Version 1.7 以上
2. 在 PD 项目根目录使用 `make` 命令进行编译，生成 bin/pd-ctl

7.2 参数说明及示例

7.2.1 简单例子:

单命令模式：

```
./pd-ctl store -d -u 127.0.0.1:2379
```

交互模式:

```
./pd-ctl -u 127.0.0.1:2379
```

使用环境变量:

```
export PD_ADDR=http://127.0.0.1:2379
./pd-ctl
```

7.2.2 标志 (flags)

7.2.2.1 –pd,-u

- 指定 PD 的地址
- 默认地址: http://127.0.0.1:2379
- 环境变量: PD_ADDR

7.2.2.2 –detach,-d

- 使用单命令行模式 (不进入 readline)
- 默认值: false

7.2.3 命令 (command)

7.2.3.1 store [delete]

用于显示 store 信息或者删除指定 store

7.2.3.1.1 示例

```
>> store            // 显示所有 store 信息
{
  "count": 3,
  "stores": [...]
}
>> store 1          // 获取 store id 为 1 的store
  ......
>> store delete 1   // 下线 store id 为 1 的 store
  ......
```

7.2.3.2 config [show | set <option> <value>]

用于调整配置信息 ##### 示例

```
>> config show                              //  显示 config 的信息
{
  "min-region-count": 10,
  "min-leader-count": 10,
  "max-snapshot-count": 3,
  "min-balance-diff-ratio": 0.01,
  "max-store-down-duration": "30m0s",
  "leader-schedule-limit": 8,
  "leader-schedule-interval": "10s",
  "storage-schedule-limit": 4,
  "storage-schedule-interval": "30s"
}
>> config set leader-schedule-interval 20s  // 设置 leader-schedule-interval 为 20 s
Success!
```

7.2.3.3 Member [leader | delete]

用于显示 PD 成员信息或删除指定成员 ##### 示例

```
>> member                                 // 显示所有成员的信息
{
  "members": [......]
}
>> member leader                          // 显示 leader 的信息
{
  "name": "pd",
  "addr": "http://192.168.199.229:2379",
  "id": 9724873857558226554
}
>> member delete pd2                      // 下线 PD2
Success!
```

7.2.3.4 Region

用于显示 Region 信息 ##### 示例

```
>> region                                 //  显示所有 region 信息
{
  "count": 1,
  "regions": [......]
}

>> region 2                               // 显示 region id 为 2 的信息
{
  "region": {
      "id": 2,
      ......
  }
  "leader": {
      ......
  }
}
```

8 TiDB SQL 语法文档

TiDB 支持 SQL92 标准并兼容 MySQL 语法，目前已经实现了大多数常用的 MySQL 语法。

8.1 目录

9 与 MySQL 兼容性对比

TiDB 支持包括跨行事务，JOIN 及子查询在内的绝大多数 MySQL 的语法，用户可以直接使用现有的 MySQL 客户端连接。如果现有的业务已经基于 MySQL 开发，大多数情况不需要修改代码即可直接替换单机的 MySQL。

包括现有的大多数 MySQL 运维工具（如 PHPMyAdmin, Navicat, MySQL Workbench 等），以及备份恢复工具（如 mysqldump, mydumper/myloader）等都可以直接使用。

不过一些特性由于在分布式环境下没法很好的实现，目前暂时不支持或者是表现与 MySQL 有差异。

9.1 不支持的特性

- 存储过程
- 视图
- 触发器
- 自定义函数
- 外键约束
- 全文索引
- 空间索引
- 非 UTF8 字符集
- Json 类型

9.2 与 MySQL 有差异的特性

9.2.1 自增 ID

TiDB 的自增 ID (Auto Increment ID) 只保证自增且唯一，并不保证连续分配。TiDB 目前采用批量分配的方式，所以如果在多台 TiDB 上同时插入数据，分配的自增 ID 会不连续。

9.2.2 内建函数

TiDB 支持常用的 MySQL 内建函数，但是不是所有的函数都已经支持，具体请参考语法文档。

9.2.3 DDL

TiDB 实现了 F1 的异步 Schema 变更算法，DDL 执行过程中不会阻塞线上的 DML 操作。目前已经支持的 DDL 包括：+ Create Database + Drop Database + Create Table + Drop Table + Add Index + Drop Index + Add Column + Drop Column + Truncate Table

9.2.4 事务

TiDB 使用乐观事务模型，在执行 Update、Insert、Delete 等语句时，只有在提交过程中才会检查写写冲突，而不是像 MySQL 一样使用行锁来避免写写冲突。所以业务端在执行 SQL 语句后，需要注意检查 commit 的返回值，即使执行时没有出错，commit 的时候也可能会出错。

10 TiDB 中文 FAQ

10.0.1 TiDB 是什么?

TiDB 是一个分布式 NewSQL 数据库。支持水平扩展、高可用、ACID 事务、SQL 等特性。同时 TiDB 还支持 MySQL 语法和 MySQL 协议。

10.0.2 TiDB 是基于 MySQL 开发的吗?

不是。虽然 TiDB 支持 MySQL 语法和协议，但是 TiDB 是由 PingCAP 团队完全自主开发的产品。

10.0.3 TiDB 和 TiKV 是如何配合使用？他们之间的关系是？

TiDB 是 SQL 层，主要负责 SQL 的解析、制定查询计划、生成执行器；TiKV 是分布式 Key-Value 存储引擎，用来存储真正的数据。简而言之，TiKV 是 TiDB 的存储引擎。

10.0.4 Placement Driver (PD) 是做什么的?

PD 是 TiDB 集群的管理组件，负责存储 TiKV 的元数据，同时也负责分配时间戳以及对 TiKV 做负载均衡调度。

10.0.5 TiDB 用起来简单吗？

是的，TiDB 用起来很简单。启动整套服务后，就可以将 TiDB 当做一个普通的 MySQL Server 来用，你可以将 TiDB 用在任何以 MySQL 作为后台存储服务的应用中，并且基本上不需要修改应用代码。同时你可以用大部分流行的 MySQL 管理工具来管理 TiDB。

10.0.6 TiDB 适用的场景是？

原业务的 MySQL 的业务遇到单机容量或者性能瓶颈时，可以考虑使用 TiDB 无缝替换 MySQL。TiDB 可以提供如下特性：+ 吞吐、存储和计算能力的水平扩展 + 水平伸缩时不停服务 + 强一致性 + 分布式 ACID 事务

10.0.7 TiDB 不适用于哪些场景?

如果你的应用数据量小 (所有数据千万级别行以下)，且没有高可用、强一致性或者多数据中心复制等要求，那么就不适合使用 TiDB。

10.0.8 TiDB 是如何进行权限管理的?

TiDB 遵循 MySQL 的权限管理体系，可以创建用户并授予权限。

在创建用户时，可以使用 MySQL 语法，如 `CREATE USER 'test'@'localhost' identified by '123';`，这样就添加了一个用户名为 test，密码为 123 的用户，这个用户只能从 localhost 登录。

修改用户密码时，可以使用 Set Password 语句，例如给 TiDB 的默认 root 用户增加密码：`SET PASSWORD FOR 'root'@'%' = '123';`。

在进行授权时，也可以使用 MySQL 语法，如 `GRANT SELECT ON *.* TO  'test'@'localhost';`，将读权限授予 test 用户。

请注意，在创建用户和授权时，TiDB 有几个和 MySQL 有区别：* 创建用户时，用户标识采用 user@hostname 这种语法，其中 hostname 支持精确匹配和全匹配，不支持前缀匹配，也就是只支持 “192.168.199.1” 以及 “%”，而不支持 “192.168.%” * TiDB 支持对用户授权，这里只是支持授权语法，并且记录在系统表中。但是实际上只对 DropTable 语句进行权限检查，对其他的语句并不会做权限检查。实现用户授权主要是为了兼容已有的 MySQL 业务。

10.0.9 如何对 TiDB 进行水平扩展？

当您的业务不断增长时，数据库可能会面临三方面瓶颈，第一是存储资源不够，也就是磁盘空间不够；第二是计算资源不够用，如 CPU 占用较高，第三是吞吐跟不上。这时可以对 TiDB 集群做水平扩展。

如果是存储资源不够，可以通过添加 TiKV Server 节点来解决。新节点启动后，PD 会自动将其他节点的部分数据迁移过去，无需人工介入。

如果是计算资源不够，可以查看 TiDB Server 和 TiKV Server 节点的 CPU 消耗情况，再考虑添加 TiDB Server 节点或者是 TiKV Server 节点来解决。如添加 TiDB Server 节点，将其配置在前端的 Load Balancer 之后即可。

如果是吞吐跟不上，一般可以考虑同时增加 TiDB Server 和 TiKV Server 节点。

10.0.10 TiDB 高可用的特性是怎么样的？

高可用是 TiDB 的另一大特点，TiDB/TiKV/PD 这三个组件都能容忍部分实例失效，不影响整个集群的可用性。具体见 TiDB 高可用性

10.0.11 TiDB 的强一致特性是什么样的？

TiDB 使用 Raft 在多个副本之间做数据同步，从而保证数据的强一致。单个副本失效时，不影响数据的可靠性。

10.0.12 TiDB 支持分布式事务吗？

TiDB 支持 ACID 分布式事务。事务模型是以 Google 的 Percolator 模型为基础，并做了一些优化。这个模型需要一个时间戳分配器，分配唯一且递增的时间戳。在 TiDB 集群中，PD 承担时间戳分配器的角色。

10.0.13 在使用 TiDB 时，我需要用什么编程语言？

可以用任何你喜欢的编程语言，只要该语言有 MySQL Client/Driver.

10.0.14 TiDB 的 lease 参数应该如何设置？

启动 TiDB Server 时，需要通过命令行参数设置 lease 参数（–lease=60），其值会影响 DDL 的速度（只会影响当前执行 DDL 的 session，其他的 session 不会受影响）。在测试阶段，lease 的值可以设为 1s，加快测试进度；在生产环境下，我们推荐这个值设为分钟级（一般可以设为 60），这样可以保证 DDL 操作的安全。

10.0.15 在使用 TiDB 时，DDL 语句为什么这么慢？

TiDB 实现了 Google F1 的在线 Schema 变更算法（具体参见 F1 论文 和我们的一篇 Blog）。一般情况下，DDL 并不是一个频繁的操作，我们首先要保证的是数据的一致性以及线上业务不受影响。一个完整的 DDL 过程会有 2 到 5 个阶段（取决于语句类型），每个阶段至少会执行 2*lease 时间，假设 lease 设为 1 分钟，对于 Drop Table 语句（需要两个阶段），会执行 2*2*1 = 4 分钟。除此之外，DDL 的时间还取决其他的条件，比如做 Add Index 操作时，表中已有的数据量是影响 DDL 时间的主要因素。我们也了解过 Google 内部在 F1 上是如何做 DDL，一般是提交给 DBA，DBA 再通过专用的工具执行，执行的时间会很长。

10.0.16 和 MySQL/Oracle 等传统关系型数据库相比，TiDB 有什么优势？

和这些数据库相比，TiDB 最大的特点是可以无限水平扩展，在此过程中不损失事务特性。

10.0.17 和 Cassandra/Hbase/MongoDB 等 NoSQL 数据库相比，TiDB 有什么优势?

TiDB 在提供水平扩展特性的同时，还能提供 SQL 以及分布式事务。

10.0.18 如何将一个运行在 MySQL 上的应用迁移到 TiDB 上?

TiDB 支持绝大多数 MySQL 语法，一般不需要修改代码。我们提供了一个检查工具，用于检查 MySQL 中的 Schema 是否和 TiDB 兼容。

10.0.19 TiDB 是否支持其他存储引擎?

是的。除了 TiKV 之外，TiDB 还支持一些流行的单机存储引擎，比如 GolevelDB, RocksDB, BoltDB 等。如果一个存储引擎是支持事务的 KV 引擎，并且能提供一个满足 TiDB 接口要求的 Client，即可接入 TiDB。

10.0.20 使用 go get 方式安装 TiDB 为什么报错了?

请手动将 TiDB 克隆到 GOPATH 目录，然后运行 `make` 命令。TiDB 是一个项目而不是一个库，它的依赖比较复杂，并且 parser 也是根据 `parser.y` 生成的，我们不支持 `go get` 方式，而是使用 Makefile 来管理。

如果你是开发者并且熟习 Go 语言，你可以尝试在 TiDB 项目的根目录运行 `make parser; ln -s _vendor/src vendor`，之后就可以使用 `go run`, `go test` `go install` 等命令，但是并不推荐这种做法。

10.0.21 为什么修改了 TiKV/PD 的 toml 配置文件，却没有生效？

如果要使用配置文件，请设置 TiKV/PD 的 –config 参数，TiKV/PD 默认情况下不会读取配置文件。

10.0.22 为什么 TiKV 数据目录不见了

TiKV 的 –store 参数默认值为 `/tmp/tikv/store`，在某些虚拟机中，重启操作系统会删除 /tmp 目录下的数据，推荐通过 –store 参数显式设置 TiKV 数据目录。

10.0.23 TiKV 启动报错：cluster ID mismatch

TiKV 本地存储的 cluster ID 和指定的 PD 的 cluster ID 不一致。在部署新的 PD 集群的时候，PD 会随机生成一个 cluster ID，TiKV 第一次初始化的时候会从 PD 获取 cluster ID 存储在本地，下次启动的时候会检查本地的 cluster ID 与 PD 的 cluster ID 是否一致，如果不一致则会报错并退出。出现这个错误一个常见的原因是，用户原先部署了一个集群，后来把 PD 的数据删除了并且重新部署了新的 PD，但是 TiKV 还是使用旧的数据重启连到新的 PD 上，就会报这个错误。

11 TiDB 集群故障诊断

当试用 TiDB 遇到问题时，请先参考本篇文档。如果问题未解决，请按文档要求收集必要的信息通过 Github 提供给 TiDB 开发者。

11.1 如何给 TiDB 开发者报告错误

当使用 TiDB 遇到问题并且通过后面所列信息无法解决时，请收集以下信息并创建新 Issue: + 具体的出错信息以及正在执行的操作 + 当前所有组件的状态 + 出问题组件 log 中的 error/fatal/panic 信息 + 机器配置以及部署拓扑 + dmesg 中 TiDB 组件相关的问题

11.2 数据库连接不上

首先请确认集群的各项服务是否已经启动，包括 tidb-server、pd-server、tikv-server。请用 ps 命令查看所有进程是否在。如果某个组件的进程已经不在了，请参考对应的章节排查错误。

如果所有的进程都在，请查看 tidb-server 的日志，看是否有报错？常见的错误包括：+ InfomationSchema is out of date

无法连接 tikv-server，请检查 pd-server 以及 tikv-server 的状态和日志。+ panic

程序有错误，请将具体的 panic log 提供给 TiDB 开发者。

如果是清空数据并重新部署服务，请确认以下信息：+ pd-server、tikv-server 数据都已清空

tikv-server 存储具体的数据，pd-server 存储 tikv-server 中数据的的元信息。如果只清空 pd-server 或只清空 tikv-server 的数据，会导致两边数据不匹配。+ 清空 pd-server 和 tikv-server 的数据并重启后，也需要重启 tidb-server

集群 ID 是由 pd-server 在集群初始化时随机分配，所以重新部署集群后，集群 ID 会发生变化。tidb-server 业务需要重启以获取新的集群 ID。

11.3 tidb-server 启动报错

tidb-server 无法启动的常见情况包括：+ 启动参数错误

请参考TiDB 命令行参数文档。+ 端口被占用：`lsof -i:port`

请确保 tidb-server 启动所需要的端口未被占用。+ 无法连接 pd-server

首先检查 pd-server 的进程状态和日志，确保 pd-server 成功启动，对应端口已打开：`lsof -i:port`。

若 pd-server 正常，则需要检查 tidb-server 机器和 pd-server 对应端口之间的连通性，确保网段连通且对应服务端口已添加到防火墙白名单中，可通过 nc 或 curl 工具检查。

例如，假设 tidb 服务位于 `192.168.1.100`，无法连接的 pd 位于 `192.168.1.101`，且 2379 为其 client port，则可以在 tidb 机器上执行 `nc -v -z 192.168.1.101 2379`，测试是否可以访问端口。或使用 `curl -v 192.168.1.101:2379/pd/api/v1/leader` 直接检查 pd 是否正常服务。

11.4 tikv-server 启动报错

- 启动参数错误请参考TiKV 启动参数文档。
- 端口被占用：`lsof -i:port`

请确保 tikv-server 启动所需要的端口未被占用：`lsof -i:port`。+ 无法连接 pd-server

首先检查 pd-server 的进程状态和日志。确保 pd-server 成功启动，对应端口已打开：`lsof -i:port`。

若 pd-server 正常，则需要检查 tikv-server 机器和 pd-server 对应端口之间的连通性，确保网段连通且对应服务端口已添加到防火墙白名单中，可通过 nc 或 curl 工具检查。具体命令参考上一节。

- 文件被占用不要在一个数据库文件目录上打开两个 tikv。

11.5 pd-server 启动报错

- 启动参数错误

请参考PD 命令行参数文档。+ 端口被占用：`lsof -i:port`

请确保 pd-server 启动所需要的端口未被占用：`lsof -i:port`。

11.6 TiDB/TiKV/PD 进程异常退出

- 进程是否是启动在前台

当前终端退出给其所有子进程发送 HUP 信号，从而导致进程退出。+ 是否是在命令行用过 `nohup+&` 方式直接运行

这样依然可能导致进程因终端连接突然中断，作为终端 SHELL 的子进程被杀掉。推荐将启动命令写在脚本中，通过脚本运行（相当于二次 fork 启动）。

11.7 TiDB panic

请提供 panic 的 log

11.8 连接被拒绝

- 请确保操作系统的网络参数正确，包括但不限于
- 连接字符串中的端口和 tidb-server 启动的端口是否一致
- 请保证防火墙的配置正确

11.9 Too many open files

在启动进程之前，请确保 ulimit -n 的结果足够大，推荐设为 unlimited 或者是大于 1000000。

11.10 数据库访问超时，系统负载高

首先请提供如下信息 + 部署的拓扑结构 - tidb-server/pd-server/tikv-server 部署了几个实例 - 这些实例在机器上是如何分布的 + 机器的硬件配置 - CPU 核数 - 内存大小 - 硬盘类型（SSD 还是机械硬盘）- 是实体机还是虚拟机 + 机器上除了 TiDB 集群之外是否还有其他服务 + pd-server 和 tikv-server 是否分开部署 + 目前正在进行什么操作 + 用 `top -H` 命令查看当前占用 CPU 的线程名 + 最近一段时间的网络/IO 监控数据是否有异常

12 数据迁移

12.1 概述

该文档详细介绍了如何将 MySQL 的数据迁移到 TiDB。

这里我们假定 MySQL 以及 TiDB 服务信息如下:

Name	Address	Port	User	Password
MySQL	127.0.0.1	3306	root	
TiDB	127.0.0.1	4000	root	

12.2 使用 checker 进行 Schema 检查

在迁移之前，我们可以使用 TiDB 的 `checker` 工具，来预先检查 TiDB 是否能支持需要迁移的 table schema。如果 check 某个 table schema 失败，表明 TiDB 当前并不支持，我们不能对该 table 里面的数据进行迁移。`checker` 包含在 TiDB 工具集里面，我们可以直接下载。

12.2.1 下载 TiDB 工具集

12.2.1.1 Linux

```
# 下载 tool 压缩包
wget http://download.pingcap.org/tidb-tools-latest-linux-amd64.tar.gz
wget http://download.pingcap.org/tidb-tools-latest-linux-amd64.sha256
```

```
# 检查文件完整性，返回 ok 则正确
sha256sum -c tidb-tools-latest-linux-amd64.sha256
# 解开压缩包
tar -xzf tidb-tools-latest-linux-amd64.tar.gz
cd tidb-tools-latest-linux-amd64
```

12.2.2 使用 checker 检查的一个示范

- 在 MySQL 的 test database 里面创建几张表，并插入数据:

```
USE test;
CREATE TABLE t1 (id INT, age INT, PRIMARY KEY(id)) ENGINE=InnoDB;
CREATE TABLE t2 (id INT, name VARCHAR(256), PRIMARY KEY(id)) ENGINE=InnoDB;

INSERT INTO t1 VALUES (1, 1), (2, 2), (3, 3);
INSERT INTO t2 VALUES (1, "a"), (2, "b"), (3, "c");
```

- 使用 checker 检查 test database 里面所有的 table

```
./bin/checker -host 127.0.0.1 -port 3306 -user root test
2016/10/27 13:11:49 checker.go:48: [info] Checking database test
2016/10/27 13:11:49 main.go:37: [info] Database DSN: root:@tcp(127.0.0.1:3306)/test?charset=utf8
2016/10/27 13:11:49 checker.go:63: [info] Checking table t1
2016/10/27 13:11:49 checker.go:69: [info] Check table t1 succ
2016/10/27 13:11:49 checker.go:63: [info] Checking table t2
2016/10/27 13:11:49 checker.go:69: [info] Check table t2 succ
```

- 使用 checker 检查 test database 里面某一个 table

这里，假设我们只需要迁移 table t1。

```
./bin/checker -host 127.0.0.1 -port 3306 -user root test t1
2016/10/27 13:13:56 checker.go:48: [info] Checking database test
2016/10/27 13:13:56 main.go:37: [info] Database DSN: root:@tcp(127.0.0.1:3306)/test?charset=utf8
2016/10/27 13:13:56 checker.go:63: [info] Checking table t1
2016/10/27 13:13:56 checker.go:69: [info] Check table t1 succ
Check database succ!
```

12.2.3 一个无法迁移的 table 例子

我们在 MySQL 里面创建如下表：

```
CREATE TABLE t_error (
  c timestamp(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3) ON UPDATE CURRENT_TIMESTAMP(3)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

使用 checker 进行检查，会报错，表明我们没法迁移 t_error 这张表。

```
./bin/checker -host 127.0.0.1 -port 3306 -user root test t_error
2016/10/27 13:19:28 checker.go:48: [info] Checking database test
2016/10/27 13:19:28 main.go:37: [info] Database DSN: root:@tcp(127.0.0.1:3306)/test?charset=utf8
2016/10/27 13:19:28 checker.go:63: [info] Checking table t_error
2016/10/27 13:19:28 checker.go:67: [error]
Check table t_error failed with err: line 1 column 56 near ") ON UPDATE CURRENT_TIMESTAMP(3)
) ENGINE=InnoDB DEFAULT CHARSET=latin1"
github.com/pingcap/tidb/parser/yy_parser.go:111:
github.com/pingcap/tidb/parser/yy_parser.go:124:
/home/jenkins/workspace/WORKFLOW_TOOLS_BUILDING/go/src/ \
github.com/pingcap/tidb-tools/checker/checker.go:122:  parse CREATE TABLE `t_error` (
  `c` timestamp(3) NOT NULL DEFAULT CURRENT_TIMESTAMP(3) ON UPDATE CURRENT_TIMESTAMP(3)
```

```
) ENGINE=InnoDB DEFAULT CHARSET=latin1 error
/home/jenkins/workspace/WORKFLOW_TOOLS_BUILDING/go/src/ \
github.com/pingcap/tidb-tools/checker/checker.go:114:
2016/10/27 13:19:28 main.go:68: [error] Check database test with 1 errors and 0 warnings.
```

12.3 使用 mydumper/loader 全量导入数据

我们使用 mydumper 从 MySQL 导出数据，然后用 loader 将其导入到 TiDB 里面。

注意，虽然我们也支持使用 MySQL 官方的 mysqldump 工具来进行数据的迁移工作，但相比于 mydumper/loader，性能会慢很多，对于大量数据的迁移会花费很多时间，这里我们并不推荐。

12.3.1 从 MySQL 导出数据

mydumper 是一个更强大的数据迁移工具，具体可以参考 https://github.com/maxbube/mydumper。

12.3.1.1 下载 Binary

12.3.1.1.1 Linux

```
# 下载 mydumper 压缩包
wget http://download.pingcap.org/mydumper-linux-amd64.tar.gz
wget http://download.pingcap.org/mydumper-linux-amd64.sha256

# 检查文件完整性，返回 ok 则正确
sha256sum -c mydumper-linux-amd64.sha256
# 解开压缩包
tar -xzf mydumper-linux-amd64.tar.gz
cd mydumper-linux-amd64
```

12.3.1.2 从 MySQL 导出数据

我们使用 mydumper 从 MySQL 导出数据，如下:

```
./bin/mydumper -h 127.0.0.1 -P 3306 -u root -t 16 -F 128 -B test -T t1,t2 -o ./var/test
```

上面，我们使用 -B test 表明是对 test 这个 database 操作，然后用 -T t1,t2 表明只导出 t1，t2 两张表。

-t 16 表明使用 16 个线程去导出数据。-F 128 是将实际的 table 切分成多大的 chunk，这里就是 128MB 一个 chunk。

注意：在阿里云一些需要 super privilege 的云上面，mydumper 需要加上 --no-locks 参数，否则会提示没有权限操作。

12.3.2 向 TiDB 导入数据

我们使用 loader 将之前导出的数据导入到 TiDB。Loader 的下载和具体的使用方法见 Loader 使用文档

```
./bin/loader -h 127.0.0.1 -u root -P 4000 -t 4 -q 1 -d ./var/test
```

这里 -q 1 表明每个事务包含多少个 query，默认是 1，在像 TiDB 中导入数据时，推荐用默认值。

导入成功之后，我们可以用 MySQL 官方客户端进入 TiDB，查看:

```
mysql -h127.0.0.1 -P4000 -uroot

mysql> show tables;
+----------------+
| Tables_in_test |
+----------------+
| t1             |
```

```
| t2             |
+----------------+

mysql> select * from t1;
+----+------+
| id | age  |
+----+------+
|  1 |    1 |
|  2 |    2 |
|  3 |    3 |
+----+------+

mysql> select * from t2;
+----+------+
| id | name |
+----+------+
|  1 | a    |
|  2 | b    |
|  3 | c    |
+----+------+
```

12.4 使用 syncer 增量导入数据

上面我们介绍了如何使用 mydumper/myloader 将 MySQL 的数据全量导入到 TiDB，但如果后续 MySQL 的数据有更新，我们仍然希望快速导入，使用全量的方式就不合适了。

TiDB 提供 syncer 工具能方便的将 MySQL 的数据增量的导入到 TiDB 里面。

syncer 也属于 TiDB 工具集，如何获取可以参考下载 TiDB 工具集。

假设我们之前已经使用 mydumper/myloader 导入了 t1 和 t2 两张表的一些数据，现在我们希望这两张表的任何更新，都是实时的同步到 TiDB 上面。

12.4.1 MySQL 开启 binlog

在使用 syncer 之前，我们必须保证：

- MySQL 开启 binlog 功能，参考 Setting the Replication Master Configuration
- Binlog 格式必须使用 row format，这也是 MySQL 5.7 之后推荐的 binlog 格式，可以使用如下语句打开:

  ```
  SET GLOBAL binlog_format = ROW;
  ```

12.4.2 获取同步 position

我们通过 show master status 得到当前 binlog 的 position，syncer 的初始同步位置就是从这个地方开始。

```
show master status;
+------------------+----------+--------------+------------------+-------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+------------------+----------+--------------+------------------+-------------------+
| mysql-bin.000003 |     1280 |              |                  |                   |
+------------------+----------+--------------+------------------+-------------------+
```

我们将 position 相关的信息保存到一个 syncer.meta 文件里面，用于 syncer 的同步:

```
# cat syncer.meta
binlog-name = "mysql-bin.000003"
binlog-pos = 1280
```

注意：syncer.meta 只需要第一次使用的时候配置，后续 syncer 同步新的 binlog 之后会自动将其更新到最新的 position。

12.4.3 启动 syncer

syncer 的配置文件 config.toml:

```
log-level = "info"

server-id = 101

# meta 文件地址
meta = "./syncer.meta"
worker-count = 1
batch = 1

pprof-addr = ":10081"

[from]
host = "127.0.0.1"
user = "root"
password = ""
port = 3306

[to]
host = "127.0.0.1"
user = "root"
password = ""
port = 4000
```

启动 syncer:

```
./bin/syncer -config config.toml

2016/10/27 15:22:01 binlogsyncer.go:226: [info] begin to sync \
binlog from position (mysql-bin.000003, 1280)
2016/10/27 15:22:01 binlogsyncer.go:130: [info] register slave for master server 127.0.0.1:3306
2016/10/27 15:22:01 binlogsyncer.go:552: [info] rotate to (mysql-bin.000003, 1280)
2016/10/27 15:22:01 syncer.go:549: [info] rotate binlog to (mysql-bin.000003, 1280)
```

12.4.4 在 MySQL 插入新的数据

```
INSERT INTO t1 VALUES (4, 4), (5, 5);
```

登录到 TiDB 查看 :

```
mysql -h127.0.0.1 -P4000 -uroot -p
mysql> select * from t1;
+----+------+
| id | age  |
+----+------+
|  1 |    1 |
|  2 |    2 |
|  3 |    3 |
|  4 |    4 |
|  5 |    5 |
+----+------+
```

syncer 每隔 30s 会输出当前的同步统计，如下

```
2016/10/27 15:22:31 syncer.go:668: [info] [syncer]total events = 1, insert = 1, update = 0, \
delete = 0, total tps = 0, recent tps = 0, binlog name = mysql-bin.000003, binlog pos = 1280.
```

```
2016/10/27 15:23:01 syncer.go:668: [info] [syncer]total events = 2, insert = 2, update = 0,  \
delete = 0, total tps = 0, recent tps = 0, binlog name = mysql-bin.000003, binlog pos = 1538.
```

可以看到，使用 syncer，我们就能自动的将 MySQL 的更新同步到 TiDB。

13 TiKV 性能参数调优

本文档用于描述如何根据机器配置情况来调整 TiKV 的参数，使 TiKV 的性能达到最优。

TiKV 最底层使用的是 RocksDB 做为持久化存储，所以 TiKV 的很多性能相关的参数都是与 RocksDB 相关的。

TiKV 使用了 RocksDB 的 Column Falimies 特性，数据最终存储在 RocksDB 内部的 raft、default、lock 和 write 4 个 CF 内。

raft CF 主要存储的是 raft log，与其对应的参数位于［rocksdb.raftcf］项中；default CF 存储的是真正的数据，与其对应的参数位于［rocksdb.defaultcf］项中；write CF 存储的是数据的版本信息（MVCC），与其对应的参数位于［rocksdb.write］项中 lock CF 存储的是锁信息，系统使用默认参数。

每个 CF 都有单独的 block-cache，用于缓存数据块，加速 RocksDB 的读取速度，block-cache 的大小通过参数 block-cache-size 控制，block-cache-size 越大，能够缓存的热点数据越多，对读取操作越有利，同时占用的系统内存也会越多。

每个 CF 有各自的 write-buffer，大小通过 write-buffer-size 控制。

13.1 1. 参数说明

```
[server]
# 通常情况下使用默认值。在复杂的查询比较多的情况下，例如 join 操作，聚合操作等等，
# 可以稍微调大点，但应比系统的 CPU 核数小。
end-point-concurrency = 8

[raftstore]
region-max-size = "80MB"
region-split-size = "64MB"
# 导数据过程中可以将该值设置为64MB，正常运行状态下使用默认值。
region-split-check-diff = "8MB"

[rocksdb]
# 通常情况下使用默认值就可以了，应小于CPU的核数。
max-background-compactions = 6
max-open-files = 40960

[rocksdb.defaultcf]
block-size = "64KB"
compression-per-level = "no:no:no:lz4:lz4:lz4:lz4"
write-buffer-size = "64MB"
max-write-buffer-number = 5
min-write-buffer-number-to-merge = 1
max-bytes-for-level-base = "256MB"
target-file-size-base = "32MB"
# 通常配置为系统内存的30-40%左右。
block-cache-size = "1GB"

[rocksdb.writecf]
compression-per-level = "no:no:no:lz4:lz4:lz4:lz4"
write-buffer-size = "64MB"
max-write-buffer-number = 5
min-write-buffer-number-to-merge = 1
max-bytes-for-level-base = "256MB"
target-file-size-base = "32MB"
```

```
# 通常为 defaultcf.block-cache-size 的 1/n。如果一行数据很大，
# n 通常比较大，如果一行数据比较短，n 比较小。n 通常在 4 到 16 之间。
block-cache-size = "256MB"

[rocksdb.raftcf]
compression-per-level = "no:no:no:lz4:lz4:lz4:lz4"
write-buffer-size = "64MB"
max-write-buffer-number = 5
min-write-buffer-number-to-merge = 1
max-bytes-for-level-base = "256MB"
target-file-size-base = "32MB"
# 通常配置在 256MB 到 2GB 之间，通常情况下使用默认值就可以了，
# 但如果系统资源比较充足可以适当调大点。
block-cache-size = "256MB"

[storage]
# 使用默认值就可以了。
scheduler-concurrency = 102400
# 通常情况下使用默认值就可以了。如果写入操作基本是批量写入的或者写入的行比较大，
# 可以适当调大点。
scheduler-worker-pool-size = 4
```

13.2 2.TiKV 内存使用情况

除了以上列出的 block-cache 以及 write-buffer 会占用系统内存外：

1) 需预留一些内存作为系统的 page cache;

2) TiKV 在处理大的查询的时候 (例如 select * from ...) 会读取数据然后在内存中生成对应的数据结构返回给 TiDB，这个过程中 TiKV 会占用一部分内存。

13.3 3. 导数据推荐配置

block-cache-size 的大小根据机器的内存情况进行调整。

```
[raftstore]
# 该参数的含义是如果一个region的写入超过该值就会检查是否需要分裂，
# 在导数据的情况因为只有insert操作，所以为了减少检查一般配大点，一般为region-split-size的一半。
region-split-check-diff = "32MB"

[rocksdb]
# 该参数主要影响rocksdb compaction的线程数，在导数据的情况下因为有大量的写入，
# 所以应该开大点，但应小于CPU的核数。
max-background-compactions = 6

[rocksdb.defaultcf]
compression-per-level = "no:no:no:lz4:lz4:lz4:lz4"
block-size = "16KB"
write-buffer-size = "64MB"
max-write-buffer-number = 5
min-write-buffer-number-to-merge = 1
max-bytes-for-level-base = "256MB"
target-file-size-base = "32MB"
# 通常配置为系统内存的30-40%左右。
block-cache-size = "1GB"

[rocksdb.writecf]
```

```
compression-per-level = "no:no:no:lz4:lz4:lz4:lz4"
block-size = "16KB"
write-buffer-size = "64MB"
max-write-buffer-number = 5
min-write-buffer-number-to-merge = 1
max-bytes-for-level-base = "256MB"
target-file-size-base = "32MB"
# 通常为 defaultcf.block-cache-size 的 1/n。如果一行数据很大，n 通常比较大，
# 如果一行数据比较短，n 比较小。n 通常在 4 到 16 之间。
block-cache-size = "256MB"

[rocksdb.raftcf]
compression-per-level = "no:no:no:lz4:lz4:lz4:lz4"
block-size = "16KB"
write-buffer-size = "64MB"
max-write-buffer-number = 5
min-write-buffer-number-to-merge = 1
max-bytes-for-level-base = "256MB"
target-file-size-base = "32MB"
# 如果系统内存比较充足，建议配置为2GB。
block-cache-size = "256MB"

[storage]
# 由于导数据过程中每次都是insert一批数据，为了同时能够处理更多的请求，配置为默认配置的10倍。
scheduler-concurrency = 1024000
# 如果CPU核数大于8，建议修改该参数为8
scheduler-worker-pool-size = 4
```

14 TiDB 读取历史版本数据

本文档用于描述 TiDB 如何读取历史版本数据，包括具体的操作流程以及历史数据的保存策略。

14.1 1. 功能说明

TiDB 实现了通过标准 SQL 接口读取历史数据功能，无需特殊的 client 或者 driver。当数据被更新、删除后，依然可以通过 SQL 接口将更新/删除前的数据读取出来。

另外即使在更新数据之后，表结构发生了变化，TiDB 依旧能用旧的表结构将数据读取出来。

14.2 2. 操作流程

为支持读取历史版本数据，引入了一个新的 system variable: tidb_snapshot ，这个变量是 Session 范围有效，可以通过标准的 Set 语句修改其值。其值为文本，记录了时间，格式为："2016-10-08 16:45:26.999"，一般来说可以只写到秒，比如"2016-10-08 16:45:26"。当这个变量被设置时，TiDB 会用这个时间戳建立 Snapshot（没有开销，只是创建数据结构），随后所有的 Select 操作都会在这个 Snapshot 上读取数据。

注意 TiDB 的事务是通过 PD 进行全局授时，所以存储的数据版本也是以 PD 所授时间戳作为版本号。在生成 Snapshot ·时，是以 tidb_snapshot 变量的值作为版本号，如果 TiDB Server 所在机器和 PD Server 所在机器的本地时间相差较大，需要以 PD 的时间为准。

当读取历史版本操作结束后，可以结束当前 Session 或者是通过 Set 语句将 tidb_snapshot 变量的值设为""，即可读取最新版本的数据。

14.3 3. 历史数据保留策略

TiDB 使用 MVCC 管理版本，当更新/删除数据时，不会做真正的数据删除，只会添加一个新版本数据，所以可以保留历史数据。历史数据不会全部保留，超过一定时间的历史数据会被彻底删除，以减小空间占用以及避免历史版本过多引入的性能开销。

我们使用周期性运行的 GC （ Garbage Collection，垃圾回收 ）来进行清理，GC 的触发方式为：每个 TiDB 启动后会在后台运行一个 gc_worker，每个集群中有一个 gc_worker 会被自动选为 leader，leader 负责维护 GC 的状态并向所有的 TiKV region leader 发送 GC 命令。

GC 的运行状态记录记录在 mysql.tidb 系统表中，可通过 SQL 语句进行监测与配置。

```
mysql> select variable_name, variable_value from mysql.tidb;
+-----------------------+----------------------------+
| variable_name         | variable_value             |
+-----------------------+----------------------------+
| bootstrapped          | True                       |
| tikv_gc_leader_uuid   | 55daa0dfc9c0006            |
| tikv_gc_leader_desc   | host:pingcap-pc5 pid:10549 |
| tikv_gc_leader_lease  | 20160927-13:18:28 +0800 CST|
| tikv_gc_run_interval  | 10m0s                      |
| tikv_gc_life_time     | 10m0s                      |
| tikv_gc_last_run_time | 20160927-13:13:28 +0800 CST|
| tikv_gc_safe_point    | 20160927-13:03:28 +0800 CST|
+-----------------------+----------------------------+
7 rows in set (0.00 sec)
```

其中需要重点关注的是 tikv_gc_life_time 和 tikv_gc_safe_point 这两行。

tikv_gc_life_time 用于配置历史版本保留时间（默认值为 10m），用户可以使用 SQL 进行配置。比如我们需要一天内的所有历史版本都可读，那么可以将此行设置为 24h。

tikv_gc_safe_point 记录了当前的 safePoint，用户可以安全地使用大于 safePoint 的时间戳创建 snapshot 读取历史版本。safePoint 在每次 GC 开始运行时自动更新。

需要注意的是，在数据更新频繁的场景下如果将 tikv_gc_life_time 设置得比较大（如数天甚至数月），可能会有一些潜在的问题：

1. 随着版本的不断增多，数据占用的磁盘空间会随之增加。
2. 大量的历史版本会在一定程度上导致查询变慢，主要影响范围查询（ select count(*) from t)。
3. 如果在运行中突然将 tikv_gc_life_time 配置调小，可能会导致大量历史数据被删除，造成 I/O 负担。

14.4 4. 示例

初始化阶段，创建一个表，并插入几行数据

```
mysql> create table t (c int);
Query OK, 0 rows affected (0.01 sec)

mysql> insert into t values (1), (2), (3);
Query OK, 3 rows affected (0.00 sec)
```

查看表中的数据

```
mysql> select * from t;
+------+
| c    |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set (0.00 sec)
```

查看当前时间

```
mysql> select now();
+---------------------+
| now()               |
+---------------------+
| 2016-10-08 16:45:26 |
```

```
+---------------------+
1 row in set (0.00 sec)
```

更新某一行数据

```
mysql> update t set c=22 where c=2;
Query OK, 1 row affected (0.00 sec)
```

确认数据已经被更新

```
mysql> select * from t;
+------+
| c    |
+------+
|    1 |
|   22 |
|    3 |
+------+
3 rows in set (0.00 sec)
```

设置一个特殊的环境变量，这个是一个 session scope 的变量，其意义为读取这个时间之前的最新的一个版本注意这里的时间设置的是 update 语句之前的那个时间

```
mysql> set @@tidb_snapshot="2016-10-08 16:45:26";
Query OK, 0 rows affected (0.00 sec)
```

这里读取到的内容即为 update 之前的内容，也就是历史版本

```
mysql> select * from t;
+------+
| c    |
+------+
|    1 |
|    2 |
|    3 |
+------+
3 rows in set (0.00 sec)
```

清空这个变量后，即可读取最新版本数据

```
mysql> set @@tidb_snapshot="";
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> select * from t;
+------+
| c    |
+------+
|    1 |
|   22 |
|    3 |
+------+
3 rows in set (0.00 sec)
```