TiDB’s Distributed SQL Architecture: For Scale and Reliability
Introduction

Sunny Bains
Architect, PingCAP

- Was MySQL/InnoDB team lead at Oracle.
TiDB’s unique value

- Easy to setup and start
- MySQL 8.0 compatible
- Scalable by Design
- Disaggregated Compute and Storage
- Multi-tenant ready
- Versatile
- Reliable
- Open source
Agenda

01 Design Fundamentals
   TiDB Architecture

02 Resource Control
   Empowering Consolidated Workloads with Precision Resource Allocation. DXF.

03 Online DDL
   Enhancing Database Agility with Lightning-Fast Schema Changes

04 Tools
   TiDB’s wide range of tools for managing your databases
Reference Architecture

- OLTP and optional OLAP
- Raft for consensus
- Data consistency
- Configurable region size
- Fault tolerance, across AZ
TiDB Region

A Region is TiKV’s **logical** scale unit

- Operations such as load balance, scale-out, scale-in are all based on region
- Regions are replicated using the Raft consensus protocol
  - A replicated region is called a Raft group
  - Regions are spread across the nodes in the cluster
  - A single node contains many Regions
  - Regions are stored on RocksDB, there is one instance of RocksDB per Node.
  - Rows in a Region are ordered
Placement Driver \([\text{PD}]\)

PD is the meta-data server for the cluster and coordinates the entire cluster

PD is stateless, stores the global state in etcd
PD’s stateless design allows it to achieve HA using etcd
**PD Overview**

Generates the start and commit Timestamp (TS) of distributed transactions

Handles region distribution and node failures
- Dynamic balancing and rebalancing, spread the love evenly
- Workload balancing, identify and avoid hotspots dynamically

Handles cluster configurations
- Facilitates migration of region replicas to added nodes
- Automatically manages online/offline state transitions of nodes

Multi-Zone deployment and disaster recovery
PD Cluster Monitoring

PD collects information at two levels of granularity

- **Node level**
  - Total and free disk capacity
  - The number of Regions
  - Data writing speed
  - The volume of sent/received Snapshots (used for data replication)
  - Node overload status, CPU monitoring
  - Label information (a set of hierarchical Tags)

- **Region level heartbeat messages (Raft consensus protocol related messages)**
  - The location of the Leader and Followers
  - The number of disconnected Followers
  - Data reading and writing qps
PD Scheduling strategies

PD policies are settable by the administrators

- Replication factor constraint
- Replica placement constraint
  - Policy to force spread of replicas over node/rack/dc/zone
    - e.g., ensure that the replicas are spread geographically
    - A disconnected node rejoins the cluster leading to excessive replicas
    - Ensure raft (region) leaders are spread evenly across the nodes
- Balanced space utilization across the cluster
  - Using the free and used storage on all the nodes
- Hotspot detection and mitigation
  - Using the CPU and read/write throughput metrics sent by the nodes in the cluster
- Governor for scheduling
  - Controls scheduling rate by monitoring ongoing operations, by default it tends to conservative. The speed can be adjusted via the administration interface.
PD Placement Policies

PD placement policies are settable using SQL

Create and set a placement policy

```sql
CREATE PLACEMENT POLICY myplacementpolicy
    PRIMARY_REGION = "us-east-1";
REGIONS = "us-east-1, us-west-1";

CREATE TABLE t1 (a INT) PLACEMENT POLICY = myplacementpolicy;
CREATE TABLE t2 (a INT);
ALTER TABLE t2 PLACEMENT POLICY = myplacementpolicy;
```

Modify a placement policy

```sql
ALTER PLACEMENT POLICY myplacementpolicy FOLLOWERS = 4;
// Create 5 replicas [one leader and 4 followers]
```

Drop a placement policy

```sql
DROP PLACEMENT POLICY myplacementpolicy;
```
TiKV – Distributed storage engine

CNCF Graduated Project. Written in Rust.

TiKV provides the following services
- Store and retrieve the data
- Replication and fault tolerance
- Data distribution across the storage cluster
- Distributed transaction processing

You can visualize it as a large distributed and ordered hash map that is designed for high performance and reliability.
TiKV – Data Storage Example

Example to illustrate how TiKV partitions and manages the data
TiKV - Coprocessor

The TiKV Coprocessor supports the following executors
The names are self explanatory, they can be chained together

- Table scanner
- Index scanner
- Selector (Table scanner | Index scanner)
  - Performs a filter, mostly for where.
- Aggregator (Table scanner | Index Scanner | Selector)
  - Performs an aggregation (e.g. count(*), sum(E))
- Top N elements (Table scanner | Index scanner | Selector)
  - Sorts the data and returns the top N matches, for example, order by C limit 10
TiKV – Raft Consensus Protocol

The core idea of Raft is to elect a leader and all writes then go through the leader.

The data is not considered durable until a majority of the nodes in the cluster acknowledge the write.
TiKV - Handling Failures

Hardware and network failures are a fact of life
Distributed Transactions in TiDB

- TiDB supports Read-Committed and Snapshot Isolation levels
  - The Snapshot Isolation is mapped to MySQL/InnoDB’s Repeatable Read
- TiDB uses an optimized version of the Percolator algorithm for distributed transactions
- A transaction requires a start time stamp and a commit timestamp
  - PD is responsible for handing out these timestamps
  - These timestamps are used in TiDB’s MVCC implementation

- **Async commit in TiDB**
  - The SQL nodes are the Txn Coordinators (TC)
  - The TiKV nodes are the participants
  - Works well when the transaction write set is small and Phase II time dominates

- Supports 1PC Commit Optimization
  - If transaction only updates a non-index column of a record
  - Or, Inserts a record without a secondary index,
  - Only involves a single Region
Optimizer Components

General Overview of the optimizer components
Bringing it all together

A brief look at how the SQL parser and optimizer work in TiDB
TiDB Optimizer

A brief look at how the SQL parser and optimizer work when [optional] TiFlash is installed

```
SELECT AVG(s.price)
FROM prod p, sales s
WHERE p.pid = s.pid AND p.batch_id = 'B1328';
```

IndexScan prod (pid, batch_id = 'B1328')

TableScan sales (price,pid)

Diagram:

- TiKV node 1 (Store 1)
  - Range 1
  - Range 2
  - Range 3
  - Range 4
- TiKV node 2 (Store 2)
  - Range 1
  - Range 2
  - Range 3
- TiKV node 3 (Store 3)
  - Range 2
  - Range 3
- TiKV node 4 (Store 4)
  - Range 1
  - Range 2
  - Range 4
- TiFlash node 1
  - Store 5
- TiFlash node 2
  - Store 6
Agenda

01 Design Fundamentals
   TiDB Architecture

02 Resource Control
   Empowering Consolidated Workloads with Precision Resource Allocation. DXF.

03 Online DDL
   Enhancing Database Agility with Lightning-Fast Schema Changes

04 Tools
   TiDB’s wide range of tools for managing your databases
Why TiDB Resource Control

Flow Control
- Resource quota

Resource segregation
- CPU / IO

Fine-grained resource abstract
- RU / RG

Usage tracking
- Tune / Apportion

Schedule Control
- Job priority
When there are multiple apps/databases

- Cost increase
- Hard to maintain
- Hard to cross database join

Consolidate?

- QoS
- Change -> Disaster
- Interfere

TiDB Resource Control

Yes
A typical microservice architecture: Database per service
What is Resource Control?

Manage multiple workloads in a TiDB cluster. Isolate, manage and schedule access to resources sharing the same TiDB cluster.

- **OLTP workloads** (short queries, small updates…)
- **OLAP workloads** (large batches, ad-hoc queries…)
- **Maintenance jobs** (Backup, Auto tasks…)

eg. xx workloads use too many resources and this impacts the P99 latency of small queries.

eg. Limit the resources allocated to app xxx/ user xxx.

eg. Allocate more resources to higher priority apps/users when the system is overloaded.
What is a Resource Group?

A resource group is a logical container for managing:
- CPU
- I/O

There are 3 important options for each resource group:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RU_PER_SEC</td>
<td>Rate of RU backfilling per second. Must be specified when creating a resource group.</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>The absolute priority of tasks to be processed on TiKV. The default value is MEDIUM.</td>
</tr>
<tr>
<td>BURSTABLE</td>
<td>If the BURSTABLE attribute is set, use the available free system resources even if its quota is exceeded.</td>
</tr>
</tbody>
</table>
Request Unit (RU) and Scheduling

A Request Unit (RU) is an abstract unit for measuring system resource usage. TiDB uses mClock, which is a weight and constraint based scheduler.

“...constraint-based scheduler ensures that [tasks] receive at least their minimum reserved service and no more than the upper limit in a time interval, while the weight-based scheduler allocates the remaining throughput to achieve proportional sharing.”

<table>
<thead>
<tr>
<th>Resource type</th>
<th>RU consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>2 storage read batches, 8 storage read requests and 64 KiB read request payload - consume 1 RU each</td>
</tr>
<tr>
<td>Write</td>
<td>1 storage write batch, 1 storage write request and 1 KiB write request - consume 1 RU each</td>
</tr>
<tr>
<td>SQL CPU</td>
<td>3 ms consumes 1 RU</td>
</tr>
</tbody>
</table>
Evaluate system capacity

- Estimate capacity based on hardware deployment and standard workloads
  
  CALIBRATE RESOURCE;
  +-------+
  | QUOTA |
  +-------+
  | 190470 |
  +-------+
  1 row in set (0.01 sec)

  CALIBRATE RESOURCE WORKLOAD
  OLTP_WRITE_ONLY;
  +-------+
  | QUOTA |
  +-------+
  | 27444 |
  +-------+
  1 row in set (0.01 sec)

  CALIBRATE RESOURCE START_TIME '2023-04-18 08:00:00' DURATION '20m';
  +-------+
  | QUOTA |
  +-------+
  | 27969 |
  +-------+
  1 row in set (0.01 sec)

- Estimate capacity based on actual workloads
  
  CALIBRATE RESOURCE START_TIME '2023-04-18 08:00:00' END_TIME '2023-04-18 08:20:00';
  +-------+
  | QUOTA |
  +-------+
  | 27969 |
  +-------+
  1 row in set (0.01 sec)
Manage resource groups

Create Resource Group
CREATE RESOURCE GROUP IF NOT EXISTS rg1 RU_PER_SEC = 1000 BURSTABLE;

Alter Resource Group
ALTER RESOURCE GROUP rg1 RU_PER_SEC=20000 PRIORITY = HIGH;

Drop Resource Group
DROP RESOURCE GROUP rg1;

Query Resource Group(s)
SHOW CREATE RESOURCE GROUP rg1;
SELECT * FROM information_schema.resource_groups WHERE NAME = 'rg1';
Bind resource groups

**User Level Mapping**
- CREATE USER 'user1'@'%' RESOURCE GROUP rg1;
- ALTER USER 'user1' RESOURCE GROUP rg2;
- SELECT User, User_attributes FROM mysql.user WHERE User = 'user1';

**Session Level Mapping**
- SET RESOURCE GROUP <group name>
- SELECT current_resource_group();

**Statement Level Mapping**
- Hint: /*+ resource_group( ${GROUP_NAME} ) */
  - SELECT /*+ resource_group(rg1) */ * FROM t1
  - INSERT /*+ resource_group(rg2) */ INTO t2 VALUES(2);

Statement (Hint) Level > Session Level > User Level
Resource Control Architecture

Admission Control Layer

- Quota Limits by Request Unit
- GAC
  - Maintain global token buckets
- LAC
  - Measure resources used by TiKV and TiDB (CPU + IO \(\rightarrow\) RU \(\rightarrow\) Tokens), consume tokens allocating by GAC

Scheduling Control Layer

- Enhanced mClock based scheduling
- Weight input
  - RU quota defined in resource groups
  - Priority defined in resource groups
Distributed eXecution Framework (DXF)

Apportion and control resources efficiently at the cluster level, to reduce impact on core business transactions

- Unified scheduling and distributed execution of tasks
- Unified resource management capabilities
- Provides unified capabilities for high scalability, high availability, and high performance
- Typical use cases: DDL, IMPORT, TTL, Analyze, Backup/Restore
  - Where a task processes large amount of data at both schema and table level
  - Executed periodically, but at a low frequency
Agenda

01 Design Fundamentals
   TiDB Architecture

02 Resource Control
   Empowering Consolidated Workloads with Precision Resource Allocation. DXF.

03 Online DDL
   Enhancing Database Agility with Lightning-Fast Schema Changes

04 Tools
   TiDB’s wide range of tools for managing your databases
MySQL solves DDL with MDL

MDL = Meta Data Lock

Table is locked for all sessions during the metadata (DD) update

ADD INDEX example, the metadata change still needs to block!

- MySQL uses a single instance/writer model
- Causes problems with MySQL replication
- Each MySQL replica will asynchronously run the DDL with an MDL
- Also if it’s not an instant DDL, it makes the replication lag worse
Is a distributed database different?

Client connections see and act on the same data

Issues to solve (ADD INDEX as an example):

- No synchronous update of metadata/schemas for all cluster nodes
- Need to create index entries for all existing rows in the table
- Need to update entries for concurrent user changes
The Solution

Version all schemas.

Allow sessions to use current or the previous schema version

Use sub-state transitions:

● So that version N-1 is compatible with version N

Create states that will allow the full transition:

● From state ‘None/Start’ to state ‘Public’
<table>
<thead>
<tr>
<th>Public (vN)</th>
<th>(vN-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>YES</td>
</tr>
<tr>
<td>INSERT</td>
<td>YES</td>
</tr>
<tr>
<td>UPDATE</td>
<td>YES</td>
</tr>
<tr>
<td>DELETE</td>
<td>YES</td>
</tr>
<tr>
<td>Public (vN)</td>
<td>(vN-1)</td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
</tr>
<tr>
<td>SELECT</td>
<td>YES</td>
</tr>
<tr>
<td>INSERT</td>
<td>YES</td>
</tr>
<tr>
<td>UPDATE</td>
<td>YES</td>
</tr>
<tr>
<td>DELETE</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>Public (vN+1)</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>SELECT</td>
<td>YES</td>
</tr>
<tr>
<td>INSERT</td>
<td>YES</td>
</tr>
<tr>
<td>UPDATE</td>
<td>YES</td>
</tr>
<tr>
<td>DELETE</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>Public ((vN+1))</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>SELECT</td>
<td>YES</td>
</tr>
<tr>
<td>INSERT</td>
<td>YES</td>
</tr>
<tr>
<td>UPDATE</td>
<td>YES</td>
</tr>
<tr>
<td>DELETE</td>
<td>YES</td>
</tr>
<tr>
<td>Public (vN+1)</td>
<td>Write Reorg (vN)</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>SELECT</td>
<td>YES</td>
</tr>
<tr>
<td>INSERT</td>
<td>YES</td>
</tr>
<tr>
<td>UPDATE</td>
<td>YES</td>
</tr>
<tr>
<td>DELETE</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>Public (vN+2)</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------</td>
</tr>
<tr>
<td>SELECT</td>
<td>YES</td>
</tr>
<tr>
<td>INSERT</td>
<td>YES</td>
</tr>
<tr>
<td>UPDATE</td>
<td>YES</td>
</tr>
<tr>
<td>DELETE</td>
<td>YES</td>
</tr>
</tbody>
</table>
Index backfill

t0: Session in Write Only:
   Insert (46, ‘V’)

Table (Public)

<table>
<thead>
<tr>
<th>2</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>W</td>
</tr>
<tr>
<td>15</td>
<td>K</td>
</tr>
<tr>
<td>46</td>
<td>V</td>
</tr>
</tbody>
</table>

New Index (Write Only)

<table>
<thead>
<tr>
<th>A</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>46</td>
</tr>
</tbody>
</table>
Index backfill

Table (Public)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>A</td>
</tr>
<tr>
<td>8</td>
<td>W</td>
</tr>
<tr>
<td>15</td>
<td>K</td>
</tr>
<tr>
<td>46</td>
<td>R</td>
</tr>
</tbody>
</table>

New Index (Write Only)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>46</td>
</tr>
</tbody>
</table>

Update, since table is ‘Public’

t0: Session in Write Only:
Insert (46, ‘V’)

t1: Session before Write Only:
UPDATE (46, ‘R’)

Index backfill

t0: Session in Write Only:  
Insert (46, ‘V’)

t1: Session before Write Only:  
UPDATE (46, ‘R’)

DELETE Only, cannot INSERT, due to risk of orphan index entry.
<table>
<thead>
<tr>
<th></th>
<th>Public (vN+2)</th>
<th>Write Reorg (vN+1)</th>
<th>Write Only (vN)</th>
<th>(vN-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>INSERT</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>UPDATE</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES*</td>
</tr>
<tr>
<td>DELETE</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>?</td>
</tr>
<tr>
<td></td>
<td>Public (vN+2)</td>
<td>Write Reorg (vN+1)</td>
<td>Write Only (vN)</td>
<td>Delete Only (vN-1)</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------</td>
<td>---------------------</td>
<td>-----------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>SELECT</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>INSERT</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>UPDATE</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES*</td>
</tr>
<tr>
<td>DELETE</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>Public (vN+3)</td>
<td>Write Reorg (vN+2)</td>
<td>Write Only (vN+1)</td>
<td>Delete Only (vN)</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------</td>
<td>--------------------</td>
<td>-------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>SELECT</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>INSERT</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>UPDATE</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES*</td>
</tr>
<tr>
<td>DELETE</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>
Other DDL Optimizations

RocksDB can ingest pre-generated SST files

We use pre-generated files for backfilling

- Generate SST files and ingest them into TiKV/RocksDB
- No need to write to the new index in TiKV
- Negligible impact on concurrent load
- Efficient use of network, CPU and IO

Use optimized Co-processor framework for reads

- Direct KV transactional reads are expensive
- Co-processor works on local data, avoids network overhead
# ADD INDEX Timings

10 TiDB and 15 TiKV Nodes

<table>
<thead>
<tr>
<th>Component</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>TiDB</td>
<td>16 vCPU 32 GiB RAM - c6g.4 x large</td>
</tr>
<tr>
<td>PD</td>
<td>8 vCPU 16 GiB RAM - c6g.2 x large</td>
</tr>
<tr>
<td>TiKV</td>
<td>16 vCPU 64 GiB RAM 6T Disks - m6g.4 x large</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test</th>
<th>One-column Key Index</th>
<th>Ten-columns Key Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>10TB Table with Global Sort</td>
<td>47m</td>
<td>1 hour 6 min</td>
</tr>
</tbody>
</table>
Agenda

01 Design Fundamentals
   TiDB Architecture

02 Resource Control
   Empowering Consolidated Workloads with Precision Resource Allocation

03 Online DDL
   Enhancing Database Agility with Lightning-Fast Schema Changes

04 Tools
   TiDB’s wide range of tools for managing your databases
Links

TiDB SQL Parser and Optimizer

TiKV / Placement Driver

TiFlash Column Store

OSSInsight

TiUP

Join our Slack Channel

Chaos Mesh
THANK YOU.