CEDAR: A Distributed Database for Scalable OLTP

Weining Qian
wnqian@dase.ecnu.edu.cn
The Internet economy

- Content/traffic => money
- O2O: Online => Offline, or Offline => Online
- O2O of mission-critical apps: 互联网+ (Internet+)
- OLTP is inevitable
**Phenomenal applications**

- Phenomenal - very remarkable; extraordinary.

- 180,000 tps in year 2016
Phenomenal is common

- 12306 during Spring Festival
- Black Friday promotion/second kill, ...
- The pressure is on backend (transaction | payment) systems
- Be inevitable and day-to-day more common

- All pressure will finally go to mission critical systems
- Essentially a High-throughput, scalable transaction processing problem
A brief history of DBMS

1960s
- **Flat File Based**
  - Mainframe-based file management systems were used for mostly transactional data processing; storage was tape-based
  - Batch reporting of data for managers was provided through report generators

1970s
- **Navigational DBMS**
  - ISVs marketed DBMS systems and modules for database management, report writing, and querying
  - Tape storage was common but disk-based direct storage started becoming popular
  - Databases were mostly based on a navigational model; in mid-late 1970’s relational DBMS (RDBMS) solutions emerged

1980s
- **Relational DBMS**
  - Major commercialization of RDBMS solutions took place
  - Extended application-specific relational models (e.g., engineering) were invented
  - RDBMS solutions were matured through various optimization techniques
  - Personal databases for the PC flourished

1990s
- **Distributed DBMS**
  - Shared everything/SMP architectures emerged as techniques to increase database performance
  - Distributed architectures (such as clustering) became common as distributed computing became widely adopted

2000s
- **Post Relational DBMS**
  - Unstructured data management solutions with non-relational data models emerged
  - Specialized databases and appliances start emerging (in-memory, column-oriented)
  - Very large scale distributed data processing platforms emerged

- **Parallelization** of database workloads is developed
- Distributing computing becomes widespread
- Database machines – combining proprietary software and hardware – are launched
- SSD and flash disk drive technology is perfected

[https://maxkanaskar.files.wordpress.com/2014/04/database-platform-history.png](https://maxkanaskar.files.wordpress.com/2014/04/database-platform-history.png)
One size fits all => One size fits none!

2015 - One Size Fits None!!

- Data Warehouse market
- OLTP market
- NoSQL market
- Complex Analytics
- Streaming market
- Graph analytics market
DBMS

File Systems + Data model abstraction = DBMS

Data processing abstraction
NoSQL

Distributed FS

Data model abstraction

Weak consistency

= NoSQL
NewSQL

In-memory computing + Relational model + Transaction processing = NewSQL

Fast networking
# OldSQL vs. NoSQL vs. NewSQL

<table>
<thead>
<tr>
<th></th>
<th>OldSQL</th>
<th>NoSQL</th>
<th>NewSQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data model</td>
<td>Relational</td>
<td>---</td>
<td>Relational</td>
</tr>
<tr>
<td>Interface</td>
<td>SQL</td>
<td>Variance</td>
<td>SQL</td>
</tr>
<tr>
<td>Consistency/Concurrency control</td>
<td>Strong</td>
<td>Weak</td>
<td>Strong</td>
</tr>
<tr>
<td>Fault tolerance</td>
<td>Strong</td>
<td>Fine</td>
<td>Strong</td>
</tr>
<tr>
<td>Performance</td>
<td>Poor</td>
<td>Good</td>
<td>Very good</td>
</tr>
<tr>
<td>Scalability</td>
<td>Poor</td>
<td>Good</td>
<td>Fine</td>
</tr>
</tbody>
</table>
How to scale a database system?

- **Scaling-up**
  - More disks.
  - More RAM.
  - Faster CPUs.
  - Use SSDs.

- **Sharding/partitioning**
  - Logical Partitions

- **Replication**
  - Read Request
  - Update Cache

- **Caching**
  - Query Request

The open-source OceanBase (0.4)
OceanBase 0.4 is not enough

- Simple transactions only
- Weak availability support
- Single-point transaction
- More optimization needed for query/storage
- Interface adaptibility

Not enough for mission critical apps in banks with strong-consistency, high-availability, and high-throughput complex transaction processing requirements
Features we need

- Complex transactions
- High-performance:
  - High-throughput
  - Low latency
- High-availability
- Scalability
- Elasticity
Overview (CEDAR core)
Design choices

- Separating read and write operations on different nodes
- Scalable reading on multiple nodes
- Writing to memory in one node only
  - No expensive distributed concurrent control or synchronization
- “Deep” optimization for transactions
  - To optimize data transmissions, query execution plans, and executions
- High-availability is guaranteed by log synchronization
Status = Baseline + Delta

<table>
<thead>
<tr>
<th>S-Node</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>price</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>2.0</td>
</tr>
<tr>
<td>3</td>
<td>3.0</td>
</tr>
<tr>
<td>4</td>
<td>4.0</td>
</tr>
<tr>
<td>5</td>
<td>5.0</td>
</tr>
<tr>
<td>6</td>
<td>6.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T-Node</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S-Node</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>price</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>2.0</td>
</tr>
<tr>
<td>3</td>
<td>3.0</td>
</tr>
<tr>
<td>4</td>
<td>4.0</td>
</tr>
<tr>
<td>5</td>
<td>5.0</td>
</tr>
<tr>
<td>6</td>
<td>6.0</td>
</tr>
</tbody>
</table>
Reading

- All readings need to access S-Node as well as T-Node
Writings

- All writings need to access S-Node as well as T-Node

Extra read:
1. sending a write request
2. reading static data
3. responding to S-Node
4. sending an execute plan and expression

In-memory
5. writing delta data
6. flushing log
7. returning ok to S-Node
8. responding to users
Transaction management

- Transactions are only processed on the T-Node
Pros and cons

- **Pros**
  - Massive storage
  - Scalable read
  - Efficient transaction management

- **Cons**
  - Expensive data transmission
Performance is affected by

- The lengths of locks affect
  - Degrees of parallelization
  - Latency
- Capability of S-Nodes
  - Throughput of readings
- Capability of the T-Node
  - Throughput of writings

- Most cost is for short/simple transactions
- They are easier to be scheduled
S-Node optimizations

Static data caching

Parallel readings

1. Sending a subquery plan
2. Reading delta data
T-Node

- Storage node?
  - All computation resources are for TP
  - High communication cost

- Computation node?
  - Low communication cost
  - How much work should it do?

- Balancing T-Node’s computation for queries and transaction processing
**Transaction compilation**

```plaintext
Procedure Order(p_itemType int, p_custId int, p_orderAmount int)

declare v_price, v_value double;
declare v_itemId, v_stock, v_orderAmount int;
s:select itemId, price, stock
    into v_itemId, v_price, v_stock
    from item where itemType = p_itemType
    order by price desc limit 1;
if( v_stock > p_orderAmount )
    update item set stock = stock - p_orderAmount
        where itemId = v_itemId;
    v_orderAmount = p_orderAmount;
else
    update item set stock = stock - v_stock
        where itemId = v_itemId;
    v_orderAmount = v_stock;
end
v_value = v_price * v_orderAmount;
update customer set balance -= v_value
    where custId = p_custId;
```

**Execution plan**

- **RD**: read
- **UP**: update
- **RS**: read static data

**Dependency graph**

- Control dep.
- Data dep.
T-Node optimization

Re-order T-Node ops

- S\( \xrightarrow{} \) ReadStatic(A)
- T\( \xrightarrow{} \) ReadDelta(A)
- S\( \xrightarrow{} \) ReadStatic(B)
- S\( \xrightarrow{} \) ReadStatic(A)
- T\( \xrightarrow{} \) RWDelta(A)

Postpone conflict ops

- T\( \xrightarrow{} \) RWDelta(A)
- T\( \xrightarrow{} \) RWDelta(B)
- T\( \xrightarrow{} \) RWDelta(C)
- T\( \xrightarrow{} \) RWDelta(D)
TPC-C, Smallbank, TATP Benchmarks

- Better than PG
- Better than VoltDB under complex workload
- With significant advantage when data cannot be naturally partitioned
Indexing

- Index is organized as a table.
- No distributed transactions.
- Taking advantage of the load balancing, availability of system.
Indexing overview

- **Initialization**: Preparing for the start of index construction.
- **Bulk Loading**
  - Local processing: collecting statistical information
  - Global processing: achieving load balancing based on an equi-depth histogram
- **Termination**: Scheduling the task for replication of the index for high availability.
Experimental results

- **Data Size**
  - Time (seconds) for different data sizes (10GB, 15GB, 20GB, 25GB)
  - Comparison between Read-Insert and Bulk Loading

- **Data Distribution**
  - Data % for different data distributions (uniform, gaussian, zipflaw)
  - Comparison between SERVER1, SERVER2, SERVER3, and SERVER4

- **Throughput**
  - Throughput (Requests/second) for different numbers of clients (100, 125, 150, 175, 200, 225, 250)
  - Comparison between uniform, zipflaw, gaussian, and NoIndex

- **Latency/ms**
  - Latency (milliseconds) for different numbers of clients (100, 125, 150, 175, 200, 225, 250)
  - Comparison between uniform and zipflaw
Other works on the CEDAR

- Scalable range optimistic concurrency control
- Global snapshot isolation with Paxos replication
- Scalable commit log recording/synchronization
- Data transmission optimization
- Distributed statistics monitoring
- Rule-based and cost-based query optimization
Release

Homepage: https://github.com/daseECNU/CEDAR
Email: cedar.tp@gmail.com
CEDAR是华东师范大学数据科学与工程学院（简称“DaSE”）基于 OceanBase 0.4.2 研发的可扩展的关系数据库。2016年2月1日，CEDAR项目组完成了CEDAR 0.1 版本的开发与测试，2016年9月26日，CEDAR 0.2 版本发布。

版本特性

CEDAR在OceanBase 0.4.2 的基础上新增了如下11个功能模块：

CEDAR 0.1 版本新增的功能有：

- 高可用的三集群架构（集群间选主、集群角色自动切换、日志强同步及恢复等机制）
- 多线程网络IO处理框架Libonev
- 游标
- 存储过程
- 二级索引
- 非主键多行更新
- 半连接

CEDAR 0.2 版本新增的功能有：

- SNAPSHOT ISOLATION 隔离级别
- 表锁
- 基于布隆过滤器的连接
- 日志同步优化
TPC-C, Smallbank, TATP Benchmarks

- Better than PG
- Better than VoltDB under complex workload
- With significant advantage when data cannot be naturally partitioned
Application

- One of the largest banks in China
  - To use local/open-source DBMS for transactional applications (to replace DB2)
  - Three stages: historical DB => Hybrid DB => transactional DB
  - 2013 – present (in stage 2)
  - Code name: CBase within the bank
Stage 1 => Stage 2

- **2016**
  - Transactional workload
  - Multi-clusters

- **2014**
  - Complex query
  - 2-clusters

- **2013**
  - Simple query
  - Single cluster

**Banknote serial number tracing**

**Historical DB**

**Supplier-chain app.**
Banknote serial number tracing

- All records of banknote serial numbers are stored for 90 days in the CBase
- More than 10TB data totally
- For counterfeit money detection etc.
Status

- Went online in 2013-12
- Single cluster with 9 servers

Features
- Batch load with about 100GB data/day
- Latency in several milliseconds over TBs of data
- Linear scalability with respect to number of servers
Move historical data to historical DB so that the online system is lightweighted

283 kinds of query services are supported by CBase
Status

- Went online in 2014-09
- 2-clusters, with 21 servers each
- Features
  - About 400GB data/day
  - Latency in several milliseconds for most queries
  - Active/active high-availability
- Has become the single-point sign-in gateway of nearly all business logic
Supplier-chain applications

- Hybrid workload
  - Complex transactions with tens or even hundreds of SQL operations
  - Complex analytical queries with many joins of large tables
  - High-availability requirements
Status

- Went online in 2017-03
- Multi-clusters, each with 12 servers

Features

- Scale-out like most NoSQL systems
- 5000tps for complex workload
- 5ms latency for key-search
- Complex queries are answered within 3s
- Multi-clusters are synchronized with Paxos-like protocol to provide high-availability
More applications this year

- Network Alliance (网联) e-payment clearinghouse
  - 2017-03 to 2017-04
- Loaning
  - 2017-09 to 2017-12
POC: O2O task assignment/taking
A bigger picture
A bigger picture

CEDAR

Ginkgo
CEDAR: 雪松

- **C**: Cluster-oriented
- **E**: for Enterprise applications
- **D**: scalable Dbms
- **AR**: non-traditional ARchitecture
Summary

- **SQL** support: ODBC/API interfaces
- **Transaction** support: ad-hoc transactions and store-procedures
- **Efficient** query optimization/execution: various distributed join implementation + indexing schemes
- Deployment with **High-Availability** support
- **Highly Scalable:**
  - Read/write separation
  - Hot/cold separation
- **Management/maintenance-friendly:** Import/export toolkit and monitoring/diagnose toolkit
Full-fledged DBMS
  - with SQL and Transaction Processing support

Scable architecture
  - cluster-oriented: commodity PC server with large memory, SSD drive, and high-speed network

Mission-critical-app.-oriented
  - apps in enterprises, banks, communications, etc.

GPLv2
Thanks!

Homepage: [https://github.com/daseECNU/CEDAR](https://github.com/daseECNU/CEDAR)
Email: cedar.tp@gmail.com

2010 OceanBase project started

2012.11 v0.4 with SQL (limited transaction support)

2015.3 v0.4.2 open source (GPLv2)

2015.3 v0.4.2 open source (GPLv2)

2016.1 CEDAR 0.1 HA support

2016.9 CEDAR 0.2 new Transaction Engine

2017.8 CEDAR 0.3 new Query Engine

v0.5 with HA

v1 for Aliyun (cloud)
Acknowledgement
Thanks!