Salt: Combining ACID and BASE in a Distributed Database

Authors (University of Texas Austin): Chao Xie, Chunzhi Su, Manos Kapritsos, Yang Wang, Navid Yaghmazadeh, Lorenzo Alvisi, Prince Mahajan

Presented By:
Sharanya Bathey
The tale of ACID...

• **Transaction** - Basic unit of work in an RDBMS

• **ACID**
  – Atomicity: all or nothing
  – Consistency: always leave a database in a consistent state
  – Isolation: every transaction is completely isolated
  – Durability: once committed always recorded
The tale of ACID...(Oh No!)

- What needs to be done to achieve this abstraction?
  - 2 Phase Commit
  - Locks

- What baggage do such abstractions come with?
  - Performance is sacrificed

Image Source
Rise of BASE...

- BASE
  - Basically Available - does guarantee availability, in terms of CAP theorem
  - Soft State - State of the system may change over time
  - Eventually Consistent - data will be consistent at replicas eventually
Rise of BASE... (What now?)

• Problem?
  – Complexity of programming
  – Consistency Management

Image Source
Is there no middle ground?

• Pareto Principle - 80-20 Rule

• Example:
  – Fusion Ticket (open source ticketing application)
  – Total of 11 different types of Transactions of which just 2 are performance critical
Introducing... BASE transactions

Manually Identify the key transactions that cause the performance bottleneck

Base-ify these transactions by breaking the transaction into alkaline sub transactions which are still contained within the same BASE transaction
Example of Base-ifying an ACID transaction
But the problem being?

Acid Transaction

Balance

- Read c
- Read s

Transfer 1

- Is \( c \geq 10 \)?
- \( c = c - 10 \)
- \( s = s + 10 \)

Transfer 2

- Is \( c \geq 10 \)?
- \( c = c - 10 \)
- \( s = s + 10 \)

Base-fied Transactions
# Solution being? Restrict Interaction

<table>
<thead>
<tr>
<th></th>
<th>ACID</th>
<th>BASE</th>
<th>Alkaline</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACID</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>BASE</td>
<td>No</td>
<td>Yes*</td>
<td>Yes*</td>
</tr>
<tr>
<td>Alkaline</td>
<td>No</td>
<td>Yes*</td>
<td>Yes**</td>
</tr>
</tbody>
</table>

* - intermediate committed state of alkaline sub transactions
** - only read alkaline sub transactions
Base Transactions (Contd...) 

• Major life events of BASE transactions: Accept & Commit

• How does a BASE transaction provide ACID abstraction?
  – Atomicity - once accepted will eventually commit
  – Consistency - Interactions keep ACID and BASE transactions isolated
  – Isolation - **SALT Isolation** coming up next..
  – Durability - Accepted BASE transactions are guaranteed to be durable through logging
Introducing the MVP – Salt Isolation

• Uses Locks - Isolation level chosen is Read Committed

• Locks:
  - ACID Locks - write with any other operation conflicts
  - Alkaline Locks - alkaline sub transactions isolated from ACID and other alkaline sub transactions
  - Saline Locks - ACID isolated from BASE, but increased concurrency by exposing intermediate state to of BASE transactions
So does it actually work?

• Implementation:
  – Modified MySQL to allow BASE transactions and Alkaline Sub-transactions
  – Allow alkaline and saline locks

• Optimizations:
  – Early commit - A client that issues a BASE transaction sees the transaction completion when its first Alkaline sub transaction commits
  – Failure recovery - Logging with redo and roll forward
  – Transitive dependencies - Per object queue
  – Local Transactions

• Replication of data was done across 10 partitions and each partition was three-way replicated

• Evaluated: Performance Gain, Programming Effort, Contention Ratio
Case Study

begin BASE transaction

  Check whether all items exist. Exit otherwise.
  Select \texttt{w\_tax} into @\texttt{w\_tax} from warehouse where \texttt{w\_id} = : \texttt{w\_id};

begin alkaline—subtransaction

  Select \texttt{d\_tax} into @\texttt{d\_tax}, \texttt{next\_order\_id} into @\texttt{o\_id} from district where \texttt{w\_id} = : \texttt{w\_id} and \texttt{d\_id} = : \texttt{d\_id};

  Update district set \texttt{next\_order\_id} = \texttt{o\_id} + 1 where \texttt{w\_id} = : \texttt{w\_id} AND \texttt{d\_id} = : \texttt{d\_id};

end alkaline—subtransaction

Select \texttt{discount} into @\texttt{discount}, \texttt{last\_name} into @\texttt{name}, \texttt{credit} into @\texttt{credit} where \texttt{w\_id} = : \texttt{w\_id} and \texttt{d\_id} = : \texttt{d\_id} and \texttt{c\_id} = : \texttt{c\_id}

Insert into orders values (: \texttt{w\_id}, : \texttt{d\_id}, @\texttt{o\_id}, ...);

Insert into new\_orders values (: \texttt{w\_id}, : \texttt{d\_id}, \texttt{o\_id});

For each ordered item, insert an order line, update stock level, and calculate order total

end BASE transaction
Applications used for the experiments

• TPC-C benchmark: database performance standard
  – 5 transactions (new-order, payment, stock-level, order-status and delivery)
  – New-order and payment are responsible for 43.5% of total number of transactions

• Fusion Ticket: open source ticketing application in PHP and MySQL
  – Includes several transactions
  – Create-order and payment most frequent and performance critical

• Micro benchmark:
  – Contention ratio
  – Contending transaction position
  – Read-Write ratio
Did Salt win the award? (Yes...)

Base-ify: new-order and payment

Base-ify: Create-order and payment

TPC-C

Fusion Ticket

ACID

Salt

Latency (ms)

Throughput (transactions/sec)

6.6x higher

6.5x higher
Did Salt win the award? (Contd...)

Performance gain becomes stagnant
Did Salt win the award? (Contd...)

![Graphs showing throughput vs. contention ratio and write ratio]
Discussion & Q-A

• Pros:
  – Performance gain is drastic
  – Control in the hands of the programmer

• Cons:
  – Cassandra allows batching of commands into groups to form transactions called atomic batches
  – Leverage the power of MySQL clusters - MySQL does not allow nested transactions and MySQL now has several NoSQL/NewSQL like features
  – Programming complexity - the difficulty of identifying the transactions that need to be converted and verifying the correctness is a daunting task
Thank You!