The End of an Architectural Era

By
Arpit Gagneja
Organization of Slides

• Summary
• Critique
• Competition (Related Works)
• Future
Starting with Main idea of the paper

• “One size fits all” no longer valid as RDBMS paradigm.
• Why?

Specialized architectures beat old system by order of magnitude. For example,

- Text (Specialized engines from Google, Yahoo, etc.)
- Data Warehouses (Vertica, Monet)
- Stream Processing (StreamBase and Coral8)
- Scientific and intelligent databases (MATLAB)
Current situation of RDBMS

• So, RDBMS are left out with the OLTP market only.

• This paper gives convincing results that even in OLTP market, it can be replaced by more powerful architecture (H-Store) which utilizes the modern hardware/processor to full extent.
Coming to main point

• RDBMS is now 25 years old legacy code.

• We need to design something new on a blank paper keeping in mind today’s requirement.

• No longer need to work for architectures designed for old times.
What I mean by old design?

• “Current RDBMSs were architecture for the business data processing market in a time of different user interfaces and different hardware characteristics”. (http://nms.csail.mit.edu/~stavros/pubs/hstore.pdf)

• They all follow 25 years old System R features:
  1. Disk Storage
  2. Multithreading
  3. Locking
  4. Log-based recovery
Changed ecosystem

• 1970’s: disk

• Now: main memory

Large machine in 1970’s: Megabyte of a main memory
Today: Large machines approaching 100 Gbytes (Terabyte not far!)
Grid of 20 nodes with each 32 GB is capable of holding main memory deployment.
New OLTP Design

• Main Memory (As Discussed)

• Multithreading: Today’s transaction are very fast. Consider a click to “buy” on Amazon website.

  The fast transaction execution time necessitates one thing: “Run SQL commands to completion with a single-threaded execution model.”

  (http://nms.csail.mit.edu/~stavros/pubs/hstore.pdf)

  (Continuing on next slide)
No Multi-threading

• No multi-threading simplifies DBMS to a great extent.

1. No concurrent B-Tree structure required.

2. Eliminate overhead of handles, buffers, threads and everything associated with multi-threading world.

This will result in high performance as overhead of multithreading is removed.
Grid Computing

- Only commodity hardware required. So, this is cheaper for sure. Take an example from Google!

- Partition data horizontally. Horizontal Partitioning is also known as Sharding.
  (We will see Trees related with this thing later in the slides when we will discuss H-Store)

- Easy to add and remove nodes.
High Availability (HA)

- In 1970’s when an existing machine went down, the companies like IBM get themselves engaged in setting up the new hardware from the logging tapes.

- Nowadays, the concept of a hot standby.

- But, how in memory database can do this thing better?
HA in in-memory databases

• Keep multiple replicas consistent.
• Concept of shared nothing.
• Use multiple machines in a peer-to-peer configuration. (Inter-machine replication aids in Fault Tolerance)

Note: Logging is still required in case if transaction fails. But, logs are stored in main memory and they are no longer required if transaction commits.
Main Overheads Eliminated

• So, we are moving towards very efficient design for facilitating OLTP market.

• The overheads eliminated are:

1. Disk
2. No Multi-threading
3. No more hot stand-by required

We will now discuss the design of H-Store which exactly implements these design considerations.
Prerequisites to understand H-Store

- The thing you will hear again and again is Transaction Classes.


<table>
<thead>
<tr>
<th>Transaction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>new_order</code></td>
<td>Place an order for a customer. 90% of all orders can be supplied in full by stocks from the customer's “home” warehouse; 10% need to access stock belonging to a remote warehouse. Read/write transaction. No minimum percentage of mix required, but about 50% of transactions are <code>new_order</code> transactions.</td>
</tr>
<tr>
<td><code>payment</code></td>
<td>Updates the customer’s balance and warehouse/district sales fields. 85% of updates go to customer’s home warehouse; 15% to a remote warehouse. Read/write transaction. Must be at least 43% of transaction mix.</td>
</tr>
<tr>
<td><code>order_status</code></td>
<td>Queries the status of a customer’s last order. Read only. Must be at least 4% of transaction mix.</td>
</tr>
<tr>
<td><code>delivery</code></td>
<td>Select a warehouse, and for each of 10 districts &quot;deliver&quot; an order, which means removing a record from the new-order table and updating the customer’s account balance. Each delivery can be a separate transaction; Must be at least 4% of transaction mix.</td>
</tr>
<tr>
<td><code>stock_level</code></td>
<td>Finds items with a stock level below a threshold; read only, must read committed data but does not need serializability. Must be at least 4% of transaction mix.</td>
</tr>
</tbody>
</table>
Tree Schema in H-Store

Horizontal partitioning of tables in different nodes

Node 1
- Alice
- Alice’s Order
- Alice New-Order and Order-Line

Node 2
- Bob
- Bob’s Order
- Bob New-Order and Orders-Line
Constrained Tree Applications

• Partition root table across multiple nodes based on some ranges on primary key.
  For Example, Customers having name starting with ‘A’, ‘B’, ‘C’ are present in Node 1. Customers having name starting with ‘D’, ‘E’, ‘F’ are present in other Node say Node 2 and so on.

• Similarly partition other tables such that other information related to Customer in specific node is locally available. Like in our case, orders of Customers having names starting with ‘A’, ‘B’, ‘C’ in node 1.

• CTA helps H-Store in making Transaction classes single-site.
What if data is not perfectly in Tree?
Ref:
Vertical partitioning and replication comes to our rescue!

• Item and Order-Line is making the schema non-tree structured.

• But, item table is read only, so it can be replicated at each node.
• Order-Line table can be partitioned based on order at each site or node.

• So, with the help of partitioning and replication, we can make it a Tree structured schema.
Single Site

• Replication of read only tables at each site make transaction single-sited.

• This design consideration makes the transaction execution time really small.

• In the previous slide, replication of item table at all sites help in making transaction class single-site.
One Shot

• Main property of One Shot applications: Transactions at multiple sites executed parallel without intermediate results being communicated.

• This is normally done with vertical partitioning.

• For example, order-line is vertically partitioned according to specific order in TPC-C schema.
Two-phase transactions

- To eliminate the need of undo log, there is a concept of two-phase transaction.
  
  First phase: Read-only operations at various sites.
  
  Second phase: Execute updatable transaction so that there is no possibility of integrity constraint violation.

- Strong two-phase transactions: If at phase 1, all sites produce same results regarding abort or commit of the transaction.
Sterile Transactions

• This is one of the important goal H-Store seeks to achieve in its design.

• Sterility means order in which transactions execute at different sites doesn’t matter. They will end up in same result.

• The main outcome out of sterile transaction is that we don’t need to take care of concurrent control.
What about High Availability?

Till now, we have discussed design considerations for almost everything, what is left behind is how H-Store ensures HA?

• Timestamp ordering (by NTP)
• Some small intended delay so as to properly order transaction.

The ordering and delay is mandatory to keep replicas consistent.
In nutshell!

- H-Store achieves high performance by following things:

1) Utilizing main memory
2) Making transaction single site and single shot (Eliminates Multithreading)
3) Replication across sites (This ensures high availability)
NewSQL

• H-Store is branded as “NewSQL”.

• NewSQL systems achieves the high performance of NoSQL systems and at the same time ACID compliant like traditional databases.

• VoltDB is commercial version of H-Store.
Limitations

• H-Store assumes that transactions are short.

• Till now, it is all manual to identify applications which are single-sited, two phase and one shot.

• Popularity and Community Support is slowly growing as compared to NoSQL.
Competitors

• Google Spanner
• SAP Hana
• MemSQL
• NuoDB
• Trafodion

Ref: https://en.wikipedia.org/wiki/NewSQL
Future Work

• Automate the process of identifying the applications which are single site, two-phase and one shot.

• How multi-core can be utilized better?

• Integration with data warehousing tools.
Impact on Current Research

• “One size fits all” is no longer a favourite industry paradigm.

• It is good to design system based on current needs.

• Relational model as an OLTP solution is obsolete.

• Researchers are diving deep into unstructured data. Today, we have more unstructured data as compared to structure data.

• There are options like Column oriented databases, RDF which can scale well.
Thank you!