Why multi-node DBMSs?

Reliability (high availability)
Performance/scale
Geographic distribution

3 big use cases (OLTP, Warehouse, geographic distribution / silo integration)

OLTP

hedge fund example (Getco)
Cell phone billing

"Through the OLTP looking Glass" SIGMOD 2008

Shore – minus the process switch (similar to Postgres, …)

Buffer mgmt	34%
Latching	14% (B-trees) [latch crabbing, system catalogs]
Locking	16%
logging	12%
B-tree	16%
Other Useful work	7%

Observations:

Better B-tree (say 1/3 better) – improves performance by 5%. Not very interesting
To go a lot faster – have to several of these PLUS deal with app-to-DB communication

Design goals for H-Store (VoltDB)

Multi-node (scalability)
High availability (no down time)
X10 Postgres performance

1) Main memory DBMS (main memory was getting big, even a decade ago)
2) No multi-threading!!! (ergo; no latches)
3) No logging!!!! (depend on failover to a replica – HA needed anyway)
4) No locking!!!! (stay tuned)

X10 is a least "in sight"

Design of H-Store/VoltDB

Assume a "tree schema" -- root has a primary key and some attributes
Each descendent is parent child relationship to the root, etc.

Dept - emp - child

Partition root table (e.g. 26 partitions, one for each letter)
partition descendent tables so records are co-located with parents

Allocate a core to each partition -- dedicated – i.e. 26 cores one per letter. Make sure data for that partition is on the right node.

(For now) – assume all transactions are "single partition" In other words, they access a root key and its descendents.

To avoid ruinous overhead communicating with the app program, assume a stored procedure model. Popularized by Sybase circa 1990. (SQL requires multiple round trips to do a single command). E.g. TPC-B – cash a check.

E.g. exec xact (param-1, …, param-k)

A xact is a collection of SQL code intersperced with Java code, but no "stalls"

Execution model:

Application understands the partitioning. Avoids forwarding the message.
One of the params is the root key. Xact sent there.
Executor then executes the xact to completion single threaded. Executor then works on the next one.

No locks!!!
No latches!!!

Runs about 25K xacts per core – typical disk based system does 500. I.e. more than 10X
On a sizeable cluster, runs 2M+ Xacts/sec. Haven't met anybody who wants to go faster.

****
HA: every partition has a backup copy. Every xact is sent to the backup BEFORE executing on the primary. When receive a message back, then execute on the primary. Batch up the messages to minimize TCP overhead.

If a node fails, then comes back up; refresh the node from its partner

The details:

Assume LAN network partitions are very rare. Optimize for single node failures. If primary node fails after sending the message, then the transaction will be done on the secondary. If before, then transaction is undone. If secondary fails, then primary does xact. If a second failure before 1st recovers then STOP.

Could have engineered for multiple failures (e.g. more than 2 replicas); however:

Requires yet more main memory
Very rare!
And human error, operator error much more frequent

What about non-tree schemas (generates multi-part) and multi-part update xacts on tree schemas. <<Can deal with multi-part reads. Details are a little tricky.>>

H-Store really slow. Want a different architecture.

***

Consider H-store-style partitioning and multi-part xacts

Problem 1: 2 partitions on different nodes; 2 items

Item A (P1)	Item B (P2)

Xact-1: read A; write B
Xact-2: read B; write A

Possibly non-serializable. Have to use locks -- so assume dynamic locking (MVCC is more popular. But I don't have time to explain that)

Possible Schedule

(1, R, A)
(2, R, B)
(1, W, B) -- has to wait
(2, W, A) -- has to wait distributed deadlock!!!!

Solutions: assemble waits for graph; look for a cycle; kill somebody
Timeout

Problem 2: active-active

H-Store is so called active-active – requires deterministic scheduling!!– i.e. xacts are run in the same order at the primary and secondary) which H-store guarantees.

However, dynamic locking and OCC/MVCC are not deterministic. Active-active won't work. Must use active-passive

Write a log at the primary
Copy the log over the network
Roll forward on the secondary

Malvaiya ICDE 2014
    Implemented active-passive in VoltDB (3X slower in execution) 3X faster recovery
        But failures are rare!!!!

Problem 3 (multipart)

```
Co-ordinator   ---- worker A
               ---- worker B
```

co-ordinator send subxact to A
At site A: do an update, write the log

Sends subxact to B
At site B; do an update, write the log

(Naïve) coordinator send a commit to A, send a commit to B. A commits the xact; but suppose B fails
    before he receives the commit
    or after he receives the commit and before he can force all the log records to disk.

Xact commits at A and can never commit at B

Uh-oh….

Solution

Co-ordinator sends "prepare" to A and B. Waits for a return. A and B force all log records to disk
Co-ordinator sends commit to A and B -- who commit

This works

All of this is a lot of overhead!!!!!

VoltDb: fastest thing on the planet. But scope limited
MemSQL, Hekaton, … more general but a lot slower – and may not be enough faster to be interesting. Hekaton is not doing all that well vs normal SQLServer.
***