HW, Cloud, DB, Apps

Current Trends in the Full Stack from a Data Management Perspective

Eliezer Levy, Ph.D.
eliezer.levy@huawei.com
Executive Summary

The Cloud affects the architecture of Database Management Systems in several manners.

- As functionality in the cloud is provided by modular micro-services, e.g., a highly-available cloud storage service, and options for self-service and self-tuning become the standard, architectures are adapted accordingly.
- BigData applications that tend to run in the cloud and exhibit Data Variety pose a significant challenge to DBs
- The design of a datacenter is critically dependent on optimizing HW for performance and power. This leads to commoditization of custom HW, and HW-SW Co-Design.
- A DB on the cloud poses excellent opportunity for vertical integration and co-design across HW-OS-Hypervisor-NW-DB stack.

A Cloud-Native DB is a DB that addresses these challenges and leverages these opportunities.
Agenda

• Introduction
• Applications and DBs on Infrastructure
• Infrastructure for Applications and DBs
• Conclusions
Sample of Spectrums of Applications and Infrastructures: The Memory Hierarchy

<table>
<thead>
<tr>
<th>1 TB</th>
<th>10 TB</th>
<th>100 TB</th>
<th>1 PB</th>
<th>10 PB</th>
<th>100 PB</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM</td>
<td>FLASH</td>
<td>HDD</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enterprise OLTP/OLAP/DWH</td>
<td>Human clicks</td>
<td>IoT, Sensor, Video, Genomics,...</td>
<td>DSS for humans</td>
<td>ML by machines</td>
<td></td>
</tr>
</tbody>
</table>
Outline

Applications and Databases

• OLAP:
  – Cloud Storage
  – Azure, RedShift, SnowFlake

• OLTP: Aurora

• Dealing with Data Variety

• ML and DBs

Infrastructures

• …
Cloud Storage

Amazon

- **Instance Store** provides temporary block-level storage to instances (VMs). The data in an instance store persists only during the lifetime of its associated instance.
- **Elastic Block Storage (EBS)**
- **SSD or HDD** for both instance store and EBS
- **S3** object storage

Azure Blob Types

- **Block blobs.** Comprised of variable-size blocks
- **Append blobs.** Optimized for append only.
- **Page blobs.** Used as disks for IaaS VMs. Support random access to an offset within a page. May be up to 1 TB in size.
OLAP on the Cloud

- Simply: Good-Old MPPDB on the Cloud
- Deeper analysis and special aspects:
  - Cloud Storage
  - Total separation of compute and storage tiers → orthogonal scaling
  - Multi-Tenancy
  - Cloud-Native DB for OLAP
Amazon Redshift

- **ParAccel on the Cloud**
- **Leader node**
  - SQL endpoint
  - Stores metadata
  - Coordinates query execution

- **Compute nodes**
  - Local, columnar storage
  - Execute queries in parallel
  - Load, backup, restore via Amazon S3; load from Amazon DynamoDB, Amazon EMR, or SSH

- **Two hardware platforms for Compute nodes**
  - DS2 (Dense Storage): HDD; scale from 2 TB to 2 PB
  - DC1 (Dense Compute): SSD; scale from 160 GB to 356 TB
Microsoft Azure SQL Data Warehouse

- **Control Node**
  - Coordinator of data movement and parallel computation

- **Compute Nodes**
  - SQL processing node with local compute and storage

- **Storage**
  - Persistent data is stored in Azure Storage Blobs. When Compute nodes interact with data, they write and read directly to and from blob storage

- **Data Movement Service (DMS)**
  - Technology for moving data between the nodes. DMS gives the Compute nodes access to data they need for joins and aggregations.
Snowflake I

- **Cloud-Native DWH**
  - Cloud services for management and query optimization
- **Separation of Compute and Storage**
- **Compute**
  - Separate Virtual Warehouse (VW) per tenant
  - Each VW is a shared-nothing cluster of multiple EC2 nodes
  - Predefined cluster sizes (S, M, L) can be changed dynamically
  - Nodes use local SSD/Disk caches (Instance Storage)
- **Storage**
  - Horizontal table partitions are stored in compressed columnar format on S3
Snowflake II

- **Tenant data → VWs → Clusters**
- **Support for Multiple Workloads**
  - Multiple VW (test, production, mining) may share the same data
- **Self Tuning**
  - Multiple clusters per warehouse that can be automatically paused and resumed to dynamically adapt for workload

![Create Warehouse](image)

![Diagram](image)
AWS Aurora Cloud-Native OLTP

- Conventional architectures are monolithic as entire stack is replicated
- Aurora splits the stack at the logging/storage layers and uses a scale-out, multi-tenanted storage service optimized for OLTP
- Log-structured storage
  - Many 10GB segments, each with its own redo log
  - Redo logs used to generate data pages on demand
  - Eliminates chatter between database and storage
- Single Writer, Multiple Readers, no sharding
- Highly available/durable by default
  - 6-way replication across 3 AZs
  - 4/6 write quorum and 3/6 read quorum
  - Continuous backup to S3
- Continuous state exchange protocol between storage nodes
  - Segments may have holes – missing log records
  - Storage nodes gossip to get the latest state of segments
- 5x faster than RDS MySQL 5.6 & 5.7 !!!

Conventional Architectures for OLTP Cluster

Aurora OLTP Cloud Cluster Architecture
IO Traffic among Aurora Storage Nodes

Aurora Read Scaling

MySQL Read Scaling

- Logical: Ship SQL statements to Replica
- Write workload similar on both instances
- Independent storage
- Can result in data drift between Master and Replica

Amazon Aurora Read Scaling

- Physical: Ship redo from Master to Replica
- Replica shares storage. No writes performed
- Cached pages have redo applied
- Advance read view when all commits seen

IO Flow

1. Receive record and add to in-memory queue
2. Persist record and ACK
3. Organize records and identify gaps in log
4. Gossip with peers to fill in holes
5. Coalesce log records into new data block versions
6. Periodically stage log and new block versions to S3
7. Periodically garbage collect old versions
8. Periodically validate CRC codes on blocks

Observations

All steps are asynchronous
Only steps 1 and 2 are in foreground latency path
Input queue is 46x less than MySQL (unamplified, per node)
Favor latency-sensitive operations
Use disk space to buffer against spikes in activity
Survivable Cache

- We moved the cache out of the database process
- Cache remains warm in the event of database restart
- Lets you resume fully loaded operations much faster
- Instant crash recovery + survivable cache = quick and easy recovery from DB failures

Continuous Backup: Segment-Based snapshot & log

- Take periodic snapshot of each segment in parallel; stream the redo logs to Amazon S3
- Backup happens continuously without performance or availability impact
- At restore, retrieve the appropriate segment snapshots and log streams to storage nodes
- Apply log streams to segment snapshots in parallel and asynchronously
Instant Crash Recovery: On-Demand Segment-Based Recovery

- Traditional Databases
  - Have to replay logs since the last checkpoint
  - Typically 5 minutes between checkpoints
  - Single-threaded in MySQL; requires a large number of disk accesses

- Amazon Aurora
  - Underlying storage replays redo records on demand as part of a disk read
  - Parallel, distributed, asynchronous
  - No replay for startup

Crash at $T_0$ requires a re-application of the SQL in the redo log since the last checkpoint.

Checkpointed Data  Redo Log

Adaptive Thread Pool: Polling instead of Interrupts

MySQL Thread Model

- 2 threads per connection

AURORA Thread Model

- epoll() instead of interrupt
  - Re-entrant connections multiplexed to active threads
  - Kernel-space epoll() inserts into latch-free event queue
  - Dynamically size thread pool
  - Gracefully handles 5000+ concurrent client sessions on r3.8xl
Outline

Applications and Databases

• OLAP:
  – Cloud Storage
  – Azure, RedShift, SnowFlake

• OLTP: Aurora

• Dealing with Data Variety

• ML and DBs

Infrastructures

• …
Dealing with Variety on the Cloud

- 3V’s of Big Data: Volume, Velocity, Variety
- Data Deluge → Data-to-Query KPI
- Dimensions of Variety:
  - Data model and Schema: relational, hierarchical, KV, graph
  - Storage format: columnar, row
  - ETL, data curation, data conversion
  - Query language
- Some Approaches:
  - Polystores: Unify query over different data models
  - NoDB: Query on raw files
  - AWS Athena: Schema inference, Scheme-on-Read
  - VIDA, RAW: JIT query engines
  - SQLShare: Sharing costs of data curation and queries by Cloud economics
  - Weld: Optimizing across disjoint modules using a common compact IR
Polystores I

- Unify query over multiple data models
- Extend concept of query optimization and execution plan to include multiple data models, data movement and conversion

![Diagram of Polystores I](image_url)
Polystores II

- A practical, integrative, non-intrusive approach
- Data model Islands, each with potentially multiple Storage Engines
- Extend QL and Algebra with CAST and SCOPE
- CAST: Copy and Convert between Storage Engines
- SCOPE: Used to specify the island for which a subquery is intended
- Shim: Slim Adaptor
NoDB: Access Data Files In Situ

- Redesign the query processing layers of DB systems to incrementally and adaptively query raw data files in situ, while automatically creating and refining auxiliary structures to speed-up future queries.
- Data parsing and processing occur in a pipelined fashion, the raw file is read from disk in chunks and once a tuple or a group of tuples is produced, the *scan* immediately passes those tuples upstream.
- Parse, tokenize only the values required to answer the query.
- Similarly, on-demand statistics.
- Adaptive caching and positional indexing into the raw files.
Schema-on-Read & AWS Athena

- Interactive query service for analyzing data in Amazon S3 using standard SQL
- Amazon, as usual, goes for the “lowest hanging fruit” (Nov, 2016)
- **Schema-on-Read**: Projecting a schema at query run-time independently of the underlying data format
- **Serverless**: No need to provision server VMs
- **No ETL**: You define the table on the data
- Supports CSV, JSON, and a few columnar file formats
- Scales automatically and runs queries in parallel
- Based on Presto SQL

Cloud Storage + Compute
JIT Query Engines

- Traditionally: Data adapts to Engine
- New: Adapt engine to data
- Use JIT, LLVM
- Using a catalog of Input/Output Plugins, formats are abstracted
- Expressions are generated at run-time, fusing abstract operators and concrete plugins
- Operators are agnostic to the underlying data models/formats/properties
Sharing and Cloud Economics

- Data, Views, Queries and Results are shared and reused

1) **Upload data “as is”**
   Cloud-hosted, secure; no need to install or design a database; no pre-defined schema; schema inference; some integration

2) **Write Queries**
   Right in your browser, writing views on top of views on top of views …

3) **Share the results**
   Make them public, tag them, share with specific colleagues – anyone with access can query

http://sqlshare.escience.washington.edu

```
SELECT hit, COUNT(*)
FROM tigrfam_surface
GROUP BY hit
ORDER BY cnt DESC
```
Weld: Common IR

- A runtime for data-intensive applications that optimizes across disjoint libraries and functions
  1. IR:
     - Explicit representation of data parallelism
     - Composition and nesting
     - IR as input and output of transformations and optimizations
  2. Run-Time API. Extend existing libs to expose some of their computations as IR that can be manipulated lazily (JIT) at run-time, e.g.:
     - Loop transformations: fusion, tiling,…
  3. A Compiler Backend that maps the final combined IR to efficient target code, e.g.:
     - Vectorization using Intel’s SIMD AVX2
     - Multithreading
Applications and Databases

- OLAP:
  - Cloud Storage
  - Azure, RedShift, SnowFlake
- OLTP: Aurora
- Dealing with Data Variety
- ML and DBs

Infrastructures

• ...
Machine Learning

- Data flows in ML:
  1. Data from source to compute
  2. Model from storage to compute
  3. Update Model
- Huge variances in data and model sizes, and bandwidth
- Huge variances in computational models
- ML on modern HW: How to manage this messy cross-product?
- Promising optimizations:
  - Sparse matrix calculations
  - Low precision formats instead of expensive floating point arithmetic

\[
\frac{1}{2} \sum_{r} (A_r \cdot x - b_r)^2
\]
ML & DBs

- The Pillars of DBs: Structured data, Shared data
- At present, in ML, data is not typically shared. More like “one-off” data usage
- ML is still a young and mostly ad-hoc, with no single unifying theory
- Work on optimizing data flow #1 by extending the DB kernel with ML computations. Not necessarily the ideal approach. Uphill battle.
- For example, mismatch between CNN and SQL
- Most ML development happens outside DBs
- DB is used mostly for selecting datasets and for provenance: correlating results and data sets
Summary of the “Application” Part

• Cloud services, in particular cloud storage, present new architectural opportunities for data management systems
• ML is so different from DBMS. No elegant and useful way to fit CNN
• But the notions of over-arching optimization of data movement and transformation is appealing
• The best way to deal with diversity and variety is JIT
• The traditional DBMS with a “closed” query optimization and execution scheme is not necessarily the best unifying framework for data management for data-intensive apps
Outline

Applications and Databases
• ...

Infrastructures
• The Custom HW Trend
• CPU: Architectures, Cache Coherency
• ISA: HTM, SIMD
• Accelerators: FPGA
• Memory: NVRAM
• Power
• OS + Networking + Virtualization
• The Co-Design Space for DBs
3 Generation of large-scale datacenter computing:
1. ~2000: Commercial Off-The-Shelf (COTS) computers, switches & racks
2. ~2010: Custom computers, switches & racks using COTS chips
3. ~2020: Custom computers, switches & racks using custom chips
   • Customization becomes commodity
   • Moving from horizontal Linux/x86 model to vertical integration datacenterOS/SoC
   • ARM vs x86 as an example: Much debate on the relative merits of the 2 ISA’s in the context of the 2\(^{nd}/3^{rd}\) generation
   • The openness of ARM is a key advantage over the “closed garden” world of x86
Processors

- From Instruction-Level Parallelism (ILP) and Simultaneous Multi Threading (SMT) to System-on-a-Chip (SoC), GPUs
- Heterogeneity and custom HW
Non-Volatile Memory

- Byte-addressable
- Earlier block-addressable NVMe versions
- Up to 4X system memory capacity, at significantly lower cost than DRAM
- Ultra-fast storage: Faster than network (performance inversion)
- Can deliver big memory benefits without modifications to OS or applications
- Non-volatile load/store semantics:
  - In-place updates
  - Persistent random-access writes
  - Micro-logs (4KB -> 64B) with random access
- Challenges:
  - Data corruption and memory leaks
  - Correctness: Memory fences and cache line flushing
- Rethink some of the traditional implementations
  - Move or cache data structures in DRAM?
  - Even more NUMA behavior: HBM, DRAM, PMEM
  - How to do data replication? No log file means no log shipping
Outline

Applications and Databases
- ...

Infrastructures
- The Custom HW Trend
- CPU: Architectures, Cache Coherency
- ISA: HTM, SIMD
- Accelerators: FPGA
- Memory: NVRAM
- Power
- OS + Networking + Virtualization
- Co-Design Examples:
  - Oracle RAPID
  - Our work on OLTP on the cloud
Power

- Power is the rate of doing work. It is equivalent to an amount of energy consumed per unit time. The unit of power is the joule per second (J/s), known as the Watt.
- The power-performance trade-off: Power = Ops/Sec × Energy/Op
- Energy Proportionality: Energy consumption scales with load
- Dynamic Voltage/Frequency Scaling (DVFS)
- Dark Silicon…

<table>
<thead>
<tr>
<th>Operation</th>
<th>Energy</th>
<th>Scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bit add</td>
<td>0.03 pJ</td>
<td>1</td>
</tr>
<tr>
<td>32 bit add</td>
<td>0.1 pJ</td>
<td>3</td>
</tr>
<tr>
<td>8 bit mult</td>
<td>0.2 pJ</td>
<td>6</td>
</tr>
<tr>
<td>32 bit mult</td>
<td>3.0 pJ</td>
<td>100</td>
</tr>
<tr>
<td>16 bit FP mult</td>
<td>1.0 pJ</td>
<td>30</td>
</tr>
<tr>
<td>32 bit FP mult</td>
<td>4.00 pJ</td>
<td>133</td>
</tr>
<tr>
<td>RISC instruction</td>
<td>&gt; 70 pJ</td>
<td>2300</td>
</tr>
<tr>
<td>LLC Access</td>
<td>100 pJ</td>
<td>3333</td>
</tr>
<tr>
<td>DRAM Access</td>
<td>1000-2000 pJ</td>
<td>~50,000</td>
</tr>
<tr>
<td>Network Access</td>
<td>10 nJ</td>
<td>~500,000</td>
</tr>
</tbody>
</table>
System Energy Equation – Looking at the WHOLE System

- \( E_s \) – System energy
- \( E_t \) – Transactional energy
- \( E_c \) – Computational energy
- \( E_i \) – Idle energy

\[ E_s = E_t + E_c + E_i \]

- **Transactional** – energy that is consumed in order to transfer data elements between devices over an interconnect. Specified as energy per byte
- **Computational** – energy that is consumed by execution of computational actions on data elements (In Chip energy consumption).
- **Idle** – The energy consumed in idle state
Outline

Applications and Databases
• ...

Infrastructures
• The Custom HW Trend
• CPU: Architectures, Cache Coherency
• ISA: HTM, SIMD
• Accelerators: FPGA
• Memory: NVRAM
• Power
• OS + Networking + Virtualization
• Co-Design Space
A Co-Designed DB

- Co-designed mechanisms are added at each layer
- ARM64 only as an example…

<table>
<thead>
<tr>
<th>MMDB</th>
<th>OS/HW-aware concur. contr.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VM-aware memory layout</td>
</tr>
<tr>
<td></td>
<td>NUMA-aware execution</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Linux Linaro</th>
<th>Kernel modifications for OS-assisted MVCC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Kernel bypass networking: DPDK, RDMA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ARM64 ARMv8</th>
<th>Maximal utilization of existing micro-architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ISA Extensions for NVM &amp; HTM</td>
</tr>
<tr>
<td></td>
<td>Networking</td>
</tr>
</tbody>
</table>
### The Co-Design Space

Examples of Mutual Awareness and Assistance across the layers

<table>
<thead>
<tr>
<th>Mechanism Description</th>
<th>Co-Designed Layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVM for persistence to accelerate OLTP commit processing</td>
<td>HW-assisted DB</td>
</tr>
<tr>
<td>HTM and STM for processing transactions at HW speed without limits</td>
<td>HW-assisted DB</td>
</tr>
<tr>
<td>Enhancements in ISA for HTM/NVM and their combination for OLTP</td>
<td>HW-assisted DB</td>
</tr>
<tr>
<td>Using the MMU’s CoW for consistent snapshots (HyperDB) and MVCC-like snapshot isolation. Providing transactional semantics in the HW/OS</td>
<td>OS/HW-assisted DB</td>
</tr>
<tr>
<td>Optimized memory layout of the MMDB, and specialized Virtual Memory sub-system</td>
<td>OS-aware DB</td>
</tr>
<tr>
<td>Affinity-aware request routing in KV store (MICA)</td>
<td>Networking-aware DB</td>
</tr>
<tr>
<td>Specialized IB protocol that combines RDMA and messaging for KV request/response traffic (HERD)</td>
<td>Networking-aware DB</td>
</tr>
<tr>
<td>Using a dedicated light-OS as a guest OS combined with the DB, thereby giving the DB better control of threads and memory</td>
<td>Guest OS and DB</td>
</tr>
<tr>
<td>Exposing dynamic cache-coherence control to the DB via the OS, thereby simplifying many-core interconnects and improving power efficiency</td>
<td>HW, OS and DB</td>
</tr>
</tbody>
</table>
Summary of the “Infrastructure” Part

- Cloud Operator owns full HW-SW stack and therefore can customize it
- Cloud DB poses excellent opportunity for vertical integration and co-design across HW-OS-Hypervisor-NW-DB stack
- A Cloud-Native DB is a DB that leverages these co-design opportunities
Use google for complete citations

- Snowflake https://www.snowflake.net/
- AWS Aurora https://aws.amazon.com/rds/aurora/details/
- Polystores http://dl.acm.org/citation.cfm?id=2814713
- Vida http://dias.epfl.ch/nodb
- NoDB https://cacm.acm.org/magazines/2015/12/194619-nodb/fulltext