GOOGLE ALLOYDB VS. AMAZON AURORA VS. AZURE COSMOSDB

Taras Kloba, Kyrylo Prykhno, PGConf 2022
Our customers want to modernize from their legacy proprietary databases … to standardize on open source.
Cloud offers organizations **agility, cost savings**, and **differentiated capabilities**.

That’s why **75% of all databases** are expected to be in the cloud this year.

*Source: Gartner: The Future of the DBMS Market*
Introducing AlloyDB, a PostgreSQL-compatible cloud database service.

A **cluster** contains all the resources for a PostgreSQL deployment.

A **primary instance** provides the read/write connection point for the databases in a cluster. Every cluster has one primary instance.

A **read pool** instance provides a read connection point for database data in a cluster.
INTELLIGENT DATABASE STORAGE DESIGNED AND OPTIMIZED FOR POSTGRESQL

Powers fast, predictable performance by eliminating I/O bottlenecks and offloading to storage service.

Regional storage improves cluster availability with fast, bounded failover and enables low-lag-to-read replicas.
FAST AND PREDICTABLE PERFORMANCE

Intelligent, workload-aware dynamic data organization leverages both row-based and column-based formats. Format layers of cache ensure excellent price-performance.
ADDITIONAL FEATURES

**EASY TO MANAGE**
- Automatic vacuum management
- Automatic memory management
- Automatic storage tiering
- Automatic data columnarization and query rewrite

**FULLY POSTGRESQL COMPATIBLE**
- Fully compatible with PostgreSQL 14
- Over 175 flags supported
- Over 50 extensions supported
- Move your existing PostgreSQL application as-is, with no code changes

**PREDICTABLE AND TRANSPARENT PRICING**
- No licensing or opaque I/O charges
- Great price-performance
- Right-size instance when needed
- Pay-for-what-you-use storage
3 AWS AURORA
Amazon Aurora is a cloud-based relational database engine that combines the speed and reliability of high-end commercial databases with the simplicity and cost-effectiveness of open-source databases.

Basically, they have taken PostgreSQL and MySQL and replaced the storage layer with a proprietary layer that allows it to be distributed.
PRIME DB INSTANCE
Supports read and write operations and performs the data modifications to the cluster volume. Each Aurora DB cluster has one primary DB instance.

AURORA REPLICA
Connects to the same storage volume as the primary DB instance and supports only read operations. Each Aurora DB cluster can have up to 15 Aurora Replicas in addition to the primary DB instance.
**WRITE NODE**
A single node or endpoint that makes all write requests for the database.

**READ NODES**
Multiple read-only endpoints to meet your read throughput requirements, typically deployed across multiple AZs.

**STORAGE LAYER**
A collection of machines with SSD spread across multiple AZs. Data is written here six times.

The trick is ... everything is decoupled instead of writing locally to attached persistent storage, it writes to this custom, distributed storage layer.
**AURORA: 4 OF 6 QUORUM WRITES**

The storage layer will commit a transaction when four of six copies are written.

They do this so that the database can survive the loss of an availability zone. If two copies were in an AZ, the data can still have four copies.

Reads are guaranteed when three writes are written.
SCALE (FOR READS IN A SINGLE REGION)

In order to scale Aurora, you simply add more instances on top of the shared storage, and they all have immediate access to all data written to the disk.

RESILIENCE (FOR READS IN A SINGLE REGION)

If a read node fails, it can just be recycled, and all queries can just be directed to other instances while it recovers.
AZURE DATABASE FOR POSTGRESQL—HYPERSCALE (CITUS) IS NOW AZURE COSMOS DB FOR POSTGRESQL
A Citus cluster consists of multiple PostgreSQL servers with the Citus extension.
APPLICATION

CREATE TABLE campaigns (...);
SELECT create_distributed_table('campaigns', 'company_id');

AZURE COSMOS DB FOR POSTGRESQL

METADATA

CREATE TABLE campaigns_101
CREATE TABLE campaigns_104
CREATE TABLE campaigns_102
CREATE TABLE campaigns_105
CREATE TABLE campaigns_103
CREATE TABLE campaigns_106
APPLICATION

BEGIN;
UPDATE campaigns
    SET started = true
WHERE campaign_id = 2;
UPDATE ads
    SET finished = true
WHERE campaign_id = 1;
COMMIT;

BEGIN ...
assign_distributed_transaction_id ...
UPDATE campaigns_102 ...
PREPARE TRANSACTION...
COMMIT PREPARED ...
ALL THE FUNCTIONS OF POSTGRES AVAILABLE TO CITUS CLUSTER

- JSONB
- Joins
- Functions
- Constrains
- Indexes:
  - B-tree
  - Gin
  - Brin
  - Gist
- Partial indexes
- Other extensions
- Rich datatypes
- Foreign data wrappers
- Window functions
- CTEs
- Full text search
- pg_stat_statements
4 BENCHMARKING
MY MAIN ADVICE WHEN RUNNING PERFORMANCE BENCHMARKS FOR POSTGRES IS: “AUTOMATE IT!”

Jelte Fennema
WHAT IS HAMMERDB?

• Not a database!
• Leading open-source tool for benchmarking relational databases
• Interfaces:
  ▪ Graphical
  ▪ Command Line
  ▪ Web REST interfaces
• Industry standard benchmarks
• High performance and scalability

https://hammerdb.com
• Hosted by TPC Council since 2019
  ▪ Industry standard body for database benchmarks
• TPC-OSS subcommittee
  ▪ Oversees and approves changes
• V4.1 Released on April 22, 2021
• Source code on GitHub
• Binaries @ GitHub Releases
  ▪ [https://www.hammerdb.com/download.html](https://www.hammerdb.com/download.html)
• Client natively supports Linus and Windows on x64
  ▪ GUI & CLI on both Linux and Windows
• GitHub Release Downloads
  ▪ [https://www.hammerdb.com/stats.html](https://www.hammerdb.com/stats.html)
• Test databases on any platform
**SUPPORTED WORKLOADS**

**TPROC-C = OLTP**
- Transactional workloads. Row oriented, high read and write throughput
- Derived from TPC-C

**TPROC-H = OLAP**
- Analytic, Decision Support
- Focus on ETL
- High bandwidth reads and minimal writes
- Derived from TPC-H

Using TPCC/TPC-C, TPCH/TPC-H for derives workloads not permitted (trademark violation)
• Parallel benchmarking software
  ▪ Concurrency control must be in database, not in client
• Complex workloads designed to scale and test RDBMS concepts
  ▪ Locking and latching
• Cross reference workloads across multiple database engines
  ▪ Validate concepts
• HammerDB up to 6-7 NOPM on commercial database engines on two socket servers
  ▪ High confidence levels that bottlenecks are in database software not HammerDB
SCHEMA BUILD

- Creates tables
- Creates and loads data
- Creates Indexes
- Creates functions/stored procedures
- Gathers statistics

NUMBER OF WAREHOUSES

- Define according to system scale
- Entire schema scaled based on warehouse count

STORE PROCEDURES

- New Order
- Payment
- Delivery
- Stock Level
- Order Status

VIRTUAL USERS TO BUILD SCHEMA

- Schema creates and loads data in parallels
- Use number of CPU cores/threads on HammerDB client
**SCHEMA BUILD CHOICES**

- **Warehouse**
  - `W_{10}`
  - `W_{10}`
  - `10k`

- **District**
  - `W_{10}`
  - `D_{W_{10}, D_{10}}`
  - `3k`

- **Customer**
  - `W_{30k}`
  - `C_{W_{10}, C_{D_{10}, C_{10}}}`
  - `1+`

- **History**
  - `W_{30k}`
  - `W_{10}`

- **Stock**
  - `W_{100k}`
  - `S_{W_{10}, S_{I_{10}}}`
  - `10k`

- **Order-Line**
  - `W_{300k}`
  - `OL_{W_{10}, OL_{D_{10}, OL_{O_{10}}, OL\_NUMBER}}`
  - `3+`

- **Order**
  - `W_{30k}`
  - `O_{W_{10}, O_{D_{10}, O_{10}}}`
  - `10-15`

- **New-Order**
  - `W_{9k}`
  - `NO_{W_{10}, NO_{D_{10}, NO_O_{10}}}`
  - `0-1`

- **Table Name**
  - **cardinality**
  - **Primary key**

---

**PostgreSQL TPROC-C Build Options**

- **Build Options**
  - **PostgreSQL Host:** `c.cosmosdbhammer.postgres.dat`
  - **PostgreSQL Port:** 5432
  - **PostgreSQL Superuser:** `citus`
  - **PostgreSQL Default Database:** `citus`
  - **TPROC-C PostgreSQL User:** `citus`
  - **TPROC-C PostgreSQL User Password:** `postgres`
  - **TPROC-C PostgreSQL Database:** `citus`
  - **TPROC-C PostgreSQL Tablespace:** `pg_default`
  - **EnterpriseDB Oracle Compatible:** 
  - **Citus Compatible:**
  - **PostgreSQL Stored Procedures:**
  - **Prefer PostgreSQL SSL Mode:**

- **Number of Warehouses:** 1
- **Virtual Users to Build Schema:** 1
- **Partition Order Line Table:**

---

**Table**

<table>
<thead>
<tr>
<th>table_name</th>
<th>citus_table_type</th>
<th>distribution_column</th>
<th>shard_count</th>
<th>bigint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>customer</td>
<td>distributed</td>
<td>c_w_id</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>district</td>
<td>distributed</td>
<td>d_w_id</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>history</td>
<td>distributed</td>
<td>h_w_id</td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td>item</td>
<td>reference</td>
<td>&lt;none&gt;</td>
<td>32</td>
</tr>
<tr>
<td>5</td>
<td>new_order</td>
<td>reference</td>
<td>no_w_id</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>order_line</td>
<td>reference</td>
<td>ol_w_id</td>
<td>32</td>
</tr>
<tr>
<td>7</td>
<td>orders</td>
<td>distributed</td>
<td>o_w_id</td>
<td>32</td>
</tr>
<tr>
<td>8</td>
<td>stock</td>
<td>distributed</td>
<td>s_w_id</td>
<td>32</td>
</tr>
<tr>
<td>9</td>
<td>warehouse</td>
<td>distributed</td>
<td>w_id</td>
<td>32</td>
</tr>
</tbody>
</table>
UNDERSTANDING RESULTS: NOPM vs TPM

Vuser 1: Test complete, Taking end Transactional Count.
Vuser 1: 140 Active Virtual Users configurated
Vuser 1: TEST RESULT: System achieves 1722391 NOPM from 5216947 PostreSQL TPM

NOPM
- How fast you are going
- Close relation to official tpmC

TPM
- How hard your engine is working

COMPARING PERFORMANCE
- NOPM can be compared between engines
- TPM can only be compared across the same engine
- TPM useful engineering metric to compare statistics
<table>
<thead>
<tr>
<th>LIMITS</th>
<th>GCP ALLOYDB</th>
<th>AWS AURORA</th>
<th>AZURE COSMOSDB FOR POSTGRESQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max database storage per cluster</td>
<td>5 TiB</td>
<td>128 TiB</td>
<td>2 TiB per worker nodes (max 40 TiB)*</td>
</tr>
<tr>
<td>Max read pool nodes/workers per cluster</td>
<td>20 (15 if all nodes are of the 64 vCPU machine type)</td>
<td>Up to 15 Aurora Replicas in addition to the primary DB instance</td>
<td>Up to 20 workers</td>
</tr>
<tr>
<td>Maximum concurrent connections</td>
<td>Up to 240'000</td>
<td>Up to 16'000</td>
<td>Up to 2'000*</td>
</tr>
<tr>
<td>PostgreSQL compatibility</td>
<td>14</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>Cost per month</td>
<td>USD 8'619,26</td>
<td>USD 3'671.24 + IOPS</td>
<td>USD 8'992.47</td>
</tr>
<tr>
<td>Hardware Configuration</td>
<td>vCPU: 32 – RAM: 256 GB</td>
<td>vCPU: 32 - RAM: 256 GB</td>
<td>Coordinator: 4 vCPU, 16 GB RAM Worker node: 2 nodes x 16 vCPU, 128 GB RAM</td>
</tr>
</tbody>
</table>
OLTP TESTING
(TPROC-C DERIVED FROM TPC-C)

Sample HammerDB benchmark runs

- Azure CosmosDB for PostgreSQL
- Amazon Aurora
- Google AlloyDB for PostgreSQL
PRICE/PERFORMANCE RESULTS
(PRICE/KNOPM)

- Google Alloy DB
- AWS Aurora
- Azure Cosmos DB
• TPROC-H for Analytics
• Cloud Queries
• Stream of 22 Complex queries
• PostgreSQL parallel query
• Columnstores
• More complex skill set required
Preparatory Steps
- DBGEN: Create Flat Data Files
- OGEN: create Executable Queries
- Create Test Scripts
- Create Databases

Load Test
- Create Tables
- Bulk Load Tables
- Create Indexes & Constraints
- Create Statistics
- Install Refresh Functions

Performance Test

Power Test
- Run Refresh Function 1
- Run Query Stream 00
- Run Refresh Function 2

Throughput Test
- Query Stream 01
- Query Stream 02
- ... (omitted)
- Refresh Stream with S pairs of Refresh Functions

First run?
- No
- Yes

Reboot
The lower—the better.
PRICE/PERFORMANCE RESULTS (PRICE/QPHH)

- Google Alloy DB
- Azure Cosmos DB
SUMMARY

- Run PoC(s) to get practical experience and build confidence

- Do full-scale architectural exercises, with “How do I do X?” questions instead of “Can I do X?”

- Try to approach cloud vendors for the best pricing offer
CONTACT ME DIRECTLY

TARAS KLOBA
Big Data Engineering Manager at SoftServe Inc.