Is Scalable OLTP in the Cloud a Solved Problem?

Analyzing Data Access for Distributed OLTP Architectures

Tobias Ziegler, Philip A. Bernstein*, Viktor Leis†, Carsten Binnig

Technische Universität Darmstadt, Microsoft Research*, Technische Universität München†
The Case for Shared Nothing

Michael Stonebraker
University of California
Berkeley, Ca.

ABSTRACT

There are three dominant themes in building high transaction rate multiprocessor systems, namely shared memory (e.g. Synapse, IBM/AP configurations), shared disk (e.g. VAX/cluster, any multi-ported disk system), and shared nothing (e.g. Tandem, Tolerant). This paper argues that shared nothing is the preferred approach.

1. INTRODUCTION

The three most commonly mentioned architectures for multiprocessor high transaction rate systems are:

- shared memory (SM), i.e. multiple processors shared a common central memory
- shared disk (SD), i.e. multiple processors each with private memory share a common collection of disks
- shared nothing (SN), i.e. neither memory nor peripheral storage is shared among processors
DBMS Market (Revenue): $80B/year

source: https://blogs.gartner.com/merv-adrian/2022/04/16/dbms-market-transformation-2021-the-big-picture/
1. Improve conceptual clarity by mapping the distributed OLTP landscape
2. Understand why fully distributed systems have not become standard
3. Discuss research opportunities to get there
This Talk

1. Improve conceptual clarity by mapping the distributed OLTP landscape
2. Understand why fully distributed systems have not become standard
3. Discuss research opportunities to get there

Methodology:
- Distill 4 paradigmatic architectures (“archetypes”)
- Scalability of data access path: uniform/skewed reads/writes
- Elasticity: scaling compute and storage separately
Archetype #1: Single-Writer

examples: AWS Aurora, Azure SQL Hyperscale, Google AlloyDB

uniform reads  uniform writes  skewed reads  skewed writes  elasticity
Archetype #2: Partitioned-Writer

examples: System R*, CockroachDB, Spanner

uniform reads  uniform writes  skewed reads  skewed writes  elasticity
Archetype #3: Shared-Writer (Without Cache)

- RW Node
- RW Node
- RW Node

Shared-Storage

Examples: NAM-DB, Sherman

- Uniform reads
- Uniform writes
- Skewed reads
- Skewed writes
- Elasticity
Archetype #4: Shared-Writer With Coherent Caches (“Shared-Cache”)

examples: Oracle RAC, ScaleStore

uniform reads  uniform writes  skewed reads  skewed writes  elasticity
The Case For Shared-Cache

- good scalability properties
- supports arbitrary workloads (no user-defined partitioning)
- supports arbitrary data structures (e.g., B-trees)
  - difficult implementation, little research
- Cache coherence: ✓
- Altruistic eviction: ?
- Elasticity: ?
- Transactions (ACID):
  - A+C: ✓
  - I: ?
  - D: ?
- HW/Cloud: emerging network technologies (EFA, RDMA), cloud services
So, Is Scalable OLTP in the Cloud a Solved Problem?
So, Is Scalable OLTP in the Cloud a Solved Problem?

- Single-Writer
- Partitioned-Writer
- Shared-Writer
- Shared-Cache

Uniform skewed

reads writes reads writes elasticity

No, but there's a path to getting there.
So, is Scalable OLTP in the Cloud a Solved Problem?

No, but there’s a path to getting there