A1 and FARM
scalable graph database on top of a transactional memory layer

Miguel Castro, Aleksandar Dragojević, Dushyanth Narayanan,
Ed Nightingale, Alex Shamis, Richie Khanna,
Matt Renzelmann, Tim Tan,
Chiranjeeb Buragohain,
Knut Magne Risvik

Microsoft Confidential
Distributed computing today

no general platform for low latency at scale

SQL
RDBMS

Hadoop/MapReduce

1 machine

> 1K machines
A platform for low latency computing

- SQL RDBMS
- A1 FARM
- Hadoop/MapReduce

Latency:
- ms
- s

Scale:
- 1 machine
- > 1K machines
A new frontier for low latency computing
A graph workload

- **me**
- **meeting**
- **Bob**
- **Ann**
- **Jane**

Documents:
- **doc1**
- **doc2**
- **doc3**
- **doc4**
- **doc5**
Scale out

server 1

server 2

meeting

Ann

doc5

Bob

doc2

server 3

Jane

doc3

doc4

server 4

me

Latency

me
server 1

meeting
server 2

Bob
doc1
doc2
server 3

Ann
doc5

Jane
doc3
doc4
server 4
Freshness and consistency

server 1

me

server 2

meeting

doc5

Ann

server 3

Bob

doc1

doc2

server 4

Jane

doc3

doc4
Fault tolerance
It is hard to build low latency apps at scale
FARM simplifies building low latency apps.

Transactions and replication simplify programming. Hide failures, distribution, and concurrency.

Abstraction that transactions run sequentially on a reliable server.

No compromises: consistency, availability, performance.

Strict serializability.

Millions of transactions per second with sub-millisecond latencies.

Recovery from failure to peak performance in tens of milliseconds.
FARM is enabled by three hardware trends:

1. Inexpensive DRAM
   - Currently $8/GB
   - Machines with 128 GB, container will hold more than 100 TBs

2. Non-volatile RAM
   - NVDIMMs available today: DRAM+battery+SSD

3. Fast commodity networks with RDMA
   - CX3: 40 Gb/s and < 3 µs latency on Ethernet (CX4: 100 Gb/s)
   - CX3: 35M messages per second (CX4: 150M)

This hardware is commodity and being brought into datacenters.
FARM software

new algorithms and data structures optimized for RDMA
single-object read only transactions with a single RDMA read
Transaction commit protocol optimized for RDMA operations.
read validation during commit using a single RDMA read
btree and hash table lookups with a single RDMA read

changes to OS and network card driver to speed up RDMA
efficient memory translation
connection multiplexing
FARM API

Storage APIs

Transaction CreateTransaction()
ObjBuf *Transaction::Alloc(size, locality_hint)
ObjBuf *Transaction::Read(addr, size)
ObjBuf *Transaction::OpenForWrite(ObjBuf)
void Transaction::Free(ObjBuf)
void Transaction::Commit()

Communication APIs for application level coordination
Scale-out OLTP throughput: TATP

- In-memory SQL (no fault tolerance)
- FARM (no fault tolerance)
- FARM (with fault tolerance)

30x
100x database size

transactions per second (millions)

number of servers
Why a Graph Database? On FARM?

- Many datasets naturally lend themselves to a graph structure
  - Facebook, Twitter, enterprise relationships

- Graph traversal queries are hard on relational stores

- A1 Goals
  - Scalable
  - Transactional
  - Complex traversal and subgraph queries

- Not targeted for graph compute
A1 Architecture overview

- A1 Trusted and untrusted coprocessors
- A1 graph database core
  - A1 Core API
- FARM ACID Transactions
- FARM Shared memory
- FARM Communication primitives
The A1 Data Model

```json
{
  "name": "Me",
  "title": "SDE",
  "DOB": "1970-01-01"
}
```

```json
struct PersonData {
  0: string name;
  1: string title;
  2: timestamp DOB;
}
```

```json
{
  "invited": "2015-09-01",
  "accepted": "2015-09-05"
}
```
A1 Data Layout

- Locality of reference matters
- Graph: a set of vertexes with adjacency lists
- Building blocks
  - FARM objects identified by 64 bit pointers
  - RPC primitives
  - BTree for indexes
  - Linked lists for adjacency
A1 Data Layout

Overall data layout for A1 showing vertexes, edges and indexes
BFS Query

```json
{
  "type": "Person",
  "name": "Me",
  "Attended": {
    "type": "Meeting",
    "date": "2015-09-01",
    "Attended": {
      "type": "Person",
      "Edited": {
        "type": "Document"
      }
    }
  }
}
```
Traversals Algorithm

• Designate starting host as coordinator (maintain visited list)
• Coordinate level by level traversal

  • Data Shipping

  • Query shipping
    • RPC for synchronization
    • All predicates applied locally
    • All neighbors discovered locally

  • Hybrid
BFS Traversal via Query Shipping

1. Start
2. Find neighbors
3. Coord: Partition and ship query
4. Worker: Find neighbors
5. Worker: Ship list back to coordinator
6. Back to step 3
Extensibility by Coprocessors

**Coprocessor models**

**Trusted – core extensions**
- No separation, running inside the FARM process space
- No latency

**Untrusted core hosted**
- Separate address and process space on same machine.
- 1-2us (10-25% overhead)

**Untrusted cluster hosted**
- Separate machines on the same failure/network domains
- 50-200 us (4x – 16x overhead)
Preliminary Performance

Linkbench with 1B vertexes and 4.5B edges
3X replication
Single container / network domain / failure domain

Single transactions
Vertex Creation: 375 us
Edge Create: 495 us
Edge Get: 240 us
Vertex Read: 100 us

BFS
1 hop: 700 us
2 hop: 850 us

Throughput
~ 350K transactions/second/node
Linear scale-up to 100 nodes
Backup slides
Important applications can benefit

- petabyte transactional memory

- Facebook

- Hekaton

- Delve commerce graph
Petabyte transactional memory makes it easy

- general transactional memory model
  - objects, arrays, and pointers
  - graphs, relational databases, key-value stores on the same system
- abstraction of running on a single machine
  - scale out
  - low latency
  - freshness and consistency
  - fault tolerance
- high throughput to keep costs low

it will speed up innovation as MapReduce did for batch analytics
# Data Model Comparison

<table>
<thead>
<tr>
<th>Data Model</th>
<th>Horizontal Scaling</th>
<th>Join Support</th>
<th>Language Support</th>
<th>Maturity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key-Value</td>
<td>Easy</td>
<td>None</td>
<td>None</td>
<td>Medium</td>
</tr>
<tr>
<td>Relational</td>
<td>Non-trivial</td>
<td>Good</td>
<td>SQL</td>
<td>High</td>
</tr>
<tr>
<td>Graph</td>
<td>Hard</td>
<td>Join-less</td>
<td>Specialized</td>
<td>Low</td>
</tr>
</tbody>
</table>
Scale-out OLTP throughput-latency: TATP
Scale-out OLTP recovery: TATP

![Graph showing scale-out OLTP recovery with TATP at 45ms]

- suspect probe: 1.1ms
- agreement: 4.0ms
- lease: 10.2ms
- transactions: 45.4ms
- all-active: 63.0ms
Scale-out OLTP: TPC-C benchmark

(tpmC)

(number of servers)

Millions

FARM

Oracle