# A1: A Distributed In-Memory Graph Database

Chiranjeeb Buragohain, Knut Magne Risvik, Paul Brett, Miguel Castro, Wonhee Cho, Joshua Cowhig, Nikolas Gloy, Karthik Kalyanaraman, Richendra Khanna, John Pao, Matthew Renzelmann, Alex Shamis, Timothy Tan, Shuheng Zheng

Microsoft

In Proceedings of the 2020 ACM SIGMOD

Chih-Chuan Hsu 森林所碩二 R10625016

## ABSTRACT

A1, an in-memory distributed database, is designed based on FaRM (Fast Remote Memory) distributed system, which integrates RDMA (Remote Direct Memory Access) to facilitate high-speed data access, in order to support complex queries under latency constraints in Bing search engine. Due to the higher cost of memory compared to SSD, A1 is designed according to efficient in-memory storage scheduling and data locality. A1 achieves concurrency control through an optimized graph abstraction schema and employs 2 recovery strategies to ensure data integrity.

## 1. Problem Definition

To efficiently conduct complex queries the data by Bing, they need a OLTP platform to handle large amount of data and obtain strict performance and latency control.

## 2. Prior Work

The Facebook TAO datastore uses graph data model and a two-tier approach with Memcached for read-only data to achieve low-latency. However, it introduces three main problems. First, Memcached uses primitive key-value API with little query capability. Second, cache only guarantees eventual consistency. Third, no atomicity for data updates.

## 3. Solution

### 3.1 FaRM

FaRM is a transactional distributed in-memory storage system, which uses RDMA capable Network Interface Cards (NIC). To lower latency, RDMA bypasses the OS kernel in the local machine and accesses memory without utilizing CPU in the remote machine. A1 integrates FaRM in a coprocessor model to share the resources by cooperative multithreading, making the synchronous illusion with asynchronous FaRM API Calls, and FaRM defines a set of operations on objects to allocate resources.

### 3.2 Graph Abstraction

A1 adopts graph model with an enforced schema as Table 1. Every vertex and edge must belong to one of the types defined in the graph. Vertex is stored in 2 FaRM objects, one for the header, and the other for the data. The header specifies the outgoing edge list and incoming edge list. The edge is stored in 3 FaRM objects, one for edge data, and the others specify the source vertex and destination vertex. This implementation builds a scalable data structure which supports fast lookup and update. For example, a vertex connected with thousands of edges can be flexibly updated when deleting the other connected vertex by simply traversed through the edge and delete the edge in edge object and vertex object. The vertex and corresponding edge list are allocated using locality to improve query execution.

Table 1. Analogy between relational database entities and A1 entities

| A1 | Graph | Type | Vertex/Edge | Attribute |
|---|---|---|---|---|
| Relational | Database | Table | Row | Column |

### 3.3 Query Execution

A1 aims to handle a large amount of read queries efficiently, and distribute update queries to the random backend machine. Every query is formatted as a JSON document, which can explicitly specify the traversal hierarchy through edges. For a query process, the query coordinator will assign the jobs to different FaRM workers to parallelly execute, and the coordinator will aggregate the result of queries from different workers. Coordinator will communicate with backend workers through RPC (Remote Procedure Call), and assign the work including specific vertexes to the hosting machine.

### 3.4 Recovery Strategy

A1 replicates data in memory 3-ways. However, A1 replicates data asynchronously, and it may not commit all the rights when the disaster occurs. To address the problem, A1 leverages two strategies, consistent recovery and best-effort recovery. For consistent recovery, A1 fulfills atomicity. For best-effort recovery, A1 at least recovers the data as up to date as consistent recovery, and more up to date at most of the time based on the timestamp. For example, best-effort recovery will check if the upcoming update timestamp is older than the current update. If yes, then discards it. In the update is newer, then conduct it.

## 4. Results

Bing uses A1 to support knowledge graph usage. With the increasing types of knowledge graph, A1 schedule their storage by allowing only queryable attributes to be stored in A1, while non-queryable attributes such as image data are stored elsewhere. Based on this, A1 achieve a shipping of more than 95% of local reads to workers. Also, A1 improves the average latency of knowledge serving system by 3.6X.

## 5. Critique

The most impressing part of A1 is the graph data model design. In a search engine such Bing, the entity may be linked to lots of entities. If we use relational database, it will be a huge burden to make the query using thousands of foreign keys and cause high latency. Second, the problem definition of certain implementations in A1 is appealing. For example, they don't want to focus on the large queries which exceeds the memory storage, and they will simply fail that query. They explain the goal of A1 is to handle complex queries instead of large working set. The low-latency goal is clear in their design, and that is the most difficult part to schedule.

## 6. Extension

First, A1 lacks a query optimizer, which is worth designing. Second, the mapping of users' queries to A1QL queries is a black-box in this paper, and using machine learning to calculate similarity to improve database query performance would be a new advantage for A1. Following the scheduling of large query set, we may tailor on a system integrating A1 and a new system to handle large query set. Both complex and large queries can be handled.

## 7. REFERENCES

[1] Fröhlich, B. and Plate, J. 2000. The cubic mouse: a new device for three-dimensional input. In *Proceedings of the SIGCHI Conference on Human Factors in Comput*