Building a Distributed Graph Database in Rust

ZHENGYI YANG | Rust Meetup, Sydney

About Me

- PhD Student @ Data and Knowledge Research Group, UNSW (2018 - present)
- Research Interests: Graph Database, Distributed Graph Processing, etc.
- Rust (~ 2 years) & Python (~ 6 years)

Zhengyi Yang

http://zhengyi.one

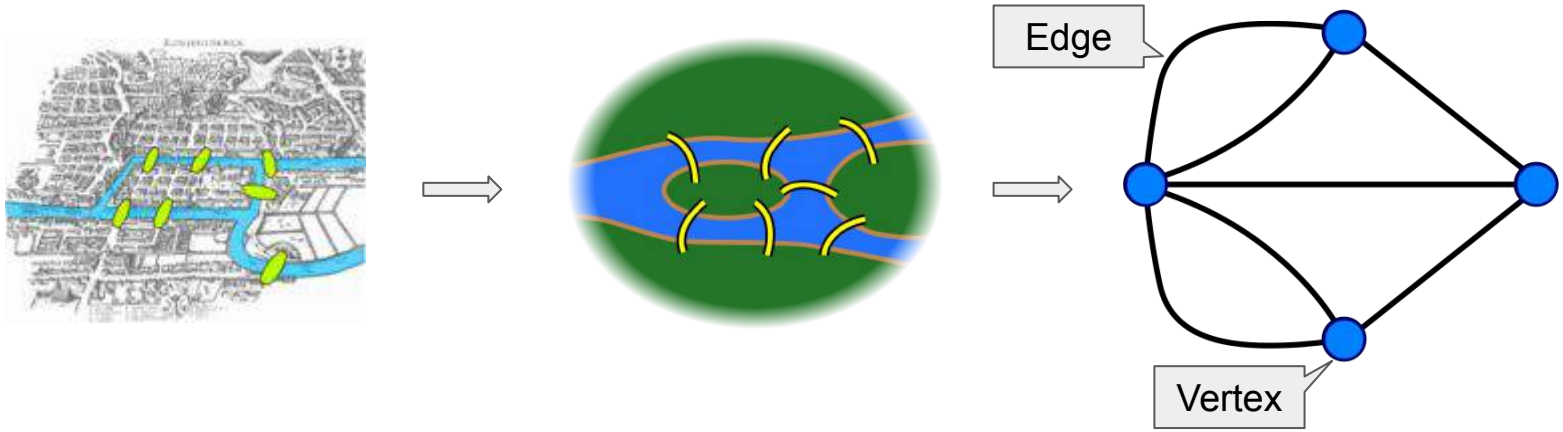
Contents

1. Introduction to Graph Database

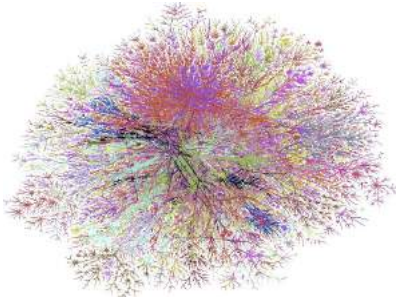
What are graph databases? Why are they so useful?

What is a graph?

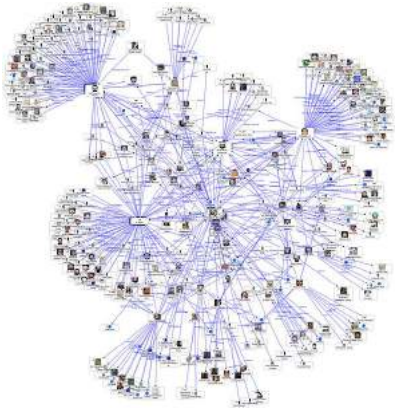
- A ***graph*** is a structure in mathematics (graph theory)
- Famous problem: Seven Bridges of Königsberg
- Optimised for handling highly connected data



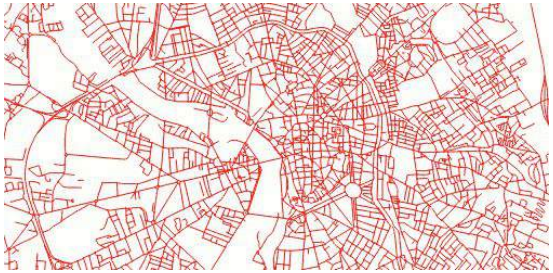
Graphs are indeed everywhere!



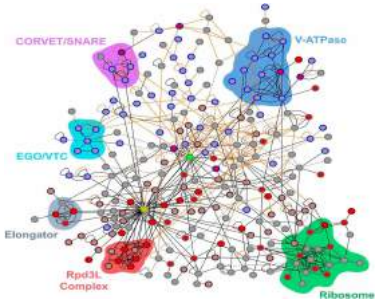
Internet



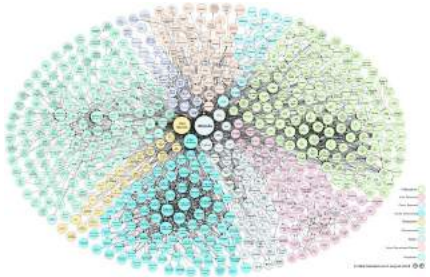
Social Networks



Road Networks



Biological Networks



Knowledge Graphs

Graphs are indeed very large!

#Vetices	Ratio
<10K	17.3%
10K-100K	17.3%
100K-1M	15.0%
1M-10M	13.4%
10M-100M	15.7%
>100M	21.3%

#Edges	Ratio
<10K	17.8%
10K-100K	17.1%
100K-1M	10.1%
1M-10M	6.9%
10M-100M	16.3%
100M-1B	16.3%
>1B	15.5%

#Bytes	Ratio
<100MB	19.0%
100MB-1G	15.7%
1G-10G	20.7%
10G-100G	14.1%
100G-1T	16.5%
>1T	14.0%

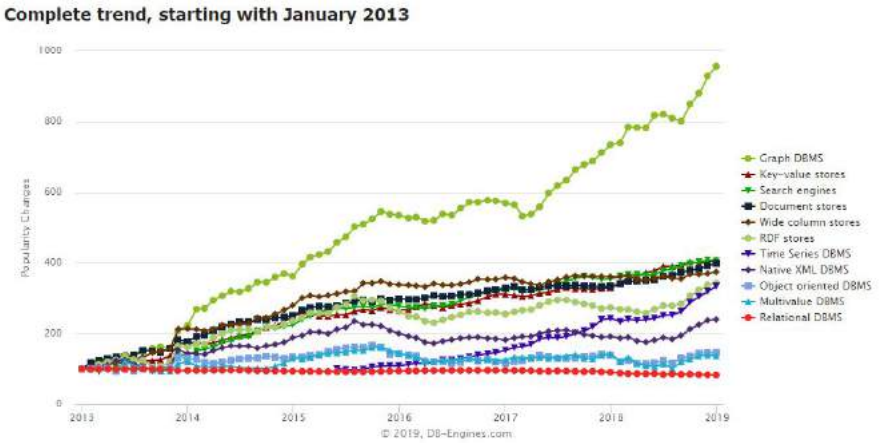
>1 trillion connections

>60 trillion URLs

>60 billion edges every 30 days

Sahu, S., Mhedhbi, A., Salihoglu, S. et al. The ubiquity of large graphs and surprising challenges of graph processing: extended survey. The VLDB Journal (2019)

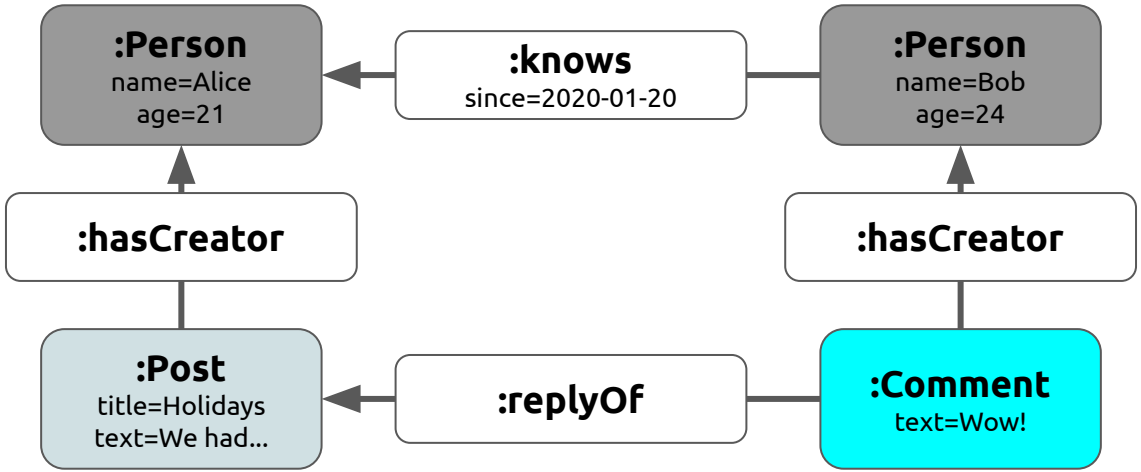
Graph DBMS Landscape



DBMS popularity trend by database model between 2013 and 2019 – DB-Engine

The graph database landscape in 2019

Labeled Property Graph Model



- **Labels:** types (or classes) of vertices and edges
- **Properties:** arbitrary *(key,value)* pairs where *key* identifies a property and *value* is the corresponding value of this property

Types of Graph Queries

Graph Pattern Matching

- Given a graph pattern, find **subgraphs** in the database graph that match the query.
- Can be augmented with other (relational-like) features, such as *projection*.

Graph Navigation

- A flexible querying mechanism to navigate the topology of the data.
- Called **path queries**, since they require to navigate using paths (potentially variable length).

Cypher Graph Query Language

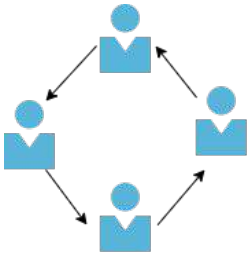
- A declarative graph querying language developed by Neo4j
- Patterns are intuitively expressed using brackets and arrows: encode vertices with “()” and edges with “->”.
  - Graph pattern query

```
MATCH (p:Person)-[:LIKES]->(:Language {name = "Rust"})
RETURN p.name
```

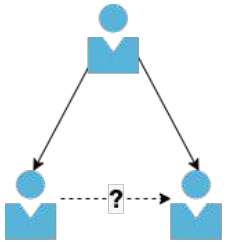
  - Path query

```
MATCH (p:Person)-[:KNOWS*1..2]->(:Person {name = "Alice"})
RETURN p.name
```

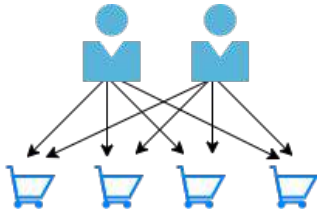
Graph Database Use Cases



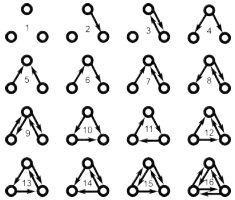
Fraud Detection



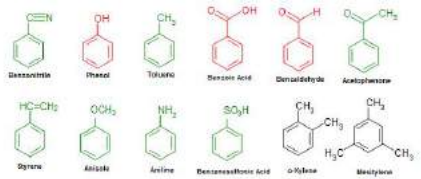
Link Prediction



Recommender System



Network Motif Computing



Chemical Compound Search

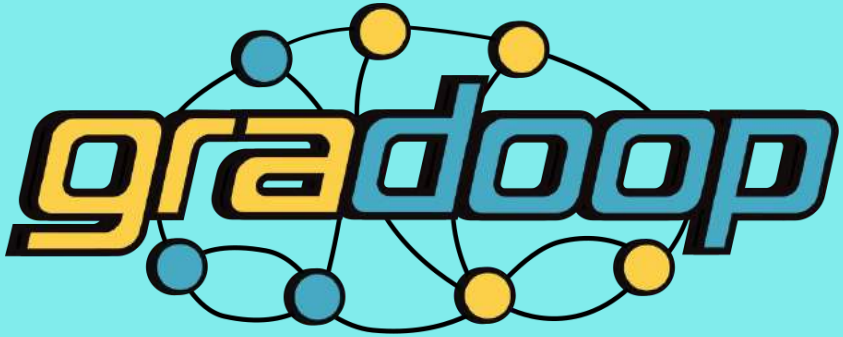
Network Monitoring and IOT

2. The Rust Approach - PatMat

Why are we building our own distributed graph database? How does it perform?

Graph Database Systems using Cypher

Single Machine
Lack Scalability

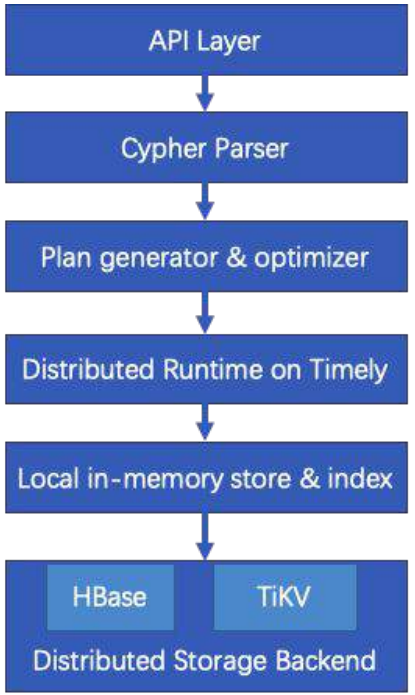


Morpheus
SQL and Cypher® in Apache® Spark

Suboptimal Algorithms
Lack Performance

PatMat: A Cypher-driven Distributed Graph Database

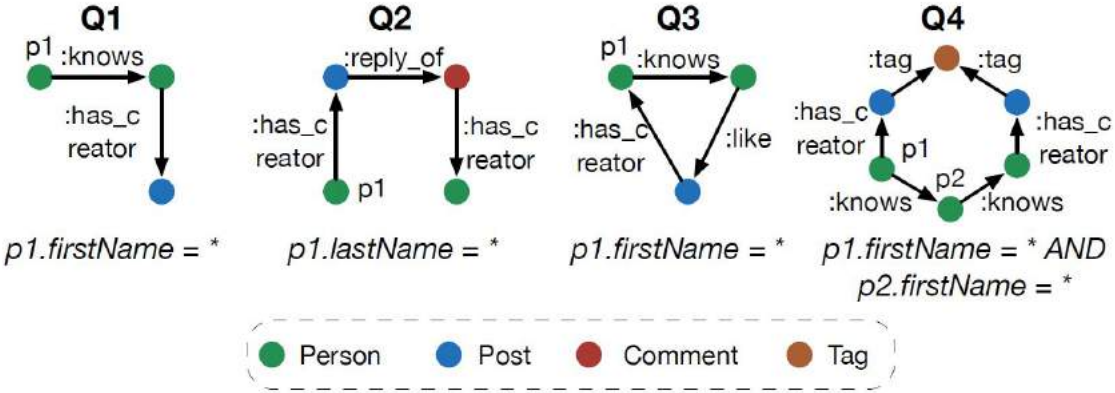
- Glue together the academic efforts on performance and the industrial efforts on expressiveness
- Targeting on high performance and scalability together with full *Cypher* support
- Started in late-2018 originally as a research project
- Practically 100% Rust, 100% safe(25k+ lines of Rust code for the core)
- Still a work-in-progress, currently all part-time developers



Hao, Kongzhang, et al. "PatMat: A Distributed Pattern Matching Engine with Cypher." *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2019.

How does PatMat perform?

- Data Graph (LDBC_SNB benchmark)
  - Simulate a Facebook-like social network over 4 years
  - 187.11 million nodes, 1.25 billion edges (65GB in text, 170GB in Neo4j)
- Query Graph:



Single Thread Evaluation

	Q1/s	Q2/s	Q3/s	Q4/s
Neo4j	87	594	236	**182**
PatMat	**12**	**24**	**17**	256

Large Index

- Configuration: Xeon CPU E5-2698 v4 @ 2.20GHz (use only 1 thread), 512GB RAM, 2 TB disk

Distributed Evaluation

	Q1/s	Q2/s	Q3/s	Q4/s
Gradoop	OUT OF MEMORY	OVERTIME	OUT OF MEMORY	OUT OF MEMORY
Morpheus	OVERTIME	OVERTIME	OVERTIME	OVERTIME
PatMat	**2.6**	**9.4**	**5.3**	**77.3**

- Configuration: 10 machines (Xeon CPU E3-1220 V6 3.00GHz, 64GB RAM, 1 TB disk, 10GBps)

Why do existing distributed solutions perform poorly?

1. **Poor Matching Algorithms**
    a. Graph pattern matching is, in theory, NP-complete
    b. Existing solutions typically adopt naive matching algorithms resulting in high time complexity
    c. Poor matching algorithms also lead to large amount of intermediate result that significantly increase the memory consumption and communication cost
2. **High System Costs**
    a. The design and implementation of distributed systems (e.g. Spark and Flink) add overheads and increase the costs
3. **Restricted Programming Interface**
    a. Distributed engines usually provide limited APIs and programming model (e.g. Mapreduce for Spark)
    b. It is hard to implement advanced algorithms and optimizations (e.g. worst-case optimal join)

3. Rust Dependencies for PatMat

What libraries are we using? Why do we love Rust?

Timely Dataflow

- A distributed data-parallel compute engine based on the dataflow computation model (https://github.com/TimelyDataflow/timely-dataflow)
  - high-performance and low-latency
  - highly scalable and flexible
  - suitable for both streaming processing and batch processing
- The ecosystem
  - Timely Dataflow:
    - primitive operators: *unary*, *binary*, etc
    - standard operators: *map*, *filter*, etc
  - Differential Dataflow (https://github.com/timelydataflow/differential-dataflow)
    - higher-level language built on Timely Dataflow
    - operators: *group*, *join*, *iterate*, etc

Timely Example 1

```
extern crate timely;

use timely::dataflow::operators::*;
use timely::dataflow::*;

fn main() {
    timely::execute_from_args(std::env::args(), |worker| {
        let index = worker.index();
        let mut input = InputHandle::<u32, u32>::new();

        worker.dataflow(|scope| {
            scope
                .input_from(&mut input)
                .exchange(|&x| x as u64)
                .inspect(move |x|
                    println!("worker {}:\thello {}", index, x));
        });

        for round in 0..10 {
            if index == 0 {
                input.send(round);
            }
        }
    })
    .unwrap();
}
```

initialize and run a dataflow

workers are indexed 0 to (#workers-1)

define InputHandle<Timestamp, Data>

create a new input

shuffle the data to x%#workers

inspect the output

send data on Worker 0

using 4 workers

```
% cargo run -- -w 4
   Finished dev [unoptimized + debuginfo] target(s) in 0.14s
    Running `target/debug/example -w 4`
worker 1:	hello 1
worker 1:	hello 5
worker 3:	hello 3
worker 3:	hello 7
worker 1:	hello 9
worker 0:	hello 0
worker 0:	hello 4
worker 0:	hello 8
worker 2:	hello 2
worker 2:	hello 6
```

Unordered

```rust
extern crate timely;

use timely::dataflow::operators::*;
use timely::dataflow::*;

fn main() {
    timely::execute_from_args(std::env::args(), |worker| {
        let index = worker.index();
        let mut input = InputHandle::<u32, u32>::new();
        let mut probe = ProbeHandle::new();

        worker.dataflow(|scope| {
            scope
                .input_from(&mut input)
                .exchange(|&x| x as u64)
                .inspect(move |x|
                    println!("worker {}:\thello {}", index, x))
                .probe_with(&mut probe);
        });

        for round in 0..10 {
            if index == 0 {
                input.send(round);
            }

            input.advance_to(round + 1);
            while probe.less_than(input.time()) {
                worker.step();
            }
        }
    })
    .unwrap();
}
```

Monitor the progress

Loops until all workers have processed all work for that epoch

★ **Control memory consumption**

Timely Example 2

```
% cargo run -- -w 4
   Finished dev [unoptimized + debuginfo] target(s) in
0.14s
    Running `target/debug/example -w 4`
worker 0:    hello 0
worker 1:    hello 1
worker 2:    hello 2
worker 3:    hello 3
worker 0:    hello 4
worker 1:    hello 5
worker 2:    hello 6
worker 3:    hello 7
worker 0:    hello 8
worker 1:    hello 9
```

Ordered

… does it work for graph processing?

PageRank (20 iterations)	Cores	twitter_rv (41 million nodes,1.5 billion edges)	uk_2007_05 (105 million nodes,3.7 billion edges)
Spark	128	857s	1759s
Giraph	128	596s	1235s
GraphLab	128	249s	833s
GraphX	128	419s	462s
Laptop (Rust)	1	**110s**	**256s**
Timely 	128	**15s**	**19s**

Frank McSherry, Michael Isard, and Derek G. Murray. 2015. Scalability! but at what cost? (HOTOS'15)

Other Crates

- **TiKV**: fast distributed key-value database
- **rust-rocksdb** : Rust wrapper for RocksDB
- **tarpc**: pure Rust RPC framework
- **Tokio**: well-known asynchronous runtime
- **Rayon**: to do parallel computation easily
- **threadpool:** basic thread pool
- **crossbeam**: useful tools for concurrent programming
- **parking_lot**: easy-to-use locks
- **hdfs-rs:** *libhdfs* binding for Rust
- **Thrift**: connect to HBase
- **lru-rs**: efficient LRU cache
- **iron**: web API support
- **Serde**(Bincode/JSON/CBOR): serialization and deserialization
- **itertools**: extended iterators
- **FxHash/SeaHash/fnv**: fast hashing
- **rust-snappy**: fast snap compression
- **indexmap/fixedbitset**: useful data structures
- **rust-csv**: load and export in csv format
- **Clap**: parsing command line arguments
- **libc**: interoperate with C code(e.g. *libcypher*)

Graph Analytics in Rust

petgraph

Graph data structure library in Rust.
(https://github.com/petgraph/petgraph)

rusted_cypher

Rust crate for accessing a neo4j server.
(https://github.com/livioribeiro/rusted-cypher)

indradb

A simple graph database written in Rust.
(https://github.com/indradb/indradb)

… …

We love Rust !

- **Performance**
  - Blazing fast
  - No garbage collector
- **Reliability**
  - Guaranteed memory safety
  - “Fearless Concurrency”
- **Productivity**
  - Modern development tools
  - Lots of amazing libraries
- **and many more…**

Thanks!

Does anyone have any questions?

https://github.com/UNSW-database

zyang@cse.unsw.edu.au