CLOUD COMPUTING APPLICATIONS

Graphs: Databases - OLTP

Prof. Reza Farivar

Neo4j

- Neo4j is an open-source, NoSQL, native graph database that provides an ACID-compliant transactional backend
- Cypher, a declarative query language similar to SQL, but optimized for graphs
- Morpheus: Cypher for Apache Spark

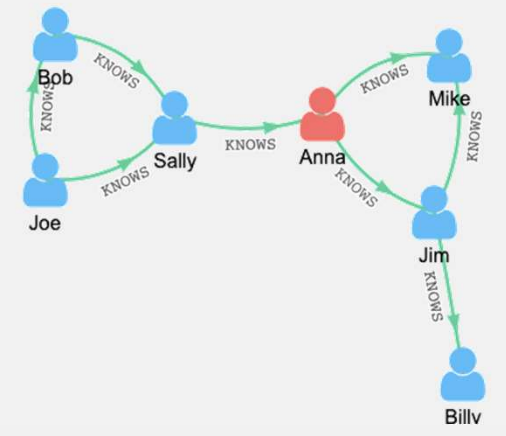
Cypher

- Nodes (vertexes) are surrounded by parenthesis: () or {p}
- Labels start with : and group the node by roles or types
  - (p:Person:Mammal)
- Nodes can have properties
  - (p:Person {name: 'Jim'})
- Relationships (edges) are wrapped with square brackets and can have properties
  - --> or -[h:HIRED]->
- Direction of relationship is specified with < >

```
MATCH
        (person:Person)-[:KNOWS]-(friend:Person)-[:KNOWS]- (foaf:Person)
WHERE
        person.name = "Joe"
        AND NOT (person)-[:KNOWS]-(foaf)
RETURN
        foaf
```

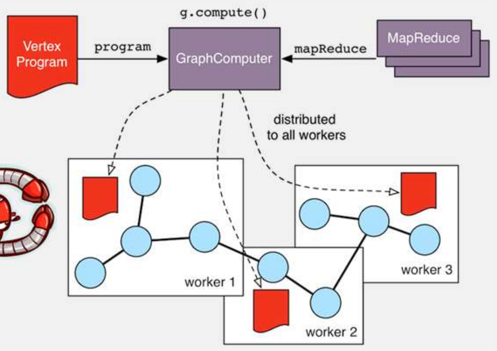
- OpenCypher is widely used
  - Neo4J, SAP HANA, AnzoGraph, Cypher for Apache Spark (CAPS), Redisgraph, Cypher for Gremlin, …

openCypher



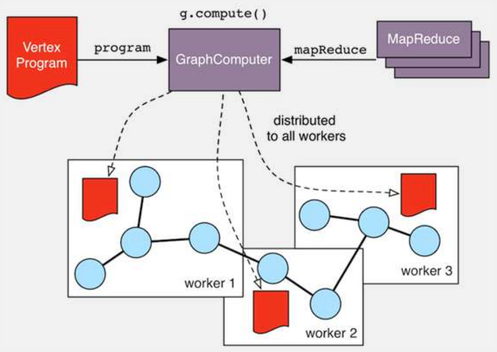
TinkerPop & Gremlin

- Supports both OLTP queries and OLAP algorithms
- TinkerPop 3 Components, altogether known as Gremlin
  - **Blueprints** → Gremlin Structure API
  - **Pipes** → GraphTraversal
  - **Frames** → Traversal
  - **Furnace** → GraphComputer and VertexProgram
  - **Rexster** → GremlinServer
- A *traversal* in Gremlin is a series of chained steps. It starts at a vertex (or edge). It walks the graph by following the outgoing edges of each vertex and then the outgoing edges of those vertices.



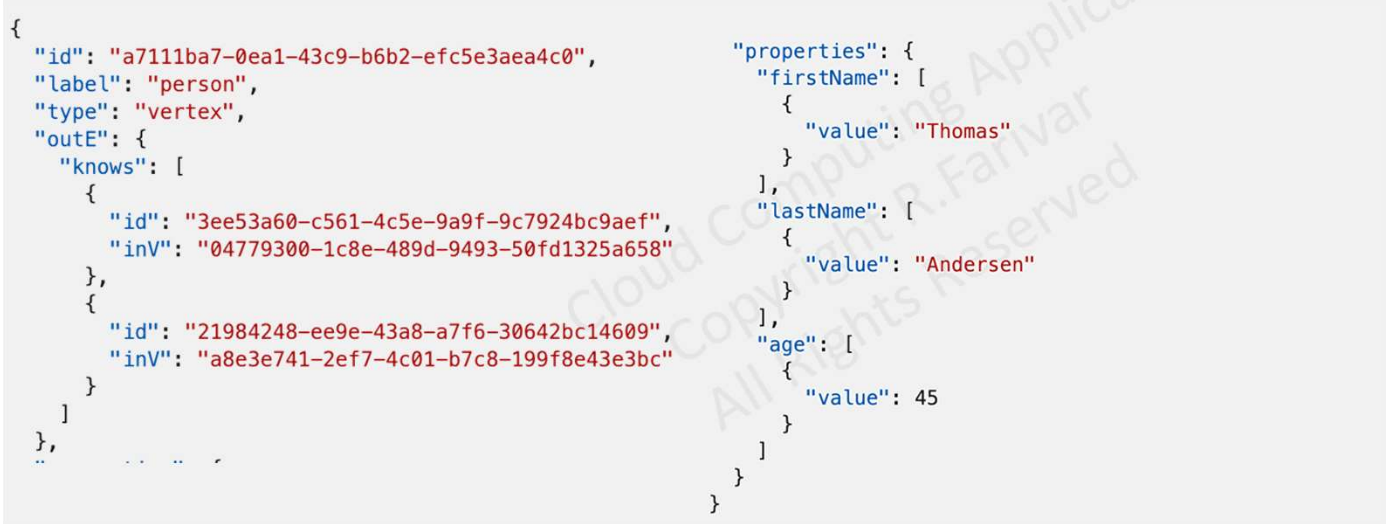
TinkerPop & Gremlin

- BSP type graph algorithms, similar to Pregel model, via VertexProgram
  - executed at each vertex in logically parallel manner until some termination condition is met (e.g. a number of iterations have occurred, no more data is changing in the graph, etc.)
  - The vertices are able to communicate with one another via messages
    - MessageScope.Local and MessageScope.Global
- At the core of TinkerPop3 is a Java8 API. The implementation of this core API is all that is required of a vendor wishing to provide a TinkerPop3-enabled graph engine.
- Gremlin is widely used
  - Amazon Neptune, Azure CosmosDB, DataStax, JanusGraph, OrientDB, …



GraphSON

- The Gremlin standard format for representing vertices, edges, and properties (single and multi-valued properties) using JSON
- Example: a vertex representation



```
{
  "id": "a7111ba7-0ea1-43c9-b6b2-efc5e3aea4c0",
  "label": "person",
  "type": "vertex",
  "outE": {
    "knows": [
      {
        "id": "3ee53a60-c561-4c5e-9a9f-9c7924bc9aef",
        "inV": "04779300-1c8e-489d-9493-50fd1325a658"
      },
      {
        "id": "21984248-ee9e-43a8-a7f6-30642bc14609",
        "inV": "a8e3e741-2ef7-4c01-b7c8-199f8e43e3bc"
      }
    ]
  },
```

```
  "properties": {
    "firstName": [
      {
        "value": "Thomas"
      }
    ],
    "lastName": [
      {
        "value": "Andersen"
      }
    ],
    "age": [
      {
        "value": 45
      }
    ]
  }
}
```

Gremlin graph traversal language (Imperative)

- **functional language**
- traversal operators are chained together to form path-like expressions
  - For example, "from Hercules, traverse to his father and then his father’s father and return the grandfather’s name."

```
gremlin> g.V().has('name', 'hercules').out('father').out('father').values('name’)

==>Saturn
```

- Find all movies that John Doe appeared in
  - `gremlin> g.V().has('name’,’John Doe’).out().values()`
- Find all relationships from a given actor/actress to John Doe (the six degrees)
  - `gremlin> g.V().has('name',’John Doe’).repeat(out().in().simplePath()) .until(has('name’,’Jane Doe')).path().by('name').by('title').limit(10)`

Gremlin Declarative vs. Imperative traversal

Give this point in traversal a name

Declarative

```
g.V().match(
  as("a").has("name","gremlin"),
  as("a").out("created").as("b"),
  as("b").in("created").as("c"),
  as("c").in("manages").as("d"),
    where("a",neq("c"))).
  select("d").
  groupCount().by("name")
```

Imperative

```
g.V().has("name","gremlin").as("a").
  out("created").in("created").
    where(neq("a")).
  in("manages").
  groupCount().by("name")
```

What are the names of the projects created by two friends?

- ...there exists some "a" who knows "b".
- ...there exists some "a" who created "c".
- ...there exists some "b" who created "c".
- ...there exists some "c" created by 2 people.
- Get the name of all matching "c" projects.

- first places a traverser at the vertex denoting Gremlin
- That traverser then splits itself across all of Gremlin's collaborators that are not Gremlin himself
- Next, the traversers walk to the managers of those collaborators to ultimately be grouped into a manager name count distribution.

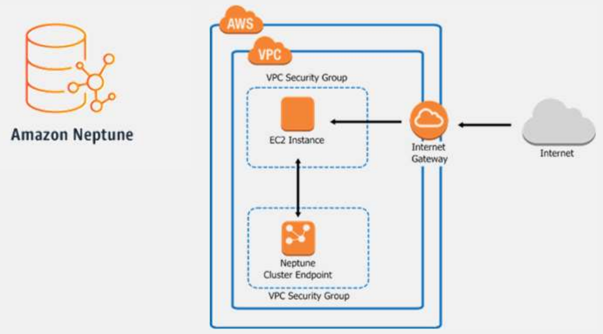
Cypher vs. Gremlin

- Gremlin has both imperative and declarative
  - At the core, Gremlin is a Groovy wrapper over underlying Java functions
  - In the imperative model, there is less room for abstraction, optimization and remoting, since you are effectively sending Groovy or Java code over the wire
  - You *can* write declarative statements
- Cypher is a declarative, non-turing complete language
  - Cypher is similar to SPARQL or SQL - a declarative, higher order description of WHAT you want, not HOW you want your results to be found
  - Well suited for remoting, standardization and optimization
- Interestingly, there is now a feature on Cypher to run on Gremlin

AWS Neptune

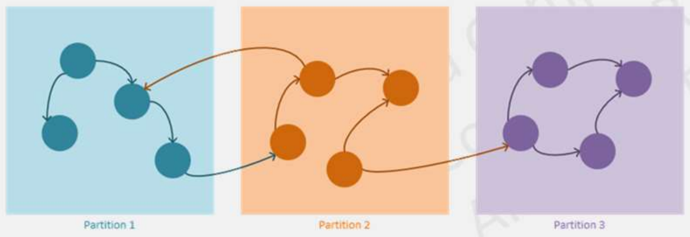
- OLTP
  - Property graph + TinkerPop Gremlin
  - RDF + SPARQL
  - each Neptune instance provides both a Gremlin Websocket Server and a SPARQL 1.1 Protocol REST endpoint
- Highly available, with read replicas, point-in-time recovery, continuous backup to Amazon S3, and replication across Availability Zones
  - replicates six copies of the data across three Availability Zones
  - instance failover typically takes less than 30 seconds.
- ACID Compliant
- HTTPS encrypted client connections and encryption at rest
- fully managed

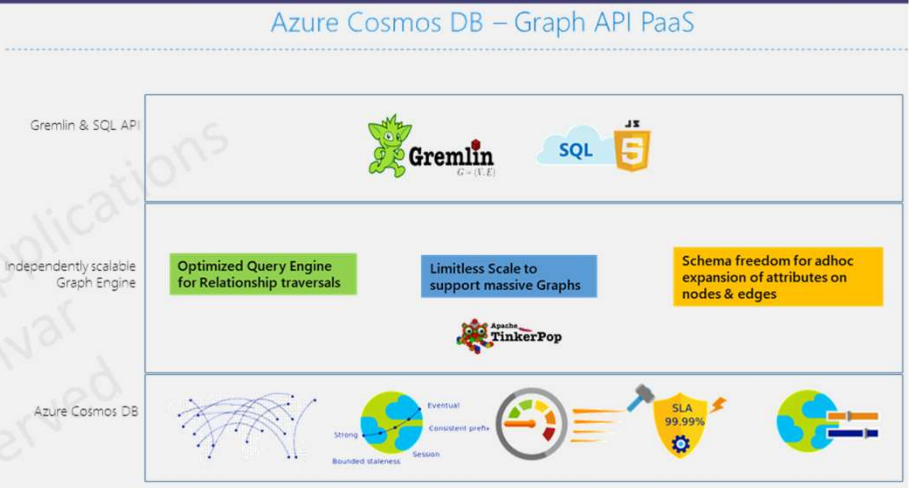
- *Possibly based on Blazegraph project?* ☺



Azure CosmosDB

- Gremlin API for queries
- Supports horizontally scalable graph databases
  - Graph partitioning
    - Vertices require a partition key
    - Edges will be stored with their source vertex
    - Edges contain references to the vertices they point to
    - Graph queries need to specify a partition key
      - g.V('vertex_id').has('partitionKey', 'partitionKey_value')




Azure Cosmos DB – Graph API PaaS
Gremlin & SQL API
Gremlin
SQL
Independently scalable Graph Engine
Optimized Query Engine for Relationship traversals
Limitless Scale to support massive Graphs
Schema freedom for adhoc expansion of attributes on nodes & edges
Apache TinkerPop
Azure Cosmos DB
Strong
Bounded staleness
Session
Consistent prefix
Eventual
SLA 99.99%

- Returns the results in GraphSON format