Introduction in Graph Databases and Neo4j

Stefan Armbruster  
@darthvader42  
stefan.armbruster@neotechnology.com

most slides from: 
Michael Hunger
The Path Forward

1. No .. NO .. NOSQL
2. Why graphs?
3. What's a graph database?
4. Some things about Neo4j.
5. How do people use Neo4j?
Trends in BigData & NOSQL

1. increasing data size (big data)
   • “Every 2 days we create as much information as we did up to 2003”
     - Eric Schmidt

2. increasingly connected data (graph data)
   • for example, text documents to html

3. semi-structured data
   • individualization of data, with common sub-set

4. architecture - a facade over multiple services
   • from monolithic to modular, distributed applications
NOSQL
NOSQL Databases

- Riak
- KeyValue
- Redis
- Couch
- Column oriented
- Cassandra
- Neo4j
- Relational
- MySQL
- Postgres
- Graph
- Mongo
Living in a NOSQL World

- RDBMS
- Key-Value Store
- Column Family
- Document Databases
- Graph Databases

Density ≈ Complexity

Volume ≈ Size
$\text{complexity} = f(\text{size}, \text{connectedness}, \text{uniformity})$
Big data is like teenage sex:
everyone talks about it,
nobody really knows how to do it,
everyone thinks everyone else is doing it, so everyone claims they are doing it...

(Dan Ariely)
DENORMALISE
Aggregate data into documents

RICHER MODEL
Connected structured data

Simple data model
Map-reduce friendly

Expressive power
Fast graph traversals
A Graph?
Yes, a graph
Leonhard Euler 1707-1783
They are everywhere
They are everywhere

http://www.bbc.co.uk/london/travel/downloads/tube_map.html
Graphs Everywhere

- Relationships in
  - Politics, Economics, History, Science, Transportation

- Biology, Chemistry, Physics, Sociology
  - Body, Ecosphere, Reaction, Interactions

- Internet
  - Hardware, Software, Interaction

- Social Networks
  - Family, Friends
  - Work, Communities
  - Neighbours, Cities, Society
Good Relationships

- the world is rich, messy and related data
- relationships are as least as important as the things they connect
- Graphs = Whole > Σ parts
- complex interactions
- always changing, change of structures as well
- Graph: Relationships are part of the data
- RDBMS: Relationships part of the fixed schema
Everyone is talking about graphs...
Everyone is talking about graphs...
Graph DB 101
A graph database...

NO: not for charts & diagrams, or vector artwork

YES: for storing data that is structured as a graph

    remember linked lists, trees?

    graphs are the general-purpose data structure

“A relational database may tell you the average age of everyone in this session,

but a graph database will tell you who is most likely to buy you a beer.”
You know relational

foo  foo_bar  bar
now consider relationships...
We're talking about a Property Graph

- Nodes
- Relationships
- Properties (each a key+value)
  - Indexes (for easy look-ups)
  - Labels (Neo4j 2.0)
Looks different, fine. Who cares?

- a sample social graph
  - with ~1,000 persons
- average 50 friends per person
- \textit{pathExists}(a,b) limited to depth 4
- caches warmed up to eliminate disk I/O

<table>
<thead>
<tr>
<th></th>
<th># persons</th>
<th>query time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational database</td>
<td>1.000</td>
<td>2000ms</td>
</tr>
<tr>
<td>Neo4j</td>
<td>1.000</td>
<td>2ms</td>
</tr>
<tr>
<td>Neo4j</td>
<td>1.000.000</td>
<td>2ms</td>
</tr>
</tbody>
</table>
Graph Database: Pros & Cons

◎ Strengths

• Powerful data model, as general as RDBMS
• Fast, for connected data
• Easy to query

◎ Weaknesses:

• Shardding (though they can scale reasonably well)
  ▶ also, stay tuned for developments here
• Requires conceptual shift
  ▶ though graph-like thinking becomes addictive
And, but, so how do you query this "graph" database?
Query a graph with a traversal

// then traverse to find results
start n=node:People(name = 'Andreas')
match (n)--()--(foaf) return foaf
Modeling for graphs
Adam

LOLCAT

SARAH

FRIEND OF

SHARED

LIKES

COMMENTED

Funny!
SHARED

LOL Cat

FRIEND_OF

ON

FUNNY

COMMENTED

LIKES

Adam

Sarah
Neo4j 2.0: Lables

Adam

Sarah

LOL Cat

FUNNY

LikES

FRIEND_OF

COMMENTED

SHARED

Photo

Person

Person

Person
Neo4j - the Graph Database
What's a Graph Database?

DEFINITION: A graph database is a database that uses graph structures with nodes, edges, and properties to represent and store information.

Graph

IS A

Neo4j

the world's leading graph database

MANAGES A

RECORDS DATA IN

CONNECT

HAVE

HAD

IDENTIFIES

PATHS

TRAVERSAL

NAVIGATES ORDER

MAPS FROM

INDEX

NODES

RELATIONSHIPS

HAVE

HAVE

PROPERTIES
Neo4j is a Graph Database

○ A Graph Database:
  • a schema-free Property Graph
  • perfect for complex, highly connected data

○ A Graph Database:
  • reliable with real ACID Transactions
  • fast with more than 1M traversals / second
  • Server with REST API, or Embeddable on the JVM
  • scale out for higher-performance reads with High-Availability
Whiteboard --> Data

// Cypher query - friend of a friend
start n=node(0)
match (n)--()--(foaf)
return foaf
Two Ways to Work with Neo4j

1. Embeddable on JVM
   - Java, JRuby, Scala...
   - Tomcat, Rails, Akka, etc.
   - great for testing
Show me some code, please

```java
GraphDatabaseService graphDb =
    new EmbeddedGraphDatabase("var/neo4j");
Transaction tx = graphDb.beginTx();
try {
    Node steve = graphDb.createNode();
    Node michael = graphDb.createNode();

    steve.setProperty("name", "Steve Vinoski");
    michael.setProperty("name", "Michael Hunger");

    Relationship presentedWith = steve.createRelationshipTo(
        michael, PresentationTypes.PRESENTED_WITH);
    presentedWith.setProperty("date", today);
    tx.success();
} finally {
    tx.finish();
}
```
@NodeEntity
public class Movie {
    @Indexed private String title;
    @RelatedToVia(type = "ACTS_IN", direction=INCOMING) private Set<Role> cast;
    private Director director;
}

@NodeEntity
public class Actor {
    @RelatedTo(type = "ACTS_IN")
    private Set<Movies> movies;
}

@RelationshipEntity
public class Role {
    @StartNode private Actor actor;
    @EndNode private Movie movie;
    private String roleName;
}
Cypher Query Language

- **Declarative query language**
  - *Describe what you want, not how*
  - *Based on pattern matching*

- **Examples:**

```
START david=node:people(name=’David’)  # index lookup
MATCH david[:knows]-friends[:knows]-new_friends
WHERE new_friends.age > 18
RETURN new_friends

START user=node(5, 15, 26, 28)  # node IDs
MATCH user--friend
RETURN user, COUNT(friend), SUM(friend.money)
```
CREATE
(steve {name: "Steve Vinoski"})
-[:PRESENTED_WITH {date:{day}}]->
(michael {name: "Michael Hunger"})
Two Ways to Work with Neo4j

2. Server with REST API

- every language on the planet
- flexible deployment scenarios
- DIY server, or cloud managed
Bindings

REST://

SpringDataNeo4j

Erlang

Microsoft .NET

JavaScript

Gremlin

PHP

NodeJS
Two Ways to Work with Neo4j

- Server capability == Embedded capability
  - same scalability, transactionality, and availability
the Real World
• Cisco.com serves customer and business customers with Support Services
• Needed real-time recommendations, to encourage use of online knowledge base
• Cisco had been successfully using Neo4j for its internal master data management solution.
• Identified a strong fit for online recommendations

• Call center volumes needed to be lowered by improving the efficacy of online self service
• Leverage large amounts of knowledge stored in service cases, solutions, articles, forums, etc.
• Problem resolution times, as well as support costs, needed to be lowered

• Cases, solutions, articles, etc. continuously scraped for cross-reference links, and represented in Neo4j
• Real-time reading recommendations via Neo4j
• Neo4j Enterprise with HA cluster
• The result: customers obtain help faster, with decreased reliance on customer support
San Jose, CA
Cisco HMP

Industry: Communications Use case: Master Data Management

• One of the world’s largest communications equipment manufacturers#91 Global 2000. $44B in annual sales.
• Needed a system that could accommodate its master data hierarchies in a performant way
• HMP is a Master Data Management system at whose heart is Neo4j. Data access services available 24x7 to applications companywide

• Sales compensation system had become unable to meet Cisco’s needs
• Existing Oracle RAC system had reached its limits:
  • Insufficient flexibility for handling complex organizational hierarchies and mappings
  • “Real-time” queries were taking > 1 minute!
• Business-critical “P1” system needs to be continually available, with zero downtime

• Cisco created a new system: the Hierarchy Management Platform (HMP)
• Allows Cisco to manage master data centrally, and centralize data access and business rules
• Neo4j provided “Minutes to Milliseconds” performance over Oracle RAC, serving master data in real time
• The graph database model provided exactly the flexibility needed to support Cisco’s business rules
• HMP so successful that it has expanded to include product hierarchy
Industry: Logistics  Use case: Parcel Routing

- One of the world’s largest logistics carriers
- Projected to outgrow capacity of old system
- New parcel routing system
- Single source of truth for entire network
- B2C & B2B parcel tracking
- Real-time routing: up to 5M parcels per day

- 24x7 availability, year round
- Peak loads of 2500+ parcels per second
- Complex and diverse software stack
- Need predictable performance & linear scalability
- Daily changes to logistics network: route from any point, to any point

- Neo4j provides the ideal domain fit:
- A logistics network is a graph
- Extreme availability & performance with Neo4j clustering
- Hugely simplified queries, vs. relational for complex routing
- Flexible data model can reflect real-world data variance much better than relational
- “Whiteboard friendly” model easy to understand
• Online jobs and career community, providing anonymized inside information to job seekers

• Wanted to leverage known fact that most jobs are found through personal & professional connections

• Needed to rely on an existing source of social network data. Facebook was the ideal choice.

• End users needed to get instant gratification

• Aiming to have the best job search service, in a very competitive market

• First-to-market with a product that let users find jobs through their network of Facebook friends

• Job recommendations served real-time from Neo4j

• Individual Facebook graphs imported real-time into Neo4j

• Glassdoor now stores > 50% of the entire Facebook social graph

• Neo4j cluster has grown seamlessly, with new instances being brought online as graph size and load have increased
Industry: Communications

Use case: Network Management

- Second largest communications company in France
- Part of Vivendi Group, partnering with Vodafone

- Infrastructure maintenance took one full week to plan, because of the need to model network impacts
- Needed rapid, automated “what if” analysis to ensure resilience during unplanned network outages. Identify weaknesses in the network to uncover the need for additional redundancy
- Network information spread across > 30 systems, with daily changes to network infrastructure. Business needs sometimes changed very rapidly

- Flexible network inventory management system, to support modeling, aggregation & troubleshooting
- Single source of truth (Neo4j) representing the entire network
- Dynamic system loads data from 30+ systems, and allows new applications to access network data
- Modeling efforts greatly reduced because of the near 1:1 mapping between the real world and the graph
- Flexible schema highly adaptable to changing business requirements
Industry: Web/ISV, Communications  Use case: Network Management

Hewlett Packard

- World’s largest provider of IT infrastructure, software & services
- HP’s Unified Correlation Analyzer (UCA) application is a key application inside HP’s OSS Assurance portfolio
- Carrier-class resource & service management, problem determination, root cause & service impact analysis
- Helps communications operators manage large, complex and fast changing networks

- Use network topology information to identify root problems causes on the network
- Simplify alarm handling by human operators
- Automate handling of certain types of alarms
- Help operators respond rapidly to network issues
- Filter/group/eliminate redundant Network Management System alarms by event correlation

- Accelerated product development time
- Extremely fast querying of network topology
- Graph representation a perfect domain fit
- 24x7 carrier-grade reliability with Neo4j HA clustering
- Met objective in under 6 months
Industry: Communications
Use case: Resource Authorization & Access Control

- 10th largest Telco provider in the world, leading in the Nordics
- Online self-serve system where large business admins manage employee subscriptions and plans
- Mission-critical system whose availability and responsiveness is critical to customer satisfaction

- Degrading relational performance. User login taking minutes while system retrieved access rights
- Millions of plans, customers, admins, groups. Highly interconnected data set w/massive joins
- Nightly batch workaround solved the performance problem, but meant data was no longer current
- Primary system was Sybase. Batch pre-compute workaround projected to reach 9 hours by 2014: longer than the nightly batch window

- Moved authorization functionality from Sybase to Neo4j
- Modeling the resource graph in Neo4j was straightforward, as the domain is inherently a graph
- Able to retire the batch process, and move to real-time responses: measured in milliseconds
- Users able to see fresh data, not yesterday’s snapshot
- Customer retention risks fully mitigated
“Business Service Management” requires mapping of complex graph, covering: business processes→ business services→ IT infrastructure

Embed capability of storing and retrieving this information into OEM application

Re-architecting outdated C++ application based on relational database, with Java

Actively sought out a Java-based solution that could store data as a graph

Domain model is reflected directly in the database: “No time lost in translation”

“Our business and enterprise consultants now speak the same language, and can model the domain with the database on a 1:1 ratio.”

Spring Data Neo4j strong fit for Java architecture
Teachscape, Inc. develops online learning tools for K-12 teachers, school principals, and other instructional leaders. Teachscape evaluated relational as an option, considering MySQL and Oracle. Neo4j was selected because the graph data model provides a more natural fit for managing organizational hierarchy and access to assets.

Neo4j was selected to be at the heart of a new architecture. The user management system, centered around Neo4j, will be used to support single sign-on, user management, contract management, and end-user access to their subscription entitlements.

- **Domain and technology fit** simple domain model where the relationships are relatively complex.
- **Secondary factors** included support for transactions, strong Java support, and well-implemented Lucene indexing integration.
- **Speed and Flexibility** The business depends on being able to do complex walks quickly and efficiently. This was a major factor in the decision to use Neo4j.
- **Ease of Use** accommodate efficient access for home-grown and commercial off-the-shelf applications, as well as ad-hoc use.
- **Extreme availability & performance with Neo4j clustering**
- **Hugely simplified queries, vs. relational for complex routing**
- **Flexible data model can reflect real-world data variance much better than relational**
- “Whiteboard friendly” model easy to understand
Really, once you start thinking in graphs it's hard to stop

What will you build?

Geospatial
access control
Biotechnology
linguistics
compensation data

Business intelligence
catalogs
Social computing Management
your brain
routing
Making Sense of all that

genealogy
market vectors
Get a free book

visit us at our booth downstairs

tell us your graphy ideas

leave your card

… or grab your free pdf version at

http://www.graphdatabases.com

visit http://www.neo4j.org and http://www.neotechnology.com
Cypher
a pattern-matching query language for graphs
Cypher - overview

- a pattern-matching query language
- declarative grammar with clauses (like SQL)
- aggregation, ordering, limits
- create, update, delete
Cypher: START +

- START <lookup> RETURN <expressions>
- START binds terms using simple look-up
  - directly using known ids
  - or based on indexed Property
- RETURN expressions specify result set

```cypher
// lookup node id 0, return that node
start n=node(0) return n

// lookup node in Index, return that node
start n=node:Person(name="Andreas") return n

// lookup all nodes, return all name properties
start n=node(*) return n.name
```
Cypher: MATCH

- START <lookup> MATCH <pattern> RETURN <expr>

- MATCH describes a pattern of nodes+relationships
  - node terms in optional parenthesis
  - lines with arrows for relationships

// lookup 'n', traverse any relationship to some 'm'
start n=node(0) match (n)--(m) return n,m
// any outgoing relationship from 'n' to 'm'
start n=node(0) match n-->m return n,m
// only 'KNOWS' relationships from 'n' to 'm'
start n=node(0) match n-[:KNOWS]->m return n,m
// from 'n' to 'm' and capture the relationship as 'r'
start n=node(0) match n-[r]->m return n,r,m
// from 'n' outgoing to 'm', then incoming from 'o'
start n=node(0) match n-->m<--o return n,m,o
Cypher: WHERE

- START <lookup> [MATCH <pattern>]
- WHERE <condition> RETURN <expr>
- WHERE filters nodes or relationships
  - uses expressions to constrain elements

// lookup all nodes as 'n', constrained to name 'Andreas'
start n=node(*) where n.name='Andreas' return n
// filter nodes where age is less than 30
start n=node(*) where n.age<30 return n
// filter using a regular expression
start n=node(*) where n.name =~ /Tob.*/ return n
// filter for a property exists
start n=node(*) where has(n.name) return n
Cypher: CREATE

CREATE <node>[,node or relationship] RETURN <expr>

- create nodes with optional properties
- create relationship (must have a type)

// create an anonymous node
create n

// create node with a property, returning it
create n={name:'Andreas'} return n

// lookup 2 nodes, then create a relationship and return it
start n=node(0),m=node(1) create n-[r:KNOWS]-m return r

// lookup nodes, then create a relationship with properties
start n=node(1),m=node(2) create n-[r:KNOWS {since:2008}]->m
Cypher: SET

- SET [<node property>] [<relationship property>]  
  - update a property on a node or relationship  
  - must follow a START

// update the name property
start n=node(0) set n.name='Peter'
// update many nodes, using a calculation
start n=node(*) set n.size=n.size+1
// match & capture a relationship, update a property
start n=node(1) match n-[r]-m set r.times=10
Cypher: DELETE

DELETE [<node>|<relationship>|<property>]

- delete a node, relationship or property
- must follow a START
- to delete a node, all relationships must be deleted first

// delete a node
start n=node(5) delete n

// remove a node and all relationships
start n=node(3) match n-[r]-() delete n, r

// remove a property
start n=node(3) delete n.age
You can modify and query this graph by entering statements in the input field at the bottom. For some syntax help hit the button. If you want to share your graph, just do it with.

```
start n=node(*)
match n-[r]-m
return n,r
```

<table>
<thead>
<tr>
<th>n</th>
<th>r</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node[0]{name-&gt;&quot;REPL&quot;}</td>
<td>:TRY[0] {}</td>
</tr>
<tr>
<td>Node[0]{name-&gt;&quot;REPL&quot;}</td>
<td>:BUILD[2] {}</td>
</tr>
<tr>
<td>Node[1]{name-&gt;&quot;GIST&quot;}</td>
<td>:SHARE[1] {}</td>
</tr>
<tr>
<td>Node[1]{name-&gt;&quot;GIST&quot;}</td>
<td>:BUILD[2] {}</td>
</tr>
<tr>
<td>Node[2]{name-&gt;&quot;SandBox&quot;}</td>
<td>:TRY[0] {}</td>
</tr>
<tr>
<td>Node[2]{name-&gt;&quot;SandBox&quot;}</td>
<td>:SHARE[1] {}</td>
</tr>
</tbody>
</table>

6 rows
0 ms

```
start n=node(*) match n-[r]-m return n,r
```

[http://console.neo4j.org](http://console.neo4j.org)

[This Graph: http://tinyurl.com/7cnvmlq](http://tinyurl.com/7cnvmlq)
High Availability
Scaling on a single server

- data size can increase into the billions
- however
  - performance relies on memory caches
  - server must be taken offline for backups
  - single point of failure
- For 24x7 production, it's time to introduce HA
High Availability

- **master-slave replication**
  - read/write to any cluster member
  - slave writes commit to master first (redundancy)
  - master writes are faster
  - all writes propagate to slaves (polling interval)
High Availability

- **automatic fail-over**
  - any cluster member can be elected master
  - on failure, a new master will be automatically elected
  - a failed master can re-join as a slave
  - automatic branch detection & resolution
**High Availability**

◎ **online backups**

- *backup pulls updates directly from live cluster*
- *backup is a full, directly useable Neo4j database*

◎ **to restore: shutdown cluster, distribute backup, restart**
3 Lessons Learned
I. Healthy Relationships

replace many-to-many join tables...

...with a relationship in the graph
2. Property Lists

- Don’t try to embed multiple values into a single property
- That makes it harder to traverse using these values

name: “Canada”
languages_spoken: “[ ‘English’, ‘French’ ]”

- Instead, extract “list” values into separate nodes
3. One Concept Per Node

Don’t bundle multiple concepts

Instead, break out the separate concepts...
3 Lessons Learned

- Use Relationships
- Use Relationships
- Use Relationships