CC5212-1
PROCESAMIENTO MASIVO DE DATOS
OTOÑO 2015

Lecture 10: NoSQL II

Aidan Hogan
aidhog@gmail.com

RECAP: NOSQL

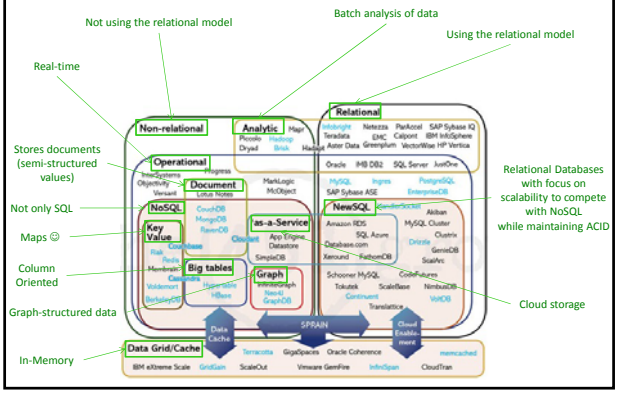
NoSQL

NoSQL vs. Relational Databases

What are the big differences between relational databases and NoSQL systems?

What are the trade-offs?

The Database Landscape



RECAP: KEY–VALUE

Key–Value = a Distributed Map

Key	Value
country:Afghanistan	capital@city:Kabul,continent:Asia,pop:31108077#2011
country:Albania	capital@city:Tirana,continent:Europe,pop:3011405#2013
…	…
city:Kabul	country:Afghanistan,pop:3476000#2013
city:Tirana	country:Albania,pop:3011405#2013
…	…
user:10239	basedIn@city:Tirana,post:{103,10430,201}
…	…

Amazon Dynamo(DB): Model

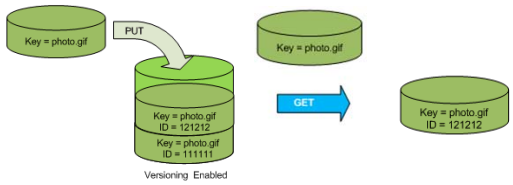
- Named table with primary key and a value

Countries	
Primary Key	**Value**
Afghanistan	capital:Kabul,continent:Asia,pop:31108077#2011
Albania	capital:Tirana,continent:Europe,pop:3011405#2013
…	…

Cities	
Primary Key	**Value**
Kabul	country:Afghanistan,pop:3476000#2013
Tirana	country:Albania,pop:3011405#2013
…	…

Amazon Dynamo(DB): Object Versioning

- Object Versioning (per bucket)
  - PUT doesn't overwrite: pushes version
  - GET returns most recent version



Other Key–Value Stores

riak

redis

RECAP: DOCUMENT STORES

Key–Value Stores: Values are Documents

Key	Value
country:Afghanistan	<country> <capital>city:Kabul</capital> <continent>Asia</continent> <population> <value>31108077</value> <year>2011</year> </population> </country>
…	…

- Document-type depends on store
  - XML, JSON, Blobs, Natural language
- Operators for documents
  - e.g., filtering, inv. indexing, XML/JSON querying, etc.

MongoDB: JSON Based

Key	Value (Document)
6ads786a5a9	{ "_id" : **ObjectId**("6ads786a5a9") , "name" : "Afghanistan" , "capital": "Kabul" , "continent" : "Asia" , "population" : { "value" : 31108077, "year" : 2011 } }
…	…

- Can invoke Javascript over the JSON objects
- Document fields can be indexed

```
db.inventory.find({ continent: { $in: [ 'Asia', 'Europe' ]}})
```

Document Stores

CouchDB relax

Couchbase

mongoDB

TABLULAR / COLUMN FAMILY

Key–Value = a Distributed Map

Countries	
Primary Key	**Value**
Afghanistan	capital:Kabul,continent:Asia,pop:31108077#2011
Albania	capital:Tirana,continent:Europe,pop:3011405#2013
…	…

Tabular = Multi-dimensional Maps

Countries				
Primary Key	**capital**	**continent**	**pop-value**	**pop-year**
Afghanistan	Kabul	Asia	31108077	2011
Albania	Tirana	Europe	3011405	2013
…	…	…	…	…

Bigtable: The Original Whitepaper

Why did they write another paper?
MapReduce solves everything, right?

MapReduce authors

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com

Google, Inc.

Abstract

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully provided a flexible, high-performance solution for all of these Google products. In this paper we describe the simple data model provided by Bigtable, which gives clients dynamic control over data layout and format, and we describe the design and implementation of Bigtable.

achieved scalability and high performance, but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Data is indexed using row and column names that can be arbitrary strings. Bigtable also treats data as uninterpreted strings, although clients often serialize various forms of structured and semi-structured data into these strings. Clients can control the locality of their data through careful choices in their schemas. Finally, the Bigtable schema parameters let clients dynamically control whether to serve data out of memory or from disk.

Section describes the data model in more detail, and Section gives an overview of the client API. Sec-

Google

Bigtable used for …

…

Bigtable: Data Model

"*a sparse, distributed, persistent, multi-dimensional, sorted map*."

- *sparse*: not all values form a dense square
- *distributed*: lots of machines
- *persistent*: disk storage (GFS)
- *multi-dimensional*: values with columns
- *sorted*: sorting lexicographically by row key
- *map*: look up a key, get a value

Bigtable: in a nutshell

(row, column, time) → value

- row: a row id string
  - e.g., "`Afganistan`"
- column: a column name string
  - e.g., "`pop-value`"
- time: an integer (64-bit) version time-stamp
  - e.g., `18545664`
- value: the element of the cell
  - e.g., "`31120978`"

Bigtable: in a nutshell

(row, column, time) → value

(`Afganistan`, `pop-value`, t_4) → `31108077`

Primary Key	capital		continent		pop-value		pop-year	
Afghanistan	t_1	Kabul	t_1	Asia	t_1	31143292	t_1	2009
					t_2	31120978		
					t_4	31108077	t_4	2011
Albania	t_1	Tirana	t_1	Europe	t_1	2912380	t_1	2010
					t_3	3011405	t_3	2013
...	...		...		...		...	

Bigtable: Sorted Keys

SORTED

Primary Key	capital		pop-value		pop-year	
Asia:Afghanistan	t_1	Kabul	t_1	31143292	t_1	2009
			t_2	31120978		
			t_4	31108077	t_4	2011
Asia:Azerbaijan	...	...	...	...	...	...
...	...	...	...	...	...	...
Europe:Albania	t_1	Tirana	t_1	2912380	t_1	2010
			t_3	3011405	t_3	2013
Europe:Andorra	...	...	...	...	...	...
...	...	...	...	...	...	...

Benefits of sorted keys vs. hashed keys?

Bigtable: Tablets

ASIA / EUROPE

Primary Key	capital		pop-value		pop-year	
Asia:Afghanistan	t_1	Kabul	t_1	31143292	t_1	2009
			t_2	31120978		
			t_4	31108077	t_4	2011
Asia:Azerbaijan	...	...	...	...	...	...
...	...	...	...	...	...	...
Europe:Albania	t_1	Tirana	t_1	2912380	t_1	2010
			t_3	3011405	t_3	2013
Europe:Andorra	...	...	...	...	...	...
...	...	...	...	...	...	...

Take advantage of locality of processing!

Bigtable: Distribution

Pros and cons versus hash partitioning?

***Horizontal** range partitioning*

Bigtable: Column Families

Primary Key	pol:capital		demo:pop-value		demo:pop-year	
Asia:Afghanistan	t_1	Kabul	t_1	31143292	t_1	2009
			t_2	31120978		
			t_4	31108077	t_4	2011
Asia:Azerbaijan	...	...	...	...	...	...
...	...	...	...	...	...	...
Europe:Albania	t_1	Tirana	t_1	2912380	t_1	2010
			t_3	3011405	t_3	2013
Europe:Andorra	...	...	...	...	...	...
...	...	...	...	...	...	...

- Group logically similar columns together
  - Accessed efficiently together
  - Access-control and storage: column family level
  - If of same type, can be compressed

Bigtable: Versioning

- Similar to Apache Dynamo (so no "fancy" slide)
  - Cell-level
  - 64-bit integer time stamps
  - Inserts push down current version
  - Lazy deletions / periodic garbage collection
  - Two options:
    - keep last *n* versions
    - keep versions newer than *t* time

Bigtable: SSTable Map Implementation

- 64k blocks (default) with index in footer (GFS)
- Index loaded into memory, allows for seeks
- Can be split or merged, as needed

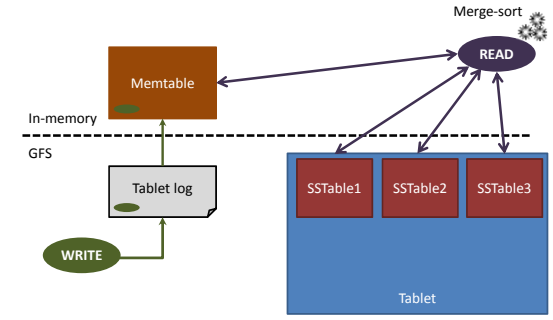
How to handle writes?

	Primary Key	pol:capital		demo:pop-value		demo:pop-year	
0	Asia:Afghanistan	t_1	Kabul	t_1	31143292	t_1	2009
				t_2	31120978		
				t_4	31108077	t_4	2011
	Asia:Azerbaijan	...	...	...	...	...	...
	...	...	...	...	...	...	...
65536	Asia:Japan	...	...	...	...	...	...
	Asia:Jordan	...	...	...	...	...	...
	...	...	...	...	...	...	...

Index:
- Block 0 / Offset 0 / Asia:Afghanistan
- Block 1 / Offset 65536 / Asia: Japan

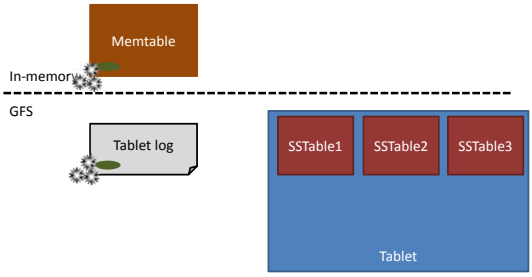
Bigtable: Buffered/Batched Writes

What's the danger here?



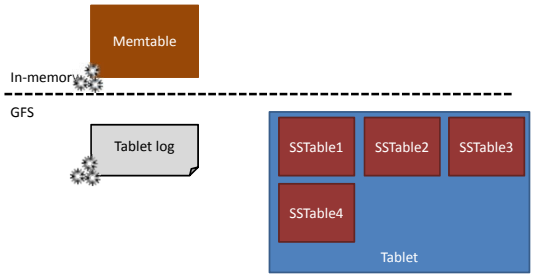
Bigtable: Redo Log

- If machine fails, Memtable redone from log



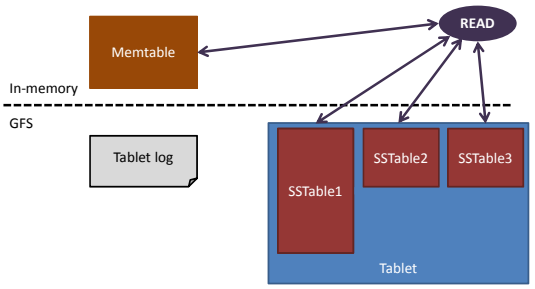
Bigtable: **Minor** Compaction

- When full, write Memtable as SSTable



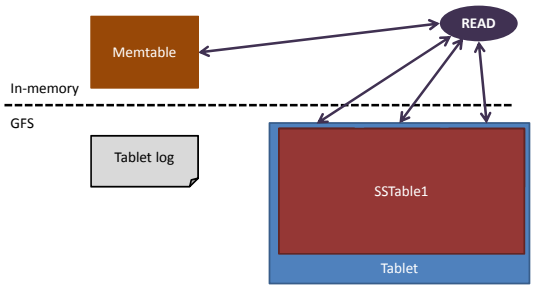
Bigtable: **Merge** Compaction

- Merge some of the SSTables (and the Memtable)

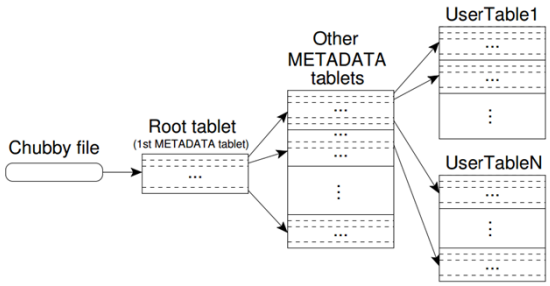


Bigtable: **Major** Compaction

- Merge all SSTables (and the Memtable)
- Makes reads more efficient!



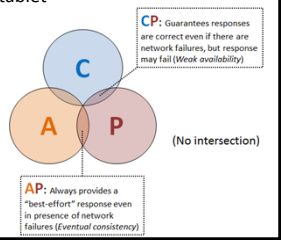
Bigtable: Hierarchical Structure



Bigtable: Consistency

- CHUBBY: Distributed consensus tool based on PAXOS
  - Maintains consistent replicas
    - Five replicas: one master and four slaves
  - Co-ordinates distributed locks
  - Stores location of main "root tablet"

Do we think it's a CP system or an AP system?



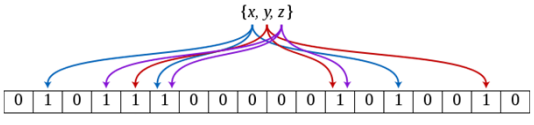
Bigtable: A Bunch of Other Things

- Locality groups: Group multiple column families together; assigned a separate SSTable
- Select storage: SSTables can be persistent or in-memory
- Compression: Applied on SSTable blocks; custom compression can be chosen
- Caches: SSTable-level and block-level
- Bloom filters: Find negatives cheaply …

Aside: Bloom Filter

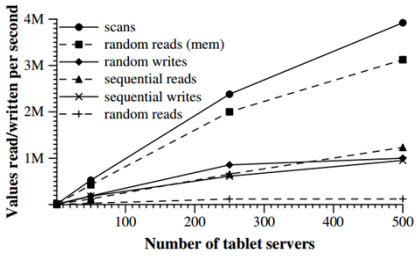
Reject "empty" queries using very little memory!

- Create a bit array of length m (init to 0's)
- Create k hash functions that map an object to an index of m (even distribution)
- Index o: set $m[\text{hash}_1(o)]$, …, $m[\text{hash}_k(o)]$ to 1
- Query o:
  - **any** $m[\text{hash}_1(o)]$, …, $m[\text{hash}_k(o)]$ set to **0** = not indexed
  - **all** $m[\text{hash}_1(o)]$, …, $m[\text{hash}_k(o)]$ set to **1** = **might** be indexed



Bigtable: an idea of performance

- Values are 1 kilobyte in size
- Results from **2006** paper



Why are random (disk) reads so slow?

The read sizes are 1 kb, but a different 64 kb block must be sent over the network (almost) every time

Bigtable: an idea of performance

- Values are 1 kilobyte in size
- Results from **2006** paper
- Average values/second per server:

Experiment	# of Tablet Servers			
	1	50	250	500
random reads	1212	593	479	241
random reads (mem)	10811	8511	8000	6250
random writes	8850	3745	3425	2000
sequential reads	4425	2463	2625	2469
sequential writes	8547	3623	2451	1905
scans	15385	10526	9524	7843

- Adding more machines add a cost!
- But overall performance does increase

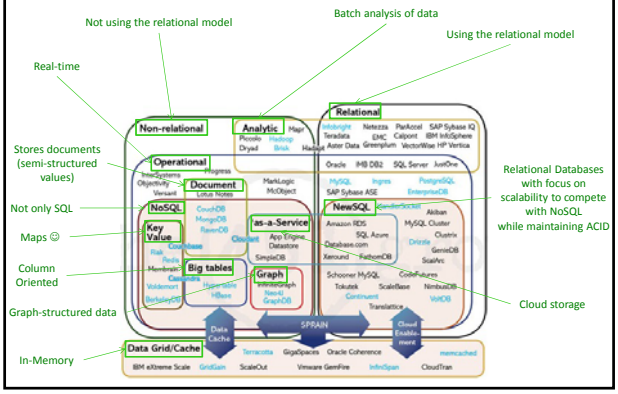
Bigtable: examples in Google (2006)

Project name	Table size (TB)	Compression ratio	# Cells (billions)	# Column Families	# Locality Groups	% in memory	Latency-sensitive?
Crawl	800	11%	1000	16	8	0%	No
Crawl	50	33%	200	2	2	0%	No
Google Analytics	20	29%	10	1	1	0%	Yes
Google Analytics	200	14%	80	1	1	0%	Yes
Google Base	2	31%	10	29	3	15%	Yes
Google Earth	0.5	64%	8	7	2	33%	Yes
Google Earth	70	–	9	8	3	0%	No
Orkut	9	–	0.9	8	5	1%	Yes
Personalized Search	4	47%	6	93	11	5%	Yes

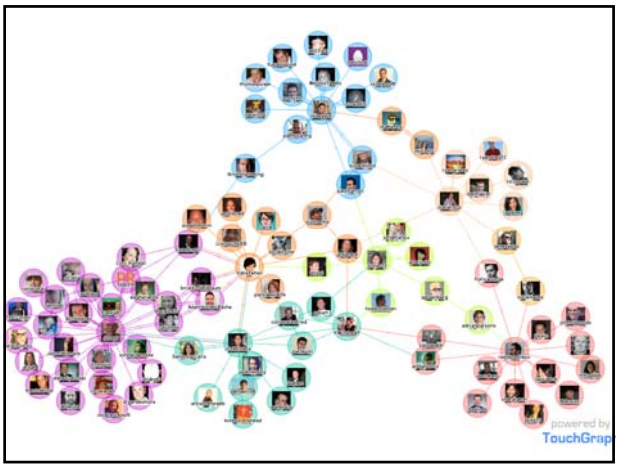
Bigtable: Apache HBase

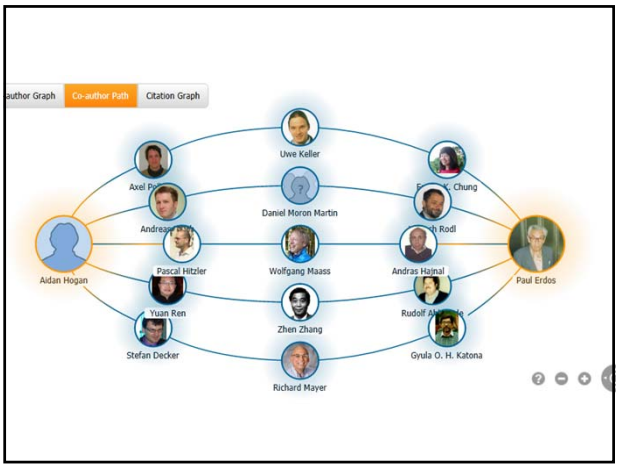
Open-source implementation of Bigtable ideas

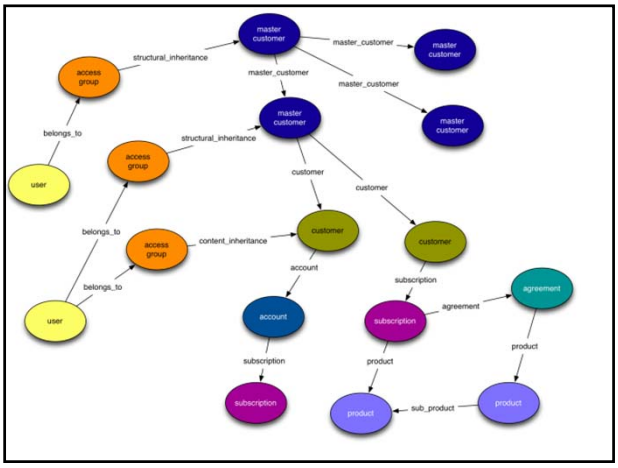
The Database Landscape

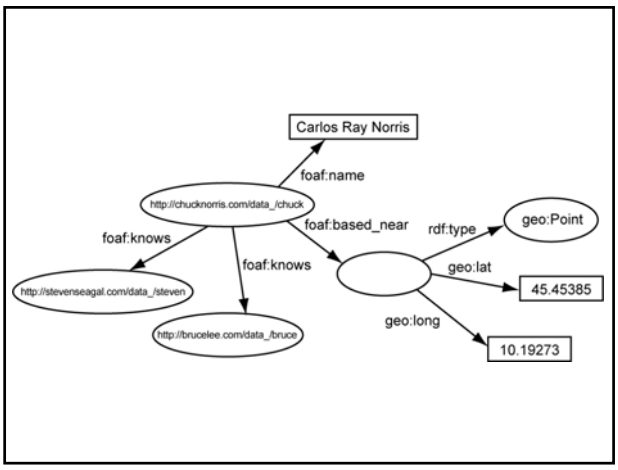


GRAPH DATABASES









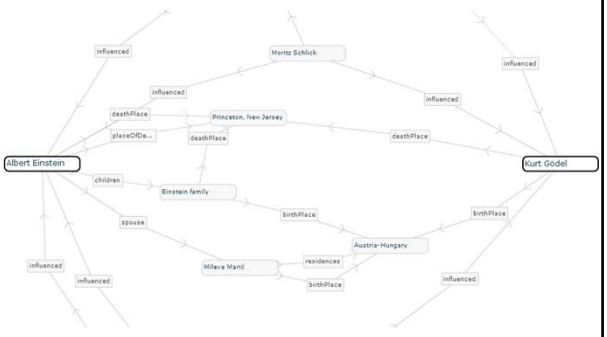
Data = Graph

- Any data can be represented as a directed labelled graph (not always neatly)]

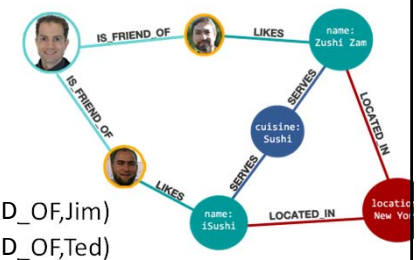
When is it a good idea to consider data as a graph?

When you want to answer questions like:
- How many social hops is this user away?
- What is my Erdős number?
- What connections are needed to fly to Perth?
- How are Einstein and Godel related?

RelFinder

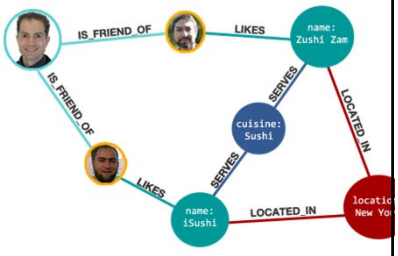


Graph Databases



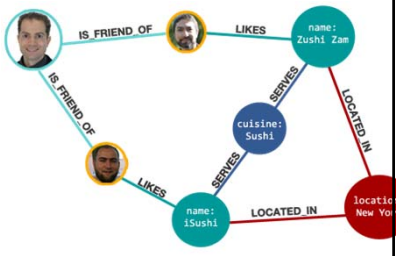
(Fred,IS_FRIEND_OF,Jim)
(Fred,IS_FRIEND_OF,Ted)
(Ted,LIKES,Zushi_Zam)
(Zuzhi_Zam,SERVES,Sushi)
…

Graph Databases: Index Nodes



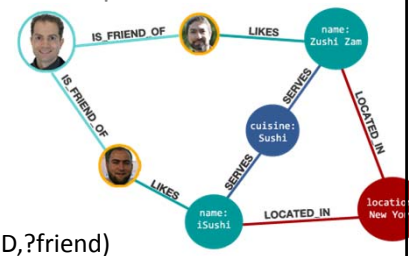
Fred -> (Fred,IS_FRIEND_OF,Jim)
(Jim,LIKES,iSushi)

Graph Databases: Index Relations



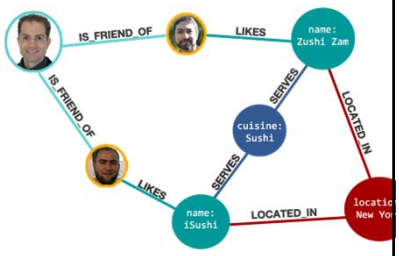
LIKES -> (Ted,LIKES,Zushi_Zam)
(Jim,LIKES,iSushi)

Graph Databases: Graph Queries



(Fred,IS_FRIEND,?friend)
(?friend,LIKES,?place)
(?place,SERVES,?sushi)
(?place,LOCATED_IN,New_York)

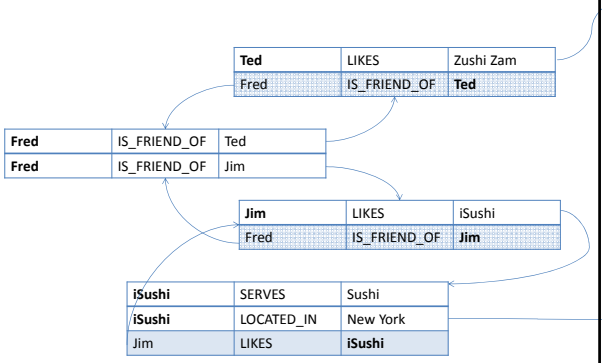
Graph Databases: Path Queries



What about scalability?

(Fred,IS_FRIEND*,?friend_of_friend)
(?friend_of_friend,LIKES,Zushi_Zam)

Graph Database: Index-free Adjacency



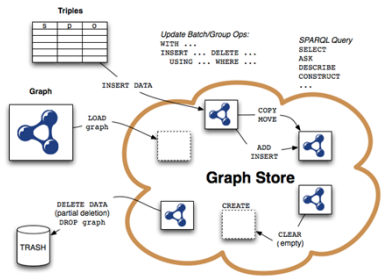
Leading Graph Database

Neo4j

Rank	Last Month	DBMS	Database Model	Score	Changes
1.	1.	Oracle	Relational DBMS	1502.74	-11.34
2.	2.	MySQL	Relational DBMS	1309.10	+16.43
3.	3.	Microsoft SQL Server	Relational DBMS	1207.80	-2.63
4.	4.	PostgreSQL	Relational DBMS	240.64	+10.41
5.	5.	MongoDB	Document store	224.62	+10.28
6.	6.	DB2	Relational DBMS	186.47	+1.89
7.	7.	Microsoft Access	Relational DBMS	145.36	+2.60
8.	8.	SQLite	Relational DBMS	89.29	-0.88
9.	9.	Cassandra	Wide column store	81.73	+3.01
10.	10.	Sybase ASE	Relational DBMS	80.00	+1.87
11.	11.	Solr	Search engine	67.16	+4.28
12.	12.	Teradata	Relational DBMS	65.53	+3.80
13.	13.	Redis	Key-value store	62.04	+3.58
14.	14.	FileMaker	Relational DBMS	55.59	+1.21
15.	16.	HBase	Wide column store	40.27	+3.66
16.	15.	Informix	Relational DBMS	36.51	-0.20
17.	19.	Elasticsearch	Search engine	32.06	+2.27
18.	17.	Hive	Relational DBMS	31.76	+0.74
19.	18.	Memcached	Key-value store	31.74	+0.76
20.	21.	Splunk	Search engine	24.68	+2.17
21.	20.	CouchDB	Document store	22.85	+0.30
22.	22.	Neo4j	Graph DBMS	21.46	+0.91
23.	23.	SAP HANA	Relational DBMS	20.19	+2.17

http://db-engines.com/en/ranking

SPARQL



Rank	Last Month	DBMS	Database Model	Score	Changes
1.	1.	Oracle	Relational DBMS	1502.74	-11.34
2.	2.	MySQL	Relational DBMS	1309.10	+16.43
3.	3.	Microsoft SQL Server	Relational DBMS	1207.80	-2.63
4.	4.	PostgreSQL	Relational DBMS	240.64	+10.41
5.	5.	MongoDB	Document store	224.62	+10.28
6.	6.	DB2	Relational DBMS	186.47	+1.89
7.	7.	Microsoft Access	Relational DBMS	145.36	+2.60
8.	8.	SQLite	Relational DBMS	89.29	-0.88
9.	9.	Cassandra	Wide column store	81.73	+3.01
10.	10.	Sybase ASE	Relational DBMS	80.00	+1.87
11.	11.	Solr	Search engine	67.16	+4.28
12.	12.	Teradata			
13.	13.	Redis			
14.	14.	FileMaker			
15.	16.	HBase			
16.	15.	Informix			
17.	19.	Elasticsearch			
18.	17.	Hive			
19.	18.	Memcached			
20.	21.	Splunk			
21.	20.	CouchDB			
22.	22.	Neo4j			
23.	23.	SAP HANA			

http://db-engines.com/en/ranking

RECAP

Recap

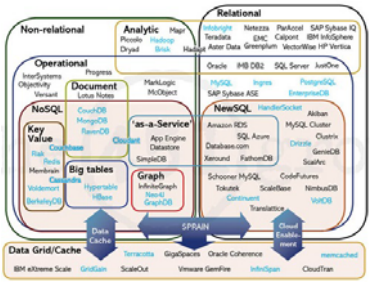
- Relational Databases don't solve everything
  - SQL and ACID add overhead
  - Distribution not so easy
- NoSQL: what if you don't need SQL or ACID?
  - Something simpler
  - Something more scalable
  - Trade efficiency against guarantees

NoSQL: Trade-offs

- **Simplified transactions** (no ACID)
- **Simplified** (or no) **query language**
  - Procedural *or* a subset of SQL
- **Simplified query alegbra**
  - Often no joins
- **Simplified data model**
  - Often map-based
- **Simplified replication**
  - Consistency vs. Availability

Simplifications enable scale to thousands of machines. But a lot of relational database features are lost!

NoSQL Overview Map



Types of NoSQL Store

- **Key–Value Stores** (e.g., Dynamo):
  - Distributed unsorted maps
  - Some have secondary indexes
- **Document Stores** (e.g., MongoDB):
  - Map values are documents (e.g., JSON, XML)
  - Built-in document functions/indexable fields
- **Table/Column-Based Stores** (e.g., Bigtable):
  - Distributed multi-dimensional sorted maps
  - Distribution by Tablets/Column-families
- **Graph Stores** (e.g., Neo4J)
  - Stores vertices and relations: Index-free adjacency
  - Query languages for paths, reachability, etc.
- **Hybrid/mix/other** (e.g., Cassandra)

Categories are far from clean: aside from graph stores, most NoSQL stores are just fancy (sometimes sorted) maps basically. ☺

Bigtable

- Column family store: (row, column, time) → value
- Sorted map, range partitioned
- PAXOS for locks, root table
- Tablets: horizontal table splits
- Column family: logical grouping of columns stored close together
- Locality groups: grouping of column families
- SSTable: sequence of 64k blocks
- Batch writes
- Compactions: merge SSTables

Questions

Schedule

- No evaluated activities allowed this week
  - No task deadline either
- Current week 11? Semester continues until week 15?
- Rough plan for rest of course:
  - Week 11 Wednesday: “open lab”
  - Week 12 Monday: Projects in Lab
    - Week 12 Tuesday: Lab 8 & 9 due
  - Week 12 Wednesday: Projects in Lab
  - **Week 13 Monday: Project Reports Due, Presentations Given**
  - **Week 13 Wednesday: HBase lab**
  - Week 14 Monday: Graph data lecture
  - Week 14 Wednesday: Unmarked lab
  - **Week 15 Monday: Wrap-up, exam preparation**
  - Week 15 Wednesday: Not sure really ☺