Big Data Technologies and Datasets

Lecture 3 – NoSQL

Based on slides by:
Prof. Lior Rokach and other sources

Outline

- NoSQL Overview
- Consistency vs. availability
- Weak consistency
- NoSQL Variants
- Choosing a NoSQL DB

Recap scalability

- On a single server:
  - Usually RDBMS
- For several servers:
  - Distributed RDBMS
- For hundreds of servers:
  - NoSQL?

כשנרצה לעשות סקיילינג
באופן הרבה יותר רחב,
נעבור לבסיסי נתונים
no sql מבוסס״

3

Trends in the Data Systems Worlds

355 systems in ranking, April 2020

Rank			DBMS	Database Model	Score		
Apr 2020	Mar 2020	Apr 2019			Apr 2020	Mar 2020	Apr 2019
1.	1.	1.	Oracle	Relational, Multi-model	1345.42	+4.78	+65.48
2.	2.	2.	MySQL	Relational, Multi-model	1268.35	+8.62	+53.21
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1083.43	-14.43	+23.47
4.	4.	4.	PostgreSQL	Relational, Multi-model	509.86	-4.06	+31.14
5.	5.	5.	MongoDB	Document, Multi-model	438.43	+0.82	+36.45
6.	6.	6.	IBM Db2	Relational, Multi-model	165.63	+3.07	-10.42
7.	7.	↑ 8.	Elasticsearch	Search engine, Multi-model	148.91	-0.26	+2.91
8.	8.	↓ 7.	Redis	Key-value, Multi-model	144.81	-2.77	-1.57
9.	↑ 10.	↑ 10.	SQLite	Relational	122.19	+0.24	-2.02
10.	↓ 9.	↓ 9.	Microsoft Access	Relational	121.92	-3.22	-22.73
11.	11.	11.	Cassandra	Wide column	120.07	-0.88	-3.54
12.	↑ 13.	12.	MariaDB	Relational, Multi-model	89.90	+1.55	+4.67
13.	↓ 12.	13.	Splunk	Search engine	88.08	-0.44	+4.99
14.	14.	↑ 15.	Hive	Relational	84.05	-1.32	+9.34
15.	15.	↓ 14.	Teradata	Relational, Multi-model	76.59	-1.25	+1.25
16.	16.	↑ 19.	Amazon DynamoDB	Multi-model	64.27	+1.75	+8.26
17.	17.	↓ 16.	Solr	Search engine	53.59	-1.50	-6.64
18.	18.	↑ 21.	SAP HANA	Relational, Multi-model	53.29	-0.98	-2.05
19.	↑ 20.	↑ 20.	SAP Adaptive Server	Relational	52.63	-0.14	-3.17
20.	↓ 19.	↓ 18.	FileMaker	Relational	52.08	-2.08	-6.34

Taken from DB-Engines.com

4

NoSQL Definition

- From www.nosql-database.org:
  - Next Generation Databases mostly addressing some of the points:
    - being non-relational
    - distributed
    - open-source
    - horizontally scalable

horizonal scaling == scale out == adding servers (instead of improving the computer)

  - The original intention has been modern web-scale databases.
  - The movement began early 2009 and is growing rapidly.
  - Often more characteristics apply as:
    - schema-free
    - easy replication support
    - simple API
    - eventually consistent (not ACID)
    - huge amount of data

5

Important Design Goals

- Scale out: designed for scale
  - Commodity hardware
  - Low latency updates
  - Sustain high update/insert throughput
- Elasticity - scale up and down with load
- High availability - downtime implies lost revenue
  - Replication (with multi-mastering)
  - Geographic replication
  - Automated failure recovery

6

RDBMS vs. NoSQL

	RDBMS	Most NoSQL
Popularity	Very High	Increasing
Tools	Many	Few
Consistency	ACID	Limited
Query Execution	Rich, Fast	Limited
Standard	SQL	No Standard
Scalability	High	Very High
Schema	Static	Dynamic

7

No SQL or Not Only SQL

- No Sql = anti relational DBMS
- Not only Sql = don't use only relational DBMS
  - RDBMS is just another option
  - This is what most big players do (e.g., Google)

- Current outlook:
  - Majority of data in 10 years is still stored in RDBMS

8

No Free Luch: Lower Priorities

disadvantages in NoSQL:

- No Complex querying functionality
  - No support for SQL
  - CRUD operations through database specific API
- No support for joins
  - Materialize simple join results in the relevant row
  - Give up normalization of data?
- No support for transactions
  - Most data stores support single row transactions
  - Tunable consistency and availability

9

Consistency vs. availability

10

ACID: (Desired) Transaction Properties

Recall ACID:

- Atomicity:
  - All actions in the transaction happen, or none happen
- Consistency:
  - A transaction preserves the consistency of the DB
- Isolation:
  - Execution of one transaction is isolated from others
- Durability:
  - If a transaction commits, its effects persist

ACID is great… but achieving it is costly…

קשה לתמוך בכל העקרונות
באופן מלא כאשר מדברים על
בסיס נתונים מבוזר....

11

CAP

- Consistency:
  - All nodes should see the same data at the same time

תמיד נראה את המידע באותה הצורה

- Availability:
  - Node failures do not prevent survivors from continuing to operate
- Partition-tolerance:
  - The system continues to operate despite network partitions

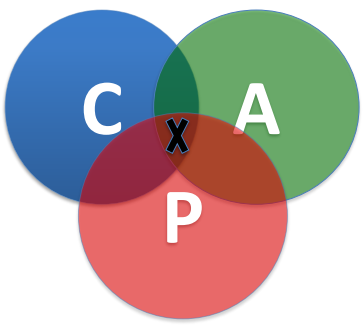
12

Network partitions:
As if I had group of servers in Israel, and another group in US, and they communicate with each other regulary

CAP Theorem (Brewer Theorem)

2000 Prof. Eric Brewer, PoDC Conference Keynote
2002 Seth Gilbert and Nancy Lynch, ACM SIGACT News 33(2)

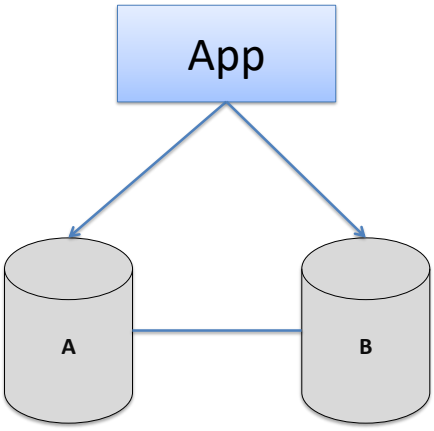
> *Of three properties of shared-data systems - data **C**onsistency, system **A**vailability and tolerance to network **P**artitions - only two can be achieved at any given moment in time.*



Only 2 out of 3 principals of CAP can be present at the same time (applicable for all kinds of DBSM

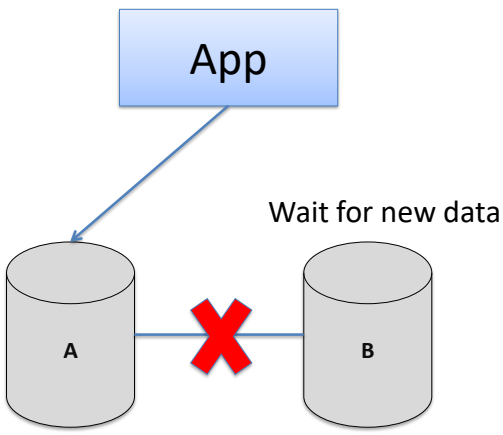
Proof (intuition)

Consistent and available
No partition.



Intuition

Available and partitioned
Not consistent, we get back old data.



15

Interplay with CAP

- To scale horizontally, we partition the data
  - Must have Partition tolerance
  - Must choose: Consistency or Availability

- Distributed RDBMS choose consistency
- NoSQL choose availability [low replica consistency]

17

Why Sacrifice Consistency?

- It is a simple solution
  - Pushes the problem to app developer
- Sacrificing availability is unacceptable in the Web

18

Why Sacrifice Consistency? (contd.)

- Consistency is not needed in many applications
- Only 10% of queries need strong consistency
  - Amazon reports only 5% of queries require Consistency
- Airline reservation systems commonly tolerate some level of conflicts
  - One of the reasons for over-booking
- RDBMS vendors also acknowledge this
  - MySQL ships with a lower isolation level

19

Why Sacrifice Consistency? (contd.)

- Data is noisy and inconsistent anyway
- Making it, say, 1% worse does not matter

20

Weak consistency

זוהי למעשה רמת ההתפשרות שאנחנו מוכנים
להגיע אליה מבחינת עקביות, כאשר אנחנו עוברים
לדבר על
NoSQL

21

אלו הם העקרונות שאנחנו מציבים לעצמנו כאשר אנחנו מדברים על בסיסי נתונים שהם NoSQL

BASE as an Alternative to ACID

Basically **A**vailable, **S**oft state, **E**ventually consistent

- Basically available: Nodes can go down, but the whole system shouldn't be affected
- Soft state (scalable): The state of the system and data may change over time (even without input)
- Eventual Consistency: Given enough time, data will be consistent across the distributed system

העקרון השני הוא סוג של נגזרת של העקרון השלישי. מצב הנתונים במערכת יכול להשתנות (לאורך זמן גם אם לא ביצענו שינויים במערכת (לדוגמה לא הכנסנו אינפוט חדש

2

BASE vs ACID

- ACID:
  - Strong consistency.
  - Less availability.
  - Pessimistic concurrency.
  - Complex.
- BASE:
  - Availability is the most important thing.
  - Willing to sacrifice for this (CAP).
  - Weaker consistency (Eventual).
  - Simple and fast.
  - Optimistic.

23

Strong and Weak Consistency

- Strong consistency - after an update completes, any future access will return the updated value.
- Weak consistency - no guarantee that future accesses will return the updated value.

The period between the update and the moment when it is guaranteed that any observer will always see the updated value is dubbed the *inconsistency window*.

24

Eventually Consistent

- A special form of weak consistency
- Guarantees that
  - If no new updates are made to the object, eventually all accesses will return the last updated value.
  - If no failures occur, the maximum size of the inconsistency window can be determined
    - Affecting factors: load, replicas, comm. delay

25

Variations of Eventual Consistency

- Read-your-writes consistency.
  - The effect of a write operation by a process on data item x will always be seen by a successive read operation on x by the same process.

אין סיכוי שאני פרסמתי פוסט בפייסבוק ואני לא אראה אותו מיידית

- Monotonic read consistency
  - If you read version X of object Y, next reads cannot read older versions (for the same process)

אם כרגע אני קורא את גרסה 3 של המערכת, אין סיכוי שבקריאה הבאה אחשף לגרסה 2 של המערכת

- Monotonic write consistency
  - Writes of the same process are chronological

אם העלאתי פוסט ואז עדכנתי את הפוסט (או עדכנתי מידע כלשהו), אז שאר המשתמשים רואים את זה בסדר שזה התבצע

26

NoSql Variants

נדבר על משפחות גדולות, שונות זו מזו, שנמצאות בתוך משפחת העל הזו שנקראת NoSQL

27

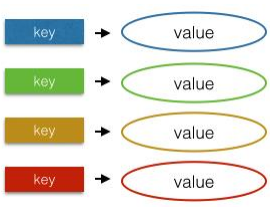
Major Impact Systems

- "Memcached demonstrated that in-memory indexes can be highly scalable, distributing and replicating objects over multiple nodes."
- "Dynamo pioneered the idea of [using] eventual consistency as a way to achieve higher availability and scalability: data fetched are not guaranteed to be up-to-date, but updates are guaranteed to be propagated to all nodes eventually."
- "BigTable demonstrated that persistent record storage could be scaled to thousands of nodes, a feat that most of the other systems aspire to."

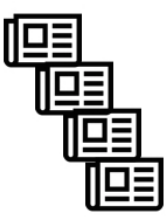
28

Some families of NoSQL Technology:

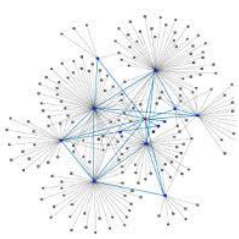
Redis **Key-Value**



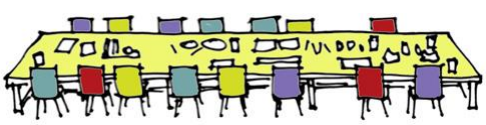
Document Mongo-DB



Graph



BigTable **Column Family**



29

הבעיה: נהייה לי שרת שהוא צוואר הבקבוק,
.ובנוסף יש עלויות תקשורת יחסית גבוהות

כאן, כדי לבזר את ההאש טייבל, נניח שיש לי שני שרתים עם פרטי מידע שונים. אני אחזיק עוד שרת, שמחזיק כל מפתח לאיזה שרת הוא שייך, ואז בהינתן פנייה עם מפתח כלשהו, הוא ינתב אותה לשרת המתאים.

Key-Value Stores

- The main idea here is using a hash table where there is a unique key and a pointer to a particular item of data.
- The Key/value model is the simplest and easiest to implement.
- But it is inefficient when you are only interested in querying or updating part of a value, among other disadvantages.

Modulus Map:
Instead of holding a "mapper" server, I'll take the key and apply an hash function on it (that will tell me where the information item is). Than, I turn to the correct server and apply another hash function to retrieve the value

Disadvantage:
a lot of time is neccessary when scaling out (adding another server for example)

30

Modulus Map vs. Consistent Hash

hash here modulu here is 3 ->

Mapped Integer	1	2	3	4	5	6	7	8	9	10	11	12
Key location (3 node)	1	2	0	1	2	0	1	2	0	1	2	0
Key location (4 node)	1	2	3	0	1	2	3	0	1	2	3	0
Key on same node?	Yes	Yes	No	No	No	No	No	No	No	No	No	Yes

אם הוספתי שרת אחד בלבד, שרת רביעי, פונקציית ההאש הפכה להיות מודולו 4. אפשר לראות שכעת רק רבע מהמידע אמור להישאר במקום שלו , ואת שאר המידע עליי להעביר. זה משהו שהופך את הסקיילינג למאוד מסובך

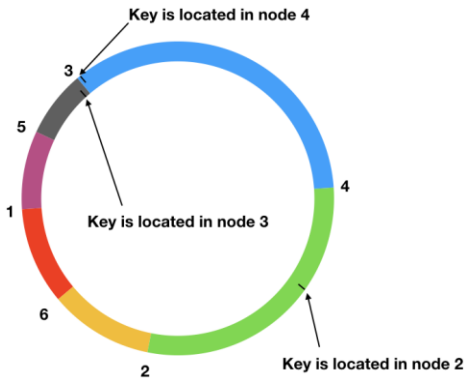
31

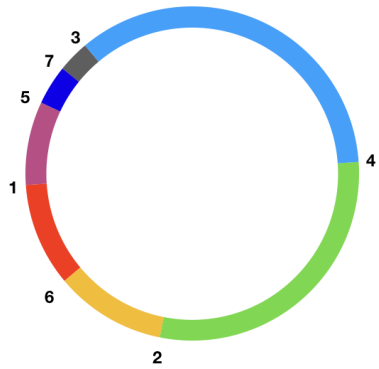
The solution: Consistent Hash:

Modulus Map vs. Consistent Hash (Contd.)

כאן אין יותר מודולו והאש - זהו המיפוי עצמו.

כעת, אם הוספת שרת 7, רק המידע שהיה שמור בשרת 3 השתנה





לדוגמה, כל טווח הערכים הירוק, נשמר בשרת 2

32

Key-Value Stores

Examples	Tokyo Cabinet/Tyrant, Redis, Voldemort, Oracle BDB
Typical applications	Content caching (Focus on scaling to huge amounts of data, designed to handle massive load), logging, etc.
Data model	collection of Key-Value pairs
Strengths	Fast lookups
Weaknesses	Stored data has no schema

33

Memcached - Intro

- Free & open source distributed memory caching
- Used by: Facebook, Wikipedia, Flickr, Twitter, YouTube, Digg, Bebo, WordPress, Craigslist, …
- Multiple client APIs: command line, C/C++, Go, PHP, Java, Python, Ruby, Perl, Windows/.NET, mySQL, PostgreSQL, Erlang, Lua, LISP, ColdFusion, io

34

Memcached – Data Model

חסרון: יכול להביא לשכפול טבלאות בבסיס הנתונים (אם למשל אני רוצה לקבל שם לפי ת.ז, ושם לפי כתובת. בכך אצטרך 2 טבלאות שונות)

- Key-Value store
  - Cache is made up of { key, value, expiration time, flags }
  - All access is O(1)
- Client software
  - Provided with a list of memcached servers
  - Hashing algorithm: chooses a server based on the key
- Server software
  - Stores keys and values in an in-memory hash table
  - Throw out old data when necessary
    - LRU cache and time-based expiration
  - Servers are unaware of each other

35

Memcached - Operations

האיי פי איי שלו הוא מאוד פשוט.
חלק מהפקודות שלו

- **get** – retrieve one or more keys: returns key, flags, bytes

- **set** – store data
- **add** – store data only if the server does not have data for the key
- **replace** – store data if the server does have data for the key

- **append** – add data after existing data
- **prepend** – add data before existing data

- **incr** key value – increment key by value
- **decr** key value – decrement key by value

- **touch** key exptime – update the expiration time

- **delete** key – remove the key and associated value
- **flush_all** – clear the cache

הפקודה מעדכנת את הטיימסטמפ של פריט מידע מסוים כדי להפוך אותו ליותר חדש, כדי להוריד את הסיכוי שידרס כאשר לדוגמה אנחנו אוזלים במקום בזיכרון

36

Document Stores

- The model is basically versioned documents that are collections of other key-value collections.
- The semi-structured documents are stored in formats like JSON.
- Document databases are essentially the next level of Key/value, allowing nested values associated with each key.
- Document databases support querying more efficiently.

37

Document Stores (contd.)

Examples	CouchDB, MongoDb
Typical applications	Web applications (Similar to Key-Value stores, but the DB knows what the Value is)
Data model	Collections of Key-Value collections
Strengths	Tolerant of incomplete data
Weaknesses	Query performance, no standard query syntax

38

Mongodb - Intro

- MongoDB is a scalable, high-performance, open source, document-oriented database
- Main properties:
  - Multiple languages driver support (e.g. Java, Python,…)
  - Connection pooling
  - Supports several types of aggregation frameworks
    - Map Reduce with JavaScript
  - Support for indexes
  - Updates are atomic.
  - Low contention locks.

39

Mongodb - Operations

- Using a database:
  use foobar

- Create a document – e.g. create a post document:
  post = {
  "title" : "My Blog Post",
  "content" : "Here's my blog post.",
  "date" : new Date()
  }

אפשר להכניס לקולקשן אחד מסמכים עם עמודות שונות, יש גמישות.

- Insert a document – e.g. insert the post doc into the blog collection:
  db.blog.insert(post)

40

Mongodb – Operations (contd.)

פקודות נוספות: ביצוע חיפוש, עדכון ומחיקה

- Read documents – e.g. retrieve all documents:
  db.blog.find()
- Read with criterion:
  db.blog.find({"title" : "My Blog Post"})

- Update documents:
  post.content="Updated Content"
  db.blog.update({title : "My Blog Post"}, post)

- Delete documents:
  db.blog.remove({title : "My Blog Post"})

41

אנחנו ממשיכים כעת לדאטה בייסים מהסוג השלישי:

Column Family Stores

- These were created to store and process very large amounts of data distributed over many machines.
- There are still keys but they point to multiple columns.
- The columns are arranged by column families.
- Google's "Bigtable: A Distributed Storage System for Structured Data" (2006).

42

Column Family Stores (contd.)

Examples	Cassandra, HBase, Riak
Typical applications	Distributed file systems
Data model	Columns → column families
Strengths	Fast lookups, good distributed storage of data
Weaknesses	Very low-level API

43

Bigtable - Intro

- Highly available distributed storage
- Built with semi-structured data in mind
  - URLs: content, metadata, links, anchors, page rank
  - User data: preferences, account info, recent queries
  - Geography: roads, satellite images, points of interest, annotations
- Large scale
  - Petabytes of data across thousands of servers
  - Billions of URLs with many versions per page
  - Hundreds of millions of users
  - Thousands of queries per second
  - 100TB+ satellite image data

44

Bigtable - Uses

- At Google, used for:
  - Google Analytics
  - Google Finance
  - Personalized search
  - Blogger.com
  - Google Code hosting
  - YouTube
  - Gmail
  - Google Earth & Google Maps
  - Dozens of others…

45

Bigtable – What is a Bigtable?

"A Bigtable is a sparse, distributed, persistent multidimensional sorted map"

Columns

Rows

sorted

	"language:"	"contents:"	anchor:cnnsi.com	anchor:mylook.ca
com.aaa	EN	<!DOCTYPE html PUBLIC…		
com.cnn.www	EN	<!DOCTYPE HTML PUBLIC…	CNN	CNN.com
com.cnn.www/TECH	EN	<!DOCTYPE HTML>…		
com.weather	EN	<!DOCTYPE HTML>…		

Webtable example

46

Bigtable - Data Model

- A table in Bigtable is a sparse, distributed, persistent multidimensional sorted map
- Map indexed by a row key, column key, and a timestamp
  - (row:string, column:string, time:int64) → uninterpreted byte array
- Supports lookups, inserts, deletes
  - Single row transactions only

ולכן יש כמה מימדים - סהכ שלושה

Columns

Rows

versions

sorted

	"language:"	"contents:"		
com.aaa	EN			
com.cnn.www	EN	<!DOCTYPE html…	t_2, t_4, t_7	
com.cnn.www/TECH	EN	<!DOCTYPE html…	t_7	
com.weather	EN	<!DOCTYPE html…	t_7, t_{15}	

Image Source: Chang et al., OSDI 2006

47

משמעות הטיימסטמפ - גרסאות שמאפשרות לנו לשמור נתונים לאורך זמן

Rows and Columns

פירוט על כל מימד במבנה

נרצה שערכים שאנחנו ניגשים אליהם יחסית הרבה ביחד, ישמרו על אותו השרת.

- Rows maintained in sorted lexicographic order
  - Applications can exploit this property for efficient row scans
  - Row ranges dynamically partitioned into tablets
- Columns grouped into column families
  - Column key = family:qualifier
  - Column families provide locality hints
  - Unbounded number of columns
- Version Number
  - Unique within each key
  - By default→ System's timestamp

48

Tablets

Big Table consists of a lot of tablets

כל טאבלט מכיל טווח של ערכים רציפים (של שורות רציפות). טאבלט אחד לא יהיה מפוצל במספר שרתים!)

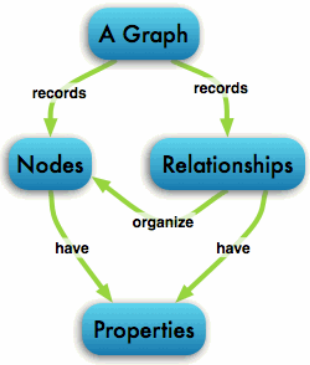
- Table partitioned dynamically by rows into tablets
  - A table starts as one tablet, as it grows, it splits into multiple tablets
- Tablet = range of contiguous rows
  - Unit of distribution and load balancing
  - Approximate size: 100-200 MB per tablet by default
  - Nearby rows will usually be served by the same server
  - Accessing nearby rows requires communication with a small # of machines
  - You need to select row keys to ensure good locality
    - E.g., reverse domain names:
      com.cnn.www instead of www.cnn.com

נרצה לעשות את זה כדי שאתרים שנמצאים תחת אותו הדומיין ימצאו תחת אותו הטאבלט. ההיפוך של הדומיין מאפשר לקרב בין אתרים שקשורים אחד לשני (כמו לדוגמה האתר של האוניברסיטה והאתר של המעבדה של דן, בנמצא בתת דומיין של האוניברסיטה)

49

Graph Databases

- Instead of tables of rows and columns and the rigid structure of SQL, a flexible graph model is used.
- The property graph contains connected entities (the nodes) and relationships (directed edges between two node-entities).
- Nodes and relationships can hold any number of attributes (key-value-pairs).
- Can scale across multiple machines.
- Support efficient graph operations.



50

Graph Databases (contd.)

Examples	Neo4J, InfoGrid, Infinite Graph
Typical applications	Social networking, Recommendations (Focus on modeling the structure of data – interconnectivity)
Data model	"Property Graph" – Nodes
Strengths	Graph algorithms e.g. shortest path, connectedness, n degree relationships, etc.
Weaknesses	Has to traverse the entire graph to achieve a definitive answer. Not easy to cluster.

51

Neo4J - Intro

- Neo4j is an open-source NoSQL graph database implemented in Java and Scala.
- With development starting in 2003, it has been publicly available since 2007.
- Neo4j is used today by hundreds of thousands of companies and organizations in almost all industries.
- Use cases include matchmaking, network management, software analytics, scientific research, routing, organizational and project management, recommendations, social networks, and more.

52

Neo4J – Main properties

- Neo4j implements the Property Graph Model (and operations above it) efficiently down to the storage level.

- As opposed to graph processing or in-memory libraries, Neo4j provides full database characteristics including:
  - ACID transaction compliance
  - cluster support
  - runtime failover
- Making it suitable to use graph data in production scenarios.

53

Neo4J Operations - Shortest Path

Calculating the shortest path (least number of relationships) between two nodes:

```
Node startNode = graphDb.createNode();
Node middleNode1 = graphDb.createNode();
Node middleNode2 = graphDb.createNode();
Node middleNode3 = graphDb.createNode();
Node endNode = graphDb.createNode();
createRelationshipsBetween( startNode, middleNode1, endNode );
createRelationshipsBetween( startNode, middleNode2, middleNode3, endNode );

// Will find the shortest path between startNode and endNode via
// "MY_TYPE" relationships (in OUTGOING direction), like f.ex:
//
// (startNode)-->(middleNode1)-->(endNode)
//
PathFinder<Path> finder = GraphAlgoFactory.shortestPath(
        Traversal.expanderForTypes( ExampleTypes.MY_TYPE, Direction.OUTGOING ), 15 );
Iterable<Path> paths = finder.findAllPaths( startNode, endNode );
```

54

Choosing a NoSQL DB

55

Classification Dimensions

- Data model
  - How the data is stored: key/value, column-oriented, ...
  - How to access data?
  - Can the schema evolve over time?
- Storage model
  - In-memory or persistent?
- API
  - SQL-like, Java etc.

56

Classification Dimensions (contd.)

- Consistency model
  - Where is the DB located in the CAP triangle? מאחר ולא ניתן לקיים את שלושת התכונות ביחד
  - This translates in how fast the system handles READS and WRITES
- Locking, waits and deadlocks פחות התמקד...
  - Support for multiple client accessing data simultaneously
  - Is locking available?
  - Is it deadlock free?

57

Classification Dimensions (contd.)

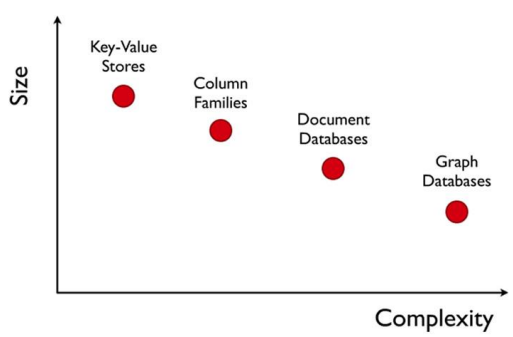
- Read/Write performance
  - Some systems are better for READS, other for WRITES
- Failure Handling
  - How each data store handle server failures?
  - Is it able to continue operating in case of failures?
  - Does the system support “hot-swap”?
- Compression
  - Is the compression method pluggable?
  - What type of compression?
- Load Balancing
  - Can the storage system seamlessly balance load?

58

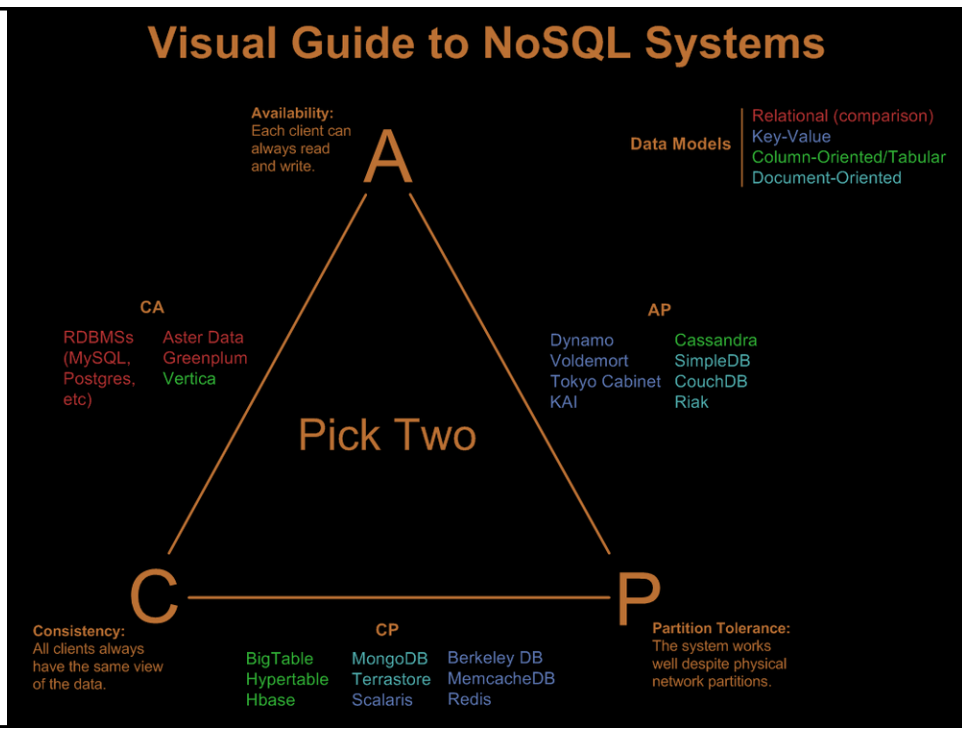
NoSQL and Scalability

- Two dimensions of scalability:
  - Data Size and Data Complexity

אם אנחנו רוצים כמויות ענקיות של מידע, נרצה להשתמש במפתח וערך



59



60

Summary

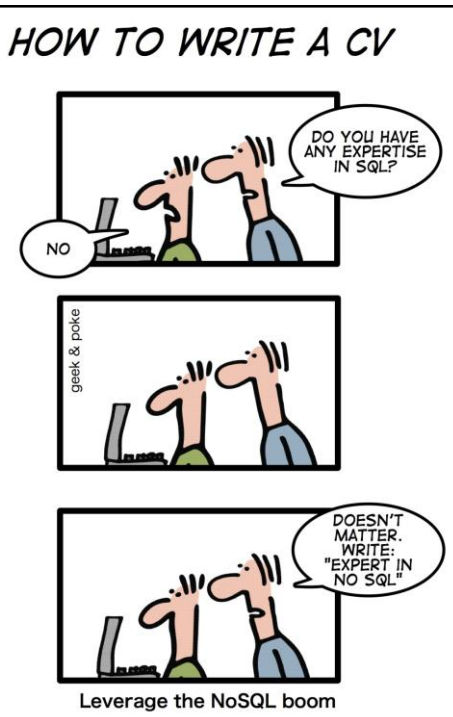
	Performance	Scalability	Flexibility	Complexity	Functionality
Key-Value Stores	high	high	high	none	variable (none)
Column stores	high	high	moderate	low	minimal
Document stores	high	variable (high)	high	low	variable (low)
Graph databases	variable	variable	high	high	graph theory
Relational databases	variable	variable	low	moderate	relational algebra

61

Summary (contd.)

- Different databases suitable for different workloads
- Evolving systems – landscape changing dramatically
- Active development community around open source systems
- In-house systems enriched or redesigned
  - MegaStore (Google): support for transactions and declarative querying
  - Spanner (Google): Rumored to have move extensive transactional support across data centers

62



63