Databases for Big Data – part 1

Valentina Ivanova

IDA, Linköping University

Outline – Today – Part 1

- RDBMS → NoSQL → NewSQL
- DBMS – OLAP vs OLTP (ACID)
- NoSQL Concepts and Techniques
  - Horizontal scalability
  - Consistency models
    - CAP theorem: BASE vs ACID
  - Consistent hashing
  - Vector clocks
- NoSQL Systems - Types and Applications
- Hadoop Distributed File System - HDFS

Outline – Next Lecture – Part 2

- Amazon DynamoDB
- HBase
- Hive
- Shark

DB rankings – September 2016

Rank Sep 2016	Rank Aug 2016	Rank Sep 2015	DBMS	Database Model	Score Sep 2016	Score Aug 2016	Score Sep 2015
1.	1.	1.	Oracle	Relational DBMS	1425.56	-2.16	-37.81
2.	2.	2.	MySQL	Relational DBMS	1354.03	-3.01	+76.28
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1211.55	+6.51	+113.72
4.	↑ 5.	↑ 5.	PostgreSQL	Relational DBMS	316.35	+1.10	+30.18
5.	↓ 4.	↓ 4.	MongoDB	Document store	316.00	-2.49	+15.43
6.	6.	6.	DB2	Relational DBMS	181.19	-4.70	-27.95
7.	7.	↑ 8.	Cassandra	Wide column store	130.49	+0.26	+2.89
8.	8.	↓ 7.	Microsoft Access	Relational DBMS	123.31	-0.74	-22.68
9.	9.	9.	SQLite	Relational DBMS	108.62	-1.24	+0.97
10.	10.	10.	Redis	Key-value store	107.79	+0.47	+7.14
11.	11.	↑ 14.	Elasticsearch	Search engine	96.48	+3.99	+24.93
12.	12.	↑ 13.	Teradata	Relational DBMS	73.06	-0.57	-1.20
13.	13.	↓ 11.	SAP Adaptive Server	Relational DBMS	69.16	-1.88	-17.36
14.	14.	↓ 12.	Solr	Search engine	66.96	+1.19	-14.98
15.	15.	15.	HBase	Wide column store	57.81	+2.30	-1.22
16.	16.	↑ 17.	FileMaker	Relational DBMS	55.35	+0.34	+4.35
17.	17.	↑ 18.	Splunk	Search engine	51.29	+2.38	+9.06
18.	18.	↓ 16.	Hive	Relational DBMS	48.82	+1.01	-4.71
19.	19.	19.	SAP HANA	Relational DBMS	43.42	+0.68	+5.22
20.	20.	↑ 25.	MariaDB	Relational DBMS	38.53	+1.65	+14.31
21.	21.	21.	Neo4j	Graph DBMS	36.37	+0.80	+2.83
22.	↑ 24.	↑ 24.	Couchbase	Document store	28.54	+1.14	+2.28
23.	23.	↓ 22.	Memcached	Key-value store	28.43	+0.74	-3.99
24.	↓ 22.	↓ 20.	Informix	Relational DBMS	28.19	-0.86	-9.76
25.	25.	↑ 28.	Amazon DynamoDB	Document store	27.42	+0.82	+7.43

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

RDBMS → NoSQL → NewSQL

DBMS history (Why NoSQL?)

- 1960 – Navigational databases
- 1970 – Relational databases (RDBMS)
- 1990 – Object-oriented databases and Data Warehouses
- 2000 – XML databases
- Mid 2000 – first NoSQL
- 2011 – NewSQL

RDBMS

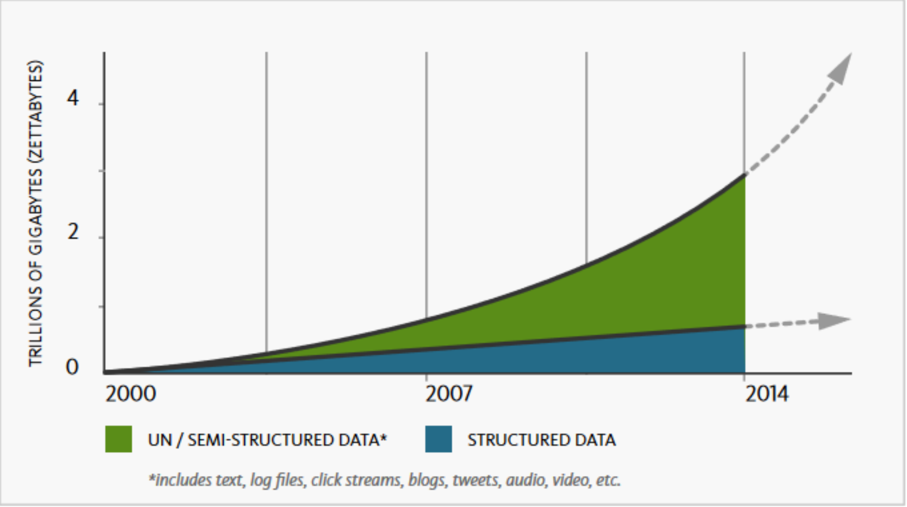
- Established technology
- Transactions support & ACID properties
- Powerful query language - SQL
- Experiences administrators
- Many vendors

Table: Item

item id	name	color	size
45	skirt	white	L
65	dress	red	M

But … – One Size Does Not Fit All[1]

- Requirements have changed:
  - Frequent schema changes, management of unstructured and semi-structured data
  - Huge datasets
  - RDBMSs are not designed to be
    - distributed
    - continuously available
  - High read and write scalability
  - Different applications have different requirements[1]



[1] "One Size Fits All": An Idea Whose Time Has Come and Gone https://cs.brown.edu/~ugur/fits_all.pdf
Figure from: http://www.couchbase.com/sites/default/files/uploads/all/whitepapers/NoSQL-Whitepaper.pdf

NoSQL (not-only-SQL)

- A broad category of disparate solutions
- Simple and flexible non-relational data models
- High availability & relax data consistency requirement (CAP theorem)
  - BASE vs ACID
- Easy to distribute – horizontal scalability
- Data are replicated to multiple nodes
  - Down nodes easily replaced
  - No single point of failure
- Cheap & easy (or not) to implement (open source)

But …

- No support for SQL → Low level programming → data analysists need to write custom programs
- No ACID
- Huge investments already made in SQL systems and experienced developers
- NoSQL systems do not provide interfaces to existing tools

NewSQL[DataMan]

- First mentioned in 2011
- Supports the relational model
  - with horizontal scalability & fault tolerance
- Query language - SQL
- ACID
- Different data representation internally
- VoltDB, NuoDB, Clustrix, Google Spanner

NewSQL Applications[DataMan]

- RBDMS applicable scenarios
  - transaction and manipulation of more than one object, e.g., financial applications
  - strong consistency requirements, e.g., financial applications
  - schema is known in advance and unlikely to change a lot
- But also Web-based applications[1]
  - with different collection of OLTP requirements
    - multi-player games, social networking sites
  - real-time analytics (vs traditional business intelligence requests)

[1] http://cacm.acm.org/blogs/blog-cacm/109710-new-sql-an-alternative-to-nosql-and-old-sql-for-new-oltp-apps/fulltext

DBMS – OLAP and OLTP

DBMS applications – OLAP and OLTP

- OLTP – Online transaction processing - RDBMS
  - university database; bank database; a database with cars and their owners; online stores

- OLAP – Online analytical processing - Data warehouses
  - Summaries of multidimensional data

    Example: sale (item, color, size, quantity)

    What color/type of clothes is popular this season?

DBMS applications – OLTP

Table: Cart

order id	Item id	quantity
1	45	1
1	55	1
1	65	2
2	65	1

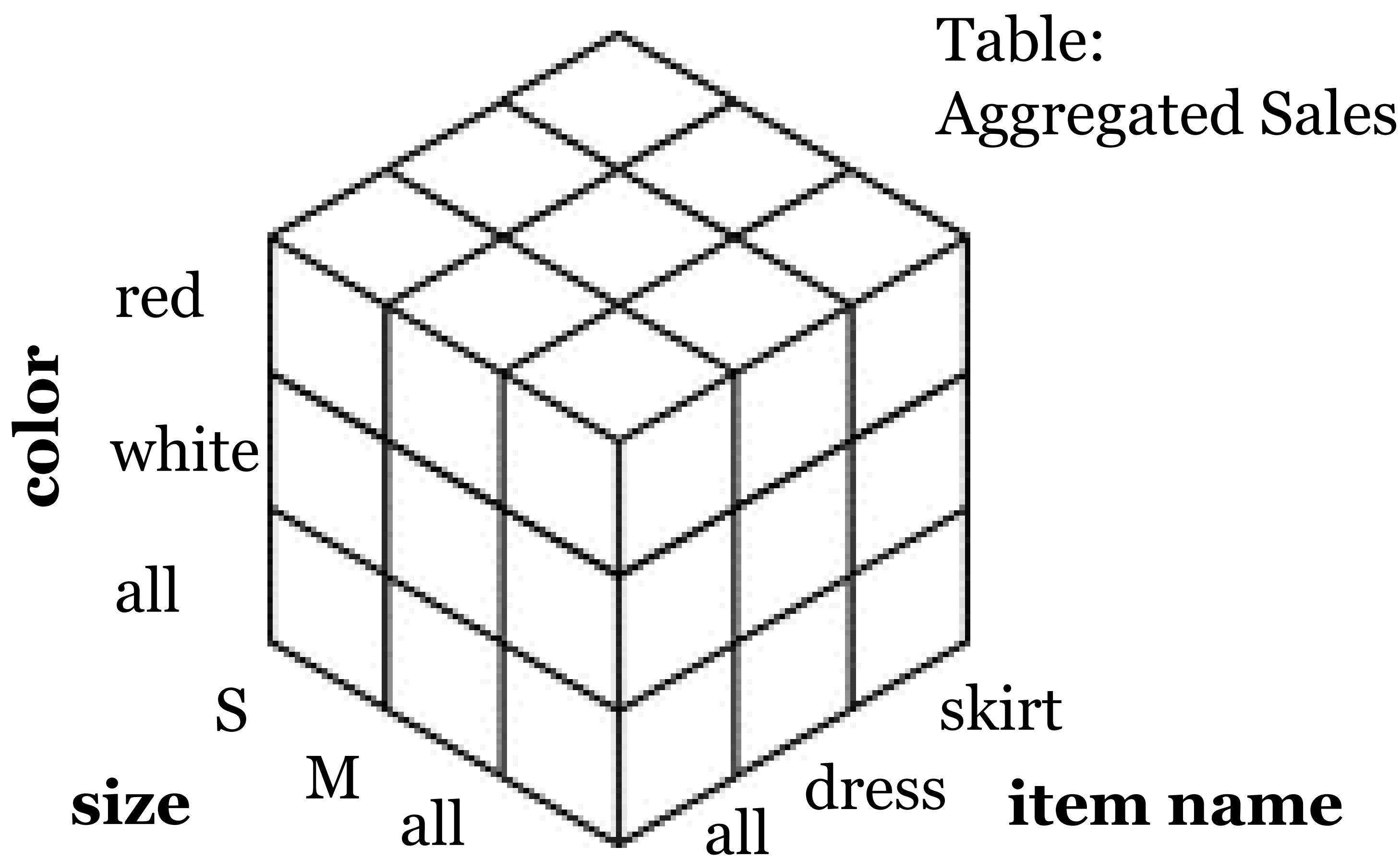
Table: Order

order id	customer
1	22
2	33

Table: Item

item id	name	color	size
45	skirt	white	L
65	dress	red	M

DBMS applications – OLAP



DBMS applications – OLAP and OLTP

- Relational DBMS vs Data Warehouse
  http://datawarehouse4u.info/OLTP-vs-OLAP.html

	RDBMS (OLTP)	Data Warehouse (OLAP)
Source of data	Operational data; OLTPs are the original source of the data.	Consolidation data; OLAP data comes from the various OLTP DBs
Purpose of data	To control and run fundamental business tasks	To help with planning, problem solving, and decision support
What the data	Reveals a snapshot of ongoing business processes	Multi-dimensional views of various kinds of business activities
Inserts & Updates	Short and fast inserts and updates initiated by end users	Periodic long-running batch jobs refresh the data
Queries	Relatively standardized and simple queries returning relatively few records	Often complex queries involving aggregations
Processing Speed	Typically very fast	Depends on the amount of data involved
Space Requirements	Can be relatively small if historical data is archived	Larger due to the existence of aggregation structures and history data;
Database Design	Highly normalized, many tables	Typically de-normalized, fewer tables
Backup & Recovery	Highly important	Reloading from OLTPs

DBMS applications – OLTP

- OLTP – Online transaction processing
  - large number of data reads, writes and updates → transactions!

```
Read-item(my-account)
my-account := my-account - 2000
Write-item(my-account)
Read-item(other-account)
other-account := other-account + 2000
Write-item(other-account)
```

DBMS applications – Transactions

- A transaction is a logical unit of database processing and consists of one or several operations.
  - Begin transaction
  - Reads and writes
  - Commit or rollback
  - End transaction
- It leaves the database in a consistent state

DBMS applications – Transactions

Transaction 1	**Time**	**Transaction 2**
Read-item(my-account)		
my-account := my-account – 2000		
		Read-item(my-account)
		my-account := my-account +1000
Write-item(my-account)		
Read-item(other-account)		
		Write-item(my-account)
Other-account := other-account + 2000		
Write-item(other-account)		

DBMS applications – Transactions

Transaction 1	Time	Transaction 2
Read-item(5000)		
my-account := 5000 – 2000		
		Read-item(5000)
		my-account := 5000 +1000
Write-item(3000)		
Read-item(other-account)		
		Write-item(6000)
Other-account := other-account + 2000		
Write-item(other-account)		

DBMS applications – Transactions

Transaction 1	**Time**	**Transaction 2**
Read-item(my-account)	↓	
my-account := my-account – 2000		
Write-item(my-account)		
Read-item(other-account)		
Other-account := other-account + 2000		
Write-item(other-account)		
		Read-item(my-account)
		my-account := my-account +1000
		Write-item(my-account)

DBMS applications – Transactions

Transaction 1	Time	Transaction 2
Read-item(5000)	↓	
my-account := 5000 – 2000		
Write-item(3000)		
Read-item(other-account)		
Other-account := other-account + 2000		
Write-item(other-account)		
		Read-item(3000)
		my-account := 3000 +1000
		Write-item(4000)

Transactions - ACID properties

- **Atomicity** → A transaction is an atomic unit: It is either executed completely or not at all
- **Consistency** → A database that is in a consistent state before the execution of a transaction, is also in a consistent state after the execution of the transaction
- **Isolation** → A transaction should act as if it is executed in isolation from the other transactions
- **Durability** → Changes in the database made by a committed transaction are permanent

NoSQL Concepts and Techniques

NoSQL Databases (not only SQL)

nosql-database.org

NoSQL Definition:

Next Generation Databases mostly addressing some of the points: being ***non-relational, distributed, open source*** and ***horizontally scalable***.

The original intention has been modern web-scale databases. ... Often more characteristics apply as: ***schema-free, easy replication support, simple API, eventually consistent/BASE*** (not ACID), a ***huge data amount***, and more.

NoSQL: Concepts

Scalability: system can handle growing amounts of data without losing performance.

- Vertical Scalability (scale up)
  - add resources (more CPUs, more memory) to a single node
  - using more threads to handle a local problem
- Horizontal Scalability (scale out)
  - add nodes (more computers, servers) to a distributed system
  - gets more and more popular due to low costs for commodity hardware
  - often surpasses scalability of vertical approach

Distributed (Data Management) Systems

- Number of processing nodes interconnected by a computer network
- Data is stored, replicated, updated and processed across the nodes
- Networks failures are given, not an exception
  - Network is partitioned
  - Communication between nodes is an issue

  → Data consistency vs Availability

Consistency models[Vogels]

- A distributed system through the developers' eyes
  - Storage system as a black box
  - Independent processes that write and read to the storage
- Strong consistency – after the update completes, any subsequent access will return the updated value.
- Weak consistency – the system does not guarantee that subsequent accesses will return the updated value.
  - inconsistency window

Consistency models[Vogels]

- Weak consistency
  - Eventual consistency – if no new updates are made to the object, eventually all accesses will return the last updated value
    - Popular example: DNS

Consistency models[Vogels]

- Server side view of a distributed system – Quorum
  - N – number of nodes that store replicas
  - R – number of nodes for a successful read
  - W – number of nodes for a successful write

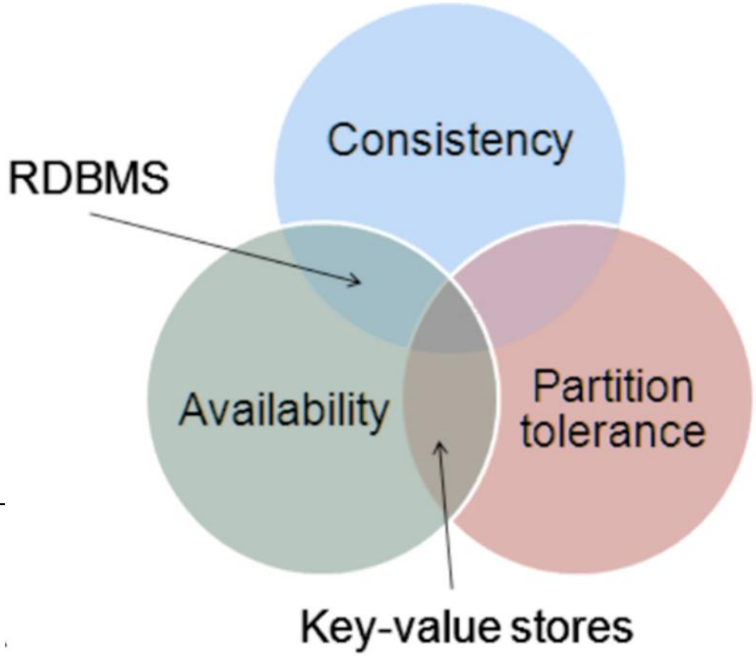
Consistency models[Vogels]

- Server side view of a distributed system – Quorum
  - High read loads – hundreds of N, R=1
  - Fault tolerance/availability (& relaxed consistency) W=1
  - R + W > N strong consistency
    - Consistency (& reduced availability) W=N
  - R + W ≤ N eventual consistency
    - Inconsistency window – the period until all replicas have been updated in a lazy manner

NoSQL: Concepts

CAP Theorem: Consistency, Availability, Partition Tolerance [Brewer]

Theorem
(Gilbert, Lynch SIGACT'2002):
only 2 of the 3 guarantees
can be given in a shared-data
system.



NoSQL: Concepts

CAP Theorem: Consistency, Availability, Partition Tolerance[Brewer]

- **Consistency**
  - after an update, all readers in a distributed system see the same data
  - all nodes are supposed to contain the same data at all times
- Example
  - single database instance will always be consistent
  - if multiple instances exist, all writes must be duplicated before write operation is completed

NoSQL: Concepts

CAP Theorem: Consistency, Availability, Partition Tolerance[Brewer]

- **Availability**
  - all requests will be answered, regardless of crashes or downtimes
- Example
  - a single instance has an availability of 100% or 0%, two servers may be available 100%, 50%, or 0%

NoSQL: Concepts

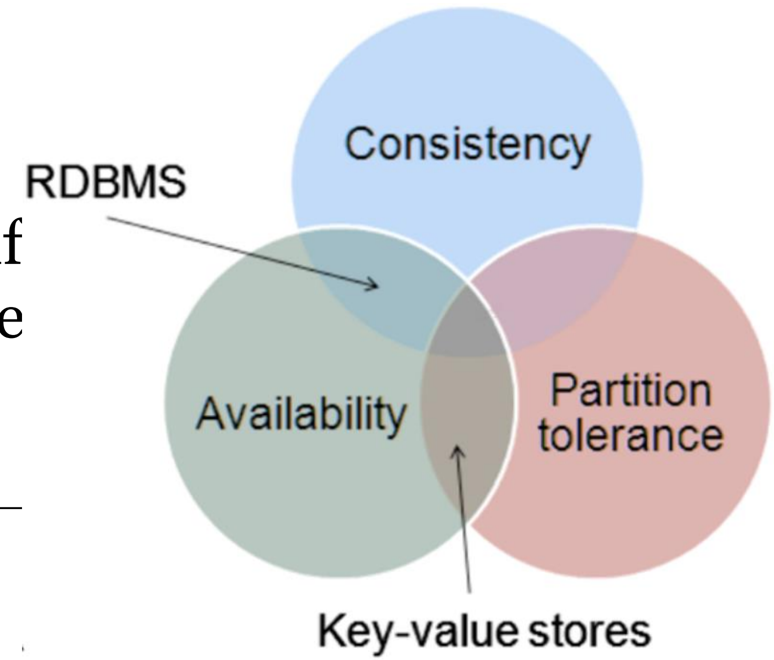
CAP Theorem: Consistency, Availability, Partition Tolerance[Brewer]

- **Partition Tolerance**
  - system continues to operate, even if two sets of servers get isolated
- Example
  - system gets partitioned if connection between server clusters fails
  - failed connection will not cause troubles if system is tolerant

NoSQL: Concepts

CAP Theorem: Consistency, Availability, Partition Tolerance[Brewer]

- (Positive) consequence: we can concentrate on two challenges
- **ACID** properties needed to guarantee consistency and availability
- **BASE** properties come into play if availability and partition tolerance is favored



NoSQL: Concepts

ACID: Atomicity, Consistency, Isolation, Durability

- **Atomicity** → all operations in a transaction will complete, or none will
- **Consistency** → before and after the transaction, the database will be in a consistent state
- **Isolation** → operations cannot access data that is currently modified
- **Durability** → data will not be lost upon completion of a transaction

NoSQL: Concepts

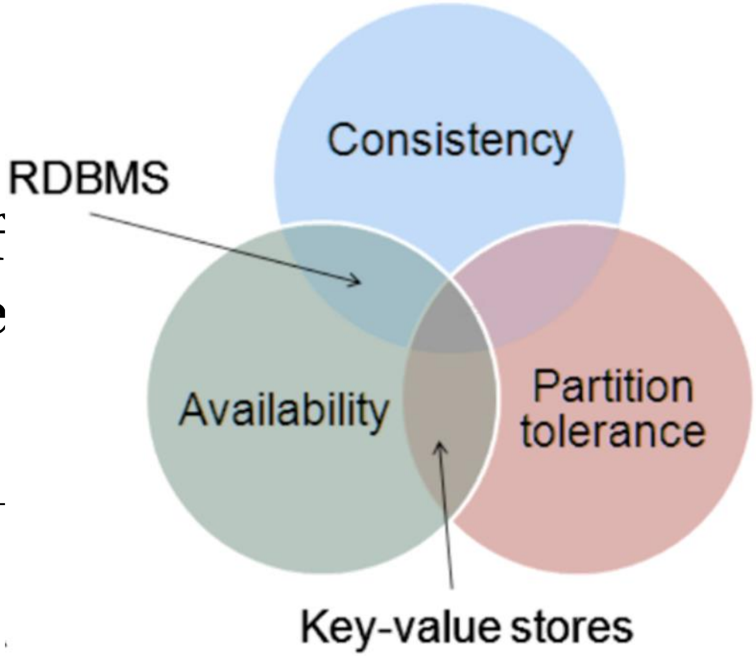
BASE: Basically Available, Soft State, Eventual Consistency [Fox]

- **Basically Available** → an application works basically all the time (despite partial failures)
- **Soft State** → is in flux and non-deterministic (changes all the time)
- **Eventual Consistency** → will be in some consistent state (at some time in future)

NoSQL: Concepts

CAP Theorem: Consistency, Availability, Partition Tolerance[Brewer]

- (Positive) consequence: we can concentrate on two challenges
- **ACID** properties needed to guarantee consistency and availability
- **BASE** properties come into play if availability and partition tolerance is favored



NoSQL: Techniques

Basic techniques (widely applied in NoSQL systems)

- distributed data storage, replication (how to distribute the data) → Consistent hashing
- distributed query strategy (horizontal scalability) → MapReduce (in the MapReduce lecture)
- recognize order of distributed events and potential conflicts → Vector clock (later in this lecture)

NoSQL: Techniques – Consistent Hashing [Karger]

Task

- find machine that stores data for a specified key k
- trivial hash function to distribute data on n nodes: h(k; n) = k mod n
- if number of nodes changes, all data will have to be redistributed!

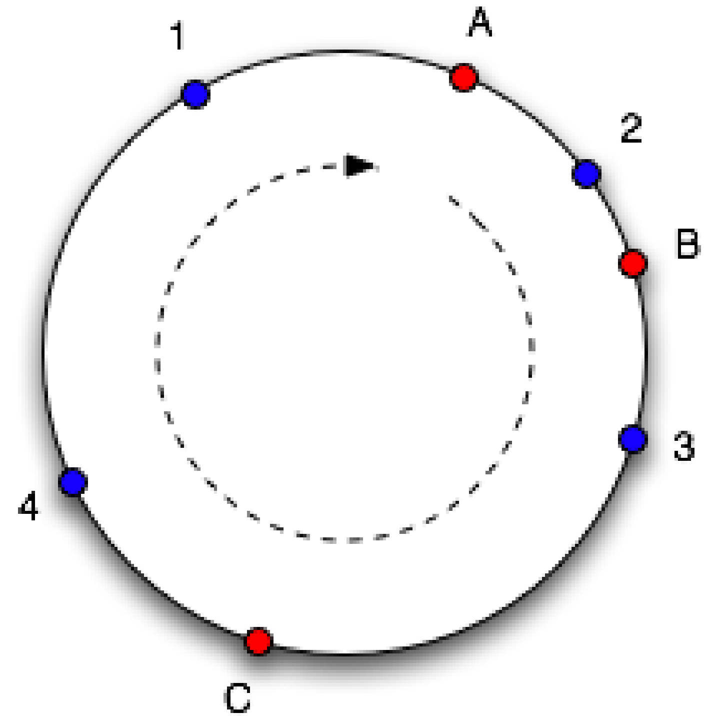
Challenge

- minimize number of nodes to be copied after a configuration change
- incorporate hardware characteristics into hashing model

NoSQL: Techniques – Consistent Hashing [Karger]

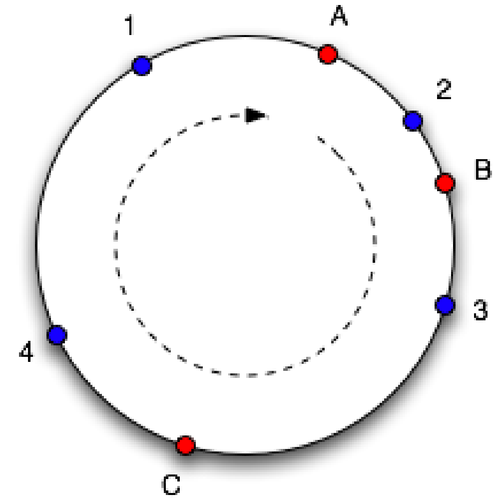
Basic idea

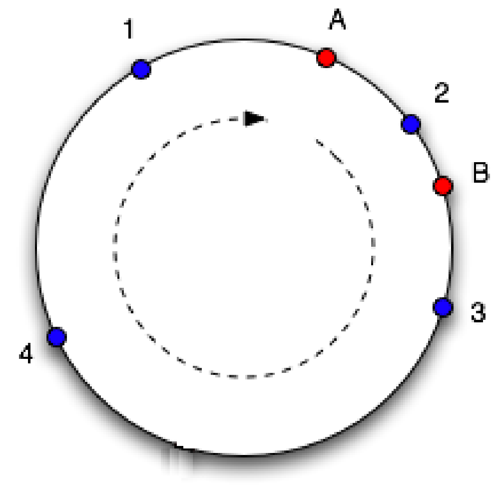
- arrange the nodes in a ring and each node is in charge of the hash values in the range between its neighbor node
- include hash values of all nodes in hash structure
- calculate hash value of the key to be added/retrieved
- choose node which occurs next clockwise in the ring



NoSQL: Techniques – Consistent Hashing [Karger]

- include hash values of all nodes in hash structure
- calculate hash value of the key to be added/retrieved
- choose node which occurs next clockwise in the ring
- if node is dropped or gets lost, missing data is redistributed to adjacent nodes (replication issue)

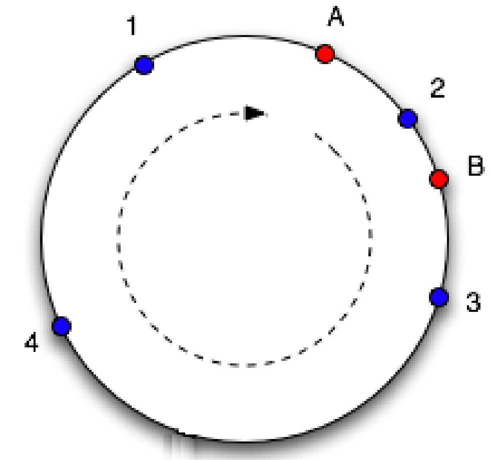


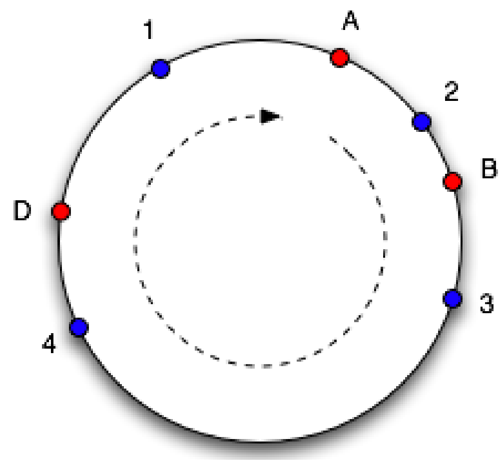


NoSQL: Techniques – Consistent Hashing [Karger]

- if a new node is added, its hash value is added to the hash table
- the hash realm is repartitioned, and hash data will be transferred to new neighbor

→ no need to update remaining nodes!





NoSQL: Techniques – Consistent Hashing [Karger]

- A replication factor r is introduced: not only the next node but the next r nodes in clockwise direction become responsible for a key
- Number of added keys can be made dependent on node characteristics (bandwidth, CPU, ...)

NoSQL: Techniques – Logical Time

Challenge

- recognize order of distributed events and potential conflicts
- most obvious approach: attach timestamp (ts) of system clock to each

event e → ts(e)

→ error-prone, as clocks will never be fully synchronized

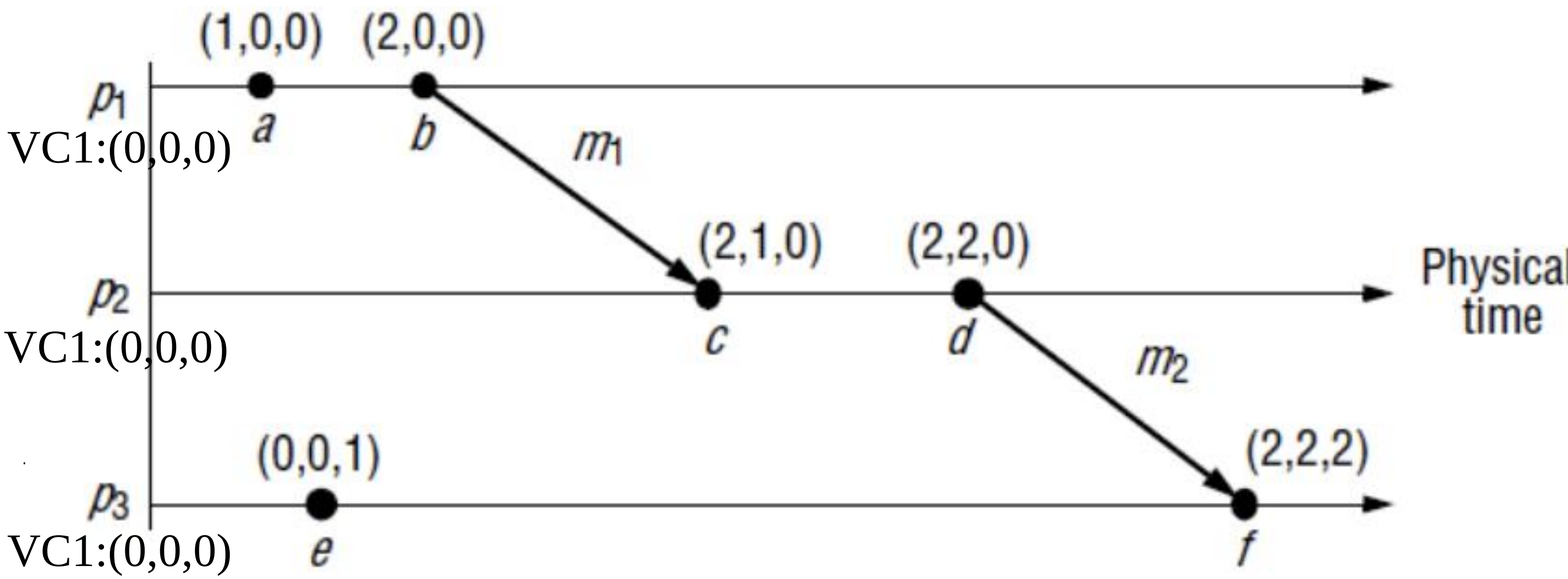
→ insufficient, as we cannot catch causalities (needed to detect conflicts)

NoSQL: Techniques – Vector Clock[Coulouris]

- A vector clock for a system of N nodes is an array of N integers.
- Each process keeps its own vector clock, V_i , which it uses to timestamp local events.
- Processes piggyback vector timestamps on the messages they send to one another, and there are simple rules for updating the clocks:
  - VC1: Initially, $V_i [j] = 0$, for $i , j = 1, 2, \dots N$
  - VC2: Just before p_i timestamps an event, it sets $V_i [i] := V_i [i] + 1$
  - VC3: p_i includes the value $t = V_i$ in every message it sends
  - VC4: When p_i receives a timestamp t in a message, it sets $V_i [j] := \max(V_i [j]; t [j])$, for $j = 1, 2, \dots N$

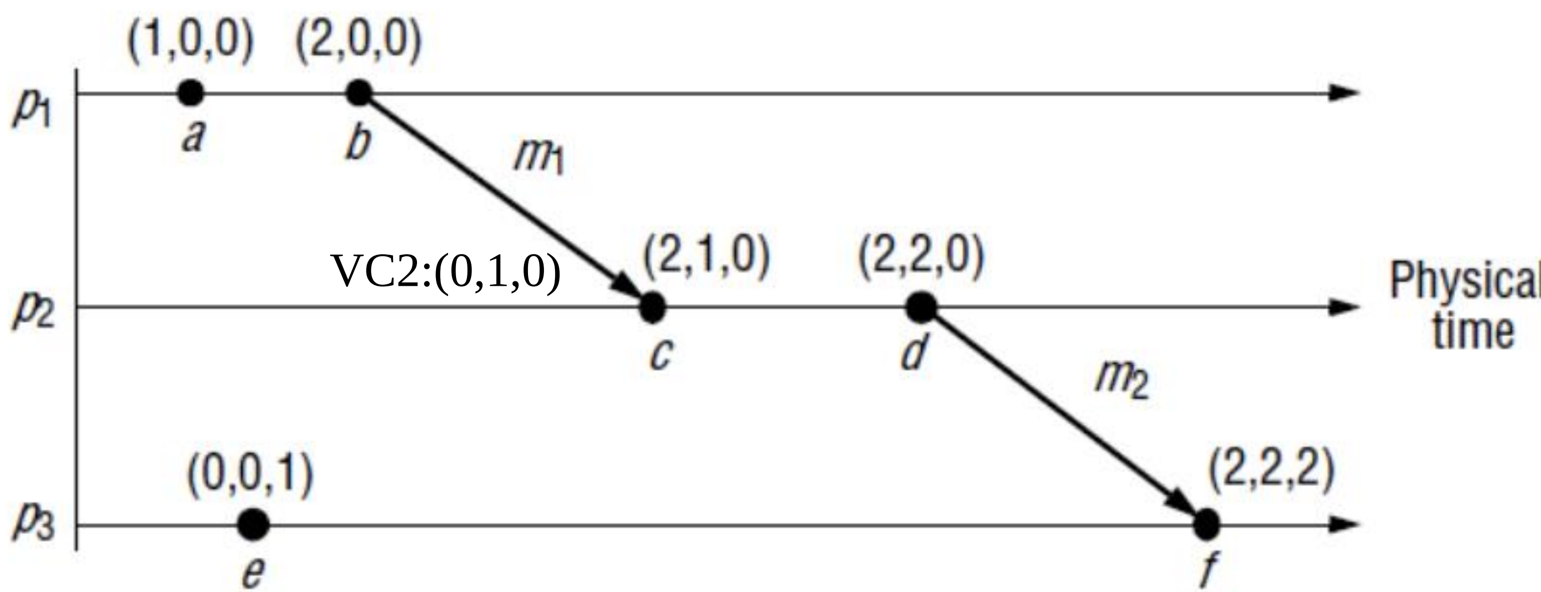
NoSQL: Techniques – Vector Clock[Coulouris]

- VC1: Initially, V_i [j] = 0, for i , j = 1, 2, … N
- VC2: Just before p_i timestamps an event, it sets V_i [i] := V_i [i] + 1



NoSQL: Techniques – Vector Clock[Coulouris]

- VC3: p_i includes the value t = V_i in every message it sends
- VC4: When p_i receives a timestamp t in a message, it sets V_i [j] := max(V_i [j]; t [j]), for j = 1, 2, … N



NoSQL: Techniques – Vector Clock[Coulouris]

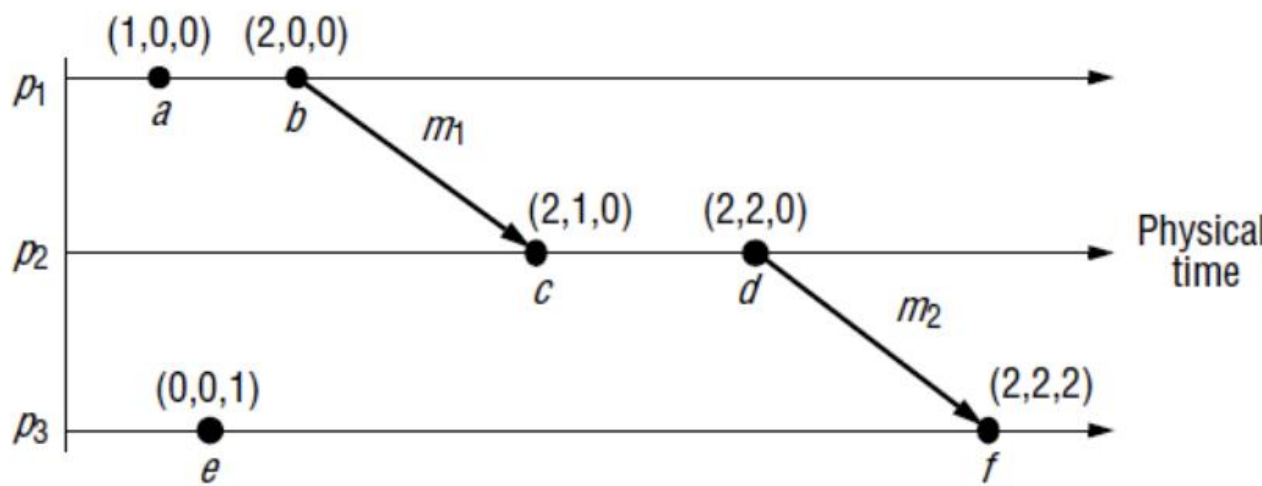
Properties:

- $V = V'$ iff $V[j] = V'[j]$ for $j = 1, 2, \dots N$
- $V \leq V'$ iff $V[j] \leq V'[j]$ for $j = 1, 2, \dots N$
- $V < V'$ iff $V \leq V'$ and $V \neq V'$

two events e and e': that $e \rightarrow e' \leftrightarrow V(e) < V(e')$

$\rightarrow$ Conflict detection! ($c \parallel e$ since neither $V(c) \leq V(e)$ nor $V(e) \leq V(c)$)

c & e are concurrent



NoSQL Systems – Types and Applications

NoSQL Classification Dimensions[HBase]

- Data model – how the data is stored
- Storage model – in-memory vs persistent
- Consistency model – strict, eventual consistent, etc.
  - Affects reads and writes requests
- Physical model – distributed vs single machine
- Read/Write performance – what is the proportion between reads and writes
- Secondary indexes - sort and access tables based on different fields and sorting orders

NoSQL Classification Dimensions[HBase]

- Failure handling – how to address machine failures
- Compression – result in substantial savings in raw storage
- Load balancing – how to address high read or write rate
- Atomic read-modify-write – difficult to achieve in a distributed system
- Locking, waits and deadlocks – locking models and version control

NoSQL Data Models

- Key-Value Stores
- Document Stores
- Column-Family Stores
- Graph Databases

- Impacts application, querying, scalability

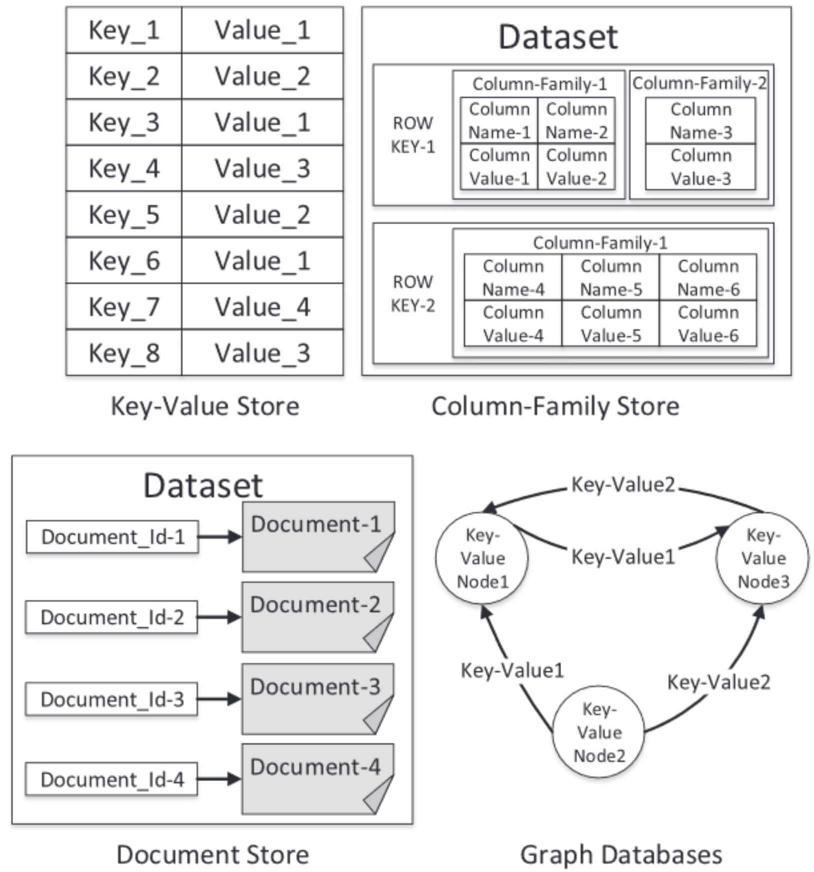


figure from [DataMan]

DBs not referred as NoSQL

- Object DBs
- XML DBs
- Special purpose DBs
  - Stream processing

Key-Value Stores[DataMan]

Key_1	Value_1
Key_2	Value_2
Key_3	Value_1
Key_4	Value_3
Key_5	Value_2
Key_6	Value_1
Key_7	Value_4
Key_8	Value_3

Key-Value Store

- Schema-free
  - Keys are unique
  - Values of arbitrary types
- Efficient in storing distributed data
- (very) Limited query facilities and indexing
  - get(key), put(key, value)
  - Value → opaque to the data store → no data level querying and indexing

Key-Value Stores[DataMan]

Key_1	Value_1
Key_2	Value_2
Key_3	Value_1
Key_4	Value_3
Key_5	Value_2
Key_6	Value_1
Key_7	Value_4
Key_8	Value_3

Key-Value Store

- Types
  - In-memory stores – Memcached, Redis
  - Persistent stores – BerkeleyDB, Voldemort, RiakDB
- Not suitable for
  - structures and relations
  - accessing multiple items (since the access is by key and often no transactional capabilities)

Key_1	Value_1
Key_2	Value_2
Key_3	Value_1
Key_4	Value_3
Key_5	Value_2
Key_6	Value_1
Key_7	Value_4
Key_8	Value_3

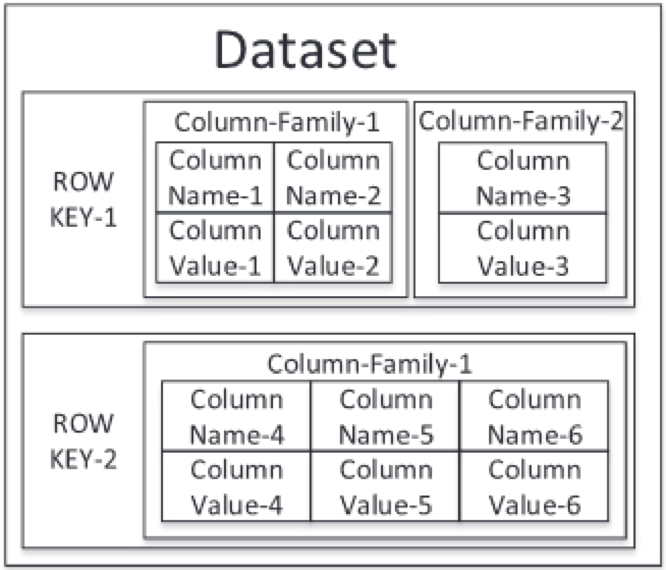
Key-Value Store

Key-Value Stores[DataMan]

- Applications:
  - Storing web session information
  - User profiles and configuration
  - Shopping cart data
  - Using them as a caching layer to store results of expensive operations (create a user-tailored web page)

Column-Family Stores[DataMan]

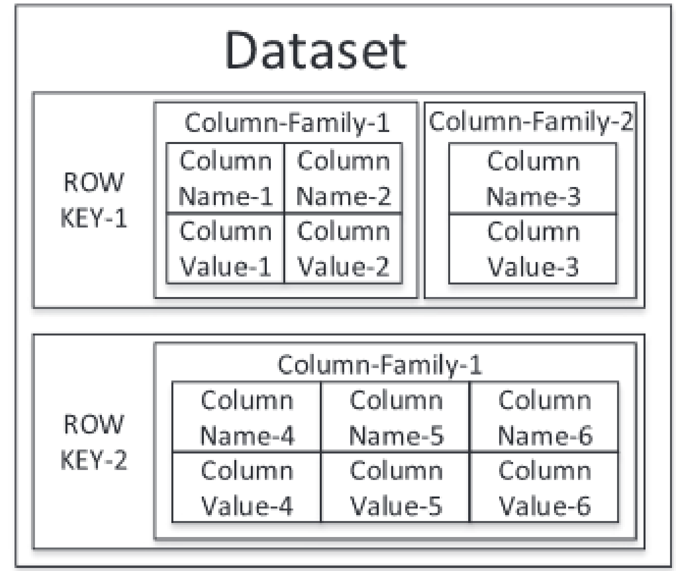
- Schema-free
  - Rows have unique keys
  - Values are varying column families and act as keys for the columns they hold
  - Columns consist of key-value pairs



Column-Family Store

- Better than key-value stores for querying and indexing

Column-Family Stores[DataMan]

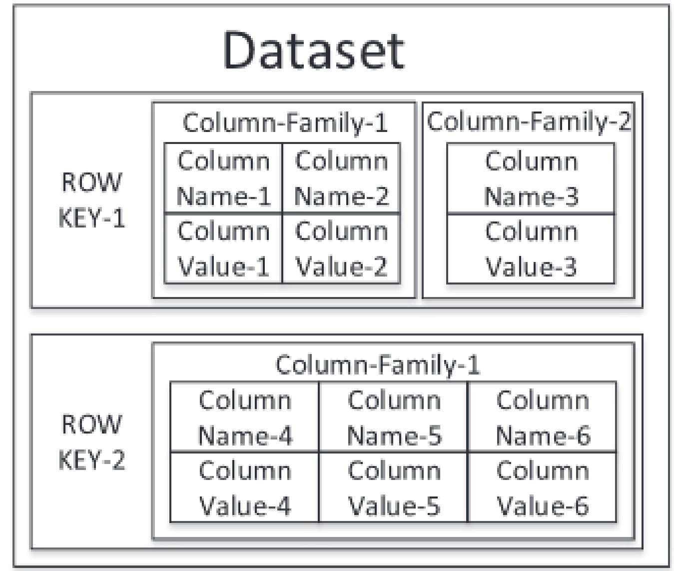


Column-Family Store

- Types
  - Googles BigTable, Hadoop HBase
  - No column families – Amazon SimpleDB, DynamoDB
  - Supercolumns - Cassandra
- Not suitable for
  - structures and relations
  - highly dynamic queries (HBase and Cassandra)

Column-Family Stores[DataMan]

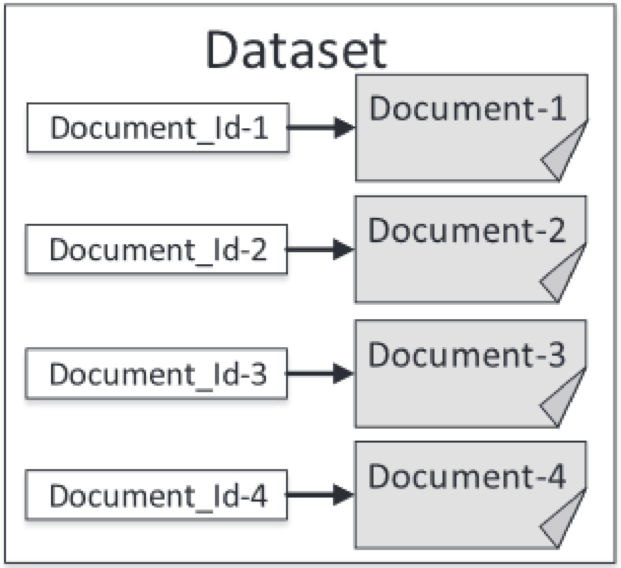
- Applications:
  - Document stores applications
  - Analytics scenarios – HBase and Cassandra
    - Web analytics
    - Personalized search
    - Inbox search



Column-Family Store

Document Stores[DataMan]

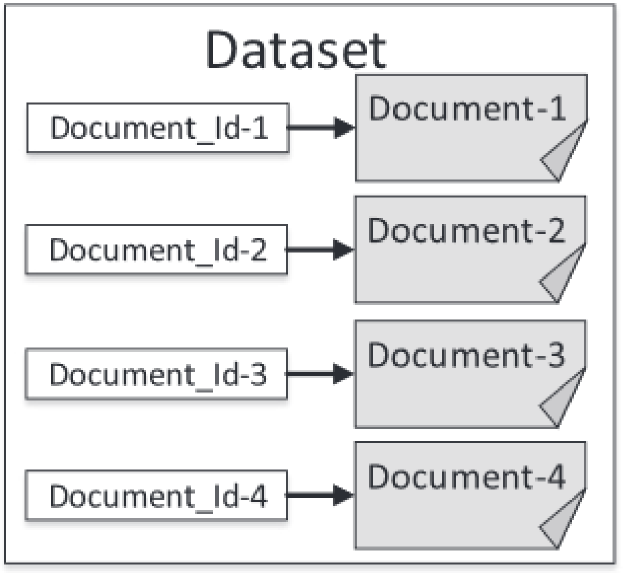
- Schema-free
  - Keys are unique
  - Values are documents – complex (nested) data structures in JSON, XML, binary (BSON), etc.
- Indexing and querying based on primary key and content
- The content needs to be representable as a document
- MongoDB, CouchDB, Couchbase



Document Store

Document Stores[DataMan]

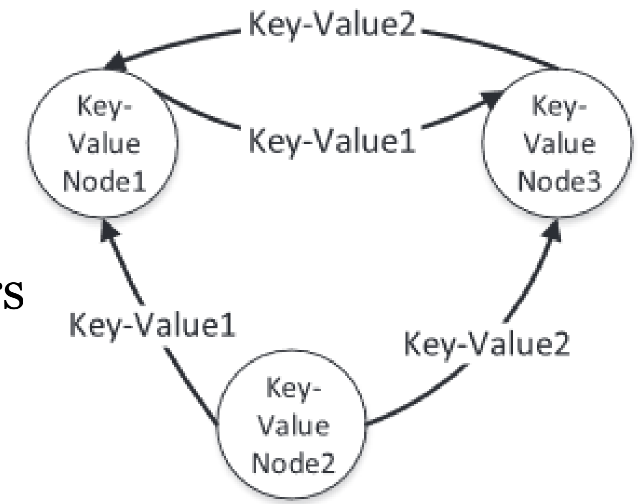
- Applications:
  - Items with similar nature but different structure
  - Blogging platforms
  - Content management systems
  - Event logging
  - Fast application development



Document Store

Graph Databases[DataMan]

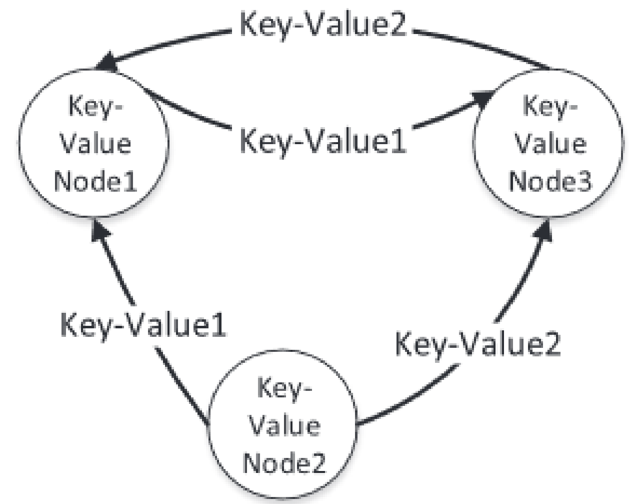
- Graph model
  - Nodes/vertices and links/edges
  - Properties consisting of key-value pairs
- Suitable for very interconnected data since they are efficient in traversing relationships
- Not as efficient
  - as other NoSQL solutions for non-graph applications
  - horizontal scaling
- Neo4J, HyperGraphDB



Graph Databases

Graph Databases[DataMan]

- Applications:
  - location-based services
  - recommendation engines
  - complex network-based applications
    - social, information, technological, and biological network
  - memory leak detection



Graph Databases

Big Data Analytics Stack

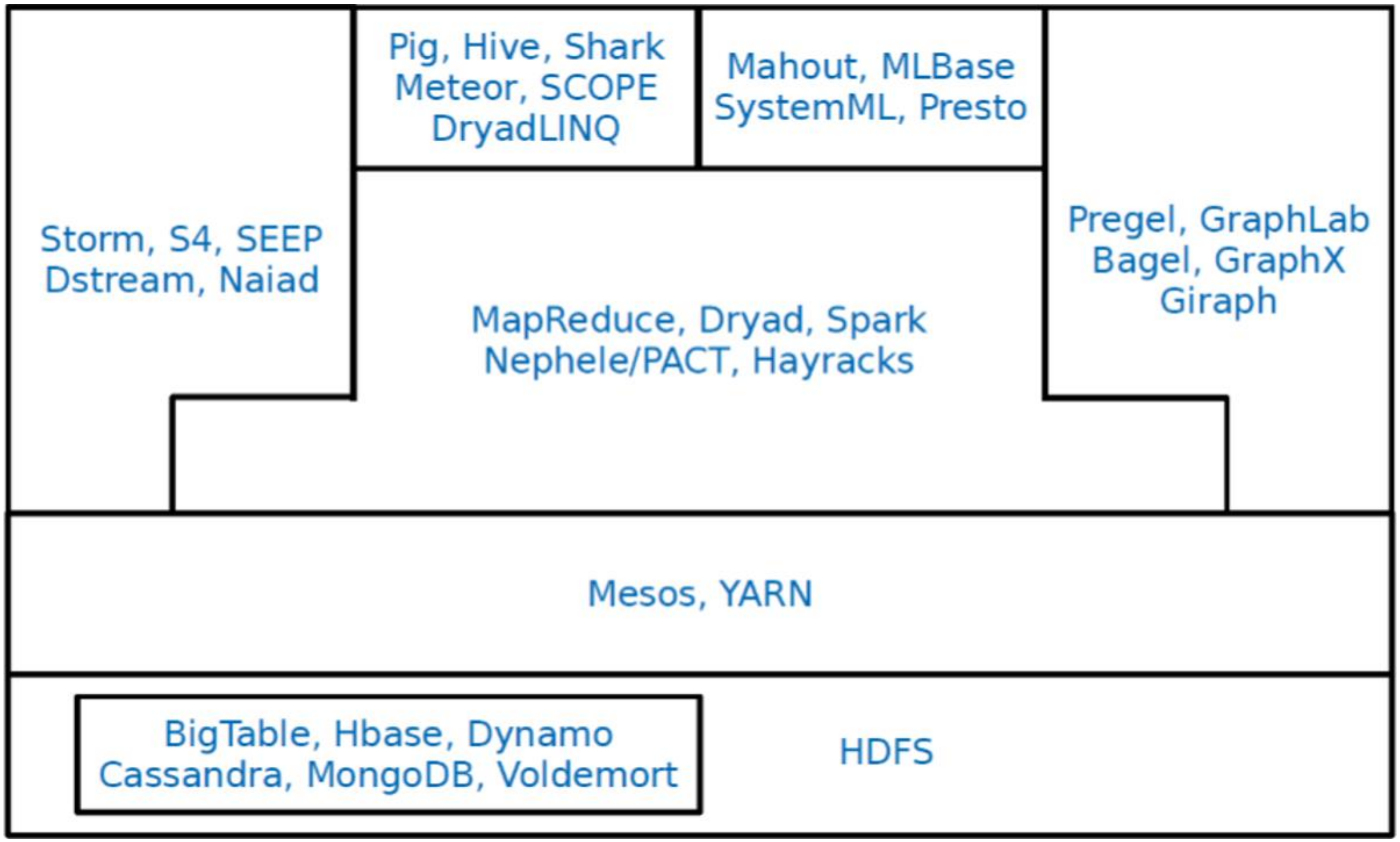


figure from: https://www.sics.se/~amir/dic.htm

HDFS[Hadoop][HDFS][HDFSpaper]

Hadoop Distributed File System

Compute Nodes[Massive]

- Compute node – processor, main memory, cache and local disk
- Organized into racks
- Intra-rack connection typically gigabit speed
- Inter-rack connection slower by a small factor

HDFS (Hadoop Distributed File System)

- Runs on top of the native file system
  - Files are very large divided into 128 MB chunks/blocks
    - To minimize the cost of seeks
  - Caching blocks is possible
  - Single writer, multiple readers
  - Exposes the locations of file blocks via API
  - Fault tolerance and availability to address disk/node failures
    - Usually replicated three times on different compute nodes
- Based on GFS (Google File System - proprietary)

HDFS is Good for …

- Store very large files – GBs and TBs
- Streaming access
  - Write-once, read many times
  - Time to read the entire dataset is more important than the latency in reading the first record.
- Commodity hardware
  - Clusters are built from commonly available hardware
  - Designed to continue working without a noticeable interruption in case of failure

HDFS is currently Not Good for …

- Low-latency data access
  - HDFS is optimized for delivering high throughput of data
- Lots of small files
  - the amount of files is limited by the memory of the namenode; blocks location is stored in memory
- Multiple writers and arbitrary file modifications
  - HDFS files are append only – write always at the end of the file

HDFS Organization

- Namenode (master)
  - Manages the filesystem namespace and metadata
  - Stores in memory the location of all blocks for a given file
- Datanodes (workers)
  - Store and retrieve blocks
  - Send heartbeat to the namenode
- Secondary namenode
  - Periodically merges the namespace image with the edit log
  - **Not** a backup for a namenode, only a checkpoint

HDFS Organization

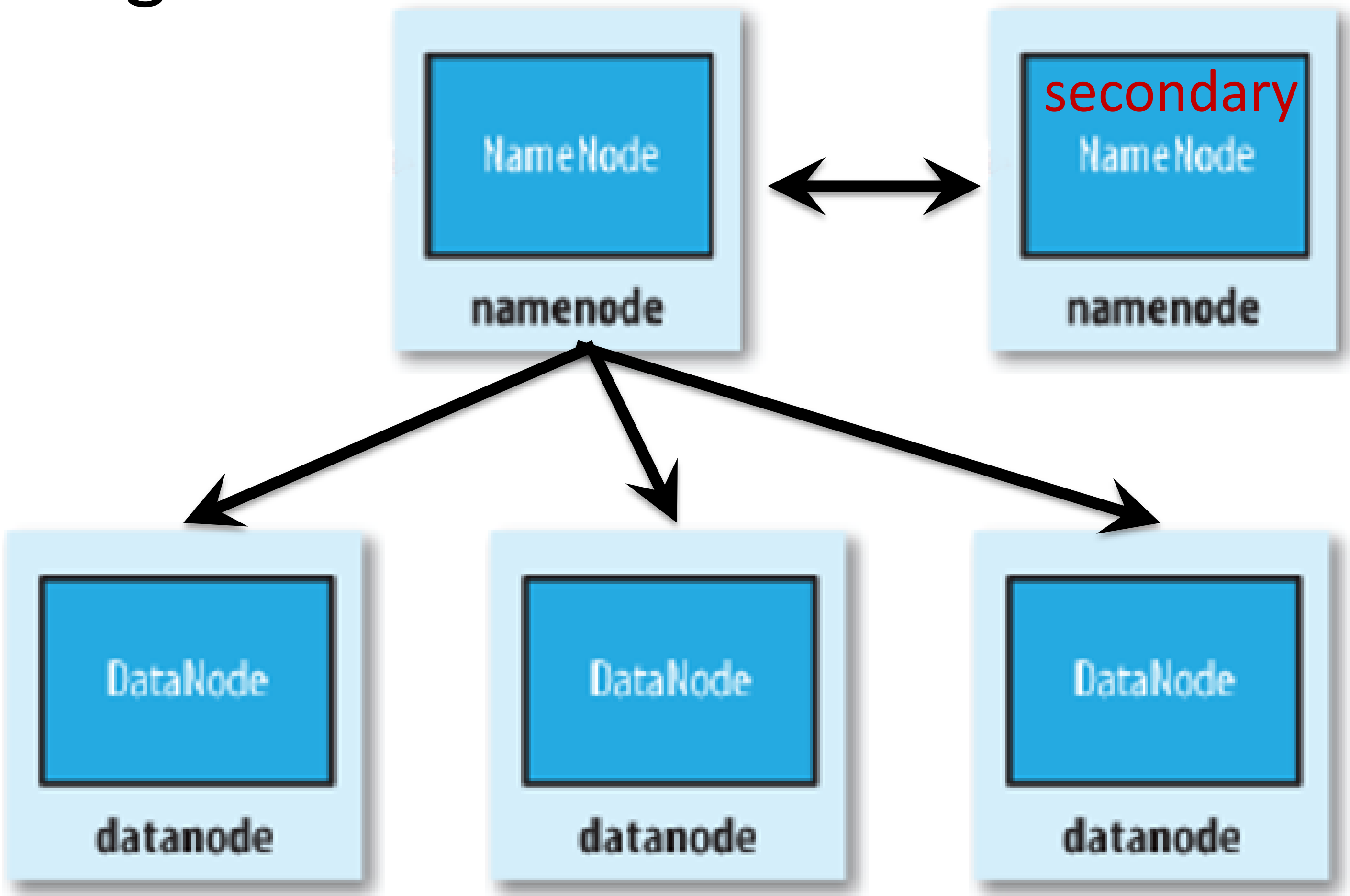
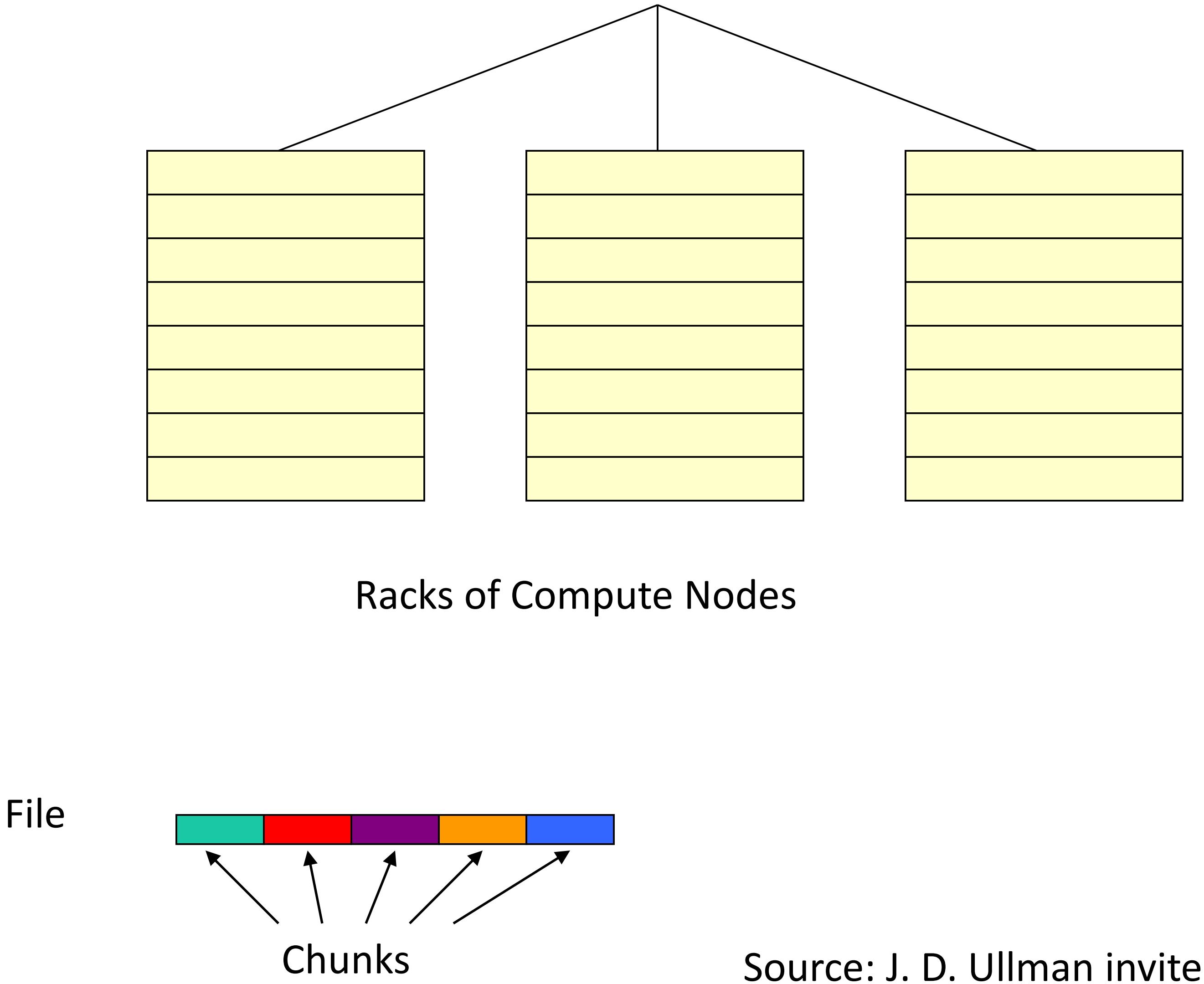


figure based on a figure from [Hadoop]

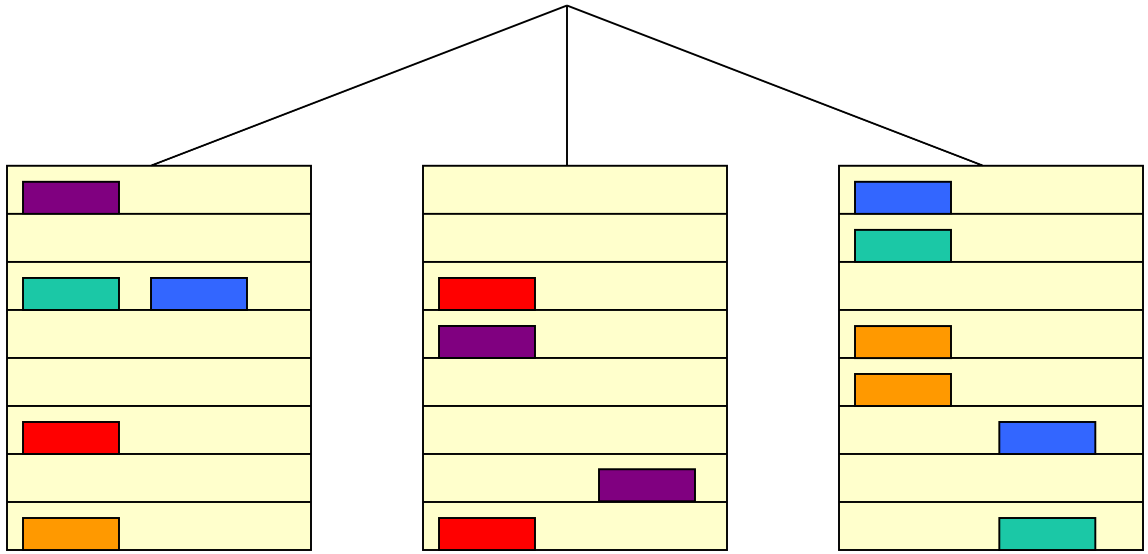
Block Placement and Replication

- Aim – improve data reliability, availability and network bandwidth utilization
- Default replica placement policy
  - No Datanode contains more than one replica
  - No rack contains more than two replicas of the same block
- Namenode ensures the number of replicas is reached
- Balancer tool – balances the disk space usage
- Block scanner – periodically verifies checksums



Source: J. D. Ullman invited talk EDBT 2011

Default HDFS Block Placement Policy



- 1st replica located on the writer node
- 2nd and 3rd replicas on two different nodes in a different rack
- The other replicas are located on random nodes

HDFS – File Reads

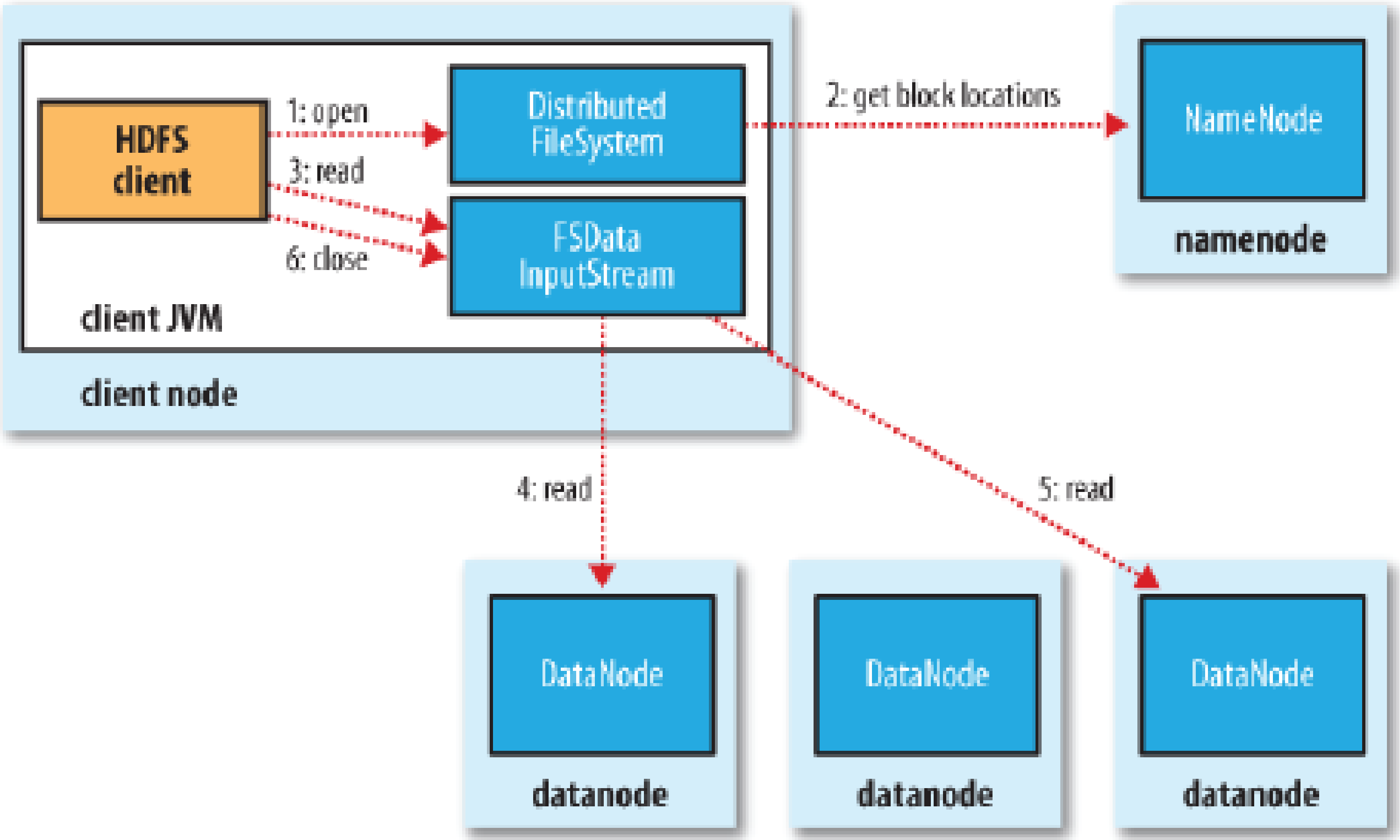


figure from [Hadoop]

HDFS – File Writes

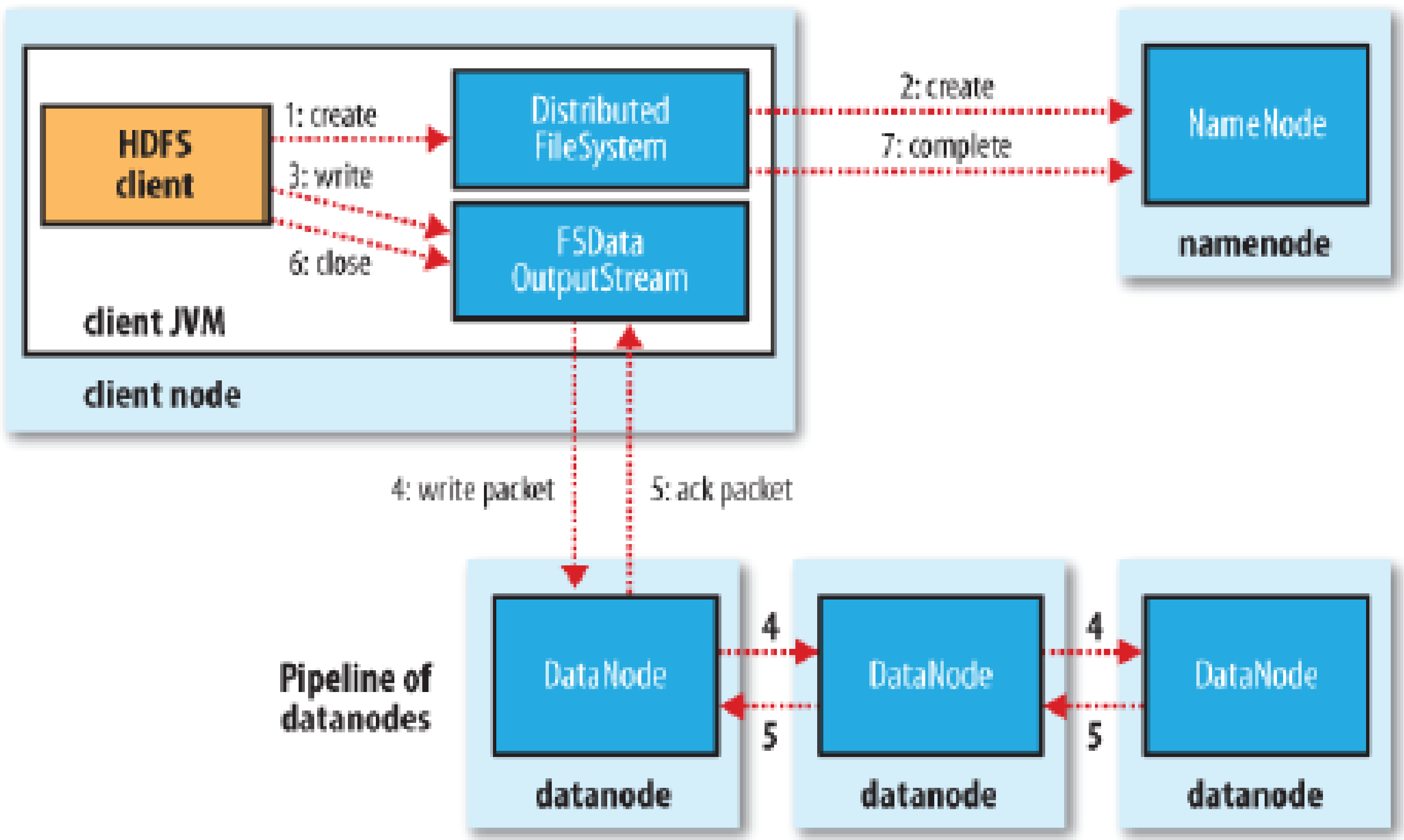


figure from [Hadoop]

HDFS – High Availability

- The namenode is single point of failure:
  - If a namenode crashes the cluster is down
- Secondary node
  - periodically merges the namespace image with the edit log to prevent the edit log from becoming too large.
  - lags the state of the primary prevents data loss but does **not** provide high availability
  - time for cold start 30 minutes
- In practice, the case for planned downtime is more important

HDFS – High Availability

- Pair of namenodes in an active stand-by configuration:
  - Highly available shared storage for the shared edit log
  - Datanodes send block reports to all namenodes
  - Clients must provide transparent to the user mechanism to handle failover
  - The standby node takes checkpoints of the active namenode namespace instead of the secondary node

HDFS commands

- List all options for the hdfs dfs
  - `hdfs dfs -help`
  - `dfs` – run a filesystem command
- Create a new folder
  - `hdfs dfs -mkdir /BigDataAnalytics`
- Upload a file from the local file system to the HDFS
  - `hdfs dfs -put bigdata /BigDataAnalytics`

HDFS commands

- List the files in a folder
  - `hdfs dfs -ls /BigDataAnalytics`
- Determine the size of a file
  - `hdfs dfs -du -h /BigDataAnalytics/bigdata`
- Print the first 5 lines from a file
  - `hdfs dfs -cat /BigDataAnalytics/bigdata | head -n 5`
- Copy a file to another folder
  - `hdfs dfs –cp /BigDataAnalytics/bigdata /BigDataAnalytics/AnotherFolder`

HDFS commands

- Copy a file to a local filesystem and rename it
  - `hdfs dfs -get /BigDataAnalytics/bigdata bigdata_localcopy`
- Scan the entire HDFS for problems
  - `hdfs fsck /`
- Delete a file from HDFS
  - `hdfs dfs -rm /BigDataAnalytics/bigdata`
- Delete a folder from HDFS
  - `hdfs dfs -rm -r /BigDataAnalytics`

References

- A comparison between several NoSQL databases with comments and notes by Bogdan George Tudorica, Cristian Bucur
- nosql-databases.org
- Scalable SQL and NoSQL data stores by Rick Cattel
- [Brewer] Towards Robust Distributed Systems @ACM PODC'2000
- [12 years later] CAP Twelve Years Later: How the "Rules" Have Changed, Eric A. Brewer, @Computer Magazine 2012. https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed
- [Fox et al.] Cluster-Based Scalable Network Services @SOSP'1997
- [Karger et al.] Consistent Hashing and Random Trees @ACM STOC'1997
- [Coulouris et al.] Distributed Systems: Concepts and Design, Chapter: Time & Global States, 5th Edition
- [DataMan] Data Management in cloud environments: NoSQL and NewSQL data stores.

References

- NoSQL Databases - Christof Strauch – University of Stuttgart
- The Beckman Report on Database Research
- [Vogels] Eventually Consistent by Werner Vogels, doi:10.1145/1435417.1435432
- [Hadoop] Hadoop The Definitive Guide, Tom White, 2011
- [Hive] Hive - a petabyte scale data warehouse using Hadoop
- https://github.com/Prokopp/the-free-hive-book
- [Massive] Mining of Massive Datasets
- [HiveManual] https://cwiki.apache.org/confluence/display/Hive/LanguageManual
- [Shark] Shark: SQL and Rich Analytics at Scale
- [SparkSQLHistory] https://databricks.com/blog/2014/07/01/shark-spark-sql-hive-on-spark-and-the-future-of-sql-on-spark.html

References

- [HDFS] The Hadoop Distributed File System
- [Dynamo] Dynamo: Amazon's Highly Available Key-value Store, 2007
- [HBaseInFacebook] Apache hadoop goes realtime at Facebook
- [HBase] HBase The Definitive Guide, 2011
- [HDFSpaper] The Hadoop Distributed File System @MSST2010