Introduction to NoSQL

Gabriele Pozzani

April 24, 2013

Outline

- NoSQL
  - Definition
  - Categories
  - Related concepts
- Modeling NoSQL stores
- Properties
- Performance
  - Relational vs NoSQL DBMSs
- Applications

What does NoSQL mean?

- NoSQL stands for:
  - No Relational
  - No RDBMS
  - NonRel
  - **Not Only SQL**
- NoSQL is
  - An umbrella term for all databases and data stores that don't follow the RDBMS principles
    - A class of products
    - A collection of several (related) concepts about data storage and manipulation
  - Often related to large data sets

Where does NoSQL come from?

- Non-relational DBMSs are not new
- But NoSQL represents a new incarnation
  - Due to massively scalable Internet applications
  - Based on distributed and parallel computing
- Development
  - Starts with Google
    - First research paper published in 2003
  - Continues also thanks to Lucene's developers/Apache (Hadoop) and Amazon (Dynamo)
  - Then a lot of products and interests came from Facebook, Netflix, Yahoo, eBay, Hulu, IBM, and many more

NoSQL and big data

- NoSQL comes from Internet, thus it is often related to the “big data” concept
- How much big are “big data”?
  - Over few terabytes ($>10^{12} \approx 2^{40}$)
  - Enough to start spanning multiple storage units
- Challenges
  - Efficiently storing and accessing large amounts of data is difficult, even more considering fault tolerance and backups
  - Manipulating large data sets involves running immensely parallel processes
  - Managing continuously evolving schema and metadata for semi-structured and un-structured data is difficult

Why are RDBMS not suitable for big data?

- The context is Internet
- RDBMSs assume that data are
  - Dense
  - Largely uniform (structured data)
- Data coming from Internet are
  - Massive and sparse
  - Semi-structured or unstructured
- With massive sparse data sets, the typical storage mechanisms and access methods get stretched

NoSQL products' categories

- NoSQL products can be categorized in
  - Document databases
  - Key/value stores
  - Sorted ordered column-oriented stores
  - Graph databases
  - XML data stores

Document databases

- Documents
  - Loosely structured sets of key/value pairs in documents, e.g., XML, JSON, BSON
  - Encapsulate and encode data in some standard formats or encodings
  - Are addressed in the database via a unique key
  - Documents are treated as a whole, avoiding splitting a document into its constituent name/value pairs
- Allow documents retrieving by keys or contents
- Notable for:
  - MongoDB (used in FourSquare, Github, and more)
  - CouchDB (used in Apple, BBC, Canonical, Cern, and more)

Document databases, JSON

```
{
  “ApacheLogRecord”: {
    “ip”: “127.0.0.1”,
    “ident” : “-”,
    “http_user” : “frank”,
    “time” : “10/Oct/2000:13:55:36 -0700”,
    “request_line” : {
      “http_method” : “GET”,
      “url” : “/apache_pb.gif”,
      “http_vers” : “HTTP/1.0”,
    },
    “http_response_code” : “200”,
    “http_response_size” : “2326”,
    “referrer” : “http://www.example.com/start.html”,
    “user_agent” : “Mozilla/4.08 [en] (Win98; I ;Nav)”,
  }
}
```

Key/value stores

- Store data in a schema-less way
- Store data as maps
  - HashMaps or associative arrays
  - Provide a very efficient average running time algorithm for accessing data
- Notable for:
  - Couchbase (Zynga, Vimeo, NAVTEQ, ...)
  - Redis (Craiglist, Instagram, StackOverflow, flickr, ...)
  - Amazon Dynamo (Amazon, Elsevier, IMDb, ...)
  - Apache Cassandra (Facebook, Digg, Reddit, Twitter,...)
  - Voldemort (LinkedIn, eBay, …)
  - Riak (Github, Comcast, Mochi, ...)

Sorted ordered column-oriented stores

- Data are stored in a column-oriented way
  - Data efficiently stored
  - Avoids consuming space for storing nulls
  - Unit of data is a set of key/value pairs
    - Identified by “row-key”
    - Ordered and sorted based on row-key
  - Columns are grouped in column-families
  - Data isn’t stored as a single table but is stored by column-families
- Notable for:
  - Google's Bigtable (used in all Google's services)
  - HBase (Facebook, StumbleUpon, Hulu, Yahoo!, ...)

Column-oriented store example

row-key	time	name	location	preferences
	t9	first name=>"...", last name=>"..."	zip=>"..."	d/r=>"...", veg/non-veg=>"..."
	t8			
	t7			
	t5			

row-key	time	contents	anchor	mime
com.cnn.www	t9		cnnsi.com	
	t8		my.look.ca	
	t6	"<html>..."	my.look.ca	"text/html"
	t5	"<html>..."		
	t3	"<html>..."		

Dealing with big data and scalability

- Traditional DBMSs are best designed to run well on a “single” machine
  - Larger volumes of data/operations requires to upgrade the server with faster CPUs or more memory
    - Vertical scaling
- NoSQL solutions are designed to run on clusters of “low-specification” servers
  - Larger volumes of data/operations requires to add more machines to the cluster
    - Horizontal scaling

Horizontal scaling

- Horizontal scaling needs to handle distributed data
  - It's tricky
  - It involves tradeoffs between speed, scalability, fault tolerance, atomicity, and consistency
- One very used approach is the MapReduce algorithm design pattern
  - Pioneered by Google and adopted by many others

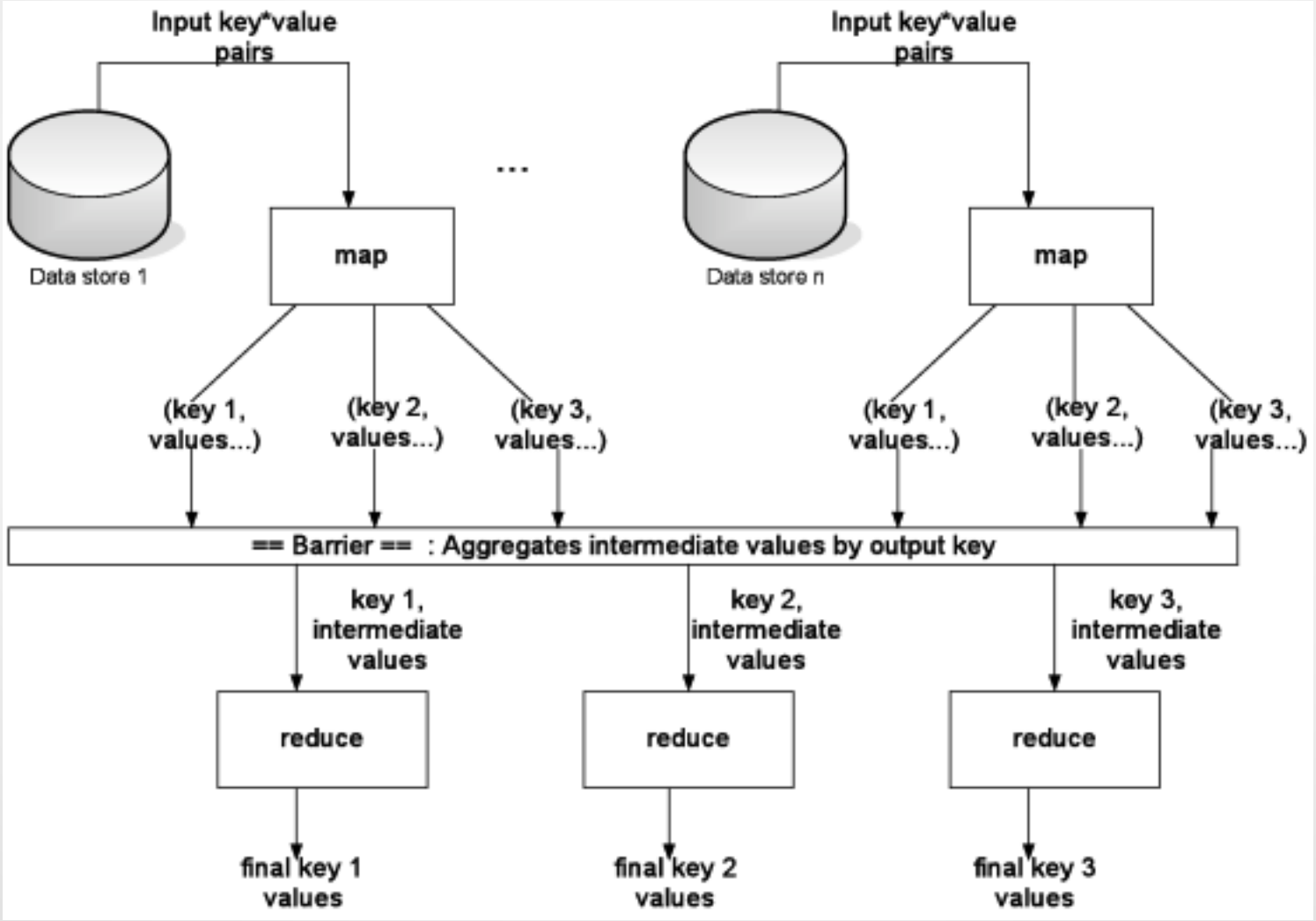
MapReduce (I)

- Originating from the functional programming world
- It consists of three steps
  1. A mapper function/script that goes through input data and outputs an unordered list of keys and values to use for calculating the final results
     - Keys are used to cluster data that will be needed to calculate a single output result
  2. A sort step that ensures that all data having same key are next to one another in the file/memory
     - Remove unplanned random accesses
  3. A reducer stage that goes through the sorted output and calculates result
     - It could involve aggregating values

MapReduce (II)

- Easily allows the process to be split across a large number of machines
- Each step runs on each node of a distributed cluster
  - Independently on the data subset local to the node
  - Isolation provides effective parallel processing
  - Values from reduce operations on different nodes are combined to get the final result
- It boosts performance even on single machines thanks to the increased locality of memory accesses

MapReduce (III)



Modeling NoSQL stores

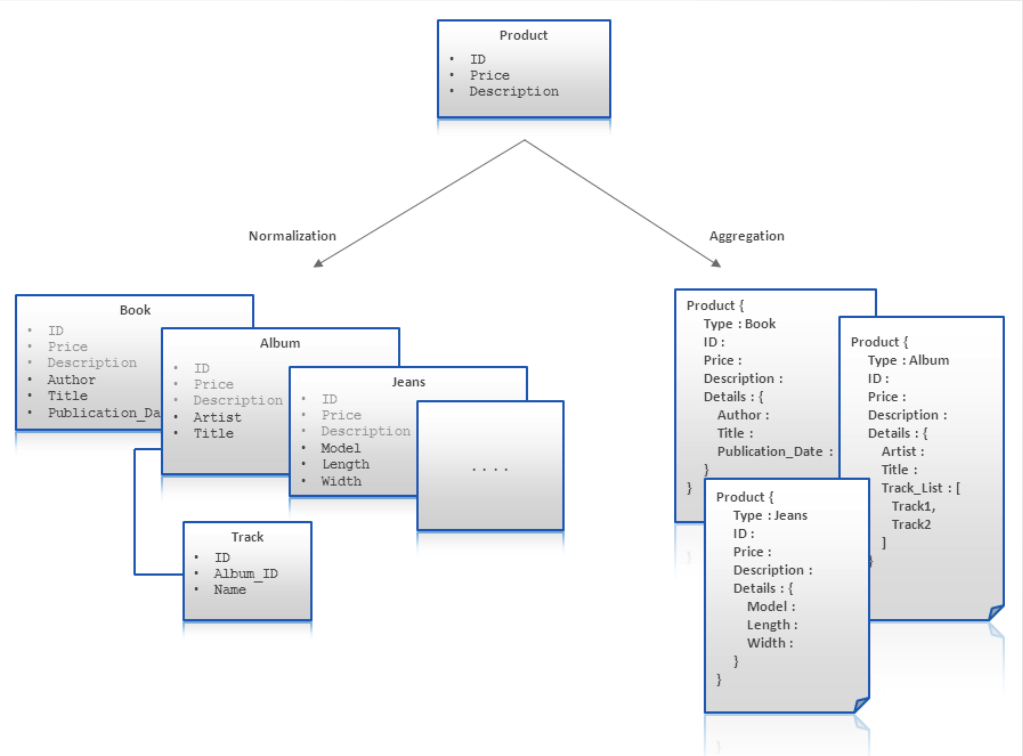
- NoSQL data modeling often starts from the application-specific queries as opposed to relational modeling:
  - Relational modeling is typically driven by the structure of available data. The main design theme is ”What answers do I have?”
  - NoSQL data modeling is typically driven by application-specific access patterns, i.e. the types of queries to be supported. The main design theme is ”What questions do I have?”
- Data duplication and denormalization are first-class citizens

Conceptual techniques (I)

1. Denormalization
   - the copying of the same data into multiple documents or tables in order to simplify/optimize query processing or to fit data into a particular model
2. Aggregates
3. Application Side Joins

Conceptual techniques (II)

1. Denormalization
2. Aggregates
   - Soft schema allows one to form classes of entities with complex internal structures (nested entities) and to vary the structure of particular entities.
3. Application Side Joins



Conceptual techniques (III)

1. Denormalization
2. Aggregates
3. Application Side Joins
   - Joins are rarely supported in NoSQL solutions
   - Joins are often handled at design time instead of at query execution time
   - When joins are inevitable, they should be handled by an application

Data modeling tecniques

- Atomic aggregates
- Dimensional reduction
- Index tables
- Composite key index
- …

Querying NoSQL stores

- Different NoSQL stores provide different querying tools and features
  - From “simple” filtering of data based on “columns” names/values (MongoDB, HBase, Redis, …)
  - To SQL-like languages (Google App Engine, HyperTable, Hive, ...)

NoSQL, No ACID

- RDBMSs are based on ACID (Atomicity, Consistency, Isolation, and Durability) properties
- NoSQL
  - Does not give importance to ACID properties
  - In some cases completely ignores them
- In distributed parallel systems it is difficult/impossible to ensure ACID properties
  - Even with a central coordinator
  - 2PL, 2PC and SS2PL can help
    - Long-running transactions don't work because keeping resources blocked for a long time is not practical

CAP Theorem

- A congruent and logical way for assessing the problems involved in assuring ACID-like guarantees in distributed systems is provided by the CAP theorem
  - At most two of the following three can be maximized at one time
    - Consistency - Each client has the same view of the data
    - Availability - Each client can always read and write
    - Partition tolerance - System works well across distributed physical networks
  - Conjectured by Eric Brewer in 2000
  - Proved by Seth Gilbert and Nancy Lynch in 2002

CAP theorem, consistency

- It alludes to atomicity and isolation
  - Gilbert & Lynch use the word "atomic" instead of consistent
- It means consistent reads and writes
  - Concurrent operations see the same valid and consistent data state
  - A service that is consistent operates fully or not at all
- “An atomic data object. Atomic, or linearizable, consistency is the condition expected by most web services today” (Gilbert & Lynch)

CAP theorem, availability

- It means the system is available to serve at the time when it’s needed
- If a system is not available to serve a request at the very moment it’s needed, it’s not available
- "Every request received by a non-failing node in the system must result in a response" (Gilbert & Lynch)

CAP theorem, partition tolerance

- A partition happens when, say, a network cable gets chopped, and Node A can no longer communicate with Node B
- Partition tolerance measures the ability of a system to continue to respond in the event a few of its cluster members become unavailable
- “No set of failures less than total network failure is allowed to cause the system to respond incorrectly” (Gilbert & Lynch)

Two out of three

- CAP theorem
  - At most two properties on three can be addressed
- The choices could be as follows:
  1. Availability is compromised but consistency and partition tolerance are preferred over it
  2. The system has little or no partition tolerance. Consistency and availability are preferred
  3. Consistency is compromised but systems are always available and can work when parts of it are partitioned

Option 1, compromising on availability

- Usual choice for transactional RDBMS under horizontal scaling
- Availability is affected by many factors, including:
  - Delays due to network lag
  - Communication bottlenecks
  - Resource starvation
  - Hardware failure leading to partitioning

Option 2, compromising on partition tolerance

- Modern interpretation of the theorem is: *during a network partition, a distributed system must choose either Consistency or Availability*
  - Because a system that is not Partition-tolerant will, by definition, be forced to give up Consistency or Availability during a partition
- In the Internet partitions are rare

Option 3, compromising on consistency

- Dropping strong consistency we get weak consistency
  - We accept “eventual consistency”
- Eventual consistency could be interpreted in two ways
  - Given a sufficiently long period of time, over which no updates are sent, one can expect that all updates will, eventually, propagate through the system and all the replicas will be consistent
  - In the presence of continuing updates, an accepted update eventually either reaches a replica or the replica retires from service
- Eventual consistency takes us to the BASE approach
  - Basically Available Soft-state Eventually consistent

Performance (I)

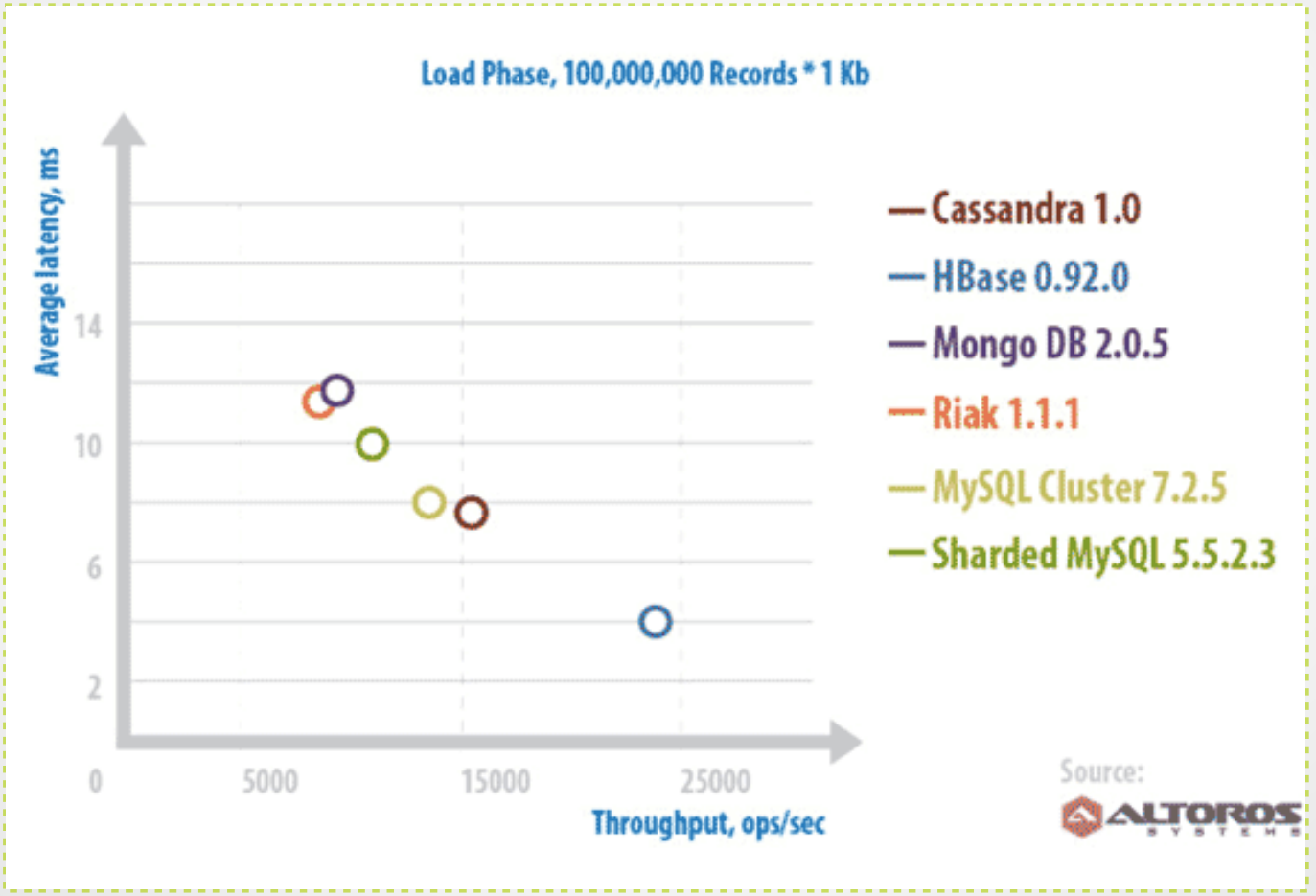
based on "A vendor-independent comparison of NoSQL databases: Cassandra, HBase, MongoDB, Riak", Sergey Bushik, October 2012.

- Tested DBMSs
  - Cassandra (column-oriented store)
  - HBase (column-oriented store)
  - MongoDB (document-oriented database)
  - Riak (key-value store)
  - MySQL Cluster (RDBMS)
  - sharded MySQL (RDBMS)

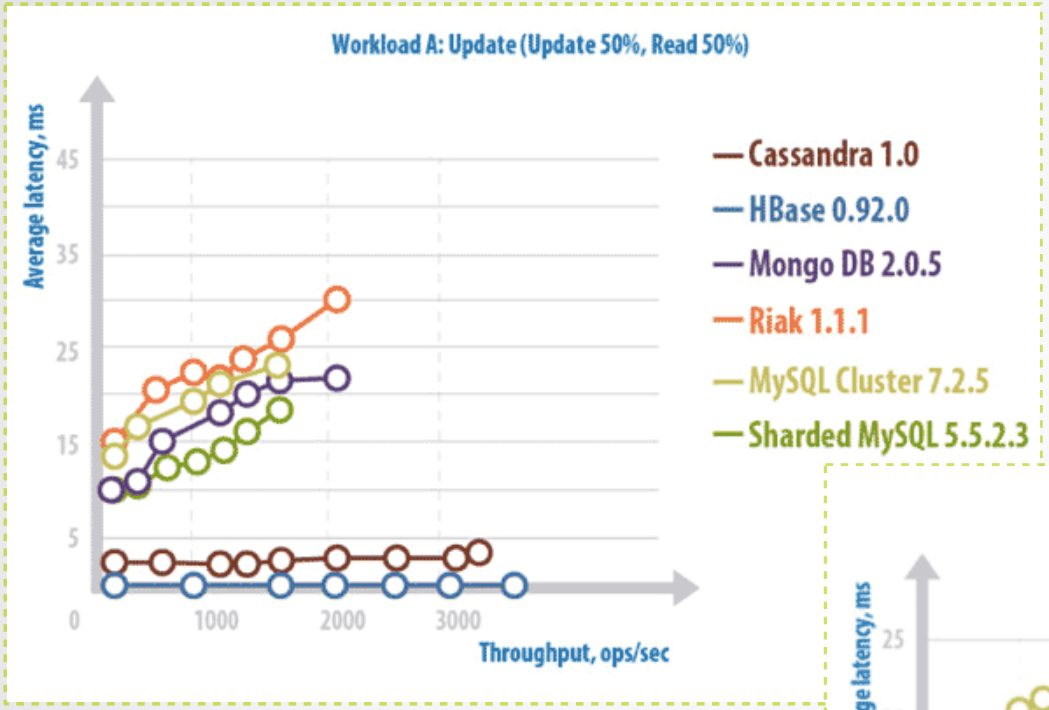
Performance (II)

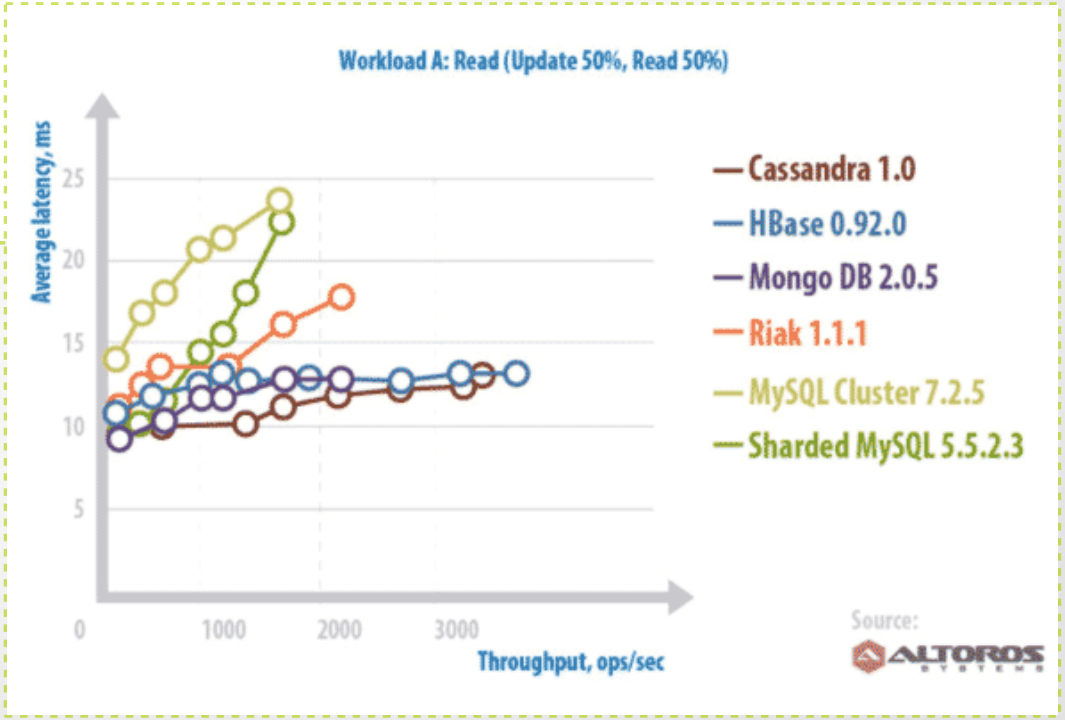
- Evaluated under different workloads:
  - Workload A: Update heavily
  - Workload B: Read mostly
  - Workload C: Read only
  - Workload D: Read latest
  - Workload E: Scan short ranges
  - Workload F: Read-modify-write
  - Workload G: Write heavily
- In each workload:
  - 100,000,000 records manipulated
  - The total size of a record equal to 1Kb
  - 10 fields of 100 bytes each per record
  - Multithreaded communications with the system (100 threads)

Performance, load

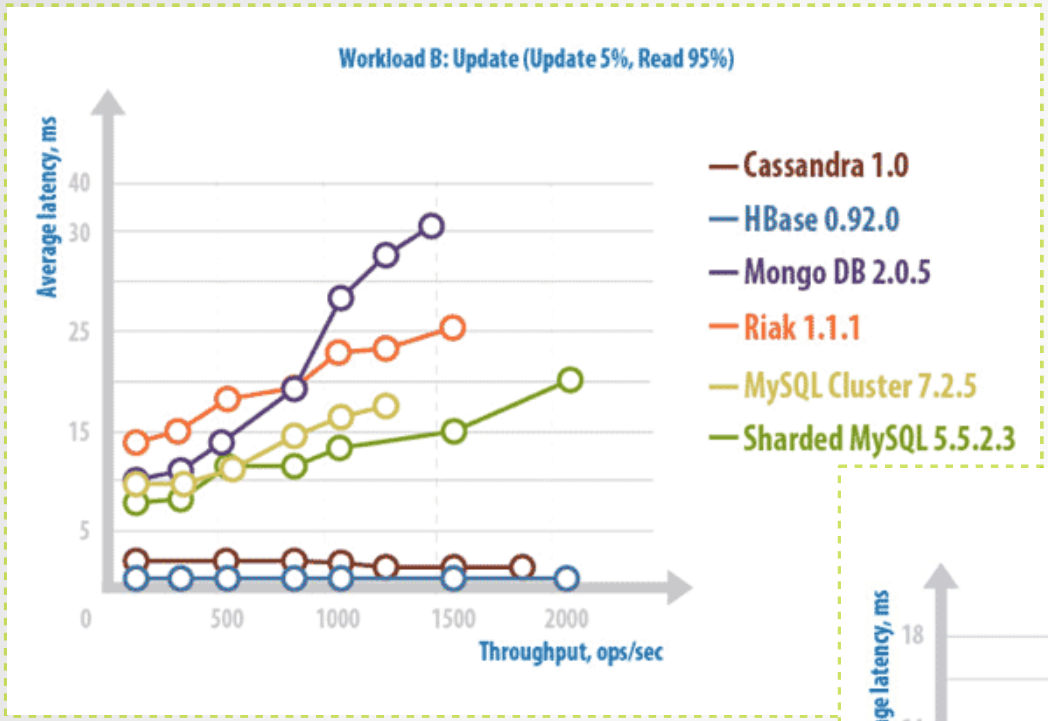


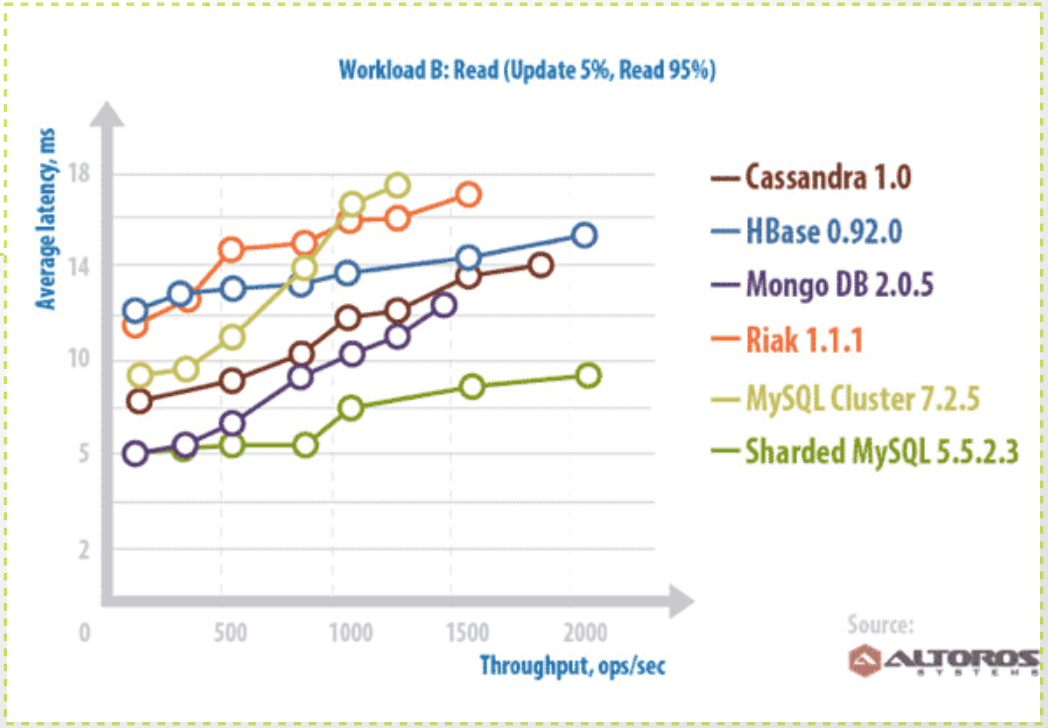
Performance, workload A



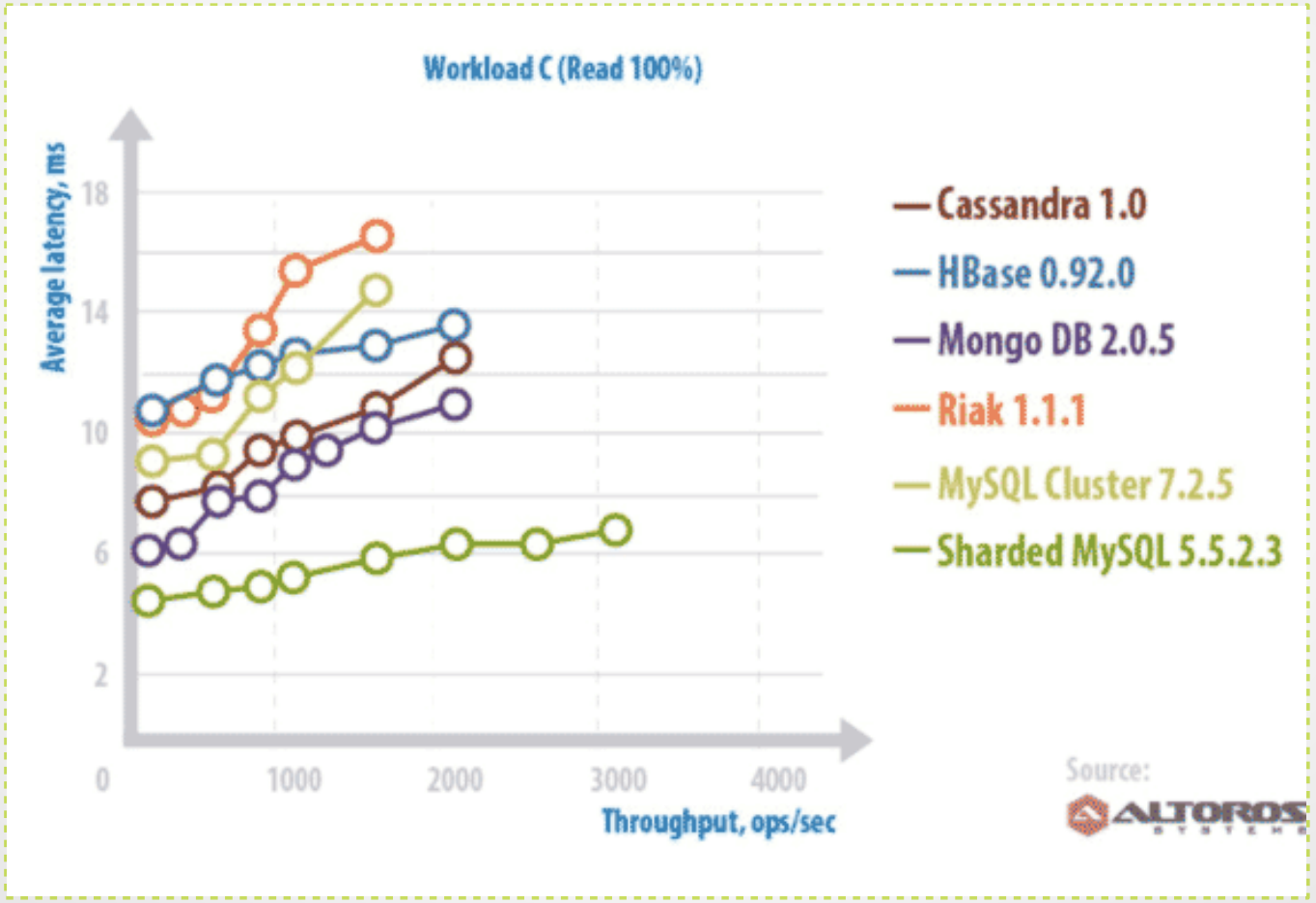


Performance, workload B

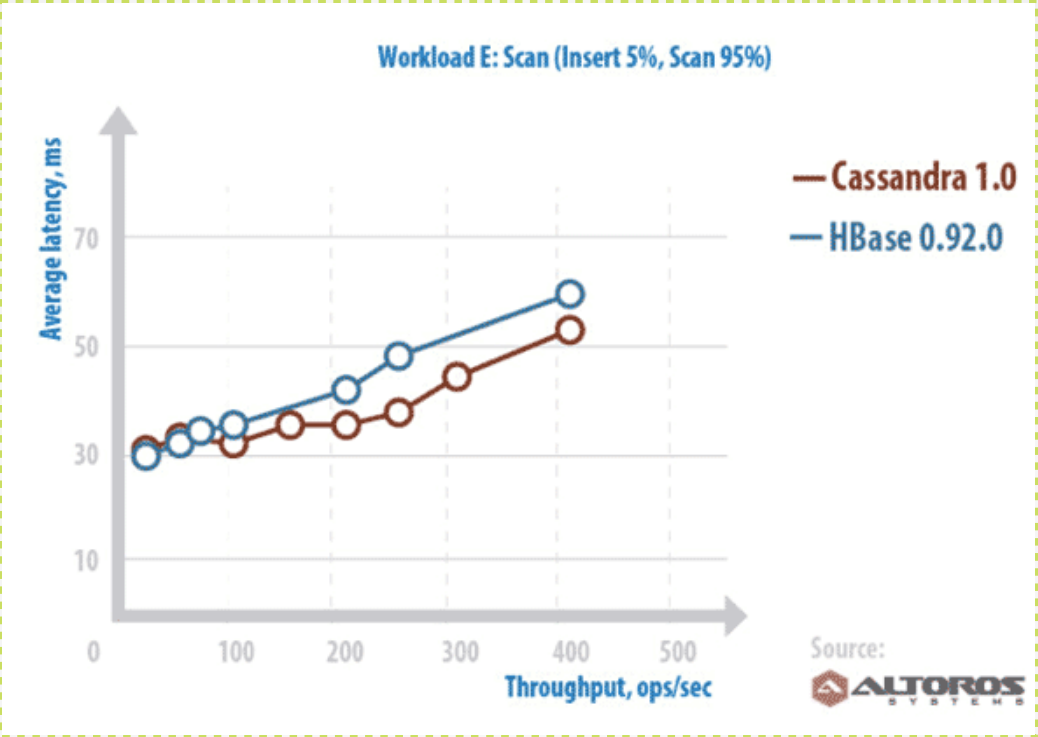




Performance, workload C

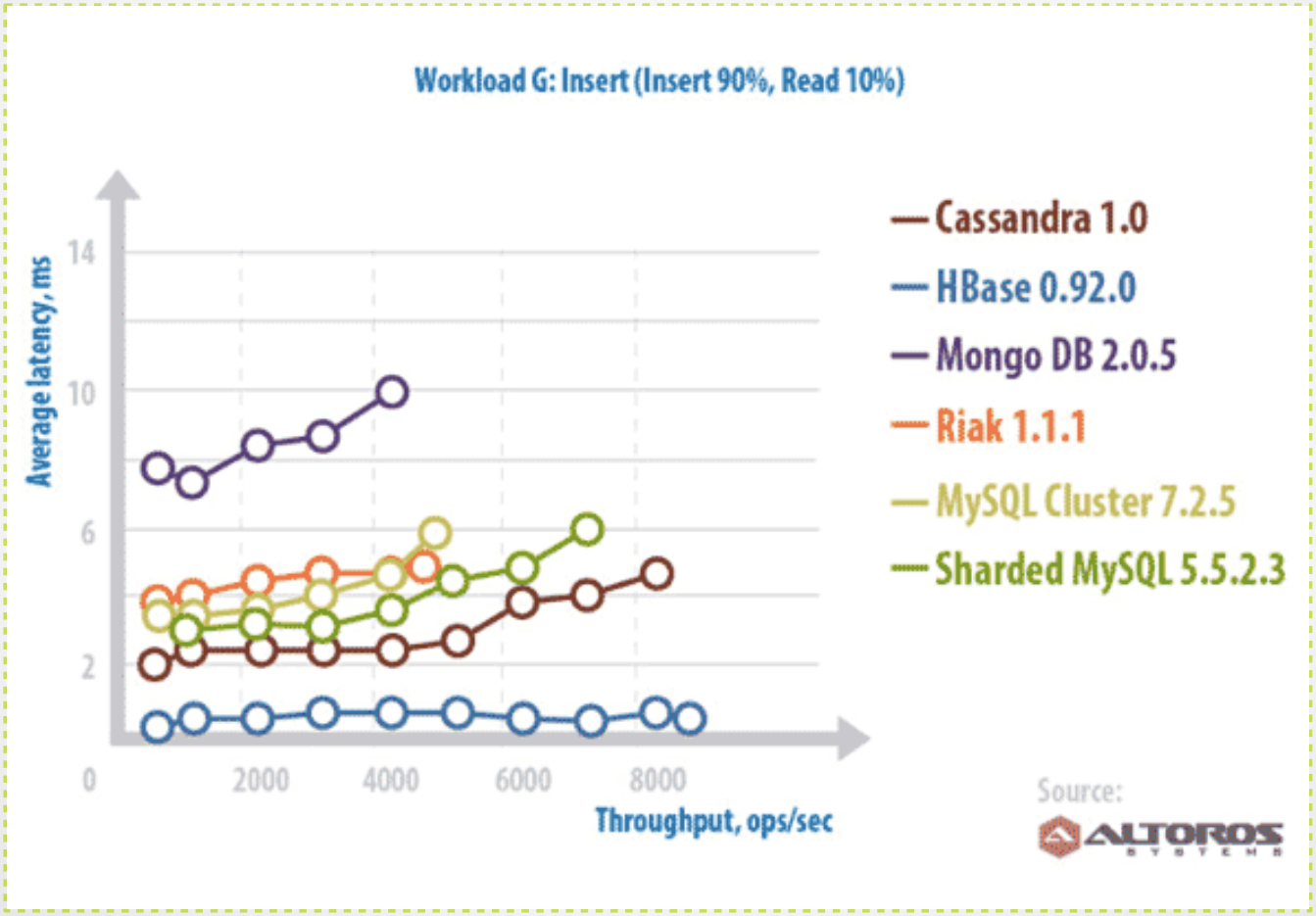


Performance, workload E



- MongoDB showed a maximum throughput of 20 ops/sec with a latency of ≈1sec
- MySQL Cluster was under 10ops/sec with a latency of 400ms
- Sharded MySQL showed near 40ops/sec with a latency of up to 400ms
- Riak does not support range scans

Performance, workload G



Performance, conclusions

- There is no perfect NoSQL database
- Every database has its advantages and disadvantages
  - Depending on the type of tasks (and preferences) to accomplish

Applications

- To DWHs
  - Pentaho Big Data
- To GIS

Application to DWH, Pentaho Big Data

- Pentaho developed some tools for big data analysis
  - Pentaho Big Data Analytics
- It provides ability to input, output, manipulate and report on data using Cassandra, Hadoop HDFS, Hadoop MapReduce, HBase, Hive, and MongoDB.
- Pentaho Instaview provides self-service analytics for the leading big data stores including Hadoop, Cassandra, HBase, MongoDB
  - Automatic data preparation (ETL, modeling)
- Extreme-scale in-memory caching using external data grid technologies such as Infinispan and Memcached

Application to GIS

- Some NoSQL solutions include support for geospatial data natively or with an extension
- Some NoSQL stores currently used to manage geospatial data include:
  - MongoDB (Foursquare, Where, Scrabbly)
  - BigTable (Google Earth)
  - Cassandra (SimpleGeo)
  - CouchDB/GeoCouch (City of Portland, Oregon's open GIS data initiative)
- Storing of POIs and GeoJSON documents

Application to GIS, Google Earth

- Google BigTable is a column-based store
  - sparse, distributed, persistent multi-dimensional sorted map

    `(row:string, column:string, time:int64) → string`
  - One table to preprocess data (70TB)
    - Each row in the table corresponds to a single geographic segment
      - Rows are named to ensure that adjacent segments are stored near
    - A column family to keep track of the sources of data for each segment
      - It has a large number of columns: one for each raw data image
      - Each segment is only built from a few images, this column family is very sparse

    and a different set of tables for indexing and serving client data (500GB)

Conclusions

- NoSQL is a set of concepts, ideas, technologies, and software dealing with
  - Big data
  - Sparse un/semi-structured data
  - High horizontal scalability
  - Massive parallel processing
- Different applications, goals, targets, approaches need different NoSQL solutions

References

- Tiwari, Shashank. *Professional NoSQL*. Wrox, 2011.
- Warden, Pete. *Big Data Glossary*. O'Reilly Media, 2011.
- Vogels, Werner (Amazon.com's CTO). *All Things Distributed. Werner Vogels' weblog on building scalable and robust distributed systems.* http://www.allthingsdistributed.com/
- Katsov, Ilya. *NoSQL Data Modeling Techniques.* http://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/
- Bushik, Sergey. *A vendor-independent comparison of NoSQL databases: Cassandra, HBase, MongoDB, Riak*. October 2012. Available online.
- Gilbert, Seth and Lynch, Nancy. *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services*. ACM SIGACT News 33.2 (2002): 51-59.
- Redmond, Eric, Wilson, Jim R. , and Carter, Jacquelyn. *Seven databases in seven weeks: a guide to modern databases and the NoSQL movement.* The Pragmatic Programmers, LLC, 2012.

Sharding

- Any DB distributed across multiple machines needs to know in what machine a piece of data is stored or must be stored
  - A sharding system makes this decision for each row, using its key
  - The main issue is to decide a rule for mapping any key to the corresponding shard
    - The rule must not introduce too much extra complexity and lookup cost
    - The rule must be flexible to adapt to changes in the cluster and data amount growing

MapReduce (IV)