NoSQL Databases

Keijo Heljanko

Department of Computer Science
University of Helsinki
keijo.heljanko@helsinki.fi

6.5-2019

Guest Lecturer

- Guest Lecturer: Prof. Keijo Heljanko, Department of Computer Science, University of Helsinki
  - Email: `keijo.heljanko@helsinki.fi`
- For more info into today's topic, attend the Aalto course: "CS-E4640 Big Data Platforms"

Business Drivers of Cloud Computing

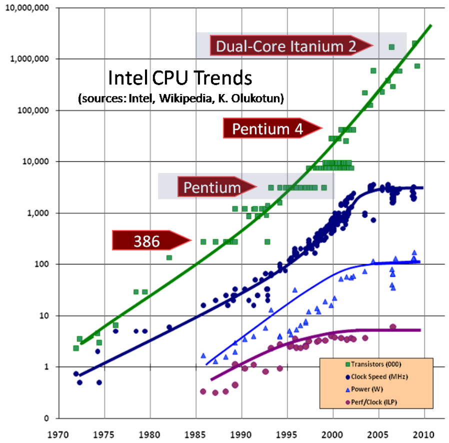
- Large data centers allow for economics of scale
  - Cheaper hardware purchases
  - Cheaper cooling of hardware
    - Example: Google paid 40 MEur for a Summa paper mill site in Hamina, Finland: Data center cooled with sea water from the Baltic Sea
  - Cheaper electricity
  - Cheaper network capacity
  - Smaller number of administrators / computer
- Unreliable commodity hardware is used
- Reliability obtained by replication of hardware components and a combined with a fault tolerant software stack

Cloud Computing Technologies

A collection of technologies aimed to provide elastic "pay as you go" computing

- Virtualization of computing resources: Amazon EC2, Eucalyptus, OpenNebula, Open Stack Compute, ...
- Scalable file storage: Amazon S3, GFS, HDFS, ...
- Scalable batch processing: Google MapReduce, Apache Hadoop, PACT, Microsoft Dryad, Google Pregel, Spark, ...
- Scalable cloud datastores (NoSQL databases): Amazon Dynamo, Riak, Apache Cassandra, MongoDB, Google Bigtable, HBase, Google Cloud Spanner...
- Distributed Coordination Services: Google Chubby, Apache Zookeeper, ...
- Scalable Web applications hosting: Amazon AWS, Google App Engine, Microsoft Azure, Heroku, ...

Clock Speed of Processors



- Herb Sutter: The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software. Dr. Dobb's Journal, 30(3), March 2005 (updated graph in August 2009).

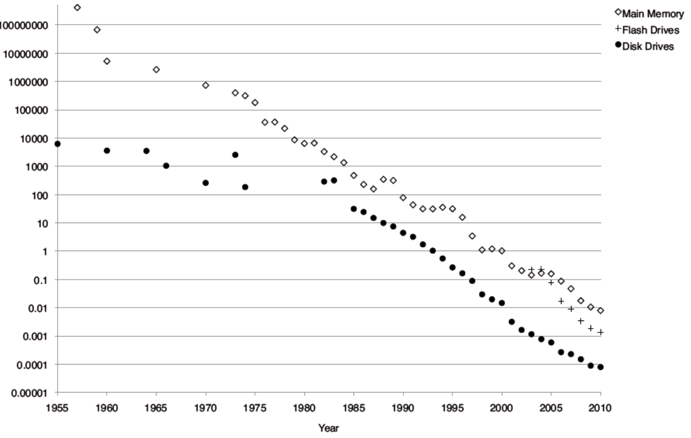
Implications of the End of Free Lunch

- The clock speeds of microprocessors are not going to improve much in the foreseeable future
  - The efficiency gains in single threaded performance are going to be only moderate
- The number of transistors in a microprocessor is still growing at a high rate
  - One of the main uses of transistors has been to increase the number of computing cores the processor has
  - The number of cores in a low end workstation (as those employed in large scale datacenters) is going to keep on steadily growing
- Programming models need to change to efficiently exploit all the available concurrency - scalability to high number of cores/processors will need to be a major focus

Dark Silicon - End of Moore's law in Sight?

- Even worse news ahead, computing will be hitting a wall: Silicon is becoming energy constrained
- We will have much more transistors than what we can switch on and off at each clock cycle
- This is called: Dark Silicon
- For implications of dark silicon, see:
    - Hadi Esmaeilzadeh, Emily R. Blem, Renée St. Amant, Karthikeyan Sankaralingam, Doug Burger: Dark Silicon and the End of Multicore Scaling. IEEE Micro 32(3): 122-134 (2012)
    - Michael B. Taylor: A Landscape of the New Dark Silicon Design Regime. IEEE Micro 33(5): 8-19 (2013)

Tape is Dead, Disk is Tape, RAM locality is King



- Trends of RAM, SSD, and HDD prices. From: H. Plattner and A. Zeier: In-Memory Data Management: An Inflection Point for Enterprise Applications

Tape is Dead, Disk is Tape, RAM locality is King

- RAM (and SSDs) are radically faster than HDDs: One should use RAM/SSDs whenever possible
- RAM is roughly the same price as HDDs were a decade earlier
  - Workloads that were viable with hard disks a decade ago are now viable in RAM
  - One should only use hard disk based storage for datasets that are not yet economically viable in RAM (or SSD)
  - The Big Data applications (HDD based massive storage) should consist of applications that were not economically feasible a decade ago using HDDs

Bump in Kryder’s Law since 2010

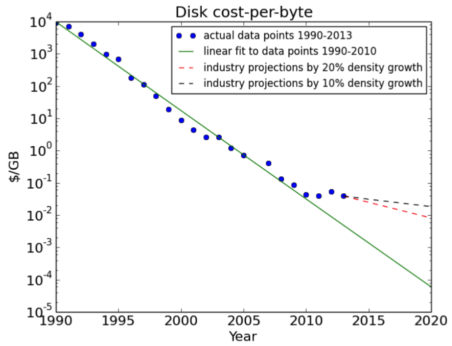


Fig. 1. Cost-per-byte decrease slowed dramatically in 2010 [2].

- Preeti Gupta, Avani Wildani, Ethan L. Miller, Daniel C. Rosenthal, Ian F. Adams, Christina E. Strong, Andy Hospodor: An Economic Perspective of Disk vs. Flash Media in Archival Storage. MASCOTS 2014: 249-254

Scaling Up vs Scaling Out

- Scaling up: When the need for parallelism arises, a single powerful computer is added with more CPU cores, more memory, and more hard disks
- Scaling out: When the need for parallelism arises, the task is divided between a large number of less powerful machines with (relatively) slow CPUs, moderate memory amounts, moderate hard disk counts
- Big Data Platforms are trying to exploit scaling out instead of scaling up

Pros and Cons of Scaling Up vs Scaling Out

- Scaling up is more expensive than scaling out. Big high-end systems have much higher pricing for a given: CPU power, memory, and hard disk space
- Scaling out is more challenging for fault tolerance. A large number of loosely coupled systems means more components and thus more failures in hardware and in networking. Solution: Software fault tolerance
- Scaling out is more challenging for software development due to larger number of components, larger number of failures both in nodes and networking connecting them, and increased latencies. Solution: Scalable cloud platforms

Warehouse-scale Computing (WSC)

- The smallest unit of computation in Google scale is: Warehouse full of computers
- [WSC]: Luiz André Barroso, Urs Hölzle: *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines* Morgan & Claypool Publishers 2009

  http://dx.doi.org/10.2200/S00193ED1V01Y200905CAC006
- The WSC book says:
  "…we must treat the datacenter itself as one massive warehouse-scale computer (WSC)."

Cloud Datastores (NoSQL Databases)

- Quite often the term Datastore is used for database systems developed for the cloud, as they often do not fully support all features of traditional relational databases (RDBMS)
- Other term used often is NoSQL databases, as the original systems did not support SQL. However, some SQL support is getting added also to cloud datastores, so the term is getting outdated quite fast
- The cloud datastores can be grouped by many characteristics

Database ACID guarantees

A traditional (centralized) database can guarantee its client ACID (atomicity, consistency, isolation, durability) properties:

- Atomicity: Database modifications must follow an ”all or nothing” rule. Each transaction is said to be atomic. If one part of the transaction fails, the entire transaction fails and the database state is left unchanged.
- Consistency (as defined in databases): Any transaction the database performs will take it from one consistent state to another.
- Isolation: No transaction should be able to interfere with another transaction at all.
- Durability: Once a transaction has been committed, it will remain so.

Consistency & Availability in Distributed Databases

We define the three general properties of distributed databases popularized by Eric Brewer:

- Consistency (as defined by Brewer): All nodes have a consistent view of the contents of the (distributed) database
- Availability: A guarantee that every database request eventually receives a response about whether it was successful or whether it failed
- Partition Tolerance: The system continues to operate despite arbitrary message loss

Brewer's CAP Theorem

- In a PODC 2000 conference invited talk Eric Brewer made a conjecture that it is impossible to create a distributed asynchronous system that is at the same time satisfies all three CAP properties:
  - Consistency
  - Availability
  - Partition tolerance
- This conjecture was proved to be a Theorem in the paper: "Seth Gilbert and Nancy A. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News, 33(2):51-59, 2002."

Brewer's CAP Theorem

Because it is impossible to have all three, you have to choose between two of CAP:

- CA: Consistent & Available but not Partition tolerant
  - A non-distributed (centralized) database system
- CP: Consistent & Partition Tolerant but not Available
  - A distributed database that can not be modified when network splits to partitions
- AP: Available & Partition Tolerant but not Consistent
  - A distributed database that can become inconsistent when network splits into partitions

Example CA Systems

- Single-site (non-distributed) databases
- LDAP - Lightweight Directory Access Protocol (for user authentication)
- NFS - Network File System
- Centralized version control - Svn
- Hadoop Distributed Filesystem: HDFS Namenode

These systems are often based on two-phase commit algorithms, or cache invalidation algorithms.

Example CP Systems

- Distributed datastores - Example: Google Bigtable, Apache HBase, Google Cloud Spanner
- Distributed coordination systems - Example: Google Chubby, Apache Zookeeper

The systems often use a master copy location for data modifications combined with pessimistic locking or majority (aka quorum) algorithms such as Paxos by Leslie Lamport.

Example AP Systems

- Filesystems allowing updates while disconnected from the main server such as AFS and Coda.
- Web caching
- DNS - Domain Name System
- Distributed version control system - Git
- “Eventually Consistent” Datastores - Amazon Dynamo, Apache Cassandra, Riak

The employed mechanisms include cache expiration times and leases. Sometimes intelligent merge mechanisms exists for automatically merging independent updates.

Scalable Cloud Data Store Features

A pointer to comparison of the different cloud datastores is the survey paper: “Rick Cattell: Scalable SQL and no-SQL Data Stores, SIGMOD Record, Volume 39, Number 4, December 2010”.
`http://www.sigmod.org/publications/sigmod-record/1012/pdfs/04.surveys.cattell.pdf`

Scalable Cloud Data Store Features (cnt.)

The Cattell paper lists some key features many of these new datastores include (no single system contains all of these!):

- The ability to horizontally scale "simple operation" throughput over many servers
- The ability to automatically replicate and to distribute (partition/shard) data over many servers
- A simple call level interface or protocol (vs SQL)

Scalable Cloud Data Store Features (cnt.)

- A weaker concurrency model than the ACID transactions of most relational (SQL) database systems
- Efficient use of distributed indexes and RAM for data storage
- The ability to dynamically add new attributes to data records (no fixed data schema)

A Grouping of Data Stores - Key-value Stores

- Key-value stores
  - Examples: Redis, Riak, Apache Cassandra (partially), Scalaris
  - A unique primary key is used to access a data item that is usually a binary blob, but some systems also support more structured data
  - Quite often use peer-to-peer technology such as distributed hash table (DHT) and consistent hashing to shard data and allow elastic addition and removal of servers

A Grouping of Data Stores - Key-value Stores(cnt

- Key-value stores (cnt.)
  - Some systems use only RAM to store data (and are used as memcached replacements), others also persist data to disk and can be used as persistent database replacements
  - Most systems are AP systems but some (Scalaris) support local row level transactions
  - Cassandra is hard to categorize as it uses DHT, is an AP system, but has a very rich data model

A Grouping of Data Stores - Document Stores

- Document Stores
  - Examples: Amazon SimpleDB, CouchDB, MongoDB
  - For storing structured documents (think, e.g., XML)
  - Do not usually support transactions or ACID semantics
  - Usually allow very flexible indexing by many document fields
  - Main focus on programmer productivity, not ultimate data store scalability

A Grouping of Data Stores - Extensible Record Stores

- Extensible Record Stores (aka “BigTable clones”)
  - Examples: Google BigTable, Google Megastore, Apache HBase, Hypertable, Apache Cassandra (partially)
  - Google BigTable paper gave a new database design approach that the other systems have emulated
  - BigTable is based on a write optimized design: Read performance is sacrificed to obtain more write performance
  - Other notable features: Consistent and Partition tolerant (CP) design, Automatic sharding on primary key, and a flexible data model (more details later)

A Grouping of Data Stores - Scalable Relational Systems

- Scalable Relational Systems (aka Distributed Databases)
  - Examples: MySQL Cluster, VoltDB, Oracle Real Application Clusters (RAC)
  - Data sharded over a number of database servers
  - Usually automatic replication of data to several servers supported
  - Usually SQL database access + support for full ACID transactions
  - For scalability joins that span multiple database servers or global transactions should not be used by applications
  - Scalability to very large (100+) database servers not demonstrated yet but there is no inherent reason why this could not be done

Eventual Consistency / BASE

- The term “Eventually Consistent” was popularized by the authors of Amazon Dynamo, which is a datastore that is an AP system - accessible and partition tolerant
- Also the term BASE (basically available, soft state, eventually consistent) is a synonym for the same approach
- Basically these are datastores that value accessibility over data consistency
- Quite often they offer support to automatically resolve some of the inconsistencies using e.g., version numbering or CRDTs - Commutative Replicated Data Types
- Example systems: Amazon Dynamo, Riak, Apache Cassandra

RDBMS Supporter View

- RDBMS can do everything the scalable cloud data stores can do with similar scalability when used properly
- SQL is a convenient expressive declarative query language
- RDBMS have 30 years of engineering experience put into them and are highly tuned
- There are a very large number of specialized RDBMS for various application domains (interactive vs. analytics, RAM vs. disk based data, etc.)
- A standardization around relational schema and SQL brings benefits in design and training, where same set of skills can be effectively employed with another RDBMS system

RDBMS Opponent View

- The RDBMS vendors have not demonstrated scalability matching that of the newly architected scalable cloud data stores (e.g., BigTable)
- Primary key lookup is more easy to understand than SQL, and thus enables a much lower learning curve
- RDBMS force data schema on applications unnecessarily
- SQL makes it “too easy” to write expensive queries by accident such as complicated joins involving many tables and several servers. If these expensive operations are removed from the query language, the complexity of a query evaluation becomes obvious to the programmer
- RDBMS systems have become dinosaurs in their 30 year rule of the database market, and the (hardware) assumptions on which they have been built are obsolete

Scalable Data Stores - Future Speculation

- Long running serializable global transactions are very hard to implement efficiently, for scalability they should be avoided at all costs. Counterexamples: Google Percolator, Google Cloud Spanner
- The requirements of low latency and high availability make AP solutions attractive but they are more difficult to use for the programmer (need to provide application specific data inconsistency recovery routines!) than CP systems
- The time of "one size fits all", where a RDBMS was a solution to all datastore problems has passed, and scalable cloud data stores are here to stay

Scalable Data Stores - Future Speculation (cnt.)

- ACID transactions are needed for, e.g., financial transactions, and there traditional RDBMS will be dominant
- Many of the new systems are not yet proven in production
- There will be a consolidation to a smaller number of data stores, once the “design space exploration” settles down

NewSQL - Google Cloud Spanner

- A beta release of Google's internal Spanner Database
- SQL support
- ACID transactions that can span multiple servers globally
- Fully consistent database with a single global database image
- Implementation based on Paxos for replication, Transaction commit priority made using physical timestamps
- All database server clocks syncronized very tightly using atomic clocks + GPS based time syncronization
- All database transactions have to wait two times the max clock drift holding write locks before committing a transaction
- Open source alternatives being developed: CockroachDB, Apache Kudu

BigTable

- Described in the paper: “Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber: Bigtable: A Distributed Storage System for Structured Data. ACM Trans. Comput. Syst. 26(2): (2008)”
  - A highly scalable consistent and partition tolerant datastore
  - Implemented on top of the Google Filesystem (GFS)
  - GFS provides data persistence by replication but only supports sequential writes
  - BigTable design does no random writes, only sequential writes are used
- Open source clone: Apache HBase

BigTable

- Sequential writes are much faster than random writes for modern hard disks
- BigTable is a write optimized design: Writes are optimized over reads
- The random reads that miss the DRAM cache used are slower than in traditional RDBMS
- The design also minimizes the number of bytes written by using efficient logging and compression of data. Also a good design principle for Flash SSD use

BigTable Data Model

- Bigtable paper describes itself as: ”. . . a sparse, distributed, persistent multi-dimensional sorted map”
- Namely, BigTable stores data as strings, which can be accessed by three coordinates:
  - `row` - an arbitrary string row key (10-100 bytes typical, max 64 KB)
  - `column` - a column key, consisting of a column family and column qualifier (name) in syntax `family:qualifier`
  - `timestamp` - a 64 bit integer that can be used to store a timestamp
  - Thus we have a map with three coordinates: `(row:string, column:string, time:int64) -> string`

BigTable Data Layout

- In BigTable data is stored sorted by the row key, and the data is automatically sharded by the row key, allowing for efficient scans in increasing alphabetical row key order
- Columns are grouped in “Column families”, and all columns in a single column family are stored together in compressed form on disk
- Some queries might not access columns in all column families, and this allows data in these column families to be easily skipped for such queries
- The timestamp field allows applications to store several copies of the same data together, e.g., Website content changes over time

BigTable Atomicity Guarantees

- Each row in BigTable can have a different number of columns
- Some rows might have thousands or even many more columns
- BigTable atomicity guarantee: Updates to a single row are atomic, irregardless of the number of column families or columns being updated
- No transactions for updating multiple rows atomically are available

BigTable Write Path

- When BigTable Tablet server receives a write, it first commits it to a commit log on GFS
- After the commit, write is added to memtable
- Once memtable fills up, all written data is sorted in memory, and stored into a sorted indexed SSTable on GFS, and the commit log file can be removed

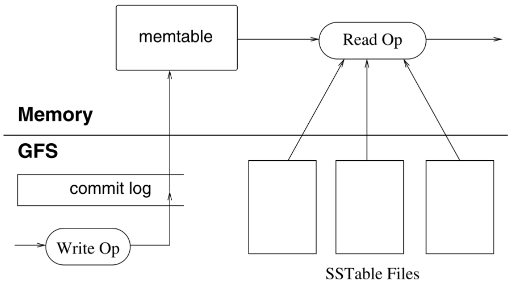


Fig. 6. Tablet representation.

BigTable Conclusions

- BigTable is a scalable write optimized database design
- It uses only sequential writes for improved write performance
- Requires periodic compactions to reduce the number of SSTable files and maintain reasonable read performance
- For read intensive workloads BigTable can be a good fit if all of the working set fits into DRAM
- For read intensive working sets much larger than DRAM, traditional RDBMS systems are still a better match

Apache HBase

- Apache HBase is an open source Google BigTable clone
- It very closely follows the BigTable design but has the following differences
  - Instead of GFS, HBase runs on top of HDFS
  - Instead of Chubby, HBase uses Apache Zookeeper
  - SSTable of BigTable are called in HBase HFile (and HFile V2)
  - HBase only partially supports fully memory mapped data

Facebook Messaging on HBase

- In the paper "Dhruba Borthakur, Jonathan Gray, Joydeep Sen Sarma, Kannan Muthukkaruppan, Nicolas Spiegelberg, Hairong Kuang, Karthik Ranganathan, Dmytro Molkov, Aravind Menon, Samuel Rash, Rodrigo Schmidt, Amitanand S. Aiyer: Apache Hadoop goes realtime at Facebook. SIGMOD Conference 2011: 1071-1080" authors from Facebook describe the infrastructure behind Facebook messaging
  - 500 million users, several Petabytes of data
  - 135 000 000 000 messages per month

Apache Cassandra

- Apache Cassandra is a cloud datastore that values low latency and availability over consistency
- Developed originally by Facebook, now also commercially supported by Datastax
- Runs behind many massive Websites: Example - Netflix
- It uses a BigTable-like data model and a write optimized storage engine based on SSTables

Apache Cassandra

- Apache Cassandra is maybe the closest open source project in spirit to Amazon Dynamo
- Low latency and availability over consistency
- Developed originally by Facebook, now also commercially supported by Datastax
- Runs behind many massive Websites: Example - Netflix
- It uses a BigTable-like data model and a write optimized storage engine based on SSTables
- Otherwise its design resembles that of Dynamo:
  - Quorum reads and writes with configurable N, W, and R
  - No vector clocks available! Automatic merging of conflicting versions during reads using “read repair”, where the data item with the largest timestamp value wins!!! (mutating data stored in Cassandra can be dangerous without proper care!)

Riak

- Another widely used AP datastore
- Just key-values, not wide rows like in Bigtable or Cassandra
- Uses consistent hashing
- Last write wins (LWW) by default, this is dangerous for mutating data!
- Implements vector clocks, which makes updating data easier by making full custom merge functions easier to write
- Implements Commutative Replicated Data Types - CRDTs, allowing fylly automatic merging for certain data types with limited operations
- Also runs large systems in production, is commercially supported by Basho

Programmer Hints for AP Datastores

1. Hint: Write once-read many: If data is written only once, there can not be two conflicting versions of the data
2. Hint: If you mutate data, do not use “last-write-wins” - it is an unsafe default, and can result in data loss
3. Hint: If you have to mutate data, use Commutative Replicated Data Types CRDTs, either supported by the datastore, or roll your own on top of vector clocks, if the datastore does not support CRDTs
4. Hint: Not all data types can be turned into CRDTs (example: set with both add and delete operation can not!), for these data types use a CP datastore instead!

Shapiro, M., Preguica, N., Baquero, C., Zawirski, M.: A comprehensive study of Convergent and Commutative Replicated Data Types. app. Rech. 7506, Institut National de la Recherche en Informatique et Automatique (INRIA), 2011

Testing Cloud Datastores

- Many cloud datastores can lose data, see: `http://aphyr.com/tags/jepsen` and `http://jepsen.io`
- Examples:
  - Broken protocol design: Redis, NuoDB, Elasticsearch, MongoDB (replication protocol can lose data before version 3.4.0)
  - Protocol bugs that have been fixed: etcd
  - Trying to be AP and CP at the same time: RabbitMQ
  - Users naively using last-write-wins that is on by default: Riak, Cassandra
  - Default options that can result in data loss: MongoDB
  - Missing a CP configuration option: Kafka, added in newer versions

Distributed Coordination Services

- Maintaining a global database image under node failures is a very subtle problem, see e.g.: "Michael J. Fischer, Nancy A. Lynch, Mike Paterson: Impossibility of Distributed Consensus with One Faulty Process, J. ACM 32(2): 374-382 (1985)"
- The Paxos algorithm is a very subtle algorithm that will be able to replicate a distributed database if enough (majority of the set of) servers are up and able to communicate with each other

Distributed Coordination Services (cnt.)

- From an applications point of view the distributed coordination services should be used to store global shared state: Global configuration data, global locks, live master server locations, live slave server locations, etc.
- One should use centralized infrastructure such as Google Chubby or Apache Zookeeper to only implement these tricky fault tolerance algorithms only once