09 – Cloud Storage

- NoSQL Databases
  - Christof Strauch

- Dynamo: Amazon’s Highly Available Key-value Store
NoSQL Databases

- Developed by companies to fulfill internal requirements
  - Some replicate ideas from Amazon Dynamo/Google bigtables
  - Some port ideas from existing databases to web technologies
  - New ideas
Taxonomy

- Data Model
  - Key-Value-Cache
    - distributed memory object caching system
    - Independent of the persistent storage
  - Key-Value-Store
    - Persistent storage
    - map/dictionary, allowing clients to put and request values per key
  - Data-Structures Server
    - Complex key (strings., lists, bitmaps, ...)

Taxonomy

- Data Model
  - Document Store
    - Extension of key-value
    - Semi structured values (documents)
      - Inclusion of keys
    - No strict schema
  - Wide Columnar Store
    - Large map indexed by
      - Row key, column key, timestamp
    - Values interpreted by applications
Taxonomy

- **Data Model**
  - **Tuple Store**
    - Stored data is organized in spaces
      - Composed of tuples (1\textsuperscript{st} value is key)
  - **Object Database**
    - Stored data are objects
Network partitions

- Network can partition
  - Hardware fault, router misconfigured, undersea cable cut, ...
  - Result: Global connectivity is lost
  - What does this mean for the properties of our system?
The CAP theorem

• What we want from a web system:
  – Consistency: All clients share the same view of the data, even in the presence of concurrent updates
  – Availability: All clients can access at least one replica of the data, even when faults occur
  – Partition-tolerance: Consistency and availability hold even when the network partitions

• Can we get all three?
  – CAP theorem: We can get at most two out of the three
    • Which ones should we choose for a given system?
  – Conjecture by Brewer; proven by Gilbert and Lynch
Common CAP choices

- Consistence + Availability (Forfeit Partitions)
  - Strong consistency
  - Single-site databases
  - Cluster databases
  - LDAP
  - xFS file system

- 2-phase-commit
- cache-validation protocols
Common CAP choices

- Consistency & Partition tolerance (Forfeit Availability)
  - Many replicas + consensus protocol
  - Do not accept new write requests during partitions
  - Certain functions may become unavailable
  - Distributed databases
  - Distributed locking
  - Majority protocols

- Pessimistic locking
- Make minority partitions unavailable
Common CAP choices

- Availability & Partition tolerance
  - Many replicas + relaxed consistency
  - Continue accepting write requests
  - Clients may see inconsistent state during partitions
  - Web caching
  - DNS
- expirations/leases
- conflict resolution
- optimistic
The current trend in cloud computing data storage is to loosen or relax the requirements of consistency in favor of more availability.

**BASE approach:**

- **Basically available:**
  - system guarantees the availability of your data; but the response can be "failure" if the data is in the middle of changing.

- **Soft State:**
  - the state of the system is constantly changing.

- **Eventually Consistent:**
  - the system will eventually become consistent once it stops receiving input.
Relaxed consistency: ACID vs. BASE

- Classical database systems: ACID semantics
  - Atomicity
  - Consistency
  - Isolation
  - Durability

- Modern Internet systems: BASE semantics
  - Basically Available
  - Scalable
  - Eventually consistent
• ACID
  - Strong consistency
  - Isolation
  - Focus on “commit”
  - Nested transactions
  - Availability?
  - Conservative (pessimistic)
  - Difficult evolution (e.g. schema)

• BASE
  - Weak consistency – stale data OK
  - Availability first
  - Best effort
  - Approximate answers OK
  - Aggressive (optimistic)
  - Simpler!
  - Faster
  - Easier evolution
BASE

- BASE is optimistic and
- Accepts that the database consistency will be in a state of flux.
- It achieves availability by supporting partial failures without total system failure
  - i.e. partition tolerance).
- To implement BASE, systems rely on
  - message queue
    - persistently store and route data to various storage services
      the perform the actual database operations.
Consistency and partitions

- Use replication to mask limited # of faults
  - Can achieve strong consistency by having replicas agree on a common request ordering
  - Even non-crash faults can be handled, as long as there are not too many of them (typical limit: 1/3)

- Partition tolerance, availability, consistency?
  - Can't have all three (CAP theorem)
  - For some services, need to drop one (usually availability)
  - If service works with weaker consistency guarantees, such as eventual consistency, can get a compromise (BASE)
    - Example: Shopping cart
Consistency levels

- **Strict Consistency** according to Lipcon
  - “All read operations must return data from the latest completed write operation, regardless of which replica the operations went to.

- **Eventual Consistency** means that readers will see writes, as time goes on:
  - “In a steady state, the system will eventually return the last written value.”
Eventually consistent system

- **Read Your Own Writes (RYOW) Consistency**
  - a client sees his updates immediately after they have been issued and completed, regardless if he wrote to one server and in the following reads from different servers.
  - Updates by other clients are not visible to him instantly.

- **Session Consistency**
  - read your own writes consistency limited to a session scope (usually bound to one server)
  - so a client sees his updates immediately only if read requests after an update are issued in the same session scope.

- **Casual Consistency**
  - if one client reads version x and subsequently writes version y, any client reading version y will also see version x.

- **Monotonic Read Consistency**
  - provides the time monotonicity guarantee that clients will only see more updated versions of the data in future requests.
Versioning of Datasets

• Timestamps
  – However, timestamps “rely on synchronized clocks and don’t capture causality” :(

• Optimistic Locking
  – implies that a unique counter or clock value is saved for each piece of data

• Multiversion Storage
  – store a timestamp for each table cell
  – Artificial values that define a order
Versioning of Datasets

- Vector Clocks
  - Tuple of all nodes clocks $V[0], V[1], ..., V[n]$
  - A time stamp $i$ is a vector $Vi[0], Vi[1], ..., Vi[n]$
    - Representing each node clock at that instant
  - Local version of vector is updated:
    - Internal operation on node $k$
      - update corresponding $Vi[k]$
    - If node $i$ sends a message to node $w$,
      - it first advances its own clock value $Vi[i]$
      - attaches the vector clock $Vi$ to the message to node $w$
Versioning of Datasets

- **Vector Clocks**
  - Local version of vector is updated:
    - if node k receives a message from node w,
      - it first advances its vector clock \( V_i[k] \)
      - merges its own vector clock with the vector clock \( V \) message attached to the message from node w so that:
        - \( V_i = \max(V_i, V_{message}) \)
    - Conflict verification
      - \( ! V_i < V_{message} \) and \( ! V_i > V_{message} \)
Partitioning

- **Memory Caches**
  - Replication of most frequently requests in memory

- **Clustering**
  - DB composed of multiple servers
    - Multiple server hidden
  - Client sees single server

- **Separating reads from writes**
  - Master handle writes
  - Slaves handle reads

- **Shardings**
  - Spliting of data among severel servers
  - Data tipically accessed on same server
Consistent Hashing

- Hash both objects and caches using the same hash function
  - machines get an interval of the hash-function’s range
  - adjacent machines can take over parts of the interval of their neighbors if those leave
    - can give parts of their own interval away if a new node joins and gets mapped to an adjacent interval.
  - client applications can calculate which node to contact in order
    - to request or write a piece of data and there is no metadata server necessary
Data replicas

- Increases reliability
- Spreads workload
  - several physical nodes are responsible for a piece of data
- Impacts algorithm
  - \( N < R+W \) (read-your-own-writes consistency)
    - \( N \) The number of replicas for the data or the piece of data
    - \( R \) The number of machines contacted in read operations
    - \( W \) The number of machines that have to be blocked in write operations
  - No immediate consistency nor isolation
Storage Layout

- **Row-Based Storage Layout**
  - Table gets serialized as its lines are appended and flushed to disk
  - Whole datasets can be read and written in a single IO operation
  - Locality of access (on disk and in cache) of different columns
  - Operating on columns is expensive
Storage Layout

- **Columnar Storage Layout**
  - serializes tables by appending their columns and flushing them to disk
  - operations on columns are fast and cheap
  - Good for analytics of columns

- **Columnar Storage Layout with Locality Groups**
  - adds the feature of defining so-called locality groups
    - groups of columns expected to be accessed together by clients
Storage layout

(a) Row-based

(b) Columnar

(c) Columnar with locality groups
Query languages

- key/value
  - often only provide a lookup by primary key or some id field and lack capabilities to query any further fields

- Query languages
  - Document columnar
  - Translation queries to database accesses
  - Limitations
Query models

- key/value
  - Consistent ashing

- Companion SQL
  - Searchable attributes are copied to a SQL db
  - Local queries will return dataset that will later be contacted
Query models

- Scatter/gather Local search
  - Query processor dispatches queries to several nodes
  - Results are processed/aggregated centrally (on processor)
Query models

• Distributed B+Trees
  – Each searchable attribute has a b-tree
  – Query begins by finding the root of the b-tree
    • From hash of attribute
  – Tree is traversed till finding the query result
NoSQL Systems

- Key-Value-Store
  - Dynamo (amazon)

- Object Database

- Document Store
  - Mongo

- Wide Columnar Store
  - Bigtable (Google)
  - Cassandra (Facebook → OSS)