AMAZON SIMPLEDB

Key-Value Stores in the Cloud

- Ideal: Abstraction of a 'big disk in the clouds':
  - Perfect durability
  - 100% availability
  - Zero latency from anywhere on earth
  - Minimal bandwidth utilization
  - Isolation under concurrent updates (**consistency**)

Finding the right tradeoff

- Read-only (or read-mostly) data
  - Replicate it everywhere
- Granularity matters: "Few large-object" tasks
  - Fewer requests, more client processing
  - More expensive to replicate or to update
- Separate solutions for large read-mostly objects vs. small read-write objects

25

25

Example Solutions

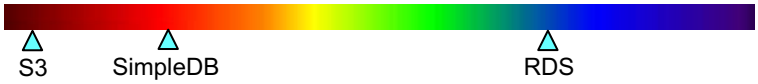
- **Simple Storage Service (S3):**
  - large objects – files, virtual machines, etc.
  - assumes objects change infrequently
  - objects are opaque
- **SimpleDB:**
  - small objects – Java objects, records, etc.
  - frequent updates; greater need for consistency
  - multiple attributes or properties

26

26

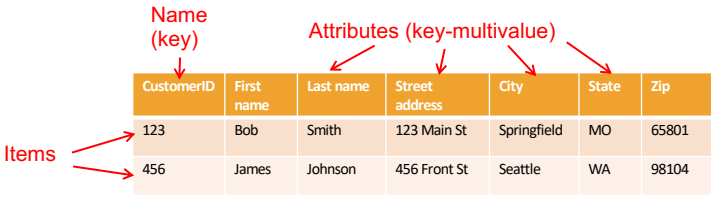
Amazon SimpleDB

- A highly scalable, non-relational data store
  - Highly scalable
  - Built-in replication
  - Automatic indexing
  - No 'real' transactions, just conditional put/delete
  - No 'real' relations, just a fairly basic select



27

SimpleDB: Data model



CustomerID	First name	Last name	Street address	City	State	Zip
123	Bob	Smith	123 Main St	Springfield	MO	65801
456	James	Johnson	456 Front St	Seattle	WA	98104

- Somewhat analogous to a spreadsheet:
  - Domains: Entire 'tables'; like buckets
  - Items: Names with attribute-multivalue sets
- It is possible to add attributes later
  - No pre-defined schema

28

SimpleDB: Basic operations

- ListDomains
- CreateDomain, DeleteDomain
- DomainMetadata

- PutAttributes
  - Also atomic BatchPutAttributes – all must succeed
- DeleteAttributes
- GetAttributes

- Select (like an SQL query)

29

29

SimpleDB: PUT and GET

- PutAttributes:
  - Specify the domain and the item name
    `[key] → [list of name/value pairs]`
  - Attribute.1.Name, Attribute.1.Value, etc.
  - Each Attribute.X has an optional Replace flag (Replace = 0 means add another value)

- GetAttributes
  - Specify domain and item name + optionally attribute
  - Can choose whether the read should be consistent or not
    - Read Follows Write

30

30

SimpleDB: Conditional Put

- Use this to guarantee consistency?
  - Issue: concurrent updates
  - Idea: implement a version number, e.g., like this:

```
do {
  List<Attributes> attribs = kvs.getAttributesFor(key);
  ... update the attribute values as we like ...
  retCode = kvs.conditionalPut(key, attribs,
     (“version”, attribs.get(“version”)));
} while (retcode == ErrorCode.ConditionalCheckFailed);
```

31

31

SimpleDB: Select

- `SELECT output_list FROM domain_name WHERE expression [sort expression] [limit spec]`
- Example: `"select * from books where author like ’Dug%' and price <= 55.90 and year is not null order by title desc limit 50"`
- Can choose whether or not read should be consistent (“read your writes”)
- Supports a cursor

32

32

COUCHDB: DOCUMENT STORE

33

33

CouchDB

- Schema-free, document oriented database
  - Documents stored in JSON format (XML in old versions)
  - no joins, no PK/FK (UUIDs are auto assigned)
  - Implemented in Erlang
    - 1st version in C++, 2nd in Erlang and 500 times more scalable
  - Replication (incremental)

34

34

CouchDB

- Schema-free, document oriented database
  - Documents stored in JSON format (XML in old versions)
  - no joins, no PK/FK (UUIDs are auto assigned)
  - Implemented in Erlang
    - 1st version in C++, 2nd
  - Replication (incremer
- Documents
  - UUID, version
  - Old versions retained

```
{
  "_id": "BCCD12CBB",
  "_rev": "1-AB764C",

  "type": "person",
  "name": "Darth Vader",
  "age": 63,
  "headware": ["Helmet", "Sombrero"],
  "dark_side": true
}
```

35

35

CouchDB

- Schema-free, document oriented database
  - Documents stored in JSON format (XML in old versions)
  - no joins, no PK/FK (UUIDs are auto assigned)
  - Implemented in Erlang
    - 1st version in C++, 2nd
  - Replication (incremer
- Documents
  - **UUID**, version
  - Old versions retained

```
{
  "_id": "BCCD12CBB",
  "_rev": "1-AB764C",

  "type": "person",
  "name": "Darth Vader",
  "age": 63,
  "headware": ["Helmet", "Sombrero"],
  "dark_side": true
}
```

36

36

CouchDB

- Schema-free, document oriented database
  - Documents stored in JSON format (XML in old versions)
  - no joins, no PK/FK (UUIDs are auto assigned)
  - Implemented in Erlang
    - 1st version in C++, 2nd
  - Replication (incremer
- Documents
  - UUID, **version**
  - Old versions retained

```
{
  "_id": "BCCD12CBB",
  "_rev": "1-AB764C",

  "type": "person",
  "name": "Darth Vader",
  "age": 63,
  "headware": ["Helmet", "Sombrero"],
  "dark_side": true
}
```

37

37

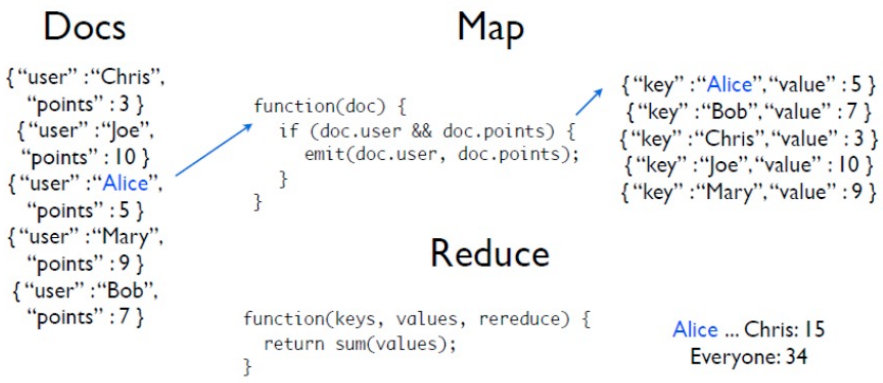
CouchDB

- REST API
  - Create: PUT /db/docid
  - Read: GET /db/docid
  - Update: POST /db/docid
  - Delete: DELETE /db/docid
- Views
  - Filter, sort, "join", aggregate, report
  - MapReduce-based
  - K/V pairs from MapReduce stored in B-tree
  - Built on demand
  - Can be materialized & incrementally updated

38

38

MapReduce Views



39

39