Cloud-based Database Systems
(No SQL Databases)

Lecture- 16 Sept. 2013
CSV888 - 2013 (Semester 1) S. Bhalla
Distributed Database System

- Distributed database system →
  - loosely coupled sites
  - [share no physical component]

- Each Database site →
  independent of other Sites + Messages

- Transactions →
  access data at one or more sites
19.1 Homogeneous Distributed Databases

Homogeneous distributed database →
- 1. All sites have identical software
- 2. All Sites → aware of each other; and cooperate in processing
  → allows change to schemas or software (Site autonomy ?)
- 3. Appears to user as a single system:
  Solaris (U. of Aizu), Distributed ORACLE

Heterogeneous distributed database
- 1. Sites → may use different schemas and software
  ‣ Difference in schema is a major problem for query processing
  ‣ Difference in software is a major problem for transaction processing
- 2. Sites → may not be aware of each other
  → provide limited facilities for cooperation in transaction processing
19.2 Distributed Data Storage

- Database → Relational data model

- Replication
  - System → multiple copies of data at different sites → faster retrieval and fault tolerance.

- Fragmentation
  - Relation → partitioned into many fragments → stored in distinct sites

- Replication and fragmentation can be combined
  - Relation → partitioned into several fragments
  - System → identical replicas of each such fragment.
Data Replication

- Replicated → A relation (or fragment of a relation) is stored redundantly in two or more sites.

- Full replication of a relation → relation is stored at all sites.

- Fully redundant databases → every site contains a copy of the entire database.
**Evolution: Cloud Based Databases**

- **1990 - 2000**  
  Software as a service - GMAIL, Yahoo Mail

- Time: more Applications + common Software Google Docs,

- Data Center → Idea: Common Services →  
  Software / Platform / Infrastructure → SaaS / PaaS / IaaS

- → Huge Data Centers + Web = CLOUD

- **Cloud Computing Companies:** Google, Yahoo, Amazon, IBM, NEC, …

- **Many Research Technologies** →

- OLD: Virtualization + Parallel Computing + Distributed Systems

- NEW: [Parallel Computing → Map Reduce],…, JSON, …KML
Computing → Cloud Computing

- Customer
  - → Buy Computer Time, Capacity, Software
  - ← at any time

- Grow or Shrink: New University / Hospital → IT Service
- Map Services, Data Storage Services

- Connect through “Web Services APIs”

- Huge Applications: Large Cloud Computing Company: Amazon, Google: Million Customers → Cloud-based Databases
Cloud-based Databases

- Applications → Different
- Example: Gmail ↔ availability and scalability Important

- Cloud Databases → Consistency ?
- Air-traffic Control, Banking ATMs, … [Not on Cloud]

- Data: Owned and Unified in Schema (owner, Client)
  - Computers operated (service provider)
  - remotely accessed
  - Transactions, Heterogeneous Systems → pending research
Cloud Computing – Data Center

- Database \(\rightarrow\) partitioned over 1000s of CPUs
- Prototype Systems:
  - Bigtable (Google)
  - Cassandra (Facebook)
  - Sherpa/PNUTS (Yahoo)
  - Data Storage of AZURE (Microsoft)
  - Simple Storage Services (S3) [Amazon] and many more

  \(\rightarrow\) provides a web interface to **Dynamo** \(\rightarrow\)**

  **Key, Value storage system**

1. Do not provide SQL?

2. Do not support Transactions (ACID)
Data Representation

- Attributes ➔ new, Changing, Complex Attributes
- Relational Model ➔ [X]
- Web Application ➔ Electronic Health Records (EHRs); Gmail; Yahoo Photo; PHRs (Microsoft Health), Maps

Cloud Computing ➔ XML, JSON, KML …

Bigtable (Google) ➔ Define their own Data Model

Web Application ➔ Do not need extensive Query (SQL – [X])

Main Task ➔ Store data with a Key Value and Retrieve
Data Representation

- Data → Access → by user Id [Simple Table lookup: Huge]
- Example: Social Networking Application:
  - Indexing, JOIN, Sub-string matching → separate modules
  - User ← shown post from all friends

- Model: Sender → Post; Messages → Users; User → check updates and receive data/messages;
- IBM/Yahoo/Amazon → Bulletin Board Message Pickups
- Cloud Data-storage → 2 Primitive Functions
- 1. Put (key, value) and 2. Get (key)
- Bigtable → range query on key values
Bigtable – Data Representation

- Record ➔ Component Attributes
  ➔ Stored separately
- Key ➔ (record-identifier, attribute-name)
- To bring a record ➔ range query/prefix matching with record key is used
- Get (key) ➔ attributes and values
- record-identifier ➔ string
- Task: Store pages from web crawler (URL: www.cs.yale.edu/people/silberschatz.html
  ➔ Record Identifier (to keep pages in cluster): edu.yale.cs.www/people/silberschatz.html
Data Representation

- Record Identifier → Quick store and Retrieve [Hash Index]
- JSON ↔ Programming Languages (directly convertible)
- C, C++, Javascript, PHP, Python, ...(Similar data representation)

- Bigtable → Multiple Applications and Multiple Tables:
  Application_name.Table_name.Record_Identifier

- Multiple Versions → Use Time-Stamps, or Integer Value

- Bigtable → Key:
  [Appln, Table](record-identifier, attribute-name, Time-stamp)
JSON – Object Transfer

- JSON → JavaScript Object Notation
- It is a lightweight data-interchange format.
- easy → for humans to read and write.; → easy for machines to parse and generate.
- → a subset of the JavaScript Programming Language
- → completely language independent
- → uses similar conventions that are of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others.
- → good data-interchange language, with parsers available for many languages..
- → represents simple data structures and associative arrays, called objects.
JSON

- Filename extension: .json
- Internet media type: application/json
- Uniform Type Identifier: public.json
- Type of format: Data interchange
- Extended from: JavaScript
- Standard(s): RFC 4627
- Website: json.org
- The official MIME type: "application/json"

Yahoo and Google Web services often use JSAON
The employees object is an array of 3 employee records (objects).
Example: HTML + Javascript

```html
<!DOCTYPE html>
<html>
<body>
<h2>JSON Object Creation in JavaScript</h2>

<p>
Name: <span id="jname"></span><br />
Age: <span id="jage"></span><br />
Address: <span id="jstreet"></span><br />
Phone: <span id="jphone"></span><br />
</p>

<script type="text/javascript">
var JSONObject = {
  "name": "John Johnson",
  "street": "Oslo West 555",
  "age": 33,
  "phone": "555 1234567"};

document.getElementById("jname").innerHTML = JSONObject.name
document.getElementById("jage").innerHTML = JSONObject.age
document.getElementById("jstreet").innerHTML = JSONObject.street
document.getElementById("jphone").innerHTML = JSONObject.phone
</script>
</body>
</html>
```
**JSON Object display ↔ Creation in HTML + JavaScript**

- Name: John Johnson
- Age: 33
- Address: Oslo West 16
- Phone: 555 1234567

- JSON is Similar to XML
- But,
  - No end tag
  - Shorter
  - Quicker to read and write
  - Can be parsed using built-in JavaScript eval()
  - Uses arrays
  - No reserved words
JSON Syntax

- JSON is built on 2 structures:
  
  1. A collection of name/value pairs.
     
     {in various languages realized as an object, record, struct, dictionary, hash table, keyed list, or associative array}.
  
  2. An ordered list of values {In most languages realized as an array, vector, list, or sequence}.

- These are universal data structures. Virtually all modern programming languages support them in one form or another.

- Dual Benefits data format is interchangeable with programming languages (both use same structures)
**JSON Syntax**

- JSON syntax → subset of JavaScript object notation syntax
  - a) Data is in name/value pairs
  - b) Data is separated by comma
  - c) Curly brackets holds objects
  - d) Square brackets holds arrays

- A) JSON Name/Value Pairs → JSON data is written as name/value pairs.

- A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

  "firstName" : "John"

- This is simple, and equals to the JavaScript statement:

  firstName = "John"
1. Huge Scale: Data-storage systems → Partition Data

Parallel Databases → Decide before

Cloud DB → Load increases, more servers down or added → add/remove dynamically

Cloud DB → Partition Data (100 MB unit)
→ Tablet (Fragment)

Partition → uses search key (record Identifier)
→ gmail (record Identifier “your Name”)
get (“your Name”) → 1 tablet

[ If get (kv) → multi-site : Load on System? ]
DB Partitions in a Cloud DB

- Get (Key Value) → Range partitioning (Map, index), OR Hash Key Function

- Replication: Tablet → site (Primary)
- Read/Update → tablet at Primary site
- Update: Primary Site → Replica (copies of Tablet)
- Copy → lags behind
- Failure and recovery → (use Logs for Consistency)
- Data (grows) → Partition Tablet
- Load (grows) → Partition Tablet
DB Partitions in a Cloud DB

- Number of Tablets >> Number of sites
  [Virtual Partitioning in Parallelism]

- Site ↔ Tablet: Index/Directory/Map

- Get (key) → Site Si [Service ↔ Map]

- Centralized: Load?
  → Replicate (Many Copies)

- Protocol: Tablet Split → update Map

- Figure → loosely Coupled [PNUTS (yahoo)]

- Bigtable (google) → Map on Google File System
Transaction and Replication

- Data-storage Systems:
  - A) ACID [X]-no
    (i) Cost of 2PC is too high
    (ii) Blocking in 2 PC, if failure? Unacceptable for Web Application,
    (iii) Secondary Index (update needs 2 PC)
  - [Best] → Transaction on 1 Tablet
    → Tablet Transaction (1 Site)
  - Sherpa/PNUTS (Yahoo): Test-and-set function
  - → Test version of data items: if not consistent [can reject]

- Tablet → Validation based Concurrency Control for Tablet
Network Partition and Failures

- Data-Storage $\rightarrow$ 1000s of sites (10000s Tablets)
- Many Sites $\rightarrow$ [X] down : Cloud DB (availability)

- [Replicate] $\rightarrow$ Tablet at Many Sites
- Site $\rightarrow$ May Fail;
- Cluster (many Sites) $\rightarrow$ May Fail
- Cloud Data-storage System $\rightarrow$ many Clusters

- [Replicate] $\rightarrow$ Tablet at Many Clusters
GFS – Cloud Data Center

- Google File System: Distributed Fault-tolerant File System → GFS (google file system)
- All **System Files** → replicate at 3-4 nodes in each cluster
- **Map** (key Data, other System Data) →
  
  n nodes in each cluster (n >> 4)

- Tablet ↔ Site (1:1 map)
- Site fails → Change **Map** to copy → Update
- Update (**Map**) → Log → Replicated copies of Log
Data-Storage Systems (Bigtable- Google)

1. Site Fails → 2. Tablet → new copy (primary copy) (loosely Coupled system)

(A) New Primary ← Use logs to recover latest version
(B) MAP ← update new map (Tablet : Primary copy)

Bigtable: Map → Index structure on GFS
GFS ← Index + Tablets

Tablet Data → not flushed to DB (Log data Immediately Flushed; LOG copy/replica → latest version)
GFS → System Files (Many Copies)
Failure of few sites/clusters → NO PROBLEM

Site fails → New Tablet copy in Map → access up-to-date LOG data
Data-Storage Systems: Yahoo

- Sherpa/PNUTS (Yahoo)
- 1. Replicate **tablets** → Many nodes in a cluster
- 2. No Distributed File System
- 3. Reliable Messaging System (Persistent Messages) → Highly Available

**Question:** Data Center Fails [Unavailable – Disaster]
- Replicate → At a remote site [Sendai → Osaka, Tokyo]
- Essential → High Availability
- Web Applications → messages (round-trips)
- delay across network ← (long distance)
- Network Partition → more (danger increases)
AJAX Applications

- AJAX $\rightarrow$ need many round trip messages
  $\rightarrow$ Connect user to nearest remote copy
  $\rightarrow$ data (near) : Application Servers

- Web Application [GMAIL] $\rightarrow$ High Availability

- Consistency $\rightarrow$: Allow updates to proceed (if Network Partition/site Fails) ; LATER

- Final State $\rightarrow$ DB is consistent

- Multi-master Replication with Lazy Propagation of updates $\rightarrow$ use PERSISTENT Messages
Persistent Message Networks

- Message Network ➔
  Guarantee (updates will reach): Asynchronous actions

- A) Updates to Secondary Index
- B) Updates to materialized views
- C) Updates from friends in Social Network Applications

- RESEARCH ➔ Traditional Distributed DB on Cloud DB ➔ **ACID + Traditional Query** on Cloud DB
Virtual Machines

- 1 Computer (1960s) → Multi-user Virtual Machines
- OS → Virtual Memory OS

Cloud: Uses VM

→ Flexibility to choose Software environment

- A) Application Software, and OS
- 1 Computer → Many VMs (1:n mapping)
- Many Computers → 1 VM (m:1 mapping)
- Many VM ↔ many Computer (dynamic n:m)
- 1 Computer ↔ 1 VM (1:1 mapping)

Virtualization: Helps to easily Parallelize DB Systems

- Each VM → runs DB code locally [Assume Homogeniety]
- **Object** → an unordered set of name/value pairs.
- **Object** → begins with `{` (left brace) and ends with `}` (right brace).

  and ends with `}` (right brace).

- Each name → followed by `:` (colon) and the name/value pairs are separated by `,` (comma).

- **Data types, syntax:** → basic types :
- **Object, Array, String, Number, Boolean, null**
1. Object
   a. an unordered collection of key:value pairs
   b. ':' character separating the key and the value,
   c. Comma-separated and enclosed in curly braces;
   d. keys must be strings distinct from each other

2. Array (an ordered sequence of values, comma-separated and enclosed in square brackets; the values do not need to be of the same type)

3. String (double-quoted Unicode (UTF-8 default), with backslash escaping)
4. Number → double precision floating-point as in JavaScript → depends on implementation

5. Boolean → true or false

6. null → (empty)

7. A value → (a) a string in double quotes, or (b) a number, or (c) true or false or null, or (d) an object or an array.

These structures can be nested.

Whitespace can be inserted between any pair of tokens.
JSON Values + JSON Objects

- JSON values can be:
  - A number (integer or floating point)
  - A string (in double quotes)
  - A Boolean (true or false)
  - An array (in square brackets)
  - An object (in curly brackets)
  - null

- **JSON objects** are written inside curly brackets,
- Objects can contain multiple name/values pairs:
  - \{ "firstName":"John" , "lastName":"Doe" \}
- This is simple, and equals to the JavaScript statements
  - firstName = "John"
  - lastName = "Doe"
JSON Arrays

- JSON arrays → inside square brackets.

- An array → can contain multiple objects:

  ```json
  {
    "employees": [
      { "firstName":"John" , "lastName":"Doe" },
      { "firstName":"Anna" , "lastName":"Smith" },
      { "firstName":"Peter" , "lastName":"Jones" }
    ]
  }
  ```

- Object "employees" is an array containing three objects (3 persons).
work with JSON within JavaScript - Example

```javascript
var employees = [
    { "firstName":"John" , "lastName":"Doe" },
    { "firstName":"Anna" , "lastName":"Smith" },
    { "firstName":"Peter" , "lastName": "Jones" }
];

first entry in the JavaScript object array →
employees[0].lastName;

Answer ← Doe

data can be modified →
employees[0].lastName = "Jonatan";
```
Converting a JSON Text → JavaScript Object

- **Common Task** → get JSON data from a web server
- → file or as an HttpRequest
- → convert the JSON data to a JavaScript object, → use the data in a web page (display).

- Making JSON Example - Object From String
- (A) Create a JavaScript string containing JSON syntax:

```javascript
var txt = '{ "employees" : [' +
    '{ "firstName":"John" , "lastName":"Doe" },' +
    '{ "firstName":"Anna" , "lastName":"Smith" },' +
    '{ "firstName":"Peter" , "lastName":"Jones" } ]}';
```
B. Convert a JSON text into a JavaScript object

- **USE** → JavaScript function `eval()`
- The `eval()` function → uses the JavaScript compiler

- Text → wrapped in parenthesis to avoid a syntax error:
  - `var obj = eval ("(" + txt + ")");`

- `eval()` function can compile and execute any JavaScript → a potential security problem.

- **Safe** → use a JSON parser to convert a JSON text to a JavaScript object. A JSON parser will recognize only JSON text and will not compile scripts.
B. Convert a JSON text into a JavaScript object

- Use the JavaScript object in your page:
- Example

  ```html
  <p>
  First Name: <span id="fname"></span><br />
  Last Name: <span id="lname"></span><br />
  </p>
  ```

  ```javascript
  <script type="text/javascript">
  document.getElementById("fname").innerHTML = obj.employees[1].firstName
  document.getElementById("lname").innerHTML = obj.employees[1].lastName
  </script>
  ```

- Try it yourself?
JSON support → JSON parsers are fast

- JSON → new browsers + new ECMAScript (JavaScript) standard.
- Web Browsers Support + Web Software Support
- Older browsers: JavaScript library →
  https://github.com/douglascrockford/JSON-js
  - Firefox (Mozilla) 3.5
  - Internet Explorer 8
  - Chrome
  - Opera 10
  - Safari 4

- jQuery
- Yahoo UI
- Prototype
- Dojo
- ECMAScript 1.5

Example:

**Object : a Person**

- string fields for first name and last name,
- a number field for age,
- contains an object representing the person's address,
- contains a list (an array) of phone number objects.
Example

```json
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021"
  },
  "phoneNumber": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ]
}
```
- JSON → a subset of JavaScript
- We can → parse JSON text into an object by JavaScript's eval() function.

Example: if JSON data (last slide) is in a JavaScript string variable contact

Then, create the JavaScript object p

```javascript
var p = eval("(" + contact + ")");
```

The contact variable must be wrapped in parentheses to avoid an ambiguity in JavaScript's syntax.
JSON - Schema

- **USE 1:** JSON parses/ browsers → Firefox 4 and IE 8

- **USE 2:** JSON Schema → provides a contract for JSON data for a given application (Similar to the XML Schema for XML.)

  JSON Schema → provide validation, documentation, and interaction control of JSON data.

  JSON Schema → is based on the concepts from XML Schema, RelaxNG, and Kwalify,

  → is intended to be JSON-based (JSON data in the form of a schema can be used to validate JSON data)

  → the same serialization/deserialization tools can be used for the schema and data (it is self descriptive).
```json
{
    "name": "Product",
    "properties": {
        "id": {
            "type": "number",
            "description": "Product identifier",
            "required": true
        },
        "name": {
            "type": "string",
            "description": "Name of the product",
            "required": true
        }
    }
}
(Continued) ....
"price":
{
  "type":"number",
  "minimum":0,
  "required":true
},
"tags":
{
  "type":"array",
  "items":
  {
    "type":"string"
  }
}
"stock":

{  
  "type":"object",
  "properties":

{  
  "warehouse":

{  
  "type":"number"
  }
  },

  "retail":

{  
  "type":"number"
  }
  }
}
}
The JSON Schema can be used to test the validity of the JSON code below:

```json
{
    "id": 1,
    "name": "Foo",
    "price": 123,
    "tags": ["Bar","Eek"],
    "stock": { "warehouse":300, "retail":20 }
}
```
Use in Ajax

- Ajax → is a term → ability of a webpage to request new data
  (after it has loaded into the web browser, usually in response to user actions on the displayed webpage)

- Ajax model → the new data is usually incorporated into the user interface display dynamically

- An example →
  while the user is typing into a search box → client-side and server promp with values from the Database

- Ajax commonly used XML earlier, now may use JSON
Example (JavaScript code):

- a client using XMLHttpRequest to request data in JSON format Server-side programming is needed

```javascript
var my_JSON_object = {};
var http_request = new XMLHttpRequest();
http_request.open("GET", url, true);
http_request.onreadystatechange = function ()
{
    var done = 4, ok = 200;
    if (http_request.readyState == done &&
        http_request.status == ok)
    {
        my_JSON_object = JSON.parse(http_request.responseText);
    }
}
http_request.send(null);
```
Comparison with other formats:

- **XML**
  1. describe structured data,
  2. to serialize objects.

- **JSON**
  1. programs for XML ↔ JSON conversions are available,
    2. JSON data is small in size
    3. XML data can have many converted forms

(In XML there are alternative ways to encode the same information → some values can be represented both as child nodes and attributes.) → automated data exchange complicated (unless the used XML format is strictly specified as programs need to deal with many different variations of the data structure.)
XML – example 1 (JSON data in XML)

```xml
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber type="home">212 555-1234</phoneNumber>
    <phoneNumber type="fax">646 555-4567</phoneNumber>
  </phoneNumbers>
</person>
```
<person firstName="John" lastName="Smith" age="25">
  <address streetAddress="21 2nd Street" city="New York" state="NY" postalCode="10021" />
  <phoneNumbers>
    <phoneNumber type="home" number="212 555-1234"/>
    <phoneNumber type="fax" number="646 555-4567"/>
  </phoneNumbers>
</person>
Why JSON? → JSON is faster and easier than XML:

Using XML → Fetch an XML document
(a) Use the XML DOM to loop through the document
(b) Extract values and store in variables

Using JSON → (a) Fetch a JSON string
(a) `eval()` the JSON string
Summary and Conclusions

1. Key Value store is good for Google Docs type of cloud Computing [Requirements: Store, find and retrieve].

2. Relational Model DBMS is not needed [General Purpose Query Requirements are missing].

- Availability is important and Scalability is important

- System is prone to disconnections and failures at component level → Key Areas: Replication studies and on-line generation of replicated copy/backup

- JSON → uses JAVASCRIPT object (Javascript does not need prior declaration of variable).