Cloud-based Database Systems
(No SQL Databases)

Reference: Chapter 19.9, Database Systems Concepts, Silberschatz, et al.

Lecture- Sept 2018

COV888 - (Semester 1) S. Bhalla

History: Distributed Database System

- Example – Distributed Oracle

- 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

- Summary - 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:

    Linux OS (IITD), Distributed ORACLE

Typical case - Applications

- Heterogeneous distributed database
  - 1. Sites (LAN) → 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 (Internet, web) → may not be aware of each other

→ provide limited facilities for cooperation in transaction processing

19.2 Distributed Data Storage

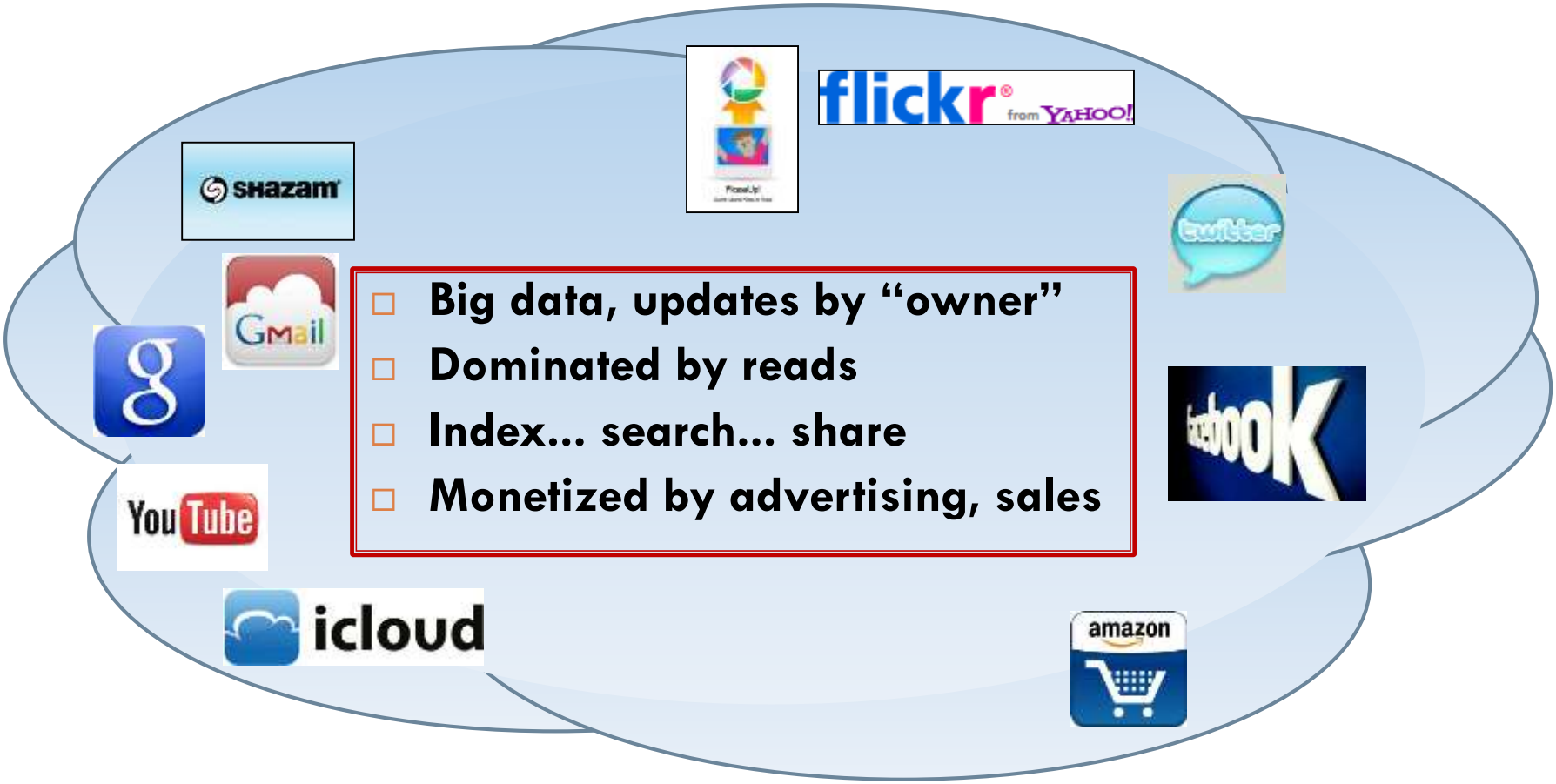
- Database → Relational data model
- Replication
  - System → multiple copies of data at different sites
    → faster retrieval and fault tolerance.
- Fragmentation
  - Relation → partitioned into many fragments
    → stored in distinct sites
- Replication and fragmentation can be combined
  - Relation → partitioned into several fragments
  - System → identical replicas of each such fragment.

Cloud Computing: The New Thing

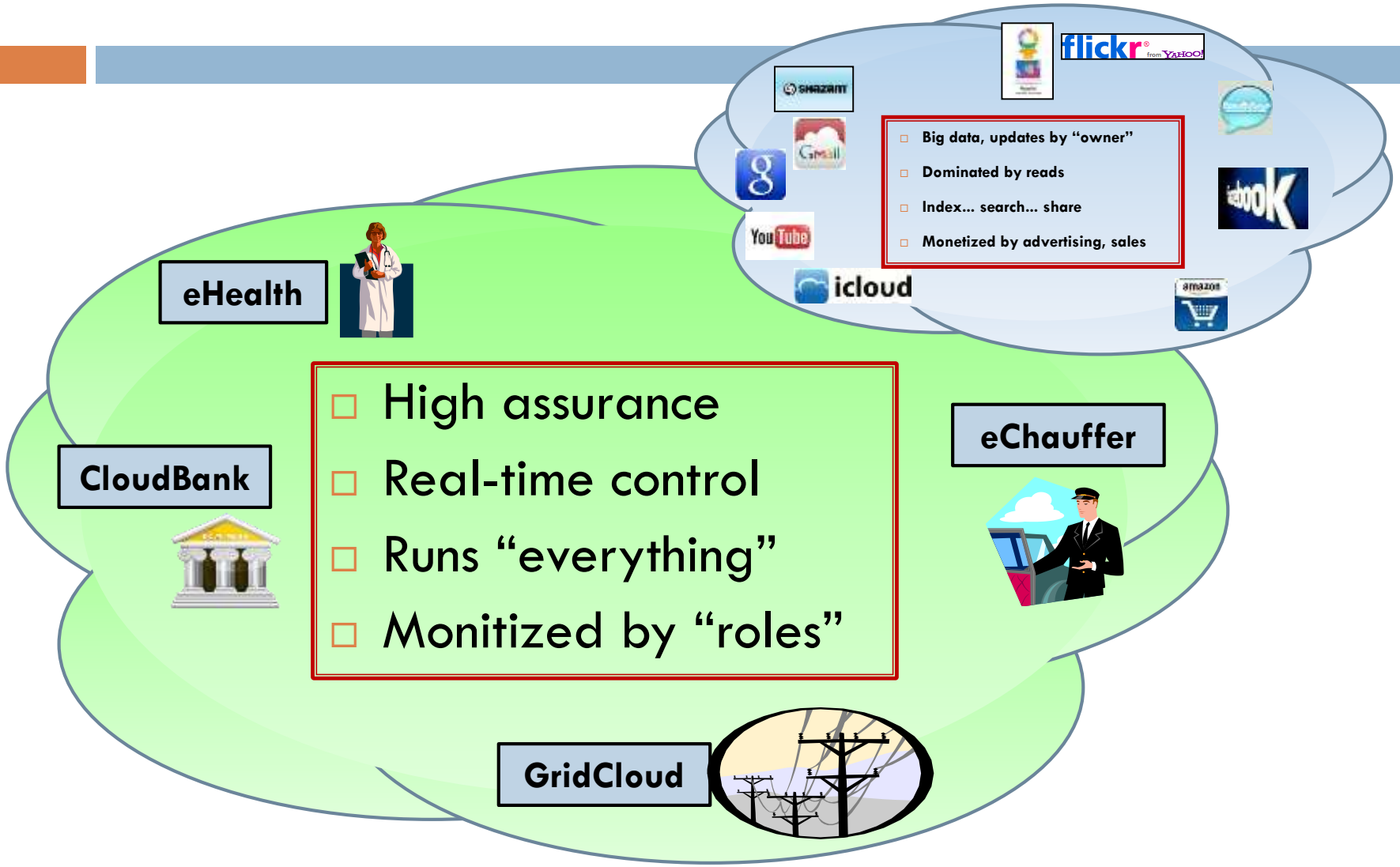
- The style of computing → supports web services, search, social networking
- Increasingly powerful and universal
- Enables a new kind of massively scaled, elastic applications

- Goals: understand the cloud, its limitations, and how to use them for telephony, banking (year 2025)
- Invent "highly assured cloud computing" options

Year 2017's Cloud: Surprisingly limited



Year 2025's cloud?



Clouds are hosted by data centers

- Huge data centers, far larger than past systems
- Very automated: far from where developers work. Close to → where power is generated
  (ship bits... not watts)
- Packed for high efficiency. Each machine hosts many applications (usually in lightweight virtual machines to provide isolation)
- Scheduled to keep everything busy (but overloads hurt performance so we avoid them)

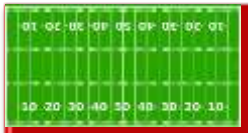
Clouds are cheaper… and winning…

Range in size from "edge" facilities to megascale.

Incredible economies of scale

Approximate costs for a small size center (1K servers) and a larger, 50K server center.

Technology	Cost in small-sized Data Center	Cost in Large Data Center	Cloud Advantage
Network	$95 per Mbps/ month	$13 per Mbps/ month	7.1
Storage	$2.20 per GB/ month	$0.40 per GB/ month	5.7
Administration	~140 servers/ Administrator	>1000 Servers/ Administrator	7.1



Each data center is
11.5 times
the size of a football field

Slide provided by Roger Barga, Head of Cloud Computing, Microsoft

Key benefits?

- Machines busier, earn more $'s for each $ investment
  - Hardware handled a whole truckload at a time
- Applications far more standardized
  - Automated management: few "sys admins" needed
  - Power consumed near generator: less wastage
  - Data center runs hot, wasting less on cooling
  - Can "rent" resources rather than owning them
- Supports new, extremely large-scale services
  - Elasticity to accomodate surging demands
  - Can accumulate and access massive amounts of data
    - But must read or process it in a massively parallel way
  - Enables overnight emergence of major companies, but scalability model does require new programming styles, and imposes new limits

Assurance properties

- Unfortunately, today's cloud
  - Has a limited security model focused on credit card transactions
  - Weakens consistency to achieve faster response times: the cloud is "inconsistent by design"
  - Pushes many aspects of failure handling to clients
- Model supported by the "CAP" and "FLP" theorems, which are cited by many application designers
- Instead, cloud favors "BASE"

Acronyms

- CAP: A theorem that says one can have just two from {Consistency, Availability, Partition Tolerance}
- FLP: A theorem that says it is impossible to guarantee "live" fault-tolerance in asynchronous systems (here, "live" ≡ certain to make progress)
- BASE: A cloud computing methodology that seeks "Basically available soft-state services with eventual consistency" and is popular in the outer layers (first tier) of the cloud. The opposite of ACID
- ACID: A database methodology: offers guaranted {Atomicity, Consistency, Isolation and Durability}.

COV888: How to do better!

- Future cloud will need stronger guarantees than we see with today's cloud
  - How can we achieve those?
  - Can still be "scalable"?

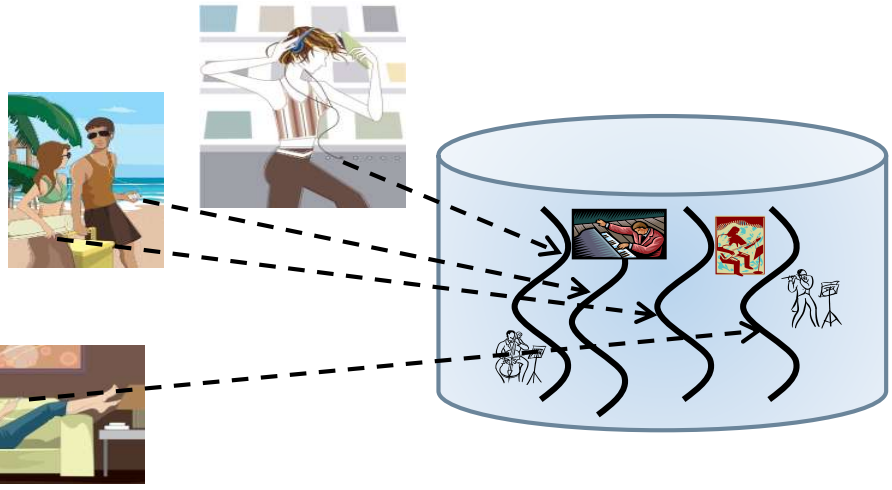
- Betting/ Hoping that the cloud will win
  - Cheaper than other options...
  - ... the cheaper option usually wins!
  - Technology also advances over time !

Making the cloud highly assured

- Overcome limitations → FLP and CAP
- Define new assurance goals (that might still be) new forms of security and consistency but are easier to achieve
- Only consider things that are real enough to be implemented and demonstrated to scale well
- Perform in a way that would compete with today's cloud platforms → A practical mindset.
- Use theoretical tools when theory helps with goals.

COV888: Topics Covered

- Cloud as having three main parts
  - The client side: Everything on your device
  - The Internet, as used by the cloud
  - Data centers, which themselves have a "tiered" structure
    - Like a dedicated and personal computer
    - Yet massively scaled with many moving parts
- Special theme: *high assurance*



The Old World and the New

- **LAN's world:** we replicated servers for speed and availability, but maintained consistency

- **New world:** *scalability* matters most of all
  - *Focus is on extremely rapid response times*
  - Amazon estimates that each millisecond of delay has a measurable impact on sales!

- **AIMs -** our premise is that we can have scalability and also have other guarantees that today's cloud lacks

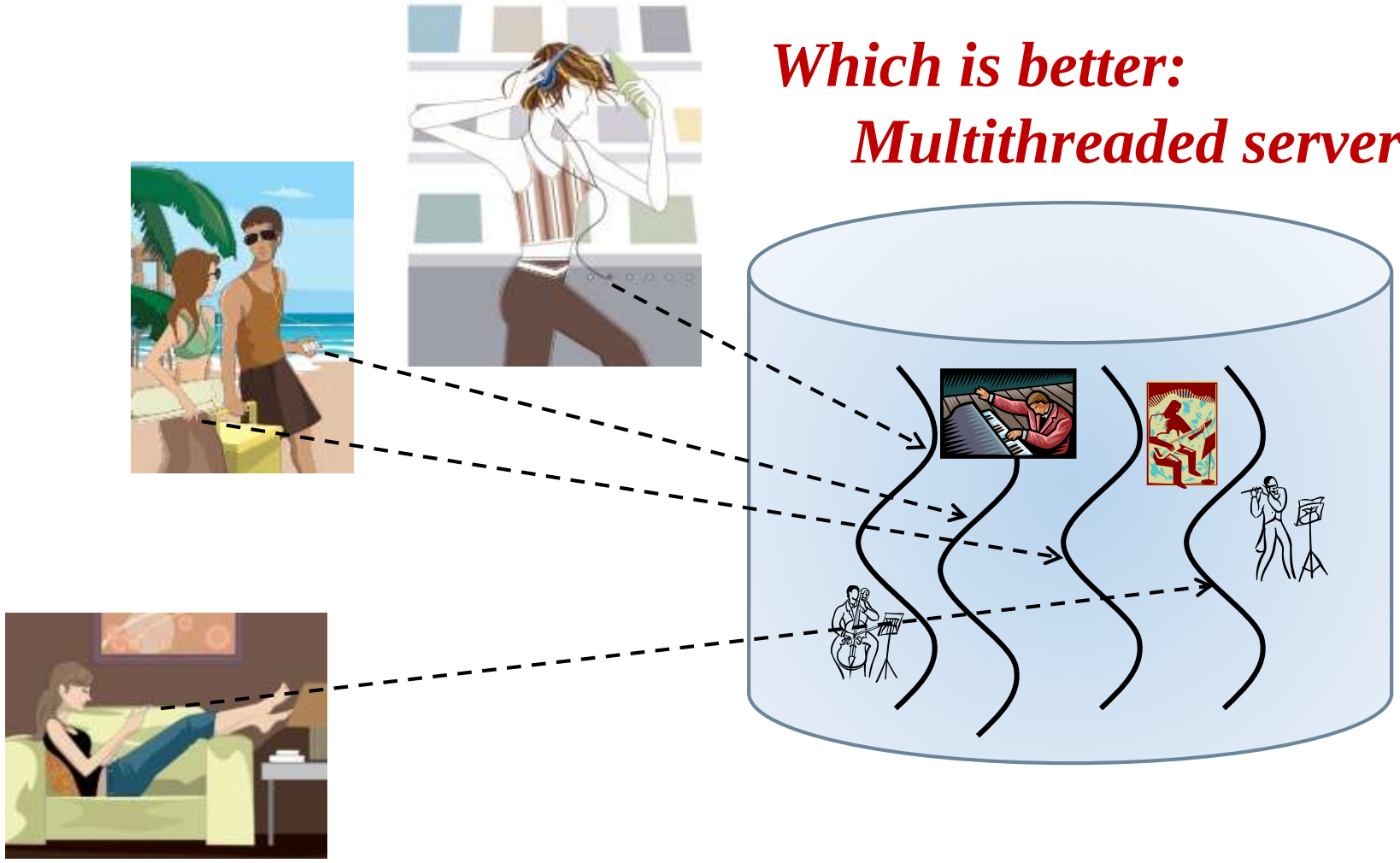
High Assurance: Many (conflicting) goals

- Security → Only authorized users (who are properly authenticated) can perform actions
- Privacy → Data doesn't leak to intruders
- Rapid response → despite failures or disruption
- Consistency and coordinated behavior
- Ability to overcome attacks or mishaps
- Guarantee that center operates at a high level of efficiency and in a highly automated manner
- Archival protection of important data

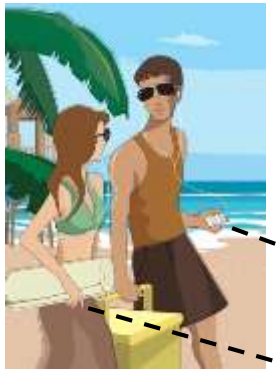
Many questions

- If we run high assurance solutions on today's cloud → what parts of the standards would limit or harm our assurance properties?
- Goal → to leverage the cloud or even run on standard clouds, yet to improve on normal options
- This forces us → to look hard at how things work

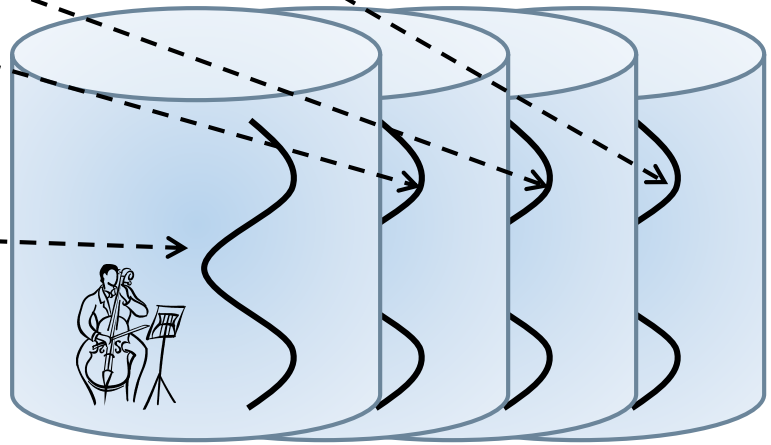
Today's cloud focuses on easy stories



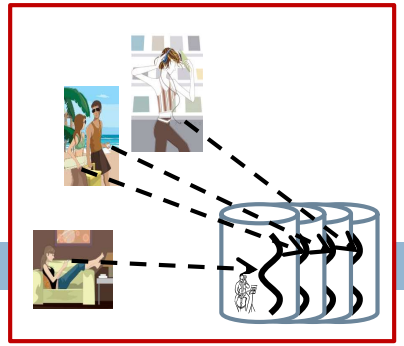
Today's cloud focuses on easy stories



Which is better:
Multithreaded servers?
Or multiple single-threaded servers?



Which scales best?



- Build it the easy way!
  - One VM per server
  - Server handles one user
  - Make the server single threaded if possible

- Why?
  - Better fit to the hardware (no lock/memory contention)
  - Quicker way to build it, but is not application specific (reuses existing application)

Acronyms! (early stage problems)

- Cloud is that it has a many acronyms:

  IaaS, SaaS, PaaS, SOAP, AWS, EC2, S3...

  - → a *very* confusing landscape!
  - Business perspective on the cloud →

  only needs to focus on a few of them, as a starting point

- What does the "aaS" mean?
  - Cloud vendors sell "services"
  - "aaS" == "as a Service"

The Important *aaS options

- **I**nfrastructure. (IaaS: Infrastructure as a Service)
  - Cloud vendor rents out some hardware
    - A network, perhaps a wide-area network
    - A machine, always "virtual" but perhaps just for you
    - A file server, again virtual, but you can save files in it
  - They operate this for you, and you pay for what you think you need (or sometimes, for what you use)
  - And they sell backup services too

- For example, you could rent a private Internet from AT&T, or 500 computers from Amazon EC2
  - AWS is elastic: you rent and pay by the hour
  - AWS can accommodate huge swings in your needs

The Important *aaS options

- **S**oftware. (SaaS: Software as a Service)
  - Cloud vendor runs some software that you use remotely
  - Classic example: SalesForce.com has a sophisticated infrastructure that manages your sales contact data

- In effect you "outsource" your sales support system and SalesForce.com runs it for you
  - Other SaaS options: accounting, billing, email, document handling, shared files…
  - They also apply patches, fix bugs…

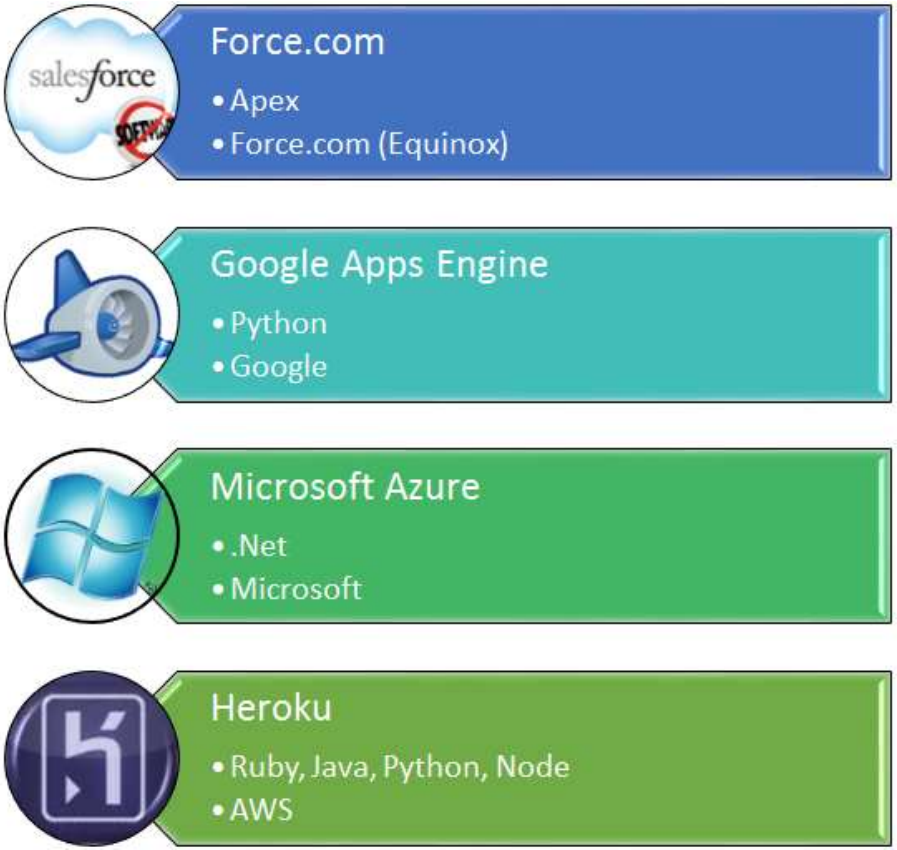
The Important *aaS options

- **P**latform. (PaaS: Platform as a Service)
  - Cloud vendor creates a sophisticated platform (typically a software environment for some style of computing, or for database applications)
  - Your folks use it to create a custom solution
  - Cloud vendor runs your solution in an elastic way

- They promise that if you use their PaaS solution, you'll benefit from better scalability, performance, ease of development or other advantages

The Important *aaS options

- **P**latform. (PaaS: Platform as a Service)



… these aren't the whole cloud

- The cloud *mixes* many models
  - Some integrate humans activity, such as outsourced audio-to-text, or Amazon's Mechanical Turk
  - There are companies with specialized roles
    - Akamai: The most famous data hosting company, → Successful for storing videos and images used in your web pages → Specialize in rapid data delivery
    - DoubleClick: You leave a frame on your web page, they put the perfect advertisement for a particular user
  - There are even cloud "HPC systems"!

    → (Rent on demand)

But some standards pervade…

- The cloud really took off →

  as an outgrowth from web sites and browsers
  - First we had browsers, HTML (a use of XML), HTTP, SSL
  - Then people had the idea of doing "client server" computing using browser web pages!
    - Called SOAP. A program makes a method call on a remote server… they encode it as a special web page
    - … this is sent to the server just as if it was a web request from a browser (in fact you can do it by hand…)
    - … result comes back in a special SOAP web page, extracted and returned to the calling program. Voila!

Web interoperability

- The web is about interoperability
- It is very easy to integrate →
- Data from multiple sources (e.g. Netflix sends you a web page and the video comes via Akamai)
  - Different styles of computing (e.g. Weather.com fills a page with their content (the images come from Akamai), but the weather forecasts are from HPC computing systems and the advertisements are from DoubleClick. The ads might include a video hosted on YouTube, but Akamai might be the real source that sends the data…
  - By agreeing that "at the end of the day, web pages are the lingua franca" a great leap forward happened

(Web pages are inefficient…)

- The encodings used in the web are terribly inefficient, though
  - So they made browsers extensible
  - You get "plug ins" from Adobe, GZip, Microsoft, … and those plug-ins "extend" the browser to understand special data representations
- Modern browsers can download and run full programs coded in Javascript, Silverlight, Caja or even true Java… and these programs can do anything at all

Open source

- The cloud has hugely benefitted from
  - **open source** (basically, source for programs is made available to customers),
  - **free open source** (same, but no fee for use), and
  - **open development** (many developers at many companies contribute).

- In fact nothing about the cloud demands "open."
- But these are certainly powerful factors that help explain the vibrant cloud ecosystem.

Open source debate

- Many companies debate open source
  - Some have policies against it

- Yet they run Linux on their servers, build programs in C++ using gcc, allow employees to install their favorite browser add-ons, use Mono to create Linux versions of their Windows applications
- Java compiles to JIT code that reverse compiles back to Java source (Company employees who contribute to some open source projects…)

Deeper connection to cloud

- The cloud is a world of open standards
  - First time → cloud tore down the high protectionist walls of proprietary products
  - At many levels, we can see how things work and jump in and modify things
  - Plug-and-play… from the client system into the network and right up to the datacenter!

- The cloud is a world of easily interconnected component technologies that play together nicely
  - And openness has been a key enabler in this happening

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 →
- → Huge Data Centers + Web = CLOUD

Data Center →Software / Platform / Infrastructure → SaaS / PaaS / IaaS

- 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

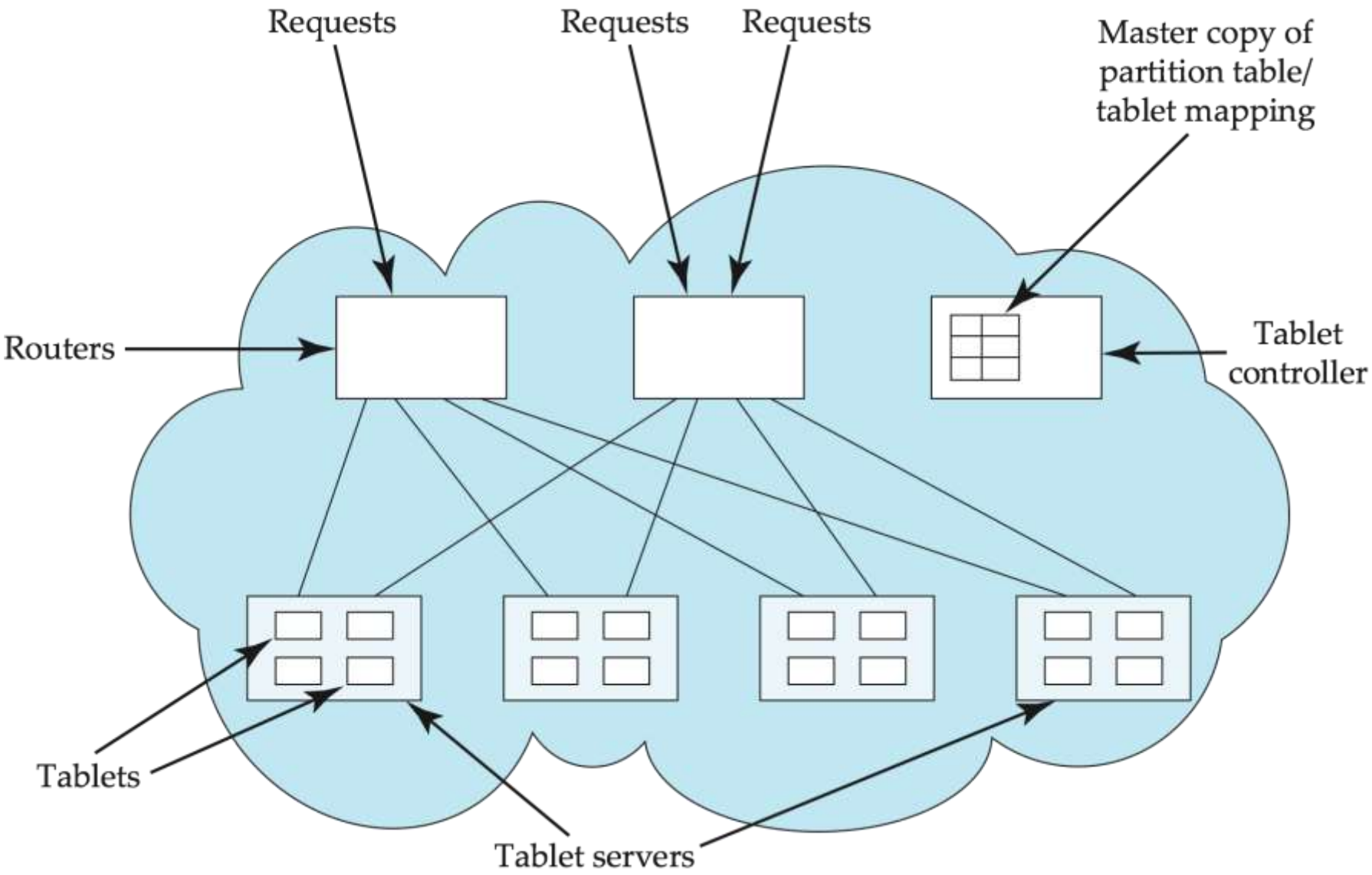
- Customer
- → Buy Computer Time, Capacity , Software
- ← at any time

- 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

Figure 19.07


Requests
Requests
Requests
Master copy of partition table/ tablet mapping
Routers
Tablet controller
Tablets
Tablet servers

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 → 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

→ provides a web interface to Dynamo →

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-storge → 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 Interger Value

- Bigtable → Key :

  [Appln,Table](record-identifier, attribute-name, Time-stamp)

Processes and Data Forms

- Applications on the WEB

- A) Common use → PHP and APACHE web server

  Database contents → XHTML, HTML + Javascript + CSS

- B) E-Commerce Sites → JSP and TOMCAT web server

  Web services → XML, KML, (xml handling xslt, xquery)

- C) Facebook, Google, Yahoo → JavaScript servers-side and JavaScript Client-side

  JSON (JavaScript Object Notation) easy handshake

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

JSON

- Object → an unordered set of name/value pairs.
- Object → begins with { (left 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

JSON

- 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)

JSON

- 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:
- {
- "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

- 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:

- 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
-
- First Name:
- Last Name:
-

- <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

JSON

- Reference: http://www.json.org/
- 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.

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"

Partitioning : Retrieval

- 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)

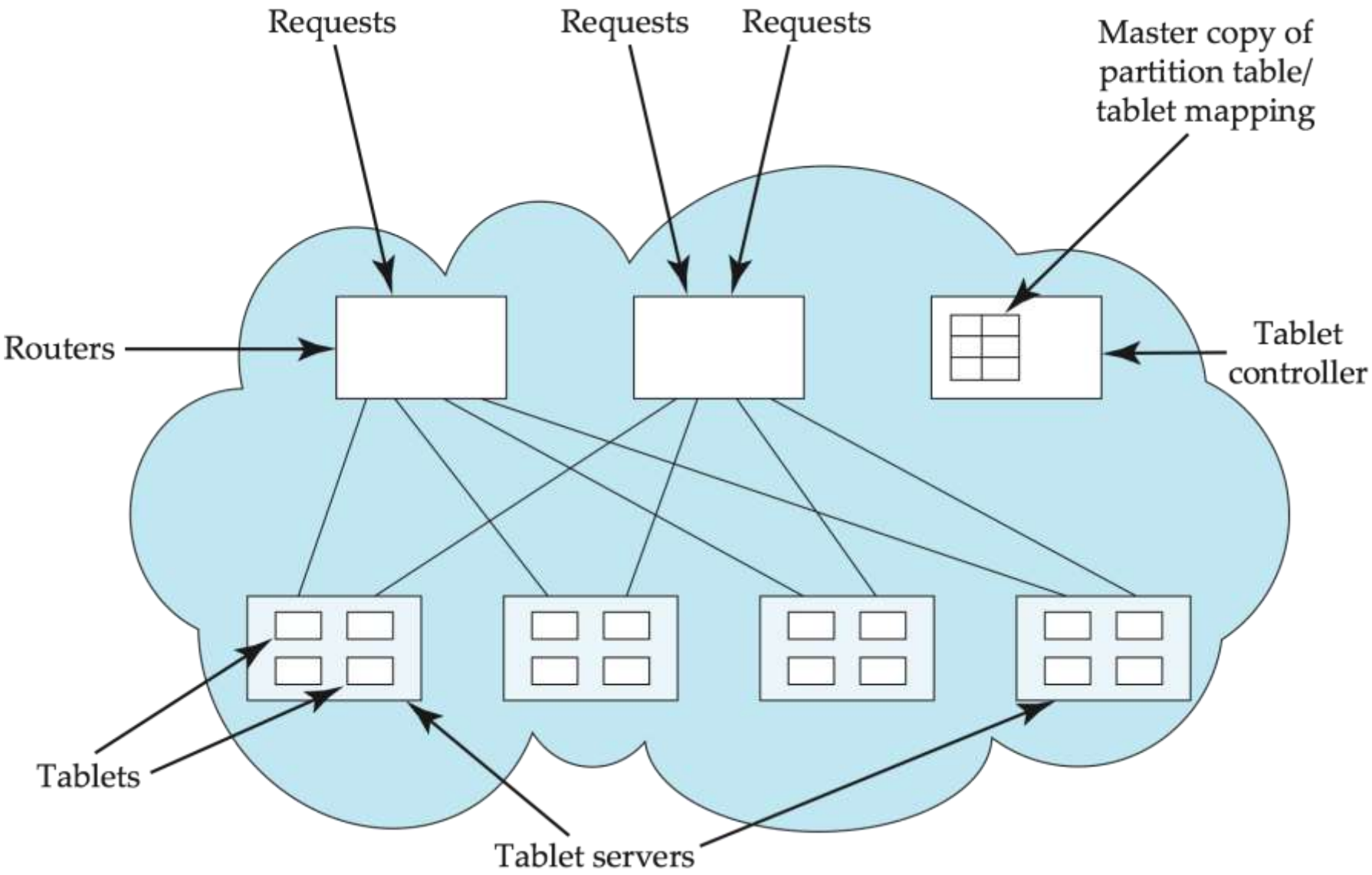
Partition → uses search key (record Identifier)

→ gmail (record Identifier “your Name”)

get(“your Name”) → 1 tablet

[If get (kv) → multi-site : Load on System ?

Figure 19.07


Requests
Requests
Requests
Master copy of partition table/ tablet mapping
Routers
Tablet controller
Tablets
Tablet servers

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 → 1000s of sites (10000s Tablets)
- Many Sites → [X] down : Cloud DB (availability)

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

- [Replicate] → 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 → need many round trip messages

  → Connect user to nearest remote copy

  → → data (near) : Application Servers

- Web Application [GMAIL] → High Availability
- Consistency → : Allow updates to proceed (if Network Partition/site Fails) ; LATER
- Final State → DB is consistent
- Multi-master Replication with Lazy Propagation of updates → use PERSISTENT Messages

Persistent Message Networks

- Message Network →

Guarantee (updates will reach): Asynchrous 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 Copmuters → 1 VM (m:1 mapping)
  - Many VM ←→ many Computer (dynamic n:m)
  - 1 Compuer ←→ 1 VM (1:1 mapping)
- Virtualization : Helps to easily Parallelize DB Systems
- Each VM → runs DB code locally [Assume Homogeniety]