Cloud Computing Overview*

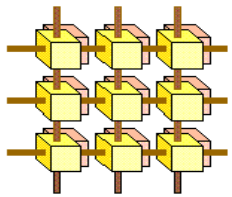
*some material excerpted from slides of
Roy Campbell, Reza Farivar at UIUC

ECE 6102

Part 1: Background and Cloud Basics

Georgia Institute of Technology

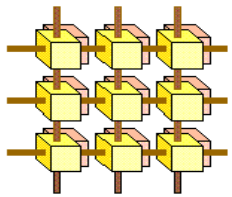
Critical Systems Laboratory



What is Cloud Computing??

NIST Definition - July 2011:

"Cloud computing is a model for enabling *ubiquitous*, convenient, *on-demand* network access to a *shared* pool of *configurable* computing resources (e.g., networks, servers, storage, applications, and services) that can be *rapidly provisioned and released* with minimal management effort or service provider interaction."

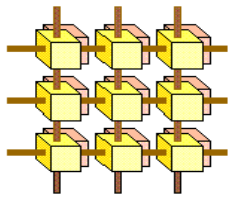


Cloud Characteristics

- On-demand self service
- Ubiquitous network access
- Location-independent resource pooling
- Rapid elasticity
- *Multi-tenancy*
  - Different cloud apps run in the same massive datacenters using the same resources
- *Pay per use*
  - Metered access, utility computing
  - Amazon EC2: current prices range from $0.10/hour to $3.20/ hour for one general-purpose VM depending on resource needs

Georgia Institute of Technology

Critical Systems Laboratory

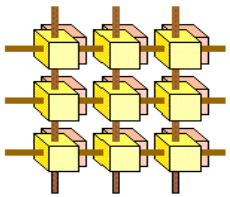


Cloud Security Concerns

- Reluctance to store and operate on sensitive data in clouds
  - Multi-tenancy: sharing of resources between different users, who might even be direct competitors
  - Cloud platform provider has full and unfettered access to file systems and even VM state(application memory)
- Encryption works well for cloud data storage and retrieval but what about applications?
- Operating on encrypted data is a hot research topic

Georgia Institute of Technology

Critical Systems Laboratory



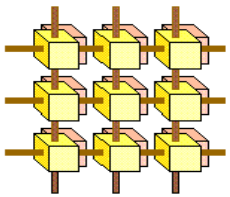
Utility Computing

"Computing may someday be organized as a public utility, just as the telephone system is organized as a public utility."

John McCarthy, 1961

Georgia Institute of Technology

Critical Systems Laboratory



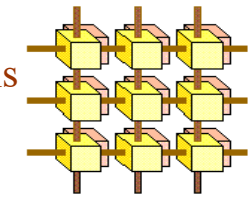
Perils of Corporate Computing

- Own information systems ☺
- However
  - Capital investment ☹
  - Heavy fixed costs ☹
  - Redundant expenditures ☹
  - High energy cost, low CPU utilization ☹
  - Dealing with unreliable hardware ☹
  - High-levels of overcapacity (Technology and Labor) ☹

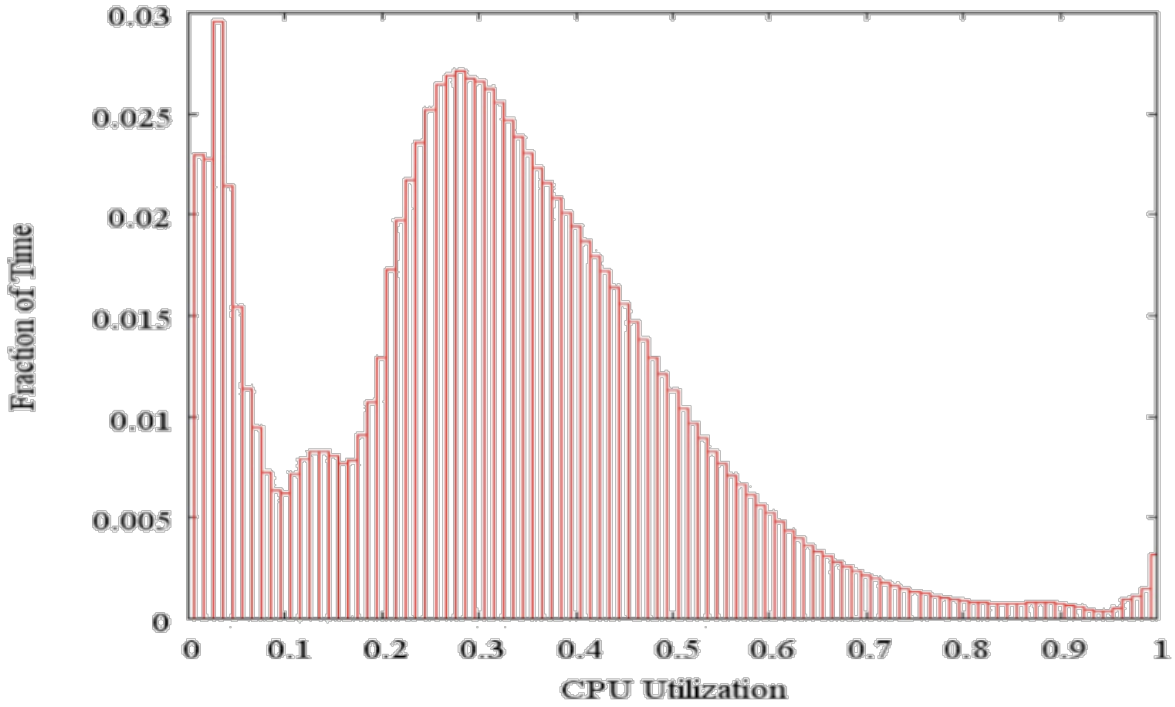
NOT SUSTAINABLE

Georgia Institute of Technology

Critical Systems Laboratory



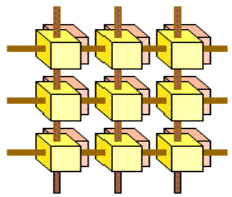
Google: CPU Utilization



Activity profile of a sample of 5,000 Google Servers over a period of 6 months

Georgia Institute of Technology

Critical Systems Laboratory

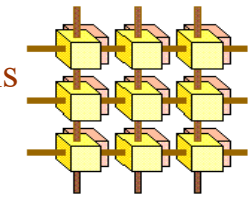


Utility Computing

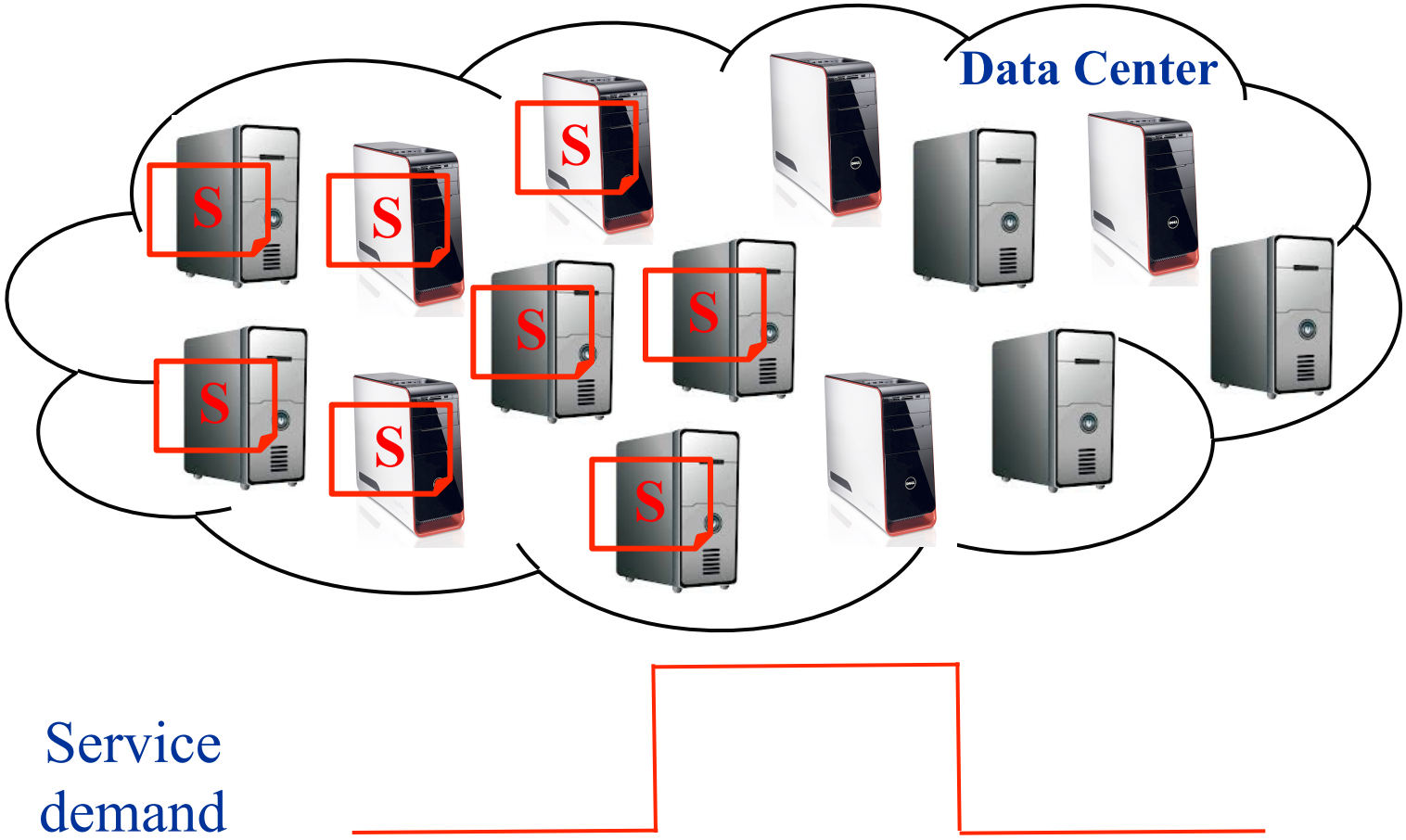
- Let economy of scale prevail
- Outsource all the trouble to someone else
- The utility provider will share the overhead costs among many customers, amortizing the costs
- You only pay for:
  - the amortized overhead
  - Your *real* CPU / Storage / Bandwidth usage
- Great for start-ups: start small and expand easily when things take off!!

Georgia Institute of Technology

Critical Systems Laboratory

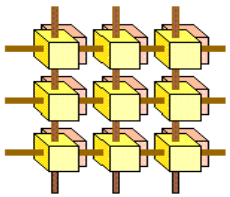


Dynamic Provisioning



Georgia Institute of Technology

Critical Systems Laboratory



Why Utility Computing Now

- Large data stores
- Fiber networks
- Commodity computing
- Multicore machines

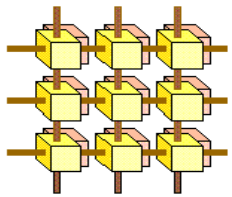
+

- Huge data sets
- Utilization/Energy
- Shared people

Utility Computing

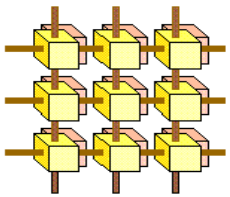
Georgia Institute of Technology

Critical Systems Laboratory



Delivery Models

- Cloud provider direct to consumer
  - gmail, other Google apps
  - Apple iCloud
  - Amazon s3
  - Microsoft Office 365
- Cloud provider to service provider to consumer
  - Netflix runs on Amazon Web services
  - Snapchat runs on Google App Engine

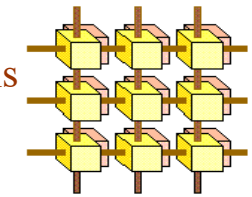


Delivery Models (continued)

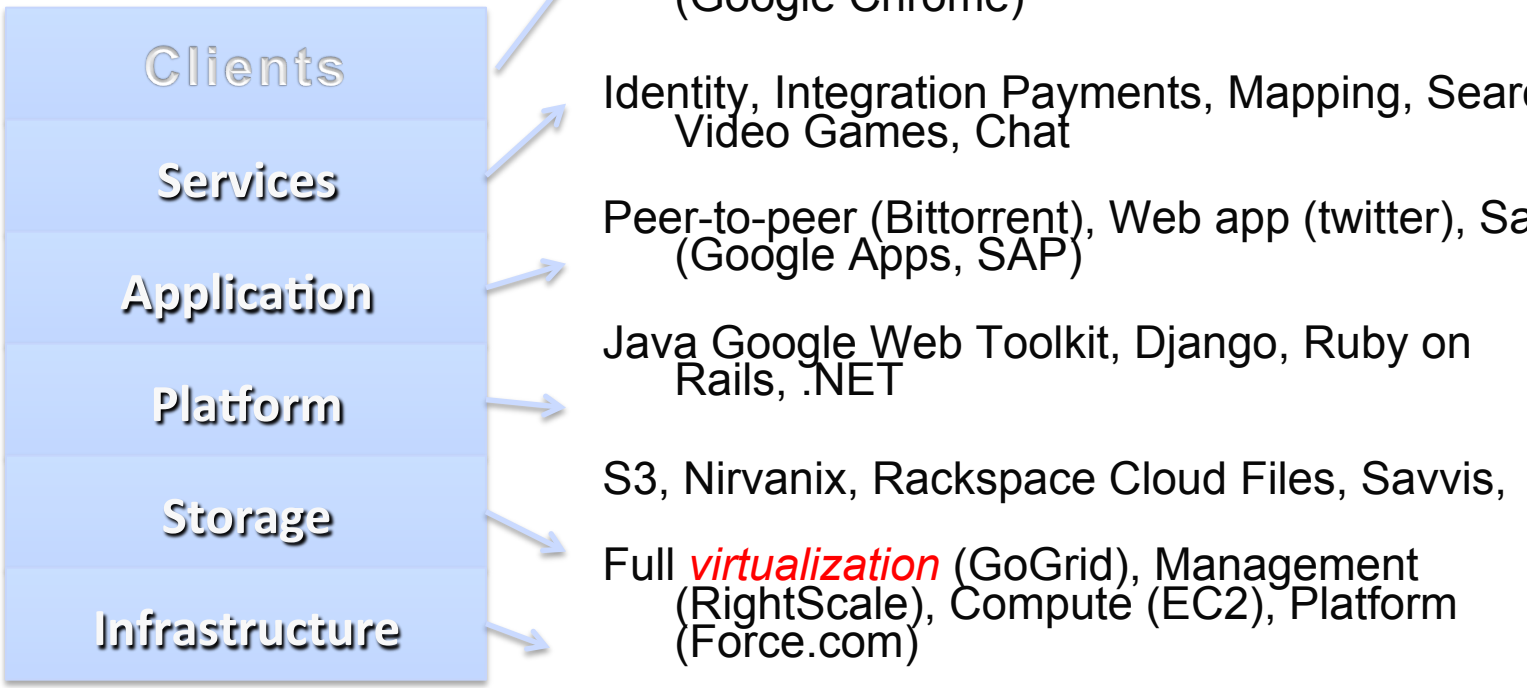
- Software as a Service (SaaS) (CP direct to consumer)
  - Use provider's applications over a network
  - SalesForce.com, gmail
- Platform as a Service (PaaS) (CP to SP to consumer)
  - Deploy customer-created applications to a cloud
  - Google App Engine, Microsoft Azure .NET
- Infrastructure as a Service (IaaS) (CP to SP to consumer)
  - Rent processing, storage, network capacity, and other fundamental computing resources
  - Run whatever software platform you want with app on top of it
  - Amazon EC2, S3

Georgia Institute of Technology

Critical Systems Laboratory



Software Stack



Mobile (Android), Thin client (Zonbu) Thick client (Google Chrome)

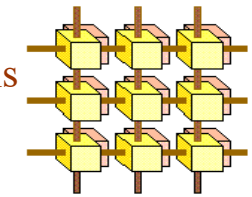
Identity, Integration Payments, Mapping, Search, Video Games, Chat

Peer-to-peer (Bittorrent), Web app (twitter), SaaS (Google Apps, SAP)

Java Google Web Toolkit, Django, Ruby on Rails, .NET

S3, Nirvanix, Rackspace Cloud Files, Savvis,

Full *virtualization* (GoGrid), Management (RightScale), Compute (EC2), Platform (Force.com)



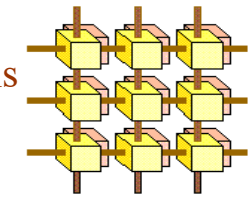
Technologies Enabling Cloud Growth

- Virtualization, Containers
  - Apps/services run in virtual machines or containers that can run on any physical machine in data center
  - Facilitates load balancing (VM migration) and app elasticity (add more VMs as demand increases, eliminate VMs as demand decreases)
- REST
  - REpresentational State Transfer
  - Simplified programming paradigm for delivering services via the Web (http)
- Big Data technologies
  - Hadoop/MapReduce/Dataflow for processing large amounts of data
  - noSQL for storing/retrieving large amounts of data

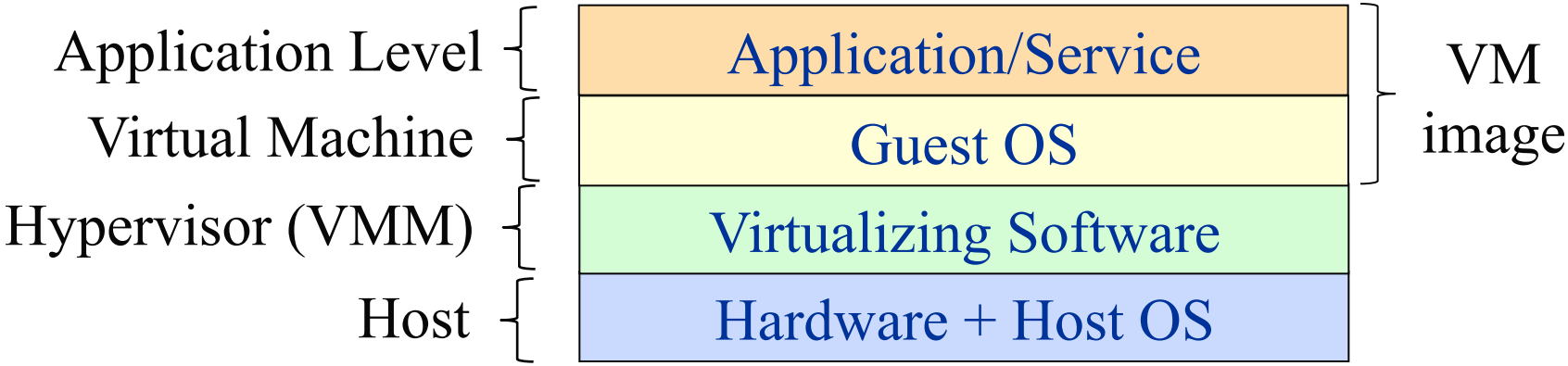
Part 2: Technology Overview

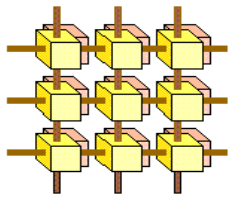
Georgia Institute of Technology

Critical Systems Laboratory



Full Virtualization



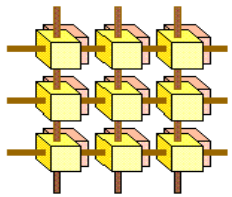


Full Virtualization, continued

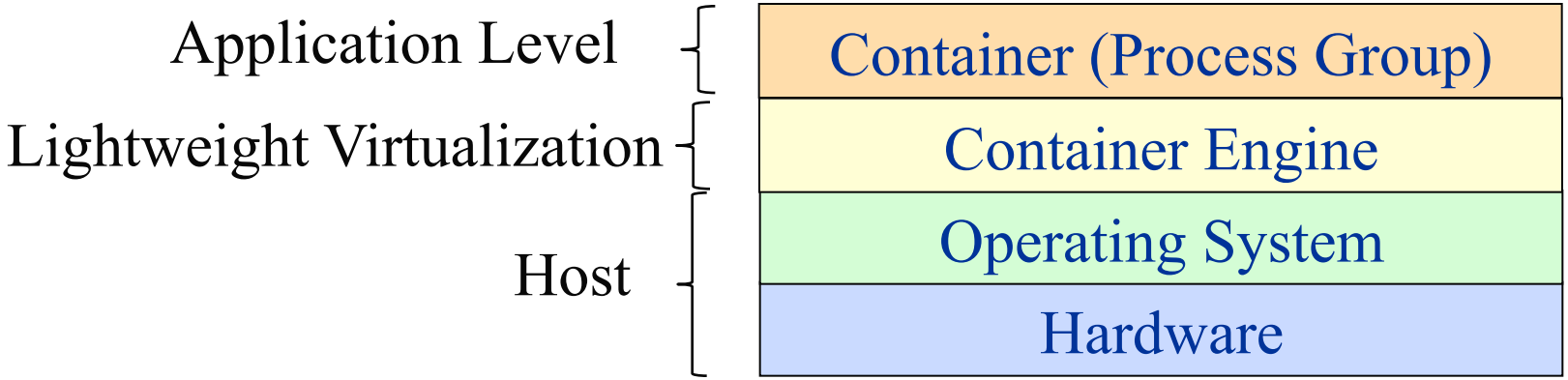
- Examples of full virtualization: KVM, Xen, VMware
- With virtualization, multiple VMs (even with different operating systems) can run over one VMM on the same physical machine
- Different VMs are strongly isolated from each other
  - One VM cannot access any resources (memory, file system, network connections) of another VM
  - This is enforced by the VMM and provides a security guarantee to VM owners (assuming VMM is not compromised)

Georgia Institute of Technology

Critical Systems Laboratory

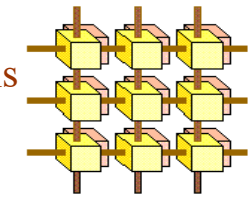


Containers



Georgia Institute of Technology

Critical Systems Laboratory

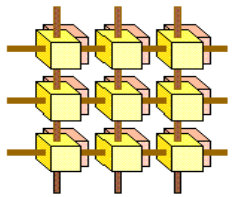


Containers, continued

- Examples of container engines: LXC, Docker
- Containers provide some isolation but it is not as strong as VM isolation
- A container is much lighter weight than a VM
- The number of containers that can be run on a single physical machine is much greater than the number of VMs per physical machine

Georgia Institute of Technology

Critical Systems Laboratory

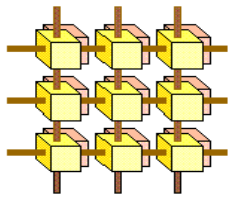


REST

- http is the new transport protocol (distributed applications and services communicate via http)
- two paradigms for distributed programming via http (Web services)
  - SOAP (simple object access protocol)
  - REST (representational state transfer)

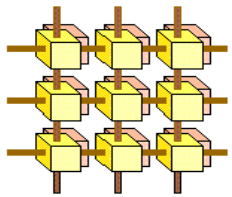
Georgia Institute of Technology

Critical Systems Laboratory

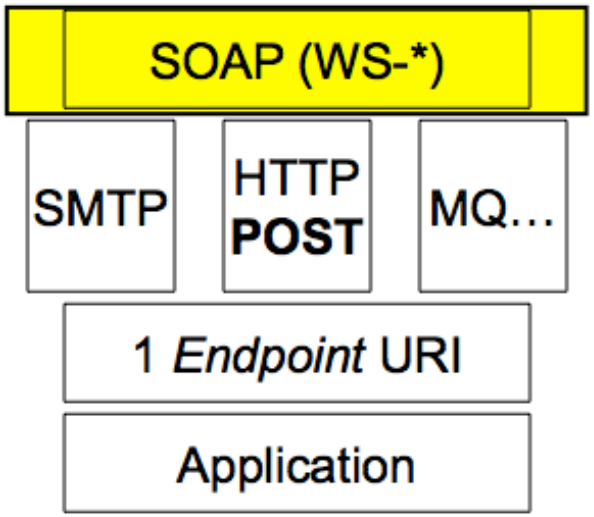


Web Services via SOAP and REST

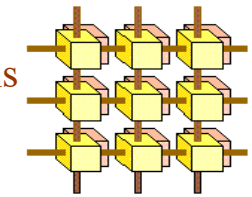
- SOAP
  - Full distributed object system with IDL (WSDL)
  - Arbitrary method calls
  - Stateful services with stateful interactions
  - Support for advanced features: security, transactions, etc.
  - Tightly coupled distributed applications: core Google apps, enterprise applications
- REST
  - REpresentational State Transfer
  - No IDL
  - Simplified stateless interactions (self-describing messages)
  - Only HTTP get, head, put, post, delete methods
  - State maintained on clients and resources, accessible by other services
  - Loosely coupled distributed applications: twitter, flickr, …



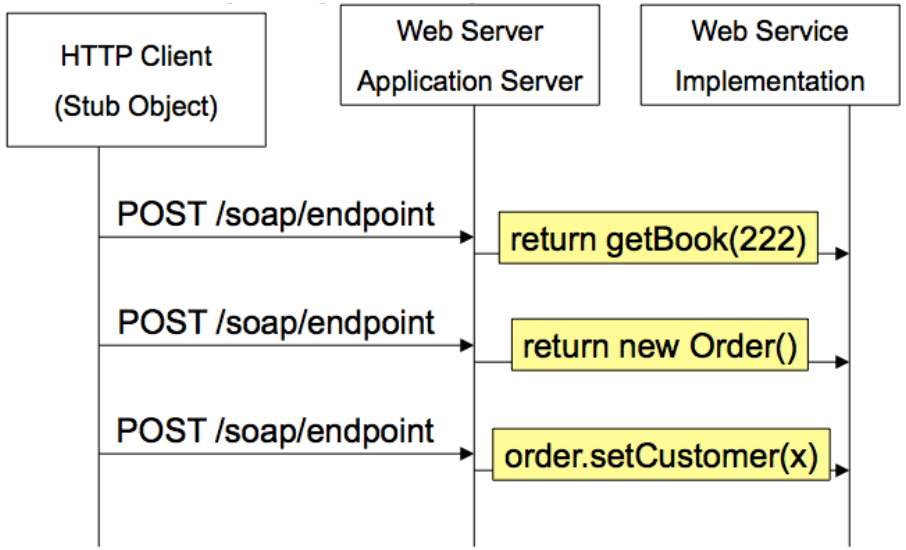
Web Services with SOAP



- HTTP is simply a transport layer for WS-SOAP
- SOAP messages are tunneled through HTTP
- There is one URI, which identifies the service



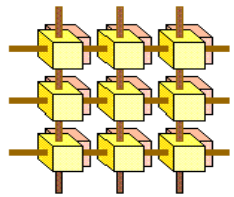
Web Services with SOAP (cont.)



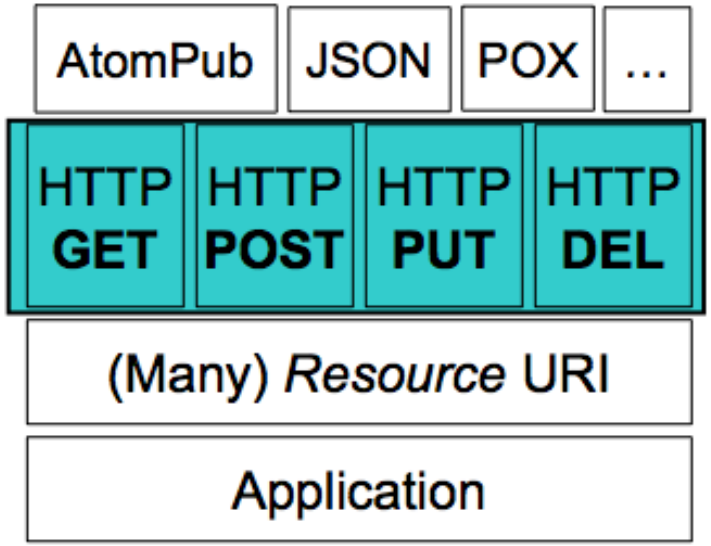
- All messages use HTTP posts and the unique service URI
- Service maintains state (“order” object maintained by service is created in one message exchange and operated on in subsequent message exchanges)
- WSDL interface description used to generate client stubs

Georgia Institute of Technology

Critical Systems Laboratory

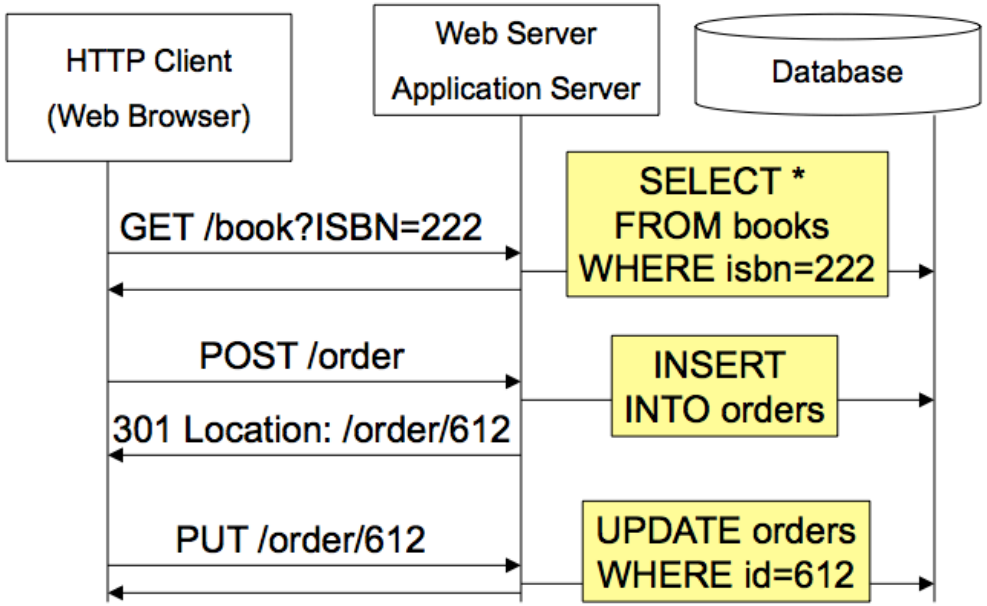


Web Services with REST

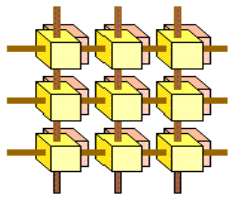


- HTTP is the application layer for WS-REST
- REST messages, for a given service, can operate on multiple resources identified by their respective URIs

Web Services with REST (cont.)



- Operations are carried out using different HTTP methods operating on resources with their own URIs
- Two resources: “books” and “orders”
- Server-side state pushed into resources, which can be accessed concurrently by different services

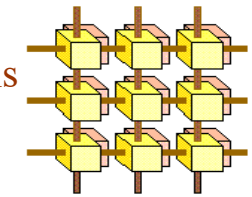


Web Services with REST (cont.)

- Communication is stateless: each client request to the server must contain all information needed to understand the request, without referring to any stored context on the server
- Application state is pushed to edges: clients and resources
- Client state can be maintained using cookies
- Server-side state pushed into resources, which can be accessed concurrently by different clients and different services

Georgia Institute of Technology

Critical Systems Laboratory

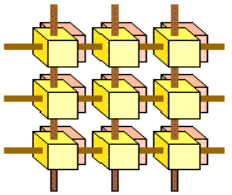


Web Services with REST: Principles

1. Identify *all* resources through URIs
2. Uniform and simple interface: HTTP get, head, put, post, delete
    - 1. and 2. ⇒ “small set of verbs applied to a large set of nouns”
3. Self-describing messages
4. Hypermedia driving application state: applications “navigate” interconnected set of resources
5. Stateless interactions

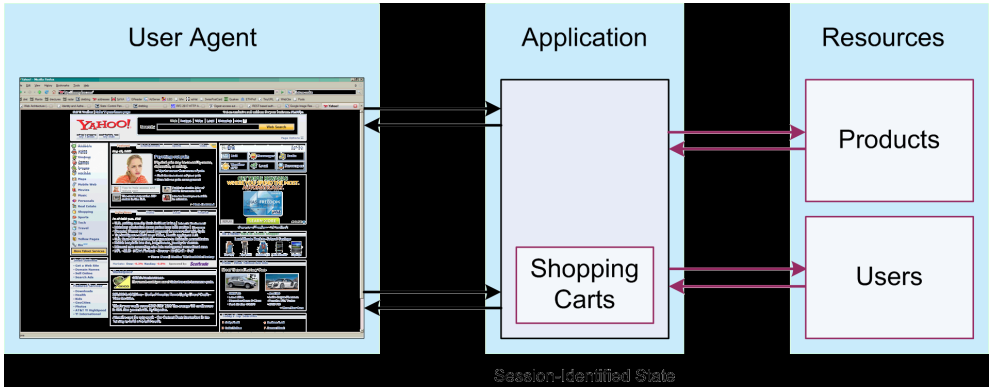
Georgia Institute of Technology

Critical Systems Laboratory

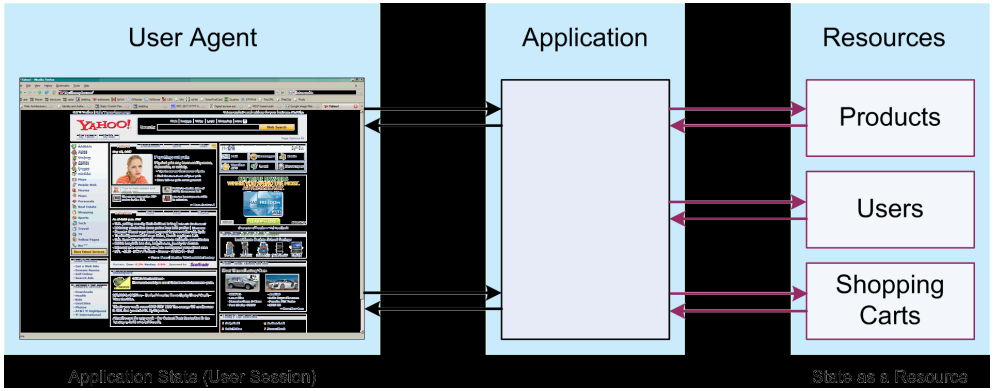


SOAP vs. Rest: State Handling

SOAP: Shopping cart is state maintained by service, available only to clients of that service that know how to access it

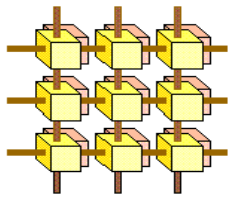


REST: Shopping cart is resource stored persistently on server, accessible via its URI to any client and any service



Georgia Institute of Technology

Critical Systems Laboratory

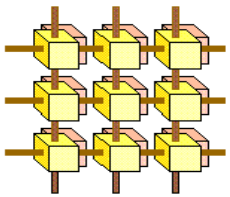


Big Data

- Data collection too large to transmit economically over Internet --- Petabyte data collections
- Computation produces small data output containing a high density of information
- Implemented in the cloud
  - data generated in the cloud
  - bring computation to data, too expensive to bring data to computation (think Google Trends operating on Google search data)
- Easy to write programs, fast turn around
- Often processed with MapReduce paradigm
  - Map(k1, v1) -> list (k2, v2)
  - Reduce(k2,list(v2)) -> list(v3)

Georgia Institute of Technology

Critical Systems Laboratory

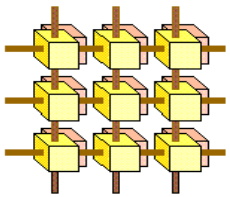


What is MapReduce?

- MapReduce
  - Programming model from LISP
  - (and other functional languages)
- Many problems can be phrased this way
- Easy to distribute across nodes
  - Imagine 10,000 machines ready to help you compute anything you could cast as a MapReduce problem!
    - This is the abstraction Google is famous for authoring
  - It hides LOTS of difficulty of writing parallel code!
  - The system takes care of load balancing, dead machines, etc.
- Nice retry/failure semantics

Georgia Institute of Technology

Critical Systems Laboratory



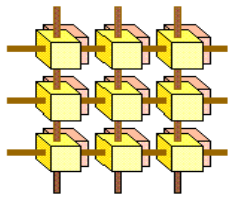
Programming Concept

- Map
  - **Perform** a function on **individual values** in a data set to create a **new list** of values
  - Example: square x = x * x
    map square [1,2,3,4,5]
    returns [1,4,9,16,25]

- Reduce
  - **Combine** values in a data set to create a new **value**
  - Example: sum = (each elem in arr, total +=)
    reduce sum [1,2,3,4,5]
    returns 15 (the sum of the elements)

Georgia Institute of Technology

Critical Systems Laboratory



MapReduce Programming Model

Input & Output: each a set of key/value pairs

Programmer specifies two functions:

```
map (in_key, in_value) →
       list(out_key, intermediate_value)
```

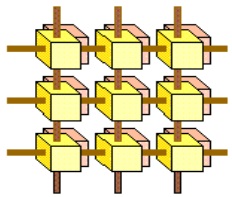
- Processes input key/value pair
- Produces list of intermediate pairs

```
reduce (out_key, list(intermediate_value)) →
          list(out_value)
```

- Combines all intermediate values for a particular key
- Produces list of merged output values (often just one)

Georgia Institute of Technology

Critical Systems Laboratory

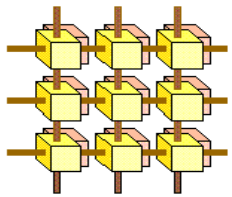


Word Count Example

- We have a large file of words, many words in each line
- Count the number of times each distinct word appears in the file(s)

Georgia Institute of Technology

Critical Systems Laboratory



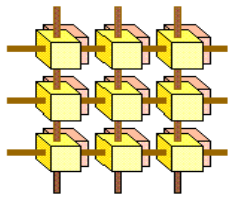
Word Count using MapReduce

```
map(key = line, value=contents):
    for each word w in value:
        emit Intermediate(w, 1)

reduce(key, values):
// key: a word; values: an iterator over counts
        result = 0
        for each (key, v) in intermediate values:
            result += v
        emit(key,result)
```

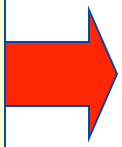
Georgia Institute of Technology

Critical Systems Laboratory

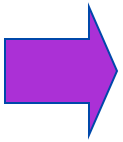


Word Count, Illustrated

see bob run
see spot throw



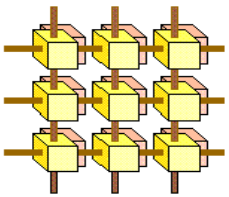
see	1
bob	1
run	1
see	1
spot	1
throw	1



bob	1
run	1
see	2
spot	1
throw	1

Georgia Institute of Technology

Critical Systems Laboratory

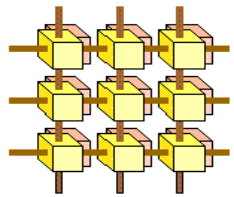


MapReduce WordCount Java Code

```
public static void main(String[] args) throws IOException {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");
    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);
    conf.setMapperClass(WCMap.class);
    conf.setCombinerClass(WCReduce.class);
    conf.setReducerClass(WCReduce.class);
    conf.setInputPath(new Path(args[0]));
    conf.setOutputPath(new Path(args[1]));
    JobClient.runJob(conf);
}

public class WCMap extends MapReduceBase implements Mapper {
    private static final IntWritable ONE = new IntWritable(1);
    public void map(WritableComparable key, Writable value,
                    OutputCollector output,
                    Reporter reporter) throws IOException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
            output.collect(new Text(itr.next()), ONE);
        }
    }
}

public class WCReduce extends MapReduceBase implements Reducer {
    public void reduce(WritableComparable key, Iterator values,
                       OutputCollector output,
                       Reporter reporter) throws IOException {
        int sum = 0;
        while (values.hasNext()) {
            sum += ((IntWritable) values.next()).get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
```



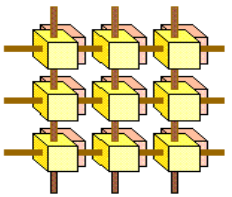
Google PageRank using MapReduce

- Program implemented by Google to rank any type of recursive "documents" using MapReduce.
- Initially developed at Stanford University by Google founders, Larry Page and Sergey Brin, in 1995.
- Led to a functional prototype named Google in 1998.
- Still provides the basis for all of Google's web search tools.
- PageRank value for a page *u* is dependent on the PageRank values for each page *v* out of the set B_u (all pages linking to page *u*), divided by the number *L(v)* of links from page *v*

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

Georgia Institute of Technology

Critical Systems Laboratory

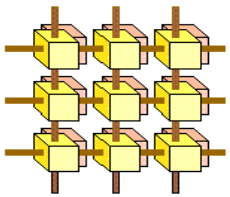


PageRank: Propagation

- Calculates outgoing page rank contribution for a page
- Map: for each object
  - If object is vertex, emit key=URL, value=object
  - If object is edge, emit key=source URL, value=object
- Reduce: (input is a web page and all the outgoing links)
  - Find the number of edge objects→outgoing links
  - Read the PageRank Value from the vertex object
  - Assign PR(edges)=PR(vertex)/num_outgoing

Georgia Institute of Technology

Critical Systems Laboratory

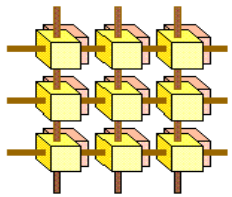


PageRank: Aggregation

- Calculates rank of a page based on incoming link contributions
- Map: for each object
  - If object is vertex, emit key=URL, value=object
  - If object is edge, emit key=Destination URL, value=object
- Reduce: (input is a web page and all the incoming links)
  - Add the PR value of all incoming links
  - Assign PR(vertex)=$\sum$PR(incoming links)
- Repeatedly execute propagation, aggregation phases until convergence

Georgia Institute of Technology

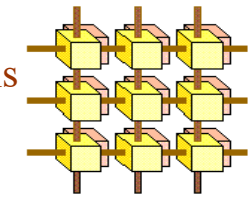
Critical Systems Laboratory



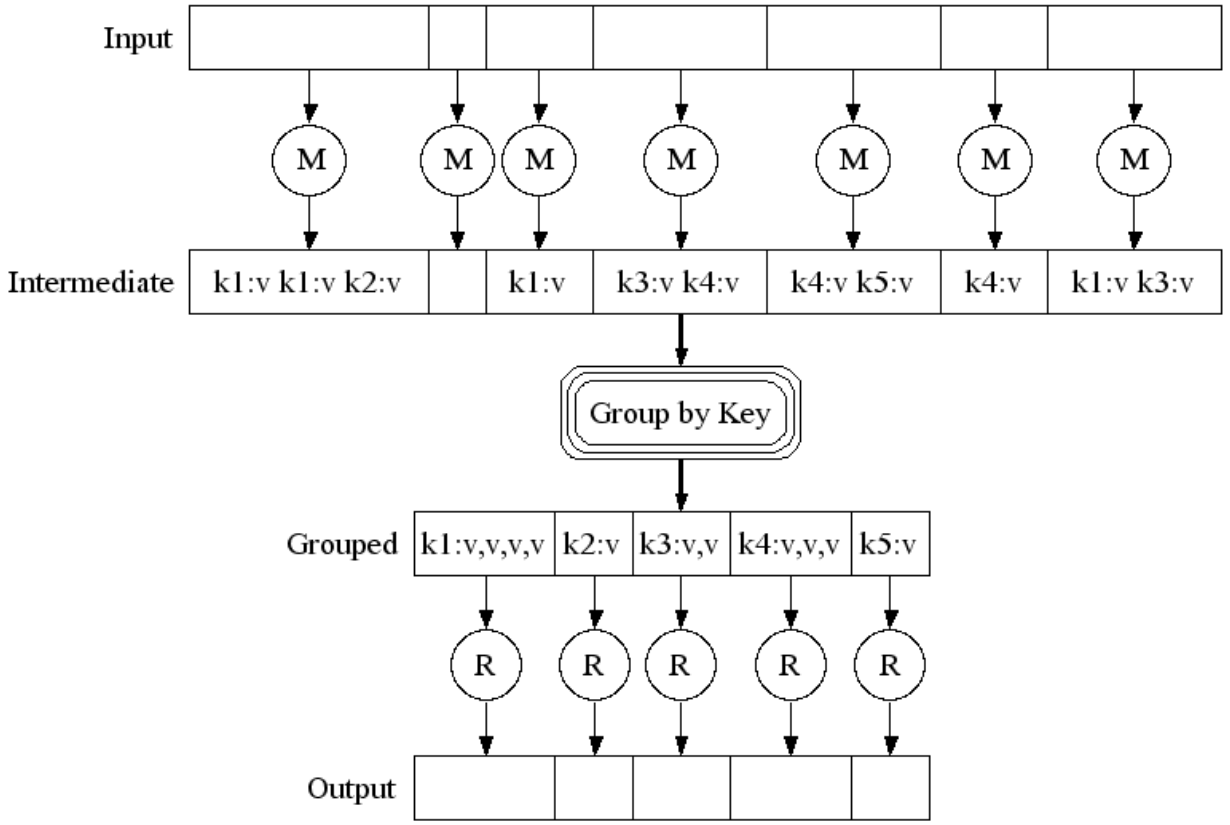
Hadoop Execution

- How is this distributed?
  1. Partition input key/value pairs into chunks, run map() tasks in parallel
  2. After all map()s are complete, consolidate all emitted values for each unique emitted key
  3. Now partition space of output map keys, and run reduce() in parallel

- If individual map() or reduce() fails, reexecute!

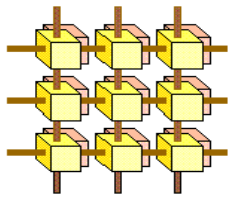


Hadoop Execution (cont.)



Georgia Institute of Technology

Critical Systems Laboratory

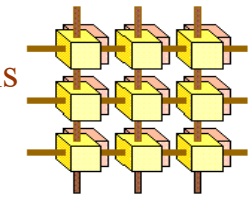


Hadoop Execution Coordination

- Split input file into 64MB sections (GFS)
  - Read in parallel by multiple machines
- Fork off program onto multiple machines
- One machine is Master
- Master assigns idle machines to either Map or Reduce tasks
- Master coordinates data communication between map and reduce machines

Georgia Institute of Technology

Critical Systems Laboratory

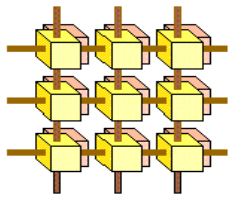


Beyond MapReduce – Data Processing Pipelines

- Google Cloud Dataflow
- Amazon Data Pipeline
- MapReduce pipelines data across the following steps:
  - Split
  - Map
  - Shuffle and Sort
  - Reduce
- General data processing pipeline allows programmer to define each step however they want and efficiently pipeline data across the steps

Georgia Institute of Technology

Critical Systems Laboratory

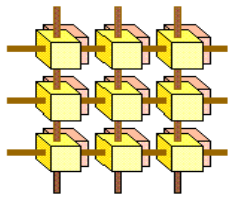


Beyond MapReduce – TfIdf Pipeline Example

- TfIdf – term frequency - inverse document frequency; importance of a term to each document in a corpus
- TfIdf pipeline steps (after splitting documents):
  - Map each document's URI to each word in document
  - Map each word to number of documents it appears in (nd)
  - Map each document's URI to total number of words in document
  - For each (word, URI), count number of occurrences of word in document with that URI
  - Merge total words and word counts, i.e. create a (wordCount, totalWordCount) pair for each (word, URI) pair
  - Compute term frequencies (wordCount/totalWordCount)
  - Compute document frequencies (nd/numberOfDocuments)
  - Compute TfIdf = termFreq * (ln 1/docFreq) for each (word, URI)

Georgia Institute of Technology

Critical Systems Laboratory



noSQL Data Services

- Most often refers to a “key-value store”
  - Data indexed by a single element, the key
  - All queries are based on the key
  - Good for large amounts of unstructured data
- Simpler and faster than fully relational database (e.g. SQL)
  - Relational databases are structured as tables
  - A complete row of the table is one record
  - Columns of the table represent different fields of the database
  - Queries can be run against any field or combination of fields
  - Good for moderate amounts of structured data