

# CS290f - Lecture 1

## Ready, Set, Go!

Scalable Internet Services, Fall 2006

Thorsten von Eicken  
Department of Computer Science  
University of California at Santa Barbara

## Course Goal

- **Scalable Internet Services**
  - How does one think about building these?
  - What are the components involved, how do they work?
  - What have others done? Read some papers.
- **Learn by doing, not listening**
  - Roll-up your sleeves and get working!
  - 1 project in 4 parts, lots of challenges, lots of work
  - Lots to learn and use: Ruby, Rails, Apache, MySQL, Web Services, Load balancing, Parallelism, Fault tolerance, and more
  - ... the lectures are 30% how-to, 70% background

2

## For today...

- Course Intro – “Scalable Internet Services”
- Web services – for project 1
- Ruby on Rails – framework for this course
- Course projects – project 1 handout

3

## Thorsten von Eicken's Background

- **Ph.D on high-performance communication in parallel computers from UC Berkeley**
- **6 years prof in CS at Cornell University**
  - high-performance communication in clusters
  - security in Java VMs for web servers
- **7 years Chief Architect at Expertcity.com / Citrix Online**
  - responsible for overall system architecture and web operations
- **Looking for the next opportunity since May 2006**
  - ... and learning Ruby on Rails
- **Visiting professor at UCSB**
  - taught “same” course in spring 2000 & winter 2001.

4

## Define "Scalable Internet Services"

### ■ "Internet Services"

- Web sites
- Web services

### ■ Web sites – typically:

- Serve web pages in HTML
- Used/viewed by humans
- Relational database backed (SQL)
  - ◆ Alternatively
    - ? Static files
    - ? Application-specific database
    - ? Live data, e.g. webcam
    - ? Live communication, e.g. chat, screen sharing, talk

5

## Define "Internet Services"

### ■ Web services

- Provide programmatic access to services / resources
- Typically RPC-style of interaction over HTTP
- Using XML to encode complex data structures
- Variety of protocols
  - ◆ REST, XML-RPC, SOAP, WSDL, WS...
- Examples:
  - ◆ Your favorite Google, Yahoo, Amazon, Flickr functionality

6

## Characteristics

### ■ A way to deliver applications and data to a large and mobile audience

- In contrast to selling software
- In contrast to selling data on CDs

*Service*

### ■ Have the potential to grow to a huge worldwide user base

- Scalability is always a consideration
- Authentication and authorization are always in play
- 24/7 operation and availability are concerns
- Contrast to locally installed single-user applications...

*Scale*

### ■ Enable communication among users

- New concepts of community, sharing, etc.

### ■ Can be updated at any time

- Eternal "beta", weekly code upgrades
- Data feeds

*Agile*

7

## Define "scalable"

- From "Building Scalable Web Sites" by Cal Henderson, chapter 9



### ■ A scalable system has three characteristics:

- The system can accommodate increased usage
- The system can accommodate an increased dataset
- The system is maintainable

### ■ Scalability is one of the most abused terms by marketing

- Don't confuse scalability and performance!
- (We'll come back to this)

8

## Demand, infrastructure, service levels

- **“As demand grows, the system can be expanded to maintain service levels”**
- **Demand**
  - Number of users
  - Amount of data
  - (frequency of use, geographic diversity, ...)
- **System**
  - Servers (cpu, memory, ...)
  - Storage (disks, storage systems, ...)
  - Bandwidth (routers, network connections, ...)
  - People (operators, programmers, sysadmins, ...)
- **Service levels**
  - Response time / download speed
  - Availability
  - Functionality

9

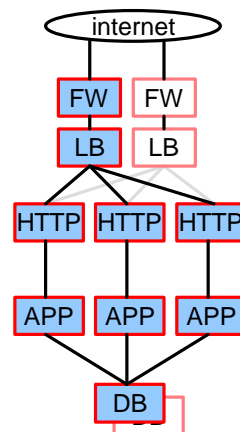
## “Scalable”

- **If users and data increase 10x can the servers, storage and bandwidth be increased to maintain response time?**
- **Second order questions:**
  - At what cost?
  - What is the demand -> cost function? [ cost = f(demand) ]
  - What is the headroom, what are limits?
    - ◆ E.g. what about another 10x increase, and another, and another?
- **Do not confuse with performance!**
  - With X+Y+Z as infrastructure, we can handle N users, M data, and provide response time R
  - Of course, good performance makes scalability easy

10

## How to build scalable web services in one slide (it's that simple)

- **Simple, stupid, share-nothing web stack**
  - Network devices – routers, firewall, load balancer, DNS, global LB, fail-over, ...
  - HTTP server – thread/process model, HTTP protocol, caching, encryption (SSL)
  - Single threaded app server – model-view-controller programming model, RoR, concurrency, atomicity, clustering, caching, threading
  - Relational database – SQL, replication, redundancy, disaster recovery, partitioning



11

## Isn't this obvious?

- **Thought experiment**
  - Your web site runs on an old P3 box and is becoming overloaded, what do you do?

12

## Other techniques

### ■ Multithreaded clustered app servers w/ coherent caches

- Typical of Java J2EE implementations
- More efficient at the small scale (one server)
- Enables more caching of data in the app servers
- More flexible in terms of system architecture: don't need to partition into multiple services as much
- Complex to program and fragile to operate, troubleshooting can be hell

13

## Other techniques

### ■ Distributed computing services

- Use distributed computing theory to implement fundamentally distributed (and thus scalable?) services
- Highly complex to build, operate, and use
- Typically limited functionality

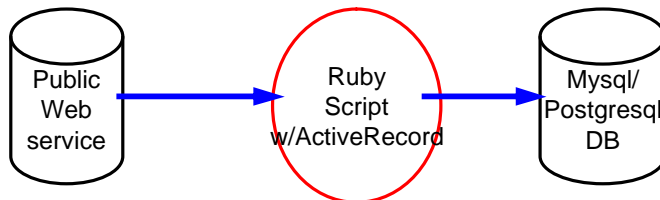
14

## Project1 preview

### ■ Goals

- Get started with Ruby
- Get familiar with web services
- Collect data for the project web site

### ■ Problem statement



15

## Robots & Crawlers

### ■ Definition attempt:

- Crawler: automated program that fetches and processes (a significant portion) of a web site's document hierarchy
- Robot: automated program that fetches and processes specific pages of a web site

### ■ Uses:

- Search engines
- Comparison shopping
- Performance measurements
- Automated testing
- Server monitoring
- Others?

16

## Manual fetch: telnet

```
[bugatti ~] telnet www.yahoo.com 80
Trying 204.71.200.67...
Connected to www.yahoo.akadns.net.
Escape character is '^]'.
GET / http/1.0
HTTP/1.0 200 OK
Content-Length: 18865
Content-Type: text/html
<html><head><title>Yahoo!</title><base
href=http://www.yahoo.com/><meta http-equiv="PICS-Label"
content="(PICS-1.1 "http://www.rsac.org/ratingsv01.html" 1
gen true for "http://www.yahoo.com" r (n 0 s 0 v 0 l
0))"></head><body onLoad="document.f.p.focus();">
<script><!--
...
<a href=r/ao>Advertising</a><p>Copyright &copy; 2001 Yahoo!
Inc. All rights reserved.</small><br><a href=r/pv>Privacy
Policy</a></form></center></body></html>
Connection closed by foreign host.
[bugatti ~]
```

17

## Manual fetch: wget

```
[bugatti ~] wget -O- http://www.yahoo.com/
--20:14:46-- http://www.yahoo.com:80/
=> '-'
Connecting to www.yahoo.com:80... connected!
HTTP request sent, awaiting response... 200 OK
Length: 16,897 [text/html]

OK -><html><head><title>Yahoo!</title><base
href=http://www.yahoo.com/><meta http-equiv="PICS-Label"
content="(PICS-1.1 "http://www.rsac.org/ratingsv01.html" 1
gen true for "http://www.yahoo.com" r (n 0 s 0 v 0 l
0))"></head><body onLoad="document.f.p.focus();">
<script><!--
...
<a href=r/hr>Jobs</a> -
<a href=r/ao>Advertising</a><p>Copyright &copy; 2001 Yahoo!
Inc. All rights reserved.</small><br><a href=r/pv>Privacy
Policy</a></form></center></body></html>
..... [100%]

20:14:46 (211.55 KB/s) - '-' saved [16897/16897]
[bugatti ~] wget -O- http://www.yahoo.com/ |& less
```

18

## Robot/crawler limitations

- Unlinked pages (plain text links)
- Authentication, cookies
- Parameters/dynamic pages
- Browser/host type
- Document types
- Others?

19

## Web services

- The W3C defines a Web service as a software system designed to support interoperable machine-to-machine interaction over a network
- Usually stands for a service with read and write capabilities exposed over HTTP and using an XML data encoding
- Common Web Services Protocols:
  - REST, XML-RPC, SOAP, WSDL, (RSS, Atom)

20

## Why Public Web Services?

### ■ Theory (and hype)

- Composition of services (interoperability)
- Selling new forms of services

### ■ Practicalities

- Robots/crawlers navigate the site anyway
- People reverse engineer URLs and issue them programmatically anyway
- For internal service decomposition most of these APIs exist anyway
  - ◆ And everyone knows how to provide and use an HTTP service

21

## Course Project

### ■ Goal

- gain hands-on experience in building and deploying a scalable web service on the internet

### ■ Project 1

- retrieve data from a web service (Google maps, Amazon books, Yahoo local, Flickr, ...)
- you will be working with this data in projects 2 through 4
- use Ruby + ActiveRecord + REST/SOAP plug-ins + mysql or postgresSQL

### ■ Project 2

- build a very, very simple web site around the data to display the items collected
- use Ruby + Rails + possibly host your site on Amazon's Elastic Compute Cloud

### ■ Project 3

- build a transactional web site around the data
- implement users accounts, authentication, keep track of (http-) session data, perform some form of atomic transactions

### ■ Project 4

- scale the application to multiple front-end servers
- load test it for many users and lots of data.
- Session data will need to be distributed, transactions will need to remain atomic.

22

## Defining your project

- All projects must be executed in **teams of two**. (Remember  $2 \neq 3$  and  $2 \neq 1$ .)
- Users can **modify data** on the site (e.g. edit, bid on, or purchase)
- users must have a **userid and password**, as well as some **per-user state** (account, preferences, etc.)
- there must be some **http session data** that follows each user around as he/she is browsing the site
- there must be some **hard atomic transactions**, such as buying the last copy of a book, winning an auction, or acquiring an edit lock on a wiki page
- there must be a "pile of data" that you can **acquire off the web**
- you must **dynamically incorporate data** from a web service (typically a 3rd party service), such as Google Maps, Yahoo! Local, Amazon Books, etc.
- written in Ruby on Rails using a SQL database for storage (mysql or postgresQL)
- your site must run on a "production" host (or set of hosts) on the internet that we designate, presumably on Amazon EC2

23

## Project example

- a computer science bookstore
- acquire the catalog information off Amazon
- make-up the inventory quantities at random
- users can create an account, log in, browse books, view book details
- When browsing, you incorporate real-time data from Amazon (e.g. ratings, rankings, or similar)
- users can add books to their shopping cart and check-out
- users can further view their account history/status

24