Developing and running big websites

Challenges tips and tricks

Jason Chambers

Atlanta Java Users Group – Java DevCon 2004

Introduction

- Advisory Developer - delta.com, flysong.com, skyteam.com
- 13 years experience
- Holds a B.Sc. (Hons) Computer Studies from Nottingham Trent University
- Interests include Java, Linux, web development
- http://jason.blog-city.com
- Sun Certified Programmer and Web Component Developer for the Java$^{\text{TM}}$ 2 platform
- Opinions expressed do not necessarily reflect those of Delta Air Lines, Inc. or Delta Technology Inc.

Contents

- Internet vs. Intranet applications
- Requirements for large web-sites
- HTTP state management
- Manageability
- Staying afloat
- Design, security, performance, capacity planning, QA
- Summary

Intranet vs. Internet

	Intranet app	Internet app
#Users	Predictable, limited	Unpredictable, unlimited
Browser	Corp standard	*
Network	Fast	Mostly dial-up
User locale	Limited	Anywhere
Availability	Office hours?	24/7/365
Customer facing	No	Yes
Brand component	No	Yes

High-level requirements

- Availability
- Serviceability
- Content management
- Capacity
- Security/Privacy
- Performance
- Scalability
- Agility
- Human factors
- Configuration management
- Manageability
- Reliability
- QA
- Reporting

Horizontal scaling

To meet these requirements - need > 1 server:

- Identical to each other – perform the same function
- How big? How many?
  - Fewer/large servers
    - Cheaper to maintain (patch, monitor etc.)
    - Faster to roll out application and/or content changes
    - Greater loss of available capacity in event of server failure
  - More/smaller servers
    - Server failure – hardly noticeable to capacity
    - Synchronizing content across the farm is challenging
    - More CPUs = costly for software with per CPU licensing model
- Spread load evenly across the servers
- > 1 server complicates HTTP state management

Vertical scaling

Bigger machines:

- More CPUs
- Faster CPUs
- More memory
- Faster disks
- When? E.g. when requests are computationally intensive

Clusters

To the outside world looks like a single machine - How?

- DNS round-robin (software)
  - DNS server cycles through 1:M mapping of name:IP
  - Cheap
  - No support for server affinity (stickiness)
  - What if server goes down?
  - What if new server added?
  - DNS changes take time to propagate across Internet
- Load-balancers (hardware)
  - DNS name maps to hardware load-balancer
  - Expensive
  - Solves problems with DNS round-robin

HTTP state management

HTTP is stateless

- No memory of prior connections and cannot distinguish one client’s request from that of another
- Strengths
  - Keeps protocol simple
  - Fewer resources consumed on server
  - Can support many simultaneous users – no client credentials and connections to maintain
- Weakness
  - Inability to track a single user as he/she traverses a web-site
  - HTTP suffers from amnesia – it doesn’t remember a thing

HTTP state management

What is a session?

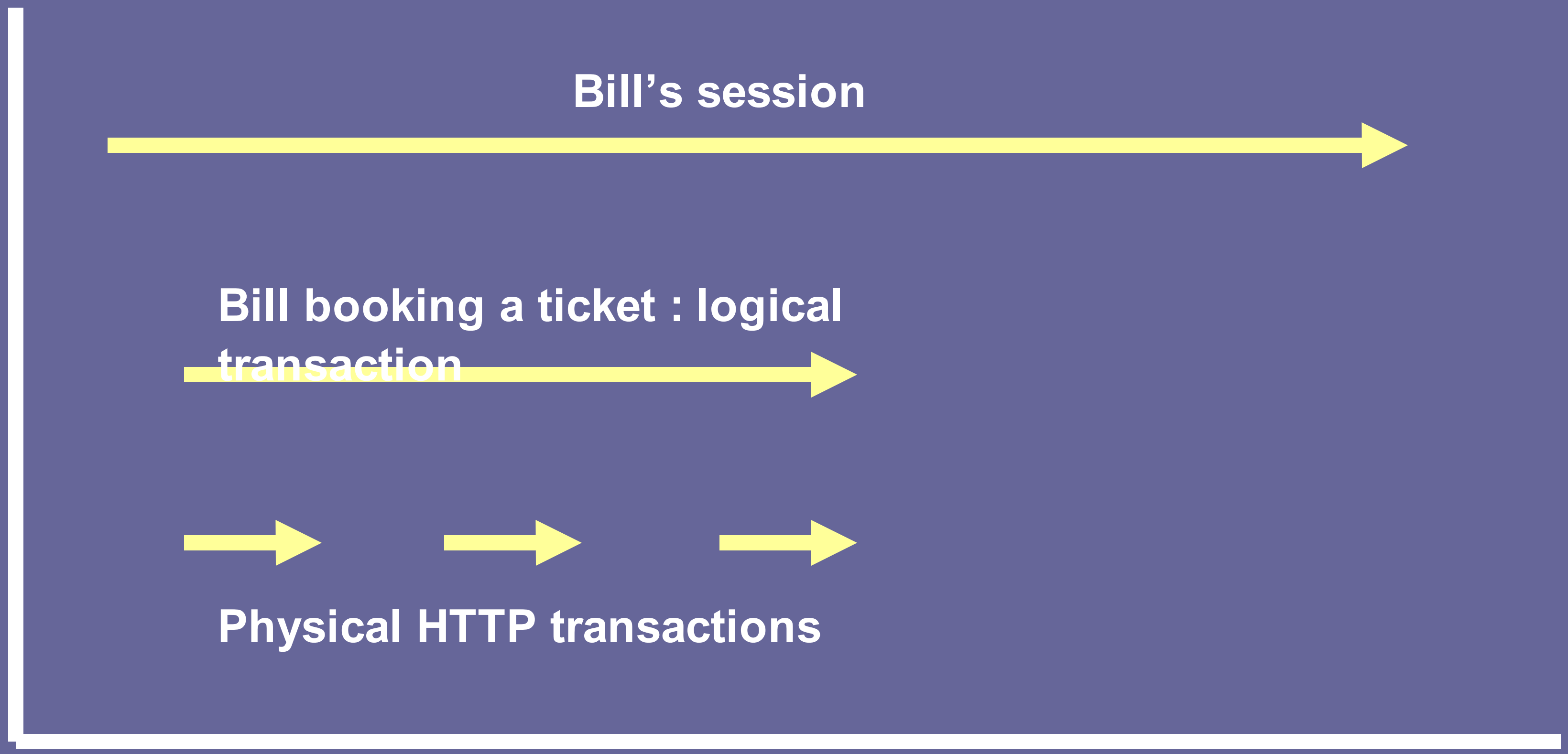
- Traditionally, a persistent connection between two hosts (client and server) that facilitates the exchange of information. Session is over when the connection is closed. E.g. an FTP session.

HTTP state management

What is a HTTP session?

- Physically – no such thing
- A virtual connections between the client and the server rather than a physical one
- Series of associated physical requests in which the client can be uniquely
- Association persists across multiple requests/connections for a specified period of time
- Accomplished by the server generating a unique token on first request (sessionID)
- Token passed back and forth between client and server

HTTP state management



Bill’s session

Bill booking a ticket : logical transaction

Physical HTTP transactions

HTTP state management

Where to store session state and transactional state?

- Many approaches to choose from
- Each have advantages/disadvantages
- Typical to use a combination of approaches

HTTP state management

What are the approaches?

- Inside
  - Web-tier: in memory - HttpSession.setAttribute()
    - Requires either server affinity “stickiness”
    - Or clustering solution (sharing of session state data)
  - Other
    - Transactional data store
    - RDB (less than optimal)
    - Specialized e.g. Chutney StateStore

HTTP state management

What are the approaches?

- Outside
  - Every technique has one trait in common – the state is shuttled back and forth with each subsequent client request
  - URL rewriting
  - Hidden fields (in forms)
  - Cookies (fine for session state – don’t use for transactional state)

Manageability

You cannot manage what you do not measure

Manageability

- Monitoring - automated
- Detection - automated
- Alert - automated
- Diagnose – manual/automated
- Corrective action – manual/automated
- Process
- Goal as always is zero impact for the customer and business
- Through trending – detect problems before your customer
- Sitescope, Tibco Hawk, JMX, Panacya,Tivoli
- Design for manageability

Staying afloat

Staying afloat

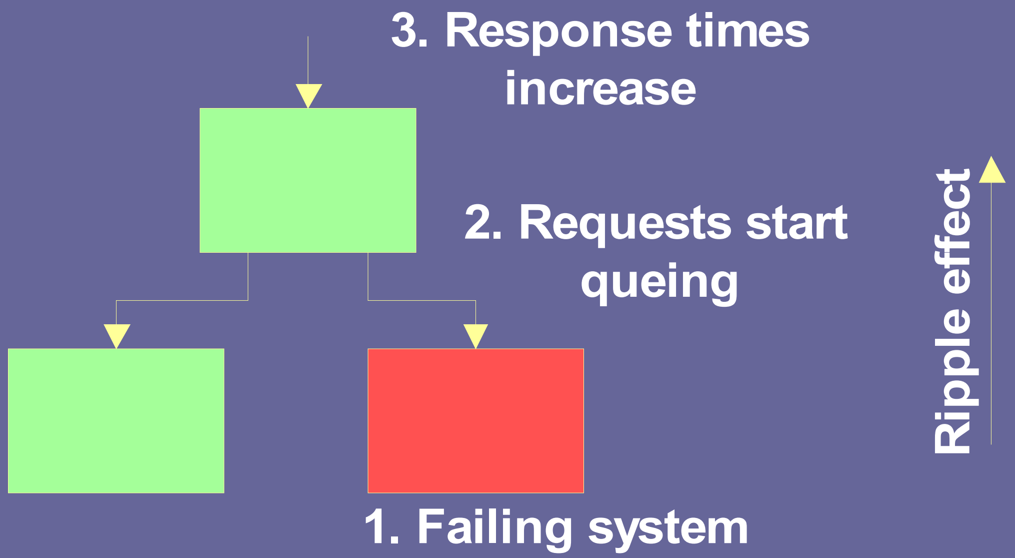
- Fast response times – always
- They may not always be a good response “Sorry, we are experiencing unprecedented load”
- Fail-fast, fail-fast, fail-fast
- Why?
- Slowdowns consume resources (threads, connections) for prolonged periods
- Other applications in the container starve
- Deck of cards
- Better to return empty handed

Staying afloat

What causes slow-downs?

- Usually, back-end systems
- A failing back-end system must be amputated – fast to preserve the health of the overall environment

Staying afloat



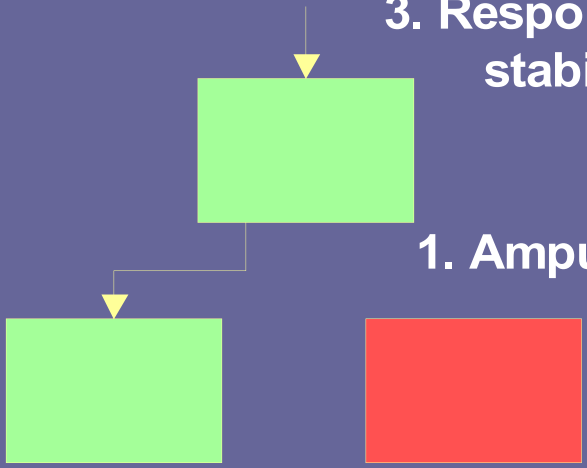
3. Response times increase

2. Requests start queing

1. Failing system

Ripple effect

Staying afloat

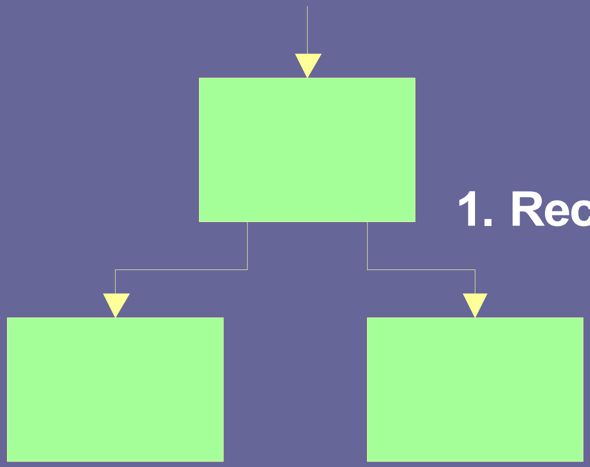


3. Response times stabilized

1. Amputate - close the valve

2. Recover

Staying afloat



1. Reconnect - open the valve

2. The patient is stabilized

Staying afloat

How to amputate fast?

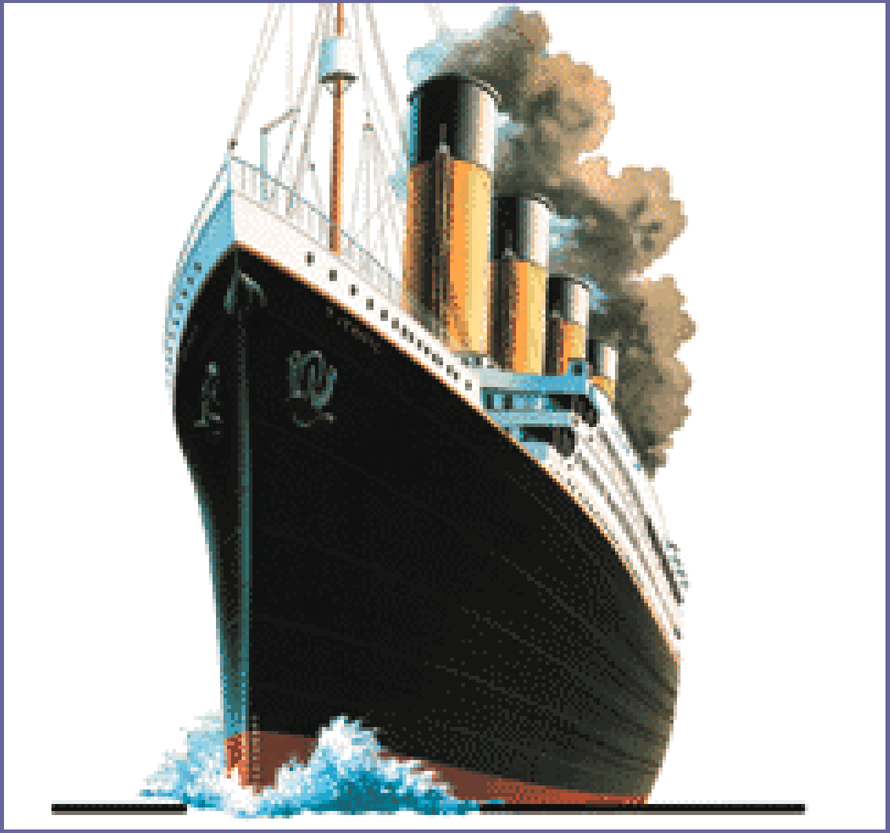
- No application changes required
- Design should cater for such an event
- Design a valve – if valve open, then attempt to connect
- If valve closed, then fail-fast
- Valve implemented as a JMX Mbean – controlled via management console

Staying afloat

Dealing with spikes

- Short bursts of activity that may exceed capacity
- Unpredictable – triggered by weather, world events
- Predictable – one day sales
- Shopping bots
- You’re only option is to throttle requests (remember fail-fast)
- IP block unscrupulous shopping bots

Infrastructure design



Hardware infrastructure design

- Horizontal partitioning
  - Separate web servers from application servers from database servers
  - Easier to scale
  - Easier to administer and monitor
- Vertical partitioning
  - Separate infrastructure for each major application group e.g. booking, flight information, SkyMiles account management
  - E.g. flight information could be down – but at least you can still sell tickets $$
- Multiple data centers

Software infrastructure design

- Avoid monolithic structures - continue with the isolation theme:
  - 1 WAR – consider breaking it into multiple WARs
  - 1 container instance (JVM) – consider breaking into multiple instances
- Size threads, db connection pools, heapsize etc. appropriately

Security

Application

- More than just HTTPS and a couple of firewalls
- https://dodgybank.com/go?custID=10 – “Hey Bill”
- Hack the URL
- https://dodgybank.com/go?custID=11 – “Hey Jason”
- … ouch
- https://dodgybank.com/go?custID=11&cksum=a0f9 - “Sorry you don’t appear to be Jason – goodbye”
- … better

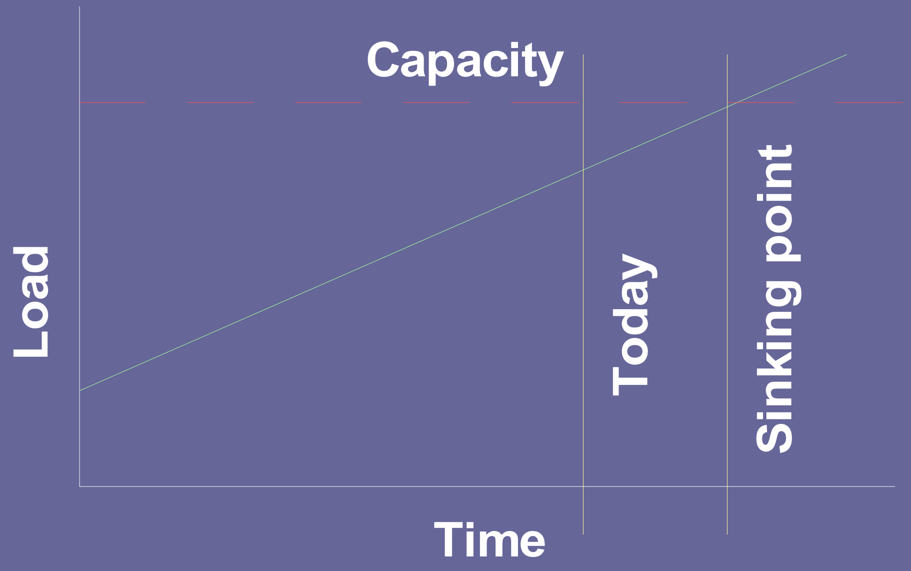
Network

- DMZ to protect your internal network from the nasty Internet

Performance

- First - Identify bottle-necks
- How fresh does the data really need to be?
- Cache everywhere – OS, Database, Application server, Application, Browser, Edge caching
- Database tuning – indexing, SQL etc.
- JVM tuning – heapsize etc.
- Appserver – connection pools, threads etc.

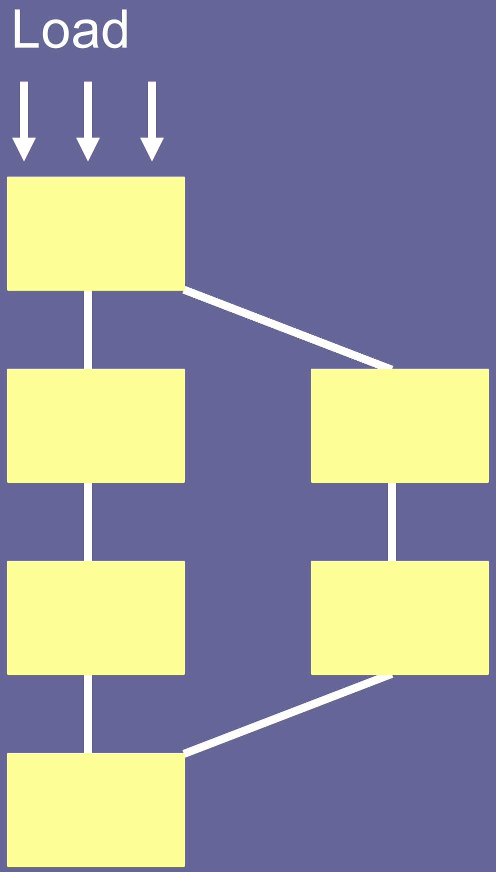
Capacity planning



QA - Load testing

- Delta.com is a distributed system
- Consists of connected components
- Components include web servers, firewalls, app servers, database servers..
- Load testing distributed systems is challenging
- How would you load test the Internet (the world’s biggest distributed system)?

QA - Load testing



- What are we testing?
- What does failure mean?
- What is the goal?
- Where are the bottle-necks – are they being masked?

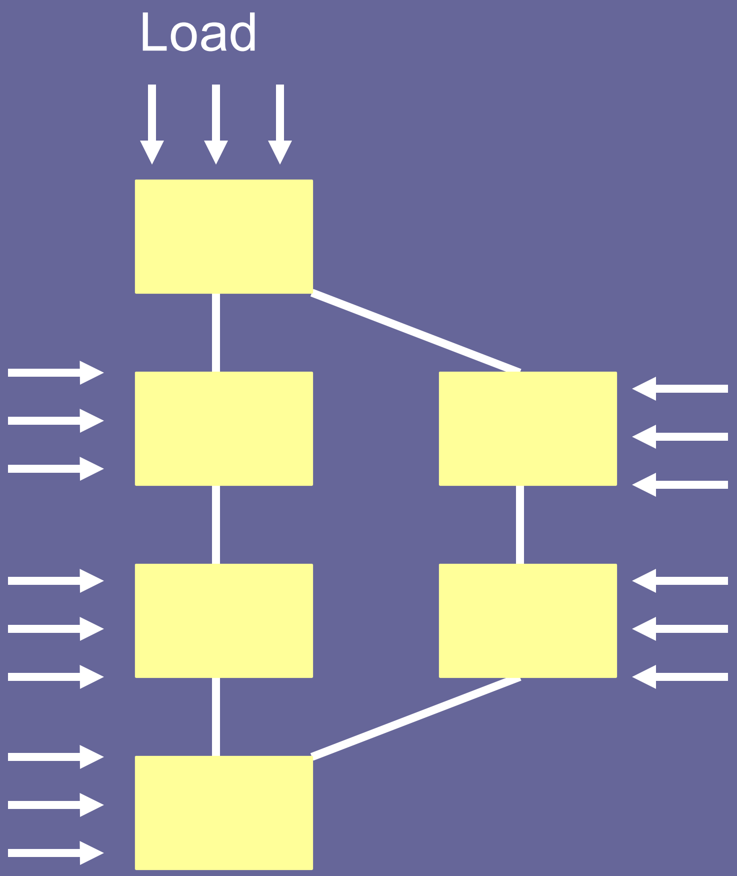
QA - Load testing

- In an ideal world:
  - Capacity(testenv) = Capacity(prodenv)
  - Isolated network
  - Test environment doubles up as DR
- In a not so ideal world:
  - Capacity(testenv) < Capacity(prodenv)
  - Is it scaled down by the same factor end to end?

QA - Load testing

TEST CELL

QA - Load testing



QA - Load testing

- Recommendations
  - Define goals
  - Isolation
  - Simple to complex strategy
  - Focus on how the application handles failure
  - Pay close attention to applications that introduce new moving parts

Summary

- Isolation
- Measure
- Redundancy (no single points of failure)
- Fail-fast
- Web site design is a symphony:
- Network design, infrastructure design, application design must be in tune

Wrap up

- Thanks for listening!
- jason_chambers@yahoo.com
- http://jason.blog-city.com