

## CS290f - Lecture 10 Web server replication

Scalable Internet Services, Fall 2006

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

*Some database material from J. Hellerstein @UCB*

## Goals and issues



### ■ Goal: multiple web servers

- Scale performance, ideally linearly in #servers
- Increase availability, ideally N-1 or N/2 can fail
  - ◆ N/2: two power grids (UPS & circuit), two networks, two ...

### ■ Assumed solved:

- Load balancing
- Failure detection (for the purpose of load balancing)

### ■ Problems: coherence / single system image

## Broad categories of approaches

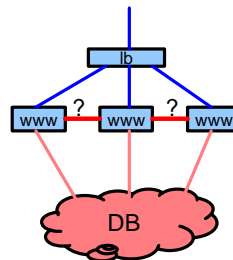
### ■ Shared nothing

- Really: shared database only

### ■ Partitioning

### ■ Active-standby (warm spare)

### ■ Cluster w/coherent object cache



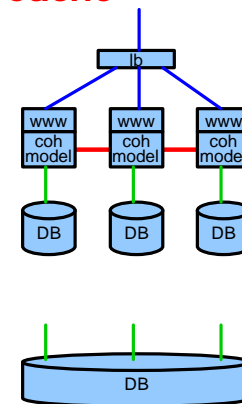
## Coherent Object Cache

### ■ Idea

- Consistent object cache
- Access model/object anywhere in a globally coherent manner

### ■ Issues

- Performance (communication)
  - ◆ Locality at odds with hiding location/communication
- ACID (database properties)
  - ◆ Durability: multiple copies
- Queries
  - ◆ Query language & algorithms
- Database updates
- Overall complexity



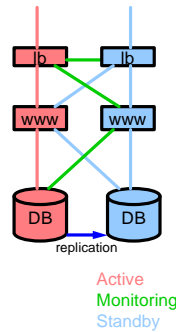
## Active-standby

### Idea

- Run on a single system
- Have a hot or warm spare
- Switch when active fails

### Setup

- Load balancers
  - ◆ Health-check each other
- Web servers
  - ◆ Lease kept in database
  - ◆ 1-row table: lease owner & lease timeout
  - ◆ E.g. 2 minute lease, renewed every 15 seconds
  - ◆ Standby web server returns 500 error to load balancer's checks
- Database
  - ◆ Automatic replication
  - ◆ Manual fail-over



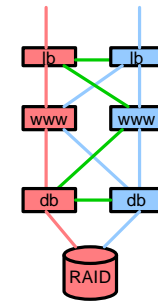
## Active-standby II

### Issues

- Standby may fail when activated
  - ◆ Why would it?
- Fail-over tends to lose some state
  - ◆ Conflict between fast/slow timeouts
- Lack of database fail-over
- Does not help scalability
- 50% overhead

### Database fail-over

- Single disk array, shared between two DB
- One DB has access to disks at any time
- Disk array may provide locking capability



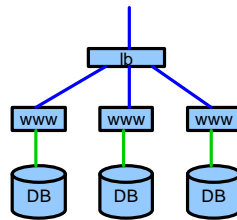
## Partitioning

### Idea

- Partition work to achieve no sharing
- Repartition upon failure, or fire-up warm spare

### Partition by ...

- Easy if users (or accounts) are independent
  - ◆ E.g. Gmail accounts, Amazon accounts & shopping carts, ...
- Partition by function
  - ◆ E.g. mail, calendar, search, photos, blog, accounting, ...
  - ◆ Tends to require tiered architecture with shared back-end functions, e.g., authentication, accounting, ...
  - ◆ Use active/standby within function



## Partitioning II

### Side-benefit: upgrades

- One partition can be in beta for the next version
- Gradually update one partition after the other
  - ◆ Reduces question/complaint volume
  - ◆ Roll-back or move problem customers back

### Side benefit: smaller systems

- 10x 6-disk databases vs. 1x 60-disk database
  - ◆ 10x 3U dual-processor, 4GB memory, 6 disks
  - ◆ 1x 4U eight-processor, 32GB memory, fibrechannel i/f, raid controller, 6x disk shelves
- 10x the administrative overhead?
  - ◆ Replication vs. big-iron hassle-factor
- Really 11x vs. 2x !

## Partitioning III

### ■ Issues

- What happens when *things* are no longer independent?
  - ◆ E.g. sharing Google calendars
- Overhead of managing partitions
  - ◆ Adding/removing partitions: repartition load?
  - ◆ Moving *things* between partitions
  - ◆ Doing everything N times
  - ◆ Configuring N+1 or circular fail-over
- Has many of the issues that active-standby has

9

## Database of record

### ■ Concept

- Subset of data truly requires ACID properties
- Store it in a database that keeps "record" of actions/events
- Keep all other data separately

### ■ Examples

- Order log, cc charge records
  - ◆ Shopping cart, CC info, package tracking are "secondary"
- Phone call detail record, outgoing email queue

### ■ Idea

- Keep DB of record small (scalability, replication)
- Keep DB or record append-mostly
- Reduce scope of hard problem...
- Avoid polluting critical data with lots of ancillary data

10

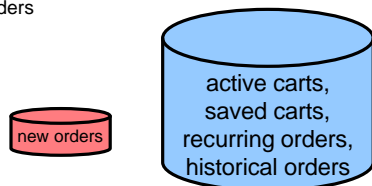
## Shopping cart example

### ■ DB of record

- Append-only order record at moment of check-out confirmation

### ■ Ancillary data

- Shopping cart at any prior stage ("order" in incomplete states)
  - ◆ It's easy to create "order" object and add "state" field...
- Wish lists, "save for later", recurring orders, ...
  - ◆ All these are just lists of items with an owner: "incomplete orders"
- Fulfilled/historical orders

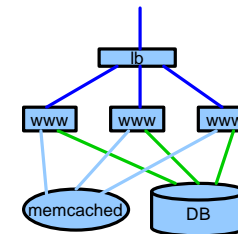


11

## Shared database approach

### ■ Build system so servers are independent of one another

- No caching of data!
- Basic operation:
  - ◆ Fetch data from database
  - ◆ Operate on data
  - ◆ Store results back into database
  - ◆ Use database to create atomicity



### ■ Push sharing into other components

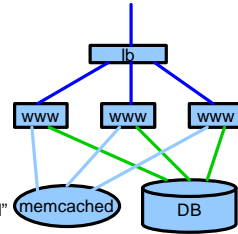
- Where replication is already solved
- Where replication is trivial (?)
- Where reliability is very high (due to simplicity or code maturity)

12

## Shared database approach II

### Observations

- Web servers are the most volatile
  - ◆ Bug fixes, new features, upgrades, ...
  - ◆ Simplicity => fewer bugs
  - ◆ Simplicity => development speed (development speed => \$\$\$)
- All web servers rely on database server
  - ◆ Database scaling and replication is "solved"
  - ◆ But it's very complex and expensive
  - ◆ Although a lot consists of "just" adding disks and memory!



### Issues

- How does "communication" through the database work?
  - ◆ Transactions & locks
- Isn't this too slow?

13

## "Agile development" and scaling

### Simplicity vs. performance tradeoff

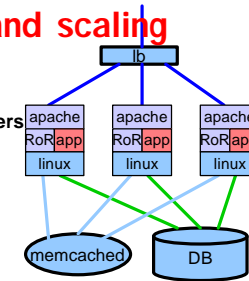
#### Assumption: critical path => programmers

- Simple => fast to program => time to \$\$\$
- Simple => easy to test => fewer bugs

- Simple => no caching => slow
- Simple => fewer optimizations => slow

#### Assumption: hardware is cheap => buy many!

- 1 programmer-year > 100 Amazon machine years!
  - ◆ (\$100k vs. \$876)
- A "quick" 1-day optimization ~ 1/2 Amazon machine year!
  - ◆ (\$100k / 245 work days ~ \$876 / 2)
- Programmers vs. machines => machines win every time!
  - ◆ **Unless adding machines is anything but trivial**
  - ◆ **Hence the importance of system architecture**



14

## Sometimes it doesn't work...

### Scale vs scale

- 1000 machines > 100x 10 machines
- 1000 machines vs. 10 machines => 10 programmer years

### Latency requirements

- Database too slow in latency

### Non-partitionable system

- ?

### ... ?

- Make sure you convince yourself first that the simple approach will not work!

15

## Toolbox

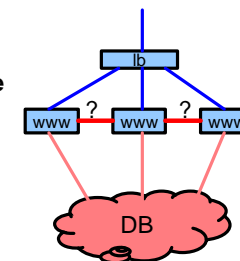
### No "right" or "best" approach

#### Use as appropriate and combine

- Shared nothing
- Partitioning
- Active-standby (warm spare)
- Cluster w/coherent object cache

#### Compare programmer time with machine cost

- Optimize the important one!



16