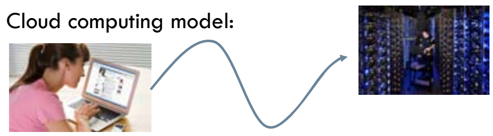
CLOUD COMPUTING

Lecture 27: CS 2110 Fall 2013

Computing has evolved...

- Fifteen years ago: desktop/laptop + clusters
- Then
  - Web sites
  - Social networking sites with photos, etc
  - Cloud computing systems
- Cloud computing model:



Styles of cloud computing

- Supporting Facebook or Google+ ("System as a service" or SaaS)
- Cornell's email and registrar system ("Platform as a service" or PaaS model)
- Rent a big pile of machines for a period of time like Netflix does ("Infrastructure as a service" – IaaS)

Main elements

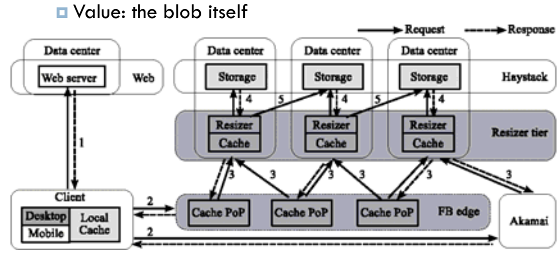
- Client computer (runs a web-enabled application, which could be in Java or could be a browser)
- The Internet (network links, routers, caching, etc)
- Massive back-end databases

Example: Facebook image "stack"

- Role is to serve images (photos, videos) for FB's hundreds of millions of active users
  - About 80B large binary objects ("blob") / day
  - FB has a huge number of big and small data centers
    - "Point of presense" or PoP: some FB owned equipment normally near the user
    - Akamai: A company FB contracts with that caches images
    - FB resizer service: caches but also resizes images
    - Haystack: inside data centers, has the actual pictures (a massive file system)

Facebook "architecture"

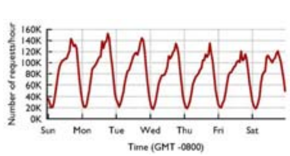
- Think of Facebook as a giant distributed HashMap
  - Key: photo URL (id, size, hints about where to find it...)
  - Value: the blob itself

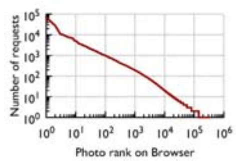


Facebook traffic for a week

7

- Client activity varies daily....





- ... and different photos have very different popularity statistics

Facebook's goals?

8

- Get those photos to you rapidly
- Do it cheaply
- Build an easily scalable infrastructure
  - With more users, just build more data centers
- ... they do this using ideas we've seen in cs2110!

Best ways to cache this data?

9

- Core idea: Build a *distributed photo cache* (like a HashMap, indexed by photo URL)
- Core issue: We could cache data at various places
  - On the client computer itself, near the browser
  - In the PoP
  - In the Resizer layer
  - In front of Haystack
- Where's the best place to cache images?
  - Answer depends on image popularity...

Distributed Hash Tables

10

- It is easy for a program on biscuit.cs.cornell.edu to send a message to a program on "jam.cs.cornell.edu"
  - Each program sets up a "network socket
  - Each machine has an IP address, you can look them up and programs can do that too via a simple Java utility
  - Pick a "port number" (this part is a bit of a hack)
  - Build the message (must be in binary format)
  - Java utils has a request

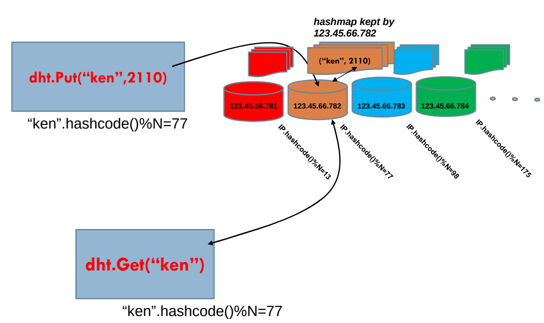
Distributed Hash Tables

11

- It is easy for a program on biscuit.cs.cornell.edu to send a message to a program on "jam.cs.cornell.edu"
- ... so, given a key and a value
  1. Hash the key
  2. Find the server that "owns" the hashed value
  3. Store the key,value pair in a "local" HashMap there
- To get a value, ask the right server to look up key

Distributed Hash Tables

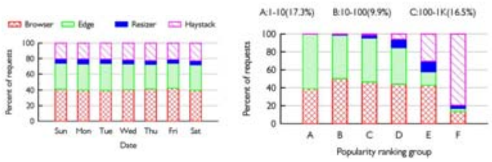
12



Facebook cache effectiveness

13

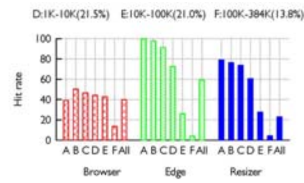
- Existing caches are very effective…
- … but different layers are more effective for images with different popularity ranks



Facebook cache effectiveness

14

- Each layer should "specialize" in different content.

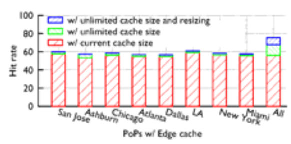


- Photo age strongly predicts effectiveness of caching

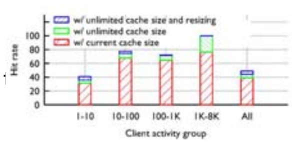
Hypothetical changes to caching?

15

- We looked at the idea of having Facebook caches collaborate at national scale…



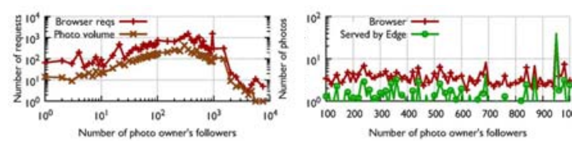
- … and also at how to vary caching based on "busyness" of the client



Social networking effect?

16

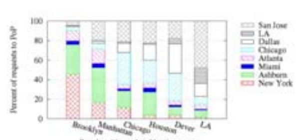
- Hypothesis: caching will work best for photos posted by famous people with zillions of followers
- Actual finding: *not really*



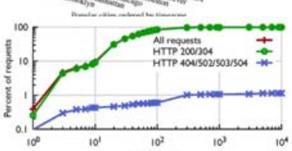
Locality?

17

- Hypothesis: FB probably serves photos from close to where you are sitting
- Finding: *Not really…*



- … *just the same, if the photo exists, it finds it quickly*



Can one conclude anything?

18

- Learning what patterns of access arise, and how effective it is to cache given kinds of data at various layers, we can customize cache strategies
- Each layer can look at an image and ask "should I keep a cached copy of this, or not?"
- Smart decisions ⇒ Facebook is more effective!

Strategy varies by layer

19

- Browser should cache less popular content but not bother to cache the very popular stuff
- Akamai/PoP layer should cache the most popular images, etc...
- We also discovered that some layers should "cooperatively" cache even over huge distances
  - Our study discovered that if this were done in the resizer layer, cache hit rates could rise 35%!

Overall picture in cloud computing

20

- Facebook example illustrates a style of working
  - Identify high-value problems that matter to the community because of the popularity of the service, the cost of operating it, the speed achieved, etc
  - Ask how best to solve those problems, ideally using experiments to gain insight
  - Then build better solutions
- Let's look at another example of this pattern

21

HIGH ASSURANCE DISTRIBUTED COMPUTING

Using Isis2: isis2.codeplex.com

High assurance cloud computing

22

- Ken's research on Isis^2 system
  - Today's cloud isn't very effective for supporting applications that need strong guarantees
  - Goal: create a cloud infrastructure that helps people build applications that can sensitive data/problems
- Target settings:
  - Smart electric power grid
  - Medical devices for ambulatory patients
  - Soldiers in on the front lines
  - Self-driving cars

Isis^2 System

- New C# library (but callable from any .NET language) offering replication techniques for cloud computing developers
- Intended to be as easy to use as a GUI framework
- Research challenges: many hard problems...

Elasticity (sudden scale changes)	Long scheduling delays, resource contention
Potentially heavily loads	Bursts of message loss
High node failure rates	Need for very rapid response times
Concurrent (multithreaded) apps	Community skeptical of "assurance properties"

Isis^2 makes developer's life easier

24

```
Group g = new Group("myGroup");
g.ViewHandlers += delegate(View v) {
    Console.Title = "myGroup members: "+v.members;
};
g.Handlers[UPDATE] += delegate(string s, double v) {
    Values[s] = v;
};
g.Handlers[LOOKUP] += delegate(string s) {
    g.Reply(Values[s]);
};
g.Join();

g.Send(UPDATE, "Harry", 20.75);

List<double> resultlist = new List<double>;
nr = g.Query(LOOKUP, ALL, "Harry", EOL, resultlist);
```

- First sets up group
- Join makes this entity a member. State transfer isn't shown
- Then can multicast, query. Runtime callbacks to the "delegates" as events arrive
- Easy to request security (g.SetSecure), persistence
- "Consistency" model dictates the ordering aseen for event upcalls and the assumptions user can make

Isis^2 makes developer's life easier

25

```
Group g = new Group("myGroup");
g.ViewHandlers += delegate(View v) {
    Console.Title = "myGroup members: "+v.members;
};
g.Handlers[UPDATE] += delegate(string s, double v) {
    Values[s] = v;
};
g.Handlers[LOOKUP] += delegate(string s) {
    g.Reply(Values[s]);
};
g.Join();

g.Send(UPDATE, "Harry", 20.75);

List<double> resultlist = new List<double>;
nr = g.Query(LOOKUP, ALL, "Harry", EOL, resultlist);
```

- **First sets up group**
- Join makes this entity a member. State transfer isn't shown
- Then can multicast, query. Runtime callbacks to the "delegates" as events arrive
- Easy to request security (g.SetSecure), persistence
- "Consistency" model dictates the ordering seen for event upcalls and the assumptions user can make

Isis^2 makes developer's life easier

26

```
Group g = new Group("myGroup");
g.ViewHandlers += delegate(View v) {
    Console.Title = "myGroup members: "+v.members;
};
g.Handlers[UPDATE] += delegate(string s, double v) {
    Values[s] = v;
};
g.Handlers[LOOKUP] += delegate(string s) {
    Reply(Values[s]);
};
g.Join();

g.Send(UPDATE, "Harry", 20.75);

List<double> resultlist = new List<double>;
nr = g.Query(LOOKUP, ALL, "Harry", EOL, resultlist);
```

- First sets up group
- **Join makes this entity a member. State transfer isn't shown**
- Then can multicast, query. Runtime callbacks to the "delegates" as events arrive
- Easy to request security (g.SetSecure), persistence
- "Consistency" model dictates the ordering seen for event upcalls and the assumptions user can make

Isis^2 makes developer's life easier

27

```
Group g = new Group("myGroup");
g.ViewHandlers += delegate(View v) {
    Console.Title = "myGroup members: "+v.members;
};
g.Handlers[UPDATE] += delegate(string s, double v) {
    Values[s] = v;
};
g.Handlers[LOOKUP] += delegate(string s) {
    Reply(Values[s]);
};
g.Join();

g.Send(UPDATE, "Harry", 20.75);

List<double> resultlist = new List<double>;
nr = g.Query(LOOKUP, ALL, "Harry", EOL, resultlist);
```

- First sets up group
- Join makes this entity a member. State transfer isn't shown
- **Then can multicast, query. Runtime callbacks to the "delegates" as events arrive**
- Easy to request security (g.SetSecure), persistence
- "Consistency" model dictates the ordering seen for event upcalls and the assumptions user can make

Isis^2 makes developer's life easier

28

```
Group g = new Group("myGroup");
g.ViewHandlers += delegate(View v) {
    Console.Title = "myGroup members: "+v.members;
};
g.Handlers[UPDATE] += delegate(string s, double v) {
    Values[s] = v;
};
g.Handlers[LOOKUP] += delegate(string s) {
    g.Reply(Values[s]);
};
g.Join();

g.Send(UPDATE, "Harry", 20.75);

List<double> resultlist = new List<double>;
nr = g.Query(LOOKUP, ALL, "Harry", EOL, resultlist);
```

- First sets up group
- Join makes this entity a member. State transfer isn't shown
- **Then can multicast, query. Runtime callbacks to the "delegates" as events arrive**
- Easy to request security (g.SetSecure), persistence
- "Consistency" model dictates the ordering seen for event upcalls and the assumptions user can make

Isis^2 makes developer's life easier

29

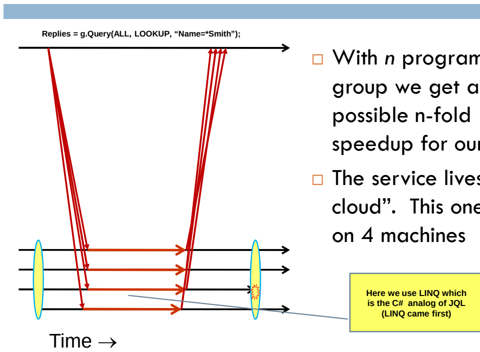
```
Group g = new Group("myGroup");
g.ViewHandlers += delegate(View v) {
    Console.Title = "myGroup members: "+v.members;
};
g.Handlers[UPDATE] += delegate(string s, double v) {
    Values[s] = v;
};
g.Handlers[LOOKUP] += delegate(string s) {
    g.Reply(Values[s]);
};
g.Join();

g.Send(UPDATE, "Harry", 20.75);

List<double> resultlist = new List<double>;
nr = g.Query(LOOKUP, ALL, "Harry", EOL, resultlist);
```

- First sets up group
- Join makes this entity a member. State transfer isn't shown
- Then can multicast, query. Runtime callbacks to the "delegates" as events arrive
- **Easy to request security (g.SetSecure), persistence**
- **"Consistency" model dictates the ordering seen for event upcalls and the assumptions user can make**

Example: Parallel search

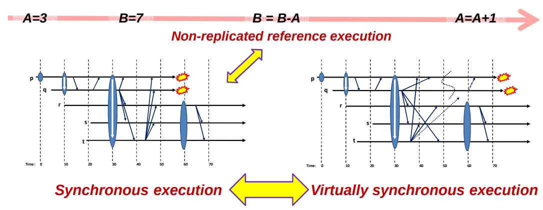


- With *n* programs in the group we get a possible n-fold speedup for our query
- The service lives "in the cloud". This one runs on 4 machines

Consitency model: How users "think" about Isis^2

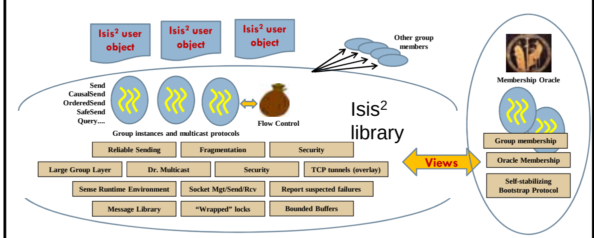
31

- **Virtual synchrony is a "consistency" model:**
  - ***Membership epochs:*** *begin when a new configuration is installed and reported by delivery of a new "view" and associated state*
  - ***Protocols run "during" a single epoch:*** *rather than overcome failure, we reconfigure when a failure occurs*



The system itself is a "community"

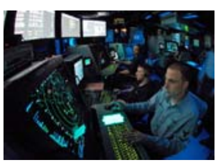
- Isis^2 is complex and concurrent… many interacting component parts that operate in concert



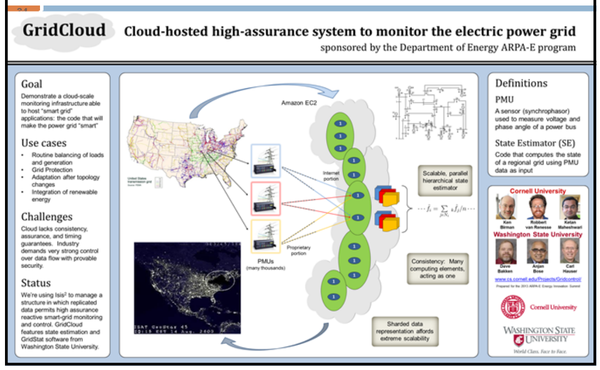
Use cases? The first Isis was used for…

33

- The New York Stock Exchange
- The French Air Traffic Control System
- The US Navy AEGIS warship

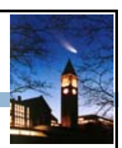


We're using Isis^2 in the "Smart Grid"



Summary

35



- The OO Java ideas we've learned matter!
  - The code people write at Facebook, or create using Isis^2, is very familiar looking
  - Not much harder than writing a GUI!
- Cornell has a great cloud computing group working on all sorts of cutting-edge questions