Distributed Systems

24/11/2014

Peer Systems - Node knows neighbour m but doesn't know destination z. prefix routing.

Google - web search: how does google achieve the indexing of the web?

Web crawlers: Web page has a link to a web page which has a link to another web page etc. forming a network. Assume all links are html for now(not pdf etc). Web crawlers start at a page (something important to them). Pull down content, don't care what it says, find all the links and pull down those pages, find all the links in those and pull them down and so on etc. Gradually, they build up a connected graph of these pages. Web crawlers also index these pages, take all words out page, ignore the obvious words (e.g. 'the'). Take all useful words and put them in a large database that links words to pages. Very large database mapping words to pages, 1 - n. Because there are only 10000-20000 words in the english language and far more webpages linking to these words.

E.g. if one word gives 20 million results, how do we find the one we're looking for? Google have to decide on ranking. First of all they build a 1-n mapping of words to urls, then they decide on a ranking, an ordering of all web pages they are indexing to determine how important they are. An algorithm called PageRank: a page is important if lots of other pages reference it. **What is page farming?**

How does google decide which page is more important? What problems do they have? Their policy for ranking is producer-centric. Google would have to deal with problem of content generated from page farming.

javaspace type model. vs. map reduce model

Google have a web page and have crawlers

Request gets sent to group of machines in data center as a whole. What comes back is combined results from all of the machines. Map reduce model. Map the search to the machines where the data is, get partial data and combine data and return results. How many unique words are there in the google index? How many words were ever written? Every machine has to count words written on pages it looks after, send results and combine them. This is an example of an embarrassingly parallel problem.

Dist Systems Lecture

27/11/2014

Pastry allows us to take a network of machines(a large set of connected nodes) and link these up and makes it possible to route messages through this network without knowing details that we would normally need to know.
Prefix routing
At each node there is a local routing decision made by that node based on info it has about its nearest nodes.
Alot of message traffic is constantly taking place among these nodes, your message will be bounced through these nodes with the intention of finding its destination.
Eg. node M knows about C, D, E and F but doesn't know about X.

Pastry is an example of a 3rd generation peer-to-peer system. At the heart of Pastry is the idea of prefix routing(see paper "Pastry: Scaleable, decentralized object location and routing for large-scale peer-to-peer systems"). We take a 128 bit number, called a GUID. Message ID: first 4 bytes of the 128 bit message. Take the message identifier coming in and compare it against your own. Look for someone with more digits of the GUID value in common than me and send the message on to them. This process continues until the destination is reached.

How the routing information emerges
Start with a routing table, this has points evenly spread around the network so that you know somebody everywhere. Table won't be as big as the entire address space but covers the whole address space.
If we find our destination node in the routing table, we have succeeded.
If we come to a dead end and can't find anyone closer, we just send it on to an arbitrary node(?).
Leaf set is the set of nodes numerically closest to the current node.
Neighbourhood set are the nodes that are physically closest to the current node in terms of network latency. How long do pings take between the two?

3 Different tables: routing table, leaf set, neighbourhood set.

Pastry Routing Algorithm:

- If D is the message target, if D is within range of the leaf set(you already know D), then just contact that node and give it the message.

- Assess the number of digits you have in common and find the node in the routing table that maps to one more correct digit, if there is a node with more correct digits, give the message to them.
- If we don't have a node with more matching digits or cannot contact this node. This will be a rare occurence. Forward to a node T who's an element of the leaf set, routing table and the neighbourhood set. If it's at least as good as us, send it on. The neighbourhood set will be the best option because it will be the closest physically, easier to transmit. They're also likely to have completely different routing information because they don't match to us.

Existing network, we have GUID values 21, 100 and 1000. We want to join the network and we have no GUID value. In order to join, we need some entry point, in this case we need an IP address, can be any node in the network. Assuming we know node A, we can communicate and ask can we join the network. What's our GUID? It can be any random we want, less than 2^128. Why is this value so large? Because we use it to generate a random number. Chance of generating 2 GUIDs of the same value is very low. Assume we generate the random GUID 30. We send a message to A to self-identify and state that we're joining. This will lead to us getting our leaf set. Every node can send back to me information from their routing table so i build up my own routing table.

Eg. Squirrel is a web proxy system based upon Pastry. 4 or 5 hops is usually the latency for getting a message from one part of the network to the other, even as the network scales up.

If someone wants to send a message to me they must know my identifier, but everyone is identified randomly. If you need to route a message to some specific known machine, Pastry isnt useful. It is useful for using resources of all of these machines together.
Taking nodes randomly in a network and providing i can generate a GUID value that everyone can agree on, i can use it to route traffic to that node. Finding nodes whose GUIDs are numerically closest.

possible exam question.
no secure network, different nodes responsible for different tasks

Need to ensure that nodes in the network can agree on mapping of a movie to a GUID.

We could have an index, or alternatively some sort of hash function. Take name of movie/thing of importance etc, and apply hash function to get ID value.

Distributed Systems - Project Overview

First step: Designing a protocol

open, close, read, write - NFS
open/close are basically null ops with NFS and read/write are key.

2.1: Are we always connected? Or is it per operation? Do we keep connection live or kill it? Maintaining connection is expensive. Do we need to keep it live? No! It's recommended that we do it per operation. Every operation that the client proxy is doing causes a new connection to be made to the remote server and back down again.

Clients establish connection, ask to do operation and then go away again. Server: loop to accept connection, pass to handler, perform operation and then disconnect.

Opening a file: We assume that we know the name of the file on Server X. We know the port number. What message do we need to send to get file?

OPEN : filename
IsNEW: 1/0 (1 means we want file created even if it doesnt exist, if it's 0 then the file should exist and we dont want it created)

We could come back later and add things such as lockID, token, payload for encryption etc.

What should come back after this message has gone to server? Some form of acknowledgement.
OK: filename
Or…
ERROR (with some number that you have defined)

For NFS, closing will be quite similar(CLOSE filename).

What should we send to read a file?
What do we generally need to read from a file in java?

read(filename, startPos, length)

How will this look in our protocol?
READ : filename
StartPos : n
length : n

(newlines break up the content)

Response to this:
OK : filename
StartPos : n
length : n
...followed by the content of the file
...followed by \n\n
(use convention where file can't contain \n\n)

Debugging info could also be included in Smart Proxy's response(eg. i tried to access x but got warning y)

What should we send to write to a file?
Do we want to allow overwriting *and* inserting? We will assume that it will always be overwriting. No requirement to support insertions.
WRITE : filename
StartPos : n
length : n

Some simple file storage system is required. Memory? A string? Use local file-system?

2.2: Security

Authentication Server(kerberos)
Basic idea: we authenticate, and what we get back is a key and a token. The key is inside the token. This is encrypted with a special key called the server key, only known by the authentication server and the file server.
Only the file server will be able to encrypt the token to get access to the key.
USERNAME : …..
test : …….

Use the password that the user gave. Look up user (in auth server), find password, decrypt to get expected string.

OK
token :
key :

What does the message I'm sending across the network look like?
Read : filename
StartPos : 0
length : n
IsNew : 1/0

This is the first place we have to think about layering. The best way to make system adaptable is to componentize the system.

TOKEN: ……..
PAYLOAD : (an encrypted string(encrypted with key from authserver) with entered data) READ: filename\nStartPos: 0\nlength : n\nIsNew : 1/0;

2 steps, generate message, encrypt with key

request to server: shipped with the token
response from server: just the payload. content is returned in an encrypted message.
All messages and responses are encrypted with the key, this is called the payload.

Should be possible to write the base system and push the encryption functionality in as a layer.

2.3 Directory Service
Basic overview: Send request to directory server saying 'where is filenameX'. Message will be returned containing ServerID, Port and filename. A datastructure will be provided that says 'here's a set of directories and here's a set of locations for these directories'. Static directory structure that we will be given and our directory server will have to respond correctly to it. Test code will be provided.
Simple structure, like a hash table. Take data structure provide, hold message in hashtable.

2.4 Replication

Our replication implementation will be based on the assumption that directories are replicated, not files. Replication: when someone changes a file, we need to spread this and ensure that all copies are consistent(immediate consistency, not eventual). (gossip vs primary copy model***)

DIRECTORY : Fileserver1.Port: master
DIRECTORY : Fileserver2.Port: slave

Write to master(primary copy), read from master or slave.
Write to primary copy, primary copy knows and passes changes through to slaves.

2.5 Caching
Looking for client-side caching. Server-side caching: too messy to look for. If client A caches data from file, client B writes to file and cache A caches data again, there should be a difference.

2 basic models for caching:
Easier model: Create an additional message type. Can't use a lease. We have to invalidate cache A if client B has written to X(where client A has cached its data from). Additional message files. A polling solution, server will let you know if file has been written to since. not ideal.
Every 5 secs, send ping message to any server you have a cache from to see if cached data is up to date. We'll probably need to timestamp files.

Harder model: if change is made, push it to anyone who has a cache. server has to know which clients have a cache.

2.6 Transactions
Persisting data to disk, roll back if something goes wrong.

Shadowing - Make a copy of directory, pointing to a new version of file2. Anything that is changed will be copied off to a new copy. Copy everything you're manipulating and then use a single operation to swap the data everyone else sees with the data you've updated.

Logging - Record every operation and also every undo operation to create a log. When something goes wrong, look at the log and see how far you got through the log. If

everything in the log is listed as finished then the task is complete. If everything in the log is not finished, you can roll back.

Use a transaction server. Ask everyone to enter ready-to-commit stage. Wait for them to respond. If they respond that they're ready, ask them to commit. If they don't all respond, abort the transaction. All nodes who were ready to commit will unroll the stuff that was part of this transaction if one node doesn't apply affirmatively. 2 phase commit.

Can be challenging to implement…

2.7 Locking

Quite simple to implement. Gain access to exclusive resource by asking for it, if you get a positive response, proceed.

You could implement a ping style model to allow a node to keep the token as long as it responds to pings or alternatively do a lease where a token is given to a node for a particular amount of time and the token is passed on after this period whether they respond or not.

The client-side code that we write must adhere to a process where a file is to be locked(a parameter to attempt to read/write) and the code must use some sort of locking server.

Notes

Implement a very simple interface to all of the segments on top of whatever structure we've built. Build the system piece by piece, build it in layers.

IPC -> RPC -> Distributed Objects -> Component Technology -> XML -> Web Services

XML - self describing data, easy for machines to parse, allows you to build hierarchical data structures. Allows you to build tools to manipulate data you've never seen.

Lecture - 4/12/2014

Distributed Objects -> Event Models. leads to coupling and software adaption, systems need to be separated into components that aren't designed to only be compatible with each other, decoupling systems to make them compatible.

All of these things end up in web services. For general web sites(eg. ticket booking) intermediary site fetches relevant information from all of this hidden technology.

Data driven routing, not knowing the end target but letting an intelligent network determine who should solve this problem.

Transformation of the web economy from pre-web services to post web-services.
Things that changed the system:
XML, change in the way we build systems, service driven systems.

Web services refers to XML web services(not REST).
1.Before XML web servers: appliance to server interaction, HTML and HTTP are used as the standards to post the data. Access server, go through firewall etc.
No relation between what the web browser is seeing and what's going on in the backend.

The way in which data was being encoded was too difficult for anyone outside of client server interaction to look at and inspect it.
pg. 7 RMI/IIOP
Object-object transfer but it was too difficult to build something to look at this content and manipulate it. Closed relationship.
This model was only for big organisations.

Why administrators say no
Real world systems are deeply complex and systems administration is a serious task, not trivial at all. The main issue to getting client-server interaction working is that there are firewalls in the way. Firewalls need to be altered to let CORBA traffic etc. through.

The web service solution has some of the same underlying tech as point 1.(HTTP) without the horrible solutions. Subvert the web delivery, use object to object invocations. Shipping HTTP requests. Use XML.
SOAP: way of encoding object-object invocations using XML for format. Flexible because anyone can look at it, admins and firewalls can look at it and stop it coming through etc.

We're just deploying services, treat it as if you're building a website. Not returning HTML, returning SOAP references.

Built on the assumption that you're publishing a service for general consumption.
What's a web service?(eg. exam Q)
Self-contained, self-describing modular *application*. Published, located and invoked across the internet. allows you to build loosely coupled systems.

Everything in dist. systems is based on standards, we've been used to large committees of people coming together and deciding on these standards.
Web Service standards are not decided upon in these ways, decided by companies(probably part of their success), give it over to a standards body once its been designed, but don't decide by committee.

First, we need a way to describe services(an IDL): WSDL.
Then we need a way to find services, because we're making use of other peoples services to deliver what we want to do. Standardized look-up. UDDI(used mostly internally in organisations, not so much externally)

How do you do invocations so that remote side can understand it? You need a marshalling standard: SOAP.

described using WSDL, located via UDDI, encoded in XML and encoded over SOAP.

XML
Exstensible: we can add a heading to an old xml doc, we can add to new doc and this won't break it.

XML Schema Definition: Encoding of info written XML that explains what another piece of XML should look like. Self-describing.

XML is just a data format, a way of encoding data.

SOAP
A way of using XML to encode an RPC. not an object invocation. Has a recursive XML structure. You can send it over any transport(HTTP, email). See critiques of SOAP in notes. Not good performance: SOAP is a dog

WSDL: an IDL. Very verbose. A flexible way of invocations(call-response but also complex interaction models: call - delegate - response etc.), but we would never want to read or write it. Not an object, deployed on a network and you can invoke methods on it.

UDDI What is a UDDI? It's a registry. Let's you do queries like <find_business "acme travel"...
Large scale, general purpose registry system. Generally not used externally, outside of businesses, you'd usually use google.

Note: How web services are now: people use a lot of services like REST. not examinable tho.

Example: Travel Agency Web Service

travel agents, banks, doctors etc are examples of information intermediary. the knowledge economy: jobs that can be replaced(eventually by computers).

Old scenario: Customer wants to book a holiday, look at deals etc. Travel agent has already previously found good deals etc, and information about packages.

New scenario: travel agent is now a machine. Travel agent is replaced by an algorithm that presents real-time information that it gets from service providers. Much more choice now.
How does this new model work?
The customer will talk to a travel agent(eg. expedia) who talks to UDDI registry to figure out who they should talk to to find out about flights. They find list of flights, you make a choice and then you go onto the next page.
Step 1. User issues a request. Now we're in a service, eg expedia.

Step 4. We get a T-model instance back from the UDDI registry with WSDL description of how to talk to relevant remote service.
Step 5. Use WSDL desc to issue a SOAP request to get prices. SOAP msg embedded in HTTP post.
XLST
Step 7. Get price info back out.
Step 8. SOAP msg embedded in HTTP request comes back.
Step 9. Travel agent enumerates all the flight prices and returns them to the user in desired HTML form.

verbose, slow technology, but wrappable among businesses

Booking a flight - what happens technically

Lecture- 8/12/2014

PASTRY routes message to whatever node is numerically closest. This doesn't allow you to address specific nodes if every node takes a random GUID value. A message can be given to some node and using a hash function of some kind… you get a rendezvous point(asynchronous communication via a shared point(eg. park bench)).

Typical exam question: **rendezvous point**
There's a volatile network and we've got nodes that are connecting to network and disconnecting too. We start with a node A, later on gets disconnected from network and then reconnected with a different number, a different network ID. This happens for clients but imagine a world where it also happens for servers. Over time, Server S ends up with a different number. Exam question example: you're allowed a stable node that has some known IP address that doesn't change, always guaranteed to be there, how could the client and server talk to each other? TCP, UDP, remote procedure calls, provide a libarry that provides some sort of API for sending messages, eg. Socket API, you'd be doing it to the stable node, expect client to talk to stable point back and forwards and server to talk to stable point back and forwards. Piggyback. They'd have to either poll the stable point or a callback model to ensure you can always grab the message.

PASTRY lets you create rendezvous points in the network, random place, not a specific node. This is all it's able to do. Both client and server can use this node as a kind of stable node, always be able to route message to node presently closest to that value

Web Cache, if you're looking for a web page you first ask cache for page, if it already has a copy of the page you're looking for it gives it you straight away, if it doesn't it goes out on your behalf(web proxy), gets it, stores it and gives it back to you. People want same stuff, only need one call to go out to the internet to get the stuff. Pages go stale. Store copy of page for some period of time, as long as requests come within time period we give copy back and do go looking for page. Typically contains combo of content that is cached and fresh content.

What does Squirrel try to do? It tries to get rid of the web proxy. How could we use a prefix routing model like PASTRY to do the web caching but without the web cache, centralized proxy server. We're going to use disk and internet capability, processing power of all computers, to cache web page together. Could you use this kind of a network to do this task?

Every peer, no servers, will be running this pastry like prefix routing capability. We take a web proxy functionality and put it on your node. Node is part server part client hybrid. Provide web proxy, your browser points to this proxy running on your machine, going to localhost web proxy instead of CS web page etc. What it does: when you go looking for a web page it will ask the peer network for the page, a complication would be to cache things locally as well. Asks all nodes for that web page, if it has it, that comes back and your web proxy presents it to the browser. Either some node will go out looking for that page, acts like its the computer science web proxy, or your browser goes out looking for the page and stores it for the rest of the nodes.

Ask the peer network if it has the page, if it has it, give it back to me, otherwise either someone get it or i'll get it and store it for the rest.

Pertinent question, how do we ask the network if it has a page? We need a way to turn URLs into GUID values, use a hash function.
Home store - node holds it itself, acts like a standard web service proxy, receive a request for page, doesn't have it in directory structure so goes and fetches it. GUID values must get spread across to all nodes for all range of URL values so it doesn't become centralized.

The msg we send hops through network til it reaches the home node which either makes decision to go out and get it and returns it.

find a node that should be delegated as responsible for getting page.

Directory model -Same msg. home node doesn't go and fetch page for us as it has a table of machines who have told us they have cached the page and delegates to them so that they return the page.

(As long as we can turn a URL into GUID?)

pg. 8 diagram
X-axis: size of cache on each peer node, how much disk space is devoted to storing pages on each node.

Y-axis: external bandwidth being used by all of the web browsing activity.

top line (no web cache) amount of bandwidth that would be use if there was no web cache at all, bottom line shows you the standard centralized cache. You can see that we're basically saving from 103 down to 87, lot's of network access being saved.
Centralized solution is as good as you can get, we have effectively infinite cache if you're caching every page.

How does the peer solution compare? Depends on how much disk space you make available on each machine. As you increase the amount of cache available you improve the performance … at relatively small disk space per node.

If you don't have enough cache to cache all of the content you will be losing some of the content.
Conclusion: You can build equivalent performance in terms of bandwidth saving using a peer method with either model with relatively little disk space on each node.
Why doesn't the CS dept. use this model?
Dependant on there being a large number of nodes in the network at any one time. You need nodes to be reasonable reliably in the network or else you're losing all content.

What does the CS department care about with providing a web proxy? **Security**

What are they most worried about? Eg. posting on terrorist networks, DLing pornograph, hosting some server internally somewhere and selling things somewhere, hosting some business somewhere.

Famous example: Doogle.dsg. Doogle is the web search server of Ireland, simple layer on top of Google, re-presents what Google is giving you. People got into trouble over this because it was put up on college network and didn't look good for the college… blah blah blah

Easiest way to solve this security issue is with a centralized server because you have control. A peer system is effectively impossible to do this with? Why? We might put in some filtering, where web proxy pre-filters sites we don't wanna see etc. It won't work cos you can replace this implementation without your own and we couldn't tell if you were running our implementation or yours.
Secure multi-parted encryption may in the future allow us to do scalable computation where we know the code we want to run ran, but currently there's no way to guarantee the code we give you is the code that's running on each nodes.

Basic problem: we'd be limited by whatever interaction protocols between nodes to filter data etc. because we can't guarantee that nodes our running our code in a peer network.

Squirrel performance: if we're not worried about this security problem we'd be saving bandwidth at least. Is it an expensive option in terms of performance? When we enter a web address we expect the page to come now, eg. delays more that 2 seconds should not be acceptable.

Any node knows about other nodes across the network, premised on you knowing other nodes, as long as you know someone this is a way of getting across the network fast. Performance-wise Squirrel works well, possible to build decentralized web proxy system that has manageable performance.
It is a viable model, only issue is security.

Lecture - 11/12/2014

Election Algorithms

Election - Choose from a set of parties one party that's going to be in charge. One party in charge. Unlike mutual exclusion this is not about fairness, just about picking one.

Bully Algorithm

P1 has noticed that there's no leader, something's gone wrong and we need to choose someone to be in charge. Assume nodes are indentified by an identifier and they know about all nodes in the system. They're ordered based on ID and everyone knows about everyone else.

Previous coordinator is dead.
P1 notices this and sends an election message to every node i know of that has a higher ranked identifier than me - assume UDP.
Sends message out to everyone and starts timer, waits till it gets an answer message back - if so, its finished. If it doesn't get any back then they're all dead - meaning it is the most important, most highly ranked node. All nodes ranked lower are now told P1 is the boss. More likely to get an answer back.
Each node that receives an election message sends back an answer message and starts their own election to see if they're the boss and so on.

Scenario: P3 is killed, the coordinator who was supposed to be in charge is gone away. When you get an answer message, each node that receives an answer message starts a timer. Timer 1: Did i get an answer message back?
Timer 2: Did i get a coordinator message back?

One node will time out waiting for coordinator message because none came through. Go back.

Eventually, they will have started elections, P2 will send a coordinator message back to P1 and we will have our system again.

Trying to establish a single node that's in charge when something has gone wrong.

If a node comes back it will start an election. When a node gets a coordinator message it knows that it is now the coordinator.(?)

The bully algorithm isn't fair, won't guarantee everyone gets a turn, whichever node has the highest identifier gets a turn. Problem with BA is that there's a lot of messages.

Ring Based Election Algorithm
Send message around ring saying 'i want to be in charge'. If your ID> the ID of the message passed to you, send your ID in the message, otherwise just pass it on. If 7 sends a message around and no node has a greater ID, the message goes all the way

around and the ID isn't replaced, node 7 will then send a message stating that it is now in charge.
This algorithm is cheaper, but it does not work if any nodes fail.
This would be used in normal circumstances just to choose a node such as a primary node, the messaging cost is low compared to bully algorithm.

There's a tradeoff: Trade topology to reduce the messaging but generally speaking, increase the fragility of the system.
Take various models and assess when it will work and when it will not work. If you can apply a model and state why you think it will work in the exam.

A different problem is mutual exclusion… will cover in revision algorithm.

Component Technology - EJB & J2EE

RPC, RMI etc do an elegant job of allowing us to talk to a remote thing but doesn't ...
how do we make sure we can do security?
how can we do proper, scale enterprise stuff?

Taking nonfunctional, non trivial aspects of a software problem and putting it into a package that makes it easier to write software systems.

Component - the idea hasn't changed although the tech has. It is essentially something smaller than an application but bigger than a class. A subsystem. A unit that's probably reusable, but it is *replaceable*. Trying to decouple it from system to make sure it works independently. You build software by plugging together various components. Modular. More like a code-reuse scenario than web services which plugs together different services.

EJB - Enterprise JavaBeans

They all have some form of a container model- effectively a hosted environment in which your code is going to run. Like a framework, you just provide the bit at the top of the stack that's called on to do the programming job, the container is doing all the plumbing work of the distributed system.

Built on top of RMI. Makes distributed programming easier.
You aren't free to design how your code works, it has to work in a specific way.

How does the model work:

there's an EJB server and an EJB container, they're provided. Then you're providing the services you write: the enterprise beans, basically just a java class.
Then you have clients, like rmi clients, access the beans.

Server basically provides the operating system for everything to run on, an application service running on top of an OS.
Container - provides application-specific things: transaction report etc.
EJBeans run inside this.
Client doesn't talk to the beans always, talks to this infrastructure and sometimes accesses the beans(hehe).

You get all this hard-to-do stuff for free provided that you stick to their regulations etc.

Think in terms of large teams of people doing stuff, could be 300 people etc. Need to start splitting work that needs to be done so that people can do it independently.
You need to allow people to develop parts independently so that you can combine it all later.

Tool-driven. different people are provided different tools.

Basic container model is all we really need to know. We're just covering the basic ideas.

JNDI Java Naming and Directory Interface
Can't just look up URI. To get hold of an object you have to go through JNDI.

You talk to the JNDI and get a context, you then use it to get the object, a class factory interface gives you access to the object.
a class factory is a class that lets you create instances of another class.

When the client talks to your stuff it won't actually talk to your code, it'll talk to the EJBObject class & interface. Like a stub. Does the under-the-hood work for client server interaction.
Finally you have metadata, XML-type structures, that allow deployers to turn on basic behaviours. Deployment specific concerns. Allows the deployers to fine tune the operation of the system.

That is the container model, everything else (for us) is irrelevant.

Microsoft .NET Remoting

Remote invocations - what goes on under the hood in doing that invocation across a network. Get a broad sense of how it works.
We're taking an invocation, we're going to marshall it, ship it across the network, send it to the server etc.
5 elements
Proxies: generate messages.
Messages: What's going to get marshalled ultimately.
Message Sinks: might compress the string, encrypt, add security details. same style as EJB stuff, less sophistication
Formatters: take the dataset and format it
Transport Channels: How are we sending this across the network, TCP to a specific port number? is it a web service? are we FTPing it?

Proxies: masquerade as a remote object. transparent proxy and a set of other objects that take the invocation and set it as a message object that's passed to the sink.
Example: you cast an object to the wrong type, you'll get a runtime error exception.
Given back an instance of a transparent proxy which is a standard class that knows nothing about your interface.
Uses metadata. interrogate metadata, uses type information, does reflection.

Message: data about to invocate. open set. you can add to this and remove stuff from it. eg. the token that was used to encrypt the content.
...If you're encoding it to be a web service call.

Formatter: turns the data set into a marshalled stream that can be shipped across the network. You can change the infrastructure, entirely configurable.

SOAP message format etc. Headers for SOAP remoting….

Client side messaging model
TransparentProxy -> RealProxy -> Goes through a series of sinks(preprocessing) -> eventually we give the invocation to the transfer.

For one simple client to server invocation there's a lot of stuff going on. Everything you do on the client side you'll be doing something similar on the server side: eg compressing on client side - decompressing on server side.

30-40 separate invocations for each client to server invocation - expensive.
.NET remoting questions will refer to how its adaptable and how its expensive

Think about how you'd design systems based on what we've done so far and how would you communicate it within a 40 minute period. Knowing the notes won't get you through the exams.

Distributed Systems Revision

Look at past papers - design topics
6 Standard Questions and their variations

Peer-to-peer
2014 Q4 (b)

Attempt 3 questions of 4.

Question Structure
All Q's carry equal marks, 33 marks for each.
Design part of the question gets about 20 marks, first part gets about 8 and the residual gets about 5 marks.

Tip: If you want to do well you must do well in the design parts. first part is a theory question: explain prefix routing, what is a holdback queue etc.
There will be a peer question, there'll be a replication question, something about an intermediary website solution, a web services themed question.
Last part of question asks for opinion that goes beyond the course possibly, eg: imagine we had a question looking at javaspaces, it might ask whether javaspaces would be a good model for doing twitter feed processing etc, provide a decent, reasoned answer whether you're right or wrong.
You can get close to passing by knowing definitions for the part 1's: prefix routing etc.
If you have a reasonable effort on top of this at 2 design questions out of 3 you're comfortably at a 2.2
Reasonably easy to pass this exam but reasonably hard to get a first.

6 Standard Questions

1. Web Service Intermediaries - expedia.com for travel bookings, etc. Classic design Q that involves defining an intermediary that can talk to other web services in real time to pull info back for the user. Other variation on this called a federation (eg. libraries with inter-library book loans). Each acts for its own clients as an intermediary.

2. Volatile Network with stable nodes - network where everything is unreliable except for a few rendezvous points. How do you build communication with nodes that can't all stay on the network. Solution: take our stable node and push/pull data. Kind of a transparency question. You can layer something that looks like sockets or RPCs. Under the hood. Changing the underlying comm. model and still being able to provide point-to-point comms.
3. Peer to Peer - a light web search is a good example. prefix routing for store and retrieve. Use prefix routing to identify a place in the network randomly where everyone can push/pull info. chat, decentralized distrubted file system. the msg you want to send you need to uniquely identify a GUID. Interesting variation could be to combine P2P with volatile network. How could you make your solution work if you only have a peer network rather than stable nodes?
4. Distributed File System: probably a good bet. Likes to ask questions about project work. Know how you built it and know how to write about it.
5. Replication - something Gossip-like. bulletin boards being replicated across distinctions. Media streaming so everyone can watch a football match etc. Key part of this Q: Data-loss. How do you make sure a failure to copy stuff can be recovered. Media Streaming: you don't care if you get a little bit of data loss.
6. Game - A game of some kind. Sequencing events. Could Replicated state across a set of nodes so the board is replicated to everyone, specific sets of events need to be accurately synchronized, eg. every node needs to agree who was killed etc. Game of Life: classic example. distributed GOL. state will change from step to step, if you take chunks of this problem and move them onto different nodes. the edge of each node impacts upon the other nodes. Might want players to be able to create patterns and follow patterns around the place. Player needs to be focused on subset of nodes, pulling info from set of nodes.
7. Grid Processing - break the problem eg. java spaces, classic grid style technology. Q: a problem either can be broken into small sections that can be independently processed(embarrassingly parallel), or else processing in nodes relies on previous results. Compute prime numbers by eliminating multiple, all the multiples of the prime no's are eliminated. If you took a number line, you could break it into chunks and give each chunk to a node to eliminate multiples of primes and discover new primes. Would need to be organized so that discovered primes could be fed down the chain. In theory the work could be broken down but each computation has to be ongoing and connected: the result of one nodes work must feed into another. This is an example of a problem that can be (to a degree) given to different nodes, but is not embarrassingly parallel.

Time, synchronization: doesn't really come up but could be a bookwork question. Sometimes you might get a question around mutual exclusion.

What Should Go Into an Answer

Take any question and it's only going to be a paragraph or 2.

A TCP based client/server ftp system …. to be used over a volatile network. Outline a replacement socket server…..
This is the volatile network question.
Client talks to Server assuming a TCP type stream connection - backwards/forwards. Only problem is: over time the client might disconnect and then maybe reconnect/reappear. Or at some later time the server might disconnect. Disconnecting may also mean it comes back with a different identity. Could be a server process running, has different network identity. How can i maintain a stream going between that client and server where you can't guarantee. Network -> client which connects/disconnects. Can't guarantee the IP addresses. Can't even guarantee they'll be on the network at the same time.
Piggyback node on network thats known to be always on(stable node) and always has same identity. Connect to it instead, think about the general operation. Opening means connecting to this stable node(known IP address). Resolve server names from DNS names to a particular node.
When the server wants to read traffic it will talk to this node and pull content from it and when it wants to write con. USing the capacity of the stable node to maintain connections to these nodes as a server to allow them to piggyback back and forth to this node.
pub/sub - publish/subscribe
Aspects of this model don't match the streaming model but you can still mimic it with some accuracy, this question is asking about transparency.
Variations on this could be: RMI model, web service calls to operate this way, a UDP model which would be much easier to deliver than a streaming connection.
Go through each of the operations for the stream interaction and state how you'd provide it. Accepting connections would have to involve polling or a callback mechanism but callback wouldn't work in a virtual network so polling for new connections would be needed.

How to Answer a Question

- Start with the question. The question is always very small. The first thing you should do is elaborate the context: what are you designing for. Provides you an opportunity to put context that can design to. If he doesn't specify something has to operate on a WAN and it would be easier to do on a LAN then state this. Set the scene in the first paragraph. What you think the core of the problem is. *2-3 mins*
- Architecture. Give an architecture. Set out the overall solution of what it looks like. What are the components? How do they interact? Where are they deployed? Look at UML as its the simplest way to describe a system. A formalised drawing. Sequence diagrams. Discuss general solution. Platform Independent Model.
- Make sure you deal with important failure modes. How it would work if things break(eg. volatile network. what happens if server goes down as client is communicating its request. Or server has pulled request and is processing it but dies while it's doing so. Some kind of time out, if client hasn't gotten response after certain time it's failed etc).
  15 mins, up to 3 pages with UML diagrams.

- Next, deal with technology: How to build, what to build with. Platform specific model. Make the central thing an SQL database? Client will literally just push a record into the db and the server will pull records, etc. Maybe some things you'd put in an EJB container. Maybe you'd deploy some things on amazon web services.
  List the set of technologies you would use for each thing. *1 or 2 paragraphs, maybe 5 mins.*
- Last piece - tell Barrett how good your solution is. Be realistic. It's normal that the idea you start with may not fully work out. Show him why it doesn't work and your thoughts of how to solve, some kind of analysis of your general solution. Tell him even if you know it's not gr8. *2 -3 mins*

Manipulate the question as much as you want to find the version of the question that you want to address. Use bullet points rather than paragraphs. Draw diagrams.

P2P

"Design a p2p based web search solution, equivalent in functionality to a centralized… that uses user browsing as an alternative to web crawling…."
Barrett's workaround: Web search system within a closed network. Make this a corporate network - don't need firewalls etc. Assuming user will have to have made open connections etc. Assume that the resources of interest are external plus internal (optional: deal with whether or not people have access to information). Going to assume desktops+laptops, ruling out mobile devices. Assume a PASTRY-like architecture. Assume prefix routing. Going to assume honest users because it's being used by an organization, question doesn't mention page farming.
('Rank queries in order' could devise some sort of a centralized ranking system.)
If it was asking for the ability to deliver analytics - it'd be easier to centralize this component.
We're going to rank pages based on access:
Architecture: We have peers and the internet. Built as a kind of proxy. The web browser talks to the proxy, the proxy will fetch the page and pull it down for display to the user and it'll index that page. Draw a diagram. Send out a series of index messages to all the other peers saying this particular url matches to this particular port. putting functionality in the processing of a standard proxy to start up an indexing process on any page it receives, returns the page the user wants to look at. Every peer will have a proxy. Assuming it's a single network and that everybody can connect to everybody.
Fetching a page returns a page, index messages go out
Sending a msg to a node using prefix routing, GUID is address for msg, define msg somewhere in the text, use the IP address of the node to uniquely identify it and the URL. every time you index a page you get lots of messages.
Show what peers look like internally. There's an indexer, a proxy component, maybe we cache pages. Client acts as a retriever of web page, pumps them into network, prefix routing.

Redesign for rank by user accesses - popularity of pages.
Need different store that matches URL to GUIDs. IF you're looking for this work this is a good url. Every time we hit a page, node has to send out message: 'for URL, here is the user id that used it'. Mapping to GUID is for URL.
Data stores that map words to URLs and that map URLs to hit rates.
That's the pushing-in problem. Now the retrieval problem:
Let's assume the search is for one word. The search goes to a particular node for one word and it responds with a set of URLs. These need to be ordered in terms of hit rate. How do we order them? For the solution above I'd have to go out and find the hit rate for all these URLs. This suggests that we'd have been better centralizing it. We could also cache the hit rates, or do a random sample.
Write about these limitations, 'centralizing would be a much cleaner solution and would be fine in a corporate setting'. The issue is that when the search is being done, this node has to return the results to the users, too much activity to be processed right now, will be a long time before we know the rankings.
A better solution would be to think about how you'd pre-run all this stuff.
Way to pre-compute the ranking: every time i say a particular word was in a url, if i instead send that message through for everyone i'm effectively distributing that hit across all the nodes. Sacrificing bandwidth to ensure that i have the information available.
To re-cap: Pull a page, get a set of msgs that need to go out, that are routed to nodes, these msgs will include the word and the URL, that will arrive at a specific node and the node will note the word and add the URL if it doesnt exist or add to reference count if it does. a count of the number of times users have hit a page. Do the hit counting at the word index level.

If it was out on the net: Build the index not based on URLs but based on domains. bandwidth becomes a concern. I'd be able to cache this information about current counts for a domain.

Technologies: proxy would be a web service opens connections on the localhost, browsers connected to it. Use PASTRY as the prefix routing link, all the communication across the network is done by a standard library like PASTRY.

Evaluate the solution: expensive in terms of bandwidth, needs to ship messaging to all nodes for every word indexed on that page. Every time a page is hit i have to ship out potentially 300 messages.

fetch a page from the net, index it, for every word to be indexed send out a message, url is added to the index if it doesnt exist, if it already exist we increment the count.
The count of how many times the URL has been hit is distributed all over the place.
Alternative: we've already sent that URL to that word.

Server side: Prefix routing etc.

Example Question

Interactive 2d motorbike game. large no. of nodes. continuous feed of frames.
How would we simplify this problem.
2 problems.

1. its a multiplayer game
2. the game can't play unless everyone's there - what happens if we lose players

Elect a host to handle coordination.
Board is too big to be put on one machine. Players will be all over the place. Protocol: elect one node - the coordinating node for anyone who's currently in the game. Divide the board space so that each node manages one area and generates so that people know what's going on. Coordinating the precise movements, a total order: one node is elected to manage the total order. Particular nodes to processing on sectors of the board. If these nodes go down we'll have a problem but the Q doesn't mention resilience so this is ok.
Ignoring the rest of the board and just thinking about players in one section, need a model where everyone is updated.
When you join the game you're given a chunk of the board that you're running, you only see the chunk you're in.
Architecture: decentralized system with central node. Need model to ensure sequencing of events. Node probably needs some visibility of what's going on around the edges. A primary copy type model may work, A is the primary copy, pushes some info out to other nodes and pushes some to clients.
Push vs Pull.
Push is good idea, game is dependent on events occurring, changes in velocity and direction are the info that's relevant. Want to minimise work being done by client's because they're playing the game, so polling would be bad. Also might be a time delay.
Every node computes the impact until some node changes. If there's a failure nodes will have to continue computing positions.
This is the broad architecture, draw sequence diagrams etc.

Technologies: Sockets? UDP with some reliability control on top. Maybe introduce some kind of pulse message where the pulse might happen every 10 events, if you haven't got the 10 events you know something's missing so go back and look for them. Something like a GOSSIP-like holdback queue.
Establish a pipeline where boundary activity is given.
The amount of info that needs to flow around the place is actually quite small so a wide range of architectures are possible. Our solution is dependent on node's staying live.