

Section: Distributed Systems and Networking

1 Announcements

- midterms coming up....
- watch for extra office hours and such
- project questions?

Disclaimer: These notes aren't substitutes for attending lecture or reading the book. Those have been created with careful revision and review. These notes are mainly intended to be a more thorough index of the lectures and reading so that you can go off and read the full treatment on the topic and be able to ask the right questions while reading those two sources. If you have comments or suggestions please send me email and I'll fix them up.

2 Recap and Intro

Last week we spent a while on queueing theory and trying to understand how they relate to queues inside a computers. Now lets switch gears a bit to something we are all very familiar with: networks. Most people have heard about the internet (and you have too if you are downloading these notes) and the concept of addresses and understand that the computers are connected in this fashion. But thats about it. This week we'll go over the high level ideas of networking and the dive into what they mean for operating systems. Now this might seem like a very simple question.... but is it? What is a network? Is it the actual wire that connects the computers together? Is it the network card plus all the wires that connect the computers together? Is it the protocols that get run on those wires? Is it the applications that get enabled because of those connections? These are some of the questions we'll try to tackle this week. Lets move away from the movie trailer style intro into networking and into the meat of the lecture.

3 The Important "ilities" of Design

Last week we spent a long time talking about various file systems and I/O devices. One of the major takeaways from that section is the design of those I/O devices and systems that handle the I/O devices is completely dependent on how they are used and what context they will be used in. Lets take file systems: we said that certain file systems are good for random access and others are good for streaming. To say one is best doesn't really capture the big picture so we must try to find away to nail down what is important in our system and thus enter the various "ilities"

1. **Availability:** In this design goal the main aim is to make sure that the system is highly available and resilient to outages and try to make sure that requests (no matter how few) can always be processed.
2. **Durability:** In this goal we make sure that the data is stored properly. Even if there are outages the data will not be lost ... think bank accounts.
3. **Reliability:** Availability just says that the system has to be processing requests but doesn't say anything about whether those requests are correct. A search engine that takes queries and doesn't produce results is still available but is obviously useless and unreliable. So reliability is the stronger guarantee on this condition.

The reason that we stress these points is that different systems have different focus when you design. When you design a banking system durability of data is extremely important.¹ However if we are designing a search engine then durability is not really essential. It would be nice if our cache of the web was durable but its not essential to the operation of the search engine. We can always recrawl part of the web if we need to regenerate the data. However all our revenue and business is based solely on availability and how many ads we can display and therefore availability is the key metric there.

Question: I've given a couple of examples where the different design principles apply, think of a few examples of applications that stress one ility over others? What are some that require a good balance of all three?

4 Remote File Systems

When you log into one of the lab computers imagine how much of pain it would be if you had to constantly figure out what machine you were last logged into and copy files from that machine to the machine you are currently on. Until now the file systems we have mainly talked about only allow you to access data that is on the local hard drives, but how do we get around that?

As our answer to everything in computer science, build a layer of abstraction to get around. This is in essence what remote and network file systems do. They take the syscalls that the CPU generates and see what physical device the data is actually stored on and route the request to that physical device. How they do this and the details are better explained clearly in the lecture notes and I am making this problem a lot simpler than it actually is.

5 Distributed System

Before we get into the details of what a distributed system is lets understand what a centralized system is. A centralized system is where a set of computers all talk to centralized server to do their work. The soda lab machines are a centralized server system since there is one big server and all the little computers talk to that big server. Obviously there are

¹If data isn't durable, you get a lawsuit. The one thing that computer scientists hate more than segfaults is explaining broken code to lawyers ... not fun

some major scalability problems with such an approach so we want to try to come up with something better.

A distributed system is a system in which physically separate computers work together on a common task (such as serving a bunch of search queries for a search engine or handling multiple banking transactions). Distributed systems originally started out as a collection of machines within a building but today they have extended well beyond this limitation to very large physical distances between the various pieces. The number of machines that can compose a distributed system has also been growing at the same pace.

5.1 Goals and Realities of Distributed Systems

The main goal behind a distributed file system is that it can improve the various "ilities" above. With multiple computers instead of one there are fewer failure points and thus high availability. Data can be replicated to make sure that it is durable and because you have more systems servicing your request the chances that the entire system go down are a lot smaller.

However anyone that has spend some time with a distributed system can tell you that this isn't always the case. Simple probability says that if you have 2 events (such as a failure of a node) and they are independent, the overall probability that there is a node failure is the sum of the probabilities and thus implying a higher failure rate. In addition it seems very wasteful to mirror the data everywhere so in practice each machine usually takes a subset of the data. These subsets are often overlapping to get some redundancy but if a machine goes down, that subset is down with it. Once these machines are distributed over a network coordinating all those machines to get your task done is a lot more difficult. Think about how much easier writing serial code is than parallel code.

In practice distributed systems might be great ideas and have lofty design goals, but their implementation is a lot more difficult just because the points of failure grow exponentially with the size of the system. But some of the tasks that we require of computers today can't be done with a single machine and they need the distributed system to do it properly. Imagine how long it would take Pixar to render a movie, if it only had one machine on which to do it.

5.2 Transparency

You have all accessed a distributed system (if you have ever used a search engine or online bank account) to do your work. You were also not aware of the complexity of the system that you were working with. When you type a query at a search engine, you don't know whether the query is being processed 50 miles , 500 miles, or 5000 miles from where you are. You are also completely unaware of how many machines are serving up requests. All you see is a simple search bar and then results. The goal of building a distributed system is a clean interface and having it just work without messing around with it for a while and thus hiding all the complex details. This idea is termed as transparency.

In order to enable transparency we have a few goals that we can use:

- **location:** We shouldn't need to know where the resources we are using are located
- **migration:** resources can move without the end user even knowing about it

- **replication:** Can't tell how many copies exist.
- **concurrency:** can't tell how many other queries the system is serving simultaneously
- **parallelism:** task might have been automatically broken up to increase performance
- **fault tolerance:** system needs to hide failures that could happen within the system.

6 Networking

Before we start any discussion on networking lets begin as any other treatment on networking does with the basic definitions.

- **Network:** Physical connection that allows to computers to communicate.
- **Packet:** unit of transfer, sequence of bits carried over the network.
- **Protocol:** agreement between two parties as to how information is to be transmitted.

6.1 The protocol stack

Most communication networks are designed in layers so that on one layer has the responsibility for everything. This makes the system a lot easier to build since there is a clean interface between the layers. On most machines there are 7 layers and they are as follows:

1. **physical layer:** responsible for the actual wire transmissions of the bits and all the singal processing to send a digital 1 or 0 through an analog channel.
2. **data-link layer:** responsible for frames or fixed length parts of packets to make sure they haven't been corrupted.
3. **network layer:** responsible for providing connections and routing packets through the network. This is commonly known as the IP layer in the internet
4. **transport layer:** partition the message into packets and worry about controlling the flow. An example of this is TCP or UDP
5. **session layer:** implements process to process communication protocols rather than just machine to machine transport protocols.
6. **presentation layer:** making sure that both parties agree on a convention on how the bytes and messages will look like.
7. **application layer:** layer that interacts with users and the networking stuff that we see regularly like email, web, etc etc

6.2 Broadcast Networks

There are two styles of networks that are in existence today. The broadcast style of network is the older style. In a broadcast network all the machines are connected together through a common wire. In order to send a message you send it to everyone and only the destination reads the message. The main problem with this is that if two pairs of computers want to talk to each other then the messages collide and you have to wait to make sure the channel is empty. Kubi's lecture notes have a much fuller treatment on the idea and the problems with it.

Question: What are the advantages and disadvantages of this scheme?

6.3 Point-to-point networks

The communication overhead is exponential with the number of machines on the network with a broadcast network since the overhead of retransmitting and colliding messages is too high. Instead we now primarily use point-to-point networks. The idea here is that each computer can set up a dedicated channel of communication to another computer for communication. The main piece of equipment to enable such a network is called a switch.

Question: What are the advantages and disadvantages of this scheme?

6.3.1 Flow Control

With point-to-point connections and switches the problem of everyone sending to one host becomes apparent. Since the switch is now the center of attention we have to make sure that packets that are destined for the non bottlenecked machines don't see a loss in performance. Imagine if the entire internet slowed down just because Google went down that would not be nice. Flow control is the term used for making sure that everyone plays fair and can get the best possible bandwidth out of the system. again Kubi's notes do a good job of talking about flow control so i wont' go into to much detail here.

Now that we have point-to-point connections between the computers we can start to talk about how to set up these channels between computers that are right next to each other or across the country. The protocols that they use to communicate with each other are essentially the same.

6.3.2 Addressing

Now that we have the various channels set up it is important to figure out how to assign each of the individual nodes with unique addresses so that we can then send the packet to the correct place. The common IP addresses provide 32 bits worth of address space (placing a limit of 4 billion devices). And each address is broken up into smaller pieces. The upper part of the address are very similar to the city and zipcode of an address while the lower parts are much more similar to the street and house number parts of a normal snail mail address. Again the discussion about what addresses map to who are covered in the lecture notes and aren't really relevant here. The important idea is that the upper bits of the address correspond to a more global position and the local parts give a much better local positioning of the address.

6.3.3 Routing

Now that we have classified different parts of the address space we can then talk about how we want to route packets from one computer to another. Any particular post office in the normal mail system doesn't know every street in the US, however it does know how to send letters to the different zip codes. In the same way routers don't know about every possible address, instead they know that if the address isn't within their local, they can forward it on to someone who knows how to deal with it. You can imagine that we are constructing a hierarchy of addresses and routing tables in order to forward packets. Managing this hierarchy with new nodes and old nodes is the complicated problem the call routing.

We briefly touch up on how it is done here. In essence each machine is part of an autonomous system (AS). That is, if you send a packet that is destined for a node within that AS, then the AS itself can route the packets properly. For example if you know that a packet is destined for an address in Berkeley, the hierarchy of Berkeley routers can forward the packets where they need to go. Connecting autonomous systems through a backbone and sending packets efficiently amongst them is one of the chief tasks of the internet backbone.

7 Conclusion

The primary focus of this week was to create a way of connecting computers together to do useful things. We see that things aren't always as simple as they seem and that connecting computers via a network makes things a lot harder. However it enables a lot and is worth the effort to get it to work. This is easily evidenced by how the internet has dramatically changed society at large and we are only seeing the tip of the ice berg.