Agorithmic Data Science: Distributed Computation

Ian Wakeman

School of Informatics
University of Sussex, Falmer
Brighton, UK

ianw@sussex.ac.uk

Recap

- Processes and their lifecycle
- Threads of Control
- Synchronization by Locks and Semaphore, running in shared memory
- The Python Multiprocessing module

Overview

- Data Centres
- Synchronization by Messages
- Remote Procedure Call
- Fault Tolerant File Systems
- Security

Networks and Computers

As data scientists, your networks will be:

- an egress link into a Data Centre, and then inter-machine links
- high bandwidth - 1 GBit/s and better
- Low latency - apart from desktop to data centre

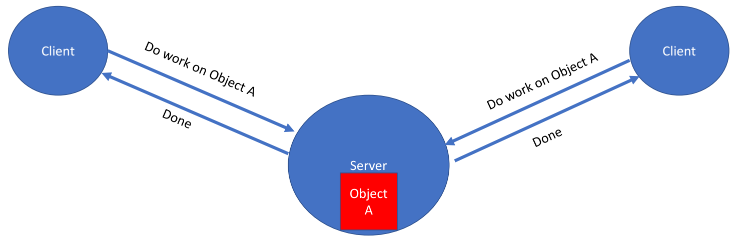
and your machines will be:

- Within a data centre
- Potentially virtual amongst many on a physical machine
- Mostly running versions of Linux
- Your processes will send messages to each other to communicate
- These message will use the TCP protocol for reliable delivery

Data Centres

Synchronization across machines

How do we ensure that the server doesn't corrupt Object A?



Synchronization by Messaging

- Having a single copy of data in memory is difficult to share over networks (can't use test&set at bottom as in a single machine)
- So copy the data to the machines that need the data, and use messages to synchronize
- We need atomic *send* and *receive* abstractions for our messages

The send abstraction

Ideal abstractions -

```
send(mailbox, message)
```

Send a message, possibly over network to specified mailbox. The mailbox is the data structure containing messages that is remotely accessible over a network
When does send return?

1. When Receiver process gets message?
2. When message is safely buffered on destination machine?
3. Immediately, if message is buffered on source node?

Choice depends upon system designer.

The receive abstraction

```
receive(mailbox, buffer)
```

Wait until mailbox has message, then copy message into buffer
In this abstraction, send and receive are atomic:

- never get portion of a message (all or nothing) - need to ensure buffer is of sufficient size
- two receivers can't get the same message - there is local synchronization on the mbox

Message Styles

- 1 way - messages flow in one direction (Pipes)
- 2 way - request response (Remote Procedure Call)

1 Way example

```
def producer:
  msg1 = [100]    # maximum message size 100 bytes
  while True:
    prepare message in msg1 # make coke
    send( mbox, msg1)

def consumer:
  msg2 = [100]
  while True:
    receive(mbox,msg2)
    process message # drink coke
```

Producer/consumer doesn't worry about space in mailbox - Handled by send/receive forcing process to block if no space

Request/Response

- Example: Read a file on a remote machine
- Also known as client server - client=requester, server=responder.
- Server provides “service” (file storage) to client

```
Client: (requesting the file)
  send("read␣/etc/passwd",mbox1)
  receive(response,mbox2)

Server:
  while True:
    receive(command,mbox1)
    decode command
    read file into answer
    send(answer, mbox2)
```

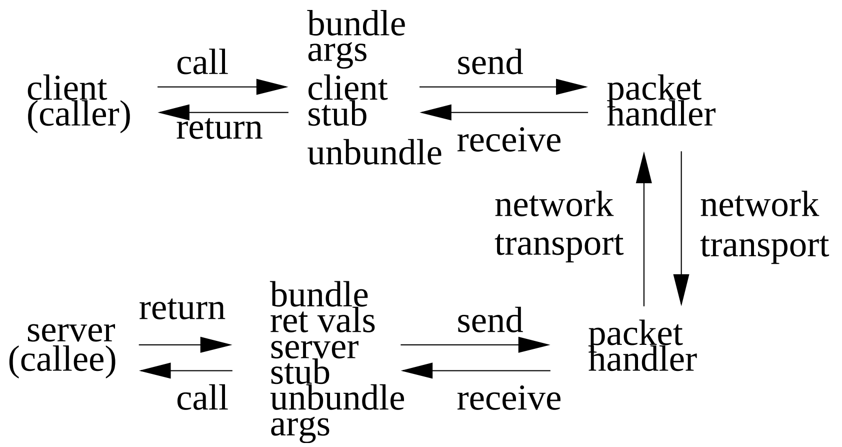
Server has to decode command, as OS has to decode message to find mailbox
What if file too big for response - then use a big message protocol (eg TCP)

Remote Procedure Call

- Call a procedure on a remote machine
- client calls:
  `rpc_read("/quadrant/random.txt")`
  translates this into call on server
  `read("/quadrant/random.txt")`

RPC Implementation

- Request Response Message passing
- "stub" provides glue on client and server



RPC Pseudo Code

```
Client stub:
  build message
  send message
  wait for response
  unpack reply
  return result

Server stub
  create N threads to wait for work to do
  while(1)
    wait for command
    decode and unpack request parameters
    call procedure
    build reply with results
    send reply
```

RPC Technologies

The more important technologies for Data Science are:

HTTP We can utilise HTTP using Javascript Object Notation (JSON) to build RPC systems. JSON mappings are available in all common programming languages, and allows easy interaction with RESTful services

XML-RPC and SOAP Microsoft created a distributed Simple Object Access Protocol (SOAP) mechanism where the concrete syntax used XML over HTTP, standardised as XML-RPC

Java RMI Java has its own RPC implementation called Remote Method Implementation, using serialised Java data structures. Hadoop uses a variant of this mechanism

There are many other variants, some of which are widely used for legacy file systems, such as sunrpc.

Fault tolerance

- If a machine fails, then we can still provide a service
- Probability of total failure reduced such as all data being lost, since data replicated across multiple machines
- If probability of failure is $pr(fail)$ for a given machine in n machines, then probability of loss of service is $pr(fail)^n$ and the availability of the service is $1 - pr(fail)^n$
- eg, if mean time between failure for 3 machines is 5 days, repair time is four hours, then assuming independence of failure, $pr(fail) = \frac{4}{5 \times 24} = 0.03$.
  Availability $= 1 - 0.03^3 = 99.996\%$
- We *replicate* data and *monitor* operations to perform fault tolerance
- Data Centre machines are commercial off the shelf (COTS) or commodity hardware with known MTBFs

The Google File System (GFS)

- GFS is designed for massive data that rarely gets updated
- 64MB chunks are replicated over *chunk* servers
- Where each chunk is, and which chunk is what file (Meta Data) is stored on *Master* servers
- Master servers monitor chunk servers through the generation of a heartbeat message
- GFS and the Hadoop Distributed File System are similar and underpin the fault tolerance of MapReduce
- Knowing the MTBF of your hardware, and the parameters of the underpinning filesystem, engineers can design in continual replacement and renewal of broken machines

Mid-Module Evaluations

Thank you for completing this short, anonymous survey - your tutors want to hear how you're finding the module and will respond to the feedback you give.

`https://bit.ly/SussexMME21`

Definitely agree	you have a clear, positive answer
Mostly agree	on balance, you agree more than not (even if just a bit!)
Neither agree or disagree	if you can, try not to use this: any other answer is better so you can have an impact
Mostly disagree	on balance, you disagree more than not (again, even if just a bit)
Definitely Disagree	you have a clear, negative answer
Not applicable	the question isnt relevant to you

Securing Machines

Goal: Prevent Misuse of distributed systems

- Definitions
- Authentication
- Private and Public Key Encryption
- Access Control and Capabilities
- Enforcement of security policies

Security Definitions

Types of Misuse

- Accidental
- Intentional

Protection is to prevent either accidental or intentional misuse
Security is to prevent intentional misuse

Three pieces to security

Authentication	Who user is
Authorisation	Who is allowed to do what
Enforcement	Ensure that people only do what they are allowed to do

A loophole in any of these can cause problems; as Data Scientists you will mostly deal with authentication, but you have resonsibility for all areas

Authentication

Common approach: **Passwords**.
Acts as a shared secret between two parties. Since only I know password, machine can assume it is me.
Problem 1 system must keep copy of secret, to check against password.
What if malicious user gains access to this list of passwords?
Encryption Transformation on data that is difficult to reverse - in particular, secure digest functions.

Secure Digest Functions

A secure digest function $h = H(M)$ has the following properties:

1. Given M, is is easy to compute h.
2. Given h, it is hard to compute M.
3. Given M, it is hard to compute M' such that $H(M) = H(M')$

Hashed Message Authentication Code

Can we use message digests to provide authentication?

- Provide a shared secret to both ends of communication K
- Generate an authentication code for message m $(K|m)$
- Hash the message authentication code $H(K|m)$
- Send message m and $H(K|m)$, and regenerate $H(K|m)$ at destination
- Only knowledge of the shared secret and an unchanged message could generate the authentication code
- Provides authentication and integrity
- Technically, an HMAC is $H(K \oplus opad|K \oplus ipad|m)$ where $opad$ and $ipad$ are constants designed to protect against corner case

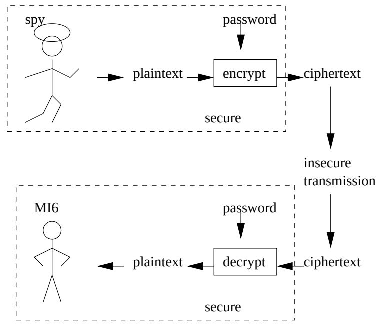
A Brief Introduction to Encryption

In distributed systems, the network between the machine on which the password is typed and the machine the password is authenticating on is accessible to everyone.
Two roles for encryption:

1. Authentication - do we share the same secret?
2. Secrecy - I don't want anyone to know this data (eg medical records)

Use an encryption algorithm that can easily be reversed given the correct key, and difficult to reverse without the key

Private Key Encryption



- From cipher text, can't decode without password.
- From plain text and cipher text, can't derive password.
- As long as the password stays secret, we get both secrecy and authentication.

Symmetric Encryption

Symmetric encryptions use exclusive ors, addition, multiplication, shifts and transpositions, all of which are fast on modern processors.

DES The Data Encryption Standard (DES) was released in 1977. The 56 bit key is too weak for most uses now, and instead, 3DES or *triple DES* is used, which has 128 bit keys.

IDEA The International Data Encryption Algorithm has 128 bit keys, and has proven strong against a large body of analysis.

AES The US based NIST defined an encryption algorithm in 2001, based on Rijndael, which offers 128, 192 or 256 bit keys.

Public Key Encryption

Public key encryption is a much slower alternative to private key; separates authentication from secrecy.
Each key is a pair K,K-1
With private key: (text)$\wedge$K$\wedge$K $=$ text
With public key:

- (text)$\wedge$K$\wedge$K-1 $=$ text, but (text)$\wedge$K$\wedge$K $\neq$ text
- (text)$\wedge$K-1$\wedge$K $=$ text, but (text)$\wedge$K-1$\wedge$K-1 $\neq$ text

Can't derive K from K-1, or vice versa

Public key directory

Idea: K is kept secret, K-1 made public, such as public directory

- For example: (I'm Ian)∧K
  Everyone can read it, but only I could have sent it (authentication).
  (Hi!)∧K-1
  Anyone can send it but only I can read it (secrecy).
  ((I'm Ian)∧K Hi!)∧K'-1
  On first glance, only I can send it, only you can read it. *What's wrong with this assumption?*

Problem: How do you trust the dictionary of public keys? Maybe somebody lied to you in giving you a key.

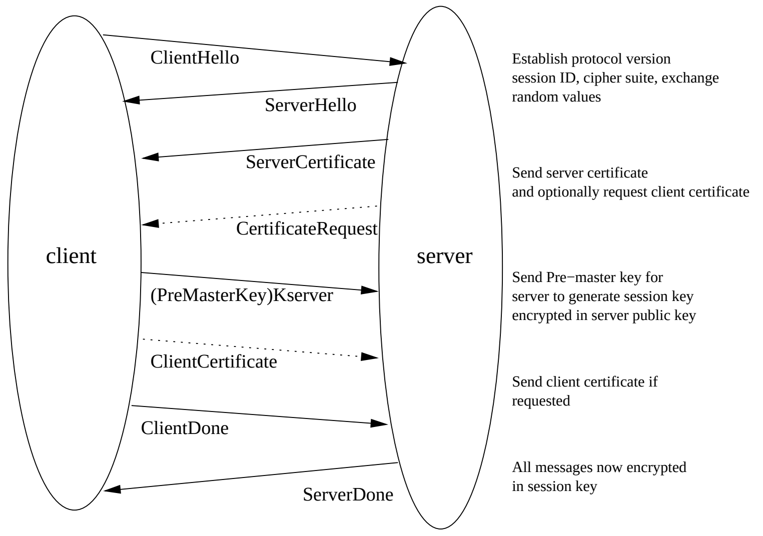
Digital Signatures and Certificates

- Given a public key K and its paired private key $K-1$, and a hash algorithm H, a digital signature over a message M is:
  $D = (H(M)) \wedge K - 1$
  Both M and D are sent
- The receiver of the message can then check $D \wedge K = H(M)$, and know that the owner of K-1 *signed* the message
- If the message is the assertion that a public key K' belongs to user U, then we have something called a *digital certificate*
- Digital certificates have been standardised, using X.509.
- Certificates can be self signed, signed by friends (PGP) or by signatories who are themselves certified by others in a trust hierarchy, eg Thawte, Verisign etc

Transport Layer Security (Secure Sockets Layer)

- Provides a techniques for data sent over a TCP connection to be encrypted.
- Uses public key technology to agree upon a key, then 3DES or whatever to encrypt the session.
- Data encrypted in blocks, optionally with compression first.
- Used in http as **https**, and for telnet, ftp etc.
- Also encountered as preferred method of authentication for SSH, as used in Amazon Web Services

SSL handshake protocol


client
server
ClientHello
ServerHello
ServerCertificate
CertificateRequest
(PreMasterKey)Kserver
ClientCertificate
ClientDone
ServerDone
Establish protocol version session ID, cipher suite, exchange random values
Send server certificate and optionally request client certificate
Send Pre−master key for server to generate session key encrypted in server public key
Send client certificate if requested
All messages now encrypted in session key

Conclusion

- Data Centres
- Synchronization by Messages
- Remote Procedure Call
- Fault Tolerant File Systems
- Security