Distributed Systems — Summary

Allan Clark

School of Informatics
University of Edinburgh

http://www.inf.ed.ac.uk/teaching/courses/ds
Autumn Term 2012

Review — Introduction

Definition of Distributed Systems

- We debated over the definition of a distributed system and decided that the distinguishing features were:
  - Independent computers
  - Coordination achieved only through message passing
- There is also the notion of transparency of distribution, that is that the distributed system should appear to the users as a single computer
  - We decided that this was a nice feature but not an essential one
- We distinguished the study of distributed systems from the study computer networks by noting:
  - Computer networking is the study of how to send messages between remote computers
  - Distributed systems is the study of how to use that capability to get work done

Review — Introduction

Challenges

- We identified several challenges involved in designing and building distributed systems:
  1. Heterogeniety
  2. Openness
  3. Security
  4. Scalability
  5. Handling of failures
  6. Concurrency
  7. Transparency

Review — Fundamental Concepts

- A distributed algorithm was defined as the steps to be taken at each process — in particular the sequence of steps taken globally is not defined
- We defined a synchronous system as one in which we have known upper and lower bounds for:
  - the time taken for each process to execute each step in the computation
  - Time taken for message delivery
  - The clock drift rate from real time for each process
- An asynchronous system has no such bounds
- We noted that all asynchronous systems could be made synchronous by assuming very large bounds
- The defining feature of a synchronous system is that the bounds were useful
- Synchronous systems allow for simpler algorithms but determining useful bounds is often difficult

Review — Fundamental Concepts

Models

- We create models of our distributed systems in order to make explicit all relevant assumptions
- Make generalisations about what is possible given those assumptions
  - The interaction model allows us to determine logical properties of the algorithm, such as termination, correctness and other properties more dependent on the application
  - The performance model allows us to improve on the abstract performance available from the interaction model by combining with performance data of the underlying machines and network mediums
  - The failure model allows us to permit (or exclude) classes of errors and reason about the effect of those errors on the operation or performance of our algorithm
  - The security model is used to assess risk of information leakage/distortion even given the use of cryptography

Review — Fundamental Concepts

Network Issues

- Latency is defined as the time it takes for data to first arrive at the destination after the send is initiated
- data transfer rate is how much data per unit of time may be transfered
- message delivery time $= \textit{latency} + \frac{\text{message length}}{\text{data transfer rate}}$
- latency affects small frequent messages which are common for distributed systems

Reliability

$$\text{Left} \longleftrightarrow 1 \longleftrightarrow 2 \longleftrightarrow 3 \longleftrightarrow 4 \longleftrightarrow 5 \longleftrightarrow \text{Right}$$

$$\text{Left} \longleftrightarrow 1 \longleftrightarrow 2 \longleftrightarrow 3 \longleftrightarrow 4 \longleftrightarrow 5 \longleftrightarrow \text{Right}$$

$$\text{Left} \longleftrightarrow 1 \longleftrightarrow 2 \longleftrightarrow 3 \longleftrightarrow 4 \longleftrightarrow 5 \longleftrightarrow \text{Right}$$

$$\text{Left} \longleftrightarrow 1 \longleftrightarrow 2 \longleftrightarrow 3 \longleftrightarrow 4 \longleftrightarrow 5 \longleftrightarrow \text{Right}$$

$$\text{Left} \longleftrightarrow 1 \longleftrightarrow 2 \longleftrightarrow 3 \longleftrightarrow 4 \longleftrightarrow 5 \longleftrightarrow \text{Right}$$

Red denotes a node at which error detection/correction occurs

- If the probability of a message getting through any channel is 0.5 then completing the trip is $0.5^6 = 0.016$
- Fortunately communication channels are generally more reliable
- $(\frac{9999}{10000})^6 = 0.9994 > \frac{999}{1000}$

UDP and TCP

- Two internet protocols provide two alternative transmission protocols for differing situations with different characteristics
- User Datagram Protocol — UDP
  - Simple and efficient message passing
  - Suffers from possible omission failures
  - Provides error detection but no error correction
- Transmission Control Protocol – TCP
  - Built on top of UDP
  - Provides a guaranteed message delivery service
  - But does so at the cost of additional messages
  - Has a higher latency as a stream must first be set up
  - Provides both error detection and correction
- IP Multicast has UDP-like failure semantics (maybe)

Review — Fundamental Concepts

Marshalling

- Marshalling is the process of flattening out a complex data structure into a series of bytes which can be sent in a message
- CORBA, Java Serialisation, XML and JSON
- Some come with instructions to the receiver on how to re-construct the flattened, others require pre-agreement on the types of the communicated data structures
- XML more general, JSON becoming popular because programmers are "fed-up" of parsing

Review — Time and Global State

Synchronising Clocks

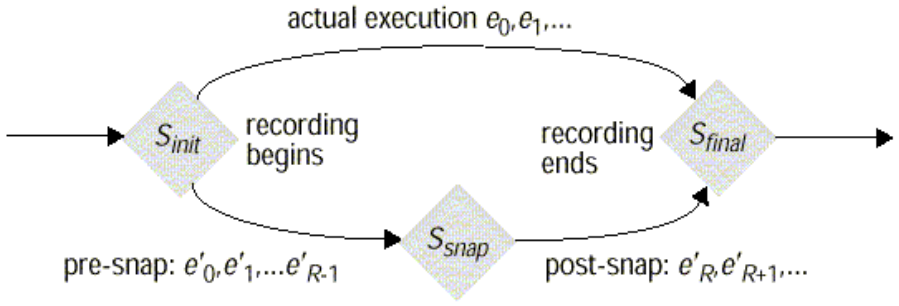
- We noted that even in the real world there is no global notion of time
- We thought that perhaps that didn't matter because we are all travelling at slow speeds relative to each other
- However our clocks are not true clocks they are mechanical and as such are subject to drift and skew
- We nevertheless described algorithms for attempting the synchronisation between remote computers
  - Cristian's method
  - The Berkely Algorithm
  - Pairwise synchronisation in NTP
- Despite these algorithms to synchronise clocks it is still impossible to determine for two arbitrary events which occurred before the other.
- We therefore looked at methods to impose a meaningful order on remote events and this took us to logical orderings

Review — Time and Global State

Logical Orderings and States

- Logical orderings based on the intuitive and simple idea of the "happens-before" relation:
  - $e_1 \rightarrow e_2$ if e_1 and e_2 occur at the same process and e_1 occurs before e_2, or:
  - e_1 is the sending of some message and e_2 is the receiving of that same message, or:
  - There exists some event $e_{1.5}$ such that: $e_1 \rightarrow e_{1.5} \rightarrow e_2$
- Lamport and Vector clocks were introduced:
  - Lamport clocks are relatively lightweight provide us with the following $e_1 \rightarrow e_2 \implies L(e_1) < L(e_2)$
  - Vector clocks improve on this by additionally providing the reverse implication $V(e_1) < V(e_2) \implies e_1 \rightarrow e_2$
  - Meaning we can entirely determine whether $e_1 \rightarrow e_2$ or $e_2 \rightarrow e_1$ or the two events are concurrent.
  - But do so at the cost of message length and scalability
- The concept of a true history of events as opposed to *runs* and *linearisations* was introduced

Global State — Chandy and Lamport — Reachability



- We looked at Chandy and Lamport's algorithm for recording a global snapshot of the system
- Crucially we defined a notion of reachability such that the snapshot algorithm could be usefully deployed in ascerting whether some stable property has become true.

Review — Time and Global State

Distributed Debugging

- Finally the use of consistent cuts and linearisations was used in Marzullo and Neiger's algorithm
- Used in the debugging of distributed systems it allows us to ascertain whether some transient property was possibly true at some point or definitely true at some point.
- Suppose we have a monitor M and two processes P_1 and P_2
  - We start with $P_1(x = 100)$ and $P_2(y = 50)$
  - M receives a message from $P_1, x = 50$
  - M receives a message from $P_2, y = 100$
  - The monitor then records four global states:
    1. $x = 100, y = 50$
    2. $x = 50, y = 50$
    3. $x = 100, y = 100$
    4. $x = 50, y = 100$
  - Both states 2 and 3 could not have occurred, but we do not know which occurred
  - If we wish to know whether the sum was ever 200 then we say "possibly" but not "definitely"

Review — Time and Global State

Distributed Debugging

- The purpose of a snapshot algorithm was to record a global state that is logically consistent with some state which was actually experienced, but there is no attempt to record a state which was "actually experienced"
- Distributed debugging on the other hand hopes to record such "actually experienced" states, and is conservative in the sense that it considers more states than may actually have occurred

Review — Coordination and Agreement

Mutual Exclusion and Election

- We looked at the problem of Mutual Exclusion in a distributed system
  - Giving four algorithms:
    1. Central server algorithm
    2. Ring-based algorithm
    3. Ricart and Agrawala's algorithm
    4. Maekawa's voting algorithm
  - Each had different characteristics for:
    1. Performance, in terms of bandwidth and time
    2. Guarantees, largely the difficulty of providing the *Fairness* property
    3. Tolerance to process crashes
- We then looked at two algorithms for electing a master or nominee process; ring-based and bully algorithms
- Then we looked at providing multicast with a variety of guarantees in terms of delivery and delivery order

Review — Coordination and Agreement

General Consensus

- We then noted that these were all specialised versions of the more general case of obtaining consensus
- We defined three general cases for consensus which could be used for the above three problems
- We noted that a synchronous system can make some guarantee about reaching consensus in the existance of a limited number of process failures
- But that even a single process failure limits our ability to guarantee reaching consensus in an asynchronous system
- In reality we live with this impossibility and try to figure out ways to minimise the damage

Review — Distribution and Operating Systems

Operating System Characterisations

- Distributed Operating Systems are an ideal allowing processes to be migrated to the physical machine more suitable to run it
- However, Network Operating Systems are the dominant approach, possibly more due to human tendancies than technical merit
- We looked at microkernels and monolithic kernels and noted that despite several advantages true microkernels were not in much use
- This was mostly due to the performance overheads of communication between operating system services and the kernel
- Hence a hybrid approach was common

Review —- Distribution and Operating Systems

Concurrency, processes and threads

- We looked at processes and how they provide concurrency, in particular because such an application requires concurrency because messages can be received at any time and requests take time to complete, time that is best spent doing something useful
- but noted that separate processes were frequently ill-suited for an application communicating within a distributed system
- Hence threads became the mode of concurrency offering lightweight concurrency.
- Multiple threads in the same process share an execution environment and can therefore communicate more efficiently and the operating system can switch between them more efficiently

Review — Distribution and Operating Systems

Operating System Costs and Virtualisation

- We also looked at the costs of operating system services on remote invocation
- Noting that it is a large factor and any design of a distributed system must take that into account — in particular the choice of protocol is crucial to alleviate as much overhead as possible
- Finally we looked at system virtualisation and noted that it is becoming the common-place approach to providing cloud-based services
- Virtualisation also offers some of the advantages of a microkernel including increased protection from other users' processes

Review — Peer-to-Peer Systems

Motivations and Napster

- We began with looking at the motivations behind the development of peer-to-peer systems
  - Break the reliance of the system on a central server which may be vulnerable from attack, both technical and bureaucratic
  - Utilising the resources of those using the service such that capacity grows with the number of users
  - Providing anonymity to content providers
- The now defunct pioneering system *Napster*
  - Napster relied on a central server, but that server hosted no content, bandwidth to the central server was limited as well because no content was therefore downloaded from the central server
  - Instead the central server was merely used by remote peers to locate content and setup independent connections between peers
  - Ultimately though the reliance on a central server proved enough fodder for the entertainment industry's lawyers and Napster was shutdown

Review — Peer-to-Peer Systems

Peer-to-Peer Frameworks

- Napster however proved the feasibility of the concept and several services grew into the space left behind by Napster
- Such services do not rely on any single central server and have so far proved resilient to legal attacks
- However we focused our attention on efforts to provide a generic framework for building peer-to-peer applications
- Such frameworks currently focus on providing a distributed hash table, storing objects and replicas at multiple peers for later retrieval
- Distributed Object Location and Routing systems are an extension providing a more convenient API, in particular for objects which may be updated

Review — Peer-to-Peer Systems

Structured vs Unstructured

- Two related problems for the use of a peer-to-peer system:
  1. initially finding the resource you are interested in and thus obtaining its logical address (GUID)
  2. Routing to the logical address (GUID) once it is known
- Analogous to internet addresses which first translate the text url into an integer address and then route to that address
- For internet addresses it is efficient to do this in two stages, because once you have the integer address you can access the resource more efficiently and you may do this many times
- For file-sharing, once the file is found, that likely constitutes the one and only time that that particular user will access that particular resource
- Hence relying on unstructured search is reasonable
- Even the search is less structured than domain name lookup since domain names are an exact one-to-one mapping, file searches are not

Review — Security

- Although we noted that human error is a large cause of security breaches our concern here was technical security, which was mostly achieved through the use of cryptography
- Our assumption is that the network, atop which our distributed system is constructed, is insecure. Messages may be, deleted, read, duplicated, modified and inserted
- A man-in-the-middle attack is one in which the attacker makes independent connections with two victims and relays the messages between them.
- You can apply this to beat a master in a blind game of Chess (or Go, etc)
  - Set up two games against two masters making sure you are black in one and white in the other
  - Mirror each players moves to the opposite board
  - You will win one game and lose the other
- Usually though the man-in-the-middle masquerades as each of the two

Review — Security

Cryptography

- Modern cryptography makes use of algorithms which distort a message such that it is difficult/infeasible to recover the original message without knowledge of the key
- Shared secret-key algorithms are symmetric and make use of the same key to both encrypt and decrypt the message
- A message is secure provided that no one else knows/discovers the shared secret key
- Such that: $D(K, E(K, M)) = M$
- Public/private key algorithms are not symmetric. One key is used to encrypt the message whilst a corresponding key is used to decrypt the encrypted message:
- If K_e and K_d are a key-pair: $D(K_d, E(K_e, M)) = M$
- Generally a person publishes their public key and anyone can send a secure message to them by encrypting it with the public key

Review — Security

Hybrid Protocols

- If Alice sends a message to bob $\{M\}_{K_{Bpub}} = E(K_{Bpub}, M)$
- Bob can decrypt this $M = D(K_{Bpriv}, \{M\}_{K_{Bpub}}$, but to reply with M' Bob must use Alice's public key: $\{M'\}_{K_{Apub}} = E(K_{Apub}, M')$
- Alice can decrypt this with her private key: $M' = D(K_{Apriv}, \{M'\}_{K_{Apub}})$
- This has the attractive advantage that no pre-agreed shared secret is required. Alice and Bob can be on opposite sides of the world and still communicate in secret without risk that the sharing of a shared secret key was eavesdropped
- However public key encryption algorithms are 100/1000 times slower than shared secret key encryption, if Alice and Bob are to have a prolonged communication this is slow
- Alternatively M could have been a new shared secret key
- This uses public-key crytography to set up shared secret-key cryptography giving the benefits of both kinds

Security — Review

Digital Signature

- Rather than encrypt a message with Bob's public key Alice can instead encrypt a message with her own private key
- Doing so means that to decrypt the message requires Alice's public key which is generally available, so the message is insecure
- However, since a message decrypted with Alice's public key must have been encrypted with Alice's secret key we know for sure that Alice must have encrypted (and sent) the message
- So as to avoid encrypting an entire document, Alice may compute a digest (similar to a checksum) of the document and encrypt the digest and attach it to the document
- Digital signatures fulfil well the properties of Authentication and Unforgeable but can fail to be Non-repudiable

Review — Security

Certificates

- All public-key cryptography, including digital signatures, suffer from the problem of authenticating a public key
- That is, being sure that the public key, or the entity providing the public key, really is that of the entity advertised
  - I can claim to be Microsoft
  - Just as easily I can give you a public key, claim that that key is Microsoft's public key, and that I am therefore Microsoft
- Certificates are essentially digital signatures attached to public keys (or other digital signatures)
- Of course certificates may also be falsely claimed in the same way however one "*certification authority*" may certify many public-keys/digital signatures, such that the receiver need only trust the *certification authority* to enable trust of many others
  - For example `https` signatures

Review — Security

Key Iteration

- If the attacker knows a plaintext/ciphertext pair (M/M_e), it can simply try all possible keys K_{poss} until $M_e = E(K_{poss}, M)$
- This represents a problem for public key cryptography since the attacker can generate as many plaintext/ciphertext pairs as they require
- It means that keys must be long enough
- In addition the message must be long enough, suppose the message was just two bytes long, there are only 65,536 possible messages, and the attack knows the encryption key (ie. the public key)
  - The attacker can therefore simply iterate over all possible messages, encrypting them with public key to see if they get the same ciphertext
  - Suppose an encrypted message M_e is heard by the attacker, which they know was encrypted with the public key K_p
  - The attacker simply tries all possible M' until $E(K_p, M') = M_e$

Any Questions

End of the Course!
Thank you for your Attention!
Good luck with your Exams!
Any Questions?