CIS 700/003: Distributed Systems meet Social Networks

Rational behavior

February 16, 2010
Where are we?

- Measurement and analysis of MAD systems ✔
- Building MAD systems ✔
- Faults and Misbehavior
  - Rational behavior ✔ Today
  - Byzantine behavior
  - Defenses
- Internet crime
- Privacy and confidentiality
- Novel opportunities
- Experience
My discussion points

- BitTorrent: Apparent simplicity vs. actual complexity
- Utility function: Realistic or not?
- Rational model: Pros and cons
- How many users could actually cheat in this way?
- Other approaches, e.g., currency
- "Upload costs can therefore be extremely complex"
- Role of the system designer
- Definition of fairness
Reasons for choosing these papers

- **Paxos paper**
  - Introduces consensus problem, state machine replication
    - Two very fundamental, widely used techniques in distributed systems
    - Good basis for understanding the PBFT paper

- **BitTorrent is an auction**
  - Opportunity to discuss the rational model
  - Shows how concepts from economics (auctions!) can be useful in designing and understanding MAD systems
  - Demonstrates how complex protocol analysis can be - even for such a simple protocol!
Recap: The rational model

- Assumption: Each participant chooses his actions to **maximize** his own utility
  - Example: Participant wants to download all the pieces while sending as few bytes as possible
  - Participants modify software if relevant action is not offered
  - Rational vs selfish

- What does this buy us?
  - More realistic than assuming everyone is altruistic (i.e., does what the protocol tells them to do) → Systems built for rational nodes tend to be **more robust** in practice
  - Model is simple enough to treat **analytically**
  - Can apply **rich literature** on game theory, economics, ...
### Rational model: Theory and practice

<table>
<thead>
<tr>
<th>Model</th>
<th>Reality</th>
</tr>
</thead>
</table>
| Utility functions of all participants are known | Some aspects of a participant's utility may not be fully known  
Example: Script kiddie derives pleasure from gaming the system, doesn't care about bandwidth |
| Each participant is able to determine his optimal action | Participants may be ignorant or misinformed (bounded rationality)  
Example: Participant is not aware that he can underreport pieces |
| Each participant chooses the actions to maximize his utility | Participants may be lazy or make mistakes  
Example: Participant misconfigures his client, accidentally uploads at twice the intended rate |

- In practice, not all participants act rationally and/or selfishly all the time
  - But it often is a good approximation
Protocol analysis is difficult

- BitTorrent is a simple protocol, and yet it has proven to be surprisingly hard to analyze

- A small selection of BitTorrent papers:
  - 2003: "Incentives build robustness in BitTorrent"
    Cohen, IPTPS 2003
  - 2004: "Modeling and performance analysis of BitTorrent-like peer-to-peer networks"
    Qiu and Srikant, SIGCOMM 2004
  - 2006: "Rarest first and choke algorithms are enough"
    Legout et al., IMC 2006
  - 2006: "Exploiting BitTorrent for fun (but not profit)"
    Liogkas et al., IPTPS 2006
  - 2006: "Free riding in BitTorrent is cheap"
    Locher et al., HotNets 2006 (BitThief)
  - 2007: "Do incentives build robustness in BitTorrent?"
    Piatek et al., NSDI 2007 (BitTyrant)
  - 2008: "BitTorrent is an auction"
    Levin et al., SIGCOMM 2008
The part-time parliament

Leslie Lamport
Why should we care?

- Some services are so important that a failure or downtime would be a disaster
  - Examples: Central synchronization service for Google

- For such a service, even the best individual machine may not be reliable enough!
  - Idea: Set of machines implements the service collectively
  - Result: Service is available as long as a certain fraction of the machines are working
Replicated service

How does this work?

- Client sends its request to each of the machines
- The machines coordinate and each return a result
- Client chooses one of the results, e.g., the one that is returned by the largest number of machines
- If a small fraction of the machines returns the wrong result, or no result at all, they are 'outvoted' by the other machines

© 2010 Andreas Haeberlen

University of Pennsylvania
Challenges

- Faults must not be correlated
  - Otherwise, all machines may fail at the same time
  - Challenge: Bugs, power failures, misconfiguration, ...

- Each of the machines must process the requests in the same way
  - Otherwise, their state will diverge → Not obvious what the 'correct' result should be
  - Idea: Machines can implement a deterministic state machine!

- All machines must process the requests in the same order
  - Otherwise, their state may diverge
  - Idea: Need consensus on the order in which to process
Consensus

- **Intuition:** Each process may propose a value, and then the processes agree on which value they want to use

- **Formally, a solution must satisfy the following:**
  - **Termination:** Every correct process eventually decides
  - **Validity:** If all processes propose the same value \( v \), then every correct process decides \( v \)
  - **Integrity:** Every correct process decides at most one value, and it can only decide values that have been proposed
  - **Agreement:** If some correct process decides \( v \), then every other correct process also decides \( v \)
FLP: Consensus is "impossible"!

- No asynchronous algorithm for agreeing on a one-bit value can guarantee that it will terminate in the presence of crash faults
  - Even if no crash faults actually occur

- What now?
  - Change the problem statement: Randomized algorithms, approximate agreement, k-set agreement, ...
  - Change the assumptions: Assume bounds on message delays, or that we have an unreliable oracle (failure detector) that tells us when a node crashed
  - Paxos: Guarantees safety, but not liveness

The Paxos algorithm

- Scenario:
  - There is a replicated append-only log (the ledgers)
  - Nodes can propose a new entry to append, and they can accept proposals made by other nodes
    - Each proposal has a version number
  - If multiple proposals are made concurrently, some nodes may even accept more than one proposal
    - But, if a node has accepted a proposal with version number n, it will not accept any further proposals with version numbers smaller than n
  - A proposal accepted by a majority of the nodes is passed

- Note: Each proposal is for a specific entry
  - Multiple instances of the protocol can be active concurrently
  - Example: Propose XYZ as entry 12 and ABC as entry 13
Model

- **Network:**
  - May lose messages (messenger leaves forever)
  - May duplicate messages
  - Asynchronous (messages can be delayed arbitrarily)
  - But: No message corruption

- **Nodes:**
  - Can fail by crashing (legislator leaves the Chamber)
  - No central clock (hourglass timers)
  - But: Have some persistent memory (ledgers)
  - But: Strictly follow the protocol - no lying, data corruption...
Phase 1: Prepare

- Suppose a node A wants to propose a value X for entry n:
  - Node A chooses a new version number v
  - Node A sends PREPARE(n, v) to a majority of the other nodes

- Intuition: PREPARE(n, v) means
  - May I make a proposal for entry n with version number v?
  - If so, can you suggest a value I should use?

- Note: Fairness is not a goal; even though A is suggesting X, it is happy with other values too
Phase 1: Prepare

- If a node B receives PREPARE(n, v) from A:
  - If B has already acknowledged a PREPARE(n, v') with v'>v, then it does nothing
  - If B has previously accepted any proposals for entry n, it responds with ACK(n, v, v', X'), where v' is the highest version number of any proposal it has accepted for entry n, and X is the corresponding value
  - Otherwise, B responds with ACK(n, v, -, -)

- Intuition: An ACK means
  - Yes, go ahead and make your proposal
  - You should choose value X' (if v' and X' are given), or: any value is fine with me (if v' and X' are not given)
  - I won't accept any further proposals with version numbers <v
Phase 2: Accept

- If A receives ACKs from a majority of the other nodes, it issues $\text{ACCEPT}(n, v, X'')$
  - $X''$ is the value from the ACK with the highest version number, or the original $X$ if none of the ACKs had a value.

- If B receives $\text{ACCEPT}(n, v, X'')$
  - If B has already responded to a $\text{PREPARE}(n, v')$ with $v' > v$, then B does nothing.
  - Otherwise B accepts the proposal and sends $\text{ACCEPT}(n, v, X'')$ to all the learners.

- If a learner L receives $\text{ACCEPT}(n, v, X'')$ from a majority of the acceptors, it decides $X''$
  - L then sends $\text{DECIDE}(n, X'')$ to all the other learners.
  - If another learner receives $\text{DECIDE}(n, X'')$, it decides $X''$.
Example

A's quorum

B's quorum

A: PREPARE(3,5)

C,D,E,G,H: ACK(3,5,-,-)

A: ACCEPT(3,5,X)

B: PREPARE(3,8)

D,F,G,I: ACK(3,8,-,-)

E: ACK(3,8,5,X)

B: ACCEPT(3,8,X)

D,...,I: ACCEPT(3,8,X)

C: DECIDE(3,X)
Recap: Paxos

- **Goal: Build a replicated service**
  - Multiple machines acting 'as if' they were a single machine
  - Can mask faults if not too many happen simultaneously

- **Paxos implements an important building block:**
  A consistent append-only log
  - Useful to make the replicas agree on the order in which to process requests → prevent divergence
  - More generally, consensus is useful in many other scenarios

- **But: Paxos assumes crash faults**
  - Malicious nodes can easily disrupt the algorithm by telling lies
  - Can we build a similar protocol that can tolerate malicious nodes as well?
Where are we?

- Measurement and analysis of MAD systems ✔
- Building MAD systems ✔
- Faults and Misbehavior
  - Rational behavior ✔
  - Byzantine behavior ❌ Next time
  - Defenses
- Internet crime
- Privacy and confidentiality
- Novel opportunities
- Experience
Stay tuned!

Next week you will learn:

How the old Roman Empire became a threat to modern distributed systems