COSC441

Distributed Programming

Martelli Model of Scalability

- 1 core: single thread and single process
- 2-8 cores: multiple threads and multiple processes
- >8 cores: distributed processing
- Alex Martelli (a Pythonista). Claims that as time goes on, the multi-thread approach loses relevance.
- TILE64, 2007, 64 cores. I have a Parallela 18-core “card” @ USD99, they have a 66-core machine @ USD750, and have made a 1024-core chip.
- But eventually distributed is the way to go.

Shared-memory concurrency

- Data structures stay put in memory
- Threads communicate through shared data structures
- and perhaps by signals/events
- Access is controlled by locks
- Locking is expensive, so managing communication is important
- There is a common notion of time, more or less

Distributed programming

- Computations take place on many machines
- Shared memory either does not exist or is simulated (OpenSHMEM)
- Processes communicate by exchanging messages
- Data are copied through messages
- Communication is expensive
- There is no global time

Issues with messages

- Latency, bandwidth, capacity (last week)
- In-order *vs* out-of-order
- Delivery
- Reliability
- Security
- Network and process failure
- Time

Ordering

- Process A sends message x to process B
- Process A sends message y to process B
- Which message does B see first?
- Why might that happen?
- What can we do about that?
  - sequence numbers
  - buffering
  - retransmission

Delivery

- Process A sends message x to process B
- How many copies of x does B get?
  - *at most once*
  - *at least once*
  - *exactly once*
- How can we deal with these?
- Synchronous *vs* asynchronous, ACK/NAK.

Reliability

- Messages may be corrupted
- Ethernet packets have a 32-bit CRC check
- IPv4 packets have a 16-bit *header* checksum
- TCP has the same kind of 16-bit checksum but includes the payload as well as the header
- IPv6 omits the IP header checksum
- Data may be corrupted at the hardware level (electrical noise) or the software level (broken code in the network stack).
-

Security

- Eavesdroppers may capture data
- They may replay data later
- They may intercept and change data
- Not so much an issue within a data centre (LAN) but definitely an issue at wider scale (WAN).

Failure

- Communication channels may fail (backhoes and squirrels)
- Power may be lost (Delta installing new power poles disconnected power to my house this morning)
- Any layer of software may be buggy
- Is wire cut? Is machine dead? Did OS crash? Is user-level process dead? Can be hard to tell.

Byzantine Failure

- Nonresponsiveness is not too hard to deal with
- The Byzantine Generals problem
- *Malicious* failure: hardware + software look OK but are acting malevolently.
- Think: links in messages from spammers

Time

- There is no such thing as NOW.
- Distributed programming lives in a relativistic world.
- Processing elements communicate through messages which take varying amounts of time to arrive.
- You *never* know what is happening elsewhere NOW.

Happens-before/Lamport clocks

- There is a *partial order* on events.
- If event X precedes event Y at the same place, X happens before Y.
- Sending a message happens before receiving it.
- The partial order is generated by those.
- Any partial order can be completed to a total order, but it's not metric.