Lecture 31: Concurrency and Parallelism (I)

Announcement: HKN (Eta Kappa Nu) will be holding a review session for the upcoming midterm this Sunday, 4/8, from 1100-1300 in 306 Soda Hall (HP Auditorium). Bring yourselves, some pencil and paper, and any questions that you may have.

Definitions

• Sequential Process: Our subject matter up to now: processes that (ultimately) proceed in a single sequence of primitive steps.
• Concurrent Processing: The logical or physical division of a process into multiple sequential processes.
• Parallel Processing: A variety of concurrent processing characterized by the simultaneous execution of sequential processes.
• Distributed Processing: A variety of concurrent processing in which the individual processes are physically separated (often using heterogeneous platforms) and communicate through some network structure.

Purposes

We may divide a single program into multiple programs for various reasons:

• Computation Speed through operating on separate parts of a problem simultaneously, or through
• Communication Speed through putting parts of a computation near the various data they use.
• Reliability through having multiple physical copies of processing or data.
• Security through separating sensitive data from untrustworthy users or processors of data.
• Better Program Structure through decomposition of a program into logically separate processes.
• Resource Sharing through separation of a component that can serve multiple users.
• Manageability through separation (and sharing) of components that may need frequent updates or complex configuration.

Communicating Sequential Processes

• All forms of concurrent computation can be considered instances of communicating sequential processes.
• That is, a bunch of "ordinary" programs that communicate with each other through what is, from their point of view, input and output operations.
• Sometimes the actual communication medium is shared memory: input looks like reading a variable and output looks like writing a variable. In both cases, the variable is in memory accessed by multiple computers.
• At other times, communication can involve I/O over a network such as the Internet.
• In principle, either underlying mechanism can be made to look like access to variables or explicit I/O operations to a programmer.

Distributed Communication

• With sequential programming, we don’t think much about the cost of “communicating” with a variable; it happens at some fixed speed that is (we hope) related to the processing speed of our system.
• With distributed computing, the architecture of communication becomes important.
• In particular, costs can become uncertain or heterogeneous:
  - It may take longer for one pair of components to communicate than for another, or
  - The communication time may be unpredictable or load-dependent.

Simple Client-Server Models

• Example: web servers
• Good for providing a service
• Many clients, one server
• Easy server maintenance.
• Single point of failure
• Problems with scaling
Variations: on to the cloud

- Google and other providers modify this model with redundancy in many ways.
- For example, DNS load balancing (DNS = Domain Name System) allows us to specify multiple servers.
- Requests from clients go to different servers that all have copies of relevant information.
- Put enough servers in one place, you have a server farm. Put servers in lots of places, and we have a cloud.

Communication Protocols

- One characteristic of modern distributed systems is that they are conglomerations of products from many sources.
- Web browsers are a kind of universal client, but there are numerous kinds of browsers and many potential servers (and clouds of servers).
- So there must be some agreement on how they talk to each other.
- The IP Protocol is an agreement for specifying destinations, packaging messages, and delivering those messages.
- On top of this, the transmission control protocol (TCP) handles issues like persistent telephone-like connections and congestion control.
- The DNS handles conversions between names (inst.eecs.berkeley.edu) and IP addresses (128.32.42.199).
- The HyperText Transfer Protocol handles transfer of requests and responses from web servers.

Example: HTTP

- When you click on a link, such as http://inst.eecs.berkeley.edu/~cs61a/lectures, your browser:
  - Consults the DNS to find out where to look for inst.eecs.berkeley.edu.
  - Sends a message to port 80 at that address:
    GET `cs61a/lectures HTTP 1.1
  - The program listening there (the web server) then responds with
    HTTP/1.1 200 OK
    Content-Type: text/html
    Content-Length: 1354
    <html> ... text of web page
  - Protocol has other messages: for example, POST is often used to send data in forms from your browser. The data follows the POST message and other headers.

Peer-to-Peer Communication

- No central point of failure; clients talk to each other.
- Can route around network failures.
- Computation and memory shared.
- Can grow or shrink as needed.
- Used for file-sharing applications, botnets (!).
- But, deciding routes, avoiding congestion, can be tricky.
- (E.g., Simple scheme, broadcasting all communications to everyone, requires $N^2$ communication resource. Not practical.
- Maintaining consistency of copies requires work.
- Security issues.

Clustering

- A peer-to-peer network of "supernodes," each serving as a server for a bunch of clients.
- Allows scaling: could be nested to more levels.
- Examples: Skype, network time service.