CS 686: Special Topics in Big Data

Parallel Computing

Lecture 16

Today's Agenda

- Revisiting Jepsen
- Supercomputing
- Message Passing
- Remote Procedure Calls
- Grid Computing
- Distributed Applications

Today's Agenda

- **Revisiting Jepsen**
- Supercomputing
- Message Passing
- Remote Procedure Calls
- Grid Computing
- Distributed Applications

Jepsen: Hazelcast

- Related to our recent consistency discussions: recent article in the *Jepsen* series on Hazelcast
- Hazelcast is a distributed in-memory data grid
  - Provides shared data structures for coordination and synchronization
    - **Somewhat** like Chubby, ZooKeeper
- https://jepsen.io/analyses/hazelcast-3-8-3

Jepsen: Hazelcast Highlights

- Locks that don't lock!
  - (under a network partition)
- Unique IDs that aren't unique!
- 500 second waits for the cluster to repair itself
- "Finally, almost all uses of lock services for safety in distributed systems are fundamentally flawed: users continue to interpret distributed locks as if they were equivalent to single-node mutexes"
  - Lock services cannot guarantee exclusion
- I hope they never test any of my software!

Today's Agenda

- Revisiting Jepsen
- **Supercomputing**
- Message Passing
- Remote Procedure Calls
- Grid Computing
- Distributed Applications

Supercomputing

- Pioneered by Seymour Cray, ~1960s
- Observed that simply making the CPU much faster wasn't all that beneficial
  - Still have to wait for other components to catch up
  - (Assuming the CPU drives everything)
- Instead, Cray designed a system that linked 10 simple computers
  - Each of the 10 PPUs were responsible for shuffling data in and out of memory

CDC 6600

CDC 6600: Tech Specs

- 60-bit CPU, ten 12-bit I/O processors
- 3 megaFLOPS
- Memory: 128K 60-bit words
- Dual video display console
  - Pretty cool: vector system instead of raster
- Storage: 2 MB
  - Could add magnetic drum storage for expansion!
- Yours for ~$10m

System Design

- Original supercomputers used custom hardware to accelerate performance and allow parallelism
- Over time, more off-the-shelf components were used instead
  - Huge leaps in performance of commodity CPUs
- There are still some advantages over a standard cluster:
  - Better interconnects (e.g., Infiniband)
  - Better integration

Top500

- A list of the top 500 supercomputers is available at: https://www.top500.org
- Current #1: China's Sunway TaihuLight
  - 93 PetaFLOPS
- #2 is at 33.9 PFLOPS
- Some of these machines have **millions** of cores
- See also: Green500 https://www.top500.org/green500/

Beowulf Clusters

- Over the years, many big computing tasks have migrated away from supercomputing platforms
  - At the same time, supercomputers look more and more like clusters
- "Beowulf" terminology coined in 1994 @ NASA
- Grab a bunch of commodity PCs, install software like OpenMPI, MPICH
  - "Supercomputer" on the cheap!

Today's Agenda

- Revisiting Jepsen
- Supercomputing
- **Message Passing**
- Remote Procedure Calls
- Grid Computing
- Distributed Applications

Message Passing

- The basic idea behind parallelism is **divide and conquer**
- To do this, we need to coordinate across processing units in our cluster via **messages**
- We could use **sockets**
  - Who even does that? (Apart from 686 students)
  - Wrong level of abstraction for high performance computing (HPC) applications
  - Every cluster/supercomputer is different

Message Passing Interface

- Message passing is the most common paradigm for programming distributed memory systems
- Processors coordinate their activities by sending messages to each other across the network
  - Infiniband
  - Ethernet
- Message Passing Interface, or just **MPI**, gives us communication primitives to do this

MPI Standard

- There are multiple implementations of MPI that target a single standard
- This allows hardware-specific optimizations: your Cray supercomputer probably ships with its own special version of MPI
  - Knows about the structure of the communication interconnects
- This leads to better performance but also compatibility issues and the usual arguments over the spec

A Simple MPI Program

- With MPI, you write one program and then run it in parallel across multiple PUs
  - PUs can distinguish between one another by their **ranks** (identifiers) – nicer than IP addresses!
- Point to point and collective communication are supported
- This approach **does not** consider network failures
  - Not great if you're operating at Google's scale!

MPI Use Cases

- MPI is great for coordinating supercomputing/HPC jobs
- Used extensively for atmospheric modeling, simulations, etc.
- Servicing web requests, working with failures… not so much.

Today's Agenda

- Revisiting Jepsen
- Supercomputing
- Message Passing
- **Remote Procedure Calls**
- Grid Computing
- Distributed Applications

Remote Procedure Call (RPC)

- Simple building block for distributed systems: allow programs to execute **procedures** on other machines
- Execute your code transparently in another system's address space
  - Works just like a local procedure call
- Awesome! Makes writing a distributed application almost invisible to the developer

RPC Implementation

- Under the hood, RPC uses message passing
- **Client** asks **server** to execute a method, and waits for a response
  - Can be either blocking or non-blocking
  - For non-blocking RPCs, we can use **futures** to serve as a placeholder for the result
- Unlike a local call, RPCs may fail or incur much more variable latencies
  - So, maybe not 100% transparent to the developer…

Remote Method Invocation

- A (once) very popular RPC API for Java
- Hides even more complexity from the developer
- Great for the world of the 90s and early 00s, but has some drawbacks:
  - Introduces additional context switching
  - Difficult to deal with heterogeneity in hardware
  - Java only! This is the big one
    - Nowadays we need to communicate across programs transparently

RPCs in General

- RPCs are great for shuffling data around and synching up distributed components
- You can likely achieve better performance and build a more robust system with simple message passing
  - The cost? More time spent dealing with low-level details
- How do we deal with concurrency? Going several calls deep? (Machine A → Machine B → Machine C)

Non-Blocking Event Loop

- An alternative approach to using RPC: **events**
- Rather than method calls, just send an event to the remote machine
  - Asynchronous by design
  - Encourages stateless communications
- On the server side, process incoming events in an **event loop**
  - node.js

Event Loop Design

- One thread (or maybe a few) process incoming message packets
  - Does not block waiting for messages
  - A bit of data came in? Throw it in a buffer and move on
- Unwrap the packets, deserialize into an event, and place the into a work queue
- The rest of the threads handle events
  - Not so great for frequent back-and-forth comm.

REST

- Representational state transfer (REST) is closely related to event-based systems
- Generally operates over HTTP endpoints
- GET, POST, etc. to URIs
- **Stateless** data transfer
- Can use XML, HTML, but JSON is most common

Today's Agenda

- Revisiting Jepsen
- Supercomputing
- Message Passing
- Remote Procedure Calls
- **Grid Computing**
- Distributed Applications

Grid Computing

- Thus far we've focused on communications and data transfer
- **Grid** computing aims to provide processing resources at scale
- Modeled after the electric grid: use resources as you need them
  - Better utilization of hardware between organizations (for instance, universities)

Hardware

- Grids are “super virtual computers” created by combining a large amount of machines
- Connected by commodity/standard networking hardware such as Ethernet
  - May span large geographic regions
- Like a traditional cluster but span across organizations
- Loosely coupled

Making it Work

- This may sound like a recipe for disaster!
  - Extreme heterogeneity
- However, **grid middleware** helps handle the heavy lifting for us
- When launching an application on a grid, we can specify type of resources we need
  - Software libraries, architectures, particular hardware features, etc.

Volunteer Computing

- **Volunteer computing** is one way to create a grid
- Connect to the grid, use it, and also volunteer your own resources when you don't need them
- Leads to better all-around utilization
- **But**, many grid technologies were designed back when workstations/servers ran 24/7
  - These days, maybe it's better to shut our PCs down (or sleep!) when we're not using them

Cycle Scavenging

- Cycle scavenging is another form of volunteer computing
- SETI@Home, Folding@Home
- Install a special screensaver that looks for extraterrestrial intelligence while you're making coffee!
- And: "involuntary" cycle scavenging
  - I should be using the lab machines here to mine bitcoin, right?!

Cloud Computing

- Then cloud computing (*ahem* Amazon) came along…
- Realizes many goals of the grid computing movement
  - Also makes many of the same mistakes
    - Talk to a grid computing researcher sometime
- Better: **elasticity**
  - Expand and contract your resource pool as needed

Today's Agenda

- Revisiting Jepsen
- Supercomputing
- Message Passing
- Remote Procedure Calls
- Grid Computing
- **Distributed Applications**

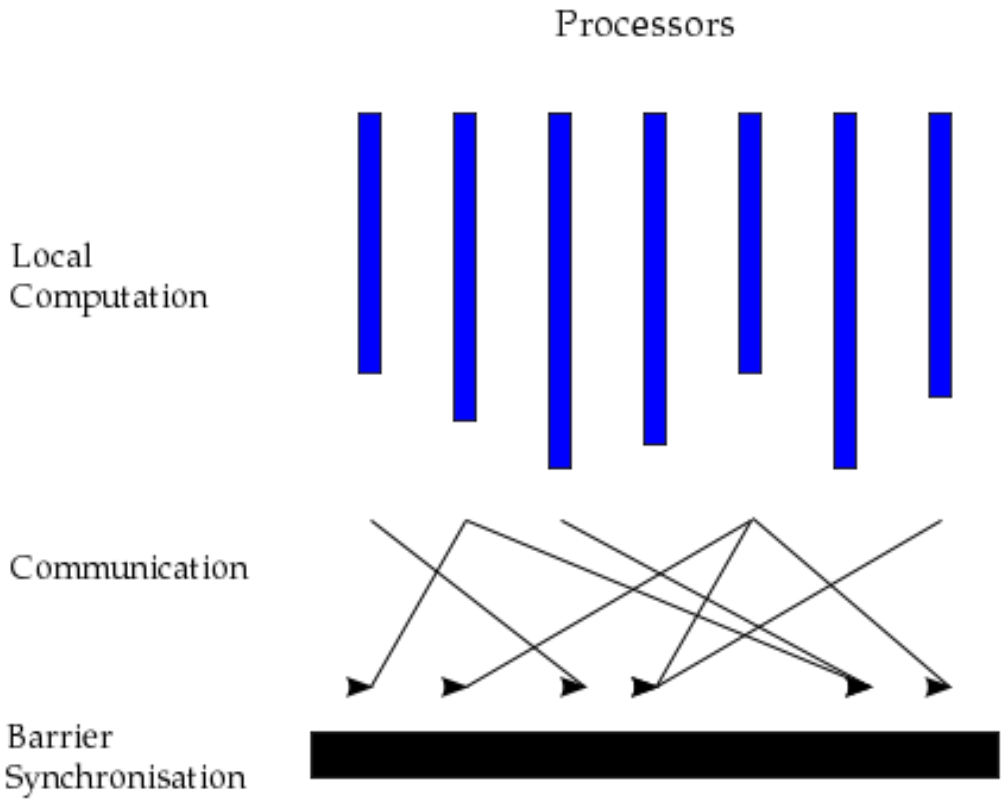
Distributed Applications

- So, we can:
  - Communicate (pass messages)
  - Run processes (or procedures) on other machines
  - Share resources
- But this all has a very traditional **process centric** view of computing
- Why not target an execution model that was designed to be distributed in the first place?

Bulk Synchronous Parallel

- Computing paradigm that consists of:
  - Threads
  - Network communication
  - Synchronization
- Somewhat of a precursor to **MapReduce**

Bulk Synchronous Parallel



https://en.wikipedia.org/wiki/File:Bsp.wiki.fig1.svg

MapReduce

- Distributed computing paradigm
- Two steps:
  - Map: filter, sort, produce local summaries
  - Reduce: combine to produce the result(s)
- Or, **split-apply-combine**
- Based on the map() and reduce() procedures from functional computing

MapReduce Innovations

- Give the user a constrained framework and make them fit their problem to it
  - Parallelism is automatic
  - Fault tolerance can be taken care of
  - Development time is reduced
- ***Push computations to the data***
  - (or: don't **pull** data to the computation)

Thought Experiment: M/R

- What would it take to add basic MapReduce functionality to your DFS?
- What's the most basic system we could design for this?
- How about **ssh**?