Lecture 30 — Clusters & Cloud Computing

Patrick Lam
patrick.lam@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

November 23, 2020

More, More, More

So far, we've seen how to make things fast on one computer:

- threads;
- compiler optimizations;
- GPUs.

To get a lot of bandwidth, though, you need lots of computers,
(if you're lucky and the problem allows!)

More, More, More

Today: programming for performance with multiple computers.

Key Idea: Explicit Communication

Rust encourages message-passing, but a lot of your previous experience when working with C may have centred around shared memory systems.

Sometimes: no choice! Such as GPU.

Recently, GPU programming: explicitly copy data.

Communication over the network is much more expensive than within the same system.

What is MPI?

Message Passing Interface:

A language-independent communation protocol
for parallel computers.

This is, unfortunately, no longer the way.

REST

We've already seen asynchronous I/O using HTTP (curl) $\rightarrow$ REST!

You may have also learned about sockets $\rightarrow$ too low level.

The REST API approach is at a reasonable level of abstraction.

Synchronous REST

REST APIs are often completely synchronous, but don’t have to be:

You can set up callbacks or check back later.

The remote machine has to be available at the time of each call...

Apache Kafka: a self-described “distributed streaming platform”.

Kafka

Producers write a record into a topic and consumers take the item from the topic and doing something useful with it.

A message remains available for a fixed period of time and can be replayed if needed.

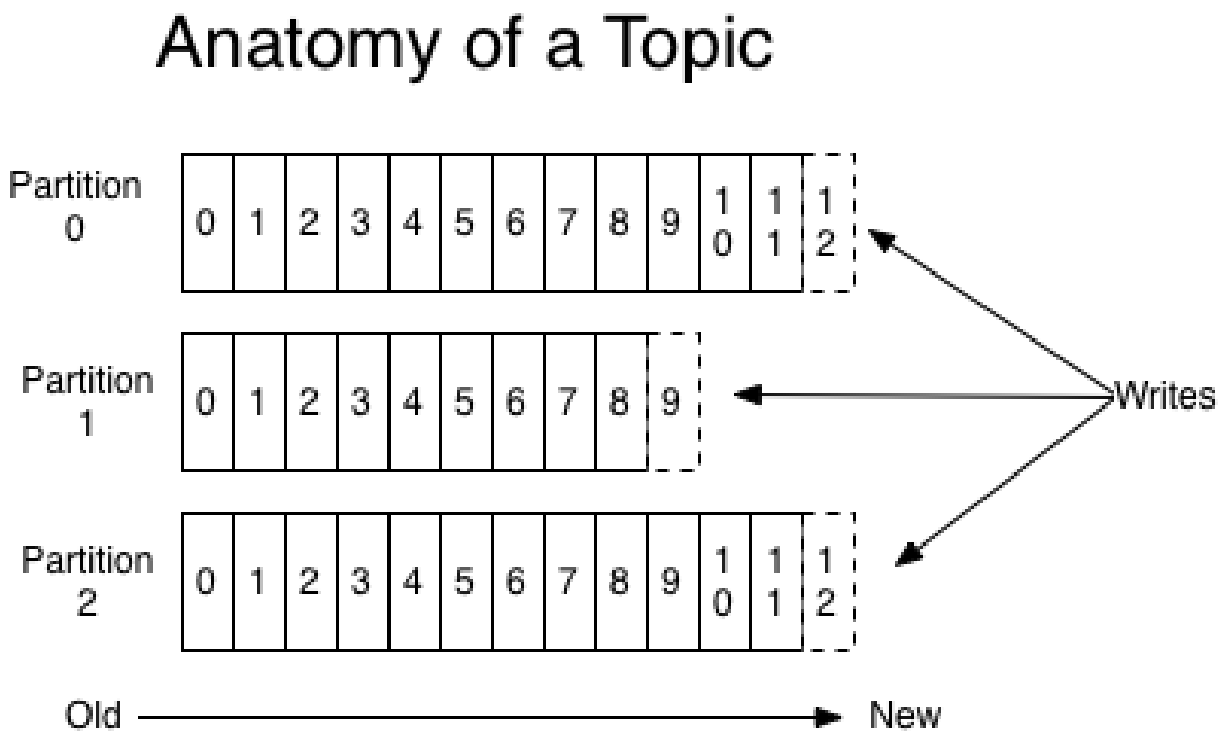
Publish-subscribe model.

Change is Hard

Kafka's basic strategy is to write things into an immutable log.

The log is split into different partitions.

Consumers read from each one of the partitions and writes down its progress.



Kafka Provisioning

We can provision the parallelism that we want, and the logic for the broker.

Consumers can take items and deal with them at their own speed.

Messages are removed from the topic based on their expiry.

Hurry up and Queue

In something like a standard queue, there's a little bit of pressure to get items out of the queue quickly.

You might think that it's a solution to take the item out of the queue in one transaction and then process it later.

That's okay only if you've successfully saved it to a database or other persistent storage.

Alternatives: SQS and SNS

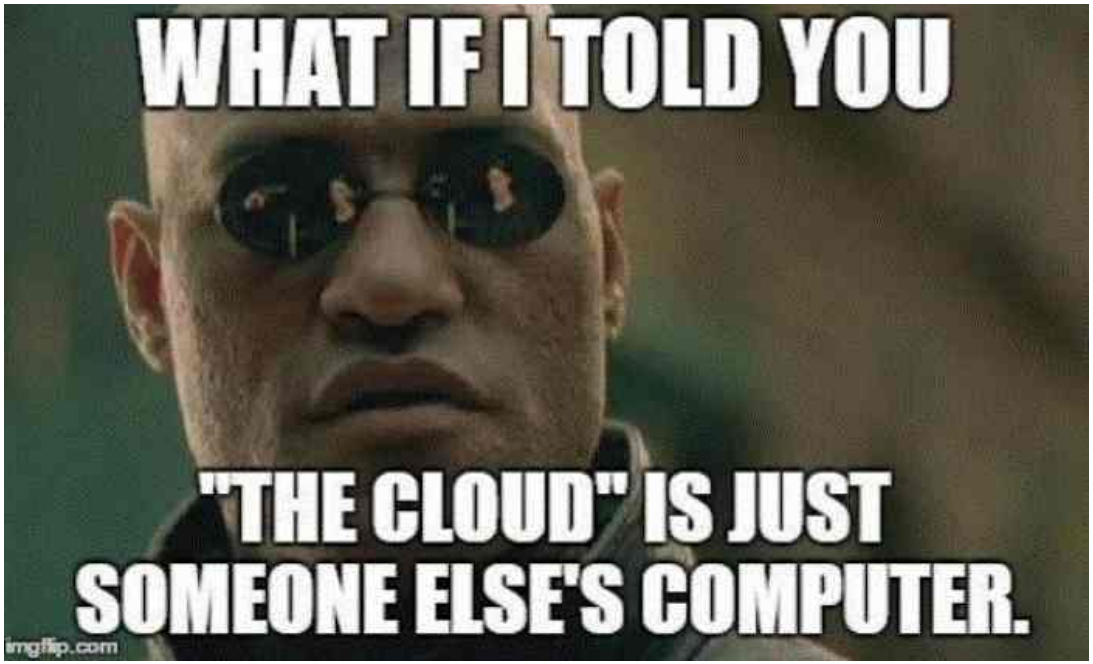
SNS (Simple Notification Service) and SQS (Simple Queueing Service).

They are, broadly speaking, just other ways to decouple the communication of your programs.

SNS is good for sending lots of messages to multiple receivers.

SQS is more for batches of work where it’s not particularly time-sensitive and the item will be consumed by a worker.

How would you know the difference?



Using a Cluster

Historically:

- find $$$;
- buy and maintain pile of expensive machines.

Not anymore!

We'll talk about Amazon's Elastic Compute Cloud (EC2) and principles behind it.

Evolution of servers

You want a server on the Internet.

- Once upon a time: physical machine; shared hosting.
- Virtualization:
- Clouds

Servers typically share persistent storage, also in the cloud.

Paying for Computes

Cloud computing: pay by the number of instances that you've started up.

Providers offer different instance sizes: vary in cores, memory, GPU...

Launching Instances

Need more computes? Launch an instance!

Input: Virtual Machine image.

Mechanics: use a command-line or web-based tool.

New instance gets an IP address and is network-accessible.
You have full root access to that instance.

What to Launch?

Amazon provides public images:

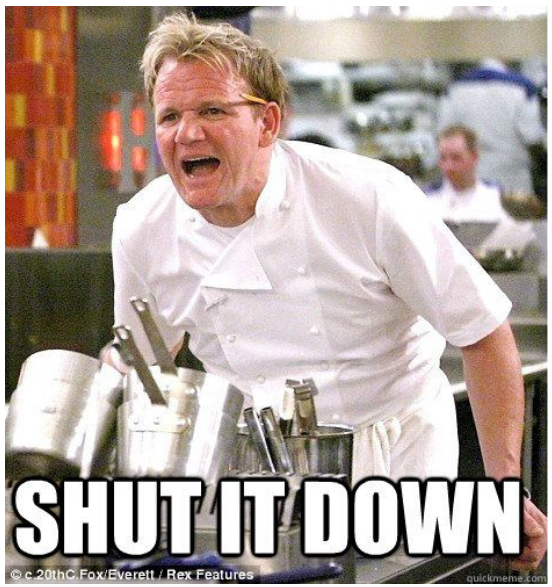
- different Linux distributions;
- Windows Server; and
- OpenSolaris (maybe not anymore?).

You can build an image which contains software you want, including Hadoop and OpenMPI.

Cleanup

Presumably you don’t want to pay forever for your instances.

When you’re done with an instance:



All data on instance goes away.

Data Storage

To keep persistent results:

- mount a storage device, also on the cloud (e.g. Amazon Elastic Block Storage); or,
- connect to a database on a persistent server (e.g. Amazon SimpleDB or Relational Database Service); or,
- you can store files on the Web (e.g. Amazon S3).

Clusters vs. Laptops

Key idea: scaling to big data systems
introduces substantial overhead.

Up next: Laptop vs. 128-core big data systems.

Don’t Guess, Measure

Are big data systems obviously good?
Have we measured (the right thing)?

The important metric is not just scalability;
absolute performance matters a lot.

Why Scale?

Don’t want: scaling up to *n* systems
to deal with complexity of scaling up to *n*.

Or, as Oscar Wilde put it:
“The bureaucracy is expanding to meet the needs of the expanding bureaucracy.”

Methodology

Compare: competent single-threaded implementation vs. top big data systems.

Domain: graph processing algorithms— PageRank and graph connectivity
(bottleneck is label propagation).

Subjects: graphs with billions of edges
(a few GB of data.)

Results

Twenty pagerank iterations

System	cores	twitter_rv	uk_2007_05
Spark	128	857s	1759s
Giraph	128	596s	1235s
GraphLab	128	249s	833s
GraphX	128	419s	462s
Single thread	1	300s	651s

Label propagation to fixed-point (graph connectivity)

System	cores	twitter_rv	uk_2007_05
Spark	128	1784s	8000s+
Giraph	128	200s	8000s+
GraphLab	128	242s	714s
GraphX	128	251s	800s
Single thread	1	153s	417s

Takeaways

- “If you are going to use a big data system for yourself, see if it is faster than your laptop.”
- “If you are going to build a big data system for others, see that it is faster than my laptop.”

Movie Hour, featuring NoSQL Bane

Let’s take a humorous look at cloud computing: James Mickens’ session from Monitorama PDX 2014.

`https://vimeo.com/95066828`