Course Summary: Putting it All Together

CS675: Distributed Systems (Spring 2020)
Lecture 12

Yue Cheng

Announcements

- This is my last lecture of the semester

- Next class, it's your turn:
    - Project presentation: 05/13

- Project report + src due: 05/15

- Final take-home exam: 05/16

Back in Lecture 1...

Distributed systems: What?
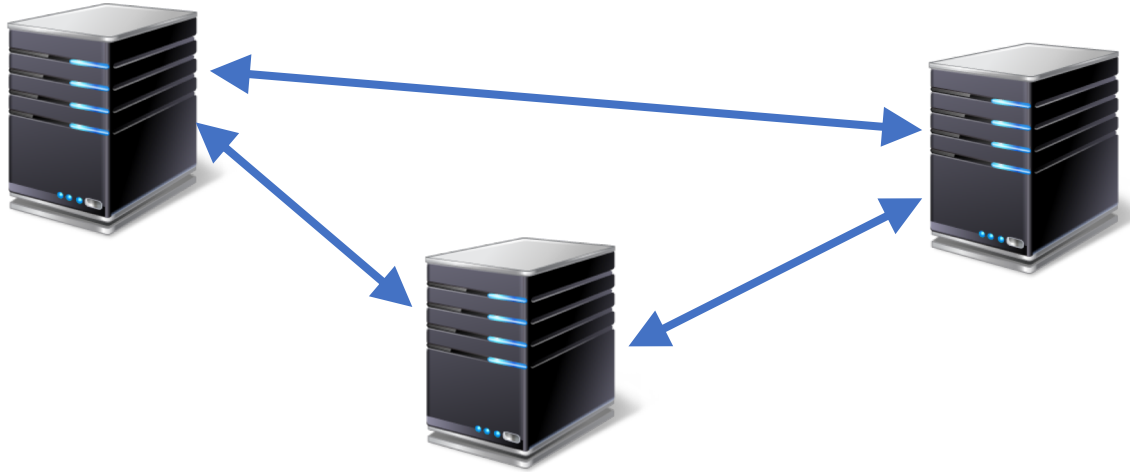
- Multiple cooperating computers
    - Connected by a network
    - Doing something together
- Storage for big websites, MapReduce, etc.
- Lots of critical infrastructure is distributed

Distributed systems: Why?

- Or, why not 1 computer to rule them all?

- Failure

- Limited computation/storage

- Physical location

Distributed systems: Why?

- Or, why not 1 computer to rule them all?

- Failure ➤Fault tolerance

- Limited computation/storage ➤Scalability

- Physical location ➤Availability, low latency

Goals of "distributed systems"

- Service with higher-level abstractions/interface
  - E.g., key-value store, programming model, …

- High complexity
  - Scalable (scale-out)
  - Reliable (fault-tolerant)
  - Well-defined semantics (consistent)

- Do "**heavy lifting**" so app developer doesn't need to

Applications

| Web apps | Data processing | Data storage | Emerging apps? |

Resource management

| Compute resources | Memory resources | Storage resources | Network resources |

Datacenter infrastructure

Theme

- Fundamental building blocks

- Abstractions and programming models

- Distributed systems: Looking forward

Theme

- Fundamental building blocks

- Abstractions and programming models

- Distributed systems: Looking forward

Distributed system building blocks

- Remote procedure calls (RPCs)

Distributed system building blocks

- Remote procedure calls (RPCs)

- Time & clocks

Distributed system building blocks

- Remote procedure calls (RPCs)

- Time & clocks

- Distributed consensus algorithms

Distributed system building blocks

- Remote procedure calls (RPCs)

- Time & clocks

- Distributed consensus algorithms

- Replication, sharding, consistent hashing

Theme

- Fundamental building blocks

- **Abstractions and programming models**

- Distributed systems: Looking forward

How to program many computers?

```
cat data.txt
      | tr –s '[[:punct:][:space:]]' '\n'
      | sort | uniq -c


SELECT count(word), word FROM data
      GROUP BY word
```

Q: How would you implement a distributed framework
to scale out the above computations?

MapReduce abstraction

MapReduce Word Count:

1. In parallel, send to worker:
    - Compute word counts from individual files
    - Collect results, wait until all finished

2. Then merge intermediate output

3. Compute word count on merged intermediates

MapReduce abstracts away distributed system management tasks including scheduling, load balancing, fault tolerance, etc.
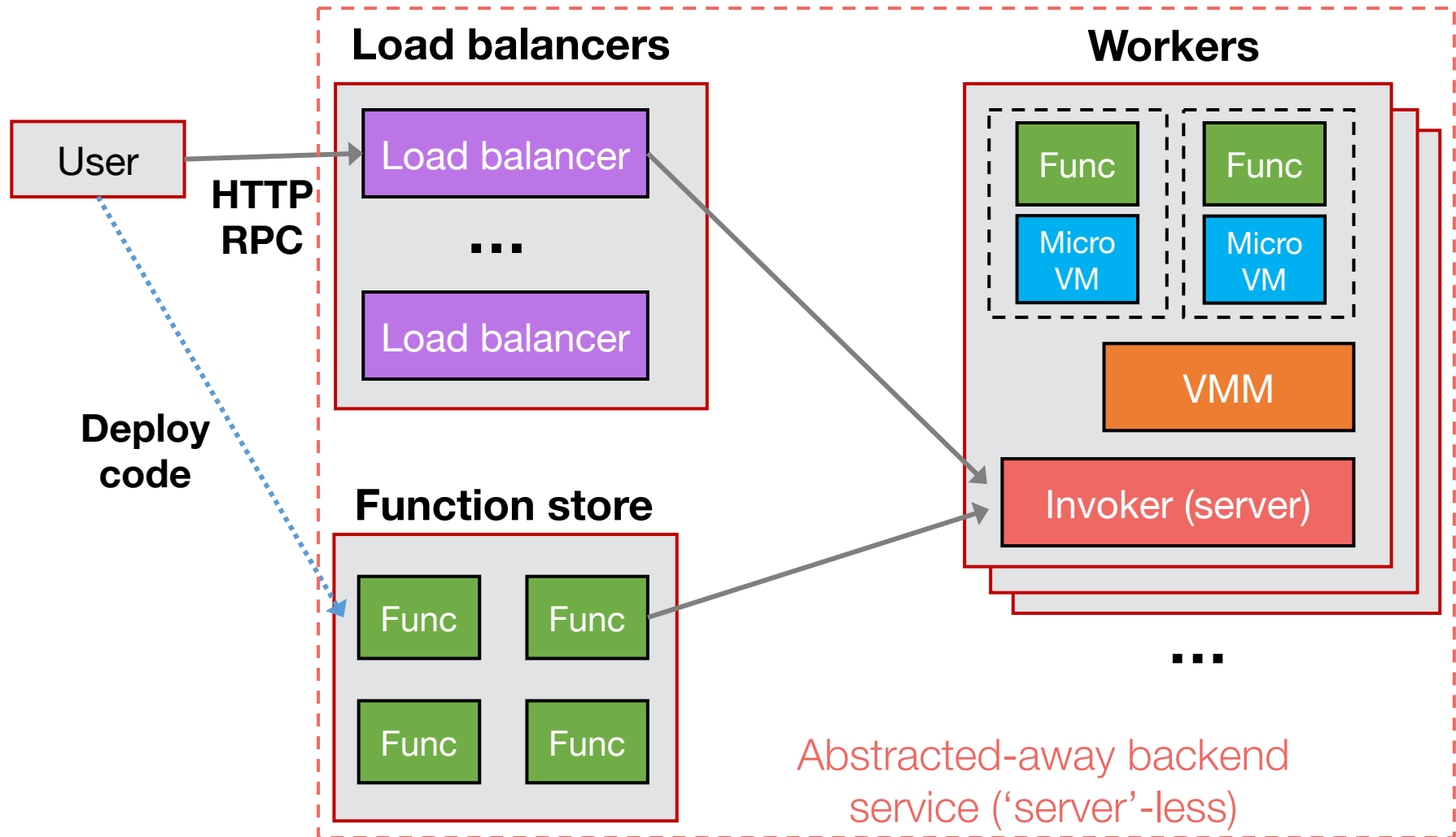
Programming models

- MapReduce

- Spark

Serverless computing abstraction

Serverless computing is a programming abstraction that enables users to upload programs, run them at (virtually) any scale, and pay only for the resources used

Serverless computing abstraction

Load balancers

User — HTTP RPC → Load balancer ... Load balancer

Deploy code

Function store: Func Func Func Func

Workers: Func / Micro VM, Func / Micro VM, VMM, Invoker (server) ...

Abstracted-away backend service ('server'-less)

Serverless computing abstraction

- The abstraction is powerful
  - To express a wide variety of stateless applications such as image processing, ETL

- And yet, the abstraction needs to be augmented
  - For supporting more interesting (complicated) applications such as
    - MapReduce batch processing
    - Distributed machine learning
    - Massive-parallel scientific computing
    - …

Theme

- Fundamental building blocks

- Abstractions and programming models

- **Distributed systems: Looking forward**

Next-generation distributed systems?

Workload?

Distributed Systems?

Hardware
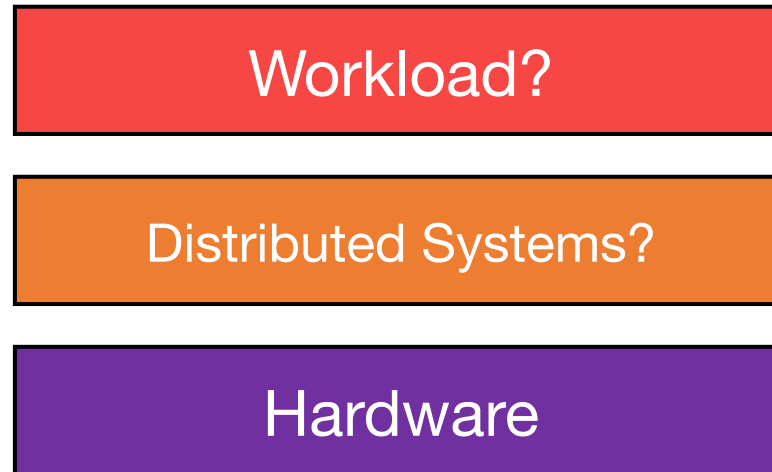
RIP client-server era?

Course summary

Applications

Web apps	Data processing	Data storage	Emerging apps?

Resource management

Compute resources	Memory resources	Storage resources	Network resources

Datacenter infrastructure

Fundamentals

RPC	Time

Consensus

Replication/ sharding

. . .

Abstractions

Programming models

Map Reduce	Spark

Serverless computing

. . .