# Distributed Resource Allocation

Distributed Systems Spring 2019

Lecture 24

# Agenda

1. How to allocate computing resources for distributed applications?
2. What distributed systems principles can be used in allocation?

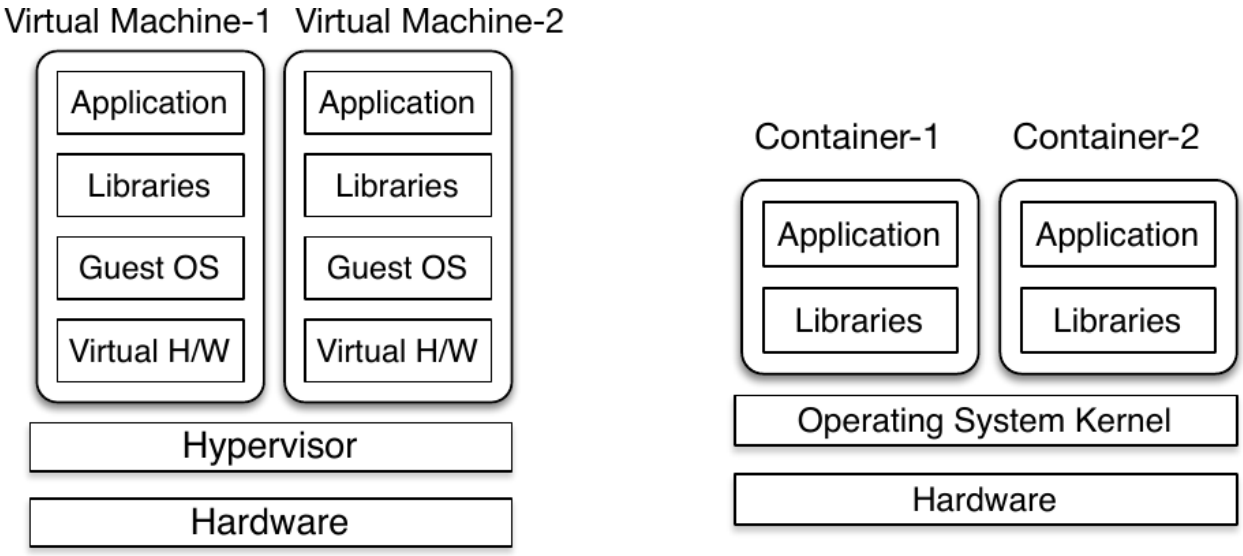
# Computing Resources

All applications and processes need:

1. CPU cycles (i.e., bandwidth)
2. Physical memory
3. Disk bandwidth
4. Network bandwidth
5. More recently: Special purpose accelerators (GPUs, FPGAs, ASICs)

Common to express resource allocations in form of resource vectors.

# Virtualization For Resource Allocation




- Virtualization makes fine-grained resource allocation easy
- Containers and VMs serve as units of allocation
- Resource management layer (i.e., OS or hypervisor) can set resource limits on the VM or container
- Resource limits can often by dynamically changed (e.g., reduce CPU allocation to 2 cores from 4)

# Resource Allocation In Clusters

- Clusters consist of large numbers of servers ($10^2 - 10^6$)
- Resources can be allocated from **multiple** servers
- Resources allocated as VMs or containers on individual servers
- Allocation decisions made by cluster management software
  - OpenStack, VMWare for VMs
  - Kubernetes, Mesos, Docker swarm for containers
  - Slurm, Torque for HPC...

# High-Level Resource Allocation Flow

1. Applications/Users submit resource requirements (**R**)
   - Total number of resources (CPU cores, memory, I/O bandwidth), or
   - Size of container $\times$ number of containers
2. Each server has a hardware capacity (e.g., 48 cores, 512 GB memory) (**C**)
3. Cluster manager finds free resources on servers to satisfy allocation request
4. In practice, many other allocation constraints:
   - Application quotas: does user have enough "credits"
   - Job start/end deadlines
   - Affinity: Containers should be running on same/nearby servers
   - Anti-affinity: Containers on different servers for fault tolerance
   - Co-location: Applications should not be running on servers with another application

# Resource Allocation Policies

- At a high level, resource allocation is a bin-packing problem
- Also called the "placement" problem
- Which servers to place the containers on?
  - Best fit: Allocate resources from server with most free resources available
  - Worst fit: Server with least free resources
  - First fit: Sort by
- However, this is a *multi dimensional* packing problem : resources=(CPU, mem, disk, network)
- Use cosine similarity between resource requirement and availability vectors: fitness $= \frac{\mathbf{r} \cdot \mathbf{a}}{|\mathbf{r}||\mathbf{a}|}$

# Centralized Resource Allocation

- Cluster manager runs on a single server
- Resource allocation state is centralized
- Set of available servers, resources on each server, map of applications to servers, ...
- If a new application wants resources:
  1. Find best allocation according to placement policy
  2. Update local state (server resource map)
  3. Allocate resources in form of containers/VMs..
- All the advantages and drawbacks of a centralized approach
- Used by Kubernetes, Slurm, OpenStack, VMWare,....

# Distributed Resource Allocation

- Centralized allocation can add to latency
- Allocation in inherently sequential:
- For each new application request, need to serialize access to server resource map using locks
- Note: This problem is only at *large* scale ($10^3$ servers)
- Many applications launches short-lived tasks (e.g., Spark)

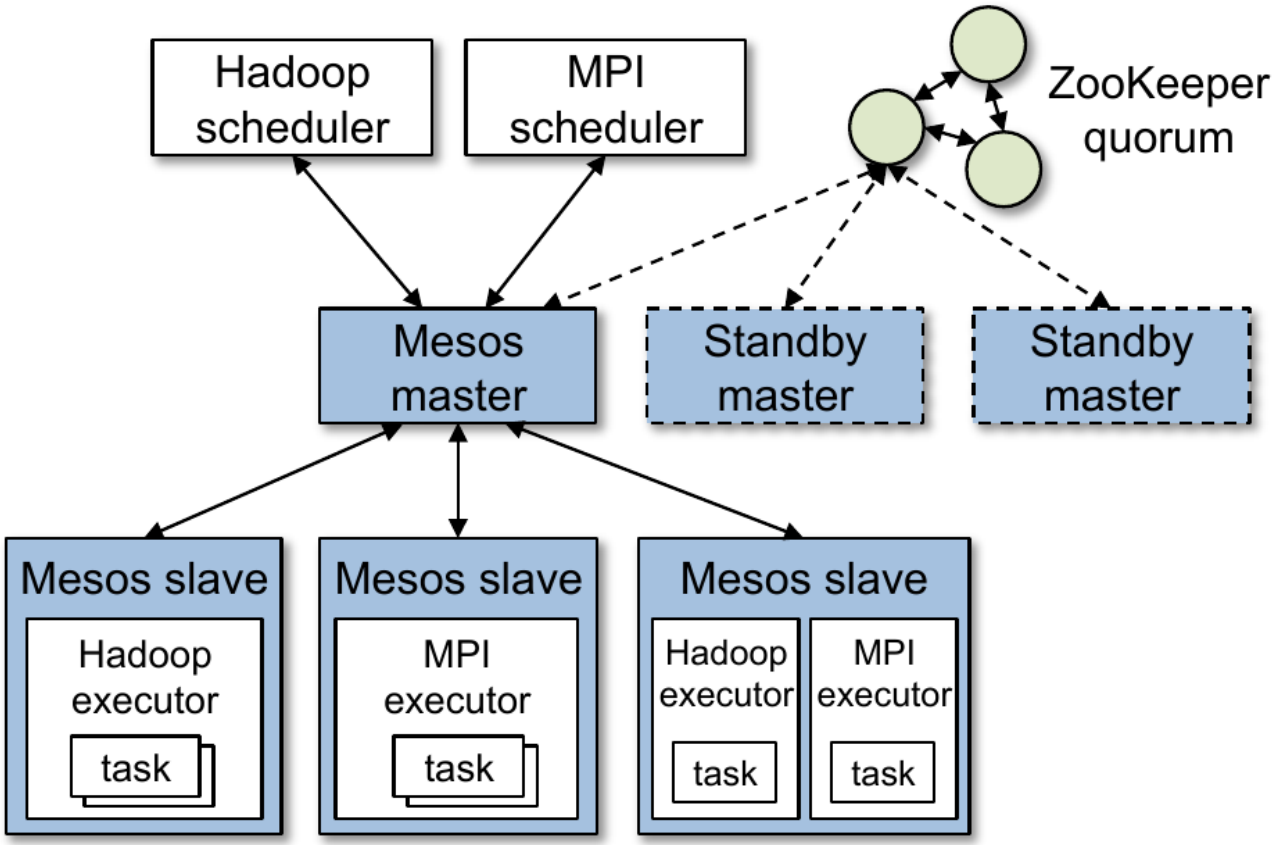
## Distributed Allocation in Sparrow (SOSP 2013)

- No need to contact centralized resource manager
- Key Idea: "Probe" multiple servers, and select one with lower waiting times
- **Power of two choices**: Near-perfect load balancing if you select two servers and pick one with lower load

# Mesos

- Radically different view on cluster resource management
- Neither centralized nor distributed
- Conventionally: allocate resources that have been requested
- Mesos: If there are free resources available, always offer them to applications
  - Work-conserving approach, similar to conventional OSes
  - You dont ask for specific resources before launching an application
- Usecase: Large dedicated clusters for data processing
- Originally co-developed with Spark (short tasks)

# Mesos Architecture



# Fairness In Resource Allocation

- How many resources to allocate to each application?
- Total cluster capacity: 9 CPUs, 18 GB RAM
- User A: Tasks with 1CPU, 4GB
- User B: Tasks with 3CPUs, 1GB
- How many tasks to run from User A and B?
- For CPU: $1a + 3b \leq 9$. Mem: $4a + b \leq 18$
- What is the objective function?
- Many ways of allocating with different tradeoffs:
  - Sharing incentive: Each user better off sharing the cluster
  - Strategy proofness: Cannot improve allocation by lying
  - Envy freeness: A user should not prefer allocation of another (i.e., Fairness)
  - Pareto efficiency: Not possible to increase allocation of a user without decreasing allocation of another

# Dominant Resource Fairness

- Dominant Share: What resource is a user most constrained by?
- A: (1, 4), B: (3, 1). Capacity=(9, 18)
- A's dominant resource is memory ($4/18 > 1/9$)
- Dominant Resource Fairness: Equalize each application's dominant resource
- $4a/18 = 3b/9 \rightarrow a = 3b/2$