CS 6453 – LECTURE 6: MESOS PLATFORM

REUBEN RAPPAPORT
WHAT IS THE PROBLEM?

• There are many existing frameworks for cluster computing
• Generally, different frameworks are best for each application
• Obvious problem: How to share cluster between frameworks
  • Static Partition
  • Allocate VMs on a per framework basis
• None of these perform well with fine grained tasks
MESOS PLATFORM

• Thin resource sharing layer
• Allows multiple cluster frameworks to run simultaneously
• Provides common interface for all frameworks to access resources
• Decentralized scheduler
  • Works on resource offer model
  • Mesos decides how many resources to offer each framework
  • Frameworks decide which offered resources to use for what
WHY IS IT INTERESTING?

• Resource sharing allows for new and exciting cluster configurations
• Can run multiple instances of same framework on different workloads as an experiment
• Much easier to write specialized frameworks that only solve a single problem
RELATED WORK

- High Performance Computing has a large literature on cluster management
  - Optimized for setup with course grained monolithic jobs
  - Designed for specific specialized hardware
- Cloud computing services (eg EC2)
  - VM level abstraction is much more course grained than Mesos
  - No ability to specify placement needs
- Fair usage of cache by multiple users with shared files (FairRide)
- Fair allocation of network resources in cloud computing (FairCloud)
- Many cluster management frameworks contain their own frameworks (Quincy, Condor, etc)
MESOS MODEL

- Mesos master consists of pluggable allocator
  - Decides how to assign resource offers
  - Other masters run on standby for fault tolerance
  - Master consists of soft state only – it’s entirely reconstructable from the schedulers and slaves
MEOS MODEL

- Frameworks consist of two components
  - Schedulers accept or reject resource offers and decide which tasks to run where
  - Slaves actually run tasks and report their status to the allocator
  - Slaves are isolated using containers
SCALABILITY

• To avoid sending unnecessary resource offers Mesos allows schedulers to specify filters
  • Boolean predicate resource offer must satisfy in order to be sent in the first place
  • Scheduler is still free to reject or accept tasks which satisfy it
• Mesos allows schedulers to create duplicates of themselves running on standby
  • When the master scheduler fails it is replaced by one of these
DEALING WITH WAYWARD SCHEDULERS

- Schedulers are assigned a *guaranteed allocation*
  - When they are under this limit their tasks are safe
  - If they go over it then the allocator reserves the right to kill their tasks if needed
- Until an offer has been rejected, Mesos counts it towards the scheduler it was sent to’s total allocation
  - This incentivizes quick offer processing
- If a scheduler takes too long to reply to an offer Mesos will rescind it
EVALUATION SETUP

- Comparison of running workloads on Mesos vs running them with static partitioning
- Four workloads
  - Hadoop mix based off Facebook workload dataset
  - Large Hadoop mix emulating batch workload
  - Spark machine learning job
  - Torque/MPI raytracing job
EVALUATION RESULTS

- Mesos scales resource allocation as demand changes
- Much better utilization than static partitioning
- Ability to scale up in short bursts when demand allows it improves performance
EVALUATION RESULTS

- Utilization results much better than static partitioning overall
- Mesos shows a stronger improvement for memory utilization than for CPU
  - This is likely due to its strong focus on data locality in assigning fine grained tasks
• Mesos allows CPU share to scale with demand as the relative needs change
• Fine grained task allocation makes adjusting to changes rapid
The tachyon ray tracing job is the only one which performed worse on Mesos than on the static partition.

- This is likely a result of the job’s long task times and strong interdependency – it runs as slowly as the slowest node so stragglers drag it down.

Overall the Mesos platform imposes about a 4% overhead.

In a separate scalability experiment Mesos ran on a 50,000 node system without imposing a significant additional overhead.
DOWNSIDES

• Mesos works best when jobs are shortlived and small relative to the size of the cluster

• Individual frameworks don’t have enough knowledge to implement preemption or policies that require views of the whole cluster
  • Frameworks trying to implement gang scheduling will be incentivized to hoard resources, possibly resulting in deadlock until the allocator begins to forcibly terminate tasks
GOING FORWARD

• Possible future experiments
  • Run several instances of the same framework side by side and compare their performance on differing workloads
  • Characterize the effect that frameworks with certain characteristics have on other frameworks running on the cluster – do greedy frameworks starve more timid ones?
GOING FORWARD

• Holy grail in this space would be a decentralized scheduler that can perform just as well as a central one
• Mesos does a reasonable job of approximating it but falls fall short of optimality and incurs an overhead (albeit not a large one)
• Probably not achievable – the best we can do is try to build better and better approximations