Scheduling Deep Dive
Mesos, Kubernetes and Docker Swarm
Dharmesh Kakadia
$whoami

- Work @Azure HDInsight
- Ex - Microsoft Research
- Research on Scheduling
- Love distributed systems
- Interested in large scale Data & Cloud
So many schedulers, so little time

- Kubernetes
- Mesos
- SwarmKit
- Nomad
- ECS/ACS/...
- Firmament/Sparrow
- YARN
- Borg/Appolo/Omega/...
- ...

So many choices
So little time!
Difficult choices

• Community
• Ease of use for developers/operators
• Fault tolerance
• Resource optimizations, oversubscription, reservations, preemption,..
• Scalability
• Extensibility
• Debuggability
• *-bility
What is scheduling?

\[\text{assignment}(\text{task}, \text{node}) = \text{schedule}(\text{resources}, \text{requests}, \text{context})\]

- Resource allocation
- Task scheduling
Common terminology

• Resource(s) : A (set of) vector of CPU, Memory, Network etc.
• Request(s) : A (set of) resource vector, asked from workload to scheduler
• Container/Task : A self contained unit of work
• Service/Framework : A set of related containers/tasks
• Resource allocation : How the resources are assigned to various
• Scheduling : What/How tasks run on given resources
• Constraints/predicates : A set of hard restrictions on where tasks can run
• Unit of scheduling : The minimum entity that is accounted by scheduler
Mesos

• A distributed systems kernel.
• Provides primitives to write data center scale applications – frameworks.
• Manages resource allocation only and leaves the task scheduling framework.
1. The Mesos master receives the resource offers from slaves. It invokes the allocation module and decides which frameworks should receive the resource offers.

2. The framework scheduler receives the resource offers from the Mesos master.

3. On receiving the resource offers, the framework scheduler inspects the offer to decide whether it's suitable.
   - If it finds it satisfactory, the framework scheduler accepts the offer and replies to the master with the list of executors that should be run on the slave, utilizing the accepted resource offers.
   - Or the framework can reject the offer and wait for a better offer.

4. The slave allocates the requested resources and launches the task executors. The executor is launched on slave nodes and runs the framework's tasks.

5. The framework scheduler gets notified about the task's completion or failure. The framework scheduler will continue receiving the resource offers and task reports and launch tasks as it sees fit.

6. The framework unregisters with the Mesos master and will not receive any further resource offers. This is optional and a long running services may not unregister during the normal operation.
Mesos Allocator

• Mesos allocator is based on online Dominant Resource Fairness (DRF) called HierarchicalDRF.
• DRF generalizes the fairness concepts to multiple resources.
• Dominant resources share: Resource for which the user has the biggest share.

For example, if total resources are <8CPU,5GB>, user has <2CPU,1GB>, the user’s dominant resources will be \(\max(\frac{2}{8}, \frac{1}{5}) = 0.25\). DRF applies the fairness on the dominant resource.
DRF properties

• Strategy proof
• Incentive to share
• Single resource fairness
• Envy Free
• Bottleneck fairness
• Monotonicity
• Pareto efficient
Advanced Mesos scheduling

• Pure DRF might not be sufficient for reflecting organization priorities
• Production resources probably are more important then an intern experiment
• Weighted DRF, divides the dominant share with configured weights
• Specify --weights and --roles flags to master
Advanced Mesos scheduling

• Without reservation, you are not guaranteed to get back the resources. Not good for cache/storage scenarios.
• Reservation allows guaranteed resources on slaves.
  • Static reservation: managed through --resources flag on the slave
  • Dynamic reservation: manage via reservation API/endpoint
• Oversubscription support has just landed in Mesos
• Preemption
DRF is good when

• All frameworks have work to do and don’t hold up resources
• A frameworks resource requirement does not change dramatically or changes with the availability of resources
• Framework resource requirement is clear apriori
• All frameworks behave similarly when waiting for more resources
Mesos in the data center
Mesos PaaS frameworks

• Many different implementations of the container management platforms on mesos.
  • Marathon
  • Aurora
  • Cook
  • Singularity

• Marathon supports rich constraints (Unique, Cluster, Group_by, Like, Unlike, Max_per)
Extending Mesos

• Extend task scheduling via framework scheduler implementation
• Extend core functionalities via Mesos Modules
  • Provides a generic integration/extension points
  • Allows extending Mesos without bloating the codebase.
  • Pass in --modules flag Json containing module specification

```json
{
  "libraries": [  
    {  
      "file": "/path/to/libfoo.so",
      "modules": [  
        {  "name": "org_apache_mesos_bar" },
        {  "name": "org_apache_mesos_baz" }
      ]
    ]
  }
}
```
Extending Mesos Allocator Module

• To use custom allocator module, specify --allocator flag on master with the name of the new module.
• Default Allocator is HierarchicalDRFAllocatorProcess.
• HierarchicalDRFAllocatorProcess uses a sorter to decide the order in which frameworks are offered resources.
• If you just want to change how that sorting works, you can implement just a sorter implementation
Kubernetes

- Container orchestration and management
- Scheduling: Filter, followed by ranking

For each pod:
  - Filter nodes with at least required resources
  - Assign the pod to the “best” node. Best is defined with highest priority.
  - If multiple nodes have the same highest priority, choose at random.
Filter Predicates in Kubernetes

- PodFitsResources
- CheckNodeMemoryPressure
- CheckNodeDiskPressure
- PodFitsHostPorts
- HostName
- MatchNodeSelector
- NoDiskConflict
- NoVolumeZoneConflict
- MaxEBSVolumeCount
- MaxGCEPDVolumeCount
Ranking in Kubernetes

• finalScoreNodeA = (weight1 * priorityFunc1) + (weight2 * priorityFunc2)

• Default ranking strategies:
  • LeastRequestedPriority
  • BalancedResourceAllocation
  • SelectorSpreadPriority/ServiceSpreadingPriority
  • CalculateAntiAffinityPriority
  • ImageLocalityPriority
  • NodeAffinityPriority
Extending Kubernetes

• You can change the default scheduler policy by specifying --policy-config-file to the kube-scheduler

```json
{
    "kind": "Policy",
    "apiVersion": "v1",
    "predicates": [
        {
            "name": "PodFitsHostPorts",
        },
        {
            "name": "PodFitsResources",
        },
    ],
    "priorities": [
        {
            "name": "LeastRequestedPriority",
            "weight": 1,
        },
        {
            "name": "BalancedResourceAllocation",
            "weight": 1,
        },
        {
            "name": "hardPodAffinitySymmetricWeight",
            "weight": 10
        }
    ]
}
```

• If you want to use custom scheduler for your pod instead of the default kube-scheduler, specify spec.schedulerName
Advanced Kubernetes Scheduling

- Resource Quality of Service proposal
- Resource limits and Oversubscription
  - BestEffort
  - Guaranteed
  - Burstable
- Admission control limit range proposal
Docker Swarm/Swarmkit

- Leverages the familiar Docker API and tooling to run containers across multiple docker hosts.
SwarmKit node selection algorithm

• Filter based approach to find the “best” node for the task.
• Manager accepts service definition and converts them to tasks. Then it allocates resources and dispatches tasks to nodes.
• Orchestrator makes sure that service have right number of tasks running. Scheduler assigns tasks to available nodes.
• Constraints are AND matched
• Strategies : only spread right now. Schedule task on the least loaded nodes (after filtering them based on resources and constraints).
• Pipeline runs a set of filter on nodes
SwarmKit constraints

- ReadyFilter – if node is up and ready.
- ResourceFilter – if node has sufficient resources for the task
- PluginFilter – if node has required plugins installed – volume/network plugins
- ConstraintFilter – any key-value based filtering
- PlatformFilter – filter nodes with specific platform – x86/OS etc
- HostPortFilter – are required ports available
Future features

• Smarter Rescheduling
• Richer resource specification
• Resource estimation (from history/statistics/traces)
• Better context to the scheduler
• We are leveraging cloud for data, but we should also leverage data for cloud.
Thanks !