Distributed Resource Scheduling Frameworks

Is there a clear winner?

- NAGANARASIMHA G R & VARUN SAXENA

Who we are !

Naganarasimha G R

- System Architect @ Huawei
- Apache Hadoop Committer
- Working in Hadoop YARN team.
- Hobbies :
  - Chess, Cycling

Varun Saxena

- Senior Technical Lead @ Huawei
- Apache Hadoop Committer
- Working in Hadoop YARN team.
- Hobbies :
  - Photography

Swarm

Paragon

Nomad

YARN

IBM HPC

Borg

Omega

Kubernetes

Apollo

Hawk

Mercury

Tarcil

Mesos(Marathon)

Cloud Foundry (Diego)

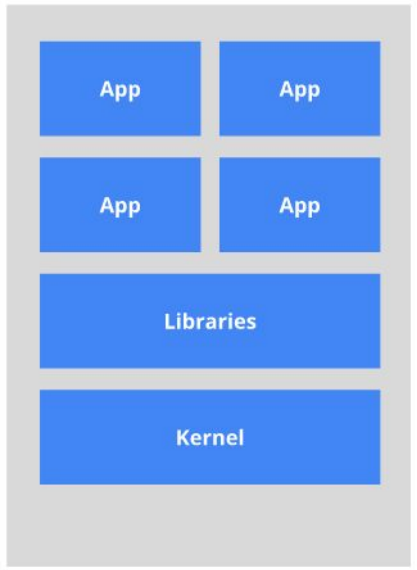
Sparrow

Agenda

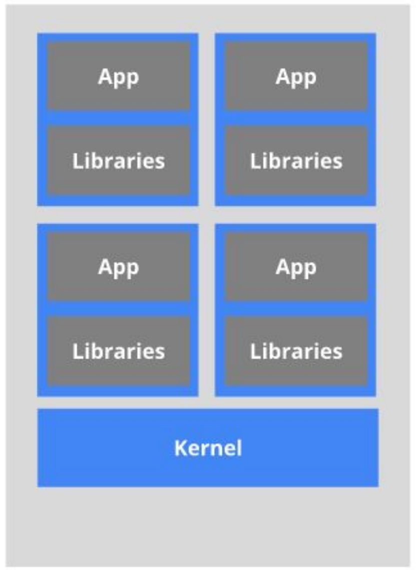
- **Aspects of Distributed Scheduling Framework**
- Architectural evolution of resource scheduling
- Overview of prominent open source schedulers
- Functional comparison between prominent schedulers
- Upcoming features in YARN, bridging the gap

Aspects of Distributed Scheduling Framework

- ❑ Ability to support varied resources types and ensuring isolation
  - ❑ Support of multiple resource type (CPU, Mem, Disk, Network, GPU etc...)
  - ❑ Pluggable resource type
  - ❑ Hierarchical/nested resource types
  - ❑ Macro(logical partition) and Micro(cgroups) isolation
  - ❑ labelling of nodes
- ❑ Ability to orchestrate Containers
  - ❑ Support for multiple container types(Docker, Rocket)
  - ❑ Manage life cycle of Containers
  - ❑ Support repository Management of Container Images



Heavyweight, non-portable
Relies on OS package manager



Small and fast, portable
Uses OS-level virtualization

Aspects of Distributed Scheduling Framework

- Ability to support wide variety of applications
  - Big Data (stateful, DAG, ad hoc, batch)
  - Long running services (stateless, stateful apps)
  - Support of DevOps and MicroServices Model
- Networking support
  - Network proxy/wiring of containers
  - DNS support
  - Service discoverability
- Disk Volumes (Persistence storage)
  - Ability to mounting of multiple Types of Persistent volumes
    - local Block Storage (SSD/SATA)
    - Raid based persistent disks (SSD/SATA).
    - Software based storages : NFS
    - Elastic storage for Files/ Objects (GlusterFS , AWS)
  - Dynamic mounting

Aspects of Distributed Scheduling Framework

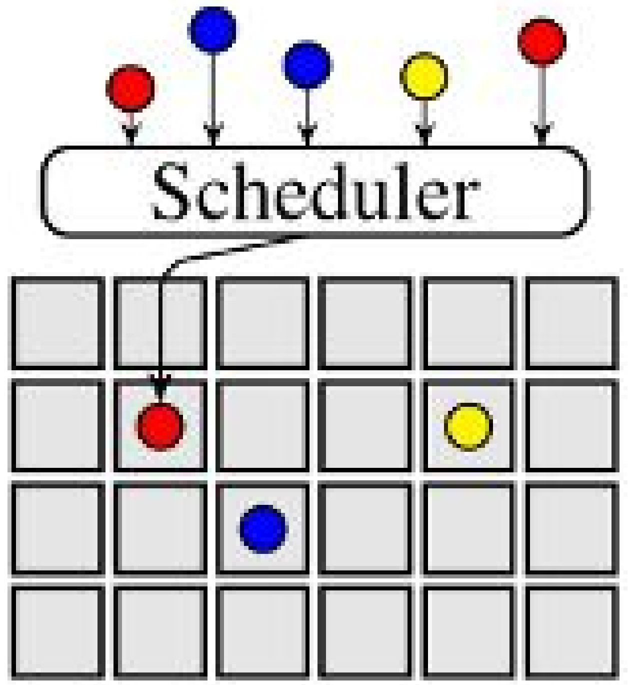
- Scalability and Reliability
  - Daemon Services reliability and scalability
  - Application reliability.
  - Application recoverability
  - Integrated Load Balancer
- Security
  - Namespaces
  - RBAC
  - Pluggable authentication for the enterprise. LDAP integrations ...
  - enforce secure communication in all layers, App - Service , Clients - Service, Clients - Apps
- Others
  - Automatable : Deploy and Build
  - DevOps Collaboration

Agenda

- ❑Aspects of Distributed Scheduling Framework
- ❑**Architectural evolution of resource scheduling**
- ❑Overview of prominent open source schedulers
- ❑Functional comparison between prominent schedulers
- ❑Upcoming features in YARN, bridging the gap

Architectural evolution of resource scheduling

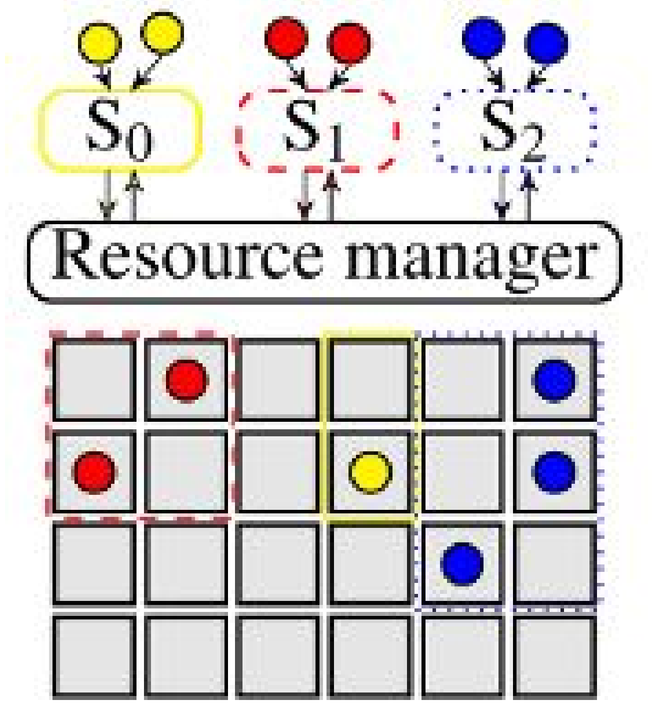
Monolithic Scheduling



- ❑ Many of the cluster schedulers are Monolithic. Enterprise- IBM HPC, Open source - Kubernetes, JobTracker in Hadoop v1
- ❑ A single scheduler process runs on one machine and assigns tasks to machines and it alone handles all different kinds of workloads. All tasks run through the same scheduling logic .
- ❑ Pro's
  - ➢ Sophisticated optimizations to avoid negative interference between workloads competing for resources can be achieved using ML tech. Ex- Yarn, Paragon and Quasar
- ❑ Con's
  - ➢ Support different applications with different needs, Increases the complexity of its logic and implementation, which eventually leads to scheduling latency
  - ➢ Queueing effects (e.g., head-of-line blocking) and backlog of tasks unless the scheduler is carefully designed.
  - ➢ Theoretically might not be scaleable for very large cluster. Ex. Hadoop MRV1

Architectural evolution of resource scheduling

Two Level Scheduling



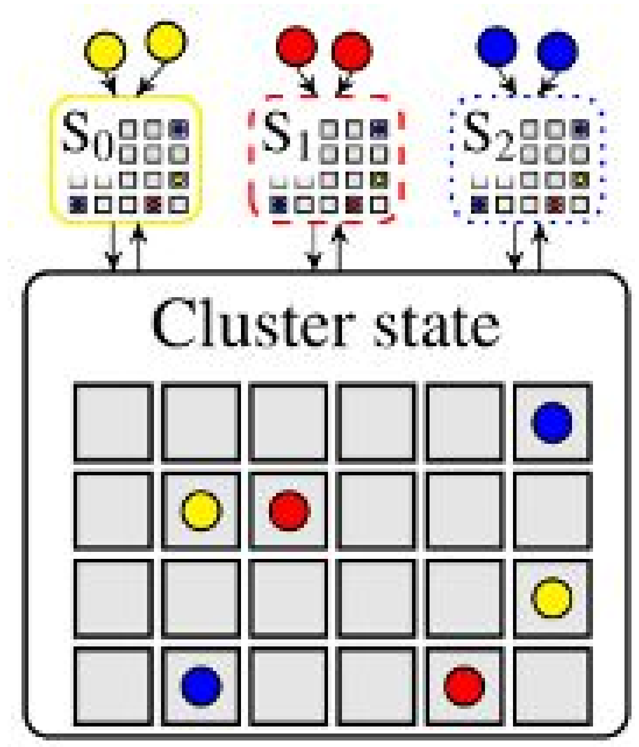
- ❏ Separates the concerns of resource allocation and App's task placement.
- ❏ Task placement logic to be tailored towards specific applications, but also maintains the ability to share the cluster between them.
- ❏ Cluster RM can offer the resources to app level scheduler (pioneered by Mesos) or application-level schedulers to to request resources.

- ❏ Pro's
  - ➢ Easy to carve out a dynamic partition out of cluster and get the application executed in isolation
  - ➢ A very flexible approach that allows for custom, workload-specific scheduling policies.

- ❏ Con's
  - ➢ Information hiding: Cluster RM will not be aware of the App's task and will not be able(/complicates) to optimize the resource usage (preemption)
  - ➢ Interface become complex in request based model.
  - ➢ Resource can get underutlized.

Architectural evolution of resource scheduling

Shared State Scheduling

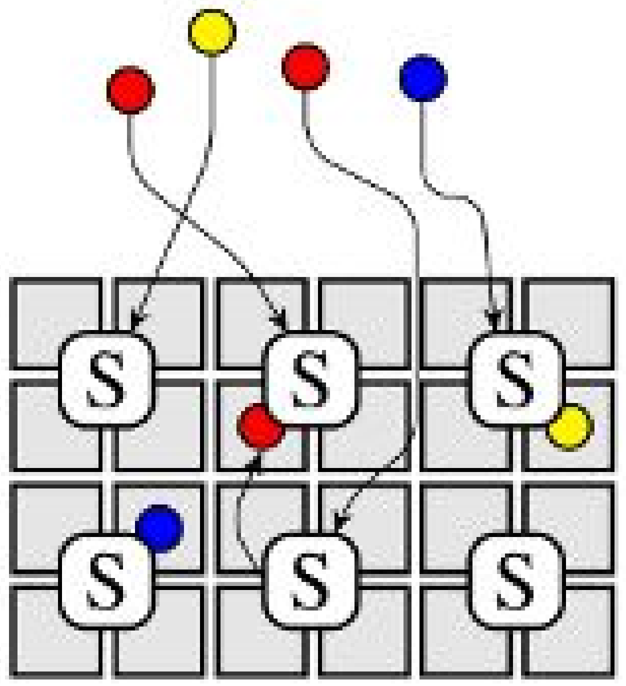


- ❑ Multiple replicas of cluster state are independently updated by application-level schedulers.
- ❑ Task placement logic to be tailored towards specific applications, but also maintains the ability to share the cluster between them.
- ❑ Local scheduler issues an optimistically concurrent transaction to update local changes to the shared cluster state.
- ❑ In the event of transaction failure(another scheduler may have made a conflicting change) local scheduler retries.
- ❑ Prominent examples : google's omega, Microsoft's Apollo, Hashicorp's Nomad, of late Kubernetes something similar.
- ❑ In general shared cluster state is in single location but it can be designed to achieve "logical" shared-state materialising the full cluster state anywhere. ex Apollo

- ❑ Pro's
  - ➢ Partially distributed and hence faster.
- ❑ Con's
  - ➢ Scheduler works with stale information and may experience degraded scheduler performance under high contention.
  - ➢ Need to deal with lot of split brain scenarios to maintain the state. (although this can apply to other architectures as well)

Architectural evolution of resource scheduling

Fully Distributed Scheduling



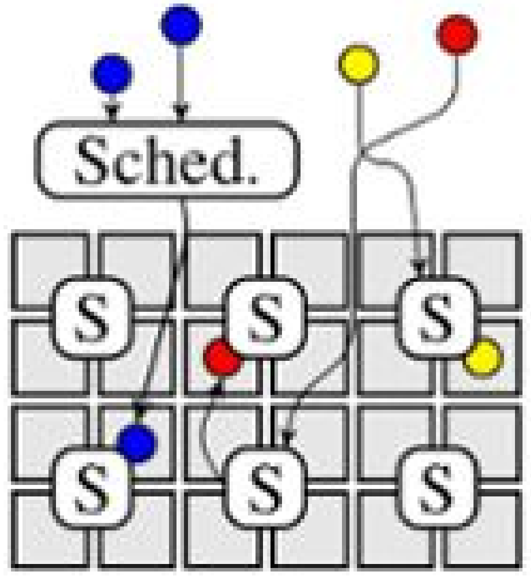
- ❑ Based on hypothesis that the tasks run on clusters are becoming ever shorter in duration and multiple shorter jobs even large batch jobs can be split into small tasks that finish quickly.
- ❑ Workflow :
  - ➢ Multiple Independent schedulers servicing the incoming workload
  - ➢ Each of these schedulers works with its local or partial (subset) of the cluster. No cluster state to be maintained by schedulers.
  - ➢ Based on a simple "slot" concept that chops each machine into n uniform slots, and places up to n parallel tasks.
  - ➢ Worker-side queues with configurable policies (e.g., FIFO in Sparrow),
  - ➢ Scheduler can choose at which machine to enqueue a task which has available slots satisfying the request.
  - ➢ If not available locally then will try to get the slot for other scheduler.
- ❑ Earliest implementers was sparrow.
- ❑ Federated clusters can be visualized similar to Distributed Scheduling albeit if there is no central state maintained.

- ❑ Pro’s
  - ➢ Higher decision throughput must be supported by the scheduler. spread the load across multiple schedulers.

- ❑ Con’s
  - ➢ Difficult to enforce global invariants (fairness policies, strict priority precedence)
  - ➢ Cannot support application-specific scheduling policies. For example Avoiding interference between tasks (as its queued),, becomes tricky.

Architectural evolution of resource scheduling

Hybrid architectures



- ❏ Considered mostly academic.
- ❏ Combines monolithic and Distributed scheduling.
- ❏ Two scheduling paths:
  - ➢ A distributed one for part of the workload (e.g., very short tasks, or low-priority batch workloads).
  - ➢ Centralized one for the rest.
- ❏ Priority will be given to the centralized scheduler in the event of the conflict.
- ❏ Incorporated in Tarcil, Mercury, and Hawk.
- ❏ Is also available as part of YARN, More in next slides.

Agenda

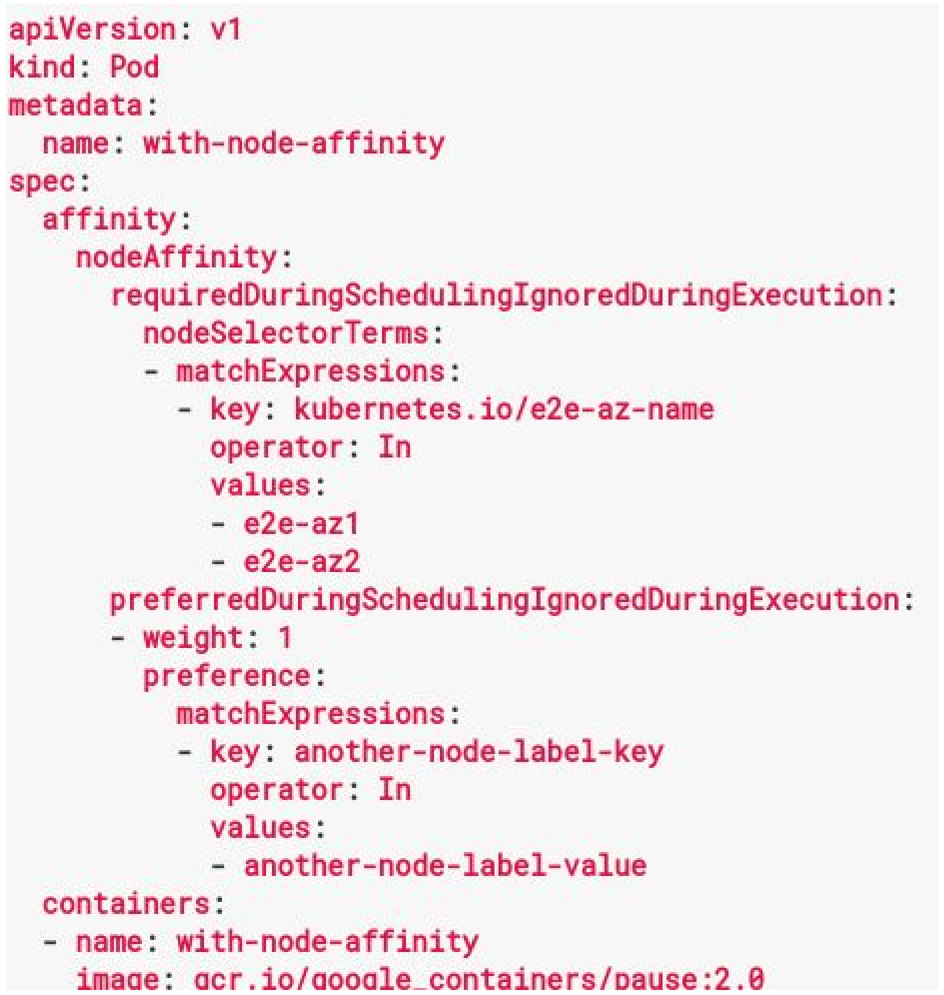
- ❑Aspects of Distributed Scheduling Framework
- ❑Architectural evolution of resource scheduling
- ❑**Overview of prominent open source schedulers**
- ❑Functional comparison between prominent schedulers
- ❑Upcoming features in YARN, bridging the gap

Overview of Kubernetes

Kubernetes Overview

- ❏ Basic abstraction is POD : Co-locating helper processes,
- ❏ Everything App/task is a Container
- ❏ Supports multiple container types: Rocket, Docker
- ❏ Mounting storage systems and dynamic mount of volumes
- ❏ Simple interface for application Developer : YAML
- ❏ Multiple templates /views for the end application
  - ❏ POD
  - ❏ Deployment
  - ❏ ReplicationSet
  - ❏ DaemonServices

- ❏ Supports Multiple Schedulers and lets application to choose containers
- ❏ Default scheduler tries to optimize scheduling by bin packing. And less load tries to pick up the node will less load
- ❏ Supports Horizontal POD scaling for a running app

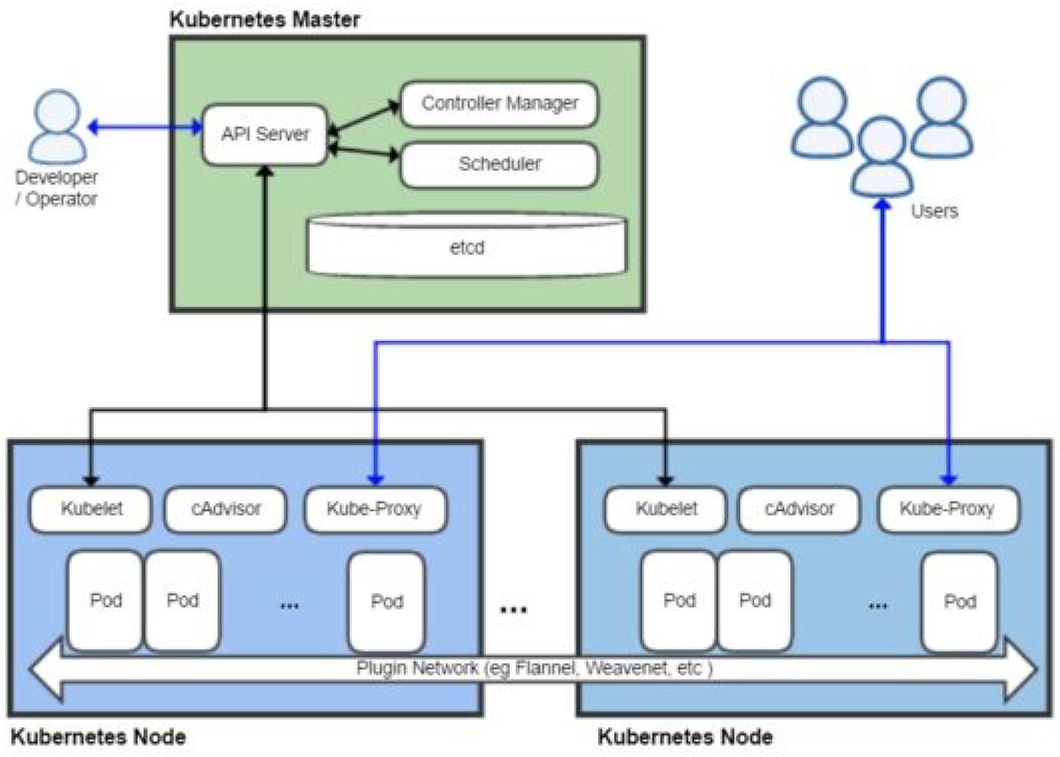


```yaml
apiVersion: v1
kind: Pod
metadata:
  name: with-node-affinity
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: kubernetes.io/e2e-az-name
            operator: In
            values:
            - e2e-az1
            - e2e-az2
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 1
        preference:
          matchExpressions:
          - key: another-node-label-key
            operator: In
            values:
            - another-node-label-value
  containers:
  - name: with-node-affinity
    image: gcr.io/google_containers/pause:2.0
```

Kubernetes YAML file

Overview of Kubernetes

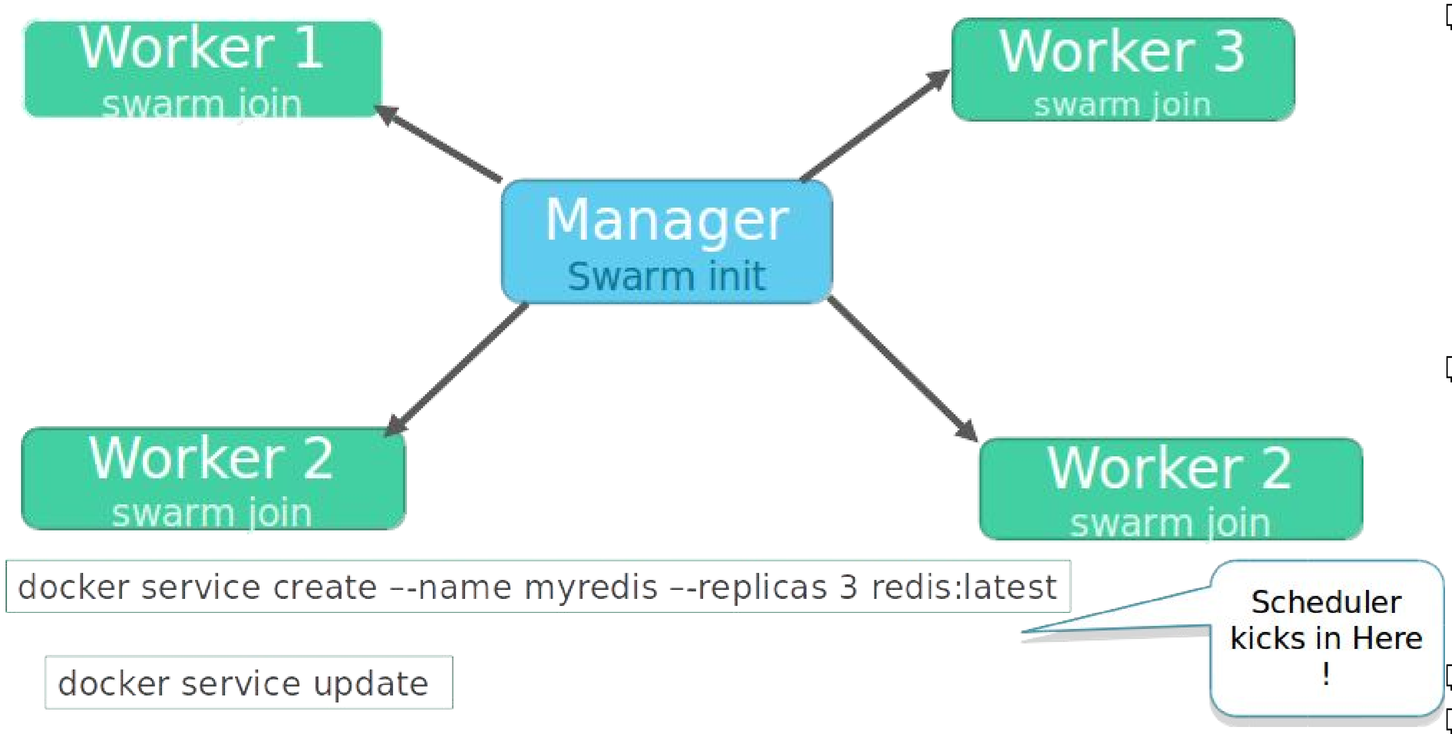
Kubernetes Architecture



1. Master – Cluster controlling unit
2. etcd – HA Key/value store
3. API Server - Observing the state of the cluster
4. Controller Manager – runs multiple controllers
5. Scheduler Server – assigns workloads to nodes
6. Kubelet - server/slave node that runs pods
7. Proxy Service – host subnetting to external parties
8. Pods – One or more containers
9. Services – load balancer for containers
10. Replication Controller – For horizontally-s

Overview of Swarm

Swarm Scheduling Overview



```
docker service create --name myredis --replicas 3 redis:latest
```

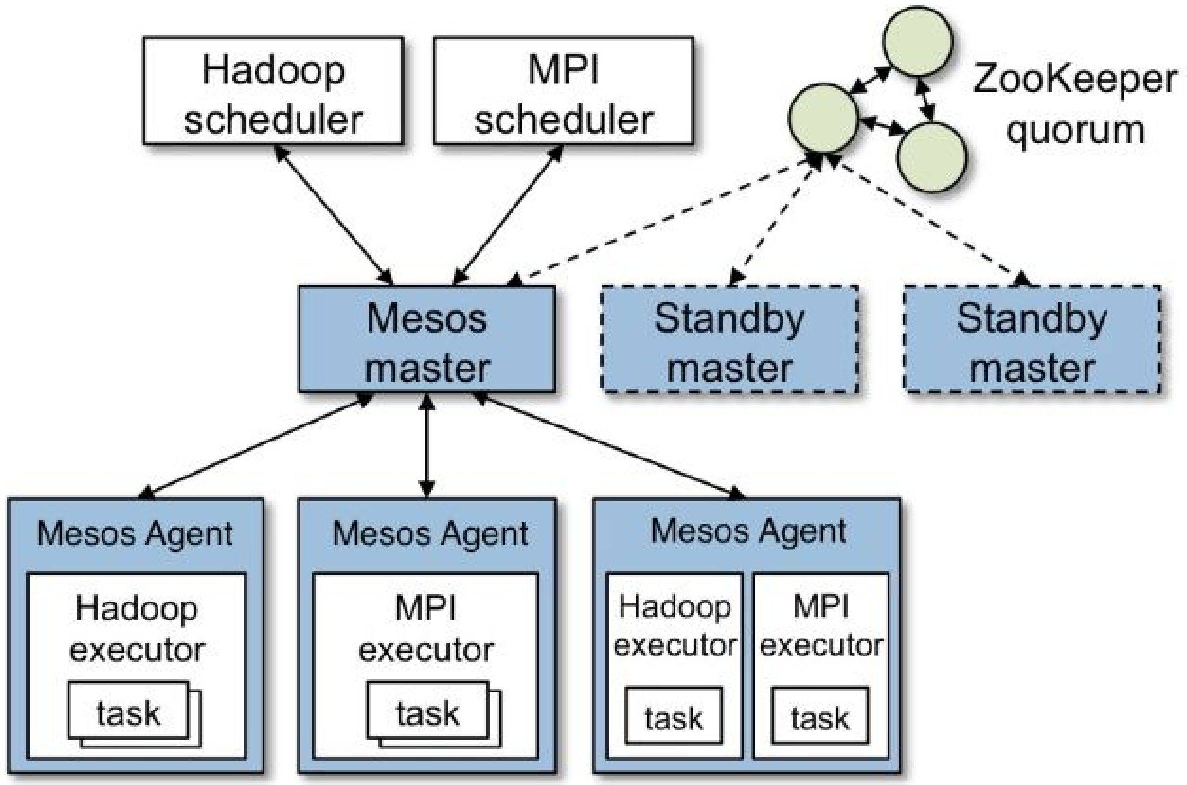
```
docker service update
```

- ❏ Main job of scheduler is to decide which node to use when running docker container/service.
- ❏ Resource Availability : Scheduler is aware of resources available on nodes.
- ❏ Labels and Constraints
  - Label : attribute of the node
    E.g. environment = test, storage = ssd
  - Constraints
    Restrictions applied by Operator while creating a service.
    E.g. docker service create --constraint node.labels.storage==ssd …
- ❏ Strategy : What happens when two nodes are similar
  - Spread strategy : schedule tasks on the least loaded nodes, provided they meet the constraints and resource requirements.
  - Swarm standalone : supported Bean pack and Random strategy

- ❏ NOT SUPPORTED IN SWARM MODE YET !
- ❏ Affinity and Anti Affinity
  - Affinity: two containers should be together
  - Anti Affinity : two containers should not be together

Overview of Mesos

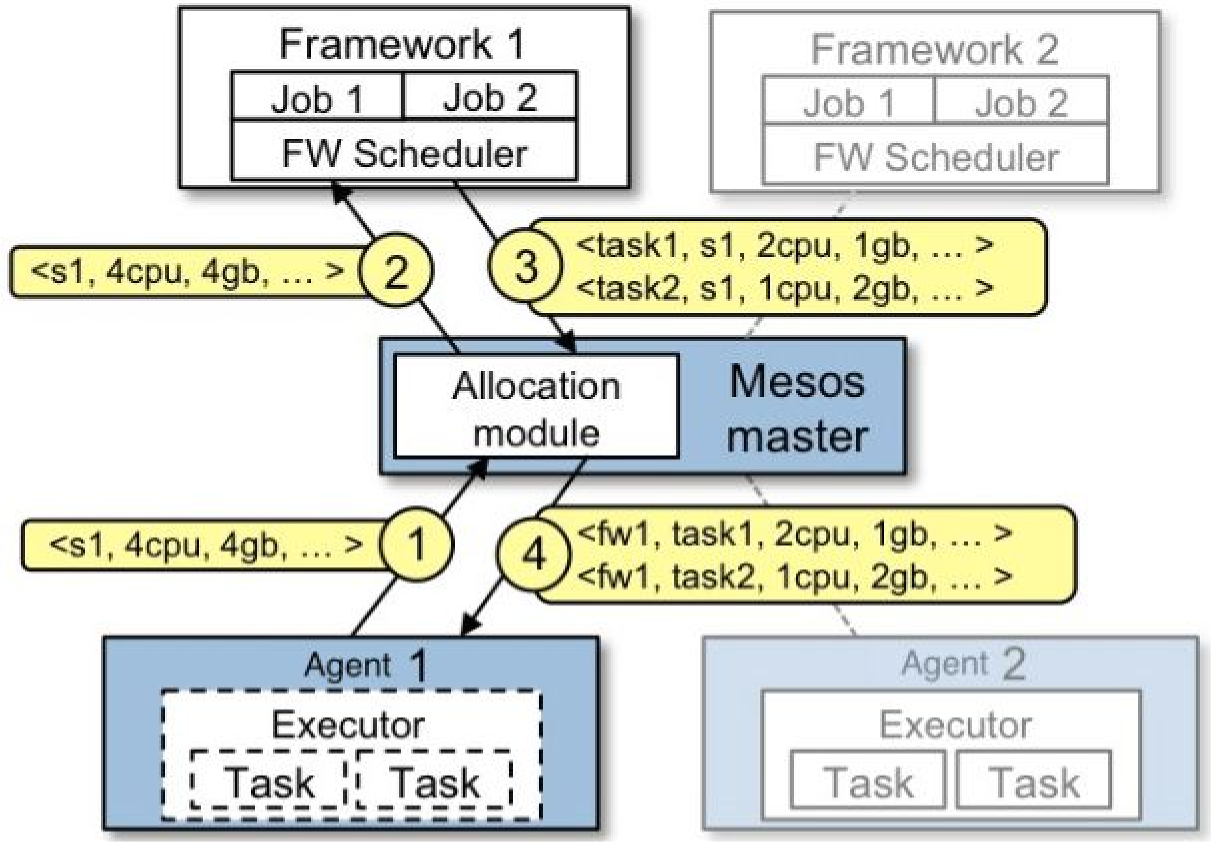
Mesos Architecture



- ❑ Master – Enable sharing of resource
- ❑ Slave – Execute the task in it
- ❑ Cluster – a group of machines
- ❑ ZooKeeper – distributed synchronization/configuration
- ❑ Framework – scheduler + executor
- ❑ Scheduler – Accept resources
- ❑ Executor – Run the Framework task
- ❑ Task – a job to run
- ❑ Containerizer – run & monitor executors

Overview of Mesos

Mesos Scheduling Overview



- ❑ Works on Offer based model.
- ❑ Mesos has two levels of scheduling;
  - one intra-Frameworks level
  - Inter-Framework level application specific.
- ❑ Supports Pools' and ACLs'

Overview of Mesos

Mesos Frameworks :

❑ Dev -OPS : **VAMP**

❑ Long Running services :
  - **Marathon** : Mesosphere’s solution which automatically handles hardware or software failures and ensures that an app is “always on”.
  - **Aurora** : Apache’s project.
  - **Singularity** : for one off tasks and scheduled jobs

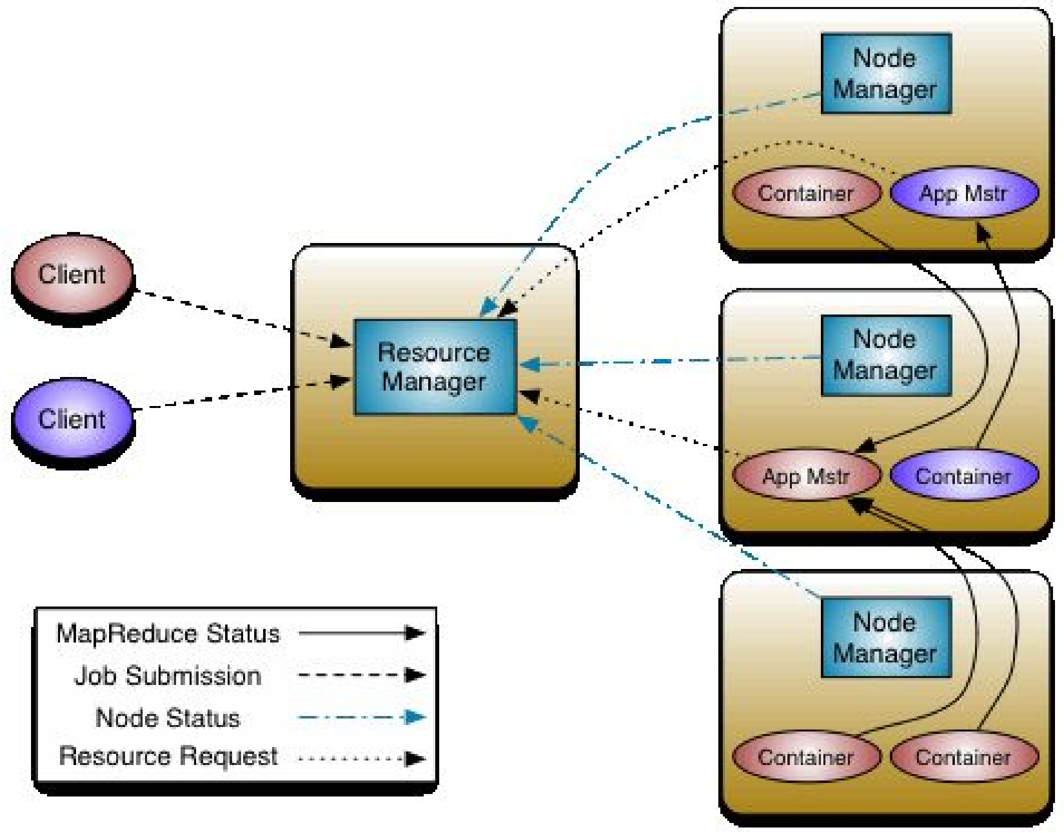
❑ Bigdata processing:
  - **Hadoop** Running Hadoop on Mesos distributes MapReduce jobs efficiently across an entire cluster.
  - **Spark** is a fast and general-purpose cluster computing system which makes parallel jobs easy to write.
  - **Storm** is a distributed realtime computation system.

❑ Batch Scheduling :
  - Chronos is a distributed job scheduler that supports complex job topologies. It can be used as a more fault-tolerant replacement for Cron.

❑ Data Storage : **Alluxio, Cassandra, Ceph**

Overview of YARN

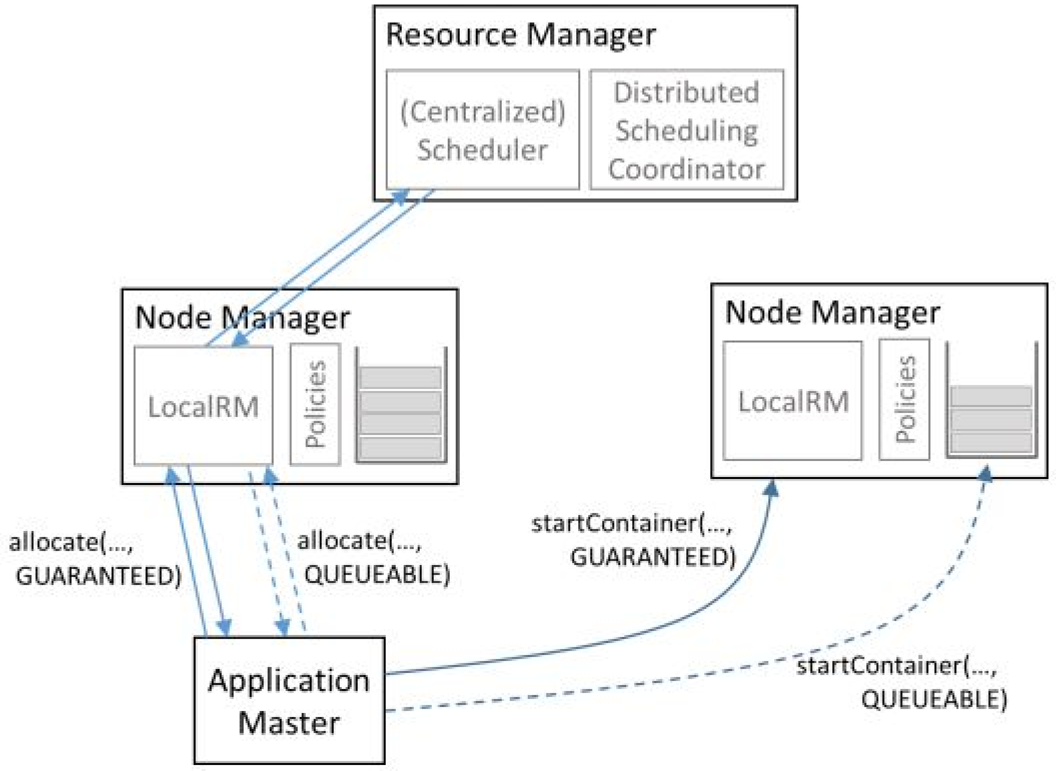
YARN Architecture overview :



- ❑ Core philosophy
  - ➢ Allocate resources very close to the data.
    - ● supports each RR to specify locality information
    - ● Supports delayed scheduling to ensure locality of data
  - ➢ Containers are primarily considered as non-preemptable.
    - ● During all kind of failovers priority is given to ensure that running containers continue to finish
    - ● Even during preemption we try to provide opportunity(time window) for the app to finish or checkpoint the containers state.

Overview of YARN

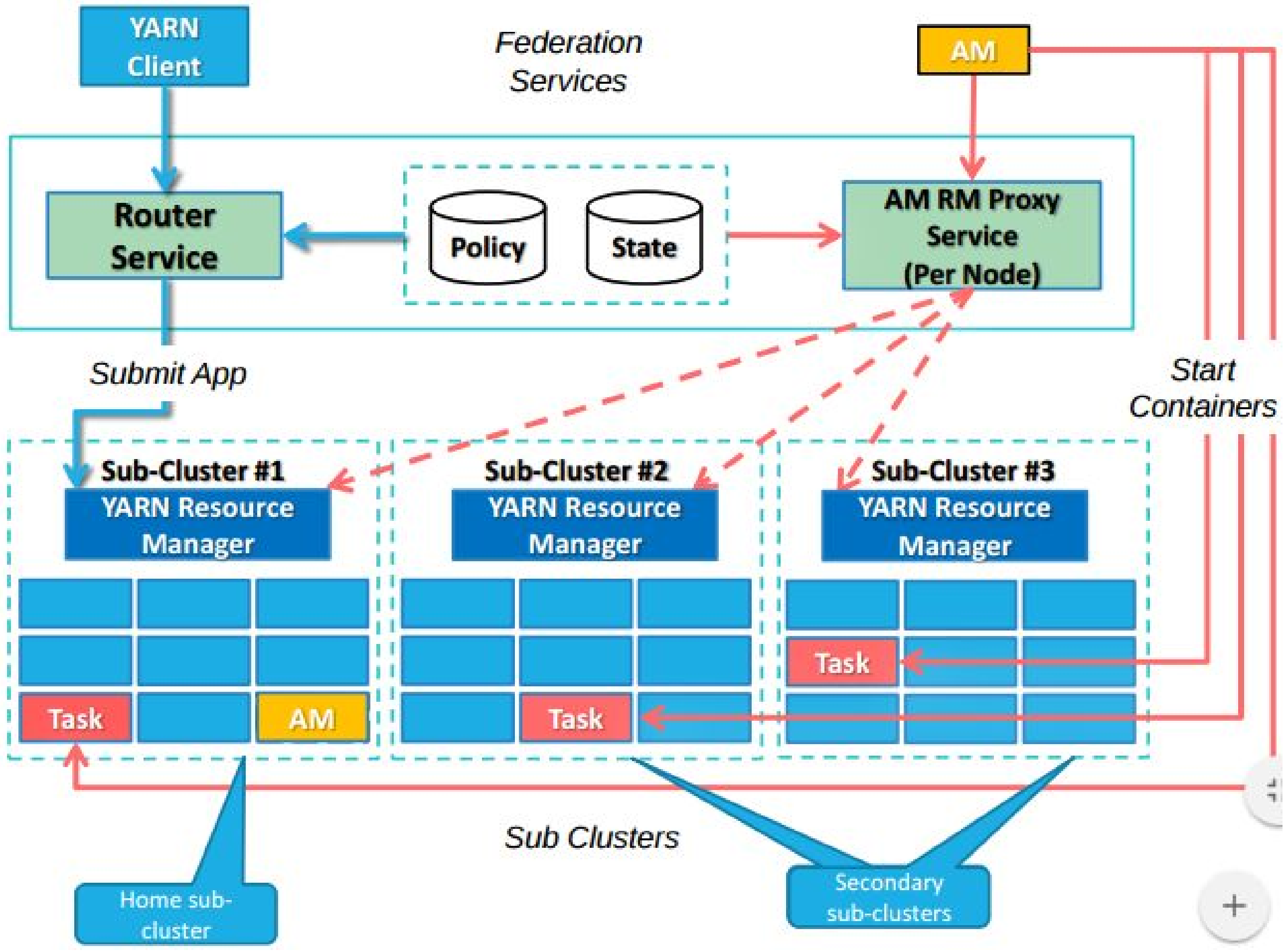
YARN Key features : Distributed Scheduling : (YARN-2877)



- ❑ Distributed + Centralized = achieves faster scheduling for small tasks without obstructing the application/queue/tenant related guarantees.
- ❑ Each NM is considered to have resource slots and resource requests are queued up.
- ❑ NM proxies the AM-RM communication and decorates the request and sends to RM.
- ❑ Distributed scheduling co-ordinator of RM sends and appends cluster stats information (all NM queued up resource requests information) to the AM-RM communication response (allocate call).
- ❑ NM on receiving the stats can schedule the opportunistic containers requested by the app based on policy
- ❑ Pluggable policy to pick the node effectively
- ❑ At NM priority is given to start the containers allocated by RM and if free picks from the opportunistic containers queue

Overview of YARN

YARN Key features : Federated Scheduling : (YARN-2915)



- ❑ A large YARN cluster is broken up into multiple small subclusters with a few thousand nodes each. Sub clusters can be added or removed.
- ❑ Router Service
  - Exposes ApplicationClientProtocol. Transparently hides existence of multiple RMs' in subclusters.
  - Application is submitted to Router.
  - Stateless, scalable service.
- ❑ AM-RM Proxy Service
  - Implements ApplicationMasterProtocol. Acts as a proxy to YARN RM.
  - Allows application to span across multiple sub-clusters
  - Runs in NodeManager.
- ❑ Policy and State store
  - Zookeeper/DB.

Overview of YARN

YARN Key features : YARN supports Docker ! (YARN-3611)

```
hadoop jar $HADOOP_PREFIX/share/hadoop/mapreduce/hadoop-mapreduce-examples-2.8.0.jar \
  teragen \
     -Dmapreduce.map.env="yarn.nodemanager.docker-container-executor.image-name=sequenceiq/hadoop-docker:2.4.1" \
   -Dyarn.app.mapreduce.am.env="yarn.nodemanager.docker-container-executor.image-name=sequenceiq/hadoop-docker:2.4.1" \
  1000 \
  teragen_out_dir
```

- ❑ Limited support in the released version(2.8) but 2.9 more features are expected to come
- ❑ supports cgroups resource isolation for docker containers.
- ❑ Supports multiple networks while launching but port Mapping to host port is yet to be done.
- ❑ Supports individual task/request to select to be run in docker container environment.
- ❑ By design can support other Container runtime environments but current support is only for docker
- ❑ Does not support launching of docker containers in Secured environment yet.
- ❑ Does not support mounting of external volumes yet.

Overview of YARN

YARN Key features : Rich Placement Constraints in YARN (YARN-6592)

- ➢ AllocationRequestID
- ➢ Priority
- ➢ AllocationTags: tags to be associated with all allocations returned by this SchedulingRequest
- ➢ ResourceSizing
    - ○ Number of allocations
    - ○ Size of each allocation
- ➢ Placement Constraint Expression

Scheduling Request

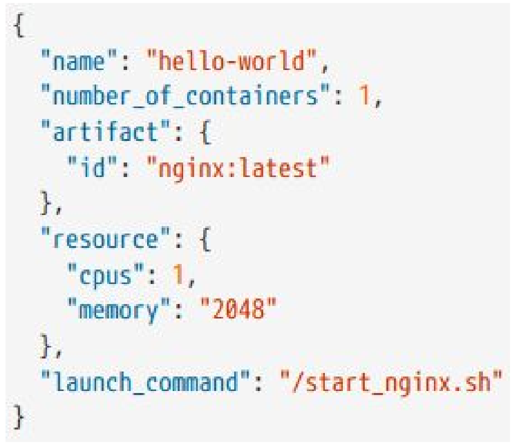
```
{
        Priority: 1,
        Sizing: {Resource: <8G, 4vcores>, NumAllocations: 1 },
        AllocationTags: ["hbase-rs"],
        PlacementConstraintExpression: {
           AND: [ // Anti-affinity between RegionServers
                {Target: allocation-tag NOT_IN “hbase-rs”, Scope: host },
                // Allow at most 2 RegionServers per failure-domain/rack
                { MaxCardinality: 2, Scope: failure_domain }
           ]
        }
},
```

Sample scheduling Request

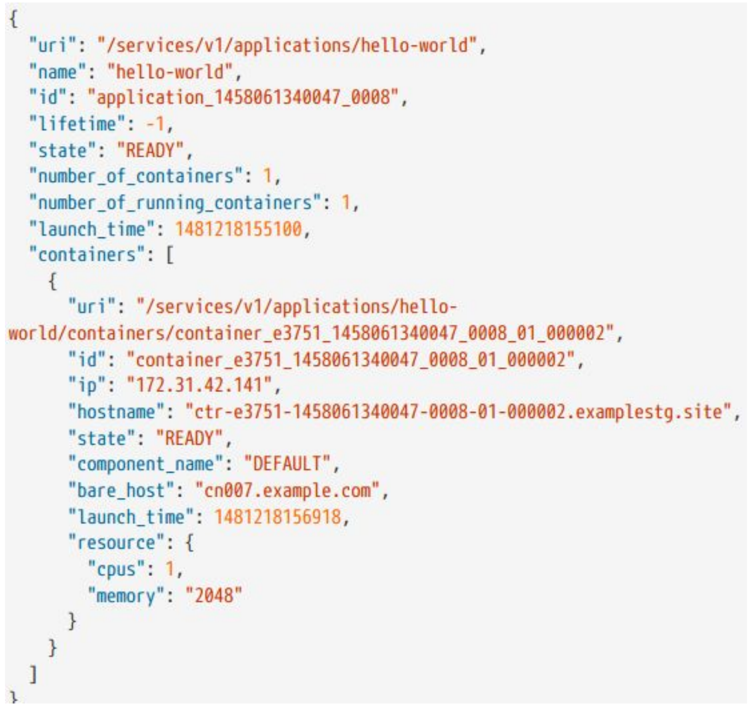
Overview of YARN

YARN Key features : Simplified API layer for services (YARN-4793)

POST URL - http://host.mycompany.com:8088/services/v1/applications



```
{
  "name": "hello-world",
  "number_of_containers": 1,
  "artifact": {
    "id": "nginx:latest"
  },
  "resource": {
    "cpus": 1,
    "memory": "2048"
  },
  "launch_command": "/start_nginx.sh"
}
```



```
{
  "uri": "/services/v1/applications/hello-world",
  "name": "hello-world",
  "id": "application_1458061340047_0008",
  "lifetime": -1,
  "state": "READY",
  "number_of_containers": 1,
  "number_of_running_containers": 1,
  "launch_time": 1481218155100,
  "containers": [
    {
      "uri": "/services/v1/applications/hello-
world/containers/container_e3751_1458061340047_0008_01_000002",
      "id": "container_e3751_1458061340047_0008_01_000002",
      "ip": "172.31.42.141",
      "hostname": "ctr-e3751-1458061340047-0008-01-000002.examplestg.site",
      "state": "READY",
      "component_name": "DEFAULT",
      "bare_host": "cn007.example.com",
      "launch_time": 1481218156918,
      "resource": {
        "cpus": 1,
        "memory": "2048"
      }
    }
  ]
}
```

GET URL - http://host.mycompany.com:8088/services/v1/applications/hello-world

- ❑ Create and manage the lifecycle of YARN services by new services API layer backed by REST interfaces.
- ❑ Supports for both simple single component and complex multi-component assemblies
- ❑ Other important complementing features :
  - ➢ Resource-profile management (YARN-3926),
  - ➢ service-discovery (YARN-913/YARN-4757).
  - ➢ REST APIs for application-submission and management (YARN-1695).
  - ➢ Support of System(daemon) services. YARN-1593

Agenda

- ❑Aspects of Distributed Scheduling Framework
- ❑Architectural evolution of resource scheduling
- ❑Overview of prominent open source schedulers
- ❑**Functional comparison between prominent schedulers**
- ❑Conclusion

Functional comparison between prominent schedulers

Feature / Framework	K8s	Mesos	YARN	Swarm
Architecture	Monolithic (shared state on support of multi scheduler)	two-level	monolithic/ two-level / hybrid	monolithic
Resource granularity	Multi dimensional	Multi dimensional	RAM/CPU (Multi dimensional after resource profile)	Multi dimensional
Multiple Scheduler support	on going	Yes - frameworks can further schedule	Partial (fair / capacity) not at the same time but apps can have their logic.	No
Priority preemption	Yes	Ongoing	Yes (further optimizations are on going YARN-2009)	No
Over subscription	Yes	Yes	Ongoing (YARN-1011)	No
Resource Estimation	No	No	Solutions being devloped as external components but supports **reservation queues**	No
Resource Isolation	Partial (but pluggable)	Partial (but pluggable)	Partial (but pluggable)	No

Functional comparison between prominent schedulers

Feature / Framework	K8s	Mesos	YARN
Support for Coarse grained isolation (partitions / pools)	N (Namespaces : logical partitions)	N (supports logical pools)	Supports partitions
Support multiple Container runtimes	Yes	Predominantly dockers	Partial Dockers (will be available in 2.9) but pluggable interface
Support Variety of applications	Yes but stateful application support is on going. Supports concept of PODS,Daemon services ..	Framework level support Support pods aka task groups.	Ongoing support for simplifying services. Pod concept not supported
Security	Supports pluggable authentication and SSL. Supports Kerberos	Supports CRAM-MD5 authentication using Cyrus SASL library. Pluggable authentication work is ongoing	Supports SSL, Kerberos
Disk Volumes provisioning	Yes	Yes	No

Functional comparison between prominent schedulers

Feature / Framework	K8s	Mesos	YARN
Disk Volumes provisioning	Yes	Yes	No
Scalability and Reliability	SPOC as there is single process which holds the whole state, And possible load on ETCD as cluster size increases	Good as the state is distributed across multiple frameworks	Good, separation between app and resource data
Suitable for Cloud	Yes	Yes	Fairly
Suitable for standalone BigData	Ongoing	Yes	Yes

Agenda

- ❑Aspects of Distributed Scheduling Framework
- ❑Architectural evolution of resource scheduling
- ❑Overview of prominent open source schedulers
- ❑Functional comparison between prominent schedulers
- ❑**Conclusion**

Conclusion

- ❑ All schedulers are in fact trying to solve the same set of problems, duplicating effort building various shapes and sizes of resource managers, container managers or long-running service schedulers.
- ❑ It will lead to a fragmented experience with different terminology and concepts for very similar things, different interfaces or APIs, different troubleshooting procedures , documentation and so on, which will only be driving up operations costs.

References

- YARN - https://issues.apache.org/jira/browse/YARN/ & http://hadoop.apache.org/docs/current/
- Swarm scheduling - https://www.slideshare.net/AtharvaChauthaiwale/docker-swarm-scheduling-in-112
- Kubernetes Design - https://github.com/kubernetes/community/blob/master/contributors/design-proposals/architecture.md
- Mesos Architecture - http://mesos.apache.org/documentation/latest/architecture/
- Kubernetes Scheduler - http://stackoverflow.com/questions/28857993/how-does-kubernetes-scheduler-work
- Mesos Scheduler - http://cloudarchitectmusings.com/2015/04/08/playing-traffic-cop-resource-allocation-in-apache-mesos/
- Kubernetes – Marathon - https://www.quora.com/What-is-the-difference-between-Googles-Kubernetes-and-Mesospheres-Marathon
- Scheduler architectures : https://www.cl.cam.ac.uk/research/srg/netos/camsas/blog/2016-03-09-scheduler-architectures.html#fn3l
- CONTAINER ORCHESTRATION COMPARISON GUIDE - https://apprenda.com/thank-you/gaw-container-orchestration-comparison-guide-k/
- Future of Resource Manager by IBM - https://www.ibm.com/developerworks/community/blogs/1ba56fe3-efad-432f-a1ab-58ba3910b073/entry/thoughts_on_future_of_resource_managers_and_schedulers_in_the_cloud?lang=en

and many more ...

Thank You !