Kubernetes & YARN: a hybrid container cloud

Jian He & Bushuang Gao
About us

• Jian He
  • Staff Engineer @Alibaba cluster management team
  • Staff Engineer @Hortonworks
  • Hadoop Committer & Project Management Committee member

• Bushuang Gao
  • Senior Engineer @Alibaba
Agenda

- What/Why co-location
- Co-location @ Alibaba
- Kubernetes & YARN: a hybrid architecture
- Resource Isolation
- Kubernetes vs YARN
- Future
What/Why co-location
Gartner has long talked about the "80% rule": that 80 percent of IT budgets get spent simply "keeping the lights on".

- The average data center CPU utilization is about 10%.
Workloads in Alibaba data center

- Service
  - Commercial platform, taobao, Tmall, retail apps, search,
  - Payment platform: Alipay
  - Middleware: queuing service, cache system, database

- Job
  - Big data batch jobs: MR, spark
  - Realtime: flink
Sigma: Alibaba cluster management system for online services
Fuxi: Alibaba cluster management system for batch jobs
## Services vs Jobs

<table>
<thead>
<tr>
<th></th>
<th>Online service</th>
<th>Batch job</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>Online shopping website, payment system</td>
<td>MR, spark, flink</td>
</tr>
<tr>
<td>Latency</td>
<td>Sensitive</td>
<td>Insensitive</td>
</tr>
<tr>
<td>Priority</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>Traffic pattern</td>
<td>Peak during daytime and low during night</td>
<td>Peak during night and low during daytime</td>
</tr>
<tr>
<td>Fault tolerance</td>
<td>should not fail, high availability</td>
<td>Fail and retry</td>
</tr>
</tbody>
</table>

**Complementary!**
Now: co-located clusters

Sigma: Alibaba cluster management system for online services
Fuxi: Alibaba cluster management system for batch jobs
Google Borg

Borg paper mentions 20% - 30% more machines if segregating prod and non-prod workloads.
What is co-location

- A hybrid deployment that supports both online services and offline jobs using the same hardware.
Key concepts in co-location

• **Resource sharing**: Make services and jobs share a single cluster and have them run on the same hardware

• **Resource Isolation**: Rely on intelligent scheduling and resource isolation to guarantee the SLA.
Why co-location

• Two separated cluster incur huge resource wastage. We need to increase resource utilization.

• 1 hour peak traffic on double 11 requires a lot of extra resources. However, those resources will be idle after the peak time.

• Lend those resources to computation jobs
Co-location @Alibaba
Sigma: Alibaba cluster management system for online services
Fuxi: Alibaba cluster management system for batch jobs
Co-located vs separated cluster CPU utilization

Co-located: 40%
Separated: 10%
RT Impact on online service within 5%
Two core concepts in co-location

Scheduling
Efficient placement of service container and tasks

Isolation
When placed together, don’t affect each other
Scheduling

• How to make each system schedule efficiently
• Service: optimal decision with more scheduling time
  • Spreading, anti-affinity scheduling.
• Job: sub-optimal decision with less scheduling time.
  • Relax locality
Together: time sliced scheduling

- Online workload low 1:00am – 6:00am
- Offline jobs scale up while online workload remains idle
- Offline jobs scale down while online workload comes back
Normal day <-> Sales day

- **Online service**
- **Offline job**

Co-located cluster

Separated cluster

- Normal day
- 1 hour switch
- Sales day
Kubernetes & YARN: a hybrid architecture
resource sharing

Online service usage

Kubernetes

Offline job resource usage

YARN

Online service resource quota

Offline job resource quota
resource sharing

Online service usage
Buffer
Offline job resource usage

Kubernetes

Over-subscription

YARN

Online service resource quota
Offline job resource quota
resource sharing

Online service usage

Kubernetes

Offline job resource usage

YARN

Online service resource quota

Offline job resource quota
Isolation
Interferences
CPU Scheduler interference

- Affect two tasks on the same core
- Task preempt (Alikernal feature)
  - Online service can preempt offline job more easily
- cpu.shares
  - Online service has more shares than offline job
HT Interference

- 2 HT on two logic cores share L1-d/L1-i/L2/TLB Cache
- Eliminate the need to BIND the online service and offline jobs to exclusive cores

<table>
<thead>
<tr>
<th>Core</th>
<th>Neighbor HT</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Idle</td>
<td>job run</td>
</tr>
<tr>
<td>Idle</td>
<td>Service run</td>
<td>Job not run</td>
</tr>
<tr>
<td>Job run</td>
<td>Service wakeup</td>
<td>Job leave</td>
</tr>
</tbody>
</table>
CPU

Online service: Exclusive core(0-3)

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Online service: shared core(4-15); cpu.share 1024*8

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Offline job: shared core(4-15); cpu.share 1024*2

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Offline job: shared core(0-15) cpu.share 2
Memory Isolation

- Memory.priority
  - memory reclaim for online service will be less than that for offline job
- memory.use_priority_oom
  - Kill lower priority offline jobs when OOM
- memcg wmark_ratio
  - Similar to kswapd, when memory hits wmark_ratio, start memory reclamation
Kubernetes vs YARN
# Design Principal

<table>
<thead>
<tr>
<th>Kubernetes</th>
<th>Focus on scheduling long running service. Driving current state towards desired state with control loops. Ideal for automated daily operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN</td>
<td>Focus on scheduling batch Jobs Has first-class ‘application’ concept. Richful hierarchical queue model for resource quota management Bare container, no POD concept</td>
</tr>
</tbody>
</table>
## API Design

<table>
<thead>
<tr>
<th>Kubernetes</th>
<th>CRUD API is mostly singleton. I.e. Pod Rich command line tools for daily operations.</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN</td>
<td>CRUD API is batchful. I.e. Create a list of containers</td>
</tr>
</tbody>
</table>
## Scheduling Unit

<table>
<thead>
<tr>
<th>Kubernetes</th>
<th>Container centric – bottom up. Container is the primitive. Other primitives such as replicaset, deployment are built around containers.</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN</td>
<td>Application centric: top down. Scheduling sequence: Queue -&gt; user -&gt; application -&gt; container request</td>
</tr>
</tbody>
</table>
## Communication

| Kubernetes          | Based on api-server watch mechanism  
<table>
<thead>
<tr>
<th></th>
<th>Everything stored in etcd</th>
</tr>
</thead>
</table>
| YARN                | Based on RPC                         
|                     | Only application-level metadata persisted.  
|                     | Container data is not persisted.      
|                     | Recover from in-memory state from peers |
## Container Runtime

<table>
<thead>
<tr>
<th>kubernetes</th>
<th>CRI compatible. Docker etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>YARN</td>
<td>Docker + TAR ball</td>
</tr>
</tbody>
</table>
Larger scale

- More resource dimension
- Improve stability at large scale
- Expand Alibaba co-location scale (Fuxi & sigma)
Intelligent auto-rebalancing

• Learn online service and offline jobs patterns and do prediction to auto-rebalance.
• Flex service and job cluster scale more efficiently on sales day events.
More

- Consolidate the colocation technologies @Alibaba and apply those in kubernetes & YARN co-location project
Thanks !