Mercury: Hybrid Centralized and Distributed Scheduling in Large Shared Clusters

Konstantinos Karanasos Sriram Rao Carlo Curino Chris Douglas
Kishore Chaliparambil Giovanni Matteo Fumarola Solom Heddaya
Raghu Ramakrishnan Sarvesh Sakalanaga
Microsoft Corporation
{kokarana, sriramra, ccurino, cdoug, kishorec, gifuma, solomh, raghu, ssakala}@microsoft.com

Datacenter-scale computing for analytics workloads is increasingly common. High operational costs force heterogeneous applications to share clusters for achieving economy of scale. Scheduling such large and diverse workloads is inherently hard, and existing approaches tackle this in two alternative ways: 1) centralized solutions offer strict enforcement of scheduling invariants (e.g., fairness, capacity) for heterogeneous applications, 2) distributed solutions offer scalable, efficient scheduling for homogeneous applications.

We argue that these solutions are complementary, and advocate a blended approach. Concretely, we propose Mercury, a hybrid resource management framework that supports the full spectrum of scheduling, from centralized to distributed. Mercury exposes a programmatic interface that allows applications to trade-off between scheduling overhead and execution guarantees. Our framework harnesses this flexibility by opportunistically utilizing resources to improve task throughput. Experimental results on production-derived workloads show gains of over 35% in task throughput. These benefits can be translated by appropriate policies into job throughput or job latency improvements. We have implemented and are currently contributing Mercury as an extension of Apache Hadoop/YARN.

Below we briefly describe Mercury. More details can be found in our technical report.

Mercury Design The most critical component of our system is the Mercury Resource Management Framework, which includes a central scheduler running on a dedicated node, and a set of distributed schedulers running on (possibly a subset of) the cluster nodes. This combination of schedulers performs cluster-wide resource allocation to jobs for the same underlying pool of resources. Mercury uses two types of allocation units (or containers): GUARANTEED and QUEUEABLE. The former, allocated by the central scheduler, offer execution guarantees and more careful placement. The latter, allocated by one of the distributed schedulers, offer lower allocation latency but no execution guarantees (they can be killed by GUARANTEED containers).

Framework policies These policies determine all scheduling decisions in Mercury and can be divided into three categories. Invariants enforcement policies impose global scheduling invariants, including capacity/fairness for the GUARANTEED containers and quotas for the QUEUEABLE ones. Placement policies map requests to available resources. Finally, load shaping policies maximize cluster efficiency by dynamically re-balancing load across nodes, reordering the tasks within a node’s queue, etc.

Application policies Each application implements a policy that determines the desired type of container for each task. This allows applications to tune their scheduling needs from fully-centralized to fully-distributed scheduling (and any combination in between). Information including the type of the job, the estimated task duration and the job progress can be exploited by these policies.

Experimental results We have deployed Mercury on a cluster of 250 machines, and have evaluated it against various workloads, both synthetic and production-derived from Microsoft clusters. When compared to stock YARN, Mercury achieves a task throughput improvement from 12 to 45% depending on the workload. Our policies can translate task throughput gains into improved job throughput (36.3% gain), as well as improved job latency for 80% of the jobs. For the production-derived workloads Mercury leads to a 35% task throughput improvement.
Mercury: Hybrid Centralized and Distributed Scheduling in Large Shared Clusters

Konstantinos Karanasos, Sriram Rao, Carlo Curino, Chris Douglas, Kishore Chaliparambil, Giovanni Matteo Fumarola, Solom Heddaya, Raghu Ramakrishnan, Sarvesh Sakalanaga

Motivation
- Cloud-scale shared clusters
- Heterogeneity of workloads
  - Task duration in [0.1, 10K+] sec
- Scheduling latency
  - 50% of tasks in Cosmos < 10sec
- Resource predictability
  - Crucial for SLA-jobs
- Sharing policies
  - Improve utilization/ROI

Key Insight
- Centralized and Distributed schedulers are complimentary
  - Mercury follows a hybrid approach
  - “Trade performance guarantees for allocation latency”
  - Expose API for applications to choose scheduling type per task based on their needs

Resource Types
- GUARANTEED resources:
  - Allocated by Central Scheduler
  - Guaranteed to start and complete
- QUEUEABLE resources:
  - Allocated by Distributed Scheduler
  - Executed opportunistically
  - Can be queued/preempted/killed

Framework Policies
- Invariants enforcement
  - Capacity, fairness, quotas
- Placement policies
  - Dynamic load rebalancing
  - Queue reordering
- Load shaping policies
  - Resource policing

App Policies
- Determine container type for each task of a job
  - Currently based on runtime
  - hybrid-GQ (td, pg)
  - More involved policies
    - Consider type of task, job progress, cluster load

Implementation/OSS
- Implemented on top of Apache YARN
- YARN RM as the Central Scheduler
- Added a Distributed Scheduler per node
- Leverage existing framework policies
- Added distributed-related policies
- Open source: Apache JIRA YARN-2877

Experimental Results
- 250-node cluster
- Synthetic and Microsoft-based workloads
- Using (modified) YARN, Tez and Gridmix

Up to 41.4% task throughput gain

Up to 66% mean job latency gain

kokarana@microsoft.com
Cloud and Information Services Lab (CISL)

Microsoft