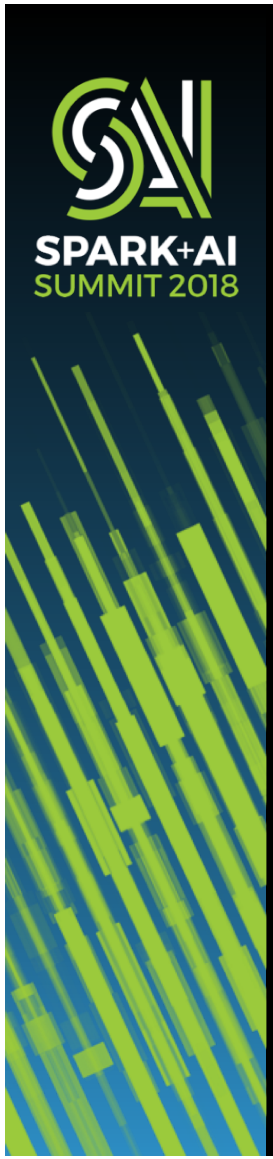
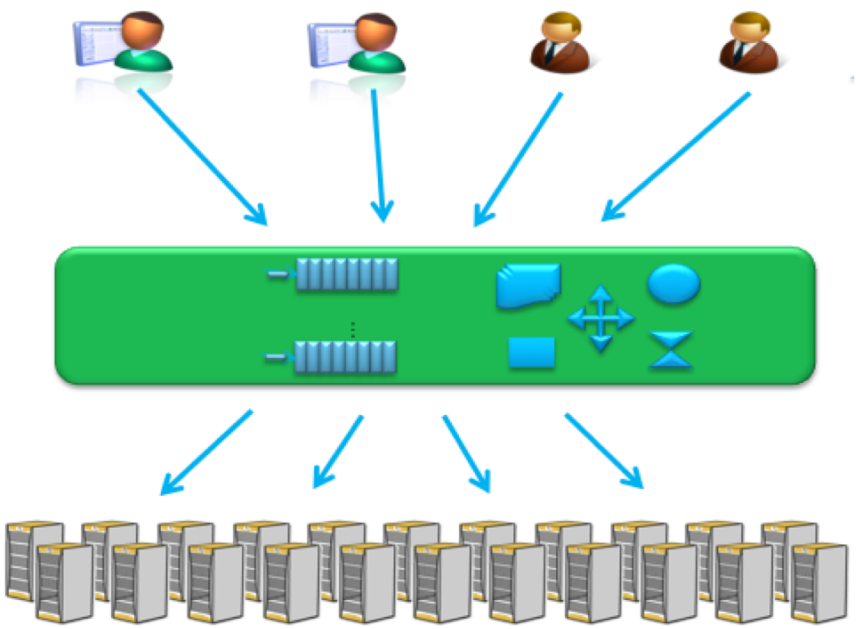


Dynamic Priorities for Apache Spark Application's Resource Allocations

Michael Feiman, IBM
Shinnosuke Okada, IBM

#Dev6SAIS

SLA for “Spark as Service” use case



- Spark as a Service:
  - Multiple tenants submit multiple Spark applications
- Service Level Agreement (**SLA**)
  - From service provider: **required resources** (CPU, memory, network, IO, etc.) are constrained mostly by cost
  - From user: **required runtime** to finish specific application or data processing flow

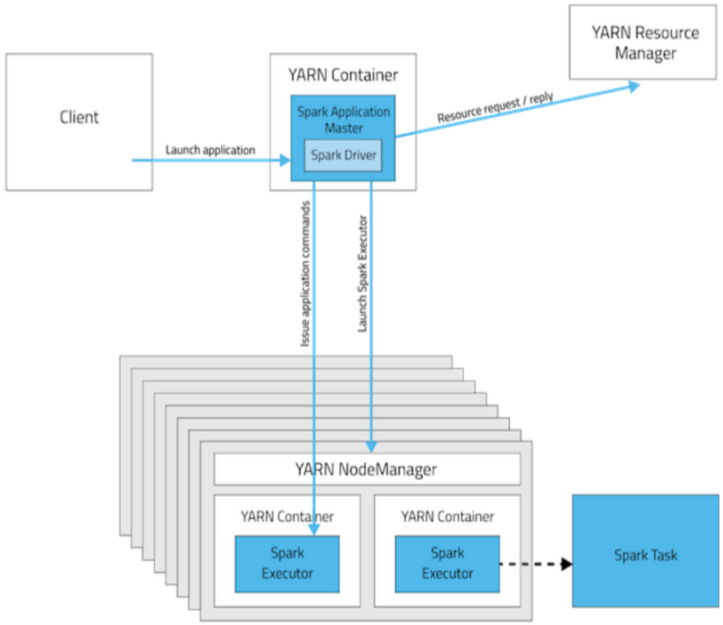
Scheduling priorities use case

- Scheduling priorities can be used as a way to *reconcile* user's SLA requirements (runtime, deadline, etc.) with SLA in terms of resource provider (CPU, memory, IO, etc.)
- Workload runtime is affected by multiple factors specific to both applications and environments that are *changing overtime, so* it is hard to predict priority based on required SLA when applications are submitted
- The user or SLA automating system should be able to *modify priority dynamically* for submitted applications at any stage in its life-cycle (pending, running, suspended, etc.)

Use case: Mission-critical applications use priority:

1. For **pending** applications - to **"*jump the queue*"** and startup earlier
2. For **running** applications - to get more resources so the applications finish earlier to **"*meet deadline*"**

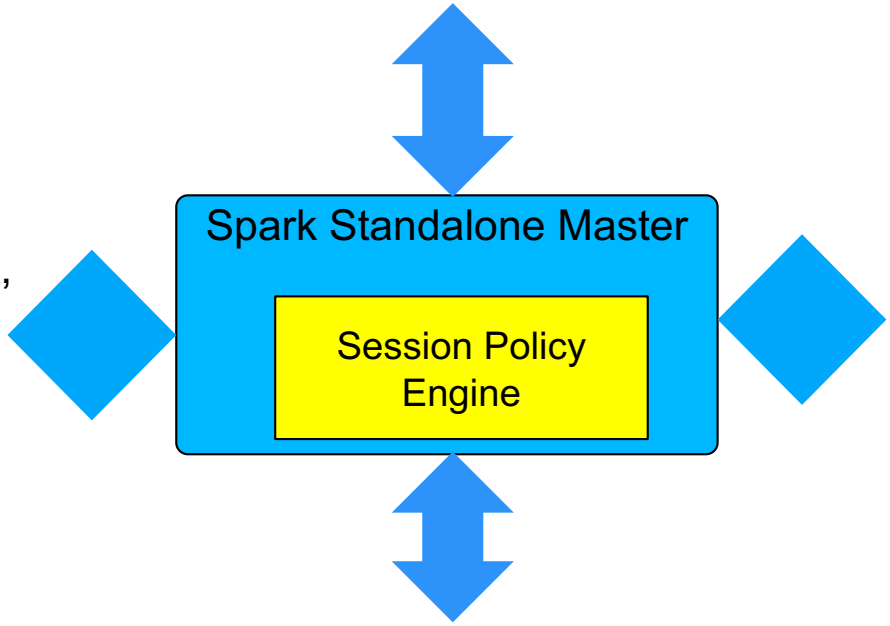
Scheduling across Spark applications



- **Standalone mode:** By default, applications submitted to the standalone mode cluster will run in first-in, first-out (**FIFO**) order. No priorities.
- **Mesos:** Uses static partitioning when **spark.mesos.coarse=true**, and users can optionally set **spark.cores.max** to limit each application's resource share as in the standalone mode. No priorities.
- **YARN:** The **--num-executors** controls how many executors will be allocated with **--executor-memory** and **--executor-cores** controls the resources per executor. Optional **Dynamic Resource Allocation** - Resources requested by a set of *heuristics*. Driver requests executors in rounds. The actual request is triggered when there have been *pending tasks for a period of time*; the number of executors requested in each round *increases exponentially* from the previous round.
- **YARN Fair Scheduler:** Priority only by queue.
- **YARN Capacity Scheduler:** Only for FIFO policy.

What is session scheduler?

Job Submission Interface:
Spark-Shell, Spark-submit, RESTful APIs, etc.

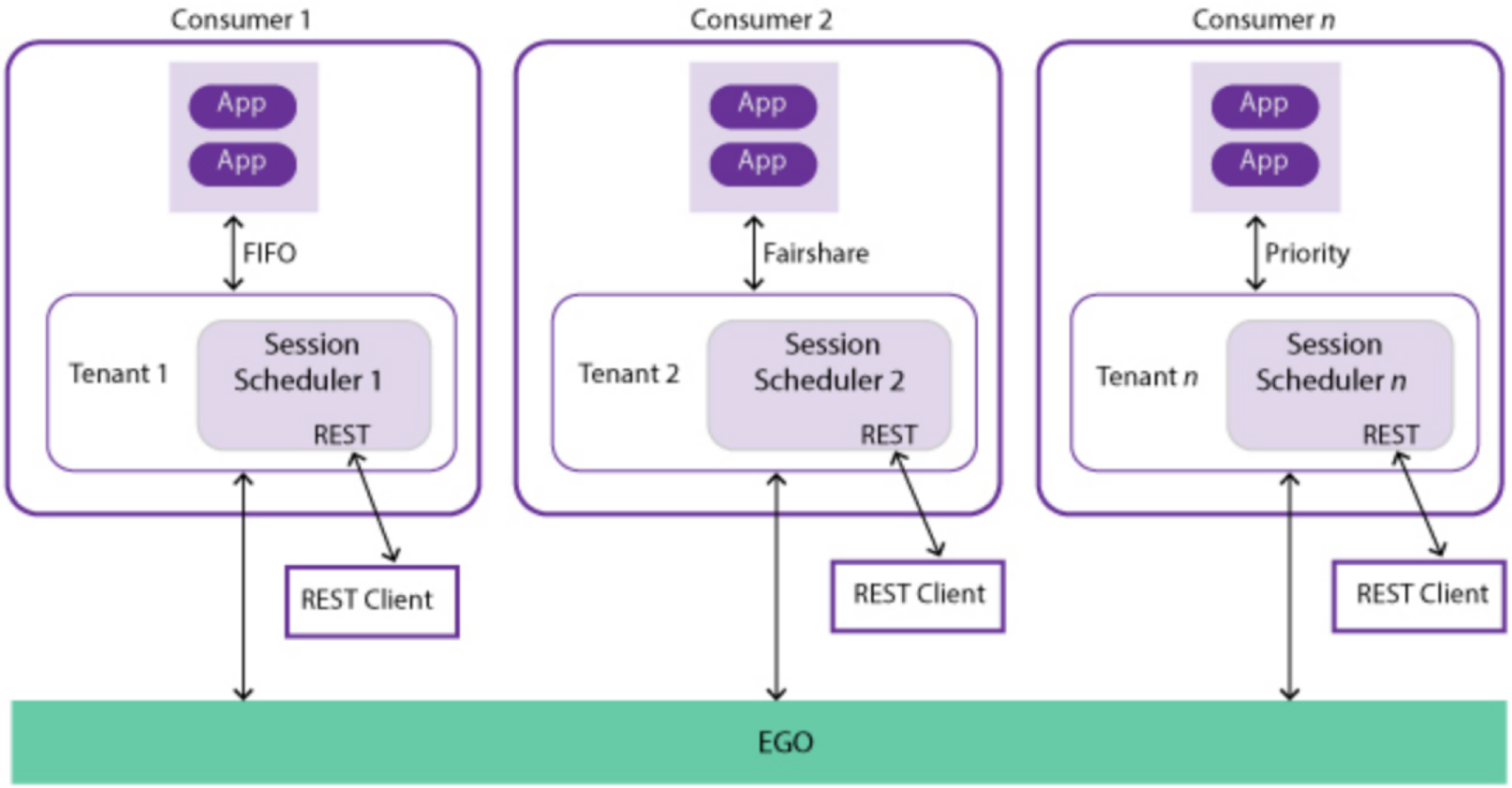


Execution:
Execution plan for driver/executor/task, Flex up/down, Preemption

Metrics:
Monitor, Analytics, Usage
Policy:
Priority, Share Ratio, Plan

Interface to Resource Manager

Two scheduling layers in IBM Spectrum Conductor

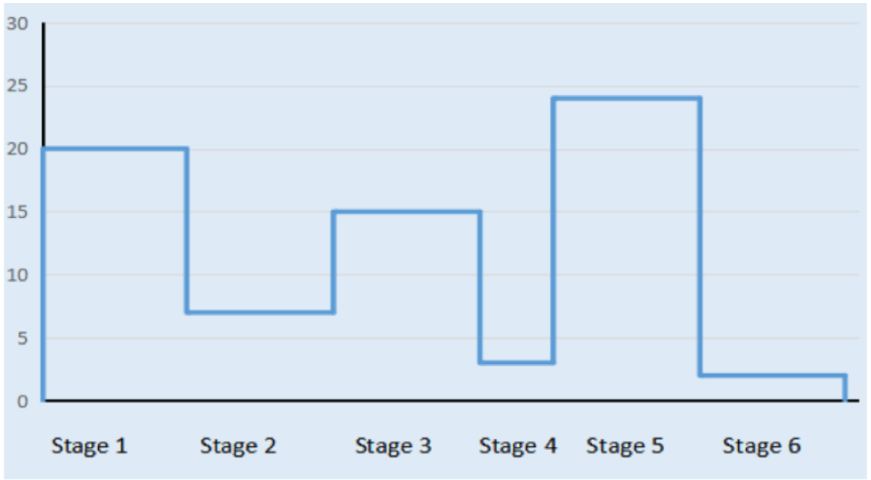


Workload Management layer

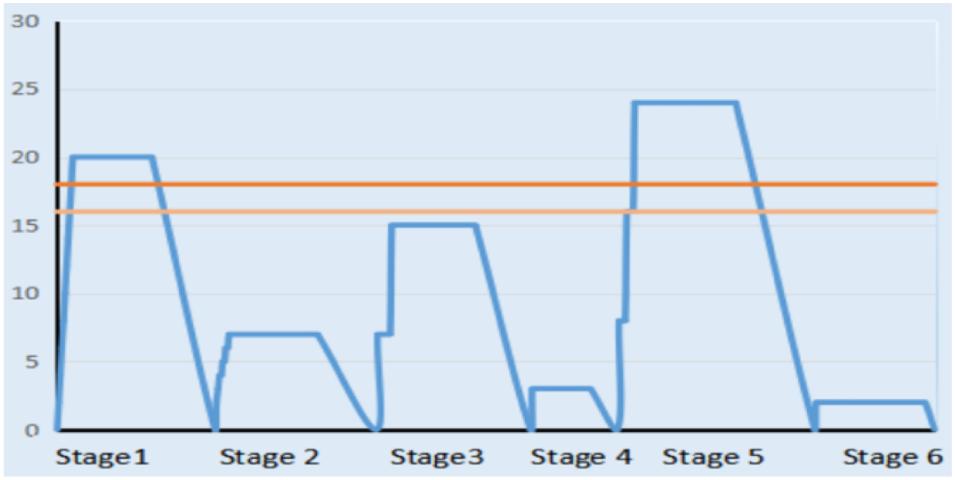
Resource Management layer

Fine-grain Spark scheduling

- Each application stage requires different number of tasks
- Driver needs to dynamically adjust the request for resources in each stage
- Session scheduler aggregates current total demand to resource manager
- Session scheduler assigns resources to application dynamically based on driver's demand, policy and application priority



Ideal tasks demand



Actual resources been assigned to application

Spark session scheduler policies extension

New Spark property *spark.ego.app.schedule.policy* can take 3 values:

1. ***fifo*** - **First-in, first-out scheduling:** meets the requirements of applications submitted earlier first. Applications that are submitted earlier get all the demanded resources that the scheduler can provide. Applications that are submitted later get the remaining resources.

2. ***priority*** - **Priority scheduling policy:** priority scheduling meets the requirements of higher priority applications and applications submitted earlier first. With this policy, before offering executor resources, applications are sorted by priority and submission time. The scheduler allocates resources first by priority, followed by submission time, which means that a higher priority application gets all demanded resources that the scheduler can provide. Lower priority applications get the remaining resources.

3. ***fairshare*** - **Fair-share scheduling:** shares the entire resource pool among all submitted applications. Applications get their own deserved number of resources according to their priority.

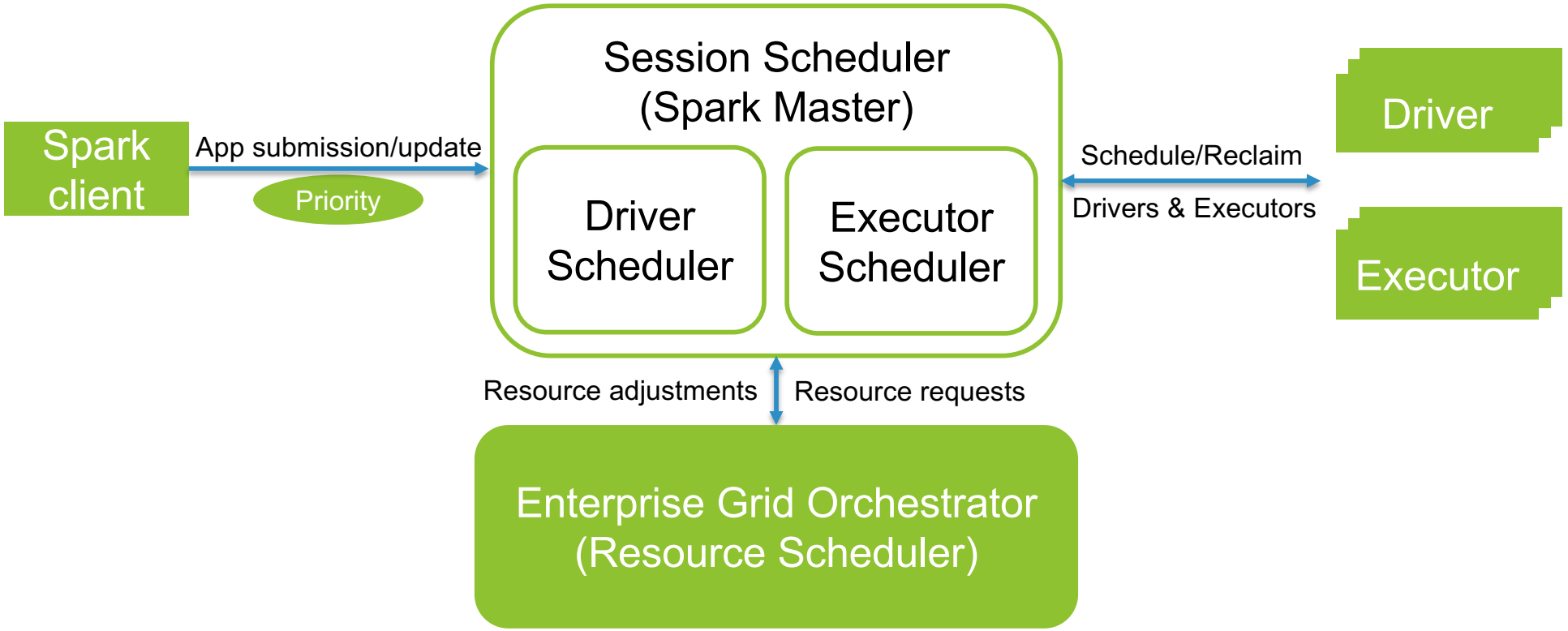
Priorities range, limits, and reclaim

- **Priority**: **spark.ego.priority** property
  - Integer value in range from **1** to **10000**
  - Default 5000

- **Reclaim**: When a higher priority application is submitted or a higher priority application's demand increases, the scheduler reclaims resources from other applications to meet the requirements of the higher priority application. *Reclaim timeout* can be configured per session scheduler.

- **Limits**: The number of tasks in different stages of a job can fluctuate widely. High-priority jobs or jobs with an earlier submission time might want to get more resources while the resources are occupied by low priority jobs. To control behavior, users can still tune the **spark.ego.slots.max** parameter to control the maximum number of slots each application can get (comparable to *spark.executor.cores*)

Implementation details



Priorities for Spark drivers

- Driver's resource allocation adjustments are expected to follow the first-in, first-out (FIFO) time sequence.
  - Drivers are expected to be *non-pre-emptive*
  - SLA for the drivers can control only the driver's *start time*
  - Dynamic application priority is used to adjust the *sorting order* in the driver's pending queue

Priorities for Spark application's tasks

- The Spark application priority is translated by the executor scheduler to adjustments in resources allocated for Spark tasks between applications submitted to the same scheduler
  - The priorities translation algorithm is determined by a scheduling policy, so for each scheduling cycle:
    - **Priority policy** - applications are sorted by their priorities – the highest priority application is given all resource allocations that it currently requires for its tasks, followed by the next application in the priority order
    - **Fairshare policy** - Scheduler calculates the priority-weighed allocation shares among all submitted applications from the entire resource pool. If an application currently requires fewer tasks, then entitled by its share, remaining resources are returned to the shared pool and then shared between the rest of the applications that use the same sharing policy

Re-balancing and pre-emption

- Resource allocation adjustments on next scheduling cycle might result in *re-balancing* resource between applications.
- Pre-emptive or non pre-emptive release of resource is set in the r*eclaim timeout* value:
  1. *zero timeout* - Spark driver kills any running tasks and returns resources at once to the scheduler
  2. *positive timeout* - task will be canceled after the timeout value is expired
  3. *non-pre-emptive* waiting for task to finish before releasing resource

Client APIs for dynamic priority

- For application submission: standard RESTful POST API extended to include new optional field **"priority"** in its body with value 1-10000, default 5000
- For priority modification: extended with new RESTful POST API :

<Master-URL>/v1/submissions/**update**/<***driver-id/app-id***>

```
Body:
{
  "action" : "UpdateSubmissionRequest",
  "clientSparkVersion" : "2.2.0",
  "priority" : "10"
}
```

```
Response:
{
  "action" : "UpdateSubmissionResponse",
  "message" : "Update request for driver
driver-20170713144252-0001-0951b2f6-dab5-
4fef-972c-19bc62d2df09 submitted.",
  "sesssionSchedulerVersion": "1.0.0",
  "submissionId" : "driver-20170713144252-
0001-0951b2f6-dab5-4fef-972c-19bc62d2df09",
  "success" : true
}
```

- **Note**: When it is called with ***driver-id*** then priorities are updated for both associated driver and application. When it is called with ***app-id***, then only associated application priority is updated.

Authorization for dynamic priorities

- Authorization for the priority management is key for enterprise customers
- Need the following role-based permissions:
  - Authorized users to set and update priorities of their own applications (valid range 1-5000)
  - Admins to set and update priorities of everyone's applications (valid range 1-10000)

Auditing for dynamic priorities

- Admins can audit the usage of dynamic priorities from the Spark master log:

```
18/05/23 16:40:41 INFO EgoApplicationManager: Registering app pyspark-shell from sender
172.29.11.126:59612, reserve:0, priority:5000, limit:2147483647
18/05/23 16:40:41 INFO EgoApplicationManager: Registered app pyspark-shell with ID app-
20180523164041-0000-2b21c977-b9c0-4bcd-ba2c-f51a969ad360
...
18/05/23 16:41:55 WARN PolicyHierarchy: Admin updated the priority of app-
20180523164041-0000-2b21c977-b9c0-4bcd-ba2c-f51a969ad360 from 5000 to 10000 for GPU
resources.
18/05/23 16:41:55 WARN PolicyHierarchy: Admin updated the priority of app-
20180523164041-0000-2b21c977-b9c0-4bcd-ba2c-f51a969ad360 from 5000 to 10000 for CPU
resources.
...
```

Test case & demo

- IBM Spectrum Conductor with Spark v2.2.1
  - Spark 2.2.0
- Case 1: Driver scheduler
  - Only 1 driver slot in the cluster
  - 3 drivers are submitted
  - Update the driver priority to “jump the queue”
- Case 2: Executor scheduler
  - Fairshare scheduling with pre-emption enabled (timeout: 0)
  - 2 Jupyter notebooks
  - Update application priorities

References

- Dynamic priority feature in IBM Spectrum Conductor Knowledge Center - https://www.ibm.com/support/knowledgecenter/en/SSZU2E_2.2.1/developing_instances/priority.html
- IBM Spectrum Conductor in IBM Developer Works - https://developer.ibm.com/storage/products/ibm-spectrum-conductor-spark/

Q & A section

IBM Sessions at Spark+AI Summit 2018 (Tuesday, June 5)

Date, Time, Location & Duration	Session title and Speaker
Tue, June 5 | 11 AM 2010-2012, 30 mins	Productionizing Spark ML Pipelines with the Portable Format for Analytics Nick Pentreath (IBM)
Tue, June 5 | 2 PM 2018, 30 mins	Making PySpark Amazing—From Faster UDFs to Dependency Management and Graphing! Holden Karau (Google) Bryan Cutler (IBM)
Tue, June 5 | 2 PM Nook by 2001, 30 mins	Making Data and AI Accessible for All Armand Ruiz Gabernet (IBM)
Tue, June 5 | 2:40 PM 2002-2004, 30 mins	Cognitive Database: An Apache Spark-Based AI-Enabled Relational Database System Rajesh Bordawekar (IBM T.J. Watson Research Center)
Tue, June 5 | 3:20 PM 3016-3022, 30 mins	**Dynamic Priorities for Apache Spark Application's Resource Allocations Michael Feiman (IBM Spectrum Computing) Shinnosuke Okada (IBM Canada Ltd.)**
Tue, June 5 | 3:20 PM 2001-2005, 30 mins	Model Parallelism in Spark ML Cross-Validation Nick Pentreath (IBM) Bryan Cutler (IBM)
Tue, June 5 | 3:20 PM 2007, 30 mins	Serverless Machine Learning on Modern Hardware Using Apache Spark Patrick Stuedi (IBM)
Tue, June 5 | 5:40 PM 2002-2004, 30 mins	Create a Loyal Customer Base by Knowing Their Personality Using AI-Based Personality Recommendation Engine; Sourav Mazumder (IBM Analytics) Aradhna Tiwari (University of South Florida)
Tue, June 5 | 5:40 PM 2007, 30 mins	Transparent GPU Exploitation on Apache Spark Dr. Kazuaki Ishizaki (IBM) Madhusudanan Kandasamy (IBM)
Tue, June 5 | 5:40 PM 2009-2011, 30 mins	Apache Spark Based Hyper-Parameter Selection and Adaptive Model Tuning for Deep Neural Networks Yonggang Hu (IBM) Chao Xue (IBM)

IBM Sessions at Spark+AI Summit 2018 (Wednesday, June 6)

Date, Time, Location & Duration	Session title and Speaker
Wed, June 6 | 12:50 PM	Birds of a Feather: Apache Arrow in Spark and More Bryan Cutler (IBM) Li Jin (Two Sigma Investments, LP)
Wed, June 6 | 2 PM 2002-2004, 30 mins	Deep Learning for Recommender Systems Nick Pentreath (IBM))
Wed, June 6 | 3:20 PM 2018, 30 mins	Bringing an AI Ecosystem to the Domain Expert and Enterprise AI Developer Frederick Reiss (IBM) Vijay Bommireddipalli (IBM Center for Open-Source Data & AI Technologies)

Meet us at IBM booth in the Expo area.