Experiences Running Different Workload Managers on Cray Platforms

Haripriya Ayyalasomayajula
Karlon West

Agenda

- Purpose
- Value
- Introduction
- Workload Managers
- Workflows
- Comparison of workload managers
- Summary
- Q&A

Purpose

- **Workload managers**
  - Help to launch jobs on underlying resources
  - Provide
    - Resource Management
    - Scheduling
- **Goals**
  - Achieve efficient resource utilization while still meeting scalability and scheduling requirements

Value

- **Important to understand the differences in the nature of workload managers**
- **Gathering requirements for future Cray architectures**
  - Run both Analytics and HPC workloads on the same platform
    - What features do we want?
    - How can we get them?

Workload management

- **Resource management**
  - Negotiation of managed resources that are required for running a job
- **Scheduling**
  - Policy by which tasks of a job are launched on the allocated resources

Workload managers on Cray Platforms

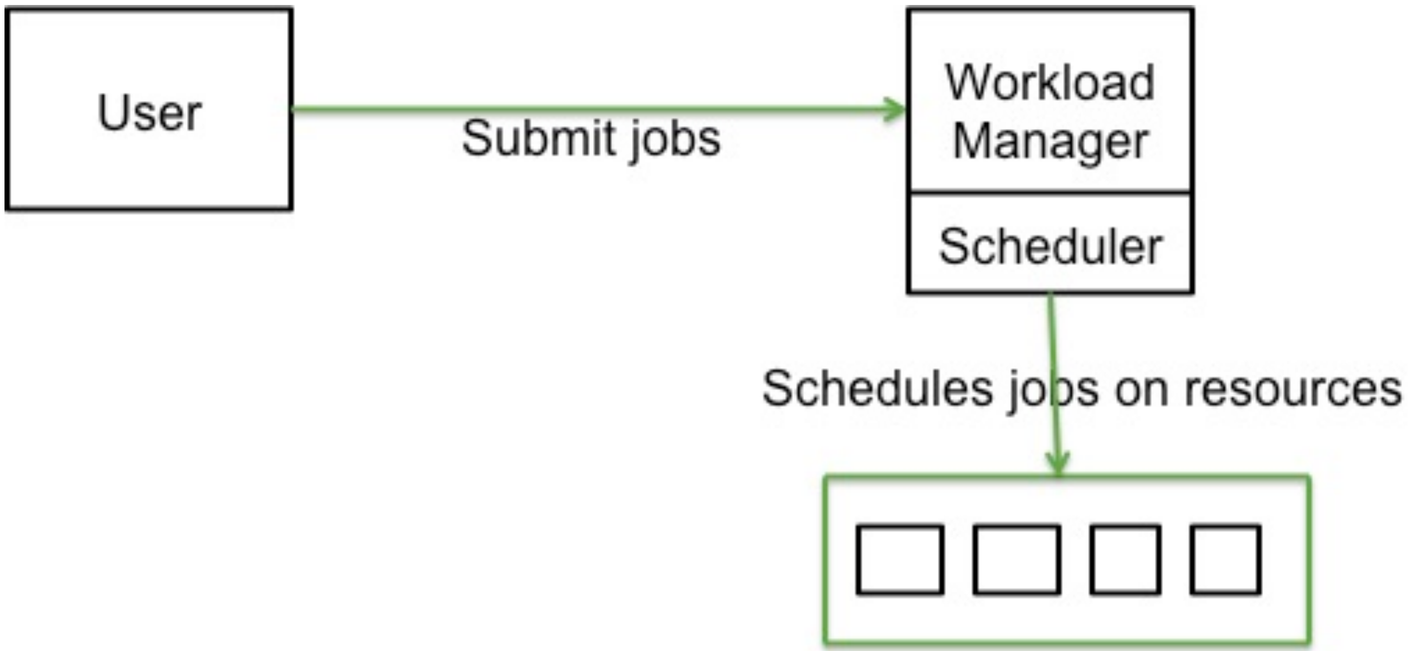
Urika-GX™

Main Resource Manager: Apache Mesos™

Workload Managers on Cray Platforms

Workload Managers: Slurm™, Moab™/Torque™

HPC workload managers



Slurm

- **Slurm knows how to configure communication among the sub-tasks**
  - Important for HPC applications
- **Anatomy:**
  - Daemons:
    - slurmctld: Central manager, monitors resources and execution of tasks
    - slurmd daemon: Runs on every compute node, responsible for running the tasks of a job on the corresponding node
  - Useful commands:
    - salloc command: Users grab an allocation of resources
    - srun command: Jobs are submitted
    - sinfo command: We can see the status of all the jobs

Moab/Torque

- **Moab**
  - Provides scheduling
  - Facilitates management tasks
  - Offers job orchestration
  - Facilitates enforcing site policies through its service level agreement (SLA) features
  - Supports batch and interactive workloads
- **Torque: open source resource manager which integrates with Moab.**

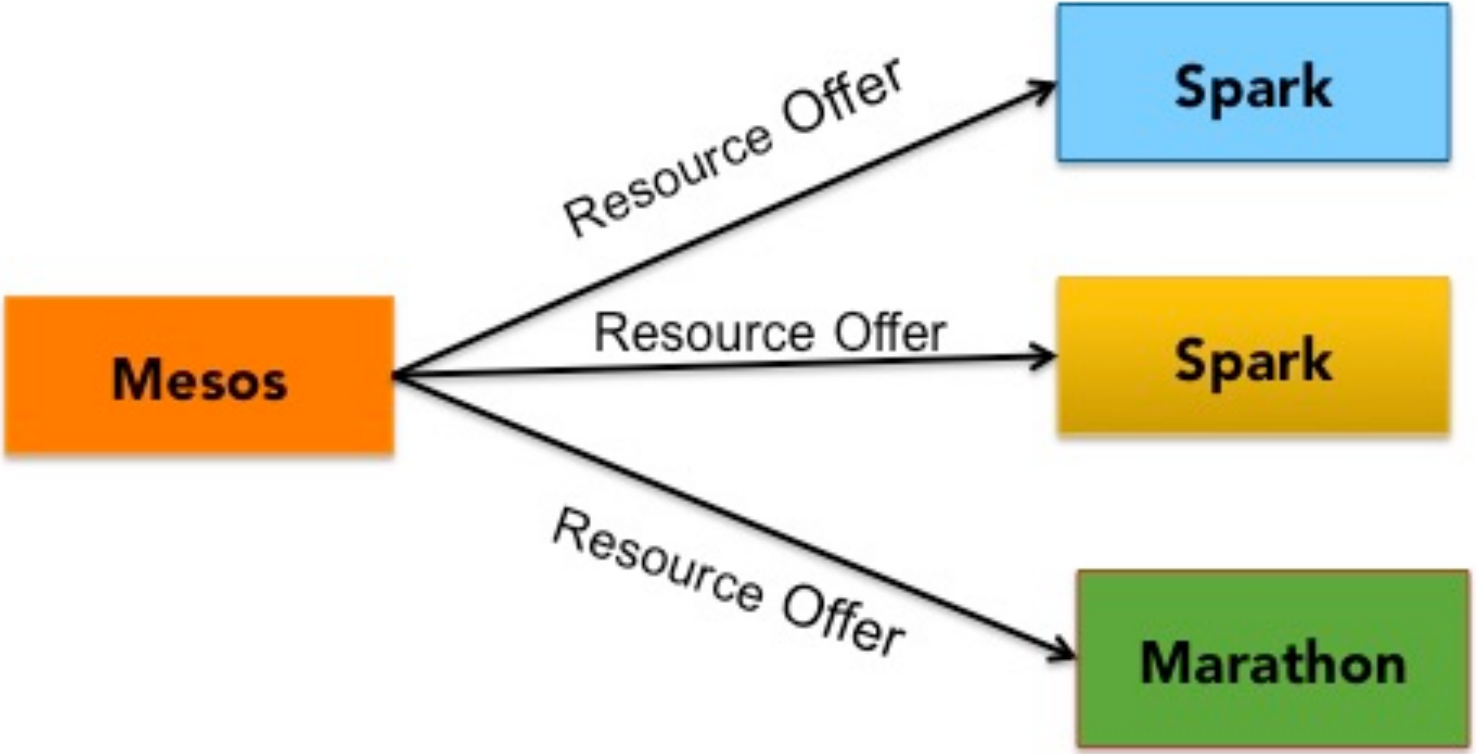
Apache Mesos

- **Main resource manager for Urika-GX**
- **Two level scheduling policy**
  - Handles resource allocation
- **Framework**
  - Programming paradigm and tools built around it
  - Register with Mesos
  - Responsible for scheduling, fault tolerance
  - Ex: Spark, Marathon
  - Each framework will address its own scheduling needs

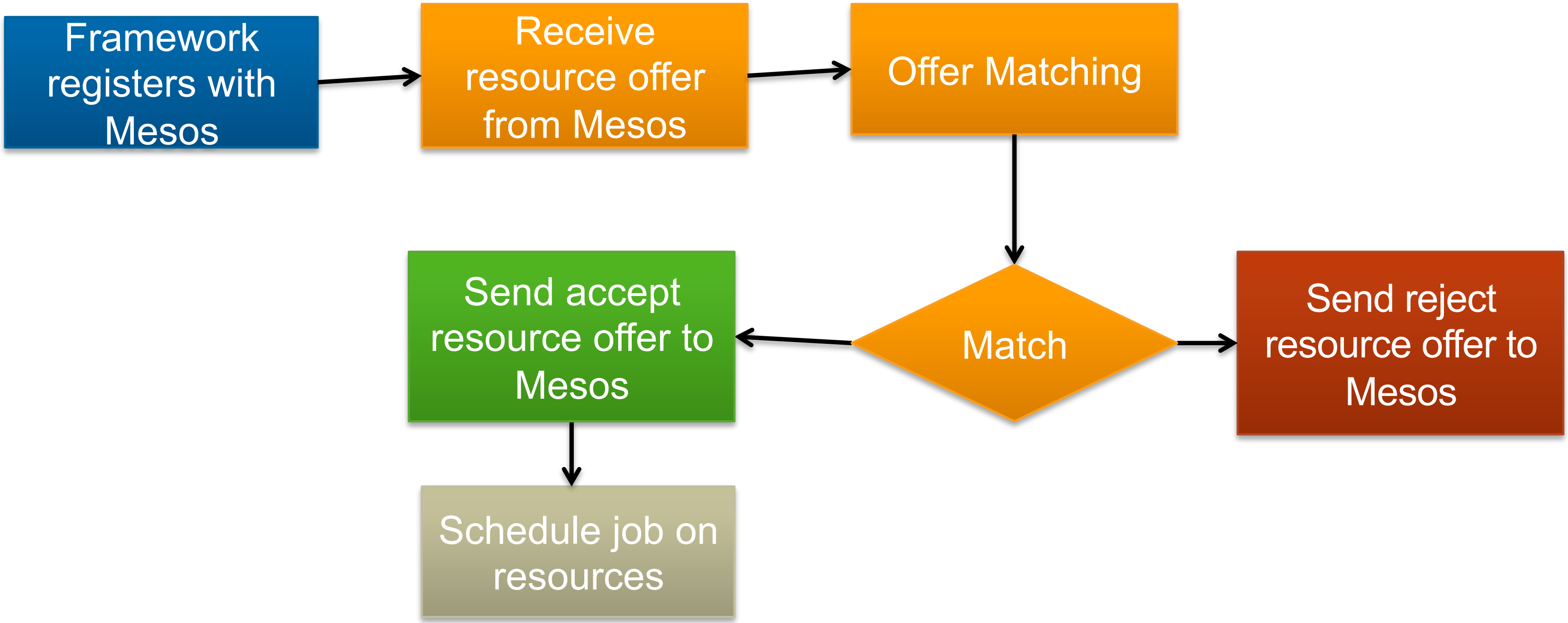
Mesos Resource Offers

- **Mesos gives resource offers to frameworks registered with it**
  - Marathon is a pre-registered framework on Mesos in Urika-GX
  - Every spark application registers as a new framework on Mesos
- **Every framework performs offer matching**
  - Each framework implements its own logic to perform offer matching
    - Available: default resources or resources specified by user
    - Compares this with resources in offer
    - If they match, accept resource offer
    - If not, reject the resource offer and wait
- **By default, spark looks for "some" match of resources**
  - Grabs the resource offer, even if it is less than the resources requested by user

Mesos giving resource offers to registered frameworks



Resource offer matching

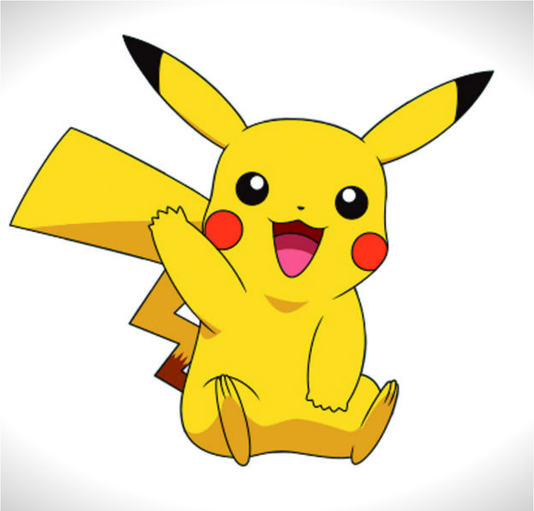


Description of Workflows
HPC Workflow

- **Dataset:**
  - Generated from multi-spectral images
  - The entries in the input file represent pixels of the spectral images
    - Each pixel is represented by the integer value of the class that it belongs to (a result of segmentation and clustering)
  - Two files of different sizes: one with 1024 rows, and other with 2048 rows

- **The application parallelizes the original algorithm using MPI using a 1-D block row-wise data distribution.**
  - Performs smoothing operation in an iterative fashion on the cells
  - The goal of smoothing is to change the class that a pixel has been assigned to, if a majority of neighboring elements have a different class
  - For each pixel, the two neighborhood pixels in each direction are analyzed and the algorithm runs ten iterations

Analytics workflow



- **Pokemon dataset:**
  - 721 Pokemon (csv format)
  - Each line consists of:
    - Id for each pokemon, Name of the pokemon, Primary type of the pokemon, Sum of the existing pokemon statistics, Hit points, base modifier for normal attacks, base damage resistance against normal attacks, special attack, base damage resistance against special attacks, and speed which determines which pokemon attacks first each round
- **Few simple spark applications**
  - List all the pokemons grouped by primary type
  - List all pokemons grouped by both primary and secondary type
  - List all pokemons grouped by generation

Running the workflows on different workload managers

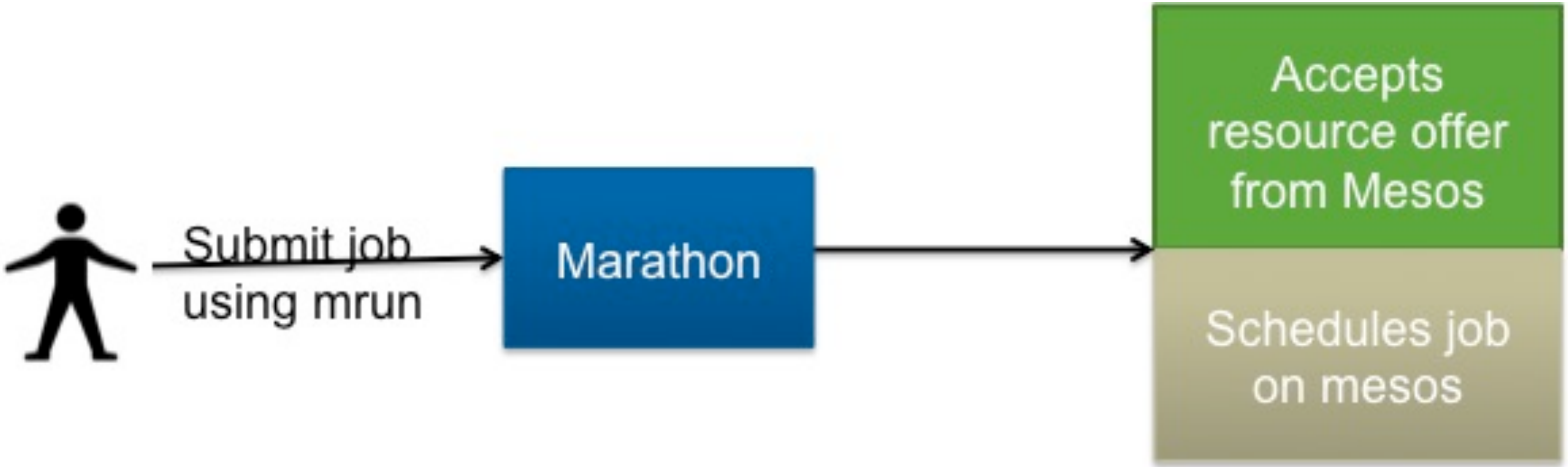
- **Running the workflow on Apache Mesos**
  - HPC
  - Analytics

HPC workflow on Mesos

- **Marathon**
  - Framework in the Mesos ecosystem
  - Supports long running web services
- **Cray developed a Marathon Framework Application Launcher called** “mrun”
  - Configures the Aries setup required for PGAS/DMAPP
- “mrun” **allows**
  - More precise control of system resources
  - Better ability to clean up error cases that may arise when running HPC tasks

- **Marathon receives resource offers from Mesos**
- **When a user submits an** mrun **job, Marathon**
  - Accepts the resources from Mesos
  - Gives them to the mrun which runs as a marathon application
- **From there, the HPC job is scheduled as a regular marathon application utilizing the resources it receives from Mesos**

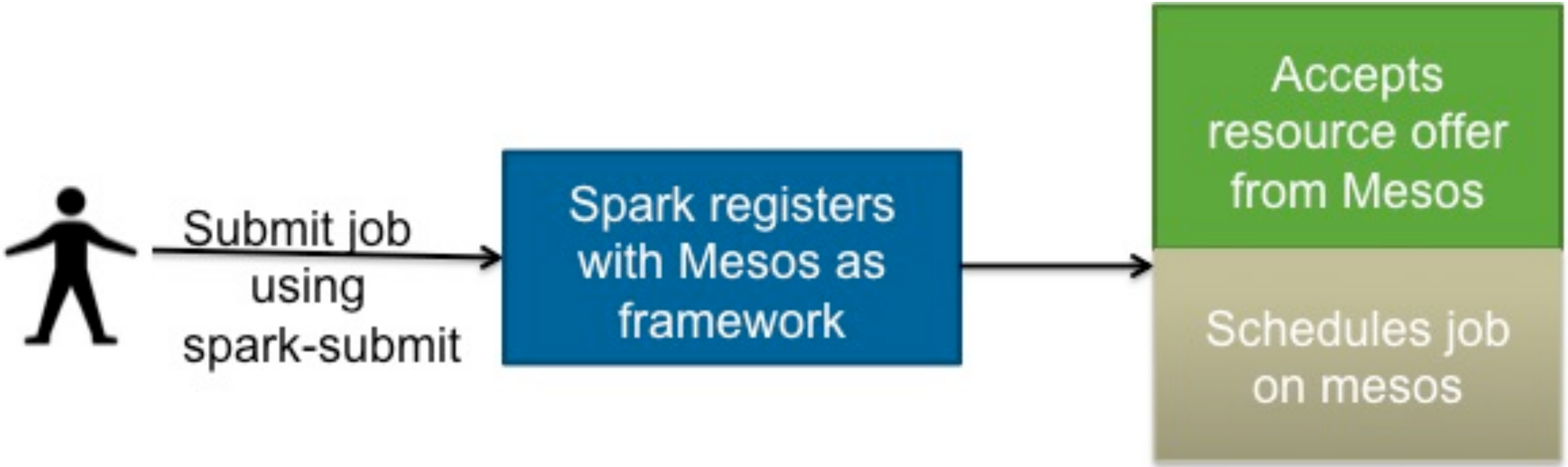
HPC workflow on Mesos



Analytics workflow on Mesos

- **Spark applications are launched using the spark-submit command**
- **Mesos gives resource offers to the current Spark application**
  - Spark choses either to accept or reject those resources
  - The resources are granted to spark
  - Spark then schedules the spark job on the offered resources
  - Once the job finishes running, the resources are released by Spark back to Mesos
- **Users can configure parameters to say what and how spark can accept resources**

Analytics workflow on Mesos



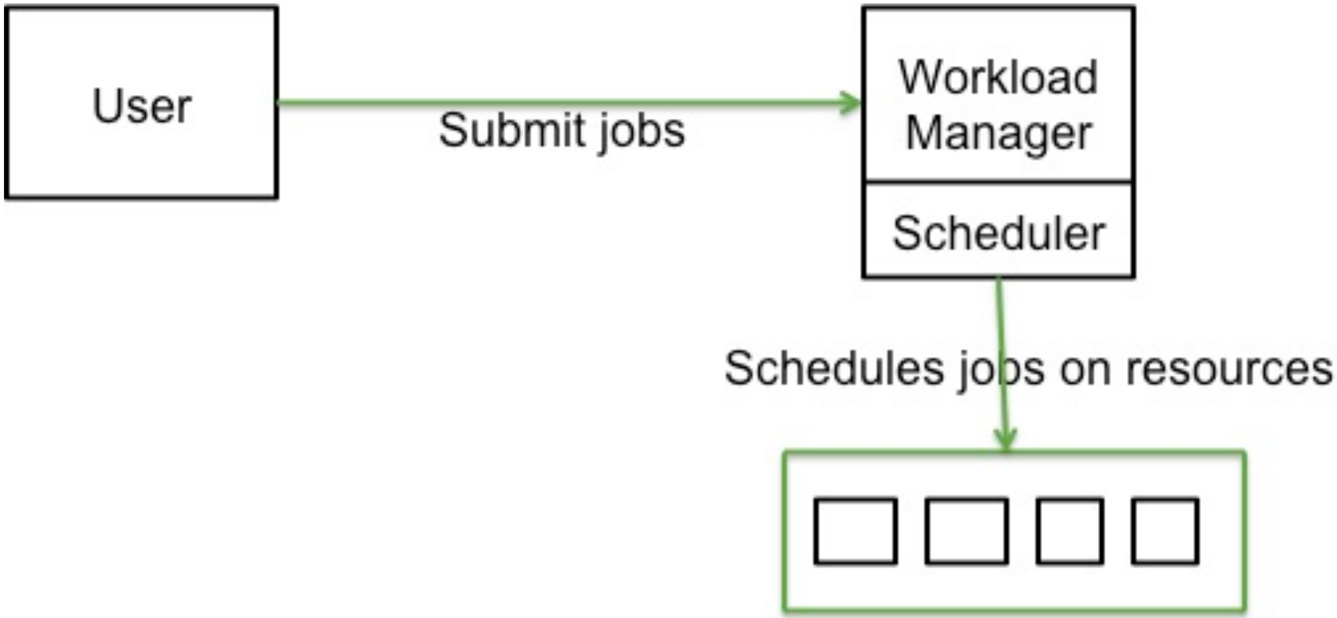
Running the workflow on different workload managers

- **Running the workflow on Slurm**
  - HPC
  - Analytics

HPC workflow on Slurm

- **Step 1: Grab an allocation using salloc**
  - The resource request is enqueued to slurmctld (the central manager)
  - User waits until the resource request can be satisfied (or is timed out) at which time the user's prompt is returned
- **Step 2: Launch job using srun**
  - srun communicates to slurmctld
  - Requests are sent to slurmd daemons running on the compute nodes to configure the Aries network across the nodes reserved
  - Fork/exec the correct number of instances of the application on each node requested
  - When the application finishes, slurm will clean-up its internal Aries network maps and return the user to their prompt, still within the original salloc environment
  - Only when the user exits from the salloc shell are the resources released back to slurm

HPC workflow on Slurm



Analytics workflow on Slurm

- **Shifter images are used to run Spark on slurm**
- **Shifter images are spun up using** salloc
  - We spin up a virtual standalone spark cluster in the allocation received
  - We run the spark stand alone cluster manager
  - This allows us to run spark jobs
  - Support for submitting jobs in batch style, interactive shell
  - Spark over shifter uses TCP/IP for communication over Aries interconnect

Comparison of workload managers

- **Platform with “n” nodes is used**
- **Multiple jobs are submitted such that the system is fully busy**
- **Record that no resources are available**
- **Submit a new job to the queue by explicitly requesting x nodes.**
- **Observe the behavior**
  - Since there are no nodes available currently, note that the job is waiting there for resources to be available
- **Free up y nodes (y <x)**
  - The number of nodes available now are less that the number of nodes requested by the job
- **Observe the behavior**

What happened on Mesos?

- Due to the resource offer model, frameworks have a flexibility to accept resources though it does not satisfy its exact requirement
- Though (number of resources available) < (number of resources requested), the applications accepted the available resources anyway
- Efficient resource utilization!

Urika-GX system with all resources available

Resources

	CPUs	GPUs	Mem	Disk
Total	1476	0	20.2 TB	7.7 TB
Used	0	0	0 B	0 B
Offered	0	0	0 B	0 B
Idle	1476	0	20.2 TB	7.7 TB

Urika-GX system with no resources available

Resources

	CPUs	GPUs	Mem	Disk
Total	1476	0	20.2 TB	7.7 TB
Used	1476	0	40.0 GB	0 B
Offered	0	0	0 B	0 B
Idle	0	0	20.1 TB	7.7 TB

Urika-GX system with only one node available

Resources

	CPUs	GPUs	Mem	Disk
Total	1476	0	20.2 TB	7.7 TB
Used	1440	0	39.1 GB	0 B
Offered	0	0	0 B	0 B
Idle	36	0	20.1 TB	7.7 TB

Active Frameworks

Find...

ID ▼	Host	User	Name	Role	Principal	Active Tasks	CPUs	GPUs	Mem	Disk	Max Share	Regi
…9396-2803dc500aa8-0045	zeno-nid00030	hayyalasom	GetPokemonInfo	*	spark	1	36	0	105.6 GB	0 B	2.439%	just r
…9396-2803dc500aa8-0043	zeno-nid00030	hayyalasom	GetPokemonInfo	*	spark	0	0	0	0 B	0 B	0%	just r
…9396-2803dc500aa8-0042	zeno-nid00030	hayyalasom	GetPokemonInfo	*	spark	0	0	0	0 B	0 B	0%	just r
…9396-2803dc500aa8-0041	zeno-nid00030	hayyalasom	GetPokemonInfo	*	spark	0	0	0	0 B	0 B	0%	just r
…9396-2803dc500aa8-0040	zeno-nid00030	hayyalasom	GetPokemonInfo	*	spark	0	0	0	0 B	0 B	0%	just r
…9396-2803dc500aa8-0037	zeno-nid00030	hayyalasom	GetPokemonInfo	*	spark	0	0	0	0 B	0 B	0%	just r
…9396-2803dc500aa8-0036	zeno-nid00030	hayyalasom	GetPokemonInfo	*	spark	0	0	0	0 B	0 B	0%	just r
…9396-2803dc500aa8-0035	zeno-nid00030	hayyalasom	GetPokemonInfo	*	spark	0	0	0	0 B	0 B	0%	just r
…b3de-144d4497db4d-0000	nid00032	marathon	marathon	*	marathon	40	1,440	0	39.1 GB	0 B	97.561%	5 hou

Active Frameworks

What happened on Slurm?

- Strict resource requirements specified when job is submitted
- Jobs sit there waiting in the queue for the exact resources to be available

Compared to Mesos, lower resource utilization is seen

XC system with all resources available

```
PARTITION   AVAIL  TIMELIMIT  NODES  STATE
workq*         up 1-00:00:00    171   idle
ccm_queue      up 1-00:00:00    171   idle
```

XC system with no resources available

```
PARTITION   AVAIL  TIMELIMIT  NODES  STATE
workq*         up 1-00:00:00    171  alloc
ccm_queue      up 1-00:00:00    171  alloc
```

XC system with only seven nodes available

```
PARTITION  AVAIL  TIMELIMIT  NODES  STATE
workq*        up 1-00:00:00    164  alloc
workq*        up 1-00:00:00      7   idle
ccm_queue     up 1-00:00:00    164  alloc
ccm_queue     up 1-00:00:00      7   idle
```

Jobs waiting in the queue though there are seven nodes available

```
JOBID      NAME      USER ST         TIME   NODES NODELIST(REASON)
15452 start_an hayyalas  R        19:20      50 nid000[04-15,20-57]
15453 start_an hayyalas  R        18:28     100 nid00[058-075,080-139,148-169]
15454 start_an hayyalas  R        17:18      14 nid00[170-183]
15457 parallel hayyalas PD         0:00      25 (Resources)
15458 parallel hayyalas PD         0:00       8 (Priority)
15459 start_an hayyalas PD         0:00       8 (Priority)
```

Pros and Cons of the workload managers

	Apache Mesos	Slurm
Resource Utilization	With its resource offer model, achieves efficient resource utilization for flexible/elastic workloads	Originally designed for HPC workloads which have strict resource requirements. Not the best option for elastic workloads like analytics style applications.
Queue support	Missing in current open source Mesos. Enterprise DCOS offers these (not open source).	Fair share scheduler, global queue support available
Flexibility	Use marathon to develop marathon launcher for HPC jobs	Use shifter containers to launch analytics workloads

Summary

- **Discussed our experiences**
  - How we launch both analytics and HPC workloads on Slurm and Mesos
  - Highlighted main differences
- **Resource offers model of Mesos helps us achieve better resource utilization as compared to Slurm**
- **Actively exploring alternatives to Mesos to overcome its limitations**
- **Desire:**
  - Efficient resource utilization
  - Fair share scheduling

More details in the original paper

- Experiences running different work load managers across Cray Platforms

Legal Disclaimer

Q&A

Haripriya Ayyalasomayajula
hayyalasom@cray.com

Karlon West
karlon@cray.com