Serverless Machine Learning on Modern Hardware

Patrick Stuedi
IBM Research

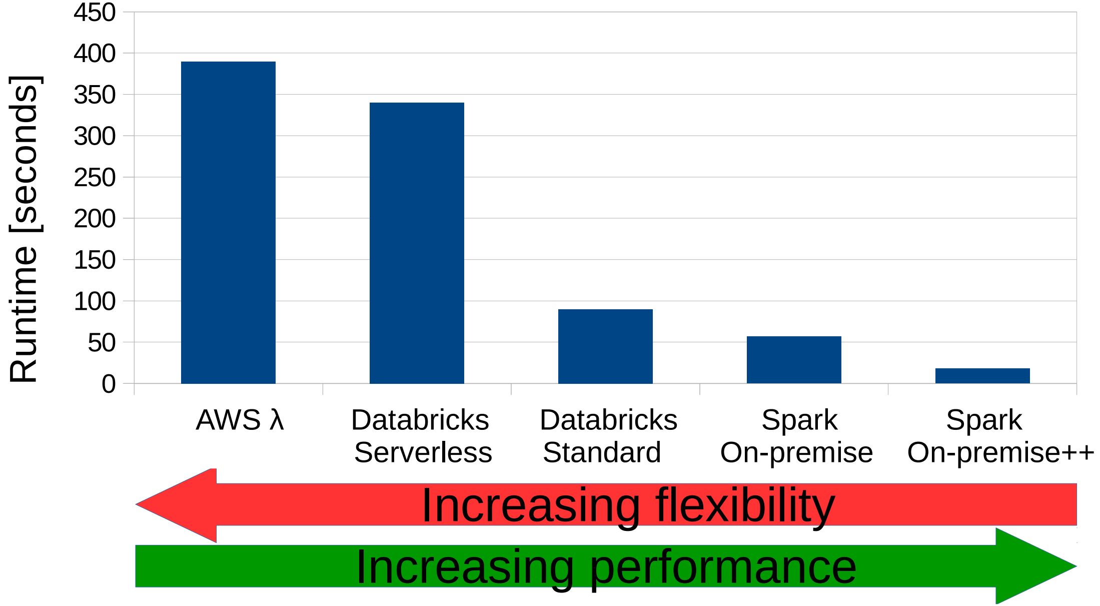
#Res6SAIS

Serverless Computing

- No need to setup/manage a cluster
- Automatic, dynamic and fine-grained scaling
- Sub-second billing
- Many frameworks: AWS Lambda, Google Cloud Functions, Azure Functions, Databricks Serverless, etc.

Challenge: Performance

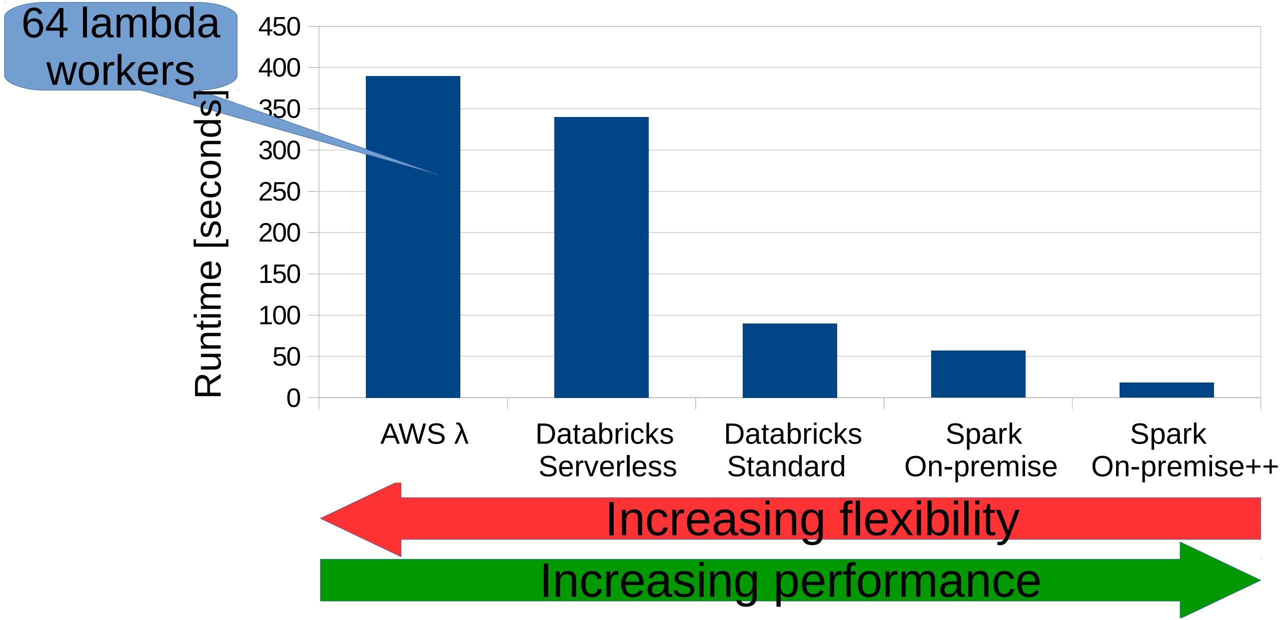
Example:
Sorting 100GB



Spark/On-Premise++: Running Apache Spark on a High-Performance Cluster using RDMA and NVMe Flash, Spark Summit'17

Challenge: Performance

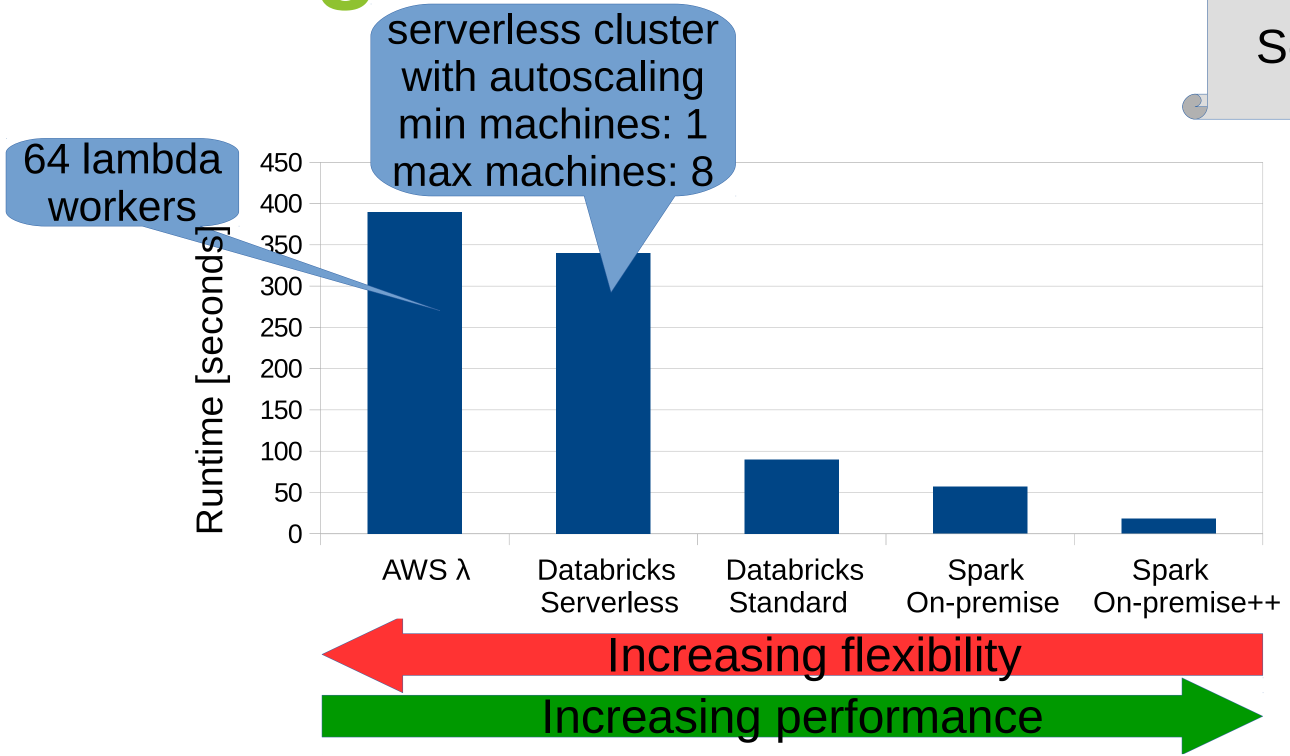
Example:
Sorting 100GB



Spark/On-Premise++: Running Apache Spark on a High-Performance Cluster using RDMA and NVMe Flash, Spark Summit'17

Challenge: Performance

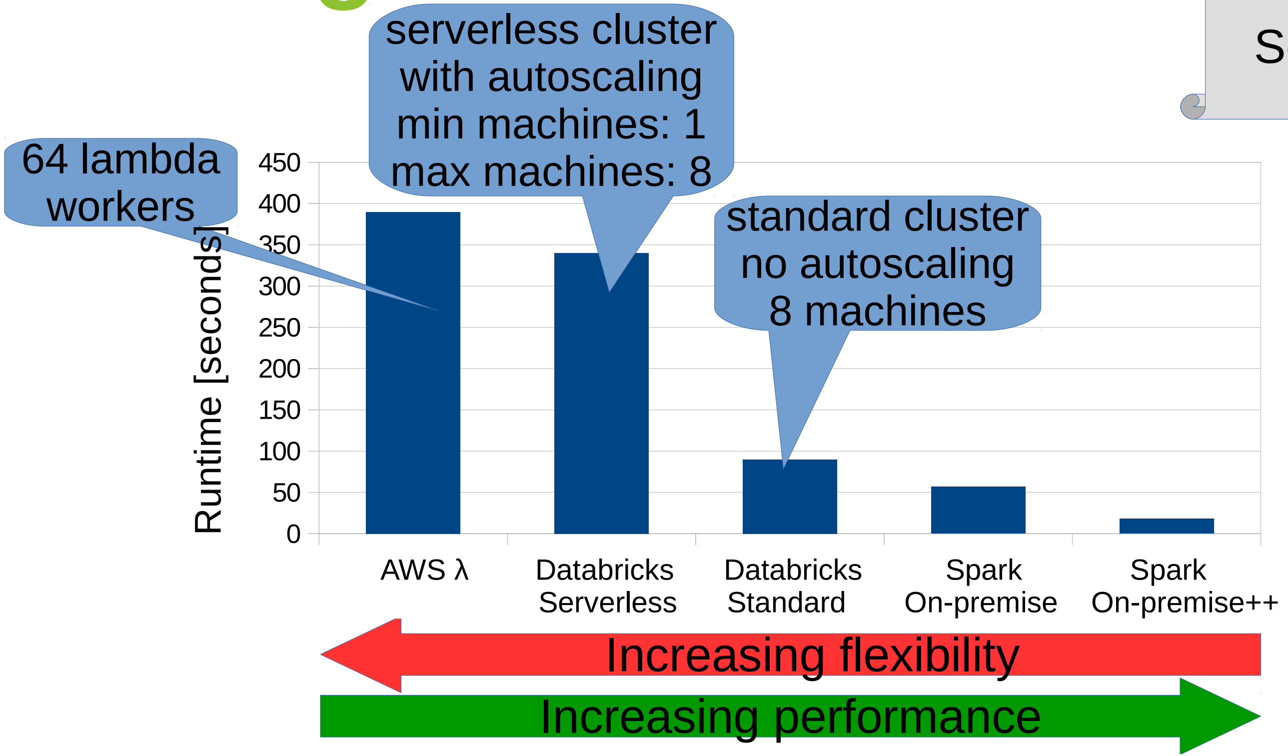
Example:
Sorting 100GB



Spark/On-Premise++: Running Apache Spark on a High-Performance Cluster using RDMA and NVMe Flash, Spark Summit'17

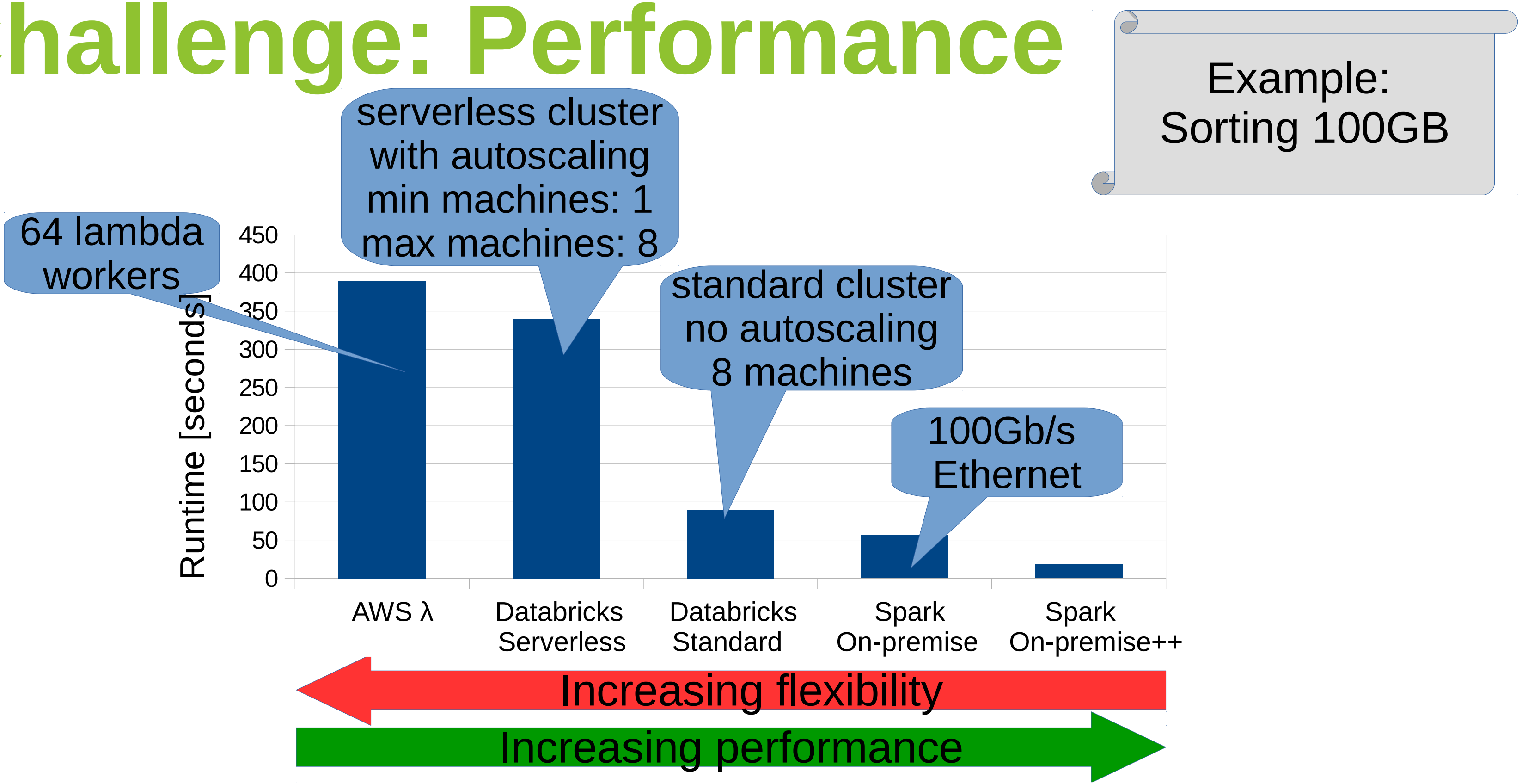
Challenge: Performance

Example:
Sorting 100GB



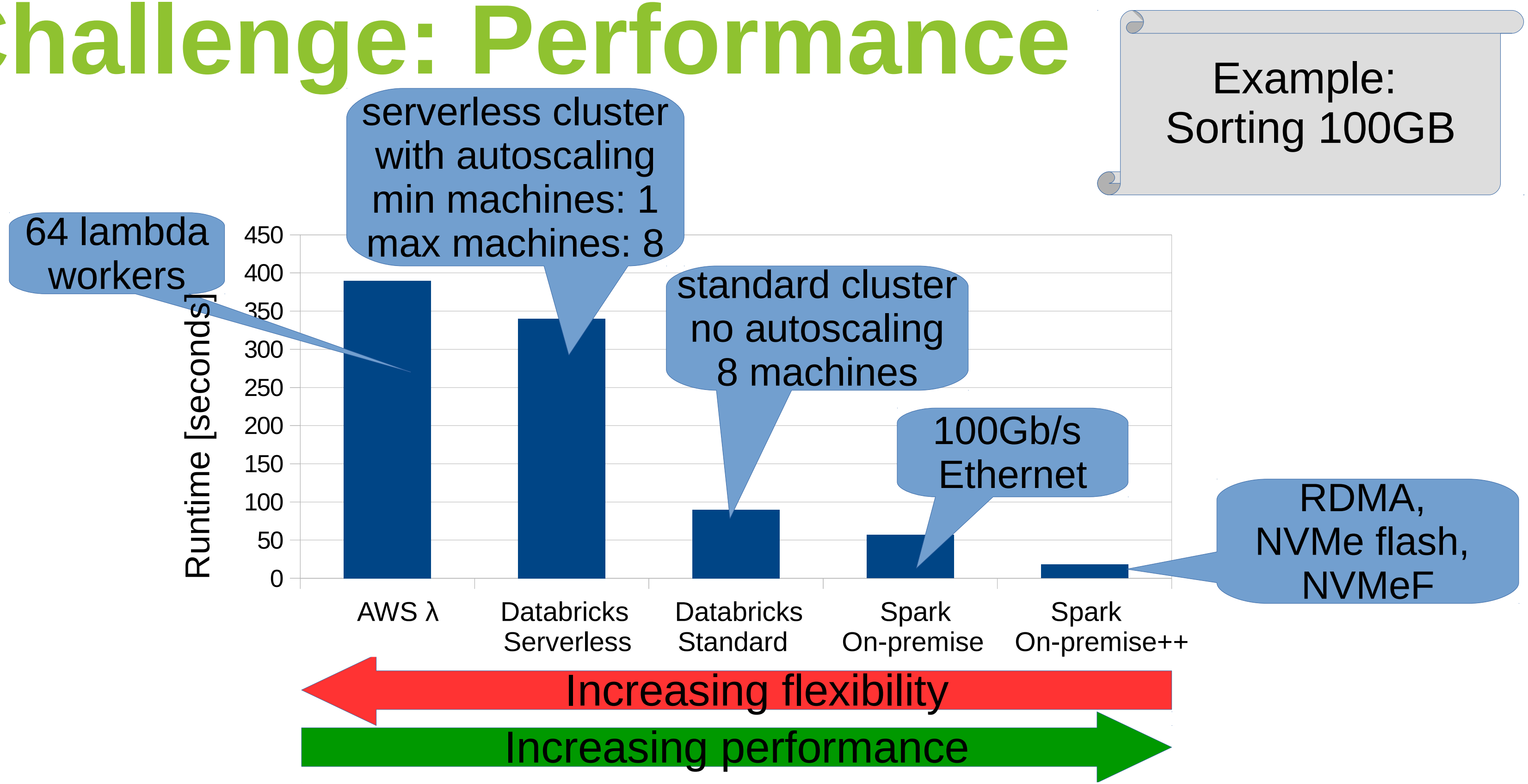
Spark/On-Premise++: Running Apache Spark on a High-Performance Cluster using RDMA and NVMe Flash, Spark Summit'17

Challenge: Performance



Spark/On-Premise++: Running Apache Spark on a High-Performance Cluster using RDMA and NVMe Flash, Spark Summit'17

Challenge: Performance



Spark/On-Premise++: Running Apache Spark on a High-Performance Cluster using RDMA and NVMe Flash, Spark Summit'17

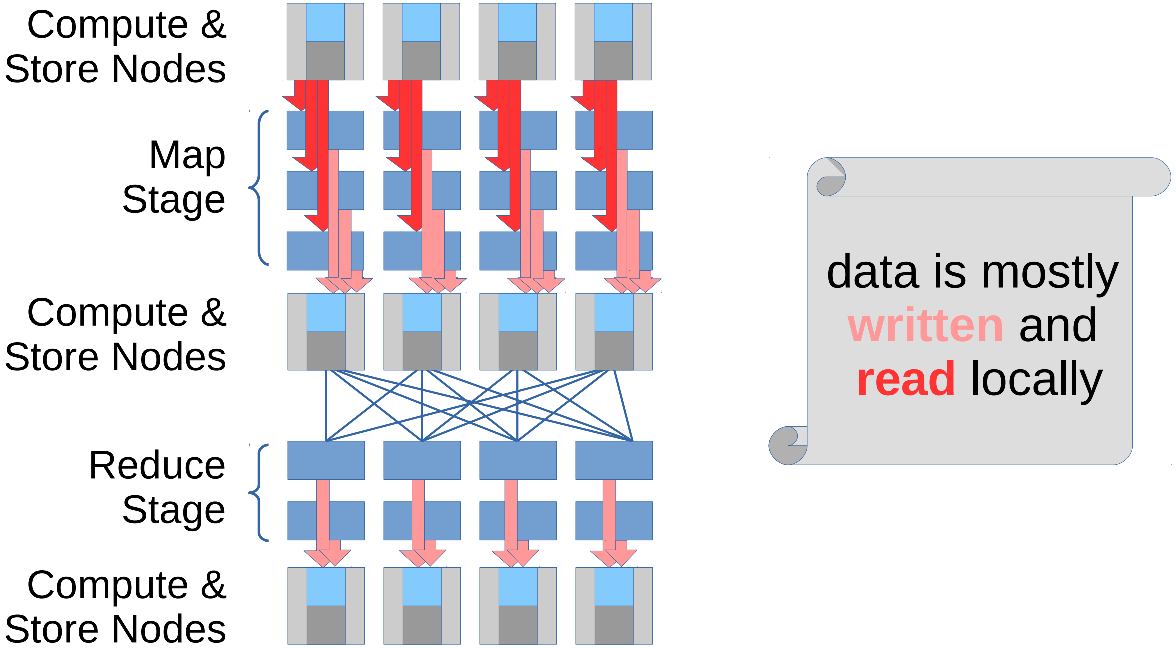
Why is it so hard?

- **Scheduler:** when to best add/remove resources?
- **Container startup:** may have to dynamically spin up containers
- **Storage:** input data needs to be fetched from remote storage (e.g., S3)
  - As opposed to compute-local storage such as HDFS
- **Data sharing:** intermediate needs to be temporarily stored on remote storage (S3, Redis)
  - Affects operations like shuffle, broadcast, etc.,

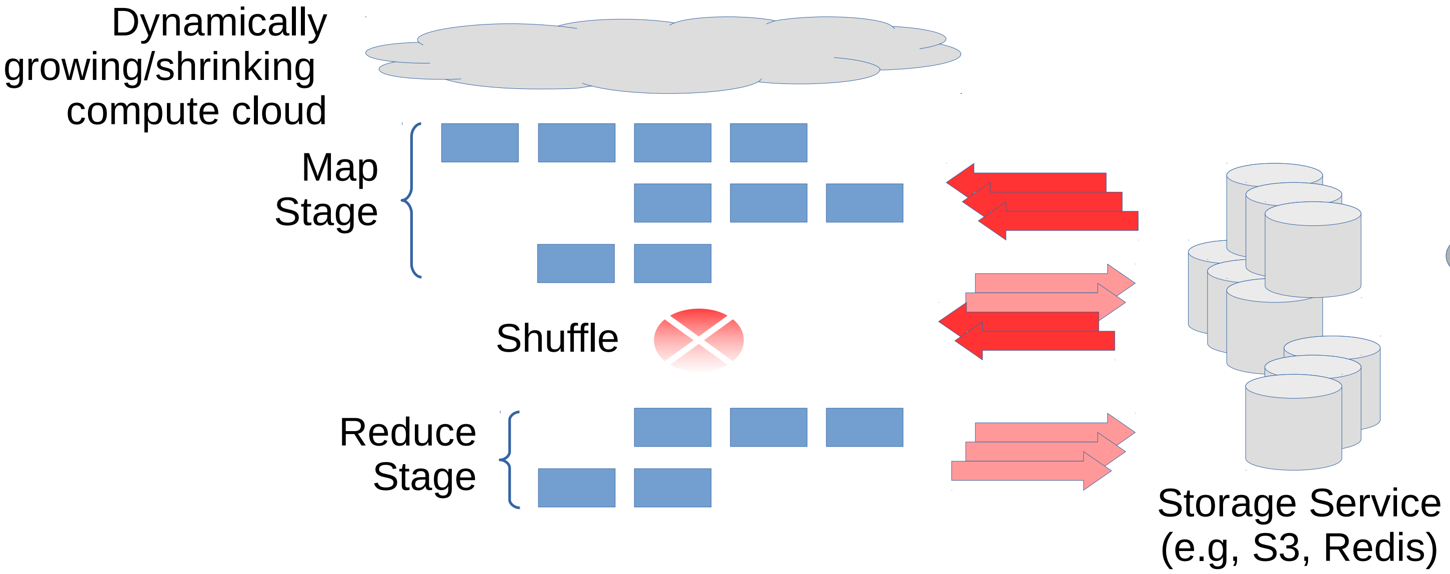
Why is it so hard?

- **Scheduler:** when to best add/remove resources?
- **Container startup:** may have to dynamically spin up containers
- **Storage**: input data needs to be fetched from remote storage (e.g., S3)
  - As opposed to compute-local storage such as HDFS
- **Data sharing**: intermediate needs to be temporarily stored on remote storage (S3, Redis)
  - Affects operations like shuffle, broadcast, etc.,

Example: MapReduce (Cluster)

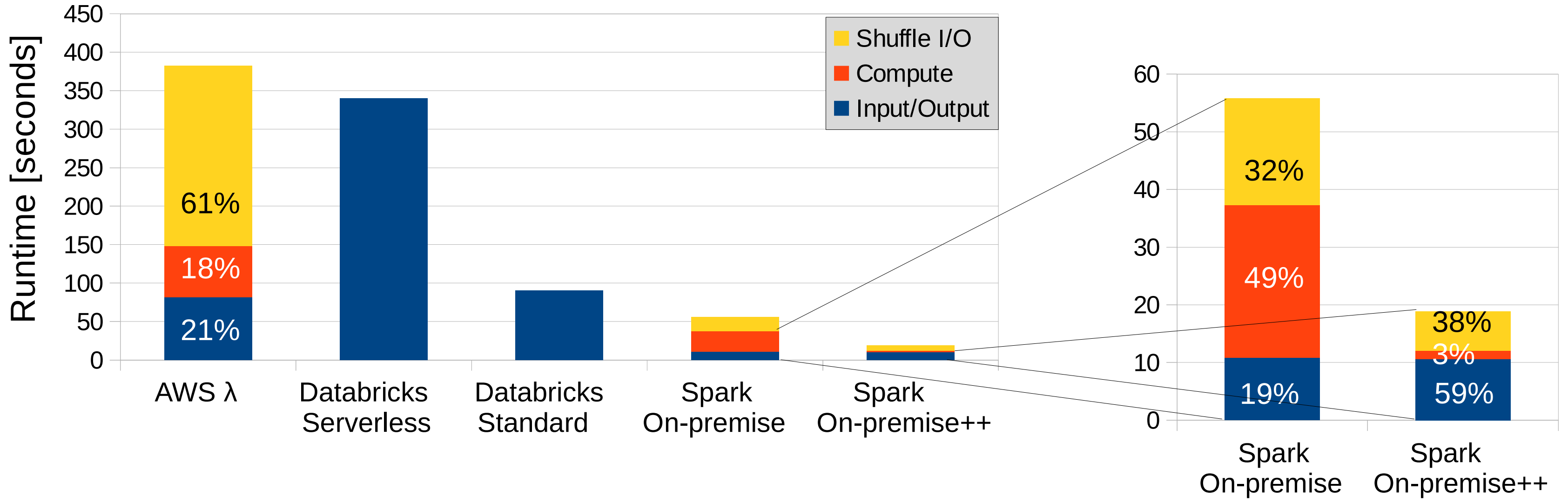


Example: MapReduce (Serverless)



data is exclusively written and read remotely

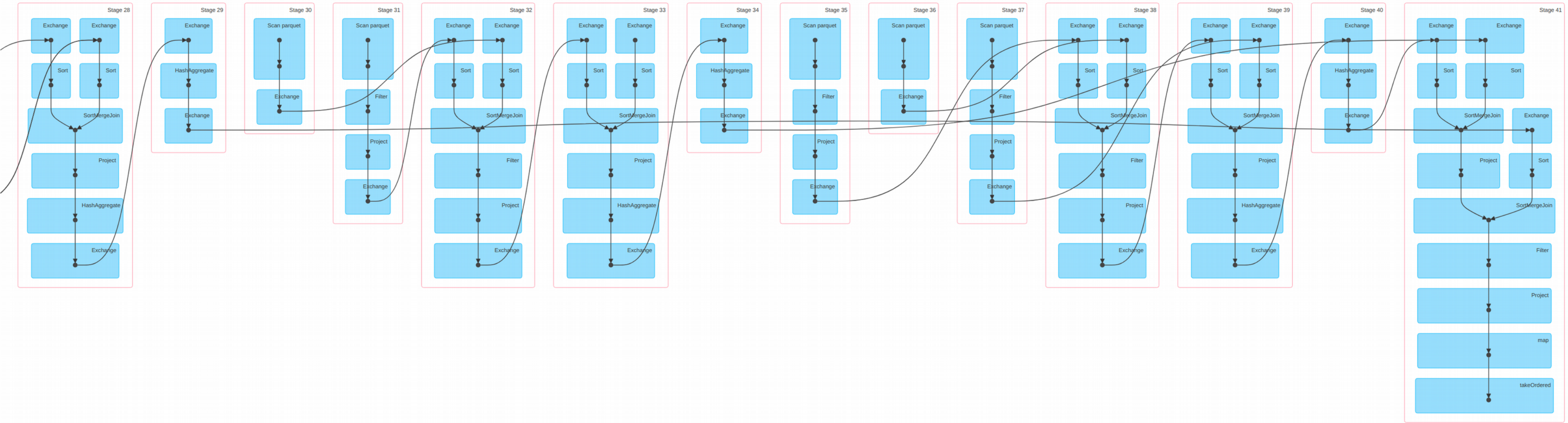
I/O Overhead: Sorting 100GB



Shuffle overheads are significantly higher when intermediate data is stored remotely

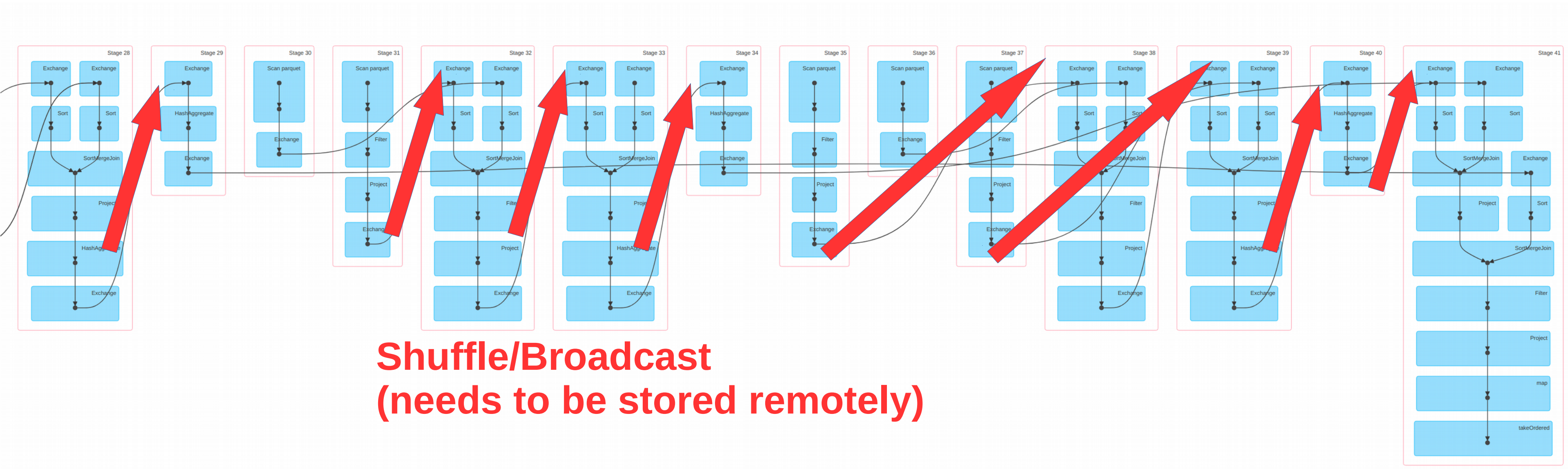
What about other workloads?

Example: SQL, Query 77 / TPC-DS benchmark



What about other workloads?

Example: SQL, Query 77 / TPC-DS benchmark

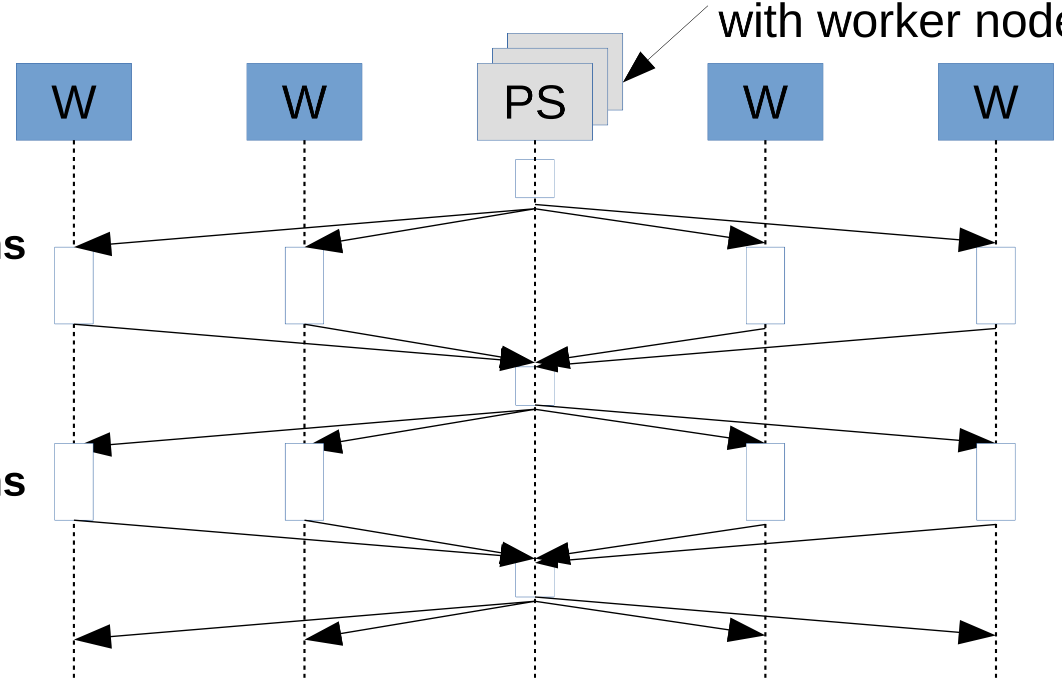


Shuffle/Broadcast
(needs to be stored remotely)

What about other workloads?

Example: Iterative ML (e.g., linear regression)

could be co-located
with worker nodes



*) fetch model params
*) compute
*) update model

:

*) fetch model params
*) compute
*) update model

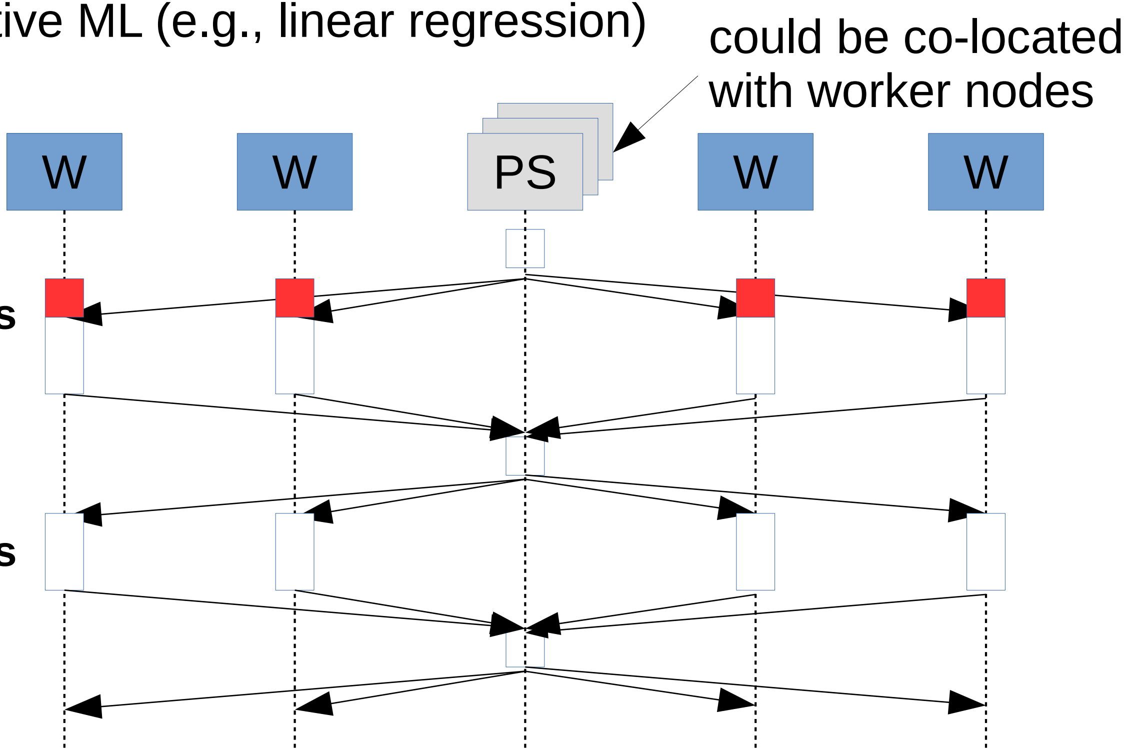
:

What about other workloads?

Example: Iterative ML (e.g., linear regression)

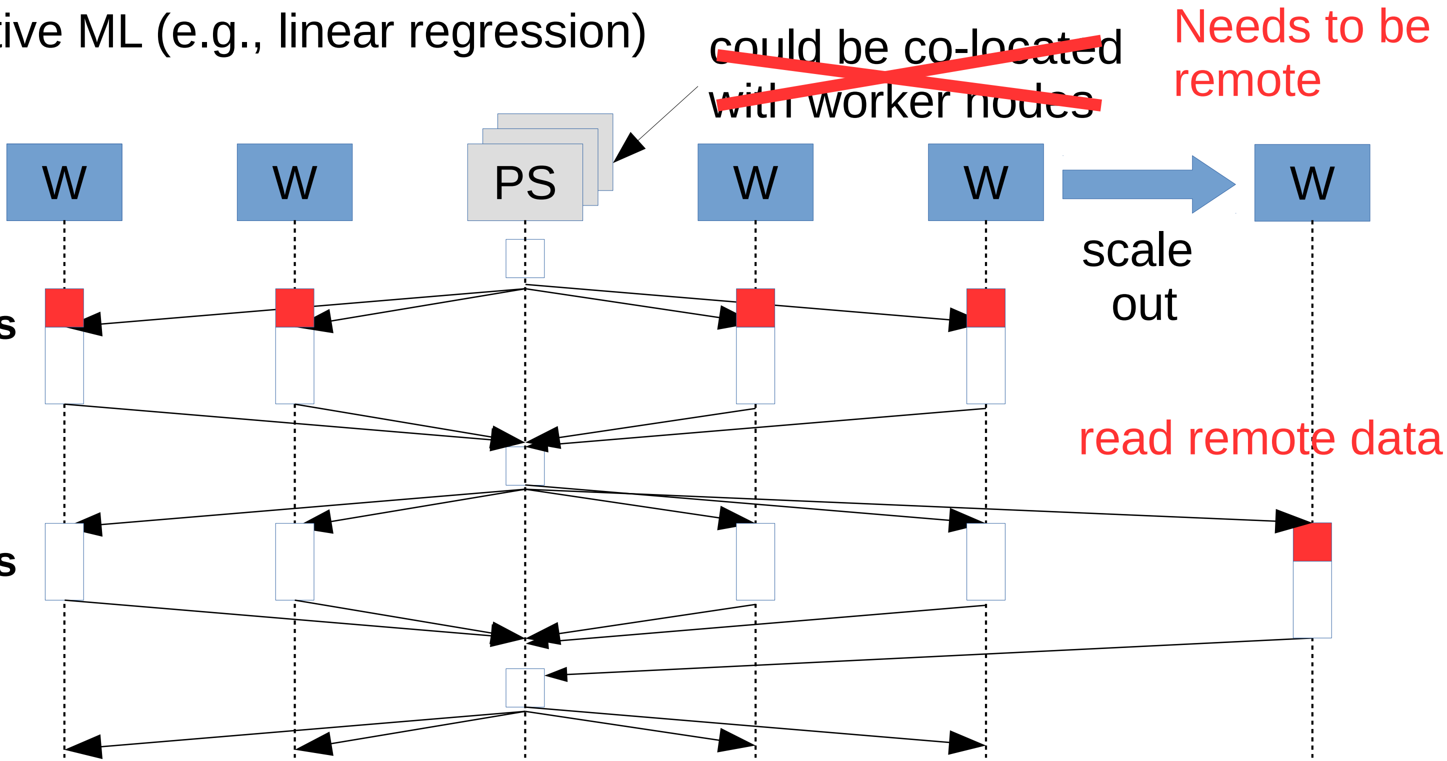
*) read training data
*) fetch model params
*) compute
*) update model

*) use cached data
*) fetch model params
*) compute
*) update model



What about other workloads?

Example: Iterative ML (e.g., linear regression)

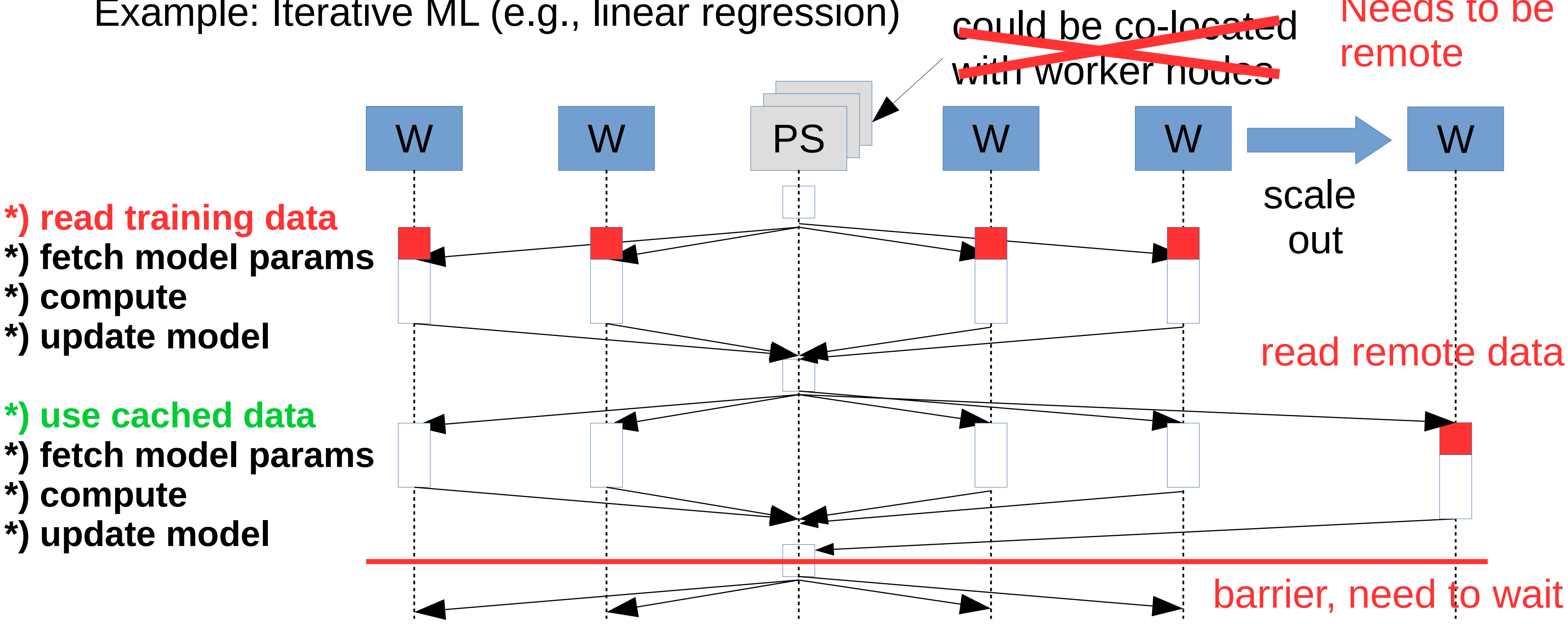


*) read training data
*) fetch model params
*) compute
*) update model

*) use cached data
*) fetch model params
*) compute
*) update model

What about other workloads?

Example: Iterative ML (e.g., linear regression)



Can we..

- ..use Spark to run such workloads in a serverless fashion?
  - Dynamic scaling of compute nodes while jobs are running
  - No cluster configuration
  - No startup time overhead
- ..eliminate the performance overheads?
  - Workloads should run as fast as on a dedicated cluster

Design Options

- **Scheduling:**
  1. Use serverless framework to schedule executors
  2. Use serverless framework to schedule tasks
  3. Enable sharing of executors among different applications
- **Intermediate data:**
  1. Executors cooperate with scheduler to flush data remotely
  2. Consequently store all intermediate state remotely

Design Options

- **Scheduling:**

High startup Latency!

1. Use serverless framework to schedule executors
2. Use serverless framework to schedule tasks
3. Enable sharing of executors among different applications

- **Intermediate data:**

1. Executors cooperate with scheduler to flush data remotely
2. Consequently store all intermediate state remotely

Design Options

- **Scheduling:**
  1. Use serverless framework to schedule executors
  2. Use serverless framework to schedule tasks
  3. Enable sharing of executors among different applications

High startup Latency!

Slow!

- **Intermediate data:**
  1. Executors cooperate with scheduler to flush data remotely
  2. Consequently store all intermediate state remotely

Design Options

- **Scheduling:**
  1. Use serverless framework to schedule executors
  2. Use serverless framework to schedule tasks
  3. Enable sharing of executors among different applications

High startup Latency!

Slow!

- **Intermediate data:**
  1. Executors cooperate with scheduler to flush data remotely
  2. Consequently store all intermediate state remotely

Design Options

- **Scheduling:**

  High startup Latency!

  1. Use serverless framework to schedule executors

  Slow!

  2. Use serverless framework to schedule tasks
  3. Enable sharing of executors among different applications

- **Intermediate data:**

  Complex!

  1. Executors cooperate with scheduler to flush data remotely
  2. Consequently store all intermediate state remotely

Design Options

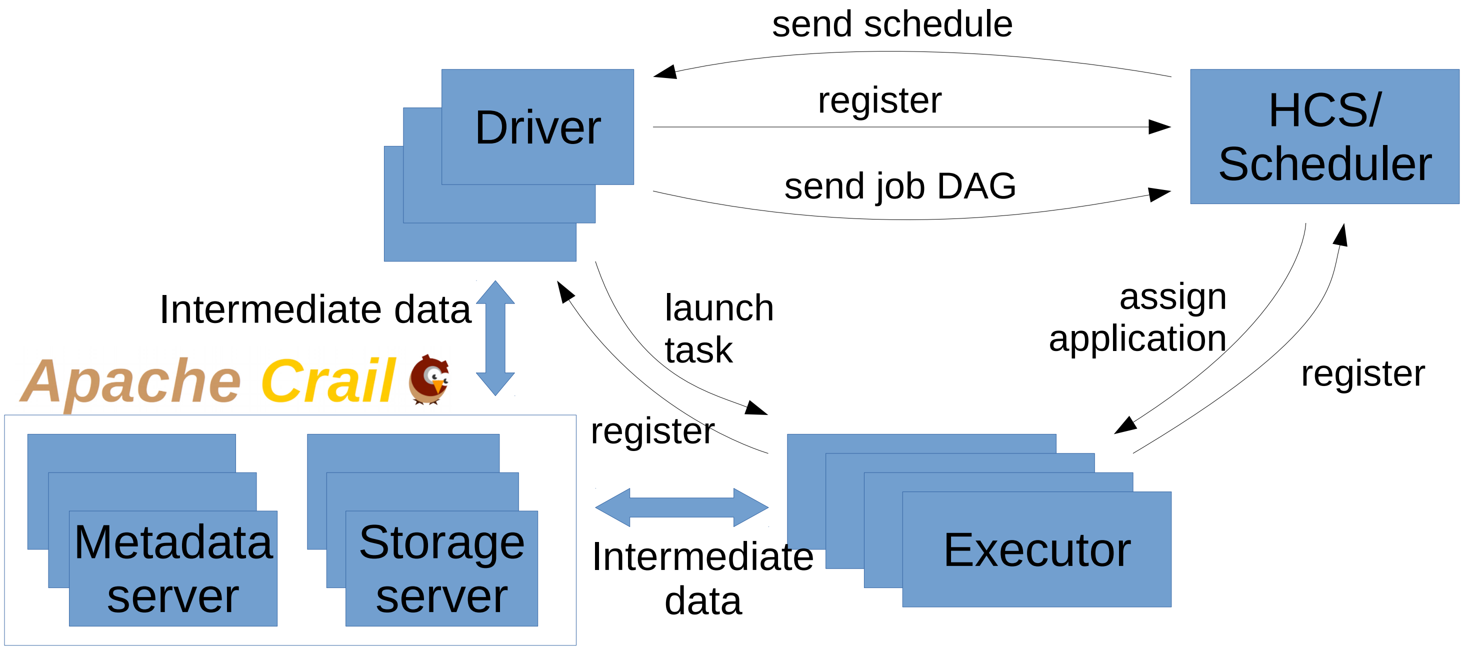
- **Scheduling:**
  1. Use serverless framework to schedule executors
  2. Use serverless framework to schedule tasks
  3. Enable sharing of executors among different applications
- **Intermediate data:**
  1. Executors cooperate with scheduler to flush data remotely
  2. Consequently store all intermediate state remotely

High startup Latency!

Slow!

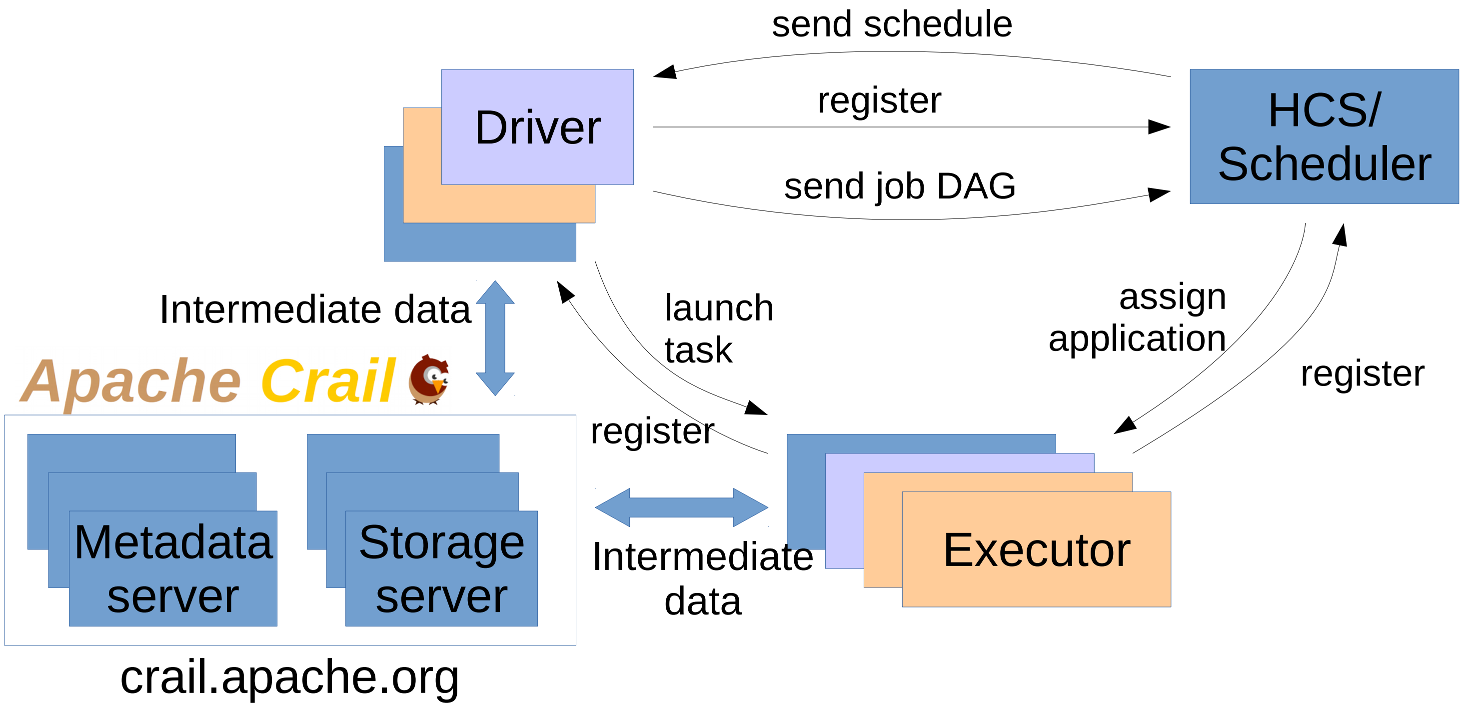
Complex!

Architecture Overview

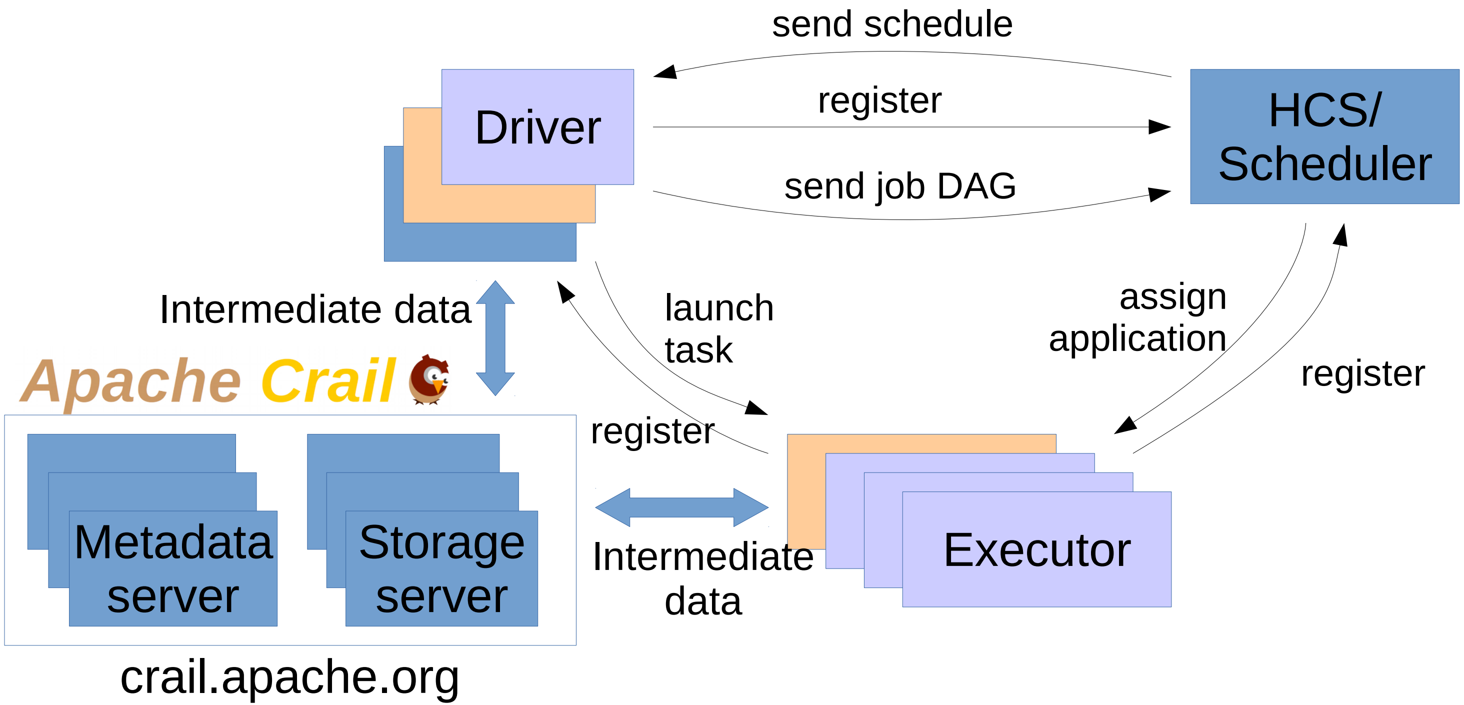


crail.apache.org

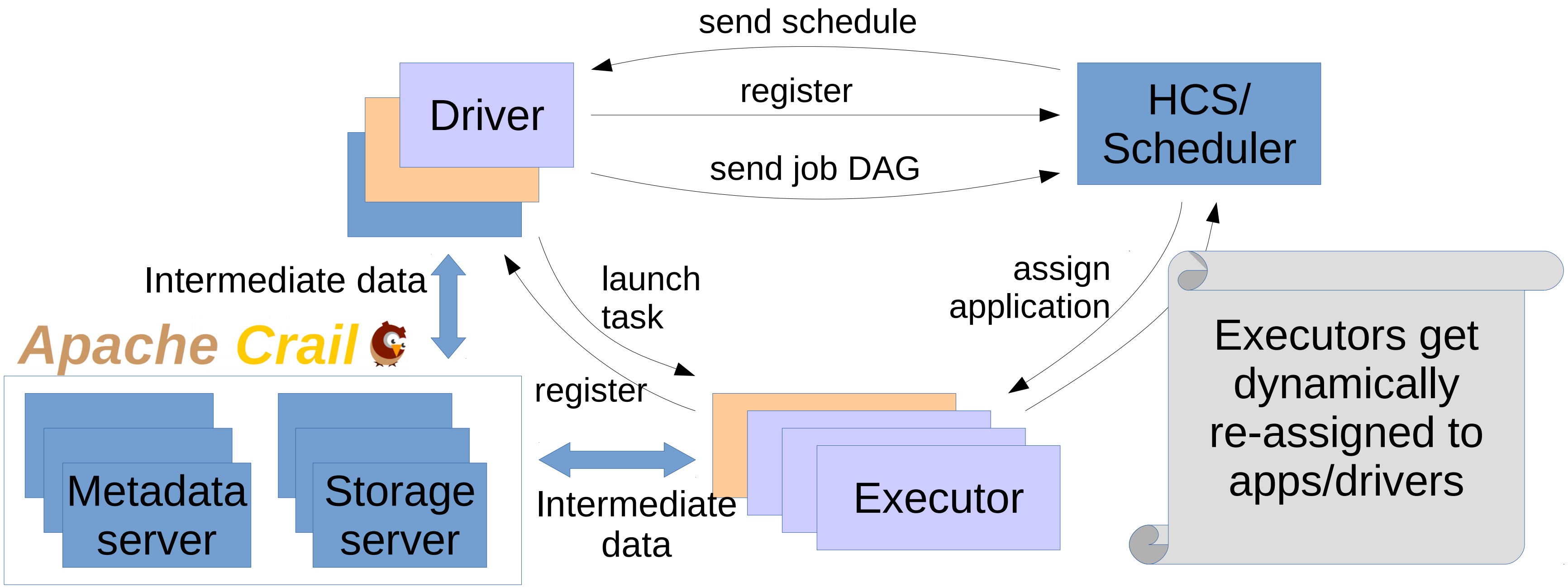
Architecture Overview



Architecture Overview

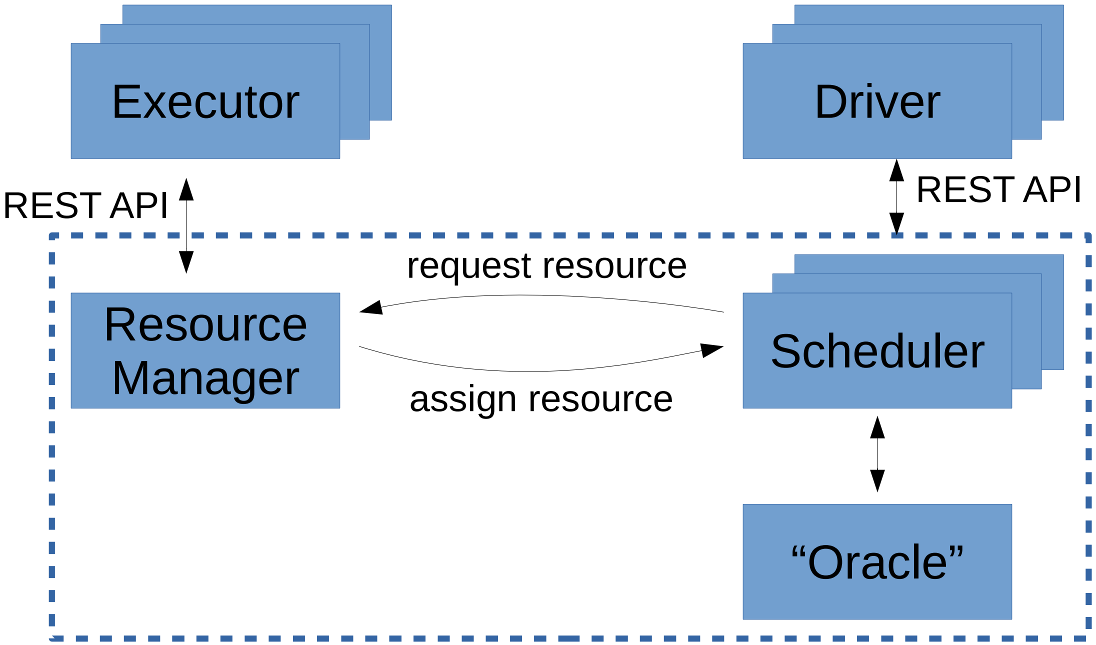


Architecture Overview

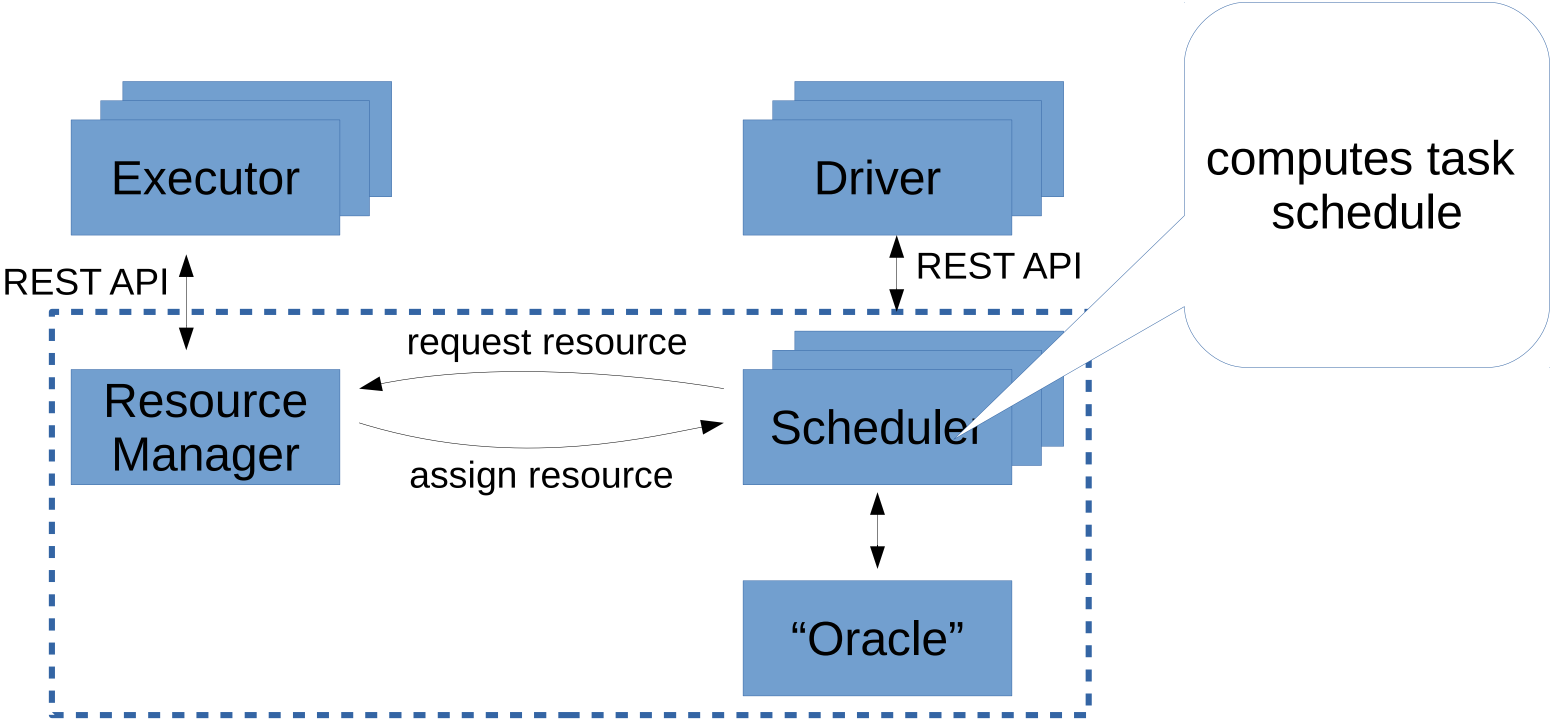


crail.apache.org

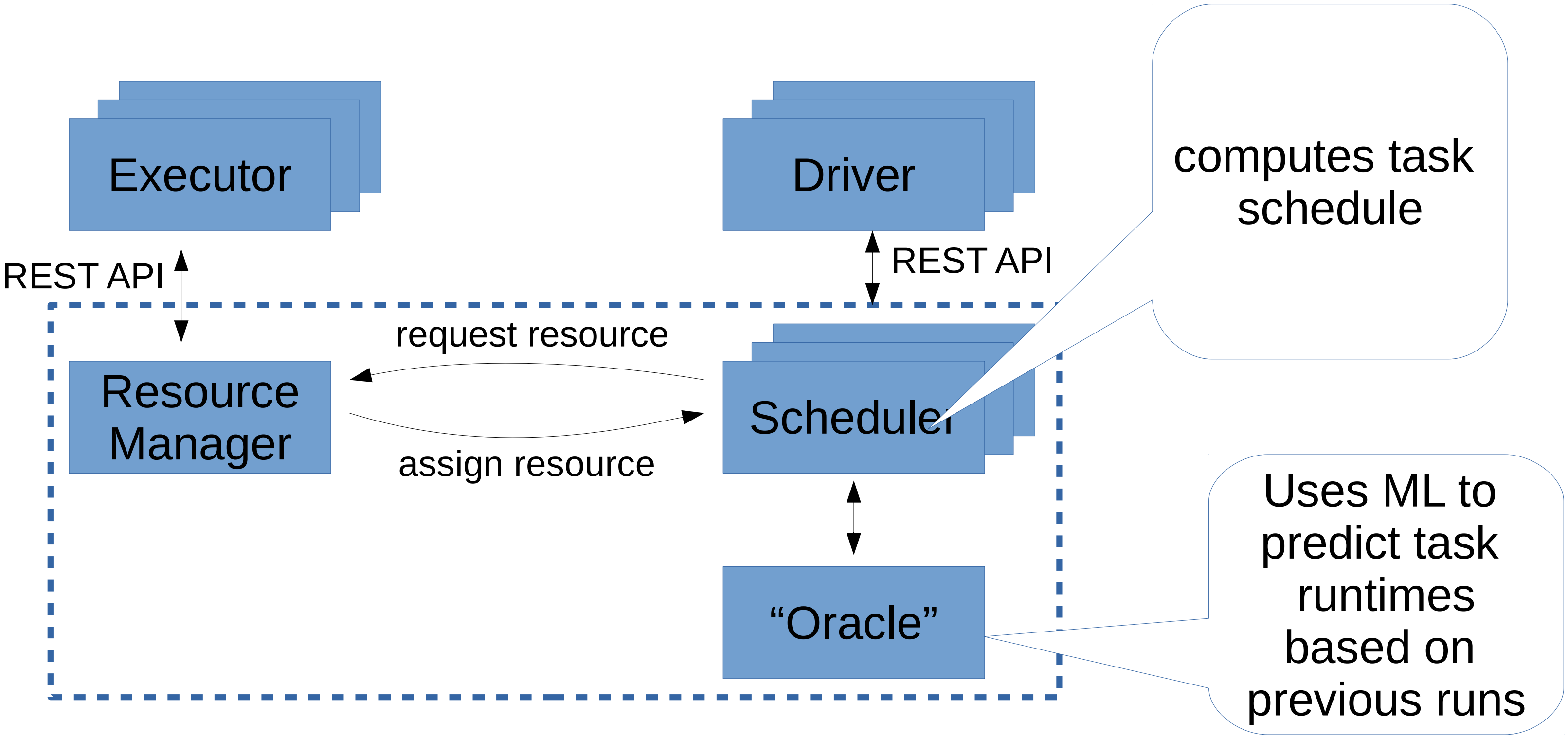
HCS Scheduler



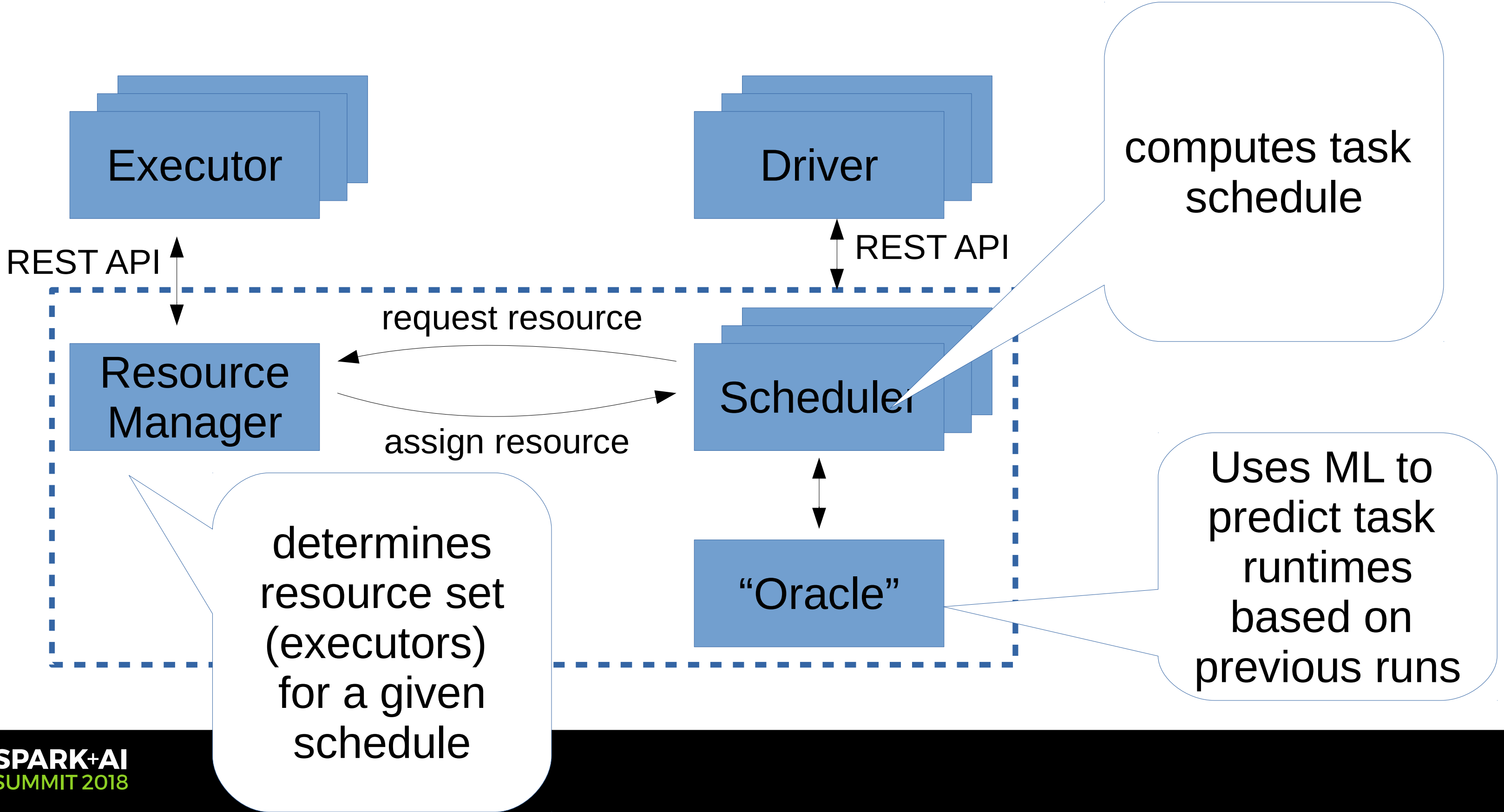
HCS Scheduler



HCS Scheduler

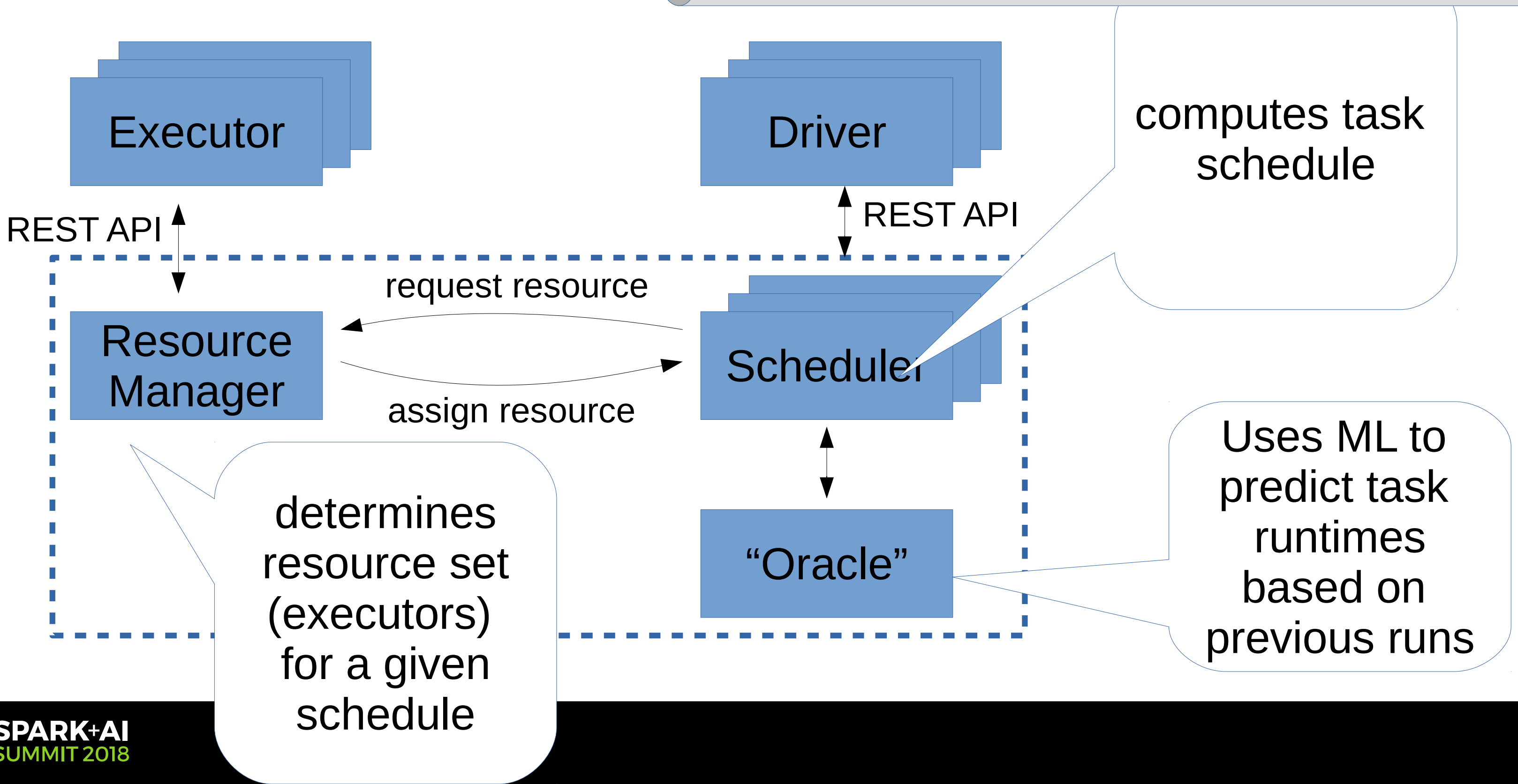


HCS Scheduler



HCS Scheduler

“The HCI Scheduler:
Going all-in on Heterogeneity”,
Michael Kaufmann et al., HotCloud’17



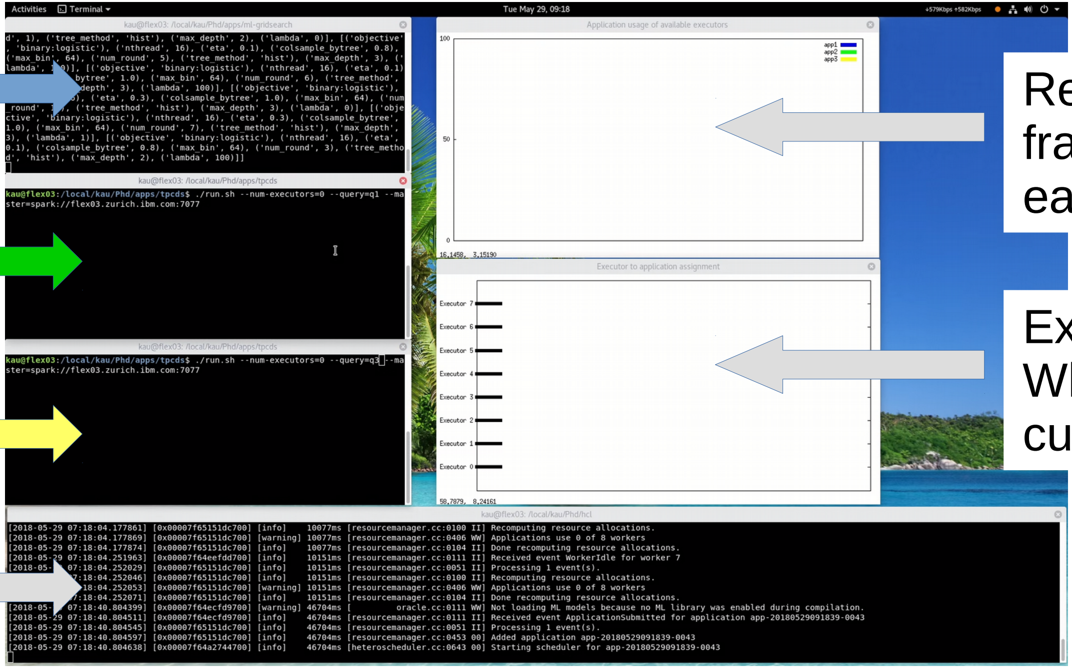
Video: Putting things together

Application 1:
GridSearch

Application 2:
SQL TPC-DS

Application 3:
SQL TPC-DS

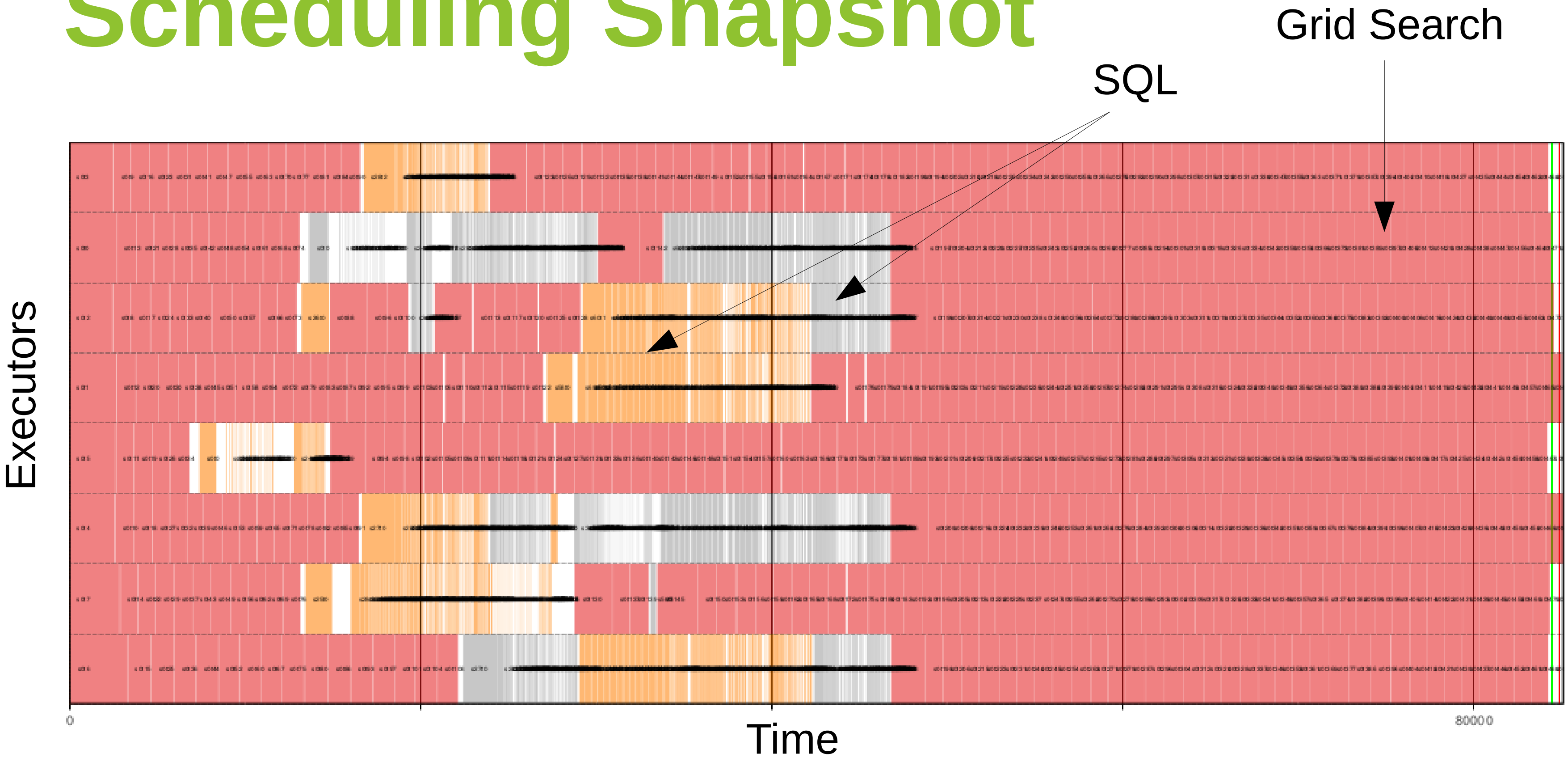
HCL
Scheduler



Resource view:
fraction of resources each app consumes

Executor view:
Which app an executor currently runs

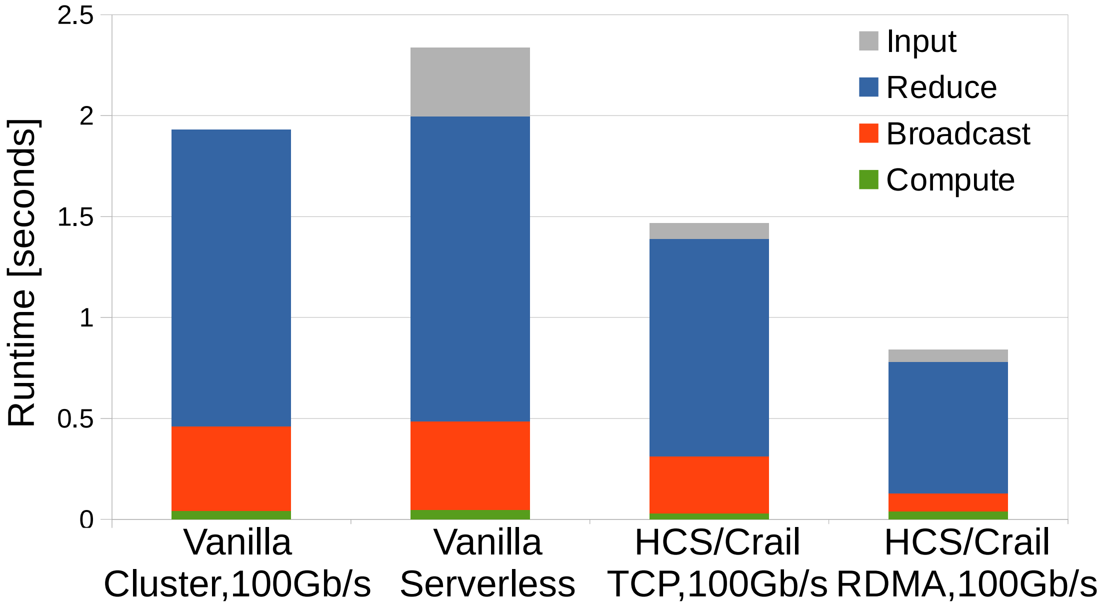
Scheduling Snapshot



Let’s look at performance...

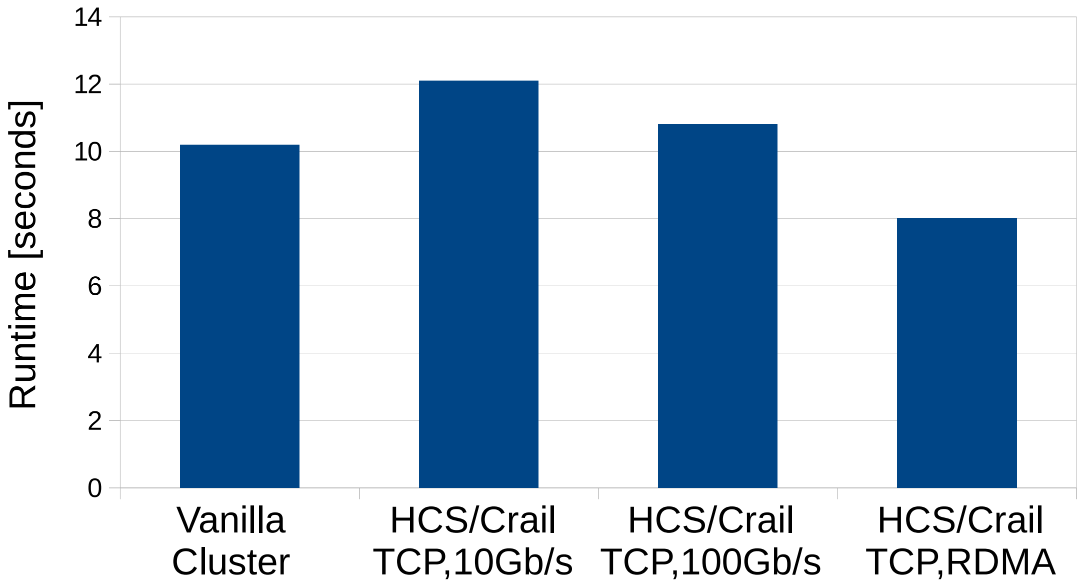
- Compute cluster size: 8 nodes: IBM Power8 Minsky
- Storage cluster size: 8 nodes, IBM Power8 Minsky
- Cluster hardware:
  - DRAM: 512 GB
  - Storage: 4x 1.2 TB NVMe SSD
  - Network: 10Gb/s Ethernert, 100Gb/s RoCE
  - GPU: NVIDIA P100, NVLink
- Workloads
  - ML: Logistic Regression using the CoCoa framework
  - SQL: TCP-DS

ML: Logistic Regression

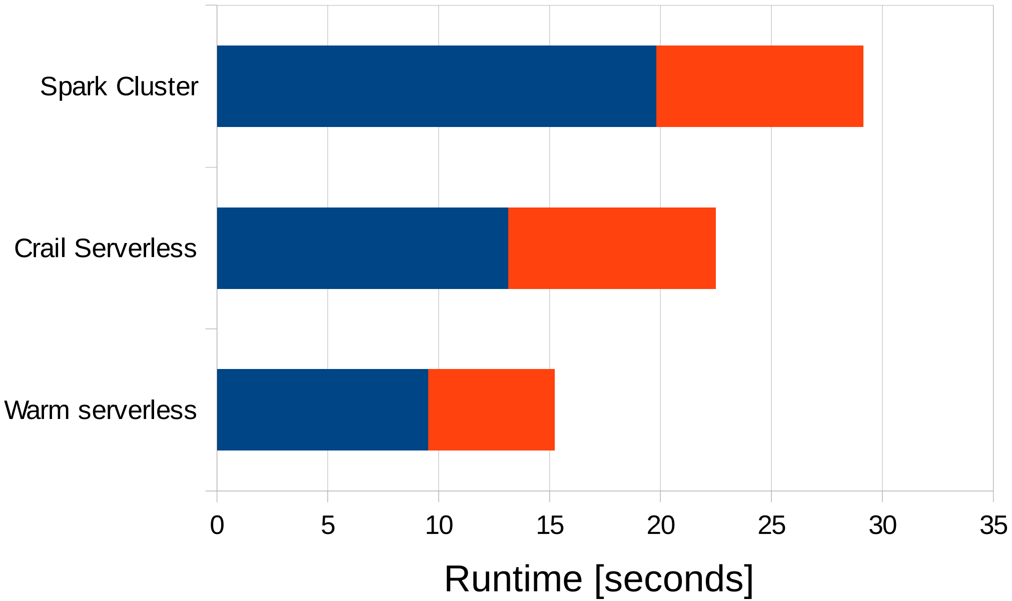


KDDA data set
6.5 GB

TPC-DS: Query #87



TPC-DS: Query #3



Conclusion

- Efficient serverless computing is challenging
  - Local state (e.g. shuffle, cached input, network state) is lost as compute cloud scales up/down
- This talk: turning Spark into a serverless framework by
  - Implementing HCS, a new serverless scheduler
  - Consequently storing compute state remotely using Apache Crail
- Supports arbitrary Spark workloads with almost no performance overhead
  - MapReduce, SQL, Iterative Machine Learning
- Implicit support for fast network and storage hardware
  - e.g, RDMA, NVMe, NVMf

Future Work

- Containerize the platform
- Add support for dynamic re-partitioning on scale events
- Add support for automatic caching
- Add more sophisticated scheduling policies

Links

- Running Apache Spark on a High-Performance Cluster Using RDMA and NVMe Flash, Spark Summit’17, https://tinyurl.com/yd453uzq
- Apache Crail, http://crail.apache.org
- HCS Scheduler, github.com/zrlio/hcs
- Spark-HCS, github.com/zrlio/spark-hcs
- Spark-IO, github.com/zrlio/spark-io

Thanks to

Michael Kaufmann, Adrian Schuepbach, Jonas Pfefferle, Animesh Trivedi, Bernard Metzler, Ana Klimovic, Yawen Wang