Apache REEF

Markus Weimer, Sergiy Matusevych
Microsoft Cloud and Information Services Lab
What is Apache REEF?

a library to simplify and unify the lower layers of big data systems on modern resource managers.
Resource Managers

- **One** cluster used by **all** workloads (interactive, batch, streaming, ...)
- Resources are handed out as containers
- Container is *slice* of a machine
  Fixed RAM, CPU, I/O, ...
- **Examples:**
  - Apache Hadoop YARN
  - Apache Mesos
  - Google Borg
Resource Managers

Enable true multi-tenancy...

Many workloads: Streaming, Batch, Interactive...

Many users: Production, Ad-Hoc, Experiments...

...but, only for sophisticated apps
Resource Managers

Enable true multi-tenancy...

Many workloads: Streaming, Batch, Interactive...

Many users: Production, Ad-Hoc, Experiments...

...but, only for sophisticated apps

Fault tolerance
Resource Managers

Enable true multi-tenancy...

Many **workloads**:
Streaming, Batch, Interactive...

Many **users**:
Production, Ad-Hoc, Experiments...

...but, only for sophisticated apps

Fault tolerance
Resource Managers

Enable true multi-tenancy...

Many **workloads**: Streaming, Batch, Interactive...

Many **users**: Production, Ad-Hoc, Experiments...

...but, only for sophisticated apps

Fault tolerance
Preemption
Enable true multi-tenancy...

Many **workloads**: Streaming, Batch, Interactive...

Many **users**: Production, Ad-Hoc, Experiments...

...but, only for sophisticated apps

Fault tolerance
Preemption
Elasticity
Example 1: SQL / MapReduce

Elasticity
Fault tolerance
Example 1: SQL / MapReduce

- Elasticity
- Fault tolerance
Example 1: SQL / MapReduce

Elasticity
Fault tolerance
Example 2: Machine learning

Fault tolerance
Elasticity
Iterative computations
Example 3: Graph processing

Fault tolerance
Elasticity
Iterative computations
Low latency communication
Silos are hard to build
Each duplicates the same mechanisms under the hood

In practice, silos form pipelines
• In each step: Read from and write to HDFS
• Synchronize on complete data between steps
→ Slow!
Fun new problems, boring old ones

Control flow setup
Master / Slave

Membership
Via heartbeats

Task Submission

Inter-Task Messaging
REEF

Breadth
Mechanism over Policy
Avoid silos
Recognize the need for different models
But: allow them to be composed
RM portability
Make app independent from low-level Resource Manager APIs.
Bridge the JVM/CLR/... divide
Different parts of the computation can be in either of them.

Resource Manager and DFS := Cluster OS
REEF := stdlib
Hadoop Ecosystem

- Pre-YARN
  - Application
    - Pig
    - Hive
  - Framework
  - MapReduce
  - Platform
    - MapReduce v1
  - Resource Management
    - Hardware / VMs
REEF Control Flow

Yarn (_circle) handles resource management (security, quotas, priorities)

Per-job Drivers (_head) request resources, coordinate computations, and handle events: faults, preemption, etc...
REEF Control Flow

Yarn (👨‍💻) handles resource management (security, quotas, priorities)

Per-job Drivers (👨‍💻) request resources, coordinate computations, and handle events: faults, preemption, etc.
REEF Control Flow

Yarn ( ) handles resource management (security, quotas, priorities)

Per-job Drivers ( ) request resources, coordinate computations, and handle events: faults, preemption, etc...

REEF Evaluators ( ) hold hardware resources, allowing multiple Tasks ( , , , , , , , , etc...) to use the same cached state.
REEF Control Flow

Yarn ( yarn icon ) handles resource management (security, quotas, priorities)

Per-job Drivers ( human icon ) request resources, coordinate computations, and handle events: faults, preemption, etc...

REEF Evaluators ( square icon ) hold hardware resources, allowing multiple Tasks ( star, light bulb, lock, clock, pi, sigma, etc...) to use the same cached state.
REEF Control Flow

Yarn ( ) handles resource management (security, quotas, priorities)

Per-job Drivers ( ) request resources, coordinate computations, and handle events: faults, preemption, etc...

REEF Evaluators ( ) hold hardware resources, allowing multiple Tasks ( , , , , , , etc...) to use the same cached state.
REEF Control Flow

Yarn (🧶) handles resource management (security, quotas, priorities)

Per-job Drivers (👤) request resources, coordinate computations, and handle events: faults, preemption, etc...

REEF Evaluators (🧶) hold hardware resources, allowing multiple Tasks (🌟,💡,🪝,⏱,π,σ, etc...) to use the same cached state.
REEF Control Flow

Yarn ( ) handles resource management (security, quotas, priorities)

Per-job Drivers ( ) request resources, coordinate computations, and handle events: faults, preemption, etc...

REEF Evaluators ( ) hold hardware resources, allowing multiple Tasks ( , , , , , , etc...) to use the same cached state.
REEF Control Flow

Yarn (ива) handles resource management (security, quotas, priorities)

Per-job Drivers (ива) request resources, coordinate computations, and handle events: faults, preemption, etc...

REEF Evaluators (ива) hold hardware resources, allowing multiple Tasks (*, ∏, ∑, π, σ, etc...) to use the same cached state.
Control Flow Take-Away

Easy to reason about

**Centralized control flow**
Evaluator allocation & configuration
Task configuration & submission

**Centralized error handling**
Task exceptions are thrown at the Driver
Evaluator failure is reported to the Driver

Scalable

**Event-Based Programming**
Driver fires requests as events to REEF
REEF fires events to the Driver

**Mostly stateless design**
REEF maintains minimal state
Majority of the state keeping (e.g. work queues) is maintained by the Driver.
Hello World!
Hello World

1. Client submits the Driver allocation request to the Resource Manager (зи)
1. Client submits the Driver allocation request to the Resource Manager ( WebSocket icon).

2. Once started, Driver ( WebSocket icon) requests one Evaluator container from YARN.
1. Client submits the Driver allocation request to the Resource Manager (เทคนิ)!
2. Once started, Driver ((457,560),(510,601)) requests one Evaluator container from YARN
3. Driver submits HelloWorld Task to the newly allocated Evaluator
Hello World

1. Client submits the Driver allocation request to the Resource Manager (🎉)
2. Once started, Driver (🎉) requests one Evaluator container from YARN
3. Driver submits a HelloWorld Task to the newly allocated Evaluator
4. HelloWorld Task prints a greeting to the log and quits

```java
LOG.log(Level.INFO, "Hello REEF!");
```
Hello World

1. Client submits the Driver allocation request to the Resource Manager
2. Once started, Driver requests one Evaluator container from YARN
3. Driver submits a HelloWorld Task to the newly allocated Evaluator
4. HelloWorld Task prints a greeting to the log and quits
5. Driver receives CompletedTask notification, releases the Evaluator and quits
public final class HelloTask implements Task {

    @Inject
    private HelloTask() {} 

    @Override
    public byte[] call(final byte[] memento) {
        LOG.log(Level.INFO, "Hello REEF!");
        return null;
    }
}
public final class StartHandler implements EventHandler<StartTime> {
    @Override
    public void onNext(final StartTime startTime) {
        requestor.submit(EvaluatorRequest.newBuilder()
                .setNumber(1).setMemory(64).setNumberOfCores(1).build());
    }
}

public final class EvaluatorAllocatedHandler implements EventHandler<AllocatedEvaluator> {
    @Override
    public void onNext(final AllocatedEvaluator allocatedEvaluator) {
        allocatedEvaluator.submitTask(TaskConfiguration.CONF
                .set(TaskConfiguration.IDENTIFIER, "HelloREEFTask")
                .set(TaskConfiguration.TASK, HelloTask.class)
                .build());
    }
}
```java
final Configuration runtimeConfig = YarnRuntimeConfiguration.CONF.build();

final Configuration driverConfig = DriverConfiguration.CONF
  .set(DriverConfiguration.DRIVER_IDENTIFIER, "HelloREEF")
  .set(DriverConfiguration.GLOBAL_LIBRARIES, EnvironmentUtils.getClassLocation(HelloDriver.class))
  .set(DriverConfiguration.ON_DRIVER_STARTED, HelloDriver.StartHandler.class)
  .set(DriverConfiguration.ON_EVALUATOR_ALLOCATED, HelloDriver.EvaluatorAllocatedHandler.class)
  .build();

DriverLauncher.getLauncher(runtimeConfig)
  .run(driverConfig, JOB_TIMEOUT);
```
Tang

Configuration is hard
- Errors often show up at runtime only
- State of receiving process is unknown to the configuring process

Command = ‘ls’

Error:
container-487236457-02.stderr:
NullPointerException at:
java...eval():1234
ShellTask.helper():546
ShellTask.onNext():789
YarnEvaluator.onNext():12
Tang

Configuration is hard
- Errors often show up at runtime only
- State of receiving process is unknown to the configuring process

Our approach:
- Use Dependency Injection
- Configuration is pure data
  → Early static and dynamic checks

Command = ‘ls’

Error:
Unknown parameter "Command"
Missing required parameter "cmd"
Configuration is hard
- Errors often show up at runtime only
- State of receiving process is unknown to the configuring process

Our approach:
- Use Dependency Injection
- Configuration is pure data
  → Early static and dynamic checks
Configuration is hard
- Errors often show up at runtime only
- State of receiving process is unknown to the configuring process

Our approach:
- Use Dependency Injection
- Configuration is pure data
  → Early static and dynamic checks
Wake: Events + I/O

Event-based programming and remoting

API: A static subset of Rx
- Static checking of event flows
- Aggressive JVM event inlining

Implementation: “SEDA++”
- Global thread pool
- Thread sharing where possible
Distributed Shell
Distributed Shell

1. Client submits Driver configuration to YARN runtime. Configuration has *cmd* shell command, and *n* number of Evaluators to run it on
Distributed Shell

1. Client submits Driver configuration to YARN runtime. Configuration has *cmd* shell command, and *n* number of Evaluators to run it on.

2. Once started, Driver requests *n* Evaluators from YARN.
Distributed Shell

1. Client submits Driver configuration to YARN runtime. Configuration has \textit{cmd} shell command, and \textit{n} number of Evaluators to run it on

2. Once started, Driver requests \textit{n} Evaluators from YARN

3. Driver submits \texttt{ShellTask} with \textit{cmd} to each Evaluator
Distributed Shell

1. Client submits Driver configuration to YARN runtime. Configuration has *cmd* shell command, and *n* number of Evaluators to run it on
2. Once started, Driver requests *n* Evaluators from YARN
3. Driver submits ShellTask with *cmd* to each Evaluator
4. Each ShellTask runs the command, logs its stdout, and quits
Distributed Shell

1. Client submits Driver configuration to YARN runtime. Configuration has `cmd` shell command, and `n` number of Evaluators to run it on.
2. Once started, Driver requests `n` Evaluators from YARN.
3. Driver submits ShellTask with `cmd` to each Evaluator.
4. Each ShellTask runs the command, logs its stdout, and quits.
5. Driver receives CompletedTask notifications, releases Evaluators and quits when all Evaluators are gone.
@Inject
private ShellTask(@Parameter(Command.class) final String command) {
    this.command = command;
}

@Override
public byte[] call(final byte[] memento) {
    final String result = CommandUtils.runCommand(this.command);
    LOG.log(Level.INFO, result);
    return CODEC.encode(result);
}
Distributed Shell – Parameters

@NamedParameter(doc="Number of evaluators", short_name="n", default_value="1")
public final class NumEvaluators implements Name<Integer> {}

@NamedParameter(doc="The shell command", short_name="cmd")
public final class Command implements Name<String> {}
public final class EvaluatorAllocatedHandler implements EventHandler<AllocatedEvaluator> {

    @Override
    public void onNext(final AllocatedEvaluator allocatedEvaluator) {

        final JavaConfigurationBuilder taskConfigBuilder =
            tang.newConfigurationBuilder(TaskConfiguration.CONF
                .set(TaskConfiguration.IDENTIFIER, "ShellTask")
                .set(TaskConfiguration.TASK, ShellTask.class)
                .build());

        taskConfigBuilder.bindNamedParameter(Command.class, command);

        allocatedEvaluator.submitTask(taskConfigBuilder.build());
    }
}
final JavaConfigurationBuilder driverConfig =
    tang.newConfigurationBuilder(DriverConfiguration.CONF
        .set(DriverConfiguration.DRIVER_IDENTIFIER, "DistributedShell")
        .set(DriverConfiguration.GLOBAL_LIBRARIES,
            EnvironmentUtils.getClassLocation(ShellDriver.class))
        .set(DriverConfiguration.ON_DRIVER_STARTED, ShellDriver.StartHandler.class)
        .set(DriverConfiguration.ON_EVALUATOR_ALLOCATED,
            ShellDriver.EvaluatorAllocatedHandler.class)
    .build());

new CommandLine(driverConfigBuilder)
    .registerShortNameOfClass(Command.class)
    .registerShortNameOfClass(NumEvaluators.class)
    .processCommandLine(args);

DriverLauncher.getLauncher(runtimeConfig).run(driverConfig.build(), JOB_TIMEOUT);
Distributed Shell: apples to apples

YARN Example: 1333 lines of code
Distributed Shell: apples to apples

YARN Example: 1333 lines of code
Simple REEF Application: 83 lines

94% of YARN Distributed Shell code is boilerplate
Distributed Shell: apples to apples

YARN Example: 1333 lines of code
Simple REEF Application: 83 lines
Interactive Fault-Tolerant Web Application on REEF: 361 lines

Still, only 27% of YARN example size
But wait, there’s more!

- **Stackable data contexts**
  Retain data between tasks
- **Injectable services**
  Drop-in services, like HTTP Endpoint, Chaos Monkey, etc.
- **Data Loader**
  Iterate over partitioned data on DFS
- **Group Communications**
  MPI-style Broadcast and Reduce operations
- **Java ↔ .NET interoperability**
  Mix and match C# and Java code for client, driver, and tasks
Group Communications
Group Communication

Also: Collective Communication

Communications with many participants
Contrast: Peer-to-peer communication

Most commonly used interface: MPI
Broadcast

Mechanism
The sender (S) sends a value
All receivers (R) receive the identical value

Use case
Distribute a model
Distribute a descent direction for line search

Optimizations
Trees to distribute the data
Be mindful of the topology of machines
Do peer-to-peer sends
...
Reduce

Mechanism
The senders (S) each send a value
Those values are aggregated
The receiver (R) receives the aggregate total

Use case
Aggregate gradients, losses

Optimizations
Aggregation trees
Pipelining
Group Communications in REEF

Focus: Elasticity
The operators tolerate addition an removal of Tasks (elasticity events)

Voluntary Elasticity
Adding and removing of nodes if and when the application chooses

Involuntary Elasticity
Failure
Soft Preemption: An application is asked to vacate some machines
Hard Preemption: The resource manager takes machines away
Involuntary Elasticity: Approach

Handled quietly
When machines disappear, operators proceed
Broadcast: It is as if the receiver never was present
Reduce: It is as if the failing sender never sent anything

Query for topology changes
The application can query for topology changes
Result: Changes since last time
We can send and receive any Java serializable data, e.g. jBLAS matrices.

```java
private final Broadcast.Sender<DoubleMatrix> modelSender;
private final Broadcast.Receiver<DoubleMatrix[]> resultReceiver;
```

// Broadcast the model, collect the results, repeat.

```java
do {
    this.modelSender.send(modelSlice);
    final DoubleMatrix[] result = this.resultReceiver.reduce();
} while (notConverged(modelSlice, prevModelSlice));
```
REEF Status

Top-level Apache Project
~20 committers
Contributors from Microsoft, UW, UCB, PureStorage, UCLA, SNU, SK Telecom
Participates in Google Summer of Code 2016
http://reef.apache.org

Releases
Java: Maven artifacts published to Maven Central
C#: NuGets published to NuGet.org
Version 0.14.0 released in March, 0.15.0 coming out soon
REEF in Production

Azure Stream Analytics
Stream processing engine *(Microsoft, .NET)*

Azure ML
Job execution on the single node runtime *(Microsoft, .NET)*

SURF
Elastic in-memory caching tier for HDFS *(SK Telecom, Java)*
Research on REEF

Resource elastic DAG scheduling
Tyson Condie, UCLA; Java

Groupers: Generalized data movement
Gon Chun, SNU & Rusty Sears, PureStorage; Java

REEFU: Corfu on REEF
Dahlia Malkhi, VMWare; Java

Geo-distributed ML
Ignacio “Nacho” Cano, UW; Java

Dolphin: Distributed deep learning framework and parameter server
Beomyeol Jeon, Joo Seong Jeon et al., SNU; Java
http://github.com/cmssnu/dolphin

Elastic execution of Infer.NET models
MSR Cambridge, Alex Beutel, CMU; .NET

Tensor Factorization
Paul Mineiro, Nikos Karampatziakis, Sergiy Matusevych with Anima Anandkumar and Furong Huang, UCI; Java
http://github.com/Microsoft-CISL/TensorFactorization-LDA

Hashing PCA
Paul Mineiro, Nikos Karampatziakis with Arun Kumar, Wisconsin; Java

Elastic group communications
Shravan Narayanamurthy, Dhruv Mahajan; Java and .NET

Elastic batch learning of linear models
Dhruv Mahajan, Keerthi Selvaraj, Shravan Narayanamurthy; Java and .NET
Thanks!

Contact  dev@reef.apache.org

Website  http://reef.apache.org
Bonus

Let’s do some machine learning!
Linear learning

Given: Dataset $X$ of Examples

Each Example consists of
a Feature Vector $x$ (e.g. words in a document) and
a Label $y$ (e.g. the fraction of people that clicked it).

Desired: A predictor

$$f(x) := f_w(x) := \langle w, x \rangle = \sum_i w_i x_i$$

Hence, we search for:

$$\hat{w} = \arg\min_w \sum_{(x,y)\in X} (f_w(x) - y)^2 + \Theta(f_w) = \sum_{(x,y)\in X} \frac{1}{2} (\langle w, x \rangle - y)^2 + \frac{\lambda}{2} \|w\|^2$$
The Algorithm

Start with a random $w_0$

**Until** convergence:

Compute the gradient

$$\partial_w = \sum_{(x,y) \in X} (\langle w, x \rangle - y)$$

Apply gradient and regularizer to the model

$$w_{t+1} = w_t - \eta (\partial_w + \lambda \|w\|)$$
On REEF

Driver requests Evaluators
On REEF

Driver requests Evaluators

Driver sends Tasks to load & parse data
On REEF

Driver requests Evaluators
Driver sends Tasks to load & parse data
Driver sends ComputeGradient and master Tasks
On REEF

Driver requests Evaluators
Driver sends Tasks to load & parse data
Driver sends ComputeGradient and master Tasks
Computation commences in sequence of Broadcast and Reduce
Ramp-up
Why the wait to start my application?

Sometimes, it is a feature, not a bug

The goal of the resource manager is high utilization
The more applications support preemption the hotter (utilized) the cluster will be
It takes time for the RM to vacate containers for you to use

Sometimes it is a bug

Not all machines are equally fast
Not all network links are equally fast
→ Some containers will take longer to load the data than others

Perfect storm

You have to wait for the last container to load the data, which might even be the slowest one

(In MapReduce, this is known as the straggler problem, typically one Reducer is slow)
Start with a random $w_0$

Until convergence:

Compute the gradient

$$\nabla w = \sum_{(x,y) \in X} 2 ((w, x) - y)$$

Apply gradient to the model

$$w_{t+1} = w_t - \nabla w$$

Not having some machines means training on a (random) subset of $X$
On REEF

First iteration

Continuously negotiates with YARN for more containers and loads data into them
On REEF

Second Iteration

Queries for topology changes.
On REEF

End state
Is it any good?