Build Large Scale Applications in YARN with

Henry Saputra (@Kingwulf) - hsaputra@apache.org
Terence Yim (@chtyim) - chtyim@apache.org

Apache Big Data Conference - North America - 2016
Disclaimer

Apache Twill is an effort undergoing incubation at The Apache Software Foundation (ASF), sponsored by Incubator. Incubation is required of all newly accepted projects until a further review indicates that the infrastructure, communications, and decision making process have stabilized in a manner consistent with other successful ASF projects. While incubation status is not necessarily a reflection of the completeness or stability of the code, it does indicate that the project has yet to be fully endorsed by the ASF.
Agenda

● Why Apache Twill?
● Architecture and Components
● Features
● Real World Enterprise Usages - CDAP
● Roadmap
● Q & A
Apache Hadoop® YARN

- MapReduce NextGen aka MRv2
- New ResourceManager manages the global assignment of compute resources to applications
- Introduce concept of ApplicationMaster per application to communicate with ResourceManager for compute resource management
Apache Hadoop ® YARN Architecture
Developing Application in YARN

- It is actually not as “simple” as it sounds
- Lots of boilerplates code with very steep learning curve
- Given the power and generic nature of YARN, developing applications directly on top of YARN could be very difficult

- **Standard components:**
  - Application Client
  - Application Master
  - Application
How YARN Application Works

1. Submit App Master
2. Start App Master in a Container
3. Request Containers
4. Start Tasks in Containers
Apache Twill

- Add simplicity to the power of YARN
  - Java thread-like programming model
  - Instead of running in multiple threads, it runs in many containers in YARN
- Incubated at Apache Software Foundation since November 2013
  - Has successfully created seven releases
public class HelloWorld {
    public static class HelloWorldRunnable extends AbstractTwillRunnable {
        @Override
        public void run() {
            LOG.info("Hello World. My first distributed application.");
        }
    }

    public static void main(String[] args) throws Exception {
        TwillRunnerService twillRunner = new YarnTwillRunnerService(new YarnConfiguration(), "localhost:2181");
        twillRunner.startAndWait();
        TwillController controller = twillRunner.prepare(new HelloWorldRunnable())
            .addLogHandler(new PrinterLogHandler(new PrintWriter(System.out, true)))
            .start();
        try {
            controller.awaitTermination();
        } catch (Exception ex) {
            ...  
        }
    }
}
Why Apache Twill

- Apache Twill provides abstraction and virtualization for YARN to reduce complexity to develop complex and distributed large scale applications
- Apache Twill allows developers to leverage the power of YARN by offering programming paradigms
- Apache Twill offers commons needs for distributed large scale application development
  - Lifecycle management
  - Service discovery
  - Distributed coordination and resiliency to failures
  - Real-time Logging
Architecture and Components

This is the only programming interface you need.
Main Apache Projects Used

- Apache Hadoop YARN
- Apache Hadoop HDFS
- Apache Zookeeper
- Apache Kafka
Main Features

1. Real-time logging
2. Resource Report
3. State Recovery
4. Elastic Scaling
5. Command Messages
6. Service Discovery
Notable New Features

1. 0.5.0-incubating
   a. Placement Policy APIs
   b. Submitting to non default YARN Queue
   c. Distributed Lock
2. 0.6.0-incubating
   a. Restart instances of runnables for Twill applications
   b. MapR Extension
   c. Remove Guava Dependencies from client APIs
3. 0.7.0-incubating
   a. Allow setting environment variable on Twill containers
   b. Support for Azure Blob Storage
Real-time Logging

YARN Resource Manager

LogHandler to receive logs from all runnables

Twill Runner

Twill Client

Node Mgr

Task

AM+ Kafka

log stream

log

log appender to send logs to Kafka
/**
 * This interface provides a snapshot of the resources an application is using
 * broken down by each runnable.
 */

public interface ResourceReport {
    // Get all the run resources being used by all instances of the specified runnable.
    Collection<TwillRunResources> getRunnableResources(String runnableName);

    // Get all the run resources being used across all runnables.
    Map<String, Collection<TwillRunResources>> getResources();

    // Get the resources application master is using.
    TwillRunResources getAppMasterResources();

    // Get the id of the application master.
    String getApplicationId();

    // Get the id of the application master.
    List<String> getServices();
}
/**
 * Information about the container the {@link TwillRunnable} is running in.
 */
public interface TwillRunResources {

    int getInstanceId();

    int getVirtualCores();

    int getMemoryMB();

    String getHost();

    String getContainerId();

    Integer getDebugPort();

    LogEntry.Level getLogLevel();

}
• Client get the resource report from Twill using the `TwillController.getResourceReport` API to return resource reporting

```java
public interface TwillController extends ServiceController {
    ...

    /**
     * Get a snapshot of the resources used by the application, broken down by each runnable.
     *
     * @return A `ResourceReport` containing information about resources used by the application or null in case the user calls this before the application completely starts.
     */
    @Nullable
    ResourceReport getResourceReport();

    ...
}
```
State Recovery

Recover state from ZooKeeper

- Twill Runner
- Twill Client
- YARN Resource Manager
  - Node Mgr
  - AM+ Kafka
  - Task
  - Task
- ZooKeeper
  - Task
  - Task
Command Messages

Send command message to runnables

Twill Runner

Twill Client

YARN Resource Manager

Node Mgr

AM+ Kafka

Node Mgr

Task

message

Task

Task

Task

Task
Elastic Scaling

- Ability to add or reduce number of YARN containers to run the application
- Twill API `TwillController.changeInstances` is used to accomplish this task

```java
/**
 * Changes the number of running instances of a given runnable.
 *
 * @param runnable The name of the runnable.
 * @param newCount Number of instances for the given runnable.
 * @return A `Future` that will be completed when the number running instances has been
 * successfully changed. The future will carry the new count as the result. If there is any error
 * while changing instances, it'll be reflected in the future.
 *
 * Future<Integer> changeInstances(String runnable, int newCount);
 */
```
Service Discovery

Watch for change in discovery nodes

Service changes

Register service

Twill Runner

Twill Client

YARN Resource Manager

Node Mgr

Task

AM+Kafka

Node Mgr

Task

Node Mgr

Task

Node Mgr

Task

ZooKeeper

Task

Task
Placement Policy API - 1 (New)

- Expose container placement policy from YARN
- Will allow Twill to allocate containers in specific racks and host based on DISTRIBUTED deployment mode
/**
 * Defines a container placement policy.
 */

interface PlacementPolicy {
    enum Type {
        DISTRIBUTED, DEFAULT
    }

    Set<String> getNames();

    Type getType();

    Set<String> getHosts();

    Set<String> getRacks();
}
Instances of TwillRunnable will be run in YARN containers
Each Twill application could have one or more instances of TwillRunnable
Twill provides ability to restart particular runnable instance without affecting other runnables
This is useful when certain runnables are not running well and you would need to restart certain instances based on the identifier
Restart Instances for Twill Runnables - 2 (New)

/**
 * For controlling a running application.
 */
public interface TwillController extends ServiceController {

  ...

  Future<String> restartAllInstances(String runnable);

  Future<Set<String>> restartInstances(Map<String, ? extends Set<Integer>> runnableToInstanceIds);

  Future<String> restartInstances(String runnable, int instanceId, int... moreInstanceIds);

  ...
}
Setting Environment Variables on Containers (New)

- Provides ability to set environment variables on the YARN containers where TwillRunnable instances are running

```java
/**
 * This interface exposes methods to set up the Twill runtime environment and start a Twill application.
 */
public interface TwillPreparer {
    ...

    // Adds the set of environment variables that will be set as container environment variables for all runnables.
    TwillPreparer withEnv(Map<String, String> env);

    /**
     * Adds the set of environment variables that will be set as container environment variables for the given runnable.
     * Environment variables set through this method has higher precedence than the one set through {@link #withEnv(Map)}
     * if there is a key clash.
     */
    TwillPreparer withEnv(String runnableName, Map<String, String> env);

    ...
}
```
Real World Enterprise Usages - CDAP

- Cask Data Application Platform (CDAP) - [http://cdap.io](http://cdap.io)
  - Open source data application framework
  - Simplifies and enhances data application development and management
    - APIs for simplification, portability and standardization
      - Works across wide range of Hadoop versions and all common distros
    - Built-in System services, such as metrics and logs aggregation, dataset management, and distributed transaction service for common big data applications needs
  - Extensions to enhance user experience
    - Hydrator - Interactive data pipeline construction
    - Tracker - Metadata discovery and data lineage
CDAP Logical View

Application

CDAP Client API

Applications

CDAP Developer API

Application Abstraction

Data Abstraction

Infrastructure

Clients

- UI
- HTTP Client
- Cmd Line Client
- Python Client

CDAP

Hadoop Stack

- YARN
- HDFS
- HBase
- ZooKeeper

- Map/Reduce
- Spark

- Service Discovery
- Life Cycle

- Streams
- Transactions
- Datasets

- Program Containers
- Monitoring

- Ad-hoc Queries
Apache Twill in CDAP

- CDAP runs different types of work on YARN
  - Long running daemons
  - Real-time transactional streaming framework
  - REST services
  - Workflow execution

- CDAP only interacts with Twill
  - Greatly simplifies the CDAP code base
  - Just a matter of minutes to add support for new type of work to run on YARN
  - Native support of common needs
    - Service discovery
    - Leader election and distributed locking
    - Elastic scaling
    - Security
CDAP Architecture Diagram
Service Discovery

- CDAP exposes all functionalities through REST
- Almost all CDAP HTTP services are running in YARN
  - No fixed host and port.
  - Bind to ephemeral port
  - Announce the host and port through Twill
    - Unique service name for a given service type
- Router inspects the request URI to derive a service name
  - Uses Twill discovery service client to locate actual host and port
  - Proxy the request and response
Long Running Applications

- All CDAP services on YARN are long running
  - Transaction server, metrics and log processing, real-time data ingestion, ...
- Many user applications are long running too
  - Real-time streaming, HTTP service, application daemon
- Not too big of a problem in non-secure cluster
  - Logs not collected, log files may get too big, ...
    - Twill build-in log collections can help
- Secure cluster, specifically Kerberos enabled cluster
  - All all Hadoop services use delegation token
    - NN, RM, HBase Master, Hive, KMS, ...
  - YARN containers doesn’t have the keytab, and it should not, hence can’t update the token
Long Running Applications in Twill

- Twill provides native support for updating delegation tokens
  - TwillRunner.scheduleSecureStoreUpdate

- Update delegation tokens from the launcher process (kinit process)
  - Acquires new delegation tokens periodically
  - Serializes tokens to HDFS
  - Notifies all running applications about the update
    - Through command message
  - Each runnable refreshes delegation tokens by reading from HDFS
    - Requires a non-expired HDFS delegation token

- New launcher process will discovery all Twill apps from ZK
  - Can run HA launcher processes using leader election support from Twill
Scalability

- Many components in CDAP are linearly scalable, such as
  - Streaming data ingestion (through REST endpoint)
  - Log processing
    - Reads from Kafka, writes to HDFS
  - Metrics processing
    - Reads from Kafka, writes to timeseries table
  - User real-time streaming DAG
  - User HTTP service
- Twill supports adding/reducing YARN containers for a given TwillRunnable
  - No need to restart application
  - Guarantees a unique instance ID is assigned
    - Application can use it for partitioning
High Availability

- In production environment, it is important to have high availability
- Twill provides couple means to achieve that
  - Running multiple instances of the same TwillRunnable
  - Use dynamic service discovery to route requests
  - Twill Automatic restart of TwillRunnable container if it gets killed / exit abnormally
    - Killed container will be removed from the service discovery
    - Restarted container will be added to the service discovery
  - Built-in leader election support to have active-passive type of redundancy
    - Tephra service use that, as it requires only having one active server
  - Built-in distributed lock to help synchronization
    - Synchronize when there is configuration changes among TwillRunnable instances
Placement Policy

- CDAP can run multiple instances for a given service type
  - Scalability
  - Redundancy for availability
- YARN doesn’t expect applications care where containers run
  - Can provide location hint, but is not guaranteed
  - Depends on the YARN scheduler
- CDAP uses Twill to control container placement
  - Different instances of the same TwillRunnable runs on different host
  - Certain TwillRunnable cannot run on the same host
    - Stream handler, Tephra transaction server
      - Both are heavy CPU and IO bound
Performance and Load Testing

- We perform load testing for CDAP components
  - Real-time stream ingestion handler
  - Tephra transaction server
- A scalable load testing framework written using Twill
  - Multiple REST clients in each TwillRunnable
    - One client per thread
  - Can gradually increase number of threads as well as number of containers
    - Use command message to increase threads
    - Use elastic scaling API to increase number of containers
  - Collect metrics through log messages
    - Use the built-in log collection support
Apache Twill in Enterprise

- CDAP, which uses Twill, is being used by large enterprise in production
- Apache Twill is proven framework
  - Has been running on different cluster configurations
    - AWS, Azure, bare metal, VMs
- Compatible with wide range of Hadoop versions
  - Vanilla Hadoop 2.0 - 2.7
  - HDP 2.1 - 2.3
  - CDH 5
  - MapR 4.1 - 5.1
Roadmap

- Expose newly added YARN features
- Smarter containers management
  - Run simple runnable in AM
  - Multiple runnables in one container
- Speedup application launch time
- Fine-grained control of containers lifecycle
  - When to start, stop and restart on failure
- Simple application launching with better classloader isolation
- Smaller footprint
  - Optional Kafka, optional ZooKeeper
- Generalize to run on more frameworks
  - Apache Mesos, Kubernetes
Thank you!

- Twill is Open Source and needs your contributions
  - http://twill.incubator.apache.org
  - dev@twill.incubator.apache.org
  - @ApacheTwill
- Contributions are welcomed!