Pluggable Shuffle Manager

Zhijiang Wang Andrey Zagrebin

Status

Current state: IN DISCUSSION

Discussion thread: https://lists.apache.org/thread.html/73d7c62a6c6dc7d3fddda79701c71b921e635766737c1b33b84df234@%3Cdev.flink.apache.org%3E

JIRA: FLINK-10653

Released: Flink 1.9+

FLIP

Motivation

Shuffle is the process of data transfer between stages, which involves in writing outputs on producer side and reading inputs on consumer side. The shuffle architecture and behavior in Flink are unified for both streaming and batch jobs. It can be improved in two dimensions:

- **Lifecycle of TaskExecutor (TM)/Task/ResultPartition:** TM starts an internal shuffle service for transporting partition data to consumer side. When task enters FINISHED state, its produced partition might not be fully consumed. Therefore TM container should not be freed until all the internal partitions consumed. It is obvious that there exists coupled implicit constraints among them, but has no specific mechanism for coordinating them work well.

- **Extension of writer/reader**: ResultPartition can only be written into local memory for streaming job and single persistent file per subpartition for batch job. It is difficult to extend partition writer and reader sides together based on current architecture. E.g. ResultPartition might be written in sort&merge way or to external storage. And partition might also be transported via external shuffle service on YARN, Kubernetes etc in order to release TM early.

Proposed Change

We propose a pluggable ShuffleManager architecture for managing partitions on JobMaster (JM) side and extending adaptive writer/reader on TM side.

(1) Shuffle Manager

```
public interface ShuffleManager {
    ShuffleMaster createMaster(Configuration flinkConfig);
    ShuffleService createService(Configuration flinkConfig);
}
```

- ShuffleManager acts as a factory for creating ShuffleMaster (JM side) and ShuffleService (TM side). Flink config could also contain specific shuffle configuration like port etc.
- Specific ShuffleManager implements how to communicate interactively between ShuffleMaster and ShuffleService. If shuffle is channel-based it can behave in a similar way as now.
- We could support cluster level config for ShuffleManager class name in the first version. Later we could further support job or edge level config by introducing predefined ShuffleType. Cluster config could contain all provided ShuffleManager implementations for each supported ShuffleType or fallback to default for some types.

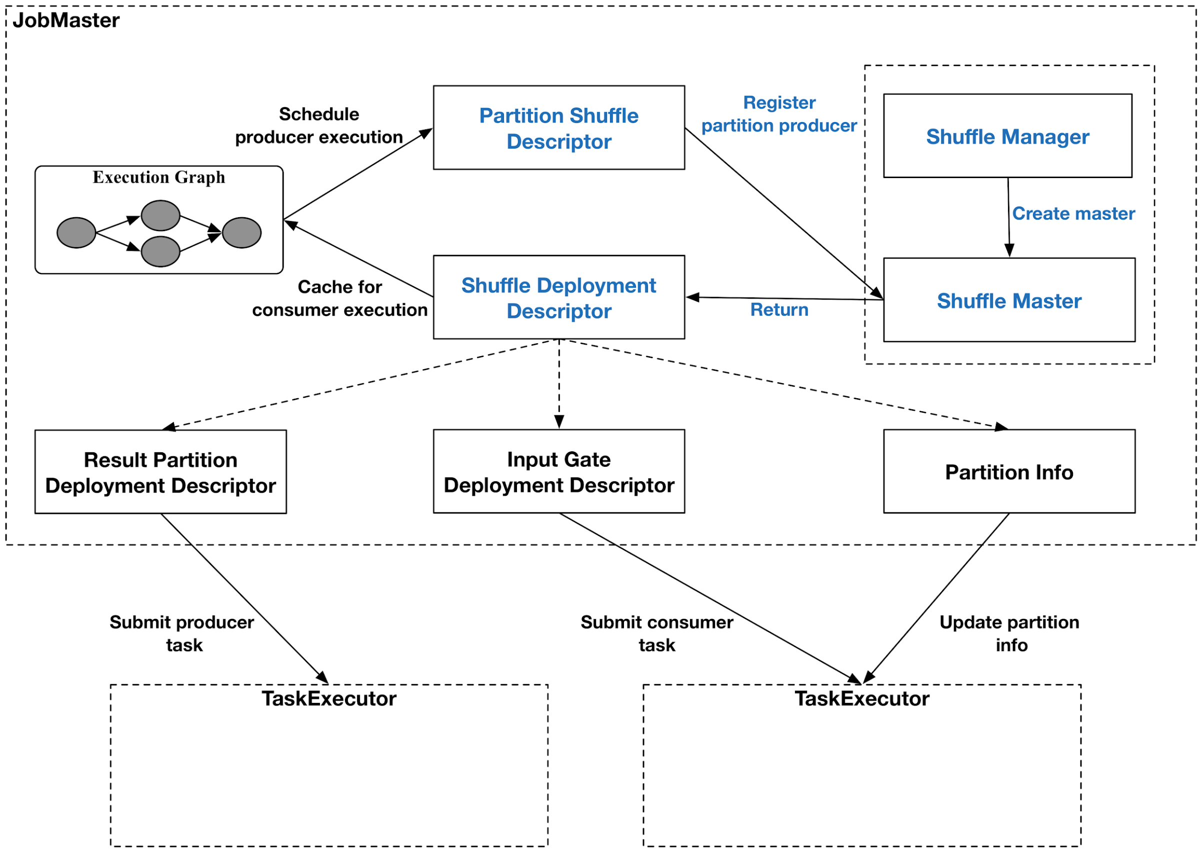
(2) Shuffle Master (JM side)

JM process creates ShuffleMaster from configured ShuffleManager per cluster, and is thus responsible for its lifecycle. ShuffleMaster is a global manager for partitions which means decoupling partition's lifecycle from task. So it could bring natural benefits for the following improvements.

- **TM release**: If TM is responsible for transporting partition data to consumer side, it could be released only when all internal tasks are in FINISHED state and all produced partitions are consumed. ShuffleMaster could provide the information whether the producer TM can be released before partition consumption is done.
- **Task failover**: If the consumer task fails or TM crashes, JM could ask ShuffleMaster whether producer's partition is still available. If partition is available for consuming, the producer task might not need be restarted which narrows down the failover region to reduce failover cost.
- **Partition cleanup**: When all the consumer tasks are in FINISHED state, the producer's partition is to deregister and cleanup with the ShuffleMaster. In case of external storage, partitions are at risk to linger after job/cluster failures. TTL mechanism is one option for handling this issue. ShuffleMaster could also provide an explicit way for manually triggering remove unused partitions.

In the first version, we only focus on migrating current existing process based on new ShuffleMaster architecture. So we define the most basic necessary methods below, and the above mentioned improvements might be forwarded step by step in priority by extending more features in ShuffleMaster.

```
public interface ShuffleMaster extends AutoClosable {
    ShuffleDeploymentDescriptor registerPartitionProducer(PartitionShuffleDescriptor psd);
    void deregisterPartitionProducer(PartionShuffleDescriptor psd);
}
```



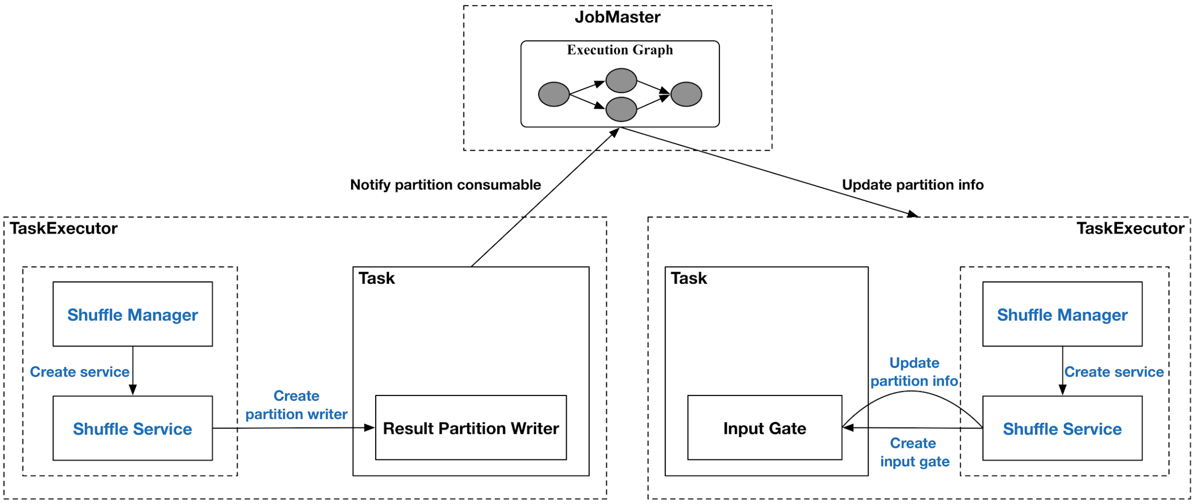
- **PartitionShuffleDescriptor** is introduced for wrapping all abstract information which JM can provide from job/execution graph, such as JobID, ExecutionAttemptID, TaskManagerLocation of producer, ResultPartitionType, ResultPartitionLocation etc. ResultPartitionType and ResultPartitionLocation are derived from graph and execution mode, so they are rather general parameters and do not belong to particular shuffle implementation.
- When producer execution is scheduled to deploy, PartitionShuffleDescriptor is created to register producer's partition with ShuffleMaster. ShuffleMaster transforms the abstract PartitionShuffleDescriptor into a specific **ShuffleDeploymentDescriptor** which would also be cached for consumer vertex if the consumer is not deployed yet.

- ShuffleDeploymentDescriptor is then put into ResultPartitionDeploymentDescriptor for submitting producer task and as a known producer inside InputGateDeploymentDescriptor for submitting consumer task. It can contain specific partition config for ShuffleService on TM side to serve partition writer and reader.
- Special **UnknownShuffleDeploymentDescriptor** could be used in InputGateDeploymentDescriptor if producer location is unknown during the deployment of consumer. JM can update it on consumer side by sending the specific ShuffleDeploymentDescriptor in partition infos when producer is deployed.

(3) Shuffle Service (TM side)

TM process creates ShuffleService from configured ShuffleManager per cluster, and is thus responsible for its lifecycle. Considering future ShuffleType config on job/edge level, TM could keep a registry of ShuffleService per ShuffleType.

```
public interface ShuffleService extends AutoClosable {
    ResultPartitionWriter createResultPartitionWriter(ResultPartitionDeploymentDescriptor rpdd);
    InputGate createInputGate(InputGateDeploymentDescriptor igdd);
    void updatePartitionInfos(Iterable<PartitionInfo> partitionInfos);
}
```



- ShuffleService is responsible for creating ResultPartitionWriter for producer task and creating InputGate for consumer task. Therefore this architecture can support extend matched writer and reader sides together. It might be useful for current ResultPartitionWriter/InputGate interfaces extending AutoClosable.
- Similar to how it is implemented currently, the scheduler/EG in JM can decide whether and when to update partition info on consumer side. E.g. always for pipelined partitions and when task is finished for blocking partitions. The producer task can also send the notification to JM when something has been produced in

pipelined partition, as now. The consumer's ShuffleService provides the way of updating internal input gate for known partition infos.
- ShuffleService should also consider the transport way between producer and consumer, e.g. via netty-based network as current default way. So ShuffleService might substitute NetworkEnvrionment in TaskManagerServices.

Future Improvement

Current ResultPartitionWriter and InputGate both operate on buffer unit with serialized record data. Certain ShuffleService implementation might benefit from operating on serialized record or even raw record directly (e.g. partial sort merge partition data).

- Abstract RecordWriter/RecordReader interfaces for handling serialized/raw record.
- ShuffleService could be further refactored to return RecordWriter/RecordReader.

New or Changed Public Interfaces

- In the first version, class name which implements ShuffleManager interface is configured for **shuffle.manager** parameter in Flink cluster level.
- In the second version, it might support job/edge level **ShuffleType** config for specific ShuffleManager implementation.

Migration Plan and Compatibility

- In the first version, the default ShuffleManager implementation is compatible with current existing behavior.
- In the second or later version, we can extend other implementations like YarnShuffleManager/KubernetesShuffleManager to be configured based on cluster environment and user requirements.

Rejected Alternatives

None so far.

Implementation Plan

All the mentioned related work could be done in at least two versions. The first version realizes the most basic architecture so that the following versions strictly build upon it.

First MVP: Refactoring to Shuffle API (preliminarily for Flink 1.8)

Introduce ShuffleMaster in JM (FLINK-11391)

- Implement *PartitionShuffleDescriptor* for covering necessary abstract info.
- Implement *ShuffleDeploymentDecriptor* generated from *PartitionShuffleDecriptor*.
- Define *ShuffleMaster* interface and create a simple implementation on JM side which relies on currently implemented *NetworkEnvironment* on TM side.
- Define *ShuffleManager* interface for creating *ShuffleMaster*.
- Introduce a Flink configuration option for *ShuffleManager* implementation. Default value for it could be <none> which serves as a feature flag at the moment to use current code paths.

Introduce ShuffleService in TM (FLINK-11392)

- Define *ShuffleService* interface and give a default implementation on TM side.
- Reuse shuffle related components from *NetworkEnvironment* in *ShuffleService*.
- Add *ShuffleService* factory method to *ShuffleManager* interface.
- Respect feature flag in Flink configuration option for *ShuffleManager*
- *notifyPipelinedConsumers* from outside of *ResultPartitionWriter* to make it not shuffle specific

Activate default shuffle implementation and remove legacy code (FLINK-11393)

Set shuffle implementation config parameter to default netty-based implementation from FLINK-11391 and FLINK-11392, instead of <none> which meant feature flag to use previous non-pluggable legacy implementation. The legacy and feature flag code should be removed.

Next steps

- Implement partition deregister and cleanup logic via *ShuffleMaster*.
- Improve TM release by checking partition consumed via *ShuffleMaster*.
- Improve task failover by checking producer's partition available via *ShuffleMaster*.
- Support job/edge level config for *ShuffleType*.
- Abstract *RecordWriter/Reader* interface for handing raw records.
- Refactor *ShuffleService* interface for returning *RecordWriter/Reader*.
- Adjust the processes in *StreamInputProcessor* and *StreamRecordWriter* based on Writer/Reader interfaces.
- Extend to *Yarn/KubernetesShuffleManager* implementations based on new interfaces.