FLIP-245: Source Supports Speculative Execution For Batch Job

<table>
<thead>
<tr>
<th>Discussion thread</th>
<th><a href="https://lists.apache.org/thread/zvc5no4yxwkt07xxpw1vo7j1p6h0lso">https://lists.apache.org/thread/zvc5no4yxwkt07xxpw1vo7j1p6h0lso</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Vote thread</td>
<td></td>
</tr>
<tr>
<td>JIRA</td>
<td><img src="FLINK-28397" alt="FLINK-28397 - [FLIP-245] Source Supports Speculative Execution For Batch Job" /></td>
</tr>
<tr>
<td>Release</td>
<td>1.16</td>
</tr>
</tbody>
</table>

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

- Motivation
- Public Interfaces
  - Supports Execution Attempt SourceEvent
  - Expose Execution Attempt ReaderInfo
- Proposed Changes
  - InputFormat sources support speculative execution
  - New sources support speculative execution
    - OperatorCoordinator
    - Improve SourceCoordinator and SourceCoordinatorContext
      - Managing SourceSplits
      - Handling Execution Attempt Operator Event
        - How to handle RequestSplitEvent
        - How to handle SourceEventWrapper
        - How to handle ReaderRegistrationEvent
        - How to handle ReportedWatermarkEvent
    - Reacting to Execution Attempt State Changes
- Whether to support concurrent execution attempts
- Fault-tolerant processing flow
- Limitations
  - No support for HybridSource
  - User defined source functions must not be affected by its speculative instances
- Compatibility, Deprecation, and Migration Plan
- Test Plan
- Rejected Alternatives

Motivation

This proposal is a follow up of FLIP-168: Speculative Execution For Batch Job. It aims to enable speculative execution to work with most sources, without changing the source connectors.

There are three types of sources: user defined source function, InputFormat source, new source introduced in FLIP-27. For user defined source functions, source speculative execution is supported without any additional work. So this proposal focused on how to support speculative Execution of the two other sources, namely, InputFormat source and new sources.

Public Interfaces

Supports Execution Attempt SourceEvent

SourceEvent is event passed between the SourceReaders and Enumerators. It is only required by some sources which have a custom event protocol between reader and enumerator. Among all Flink built-in connectors, only hybrid source would use SourceEvent.

If speculative execution of sources is enabled, one subtask/ExecutionVertex may have multiple execution attempts running at the same time. Therefore, the SplitEnumerator must be aware of the execution attempt that it receives/sends a SourceEvent. To support this, we introduce following interfaces:
/**
 * An decorative interface of {@link SplitEnumerator} which allows to handle {@link SourceEvent}
 * sent from a specific execution attempt.
 * <p>The split enumerator must implement this interface if it needs to deal with custom source
 * events and is used in cases that a subtask can have multiple concurrent execution attempts, e.g.
 * if speculative execution is enabled. Otherwise an error will be thrown when the split enumerator
 * receives a custom source event.
 */

@PublicEvolving
public interface SupportsHandleExecutionAttemptSourceEvent {
    /**
     * Handles a custom source event from the source reader. It is similar to {@link SplitEnumerator#handleSourceEvent(int, SourceEvent)} but is aware of the subtask execution attempt who sent this event.
     * @param subtaskId the subtask id of the source reader who sent the source event.
     * @param attemptNumber the attempt number of the source reader who sent the source event.
     * @param sourceEvent the source event from the source reader.
     */
    void handleSourceEvent(int subtaskId, int attemptNumber, SourceEvent sourceEvent);
}

Note: you don’t need to care these new interfaces if your sources does not use SourceEvent.

Expose Execution Attempt ReaderInfo

Currently, the SplitEnumerator gets ReaderInfo of subtasks from SplitEnumeratorContext. However, in speculative execution scenario, where one subtask can have multiple concurrent execution attempts, users may need to see the reader info of all the attempts, to know the reader’s locations to better make decisions for split assignments. To support this, we introduce interface method below:
SupportsHandleExecutionAttemptSourceEvent

```java
@Public
public interface SupportsHandleExecutionAttemptSourceEvent {
...
/**
 * Get the currently registered readers of all the subtask attempts. The mapping is from subtask id to a map
 * which maps an attempt to its reader info.
 * @return the currently registered readers.
 */
    default Map<Integer, Map<Integer, ReaderInfo>> registeredReadersOfAttempts() {
        throw new UnsupportedOperationException();
    }
...}
```

The existing `registeredReaders()` now returns the reader info of the earliest active attempt of each subtask.

## Proposed Changes

### InputFormat sources support speculative execution

Currently, after a source execution finishes to read assigned splits, it sends split requests to JobMaster. JobMaster calls `ExecutionVertex#getNextInputSplit` (host) to request a new split from `InputSplitAssigner`.

A Speculative ExecutionVertex can have multiple executions running at the same time. In order to support speculative execution of InputFormat source, we must ensure that all executions under an ExecutionVertex process the same InputSplits.

To achieve this goal, SpeculativeExecutionVertex stores all assigned splits into a list. When handling a source request, SpeculativeExecutionVertex first tries to fetch an unprocessed split from this list. If and only if all splits in the list have been consumed by this execution, Speculative ExecutionVertex would request a new InputSplit from InputSplitAssigner and store it into the list.

### New sources support speculative execution

In order to support speculative execution of new Sources introduced in FLIP-27, we need to ensure that all executions under an ExecutionVertex process the same InputSplits.

To achieve this goal, we need to do following work:

- Refactor `OperatorCoordinator` to be aware of different execution attempts of a subtask.
- Improve `SourceCoordinator` and `SourceCoordinatorContext` to correctly handle operator events, manage source splits and react to execution attempt state changes.
- Enable `SourceCoordinator` and `SourceCoordinatorContext` to know whether it should support multiple concurrent execution attempts (i.e. whether speculative execution is enabled).

### OperatorCoordinator

Before introducing speculative execution, the OperatorCoordinator did not need to know which execution sent the event when it handled an OperatorEvent from SourceOperator because an ExecutionVertex could only have one Execution running at the same time.

After introducing speculative execution, a speculative ExecutionVertex can have multiple current executions running at the same time. The OperatorCoordinator has to distinguish which execution sent the event. For example, the source coordinator needs to know which execution sends the split request. Therefore, we add an attemptNumber param to `handleEventFromOperator`.

The OperatorCoordinator also needs to be aware of the state of each attempt, to properly initialize or clean up things. Therefore, existing `subtaskFailed(…)` and `subtaskReady(…)` methods are refactored to be `executionAttemptFailed(…)` and `executionAttemptReady(…)`. 
Improve `SourceCoordinator` and `SourceCoordinatorContext`

**Managing SourceSplits**

Currently, when the source coordinator receives a split request from source operator, it requests a `SourceSplit` from `SplitEnumerator`. After the `splitEnumerator` creates a new split, `SplitEnumerator` stores the split assignment in `SourceCoordinatorContext`, `SourceCoordinatorContexts` send `AddSplitEvent` as response to `SourceOperator`.

To support speculative execution and ensure all executions of a subtask to process the same `SourceSplits`, we now cache the assigned splits in `SplitAssignmentTracker` and record whether the subtask has no more splits.

- When a new `SplitAssignment` is signaled by the `SplitEnumerator`, the new splits of a subtask will be sent to all its current registered readers.
- When a new reader is registered, all related cached splits will be send to the newly registered reader. If the subtask has no more splits, a `noMoreSplits` event will also be sent to the reader.

**Handling Execution Attempt Operator Event**

`SourceCoordinator` now handles operator event with awareness of the attempt which sends the event, below are the changes to make:

**How to handle RequestSplitEvent**

This logic can remain as is.

**How to handle SourceEventWrapper**

If the `SourceCoordinator` does not need to support concurrent execution attempts (i.e. speculative execution is not enabled), forward the event to `SplitEnumerator#handleSourceEvent(int, SourceEvent)`.

else if enumerator is a `SupportsHandleExecutionAttemptSourceEvent`, `SpeculativeSourceCoordinator` would call `SupportsHandleExecutionAttemptSourceEvent#handleSourceEvent(int, int, SourceEvent)`.

else it means the enumerator does not support handle `SourceEvent` in execution level, throws an exception.
How to handle ReaderRegistrationEvent

A ReaderInfo should be created for the attempt's reader and be registered to the SourceCoordinatorContext. If there are cached splits, the splits will be sent to the newly registered reader. If the subtask has no more splits, a noMoreSplits event will also be sent to the reader.

How to handle ReportedWatermarkEvent

ReportedWatermarkEvent is introduced in 1.15 which is used to support watermark alignment in streaming mode.

Speculative execution is only enabled in batch mode.

Therefore, SpeculativeSourceCoordinator would thrown an exception if receive a ReportedWatermarkEvent.

Reacting to Execution Attempt State Changes

When an execution attempt becomes ready, the SourceCoordinator will be notified and a SubtaskGateway will be setup for it in the SourceCoordinatorContext.

Note that this method will be invoked in two case:

- when an ExecutionVertex is newly created or reset, because its original attempt will be created at this time.
- when a speculative execution attempt is created. The SpeculativeScheduler is responsible to trigger this.

When an execution attempt fails, the SourceCoordinator will be notified and the corresponding SubtaskGateway will be removed.

Whether to support concurrent execution attempts

SourceCoordinatorContext needs to know whether concurrent execution attempts should be supported(i.e. whether speculative execution is enabled). So that itself and the SourceCoordinator can choose the proper logic to execute based on it.

SourceCoordinatorProvider is the provider of SourceCoordinator. It has to know whether concurrent execution attempts should be supported and pass this information to the SourceCoordinatorContext it creates.

```java
@Internal
public class SourceCoordinatorContext {
    ...
    boolean isConcurrentExecutionAttemptsSupported();
    ...
}
```

OperatorCoordinatorHolder also needs to be reworked to create a CoordinatorContext instance with isSpeculationEnabled flag.

SpeculativeExecutionJobVertex which is introduced in FLIP-168: Speculative Execution For Batch Job would create an OperatorCoordinatorHolder instance with true isSpeculationEnabled flag.

Fault-tolerant processing flow

After a source operator becomes ready to receive events, subtaskAttemptReady is called, SpeculativeSourceCoordinator would store the mapping of SubtaskGateway to execution attempt in SpeculativeSourceCoordinatorContext.

Then source operator registers the reader to the coordinator, SpeculativeSourceCoordinator would store the mapping of source reader to execution attempt in SpeculativeSourceCoordinatorContext.

If the execution goes through a failover, subtaskAttemptFailed is called, SpeculativeSourceCoordinator would clear information about this execution, including source readers and SubtaskGateway.

If all the current executions of the execution vertex are failed, subtaskReset would be called, SpeculativeSourceCoordinator would clear all information about this executions and adding splits back to the split enumerator of source.

The methods subtaskReady/subtaskFailed should be renamed to SubtaskAttemptReady/SubtaskAttemptFailed to perform Execution level while subtaskReset still perform ExecutionVertex level.

Limitations
No support for HybridSource

HybridSource is mainly used in streaming jobs. However, Speculative Execution only works with batch jobs. So it’s not a big problem to not support HybridSource.

User defined source functions must not be affected by its speculative instances

As mentioned in the limitation section in FLIP-168, user defined source functions must not be affected by its speculative instances, or vice versa. Otherwise, speculative execution should not be enabled.

Compatibility, Deprecation, and Migration Plan

Speculative execution of sources are enabled when speculative execution is enabled, which entails that Flink's default behavior won't change.

Test Plan

1. The changes will be covered by unit and IT cases.
2. Test the functionality in a real Standalone/Yarn/Kubernetes cluster.

Rejected Alternatives

No rejected alternatives yet.