

KYLIN-5436 Restart build job occasionally failed

Root Cause:

- A little background

When a build task is initiated, Kylin will have a thread. This thread is responsible for executing each step (i.e. 3 steps, resource probe, load data to index, update metadata), submitting the task to spark for execution (creating the spark-submit process), and updating the status of the step and the task as a whole.

- What happens when a task is restarted

When restarting a task, it first updates each step, stage, and job to ready state in a transaction, and then kills the task on yarn. When the transaction ends, it destroys the child processes associated with the task, and then sends another task ready event, so that the task can be scheduled immediately.

- What's the problem?

When restarting the task and destroying the child processes, the process exit code is not 0, i.e. it is not a successful exit. In this case, Kylin throws an exception

```
public CliCmdExecResult execute(String command, Logger logAppender, String jobId) throws ShellException {
    CliCmdExecResult r;
    if (remoteHost == null) {
        r = runNativeCommand(command, logAppender, jobId);
    } else {
        val remoteResult : Pair<Integer, String> = runRemoteCommand(command, logAppender);
        r = new CliCmdExecResult(remoteResult.getFirst(), remoteResult.getSecond(), processId: null);
    }

    if (r.getCode() != 0)
        throw new ShellException("OS command error exit with return code: " + r.getCode() //
            + ", error message: " + r.getCmd() + "The command is: \n" + command
            + (remoteHost == null ? "" : " (remoteHost:" + remoteHost + ")") //
        );

    return r;
}
```

The exception will be caught by the first thread that submitted the build task. At this point, it assumes that the build task has failed. So it updates the step it is currently executing to error,

and then the whole task is updated to error because there is a step that is now error, which means that the task was set to ready when it was restarted, but the original thread updated the task to error, thus causing the problem.

Question Supplement

```
protected void onExecuteFinished(ExecuteResult result) throws ExecuteException {
    logger.info("Execute finished {}", state:{}", this.getDisplayName(), result.state());
    MetricsGroup.hostTagCounterInc(MetricsName.JOB_STEP_ATTEMPTED, MetricsCategory.PROJECT, project, retry);
    if (result.succeed()) {
        wrapWithCheckQuit() → {
            updateJobOutput(project, getId(), ExecutableState.SUCCEEDED, result.getExtraInfo(), result.output(),
                hook: null);
        };
    } else if (result.skip()) {
        wrapWithCheckQuit() → {
            updateJobOutput(project, getId(), ExecutableState.SKIP, result.getExtraInfo(), result.output(), hook: null);
        };
    } else {
        MetricsGroup.hostTagCounterInc(MetricsName.JOB_FAILED_STEP_ATTEMPTED, MetricsCategory.PROJECT, project,
            retry);
        wrapWithCheckQuit() → {
            updateJobOutput(project, getId(), ExecutableState.ERROR, result.getExtraInfo(), result.getErrorMsg(),
                result.getShortErrMsg(), this::onExecuteErrorHook);
            killOtherPipelineApplicationOrUpdateOtherPipelineStepStatus();
        };
        throw new ExecuteException(result.getThrowable());
    }
}
```

From the code point of view, whenever a step ends (whether it succeeds or fails), the operation of updating the state will be wrapped by the method `org.apache.kylin.job.execution.AbstractExecutable#wrapWithCheckQuit`. One of its functions is to judge whether the task is in the three states of READY, PAUSED, and DISCARDED before updating. If it is, it proves that the task has been restarted, suspended, or canceled. In this case, it should not update its status, but directly throw an exception and exit.

```

boolean aborted = EnhancedUnitOfWork.doInTransactionWithCheckAndRetry() → {
    boolean abort = false;
    val parent : AbstractExecutable = getParent();
    ExecutableState state = parent.getStatus();
    switch (state) {
        case READY:
        case PAUSED:
        case DISCARDED:
            //if a job is restarted(all steps' status changed to READY) or paused or discarded, the old thread
            //in this case the old thread should fail itself by calling this
            if (applyChange) {
                logger.debug("abort {} because parent job is {}", getId(), state);
                updateJobOutput(project, getId(), state, info: null, output: null, hook: null);
            }
            abort = true;
            break;
        default:
            break;
    }
    return abort;
}, project, UnitOfWork.DEFAULT_MAX_RETRY, getEpochId(), getTempLockName());
if (aborted) {
    throw new JobStoppedNonVoluntarilyException();
}

```

According to the codes in these two pictures, the task should not be updated as an error, because the above check will be done before the update. The problem here is that when the restart is set to ready, the task is immediately scheduled and the status is changed to RUNNING. So if it is not blocked during check, it will still be updated to error.

- Phenomenon explanation

Phenomenon: When restarting the task, the first step, the second step and the whole task show failure

Explanation: The second step and the whole step become error status, see Where is the problem for an explanation of this point. The first step becomes an error because, before each step submits the task to spark, an operation of checking the parent task will be performed (if it is not RUNNING, an exception will be thrown). At this time, since the task has been set to error, this check will fail, so the first step will also show failure. eventually led to this phenomenon.

org.apache.kylin.job.execution.AbstractExecutable#checkParentJobStatus

Root Cause:

- 一点小背景

当发起一个构建任务时，Kylin会有一个线程。此线程负责执行各个step(即3个step，资源探测、加载数据到索引、更新元数据)，将任务提交到spark执行(创建spark-submit进程)，以及更新step和整个任务的状态。

- 重启任务时发生了什么

重启一个任务时，首先会在一个事务里将各个step、stage、job更新为ready状态，然后kill掉yarn上任务。当该事务结束时，会将该任务相关的子进程destory掉，然后再发一个任务ready的事件，让给任务马上调度起来。

- 问题出在哪里

重启任务，destory子进程时，进程退出码不是0即不是成功退出。此时，Kylin会抛出一个异常

```
public CliCmdExecResult execute(String command, Logger logAppender, String jobId) throws ShellException {
    CliCmdExecResult r;
    if (remoteHost == null) {
        r = runNativeCommand(command, logAppender, jobId);
    } else {
        val remoteResult : Pair<Integer, String> = runRemoteCommand(command, logAppender);
        r = new CliCmdExecResult(remoteResult.getFirst(), remoteResult.getSecond(), processId: null);
    }

    if (r.getCode() != 0)
        throw new ShellException("OS command error exit with return code: " + r.getCode() //
            + ", error message: " + r.getCmd() + "The command is: \n" + command
            + (remoteHost == null ? "" : " (remoteHost:" + remoteHost + ")") //
        );

    return r;
}
```

该异常会被第一个线程即提交构建任务的那个线程捕获到。此时，它会认为该构建任务执行失败了。所以它会更新当前它正在执行的step为error，然后因为有step为error了，所以整个任务就被更新为error。综上，也就是说，重启任务时把任务置为ready了，但是原来的线程又把任务更新为error，因此导致了这个问题。

- 问题补充


```

protected void onExecuteFinished(ExecuteResult result) throws ExecuteException {
    logger.info("Execute finished {}, state:{}", this.getDisplayName(), result.state());
    MetricsGroup.hostTagCounterInc(MetricsName.JOB_STEP_ATTEMPTED, MetricsCategory.PROJECT, project, retry);
    if (result.succeed()) {
        wrapWithCheckQuit(() -> {
            updateJobOutput(project, getId(), ExecutableState.SUCCEED, result.getExtraInfo(), result.output(),
                hook: null);
        });
    } else if (result.skip()) {
        wrapWithCheckQuit(() -> {
            updateJobOutput(project, getId(), ExecutableState.SKIP, result.getExtraInfo(), result.output(), hook: null);
        });
    } else {
        MetricsGroup.hostTagCounterInc(MetricsName.JOB_FAILED_STEP_ATTEMPTED, MetricsCategory.PROJECT, project,
            retry);
        wrapWithCheckQuit(() -> {
            updateJobOutput(project, getId(), ExecutableState.ERROR, result.getExtraInfo(), result.getErrorMsg(),
                result.getShortErrMsg(), this::onExecuteErrorHook);
            killOtherPipelineApplicationOrUpdateOtherPipelineStepStatus();
        });
        throw new ExecuteException(result.getThrowable());
    }
}

```

从代码上来看，每当一个step结束（无论成功还是失败），更新状态的操作，都会被 `org.apache.kylin.job.execution.AbstractExecutable#wrapWithCheckQuit` 这个方法包裹。它的作用之一就是在更新之前判断该任务是不是 `READY`、`PAUSED`、`DISCARDED` 这三个状态。如果是，证明该任务被重启、暂停、取消掉了，这样的话就不应该去更新它的状态，而是直接抛异常退出。

```

boolean aborted = EnhancedUnitOfWork.doInTransactionWithCheckAndRetry(() -> {
    boolean abort = false;
    val parent : AbstractExecutable = getParent();
    ExecutableState state = parent.getStatus();
    switch (state) {
        case READY:
        case PAUSED:
        case DISCARDED:
            //if a job is restarted(all steps' status changed to READY) or paused or discarded, the old thread
            //in this case the old thread should fail itself by calling this
            if (applyChange) {
                logger.debug("abort {} because parent job is {}", getId(), state);
                updateJobOutput(project, getId(), state, info: null, output: null, hook: null);
            }
            abort = true;
            break;
        default:
            break;
    }
    return abort;
}, project, UnitOfWork.DEFAULT_MAX_RETRY, getEpochId(), getTempLockName());
if (aborted) {
    throw new JobStoppedNonVoluntarilyException();
}

```

按照这两张图上的代码来说，任务不应该会被更新为error的，因为更新之前会做以上check。这里的问题出在，当重启置为ready后，任务立马被调度将状态改成了RUNNING。所以check的时候没有拦住，依然会更新为error。

- **现象解释**

现象：重启任务时，第一个步骤、第二个步骤以及整个任务显示失败

解释：第二个步骤以及整个步骤变成error状态，见 **问题出在哪里** 这一点的解释。第一步变成error则是因为，在每个步骤提交任务到spark上之前，会做一个检查父任务的操作（如果不是RUNNING则抛出异常）此时由于任务已经被置为error，所以这个check会失败，这样第一个步骤也会显示失败。最终导致了该现象。

org.apache.kylin.job.execution.AbstractExecutable#checkParentJobStatus