Asynchronous sandbox transfer within Condor

April 5, 2011

1 State of the art

The state of the art in Condor executes the following protocol:

- Schedd requests a claim from Startd
- Schedd activates claim
- if claim activation is accepted by Startd, Schedd will start to send jobs
- if claim activation is denied, claim will be relinquished; new match needs to be found
- In case claim activation is accepted, Schedd will start a Shadow for the job
- on the execute node, a Starter starts up to run the job. In case the job requires input data to be transferred, the Shadow will transfer the input sandbox to the Starter. Thereafter, the job will be run.
- Once the job terminates, the output sandbox is sent to the output destination – traditionally this used to be the Shadow, nowadays besides the Shadow it can be a URL, Globus.org, ...
- After the transfer finished, the job is officially marked as "TERMINATED" and the claim becomes available to potentially send another job. Execute directories are cleaned up by the Starter.
- On startup, Startd cleans up all contents in local execute directory to wipe out old job data.

The downside of the current protocol is that while the output sandbox is in transfer (and this sandbox can contain many GB of data), the claimed CPU slot is sitting idle, waiting for the next job to arrive. The goal of this design proposal is achieving a certain overlap of I/O and CPU work by decoupling the sandbox transfer from the actual job execution.

1.1 Work delegation

Once a claim is activated, work delegation follows the resource state. New work is delegated once the Shadow receives the final job update from the Starter (after job executed and output sandbox was transferred) and forwards this information to the Schedd. The Schedd will then delegate the next job for this claim to the old resource, which will run the job if in `CLAIMED/IDLE` state. This state transition happens after the job transfer completed.

In the current scenario, the Schedd manages only one Shadow per claim and therefore all claim information will be cleaned up upon reaping of this Shadow process.

2 New design

The claim request and -activation processes remain essentially unchanged. Changes will include:

1. Introduction of a sandbox manager object maintaining job sandboxes (done by Fermi)
   - Sandbox object contains the following information: unique sandbox identifier, physical disk location of sandbox directory, creation time, lease time
   - Sandbox manager object maintains a list of sandboxes associated with claim ID.
   - There is no persistence beyond Startd lifetime; even a failing Starter requires reinitialization (see section 4).
2. Sandbox Manager object is maintained by Startd
3. Sandbox Manager object can be responsible for directory deletion (instead of Starter) and technically it should have ownership. However, backwards compatibility requires to leave the ownership to the Starter anyway, therefore that's where directory deletion will continue to happen. The sandbox manager will call a cleanup function when a claim is released, but in most cases this will be a no-op, since the Starter already took care of it.
4. Upon claim request, Schedd creates a unique sandbox ID and puts this into the job ad. This ID is constructed from the public claim ID and another unique (randomly generated) string. This allows us to query the sandbox by claim, given that a claim can have two sandboxes at a given time (if output sandbox transfer takes longer than sending the next job in claim). The random string is generated with the same algorithm as used to generate the `transfer_key` in the file transfer object.
5. Claim activation leads to Startd accepting/rejecting the claim.

6. In the first milestones a sandbox should always be available because only data of the previous job of the same claim should still be residing on the execute node

7. The pending output transfer will be reflected in the advertised Disk attribute of the execution slot.

8. Starter starts up, input data transfer takes place, job executes, output data gets created

9. After job execution, Starter communicates job exit to Shadow and Startd

10. Starter informs Startd that CPU slot is now CLAIMED/IDLE

11. Starter tells Shadow to put job into new job state "`TRANSFERRING_OUTPUT`". In the first milestone this will not involve updating the Schedd. Therefore, the slot will indeed sit claimed/idle during the transfer. However, this update will be done in the second milestone during this update step.

12. Both updates are ultimately needed in order to update the Startd's CPU slot (when would that happen otherwise?), in order to change the job's status, and in the end to be able to inform the Schedd about the idle claim. The Startd update needs to happen first or the Schedd may try to send a job to a slot that is officially not set to claimed/idle yet.

13. Starter then continues running and transfers data to the Shadow

14. Meanwhile Startd is available to run a new job from the same claim

15. Sandbox Manager object is updated to delete information on previous sandbox once transfer succeeded. Information is also discarded in case the claim is disturbed (prematurely relinquished).

16. Two concurrent job output transfers per claim are allowed. In theory more concurrent transfers are possible, but at the moment we are capping at two, since more thought needs to be put into how to manage/throttle multiple transfers. This implies that an execution slot could be claimed idle while two output transfers are taking place.

17. A job's output sandbox is deleted upon successful transfer or - if that never happens - upon expiration of the claim. If all deletion fails or if execute machine fails unexpectedly, Startd deletes sandbox contents upon next startup.

18. Shadow reuse is limited – while a job is still transferring output, a new Shadow needs to be started for the next job in the claim.

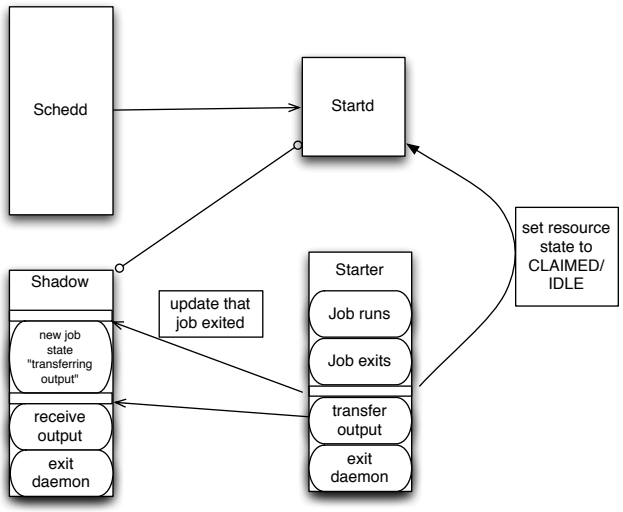

Figure 1: Daemon control flow of new design

2.1 Job suspension

If the job is running, the new approach does not change the behavior. If the job is transferring output, suspending the job should be a no-op, since transferring data creates little system load.

2.2 Job removal

If a job is removed, behavior does not significantly change. The Starter will delete the sandbox directory and the sandbox entry will be removed from the sandbox manager.

2.3 Job eviction

In case of eviction, the job's sandbox may be transferred to the submit spool, if so requested. Therafter the sandbox information is removed from the sandbox manager.

2.4 Work delegation

Once a claim is activated, work delegation follows the resource state. New work is delegated once the Starter sends the update to the Shadow that the job is done executing and should be transitioned into `TRANSFERRING_OUTPUT` state. This will update the Schedd which can now delegate the next job.

Given that more than one job can be delegated at a given time, the Schedd needs to maintain a counter of currently running Shadows per claim. This allows the Schedd to determine when the maximum number of concurrent jobs per claim (currently two, but this allows for easy extension) is reached. Only if the last Shadow in the claim gets reaped, the final cleanup can take place. This requires an additional intermediate reaper which leaves all the claim information intact for still running Shadows.

3 Assumptions

- Shadow notices job exiting and is able to put job into new state
- Starter behavior mostly unchanged
- Multiple Starters can run at once (jobs of the same claim)

4 Behavior in failure scenarios

- If the Starter dies, the job will have to be rerun entirely. With the current design it is not possible to just initiate the file transfer, even if that is the only missing or failing step.
- If the Shadow restarts and reconnects while job already is in `TRANSFERRING_OUTPUT` state, just the file transfer will be done; the job does not need to be re-run, under the assumption that the Shadow can reconnect (which requires Starter to still be around) and lease is not expired.
- If the submit machine is unreachable during the job state transition, the Starter will wait with file transfer until the job state updated. The Starter will be killed (and sandbox therefore deleted) if the lease expires. Sandbox transfer error handling will remain unchanged to state of the art implementation.

5 Limitations

- We currently do not have a way to manage actual job disk usage within Condor. The design of the sandbox manager does not provide an approach for that either. In order to recognize whether a job fails because of a filled-up disk requires monitoring of the disk state. However, we can only detect post-mortem why a job failed. Changing this "job" failure to map to a "resource allocation" failure may be beyond the scope of this work unit (but please correct me if I am wrong). As things stand, we match the job's space requirements to the slot's advertised disk space, and will correct the advertised disk space by the size of the already present output sandbox for the previous job in this claim (if still in transfer state).

- The approach will have the same performance as the current implementation if one job's output sandbox is very large or network bandwidth is very limited such that the sandbox transfer takes longer than running a second job and transferring its output sandbox.
- We will start more Shadows in this approach – it is not possible to handle two jobs in the same Shadow at once.
- On the execute side, more Starters can remain in a waiting state until they receive the update that the job entered `TRANSFERRING_OUTPUT` state.