ASYNCHRONOUS FILE TRANSFER IN CONDOR

Parag Mhashilkar (Fermi National Accelerator Laboratory)
Cathrin Weiss (Condor Project, University of Wisconsin at Madison)
Overview

- CEDPS Project
- Motivation: Asynchronous sandbox management
- Asynchronous sandbox management
- Sandbox Management & Sandbox Manager
- Sandbox Transfer in Condor
  - Current Protocol
  - Asynchronous sandbox transfer
- Prototype & Its Performance
- Ongoing & Future Work
- Acknowledgements
CEDPS Project

CEDPS: The five year project started in 2006, funded by Department of Energy (DOE)

Goals
- Produce technical innovations for rapid and dependable data placement within a distributed high performance environment and for the construction of scalable science services for data and computing from many clients.
- Address performance and functionality troubleshooting of these and other related distributed activities.

Collaborative Research
- Mathematics & Computer Science Division, Argonne National Laboratory
- Computing Division, Fermi National Accelerator Laboratory
- Lawrence Berkeley National Laboratory
- Dept of Computer Science, University of Wisconsin Madison
- Information Sciences Institute, University of Southern California
CEDPS Activities at FNAL

- Investigating data movement mechanisms for data stage-out on Grids
  - globusonline.org needs integration with SRM interface for OSG

- Supporting the integration of data movement mechanisms with scientific DH frameworks
  - Supporting the integration of globusonline.org with Dark Energy Survey (DES) data handling system

- Integration of asynchronous data stage-out mechanisms in overlay workload management systems (…this talk…)
  - Release resources at job termination. Delegate data stage-out to external agents.
  - Integrate support for globusonline.org into glideinWMS
glideinWMS

- Pilot-based WMS that creates on demand a dynamically-sized overlay condor batch system on Grid resources to address the complex needs of VOs in running application workflows

- Components
  - WMS Collector
  - Glidein Factory
  - User Pool Collector
  - User Scheduler
  - VO Frontend

- Factory knows about the sites and how to submit glideins to the sites
- VO frontend knows about the user job details
- WMS Collector acts as a dashboard for Factory - VO Frontend communication.
- Provides an end-to-end solution for several VOs in OSG
Motivation

 Typical life cycle for a job on a worker node
  • Transfer input sandbox
  • Run computational phase and produce output
  • Transfer output sandbox
 We will be focusing on output sandbox transfer during this talk.
Motivation [contd.]

- Sandbox transfer failures & inefficiencies have less impact on throughput for -
  - Short running jobs
    - Less CPU wasted
  - Jobs with small output sandbox

- But …
  - They have different set of problems (not covered here) like -
    - Higher overhead before the jobs starts running
    - Black-hole effect
Motivation [contd.]

- Seen by glideinWMS users, failures & inefficiencies have significant impact on throughput for –
  - Jobs with large input sandbox
    - CPU is not utilized when a sandbox is transferred in
  - Long running jobs
    - Output sandbox transfer can fail wasting CPU consumed by the job
  - Jobs with large output sandbox
    - CPU is not utilized when a sandbox is transferred back
    - Wasted resources in case of intermittent failures during the transfer

- Can we do better?
Asynchronous Sandbox Management

- Enhance glideinWMS by
  - Increasing the CPU utilization of Condor-managed resources in a wide area environment through CPU and network I/O overlap enabled by asynchronous transfers of sandboxes – Miron Livny

- What does this mean?
  - Pipeline the transfer of asynchronous sandboxes in Condor using globusonline.org or other transfer protocols
  - Multiple transfers can take place via transfer slots
  - New job can start running if the previous job has entered stage-out state
  - Can reattempt failed transfers as needed
  - Support multiple transfer protocols using transfer Plug-ins
First Prototype: Condor Hooks

- Support at the Application Layer (glideinWMS).
- Use Condor Hooks
  - To identify end of CPU stage
  - Initiate the output sandbox transfer
- Start another condor_startd to accept new job
Asynchronous Sandbox Management in Condor

- Rather than the application, let condor transfer the output sandboxes asynchronously
  - Generic
  - Robust
  - Reliable
  - Scalable
Sandbox Manager

- Sandbox Manager module extends the sandbox management functionality of the Condor.

Collaborative Effort

- **Fermi team** is implementing the Sandbox Manager
  - A repository for sandboxes
- **Condor team** working on output sandbox management in the Condor
  - Internal protocol changes
  - Details to follow …
Sandbox Manager

- Condor_startd interfaces with the sandbox manager to keep track of output sandboxes
- Sandbox Manager manages the sandbox objects
  - Register sandboxes
  - Update sandboxes
  - Transfer sandboxes
  - Unregister sandboxes
  - […]

Condor Week 2011 (05/04/2011)
What is necessary to integrate this into Condor?
Current sandbox transfer

New Claim
Current sandbox transfer

New Claim

Schedd

Startd
Current sandbox transfer

New Claim

Schedd \(\rightarrow\) requestClaim \(\rightarrow\) Startd
Current sandbox transfer

New Claim

Schedd \[\text{requestClaim}\] Startd

Claimed
Current sandbox transfer

New Claim

Schedd

requestClaim

Startd

State: Idle

Claimed
Current sandbox transfer

New Claim

Schedd \rightarrow requestClaim \rightarrow Startd

State: Idle
Claimed
Current sandbox transfer

New Claim

Schedd \rightarrow \text{requestClaim} \rightarrow \text{State:Idle}

fork

Shadow

Startd

Claimed
Current sandbox transfer

New Claim

Schedd

requestClaim

activateClaim

Startd

State: Idle

Claimed

Shadow

fork
Current sandbox transfer

New Claim

Schedd \[\rightarrow\] requestClaim \[\rightarrow\] Startd

fork

Shadow

State: Busy

Claimed
Current sandbox transfer

New Claim

Schedd

requestClaim

Startd
State: Busy
Claimed

Shadow
fork

fork

Condor Week 2011 (05/04/2011)
Current sandbox transfer

New Claim

Schedd

requestClaim

Startd

Claimed

State: Busy

fork

fork

Shadow

Starter
Current sandbox transfer

New Claim

Schedd

requestClaim

Startd

Claimed

State: Busy

fork

transferInputSandbox

fork

fork

fork

Shadow

Starter
Current sandbox transfer

New Claim

- Schedd
- Shadow
- Startd
- Starter

Schedd → requestClaim → Startd
Shadow → transferInputSandbox → Starter
Startd → State:Busy
Starter → runJob
Current sandbox transfer

New Claim

Schedd

requestClaim

Startd
Claimed

State: Busy

Shadow

transferInputSandbox

Starter
Current sandbox transfer

New Claim

Schedd

Startd
Claimed

State: Busy

requestClaim

Shadow

Starter

fork

transferInputSandbox

transferOutputSandbox

fork
Current sandbox transfer

New Claim

- Schedd
  - requestClaim
  - fork
- Shadow
- Startd
  - State: Busy
  - Claimed
Current sandbox transfer

New Claim

**Schedd**

**Startd**
Claimed

**Shadow**

---

requestClaim

fork
Current sandbox transfer

New Claim

Schedd

requestClaim

Startd

Claimed

State:Idle

Shadow

fork
Current sandbox transfer

Existing Claim

- Schedd
- Startd
- Shadow

State: Idle
Claimed
Current sandbox transfer

Existing Claim

Schedd

Shadow

Startd

State: Idle

Claimed
Current sandbox transfer

Existing Claim

- Schedd
- Shadow
- Startd

State: Idle
Claimed

activateClaim
sendJob
Current sandbox transfer

Existing Claim

Schedd

Shadow

sendJob

Startd

Claimed

State: Busy
Current sandbox transfer

Existing Claim

Schedd

Shadow

sendJob

Startd

Claimed

State: Busy

fork
Current sandbox transfer

Existing Claim

Schedd

sendJob

Shadow

State: Busy

Startd

Claimed

fork

Starter
Current sandbox transfer

Existing Claim

- Schedd
- Shadow
- Starter
- Startd

Transfers:
- `sendJob` from Schedd to Shadow
- `transferInputSandbox` from Shadow to Starter
- `fork` from Startd to Starter

State: Busy
Current sandbox transfer

Existing Claim

Schedd

sendJob

Shadow

transferInputSandbox

Starter

State:Busy

fork

Startd

Claimed

runJob
Current sandbox transfer

Existing Claim

Schedd

Shadow

SendJob

transferInputSandbox

Starter

Startd

Claimed

State: Busy

fork
Current sandbox transfer

Existing Claim

Schedd

sendJob

Shadow

transferInputSandbox

transferOutputSandbox

Startd

State: Busy

Claimed

Starter

fork

Condor Week 2011 (05/04/2011)
Asynchronous sandbox transfer
Asynchronous sandbox transfer
Asynchronous sandbox transfer

Schedd

Startd

Sandbox Manager
Asynchronous sandbox transfer

Schedd

requestClaim

Startd

Sandbox Manager
Asynchronous sandbox transfer

Schedd \[\xrightarrow{\text{requestClaim}}\] Startd

Claimed

Sandbox Manager
Asynchronous sandbox transfer

Schedd \rightarrow \text{requestClaim} \rightarrow \text{Startd Claimed}

State: Idle

Sandbox Manager
Asynchronous sandbox transfer

Schedd → requestClaim → Startd → State:Idle

Schedd fork

Startd Claimed

Sandbox Manager
Asynchronous sandbox transfer

Schedd ➔ requestClaim ➔ Startd

Schedd ➔ fork ➔ Shadow

State: Idle
Claimed

Sandbox Manager
Asynchronous sandbox transfer

Schedd

requestClaim

Startd
Claimed

activateClaim

fork

Shadow

State: Idle

Sandbox Manager
Asynchronous sandbox transfer

- Schedd
  - requestClaim
  - Shadow
  - fork

- Startd
  - State: Busy
  - Claimed

- Sandbox Manager
Asynchronous sandbox transfer

Schedd \(\xrightarrow{\text{fork}}\) Shadow \(\xrightarrow{\text{requestClaim}}\) Startd \(\xrightarrow{\text{registerSandbox}}\) Sandbox Manager

State: Busy

Claimed
Asynchronous sandbox transfer

Schedd

requestClaim

Startd

Claimed

registerSandbox

Sandbox Manager

fork

fork

Schedd

fork

Startd

fork

Sandbox Manager
Asynchronous sandbox transfer

- **Schedd**
  - `fork`
  - `requestClaim`

- **Shadow**
  - `fork`

- **Startd**
  - `Claimed`
  - `registerSandbox`

- **Sandbox Manager**
Asynchronous sandbox transfer

Schedd -> requestClaim -> Startd -> registerSandbox -> Sandbox Manager

Schedd -> fork -> Shadow

Shadow -> transferInputSandbox -> Starter

State: Busy

Schedd

Startd

Sandbox Manager

Claimed

Shadow

Starter
Asynchronous sandbox transfer

- **Schedd** requests claim from **Startd**
  - **Startd** claims sandbox
  - **Startd** registers sandbox with **Sandbox Manager**
  - **Startd** runs job

- **Shadow** forks
  - **TransferInputSandbox** to **Starter**
  - **Starter** runs job

States:
- **State: Busy**
Asynchronous sandbox transfer

Schedd \rightarrow \text{requestClaim} \rightarrow \text{Startd} \rightarrow \text{registerSandbox} \rightarrow \text{Sandbox Manager}

\text{Shadow} \rightarrow \text{fork} \rightarrow \text{transferInputSandbox} \rightarrow \text{Starter} \rightarrow \text{fork} \rightarrow \text{Claimed} \rightarrow \text{State:Busy}
Asynchronous sandbox transfer

- **Schedd**
  - requestClaim
  - fork

- **Shadow**
  - transferInputSandbox

- **Startd**
  - Claimed
  - registerSandbox
  - fork

- **Sandbox Manager**
  - State:Idle
Asynchronous sandbox transfer

- **Schedd**
  - Fork
  - requestClaim to **Startd**

- **Startd**
  - State: Idle
  - registerSandbox to **Sandbox Manager**
  - Claimed

- **Sandbox Manager**

- **Shadow**
  - Fork
  - transferInputSandbox to **Starter**

- **Starter**
  - Fork
  - transferOutputSandbox to **Shadow**

Condor Week 2011 (05/04/2011)
Asynchronous sandbox transfer

Schedd -> requestClaim -> Startd

State: Idle

Startd -> registerSandbox -> Sandbox Manager

Shadow -> fork -> Starter

fork

Starter

transferInputSandbox

transferOutputSandbox

State: Idle

Schedd

Startd

Starter

Schedd

Startd

Starter

Schedd

Startd

Starter
Asynchronous sandbox transfer

- Schedd
- Shadow
- Starter
- Startd
- Sandbox Manager

- Schedd 
  - requestClaim 
  - fork

- Shadow
  - fork
  - transferInputSandbox
  - transferOutputSandbox

- Starter
  - fork
  - registerSandbox
Asynchronous sandbox transfer

Schedd → Startd: requestClaim → registerSandbox
Startd: Claimed
Startd → Shadow: fork
Shadow: State:Busy
Shadow → Starter: fork
Starter: State:Busy
Starter → Shadow: fork
Shadow: State:Busy
Shadow → Schedd: fork
Schedd: State:Busy

Transfers:
- transferInputSandbox
- transferOutputSandbox
Asynchronous sandbox transfer

- Schedd
- Shadow
- Startd
- Starter
- Sandbox Manager

States and Actions:
- Schedd: requestClaim
- Startd: registerSandbox
- Starter: State: Busy
- Shadow: fork
- Transfer Input Sandbox
- Transfer Output Sandbox
Asynchronous sandbox transfer

Schedd → requestClaim → Startd

Startd Claimed → registerSandbox → Sandbox Manager

Shadow → fork → Shadow

Shadow → fork → Starter

Starter → fork → Sandbox Manager

Sandbox Manager → fork → Starter

Starter → fork → Shadow

Shadow → fork → Shadow

State: Busy

transferInputSandbox

transferOutputSandbox
Asynchronous sandbox transfer

- **Schedd**
  - requestClaim
  - fork

- **Shadow**
  - fork

- **Starter**
  - fork
  - transferInputSandbox
  - transferOutputSandbox

- **Startd**
  - registerSandbox
  - Claimed
  - fork

- **Sandbox Manager**

State: Busy

condor week 2011 (05/04/2011)
Asynchronous sandbox transfer

- Schedd
- Startd
- Sandbox Manager
- Shadow
- Starter

**Request Claim**: Schedd requests Claim from Shadow, which in turn requests Claim from Starter.

**Claimed**: Startd is claimed by Starter.

**Register Sandbox**: Startd registers the sandbox with Sandbox Manager.

**Transfer Input Sandbox**: Starter transfers the input sandbox to Shadow.

**Transfer Output Sandbox**: Starter transfers the output sandbox to Shadow.

**Run Job**: Starter runs the job.
Asynchronous sandbox transfer

- **Schedd**
  - requestClaim
  - fork

- **Shadow**
  - transferInputSandbox

- **Startd**
  - Claimed
  - registerSandbox
  - fork

- **Starter**
  - runJob

- **Sandbox Manager**
  - State: Busy
Asynchronous sandbox transfer

Schedd \rightarrow requestClaim \rightarrow Startd

Schedd \rightarrow fork \rightarrow Shadow

Shadow \rightarrow transferInputSandbox \rightarrow Starter

Startd \rightarrow registerSandbox \rightarrow Sandbox Manager

Startd \rightarrow Claimed

Sandbox Manager \rightarrow unregisterSandbox

Startd \rightarrow runJob

Starter \rightarrow State: Busy
Asynchronous sandbox transfer
Changes in Condor: Requirements
Asynchronous sandbox transfer

Changes in Condor: Requirements

- Job needs to change state once execution is completed ("transferring_output")
Asynchronous sandbox transfer

Changes in Condor: Requirements

- Job needs to change state once execution is completed ("transferring_output")
- `Startd` needs to be able to accept a new job while previous job(s) are still transferring (if in same claim)
Asynchronous sandbox transfer

Changes in Condor: Requirements

- Job needs to change state once execution is completed ("transferring_output")
- \textit{Startd} needs to be able to accept a new job while previous job(s) are still transferring (if in same claim)
- \textit{Schedd} needs to be able to forward next job in claim while previous job is in transfer mode
Asynchronous sandbox transfer
Changes in Condor: Requirements (2)
Asynchronous sandbox transfer

Changes in Condor: Requirements (2)

- Failure tolerance: System needs to deal with failure on submit and execute side
Asynchronous sandbox transfer

Changes in Condor: Requirements (2)

- Failure tolerance: System needs to deal with failure on submit and execute side
- File transfer failures need to be detected and accordingly handled: Do we rerun the whole job? How often do we retry? How long do we keep the output data around?
Asynchronous sandbox transfer

Changes in Condor: Requirements (2)

- Failure tolerance: System needs to deal with failure on submit and execute side
- File transfer failures need to be detected and accordingly handled: Do we rerun the whole job? How often do we retry? How long do we keep the output data around?
- Another open question: How many file transfers should run concurrently?
Asynchronous sandbox transfer

Changes in Condor: where are we at?
Asynchronous sandbox transfer

Changes in Condor: where are we at?

- Job state is changed to `transferring_output` once execution completed
Asynchronous sandbox transfer

Changes in Condor: where are we at?

- Job state is changed to transferring_output once execution completed
- `Startd` changes state to `claimed/idle` once job execution completed and is therefore able to accept next job in claim
Asynchronous sandbox transfer

Changes in Condor: where are we at?

- Job state is changed to `transferring_output` once execution completed.
- `Startd` changes state to `claimed/idle` once job execution completed and is therefore able to accept next job in claim.
- `Startd` maintains a sandbox manager object.
Asynchronous sandbox transfer
Prototype assumptions
Asynchronous sandbox transfer

Prototype assumptions

- No failures
Asynchronous sandbox transfer

Prototype assumptions

- No failures
- First job in claim longer in transfer mode than all the following ones
Asynchronous sandbox transfer

Prototype assumptions

- No failures
- First job in claim longer in transfer mode than all the following ones

Job 1: execute, transferOutput

Job 2: execute, transferOutput

Job 3: execute, transferOutput

Job 4: execute, transferOutput
Asynchronous sandbox transfer
Prototype limitations
Asynchronous sandbox transfer

Prototype limitations

- Not production-ready yet: *Schedd* does only know about one job per claim, others run “hidden”
Asynchronous sandbox transfer

Prototype limitations

- Not production-ready yet: *Schedd* does only know about one job per claim, others run “hidden”
- Therefore all jobs in the claim are killed once the “known” job terminates (claim is deactivated)
Asynchronous sandbox transfer

Prototype limitations

- Not production-ready yet: Schedd does only know about one job per claim, others run “hidden”
- Therefore all jobs in the claim are killed once the “known” job terminates (claim is deactivated)
- Failure on either side leads to rerun (“hidden” jobs cannot reconnect to respective Starters)
Asynchronous sandbox transfer
Prototype performance results
Asynchronous sandbox transfer

Prototype performance results

- Setup: Series of $n$ identical jobs, execution time $t$, output transfer time $u$, total time $v = t+u$
Asynchronous sandbox transfer

Prototype performance results

- Setup: Series of $n$ identical jobs, execution time $t$, output transfer time $u$, total time $v = t + u$
- First job in series has a very long running output transfer time to cover rest of series (see limitations)
Asynchronous sandbox transfer
Prototype performance results

- Setup: Series of $n$ identical jobs, execution time $t$, output transfer time $u$, total time $v = t + u$
- First job in series has a very long running output transfer time to cover rest of series (see limitations)
- No input data transfer
Asynchronous sandbox transfer

Prototype performance results

- Setup: Series of $n$ identical jobs, execution time $t$, output transfer time $u$, total time $v = t + u$
- First job in series has a very long running output transfer time to cover rest of series (see limitations)
- No input data transfer
- Short running job, producing lots of output data ($u = 2t$)
Asynchronous sandbox transfer
Prototype performance results
Asynchronous sandbox transfer

Prototype performance results

Minutes to completion

Number of jobs in claim

sync transfer
async transfer
Asynchronous sandbox transfer

What needs to be done

- Needed: Concept of multiple activations per claim
- *Schedd* must maintain a list of current activations per claim (currently there is only one activation possible)
- *Startd* needs to be able to map a particular activation to a particular *Starter*
Asynchronous sandbox transfer

When will we be able to use it?
Asynchronous sandbox transfer

When will we be able to use it?

- Soon!
Asynchronous sandbox transfer

When will we be able to use it?

- Soon!
- Feature for 7.7.x development series!
Asynchronous sandbox transfer

What else?
Asynchronous sandbox transfer

What else?

- As usual: We like to make our users happy.
Asynchronous sandbox transfer

What else?

- As usual: We like to make our users happy.
- Tell us what you like about it and how you would like to use it.
Acknowledgments

- The work is being done as a part of the CEDPS project
- This work is an ongoing collaboration between the Computing Division, Fermilab and the Condor team at University of Wisconsin-Madison
- The GlideinWMS project was developed and is supported by Fermilab Computing Division and the CMS Experiment
- The Condor project is developed and supported by Condor Team at University of Wisconsin at Madison
- Fermilab is operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy.
- The Open Science Grid (OSG) collaborates with CEDPS for solutions on data movement and stage-out