Using Stork

An Introduction

Condor Week 2006

Jeff Weber, Condor Project
Computer Sciences Department
University of Wisconsin-Madison
weber@cs.wisc.edu
http://www.cs.wisc.edu/condor
Audience

• Users already familiar with Condor, DAGMan, who need advanced data placement capabilities.

• This tutorial makes an excellent extension to this morning's Condor User's tutorial.
Tutorial Outline

• Classical data transfer problems
• Stork solutions
• Stork data placement job management
• Managing mixed data, CPU job work flows with DAGMan
Meet Friedrich*:

He is a scientist. But he has a big problem.

*Frieda's twin brother
I have a LOT of data to process.

Where can I get help?
Friedrich's problem ...

• Many large data sets to process. For each data set:
  • stage in data from remote server
  • run CPU data processing job
  • stage out data to remote server
"Classic" Data Transfer Job

#!/bin/sh
globus-url-copy source dest

Scripts often works fine for short, simple data transfers, but...
Many things can go wrong!

* Errors are more likely with large data sets:
  * “The network was down.”
  * “The data server was unavailable.”
  * “The transferred data was corrupt.”
  * “My workflow didn’t know the data was bad.”
Enter Stork:

• Creates notion of a *data placement* job: managed, scheduled just like a Condor CPU job.

• Friedrich will benefit from:
  • Built-in fault tolerance
  • Compatible with Condor DAGMan workflow manager

http://www.cs.wisc.edu/condor
Supported Data Transfers

- local file system
- GridFTP
- FTP
- HTTP
- SRB
- NeST
- SRM
- modular extensible to other protocols
Fault Tolerance

• Retries failed jobs
• Can also retry failed job using alternate protocol, e.g. “first try GridFTP, then try FTP”
• Retry “stuck” jobs
• Configurable fault responses
Getting Stork

- Stork is bundled with Condor 6.7, and all future releases
- Available as a free download from http://www.cs.wisc.edu/condor
- Currently available for Linux platforms
Friedrich Installs a “Personal Stork” on his workstation...

• What do we mean by a “Personal” Stork?
  • Condor/Stork on your own workstation, no root access required, no system administrator intervention needed

• After installation, Friedrich submits his jobs to his Personal Stork…
Friedrich's Personal Stork

Friedrich's workstation:

- Master
- Central Mgr.
- SchedD
- StartD

Stork

data jobs

DAG

CPU jobs

external data servers

N compute elements

http://www.cs.wisc.edu/condor
Your Personal Stork will ... 

• Keep an eye on your data and CPU jobs, and will keep you posted on their progress
• Throttle maximum jobs running
• Keep a log of your job activities
• Add fault tolerance to your jobs
  • Detect and retry failed data placement jobs
• Enforce data placement, CPU job order dependencies
Creating a Submit Description File

• A plain ASCII text file

• Neither Stork nor Condor care about file extensions, nor statement order.

• Tells Stork about your job:
  • data placement type, source/destination location/protocol, proxy location, input, output, error and log files to use, command-line arguments, etc.
Simple Submit File

// c++ style comment lines
[
    dap_type = "transfer";
    src_url = "gsiftp://server/path";
    dest_url = "file:///dir/file";
    x509proxy = "default";
    log = "stage-in.out.log";
    output = "stage-in.out.out";
    err = "stage-in.out.err";
]

Note: different format from Condor submit files
Running `stork_submit`

- You give `stork_submit` the name of the submit file you have created:
  
  ```
  # stork_submit stage-in.stork
  ```

- `stork_submit` parses the submit file, checks for it errors, and sends job to Stork server.

- `stork_submit` returns the new job id: the job “handle”
Sample stork_submit

# stork_submit stage-in.stork
using default proxy: /tmp/x509up_u19100

Sending request:

```
[  
    dest_url = "file://dir/file";
    src_url = "gsiftp://server/path";
    err = "path/stage-in.out.err";
    output = "path/stage-in.out.out";
    dap_type = "transfer";
    log = "path/stage-in.out.log";
    x509proxy = "default"
]
```

Request assigned id: 1  returned job id

http://www.cs.wisc.edu/condor
The Job Queue

- **stork_submit** sends your job to the Stork server:
  - Manages the local job queue
- View the queue with **stork_q**, or **stork_status** ...
Getting Job Status

• `stork_q` queries all active jobs
  
  # stork_q

• `stork_status` queries the named job id, which may be active, or complete
  
  # stork_status 12
Removing jobs

• If you want to remove a job from the job queue, you use `stork_rm`

• You can only remove jobs that you own (you can’t run `stork_rm` on someone else’s jobs unless you are root)

• You must give a specific job ID:
  
  `# stork_rm 21`  · Removes a single job
More information about jobs

• Controlled by submit file log setting
• Stork creates a log file (user log)
  • “The Life Story of a Job”
  • Shows all events in the life of a job
• *Always* have a log file
• To turn it on in submit file:
  log = “filename”;
Sample Stork User Log

000 (001.-01.-01) 04/17 19:30:00 Job submitted from host: <128.105.121...

001 (001.-01.-01) 04/17 19:30:01 Job executing on host: <128.105.121.5:...

008 (001.-01.-01) 04/17 19:30:01 job type: transfer

008 (001.-01.-01) 04/17 19:30:01 src_url: gsiftp://server/path

008 (001.-01.-01) 04/17 19:30:01 dest_url: file:///dir/file

005 (001.-01.-01) 04/17 19:30:02 Job terminated.

(1) Normal termination (return value 0)

Usr 0 00:00:00, Sys 0 00:00:00 - Run Remote Usage
Usr 0 00:00:00, Sys 0 00:00:00 - Run Local Usage
Usr 0 00:00:00, Sys 0 00:00:00 - Total Remote Usage
Usr 0 00:00:00, Sys 0 00:00:00 - Total Local Usage

0 - Run Bytes Sent By Job
0 - Run Bytes Received By Job
0 - Total Bytes Sent By Job
0 - Total Bytes Received By Job

...
My jobs have have dependencies...

Can Stork help solve my dependency* problems?

* Not your personal problems!
Friedrich learns DAGMan

- Directed Acyclic Graph Manager
- DAGMan allows you to specify the dependencies between your jobs, so it can manage them automatically for you.
- (e.g., “Don’t run job “B” until job “A” has completed successfully.”)
What is a DAG?

• A DAG is the data structure used by DAGMan to represent these dependencies.
• Each job is a “node” in the DAG.
• Each node can have any number of “parent” or “children” nodes - as long as there are no loops!
Defining Friedrich's DAG

• A DAG is defined by a text file, listing each of its nodes and their dependencies:
  # data-process.dag
  Data IN in.stork
  Job CRUNCH crunch.condor
  Data OUT out.stork
  Parent IN Child CRUNCH
  Parent CRUNCH Child OUT

• each node will run the Condor or Stork job specified by accompanying submit file
Submitting a DAG

• To start your DAG, just run `condor_submit_dag` with your dag file, and Condor will start a personal DAGMan daemon to begin running your jobs:
  
  ```
  # condor_submit_dag friedrich.dag
  ```

• `condor_submit_dag` submits a Scheduler Universe Job with DAGMan as the executable.

• Thus the DAGMan daemon itself runs as a Condor job, so you don’t have to baby-sit it.
In Review

With Stork Friedrich now can...

• Submit his data processing jobs and go home!
• Stork manages the data transfers, including fault detection and retries
• Condor DAGMan manages his job dependencies.
Additional Resources

• http://www.cs.wisc.edu/condor/stork/

• Condor Manual, Stork sections

• stork-announce@cs.wisc.edu list

• stork-discuss@cs.wisc.edu list
Questions?
Additional Slides
Important Parameters

- `STORK_MAX_NUM_JOBS` limits the number of active jobs.
- `STORK_MAX_RETRY` limits the number of job attempts before it is marked as failed.
- `STORK_MAXDELAY_INMINUTES` specifies the "hung job" threshold.
Current Restrictions

• Currently, best suited for “Personal Stork” mode

• Local file paths must be valid on Stork server, including submit directory.

• To share data, successive jobs in DAG must use shared filesystem
Future Work

• Enhance multi-user fair share
• Enhance support for DAGs without shared filesystem
• Enhance scheduling with configurable job requirements, rank
• Add job matchmaking
• Additional platform ports

http://www.cs.wisc.edu/condor
Access to Data in Condor

• Use Shared Filesystem if available

• No shared filesystem?
  • Condor can transfer files
    • Can automatically send back changed files
    • Atomic transfer of multiple files
    • Can be encrypted over the wire

• Remote I/O Socket

• Standard Universe can use remote system calls (more on this later)
Condor File Transfer

- ShouldTransferFiles = YES
  - Always transfer files to execution site
- ShouldTransferFiles = NO
  - Rely on a shared filesystem
- ShouldTransferFiles = IF_NEEDED
  - Will automatically transfer the files if the submit and execute machine are not in the same FileSystemDomain

```
Universe = vanilla
Executable = my_job
Log = my_job.log
ShouldTransferFiles = IF_NEEDED
Transfer_input_files = dataset$(Process), common.data
Transfer_output_files = TheAnswer.dat
Queue 600
```

http://www.cs.wisc.edu/condor
condor_master

• Starts up all other Condor daemons, including Stork
• If there are any problems and a daemon exits, it restarts the daemon and sends email to the administrator
• Acts as the server for many Condor remote administration commands:
  • condor_reconfig, condor_restart, condor_off, condor_on, condor_config_val, etc.
Running a DAG

- **DAGMan** acts as a “meta-scheduler”, managing the submission of your jobs to Condor based on the DAG dependencies.

![Diagram showing DAGMan, Condor Job Queue, and .dag File]

[http://www.cs.wisc.edu/condor]
Running a DAG (cont'd)

- DAGMan holds & submits jobs to the Condor queue at the appropriate times.
Running a DAG (cont’d)

• In case of a job failure, DAGMan continues until it can no longer make progress, and then creates a "rescue" file with the current state of the DAG.
Recovering a DAG

- Once the failed job is ready to be re-run, the rescue file can be used to restore the prior state of the DAG.
Recovering a DAG (cont’d)

• Once that job completes, DAGMan will continue the DAG as if the failure never happened.
Finishing a DAG

• Once the DAG is complete, the DAGMan job itself is finished, and exits.