Submit Machine Management

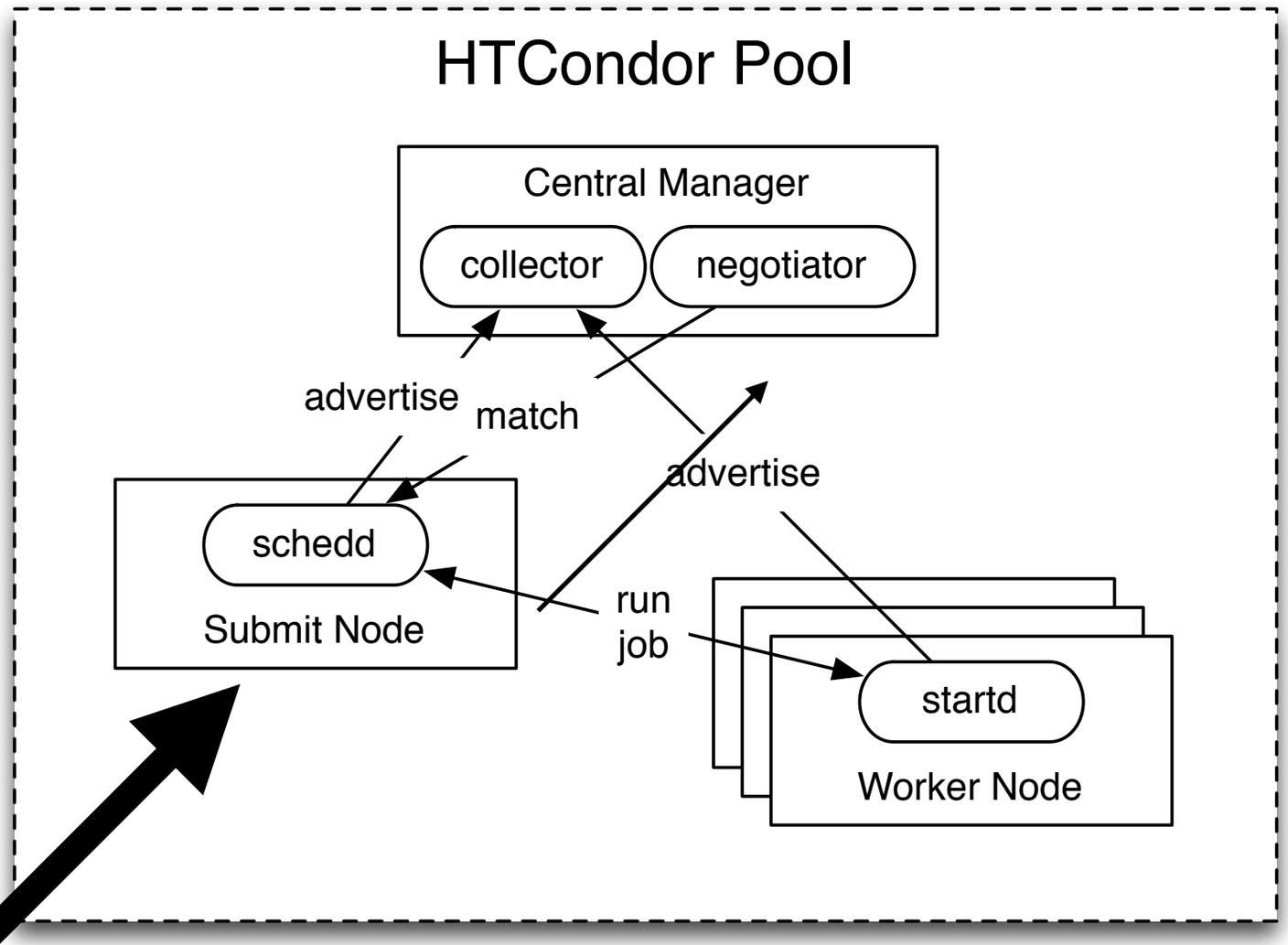
Brian Bockelman

European HTCondor Site Admins Meeting - 2014

So you want to build the ultimate submit machine?

- While you can get HTCondor to run on your toaster in a weekend, providing a high-quality scalable submit service can take significant planning and effort.
    - In this talk, we'll walk through the process of putting together the service, noting special requirements for scalability and customization hooks.
        - I focus on the *non-obvious* parts of this task; this is *not* "how to build your first submit machine".
- Roughly, three portions:
    - Spec'ing out the service.
    - Installing and Configure HTCondor.
    - Customizing user environments.

Roadmap - Where Are We?



YOU ARE HERE

Spec'ing out the Service - Setting Expectations

- Before we even get to hardware, you need to work with users to understand what kind of service is needed:
  - **Job Scale**:
    - What is the maximum number of jobs this schedd will need to run? The average?
    - How many jobs are expected to be in queue?
  - **Job Rates**: What is the expected job start and stop rates? What does the distribution look like?
  - **IO requirements**: What, if anything, do you know about your per-job input and output transfer requirements?
- In general, it's really hard to determine what the distributions look like. HTCondor keeps only rough statistics itself. I prefer to do the *highly scientific* "multiply everything by two" to determine peak scale.

Spec'ing out the Service - Hardware Considerations

- Next, I outline the hardware considerations from most important to least.
- **IO**: The schedd is a single-threaded daemon which blocks on disk IO and frequently calls fsync() on its job database.
    - Therefore, your overall scalability is limited by the latency of your storage system.
- To maintain a stable service of >10k running jobs, you will want to keep the spool directory on an SSD.
- A typical setup has:
    - A dedicated, small, low-latency storage target for spool, AND
    - A large (TBs), high-throughput storage target for user home/working directories.

Spec'ing out the Service - Hardware Considerations 2

- **Memory**: As a rule of thumb, plan on 1MB RAM per running job and 50KB per idle job.
    - Recent work in 8.3.x reduces this to 300-400KB per running job. I still prefer the above number to include a bit of a safety factor.
- **CPU**: The schedd has no CPU-bound component (the process is single-threaded anyway).
    - Base your CPU decisions on the needs of the logged-in users (i.e., compiling or running test jobs).
- **Network connectivity**: Unless you are aware of specific needs from your user base, 1Gbps is sufficient.

To shared filesystem or not?

- How do you move files between the submit and execute machines?
    - **With a shared file system**: These can be expensive and finicky, but users often love the simplicity. They don't need to know what files they use.
        - It's often difficult to carefully control usage of the shared file system - life can be chaotic!
    - **With HTCondor file transfer**: Forces users to *think* and express their file requirements inside the job.
        - Requires more work from the user - ***however***, it typically results in a more "IO friendly" job. No user hammering AFS!
        - HTCondor can throttle new transfers (future: not match machines) if the schedd is spending too much time on IO. Shared file systems typically have no concept of queueing and performance degrades massively!
        - When using file transfers, it is simpler to run jobs offsite.

OS Tweaks (for schedds with >10k jobs)

- Memory overcommit: In /etc/sysctl.conf, **sys.vm.overcommit_memory=1**
- Max socket backlog: In /etc/sysctl.conf, **net.core.somaxconn=1024**
- Max file descriptors: Set **sys.fs.file-max** to be greater than 500k (already is on most OSes!)
- Max per-process file descriptors: Set **nofile** in /etc/security/limits.d.
    - Not done commonly (see scaling talk).
- Maximum number of processes: Set **nprocs** in /etc/security/limits.d
    - Only for hosts which do lots of DAGMan / local universe.
- Beware of iptables **conntrack** module: Consider blacklisting the conntrack module if you need many TCP connections (see scaling talk).

Host Firewalls and Networking

- **DNS**: DNS is a mixed bag! HTCondor can work fine with- or without DNS; in fact, DNS failures (or slow name resolution) often cause problems for submit services. Recommendations:
    - Go all-in or all-out: don't try to mix use of IP addresses in some cases and DNS in others.
    - It is the *host name*. There should be one per host; if you use DNS, the hostname should match the public DNS name for simplicity. If you need a more complex setup, the **NETWORK_HOSTNAME** config option overrides the hostname detection logic.
        - Consider your cluster's dynamics: if there's a small number (<50) of nodes and they won't come in and out of the cluster frequently, you may not need DNS.
- The worker nodes, central manager, and schedd need to be able to contact each other via the network.
- I *highly* recommend setting **USE_SHARED_PORT=true** (in fact, the plan is to make this the future default) throughout your pool. This will allow all HTCondor daemons to use the same inbound port, TCP 9618.
- HTCondor has the ability to rewrite addresses (for TCP port-forwarding setups) and intelligently manage multiple private and public networks. While this means HTCondor can work with very adverse networking conditions, *think twice before using; they can be extremely difficult to debug*.

Host Firewalls and Networking

- With shared port enabled, the firewall configuration becomes:
    - **Inbound connections**: TCP 9618 from client hosts, the central manager, and worker nodes.
    - **Outbound connections**: Outbound connections are necessary to the central manager and worker nodes.
        - HTCondor phone home: By default, the HTCondor daemons report simple usage statistics to UW via UDP. This is a requirement from the funding agencies; consider leaving this on if you wish continued support of the software. For more, see http://research.cs.wisc.edu/htcondor/privacy.html.
        - By default, UDP updates are sent to the central manager; if desired, switch them to TCP using **UPDATE_COLLECTOR_WITH_TCP=true**. All other outgoing communication uses TCP.
- The CCB allows the worker nodes to be behind a separate stateful firewall or NAT (i.e., no inbound connectivity from the schedd). This is not typically used in site setups.

Installing and Configuring

- **Basics**:
  - Always install via RPM; I strongly discourage use of tarballs.
  - Always maintain your configurations with configuration management software such as Puppet or Chef.
  - *Never* edit condor_config or condor_config.local. ***Always*** use the config.d directory.

Logging Considerations

- Consider enabling the AuditLog; this contains a concise log of who used the schedd, what they did, and how they authenticated.
    - Essential for security incidents!
- Explicitly determine your log retention policy; default is 10MB x 2 files per log.
    - Most large sites will want to retain more. I use 100MB x 10 files.
- Set the logfile name to SYSLOG to forward a HTCondor log to **/dev/log**. Useful for sites that have an existing centralized log management scheme and/or strict retention policies.
    - In particular, sites should consider forwarding the AuditLog to syslog.

Monitoring - Host

- Host-level monitoring and alerting is critical, especially if users have a login to the submit host.
  - This is not HTCondor-specific; apply the security protections you believe needed for a generic login host.
  - Users are quicker than your alert system; typically, monitoring is best for post-crash telemetry.

Monitoring - HTCondor

- All HTCondor daemons export 5-20 critical metrics in their ClassAds.

- Recently, HTCondor delivered native integration with Ganglia. This allows you to turn the above metrics into time series.

  - When combined with host metrics (CPU usage, memory, network activity), these are a powerful mechanism for debugging problems.

  - If your site doesn't use Ganglia for monitoring, the daemon can integrate with your system by invoking a "gmetric" compatible command-line utility.

Accounting

- While condor_history is great, the logs *do* rotate eventually.
  - Don’t wait until your boss asks about accounting usage to discover this fact!
- If you set **PER_JOB_HISTORY_DIR**, then the schedd records the job ClassAd into a unique file when it leaves the queue.
  - Accounting can be done by reading each of these files and uploading to a DB.
  - Alternately, the **PER_JOB_HISTORY_DIR** captures the job execution *instances* on the remote startds. Further, this can be queried centrally (if you have admin privileges).

Configuration Knobs to investigate

- SYSTEM_PERIODIC_REMOVE / SYSTEM_PERIODIC_HOLD: Expression to either remove or hold “malformed” jobs.
  - Check out SYSTEM_PERIODIC_XXX_REASON too!
- MAX_JOBS_RUNNING / MAX_JOBS_SUBMITTED: Limit the number of jobs running / submitted to prevent users from pushing the schedd into swap.
- FILE_TRANSFER_DISK_LOAD_THROTTLE: If you are using HTCondor transfer mechanisms, this limits the amount of disk load HTCondor places on the system (suggestion: set to N for a host with N spinning disks).
- MAX_TRANSFER_{INPUT,OUTPUT}_MB: Avoid transferring excessive amounts of data per job.

Setting up the User Environment

- How does a user submit a job? It's a bit of a religious argument.
    - **School of thought #1**: Make users learn condor_submit. There's tons of documentation "on the internet", allows users to fully unlock the power of condor_submit, and is no-maintenance.
    - **School of thought #1.1**: Write a small wrapper around condor_submit to "helpfully" fix obvious errors in files or set a few site-specific defaults.
        - Alternately, can control some defaults from the user environment. I.e., add the following to **/etc/profile.d/condor.sh**:
            - export _CONDOR_AccountingGroup=\"local.`id -gn`.`id -un`\"
        - Periodically check schedd-side to see if a user is trying to game the system.
- **School of thought #2**: Any condor_* command is too damn hard to use. Replace it with a simpler site-specific interface and train them to use this.
    - *Alternately*, use **condor_qsub** because you like PBS-style scripts better!
    - *Note*: wrapper scripts require the users to play along. Do not be surprised to find they bypass your script when python bindings are used.
- **School of through #2.1**: Any command line is too hard for users; they only access the system through a webapp.

User Environments - Automating attribute settings

- **Easy**: Utilize SUBMIT_ATTRS. Add to the config file:

```
JobIsGrid = true
SUBMIT_ATTRS = $(SUBMIT_ATTRS), JobIsGrid
```

- **Medium**: Use MODIFY_REQUEST_EXPR_* to modify a user's request_* *at the startd*.

- **Medium**: Use SCHEDD_ROUND_ATTR_ to round up arbitrary attributes *at the schedd*.

- **Medium-hard**: Write a wrapper around your submit script.

- **Hard**: Use JobRouter to enforce policy schedd-side.

Tweaks

- Ideas that make user's life better:
    - Use the custom condor_q / condor_status print formats for your site.
    - Take advantage of **~/.condor/user_config** (user-specific config file, like ~/.bashrc); for example, you can created this file on first login with a PAM module to lock the user to a specific schedd.
    - Customize MOTD to tell the user a summary of their jobs on login.

User education and training

- A little bit of user education goes a long way!
    - While we have dozens of “circuit breakers” in HTCondor to prevent more common mistakes, it helps if the user doesn’t make them in the first place.
- A handful of topics to make your life easier (beyond the “standard intro”):
    - How to avoid invoking condor_q?
    - How long to wait for a job to start / what to do when a job is idle?
    - What’s an “excessive” number of jobs in the queue?

User Education - Userlog files

- HTCondor users love to write the following code to submit or monitor jobs:

```
while true
  if [ `condor_q bbockelm -run | wc -l` -lt 100 ]; then
    condor_submit some_file
  fi
  sleep 1
done
```

- This is unnecessarily wasteful of schedd resources; if enough users do the same thing, the schedd may become unresponsive.
  - Instead, take advantage of the user logs which are typically available locally and record the job lifetime.
  - Users don't even need to parse them - utilize **condor_wait** instead!

Questions?