

Scaling Up, Scaling Out

Brian Bockelman

European HTCondor Site Admins Workshop

Growth is Good!

- No matter where you start with HTCondor, it's likely you'll someday outgrow your initial deploy. When that happens, you have two (complementary) strategies
 - **Scale up:** Get more from your existing deployment architecture.
 - **Scale out:** Spread the service across additional system images.

Up versus Out

- Anecdotally, it appears *scaling out* is more popular than “up” these days.
 - “Up” requires special hardware configurations (SSD, large amounts of RAM), which may be difficult for your fabric management team to acquire and support.
 - However, “out” places a larger burden on the service operations team. Complicated, distributed setups are harder to document, monitor, debug, and understand.
- Realistically, a large service will require **both** approaches. Multiple hosts are good in that they force you to invest in monitoring, config management, and automation. Beefy hosts are good because some aspects of the effort go up linearly with the number of hosts.
- *Do not* conflate “scaling out” and redundancy. You need redundancy and disaster plans regardless of how you plan to scale.

You want the latest features

- “Developer series” is a bit of a misnomer - it’s really the “feature series”.
- If you want to scale to the leading edge (a pool with more than 20k cores), you’ll really want to consider the developer series. Currently, 8.3.x has the following improvements:
 - Collector uses completely non-blocking IO.
 - Negotiator performs bulk queries against the schedd, reducing total negotiation time.
 - (8.3.2) Significant reduction in startd<->schedd communication for running jobs.
 - Significant reduction in shadow size (more in store for 8.3.2).
- In general, if you can identify an issue - blocking IO, redundant communication, inefficient protocols - causing you heartache, it will be **improved** in the developer series. Only bugfixes go to stable series.
 - **Don’t be that site** which waits a year to try out the scalability improvement, only to find the developers fixed the wrong problem.

Scale Up - Schedd

- How can you get more from your existing schedd? Let's examine the most likely bottlenecks:
 - **I/O**: The schedd's need to fsync() its database causes it to stop responding while disk IO is outstanding. Symptoms include high IO wait in the monitoring and inability to create a TCP connection to the daemon (by default, Linux will reject new connection attempts if 128 are already waiting in the accept() queue).
 - To alleviate, consider the following steps:
 - If in a virtualized environment, work with your cloud provider to identify any possible contention for the physical drive .
 - Place the \$SPOOL directory on a separate partition. This reduces the amount of data the OS will flush to disk.
 - Utilize a SSD. This was essential for our scale tests where we hit 100,000 running jobs in a schedd.
 - **ANECDOTE**: I have seen thousands of dollars of sysadmin effort spent trying to save a few hundred dollars of hardware (SSD).

Scale Up - Schedd

- **Memory:** The 40KB / idle job and 1MB / running job memory requirements can lead to large, expensive schedd hosts. Workarounds that don't include "buy more RAM":
 - Upgrade to 8.3.2 (ETA December 23, 2014). In the 8.3.x series, we have removed a number of libraries from the shadow. Total savings is about 400KB.
 - Switching to the specially-compiled binary "condor_shadow_s" saves about 10% (set **SHADOW=\$(SBIN)/condor_shadow_s**); installing the 32-bit version of the binary gets another 10% savings.
 - Or, wait until 8.3.2; we will try to automate this process in the RPM packaging.

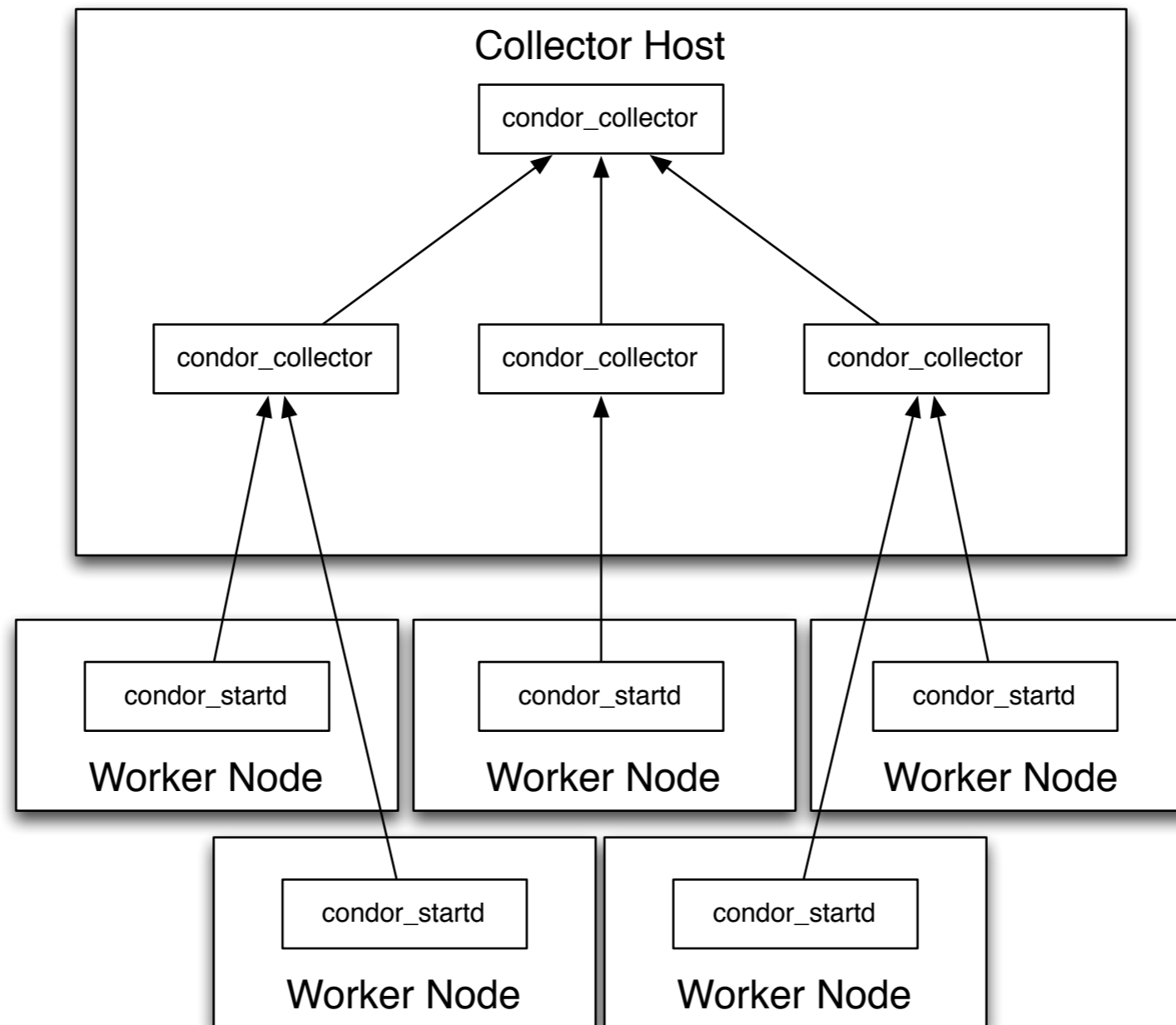
Schedd limits

- OSG scale tests have shown that, given sufficient hardware, up to 100k running jobs can be sustained by a single host.
- This level is difficult to reach if users have a direct schedd login (compiling, polluting the page cache, using disk IO, etc) or there is other heavyweight software on the host.
 - To maximize scale, consider only allowing remote submission.
 - Without SSDs, 20k running jobs may be a reasonable limit.

Scale Up - Collector

- The collector keeps no on-disk state; I/O is not relevant.
- Provision about 250KB / slot (this includes some padding) of RAM.
- The collector needs no special considerations for CPU.
- Sites with more than 20k machines may consider setting up a collector hierarchy:
 - Start up (N+ 1) collectors on the central host.
 - Each startd picks one of the N collectors at random.
 - Then, the N “child” collectors then forward ads to the parent collector via UDP. This reduces the number of TCP connections to any one collector and number of security sessions to the parent.
 - This is documented at <https://htcondor-wiki.cs.wisc.edu/index.cgi/wiki?p=HowToConfigCollectors>. If you are interested in trying this out, let us know! Improvements in the 8.3.x series may make the whole setup unnecessary; we’re interested in seeing scale test results.

Tiered Collector



Common Pitfalls - Networking

- Count your TCP connections closely!
 - If daemons update the collector with TCP, they will maintain a TCP connection *per daemon* (startd, schedd, master - but not starter/shadow).
 - If you are using CCB to bypass a NAT or firewall (relatively uncommon for a *site*), there's another connection *per daemon*.
 - Each shadow on the submit host maintains a TCP connection to the starter.
- A busy submit node can have tens of thousands of TCP connections. Given enough scale:
 - You can hit the default TCP connection limit (64K) for Linux's iptables stateful firewall.
 - You can hit the *hypervisor's* firewall limit (**esp at CERN**).
 - You can hit the limit of an on-site firewall hardware appliance (if you have an uncommon network topology).

Common Pitfalls - OS limitations

- The following OS limits may bite you for large deploys:
 - **Linux**: increase the maximum number of PIDs to be greater than the number of shadows.
 - **RHEL only**; not CentOS / SL6: Increase the maximum file descriptors to at least .5M. CentOS/SL6 defaults are fine.
 - **EL >6**: If your users run many DAGman / local universe jobs, you may need to increase the maximum number of user process >1024.
 - ~~Per-process file descriptor limits: Increase for the collector if updates go through TCP. Per-process FD limits are now a HTCondor config variable; set MAX_FILE_DESCRIPTOR.~~
 - Needed for collectors in pools with >5k startds that update via TCP.
 - Max accept backlog: Increase from 128 to 1024. May no longer be necessary starting in 8.3.2.
- As with per-process FD limits and accept backlog above, we try to eliminate the need for tweaking the OS by improving code, moving the configuration into HTCondor, or through packaging.
 - The intent is the list on this page gets shorter year-over-year, even for the largest sites!

Scale Out - Schedd

- Your pool can have an arbitrary number of schedds (although inefficiencies may arise with more than 30).
 - Adding more submit hosts is the most common way to achieve scale.
- Two common approaches:
 - **Static assignment:** Partition the workflows by manually assign tasks or users to a particular schedd.
 - **Manure spreader:** Have another submit layer randomly assign or load-balance jobs across multiple schedds.

The Manure Spreader - CMS version

- The following python snippet is how CMS selects a schedd for its CRAB3 application:

```
coll = htcondor.collector()
schedds = coll.xquery(htcondor.AdTypes.Schedd, 'StartSchedulerUniverse =?= true', \
    ['Name', 'DetectedMemory'])
choices = [(schedd["Name"], schedd["DetectedMemory"]) for schedd in schedds if \
    ("Name" in schedd) and ("DetectedMemory" in schedd)]
schedd_name = weighted_choice(choices)
```

- Once a schedd name is selected, CRAB3 submits a user's task there.
 - We balance by task, not by job.
 - CRAB3 remembers which schedd the task was sent to; when querying for task status, it only queries that schedd.

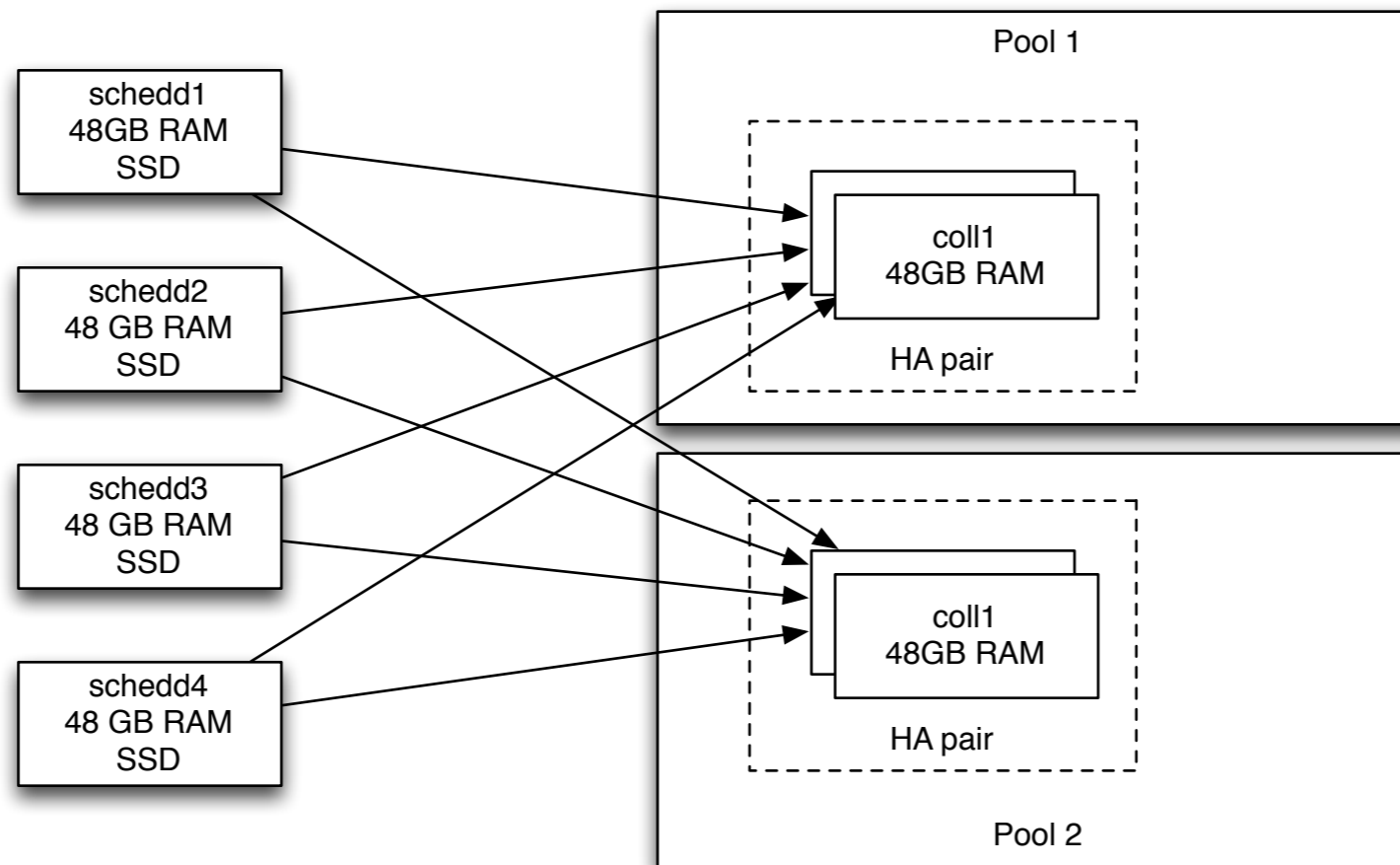
Static assignment

- For login hosts: when a user first logs in, generate a **~/condor/user_config** file, adding a well-known SCHEDD_NAME.
 - This schedd is the only one used by default for user submits and queries.
 - Rebalancing becomes an infrequent / exceptional event.
 - This becomes more feasible if the schedds are sufficiently provisioned to handle uneven load balancing.
- Dedicated (or, rather, predictable) production workflows are locked to their own schedd.
 - Think: Tier-0 for a particular experiment at CERN.

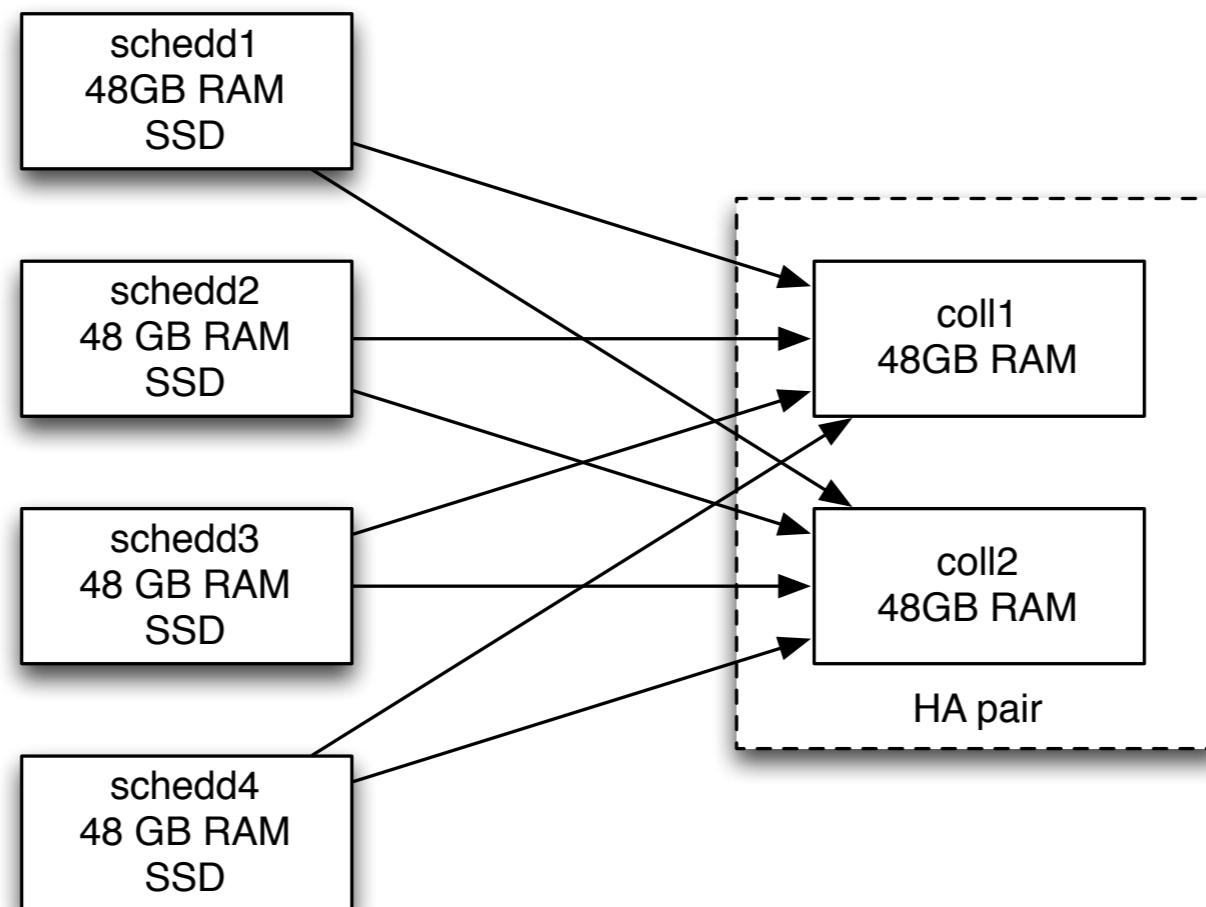
Scale Out - Collector

- We previously discussed using the collector hierarchy to aggregate startd ads. This can be done across several hosts, but likely will not improve your achievable pool size.
- Instead, consider breaking your pool into several independent pools.
 - Through *flocking*, the schedd can participate in several pools.
 - If any one pool goes down, then the schedd can utilize the others transparently.
 - Downside: accounting and fairshare becomes a headache. Currently, HTCondor provides no mechanism to synchronize usage between pools.

Example Deploy @ Scale



Example Deploy @ Scale



Communicate with Humans

- Don't think you need to solve your problems on your own! Utilize the community!
 - On the HTCondor-Users mailing list, you can find others in a similar situation or a sympathetic developer.
 - When we see similar problem areas or solution methods across large sites, we try to consider integrating solutions into future releases.
 - If you struggle with a problem in a dark corner, it greatly diminishes our ability to help you!