Preliminary evaluation of federated SLURM

Kees de Jong, Maxim Masterov
SURFsara
Amsterdam, The Netherlands
kees.dejong@surfsara.nl, maxim.masterov@surfsara.nl

***Abstract*—Central Converged Architecture (CCA) has as a goal to consolidate functionality on a common hardware infrastructure. The hypothesis is that this will create a more cost-effective system, easier management and more flexible access to High Performance Computing (HPC) and high performance storage resources. In practice this concept is planned for the integration of Cartesius [1] and Lisa [2] hardware. This research investigated the advantages and disadvantages of a federated Simple Linux Utility for Resource Management (SLURM) as a first step of a more unified integration between Cartesius and Lisa.**

***Index Terms*—Federated SLURM, HPC, unified computing**

I. Introduction

A. Background

SLURM provides the means to allocate exclusive and/or non-exclusive access to typically HPC compute resources for a duration of time [3]. Therefore, SLURM provides a scheduling framework for starting, executing, and monitoring compute jobs. These are typically parallel Message Passing Interface (MPI) jobs on a set of scheduled compute nodes, or parallel OpenMP jobs on a single scheduled compute node. SLURM also provides the intelligence to manage queues and thus congestion of the compute resources.

Support for creating a federation of clusters was included with the release of SLURM 17.11. This feature enables users to schedule jobs on a range of SLURM clusters with a unified and unique job ID [4]. Each cluster then independently attempts to schedule the job according to its own scheduling policies. Furthermore, the cluster where the job was submitted to (origin cluster) coordinates with the other clusters in the federation to schedule the job. This is done by submitting copies of the compute job (sibling jobs) to each eligible cluster.

B. Experimental setup

SURFsara intends to integrate the Cartesius and Lisa systems on e.g. the job scheduling level. Federated SLURM could assist in this ambition. In order to experiment with federated SLURM, an experimental setup was used (table I). It is assumed that a true SLURM federation will not share a uniform setup. Thus, our experimental setup reflects this by using different versions of SLURM and Linux distributions.

TABLE I
Experimental setup

Cluster name	Linux distribution	SLURM
fedora	Fedora 31	19.05.4
debian	Debian 10	18.08.5
ubuntu	Ubuntu 18.04.3 LTS	17.11.2

C. Known advantages

The main advantages of a federated cluster are as follows:

- Federation of clusters allows to synchronize resources across multiple clusters.
- Based on the requirements of a particular software users can be provided with the best system for their needs (InfiniBand, GPU, NUMA, etc.).
- Users can manage the deployment of their software to different platforms and, therefore, develop more versatile codes.
- The federation may provide better load balancing across the clusters with heterogeneous resources and architectures.
- (Theoretically) It is easy to add a new cluster to the federation and, therefore, provide users with even more resources.
- Because a federation consists of several small clusters (rather than one massive cluster), it is easier to perform fault isolation and maintenance in case of a hardware failure.

D. Known limitations

SchedMD (current maintainer of SLURM) highlighted the following known limitations of a federated cluster [4], [5].

- A federated job that fails due to resources (partition, node counts, etc.) on the local cluster will be rejected and will not be submitted to other sibling clusters even if it could run on them.
- Job arrays only run on the cluster that they were submitted to.
- Job modification must succeed on the origin cluster for the changes to be pushed to the sibling jobs on remote clusters.
- Modifications to anything other than jobs are disabled in `sview`.
- `sview` grid is disabled in a federated view.
- Limited size of federation (64 clusters).
- Federated job IDs are stored in 26 bits, not in 32, as in non-federated SLURM.
- A cluster can be a part of only one federation at a time.
- Jobs cannot span multiple clusters.
- A SLURM federation is not intended as a high-throughput environment (>50,000 jobs a day). However, this can be mediated by using `--cluster-constraint` or the `-M` submission

options to limit the amount of clusters the sibling jobs can be submitted to or directing load to.
- The load manager will have to maintain real-time communication with multiple clusters in order to determine their state and availability. Intensified data transfer between the main cluster and siblings may lead to the network congestion (taking into account slow interconnections between siblings). This may affect the data transfer and "availability" status of siblings.
- Job IDs on each cluster are independent (two jobs can have the same local ID).
- No cross-cluster job dependencies.
- No job migration between clusters.
- No unified view of system state, each cluster is largely independent.
- In case of a large SLURM federation, spread across several owners (e.g. SURF and UvA), the software management becomes complicated.

E. Known features

- `sacctmgr mod cluster` can be used to set a weight on a cluster level, used to prioritize clusters that can start a job the soonest.
- `sacctmgr mod cluster` can also be used to set features on a cluster level, which can be requested by job.
- `sbatch`, `salloc` and `srun` have federation support to submit jobs.

F. Research question

This section briefly discusses the research questions based on the use case and experimental setup (section I-A), and the known limitations (section I-D).

The term federated SLURM implies central control over independent clusters. The main research question therefore is; how federated can federated SLURM actually be and what are the advantages and disadvantages? Furthermore, how applicable is federated SLURM for the Cartesius and Lisa systems?

II. Results

A. Cluster setup

When using SLURM, a central database may be used to archive job accounting. The daemon used for this is called `slurmdbd`. The SLURM controller (`slurmctld`) is responsible for monitoring all the SLURM daemons and resources, accepts work (jobs), and allocates resources to those jobs [6]. The `slurmctld` communicates this information to the `slurmdbd` to be archived to a supported database, i.e. MariaDB [7].

The SLURM release numbers can be distinguished in two parts, the major release number and the maintenance level. Taking the SLURM 17.11.2 release number as an example, the first two numbers represents the major release number, while the last of the three represent the maintenance level. The `slurmdbd` must be at the same or higher major release number as the `slurmctld` daemon [8]. Therefore, the only valid candidate to run the `slurmdbd` in our experimental setup (table I) is the `fedora` cluster, due to having the highest major SLURM release number.

Furthermore, SLURM permits upgrades to a new major release from the past two major releases without the loss of jobs, or other state information. Thus, as also explicitly noted in the official documentation [8], the SLURM versions listed in table I are compatible. If an older major release ought to be used, then state information and RPCs would not be recognized and discarded, resulting in loss of all running and pending jobs. Therefore, the `slurmdbd` version number can only go up, and in our case a SLURM version $<$17.11 would not be compatible (even if those major releases would support federated SLURM).

If a SLURM cluster with the highest major release of the `slurmdbd` decides to leave the federation, another cluster has to step in to replace it with a `slurmdbd` major release version that is equal or higher. Otherwise, database scheme compatibility problems would arise, a SLURM downgrade path is not supported. Therefore, the best forward-compatibility is ensured by having the SLURM daemons in all clusters (regardless of which Linux distribution is used), run the same major version release number. However, we experimented with the versions listed in table I since the documentation states that these SLURM versions are compatible.

Across a (federated) SLURM cluster a uniform user (UID) and group (GID) name space must be assured. This requirement is twofold; to ensure storage permission compatibility throughout the distributed cluster (and network attached file systems), and to ensure that the `SlurmUser` option in `slurm.conf` is mapped to the same numerical UID/GID. Furthermore, accounting is maintained by user name (not UID), however, this also requires uniform user administration across the federated SLURM cluster [9]. Therefore, all federated SLURM clusters must use a uniform user administration, e.g. LDAP.

In order to authenticate the UID and GID of another local or remote process (RPC), an authentication mechanism is used called `munge` [10]. The symmetric key used for authentication must be distributed throughout the federated SLURM cluster. However, it is possible to run multiple `munge` keys by running multiple daemon instances pointing to different keys. Another authentication requirement is time (to prevent replay attacks), thus clocks need to be synchronized throughout the federated SLURM cluster, this is usually done by NTP.

B. Experimentation

As discussed in section II-A, RPC and state information should be compatible between the SLURM versions listed in table I, where the `fedora` cluster acts as the main cluster in the SLURM federation. However, several errors were observed when the SLURM federation was created. Before setting up the federated SLURM cluster, all cluster local jobs completed without issues. However, several errors were introduced after creating the SLURM federation.

On the `debian` cluster we ran the command `sinfo --federation`, which triggered the `Zero Bytes were transmitted or received` error before showing the expected federated SLURM cluster overview. However, the output of the command was incorrect. According to the output, the `fedora` and `debian` clusters were running on the same node, while the `ubuntu` cluster was missing from the overview. Similar errors, but with different combinations were observed on other nodes in the federated SLURM cluster. All `munge` keys were verified with SHA256 and all clocks were synchronized with NTP. The `slurmdbd` log on the main cluster registered the error `federation siblings not synced yet`. Network connections, SLURM daemon and database permissions were verified. However, the errors persisted.

Therefore, as recommended by ourselves in section II-A, we setup a federated SLURM cluster with the same major release versions. However, it was first attempted to experiment with the SLURM major release version included in Debian 10 and that of Fedora 31. This was attempted to eliminate the possibility of incompatibility issues with the oldest SLURM major release version running on the Ubuntu 18.04.3 LTS cluster. However, the same issues were observed.

When experimenting with federated SLURM clusters composed of the same major release versions, using the Linux distributions listed in table I, no errors were observed. The federated SLURM job IDs were assigned and job allocation was successful on all clusters in each of the SLURM federations.

III. Conclusion

In this research we experimented with the SLURM federation feature in order to facilitate job allocation on different SLURM clusters, with the assistance of federated SLURM job IDs. During our experimentation we observed that the different SLURM major release versions were not able to establish such a SLURM federation. However, as discussed in section II-A, the official documentation specified that the SLURM versions in use (table I) are compatible.

Therefore, all clusters within a SLURM federation must use the same SLURM major release version for optimal compatibility. The federated nature is mostly defined by having an independent `slurmctld` which allows each SLURM federation cluster to have control over how jobs are scheduled. However, for ease and consistency it would benefit user friendliness if the SLURM federation members have a uniform job environment. This includes e.g. mount points and paths to software or data and the name space in the module environment. With this consistency it would be possible to utilize a technique called cloud bursting [11]. When job scripts are compatible without alteration on other SLURM federation clusters, the cloud bursting technique can be used to offload automatically to other, less congested SLURM federation cluster members.

This uniform requirement somewhat complicates administration for Cartesius and Lisa. Cartesius depends on SLURM support from Atos [12] while Lisa is independent and compiles and maintains their own SLURM software. At the moment Cartesius uses SLURM 16.05, which lacks support for a SLURM federation. Lisa on the other hand runs the latest major release version SLURM 19.05. Furthermore, a uniform user administration and accounting is already in place for Cartesius and Lisa by the use of LDAP and Customer Organisation LDAP Accounting Service (COLA). However, this does not include the UIDs/GIDs for system users such as `slurm` and `munge`. This can be handled by the respected configuration management tools in use by Cartesius and Lisa. Furthermore, job environments are not uniform, even though this is not a requirement, it would make the SLURM federation more user friendly.

As discussed in section I-D, a SLURM federation can span over multiple independent clusters. But each SLURM cluster can only be a member of one federation. Therefore, careful planning is needed in establishing a SLURM federation, where consensus is needed to create a consistent and reliable environment. Furthermore, job load needs to be taken into account. However, as discussed in section I-D, with some mediations it is possible to make a SLURM federation workable for a high-throughput environment with >50,000 jobs a day. Furthermore, Amazon, Google or Microsoft HPC clouds can also be considered for cloud bursting by the use of a SLURM federation.

References

[1] Cartesius: the Dutch supercomputer. SURFsara. [Online]. Available: https://userinfo.surfsara.nl/systems/cartesius

[2] The Lisa system. SURFsara. [Online]. Available: https://userinfo.surfsara.nl/systems/lisa

[3] Slurm Workload Manager. Wikimedia Foundation, Inc. [Online]. Available: https://en.wikipedia.org/wiki/Slurm_Workload_Manager

[4] Slurm Federated Scheduling Guide. SchedMD LLC. [Online]. Available: https://slurm.schedmd.com/federation.html

[5] Federated Cluster Support. SchedMD LLC. [Online]. Available: https://slurm.schedmd.com/SLUG16/FederatedScheduling.pdf

[6] slurmctld - The central management daemon of Slurm. SchedMD LLC. [Online]. Available: https://slurm.schedmd.com/slurmctld.html

[7] MariaDB: Enterprise Open Source Database. MariaDB. [Online]. Available: https://mariadb.com/

[8] Quick Start Administrator Guide. SchedMD LLC. [Online]. Available: https://slurm.schedmd.com/quickstart_admin.html#upgrade

[9] Accounting and Resource Limits. SchedMD LLC. [Online]. Available: https://slurm.schedmd.com/accounting.html

[10] MUNGE authentication daemon. Chris Dunlap. [Online]. Available: https://dun.github.io/munge/

[11] Slurm Elastic Computing (Cloud Bursting). SchedMD LLC. [Online]. Available: https://slurm.schedmd.com/elastic_computing.html

[12] Atos website. Atos SE. [Online]. Available: https://atos.net/en/