Accounting and Resource Limits

Overview

Slurm can be configured to collect accounting information for every job and job step executed. Accounting records can be written to a simple text file or a database. Information is available about both currently executing jobs and jobs which have already terminated. The `sacct` command can report resource usage for running or terminated jobs including individual tasks, which can be useful to detect load imbalance between the tasks. The `sstat` command can be used to status only currently running jobs. It also can give you valuable information about imbalance between tasks. The `sreport` can be used to generate reports based upon all jobs executed in a particular time interval.

There are three distinct plugin types associated with resource accounting. The Slurm configuration parameters (in slurm.conf) associated with these plugins include:

- **AccountingStorageType** controls how detailed job and job step information is recorded. You can store this information in a text file or into SlurmDBD.
- **JobAcctGatherType** is operating system dependent and controls what mechanism is used to collect accounting information. Supported values are *jobacct_gather/linux*, *jobacct_gather/cgroup* and *jobacct_gather/none* (no information collected).
- **JobCompType** controls how job completion information is recorded. This can be used to record basic job information such as job name, user name, allocated nodes, start time, completion time, exit status, etc. If the preservation of only basic job information is required, this plugin should satisfy your needs with minimal overhead. You can store this information in a text file, or MySQL or MariaDB database.

The use of `sacct` to view information about jobs is dependent upon AccountingStorageType being configured to collect and store that information. The use of `sreport` is dependent upon some database being used to store that information.

The use of `sacct` or sstat to view information about resource usage within jobs is dependent upon both JobAcctGatherType and AccountingStorageType being configured to collect and store that information.

Storing the accounting information into text files is very simple. Just configure the appropriate plugin (e.g. *JobCompType=jobcomp/filetxt*) and then specify the pathname of the file (e.g. *JobCompLoc=/var/log/slurm/job_completions*). Use the logrotate or similar tool to prevent the log files from getting too large. Send a SIGUSR2 signal to the `slurmctld` daemon after moving the files, but before compressing them so that new log files will be created.

Storing the data directly into a database from Slurm may seem attractive, but it requires the availability of user name and password data not only for the Slurm control daemon (slurmctld), but also for user commands which need to access the data (`sacct`, `sreport`, and `sacctmgr`). Making potentially sensitive information available to all users makes database security more difficult to provide. Sending the data through an intermediate daemon can provide better security and performance (through caching data). SlurmDBD (Slurm Database Daemon) provides such services. SlurmDBD is written in C, multi-threaded, secure and fast. The configuration required to use SlurmDBD will be described below. Storing information directly into a database would be similar.

Note that SlurmDBD relies upon existing Slurm plugins for authentication and Slurm SQL for database use, but the other Slurm commands and daemons are not required on the host where SlurmDBD is installed. Install the slurm and slurm-slurmdbd RPMs on the server where SlurmDBD is to run.

Note if you switch from using the MySQL plugin to use the SlurmDBD plugin you must make sure the cluster has been added to the database. The MySQL plugin doesn't require this, but also will not hurt things if you have it there when using the MySQL plugin. You can verify with

```
sacctmgr list cluster
21 symb
```

If the cluster isn't there, add it (where my cluster's name was snowflake):

```
sacctmgr add cluster snowflake
30 symb
```

Failure to do so will result in the `slurmctld` failing to talk to the slurmdbd after the switch. If you plan to upgrade to a new version of Slurm don't switch plugins at the same time or you may get unexpected results. Do one then the other.

If SlurmDBD is configured for use but not responding then `slurmctld` will utilize an internal cache until SlurmDBD is returned to service. The cached data is written by `slurmctld` to local storage upon shutdown and recovered at startup. If SlurmDBD is not available when `slurmctld` starts, a cache of valid bank accounts, user limits, etc. based upon their state when the daemons were last communicating will be used. Note that

SlurmDBD must be responding when `slurmctld` is first started since no cache of this critical data will be available. Job and step accounting records generated by `slurmctld` will be written to a cache as needed and transferred to SlurmDBD when returned to service. Note that if SlurmDBD is down long enough for the number of queued records to exceed the maximum queue size then messages will begin to be dropped.

Infrastructure

With the SlurmDBD, we are able to collect data from multiple clusters in a single location. This does impose some constraints on the user naming and IDs. Accounting is maintained by user name (not user ID), but a given user name should refer to the same person across all of the computers. Authentication relies upon user ID numbers, so those must be uniform across all computers communicating with each SlurmDBD, at least for users requiring authentication. In particular, the configured `SlurmUser` must have the same name and ID across all clusters. If you plan to have administrators of user accounts, limits, etc. they must also have consistent names and IDs across all clusters. If you plan to restrict access to accounting records (e.g. only permit a user to view records of his jobs), then all users should have consistent names and IDs.

NOTE: By default only lowercase usernames are supported, but you can configure **Parameters=PreserveCaseUser** in your slurmdbd.conf to allow usernames with uppercase characters.

The best way to ensure security of the data is by authenticating communications to the SlurmDBD and we recommend MUNGE for that purpose. If you have one cluster managed by Slurm and execute the SlurmDBD on that one cluster, the normal MUNGE configuration will suffice. Otherwise MUNGE should then be installed on all nodes of all Slurm managed clusters, plus the machine where SlurmDBD executes. You then have a choice of either having a single MUNGE key for all of these computers or maintaining a unique key for each of the clusters plus a second key for communications between the clusters for better security. MUNGE enhancements are planned to support two keys within a single configuration file, but presently two different daemons must be started with different configurations to support two different keys (create two key files and start the daemons with the *--key-file* option to locate the proper key plus the *--socket* option to specify distinct local domain sockets for each). The pathname of local domain socket will be needed in the Slurm and SlurmDBD configuration files (slurm.conf and slurmdbd.conf respectively, more details are provided below).

Whether you use any authentication module or not you will need to have a way for the SlurmDBD to get UIDs for users and/or admins. If using MUNGE, it is ideal for your users to have the same id on all your clusters. If this is the case you should have a combination of every cluster's /etc/passwd file on the database server to allow the DBD to resolve names for authentication. If using MUNGE and a user's name is not in the passwd file the action will fail. If not using MUNGE, you should add anyone you want to be an administrator or operator to the passwd file. If they plan on running sacctmgr or any of the accounting tools they should have the same UID, or they will not authenticate correctly. An LDAP server could also serve as a way to gather this information.

Slurm JobComp Configuration

Presently job completion is not supported with the SlurmDBD, but can be written directly to a database, script or flat file. If you are running with the accounting storage plugin, use of the job completion plugin is probably redundant. If you would like to configure this, some of the more important parameters include:

- **JobCompHost**: Only needed if using a database. The name or address of the host where the database server executes.
- **JobCompLoc**: Only needed if using a flat file. Location of file to write the job completion data to.

- **JobCompPass**: Only needed if using a database. Password for the user connecting to the database. Since the password can not be securely maintained, storing the information directly in a database is not recommended.
- **JobCompPort**: Only needed if using a database. The network port that the database accepts communication on.
- **JobCompType**: Type of jobcomp plugin set to "jobcomp/mysql" or "jobcomp/filetxt".
- **JobCompUser**: Only needed if using a database. User name to connect to the database with.
- **JobCompParams**: Pass arbitrary text string to job completion plugin.

Slurm Accounting Configuration Before Build

You can configure SlurmDBD to communicate with a database by using **AccountingStorageType=accounting_storage/slurmdbd.** This allows the creation of user entities called "associations", which consist of the cluster, a user, account and optionally a partition.

MySQL or MariaDB is the preferred database.

NOTE: If you have an existing Slurm accounting database and plan to upgrade your database server to MariaDB 10.2.1 (or newer) from a pre-10.2.1 version or from any version of MySQL, please contact SchedMD for assistance.

To enable this database support one only needs to have the development package for the database they wish to use on the system. **Slurm uses the InnoDB storage engine in MySQL to make rollback possible. This must be available on your MySQL installation or rollback will not work.**

The slurm configure script uses mysql_config to find out the information it needs about installed libraries and headers. You can specify where your mysql_config script is with the *--with-mysql_conf=/path/to/mysql_config* option when configuring your slurm build. On a successful configure, output is something like this:

```
checking for mysql_config... /usr/bin/mysql_config
MySQL test program built properly.

90 symb
```

NOTE: Before running the slurmdbd for the first time, review the current setting for MySQL's *innodb_buffer_pool_size*. Consider setting this value large enough to handle the size of the database. Having this value too small can be problematic when converting large tables over to the new database schema or when purging old records. We recommend assigning a significant portion of the system memory to this, keeping in mind the other resource requirements on the machine running MySQL/MariaDB, somewhere between 5 and 50 percent of the available memory. When using **AccountingStoreFlags=job_env,job_script** or older SQL servers, it is also important to check the value of max_allowed_packet. When the packet size is too small and a large job script is used, the SQL server may reject the sql query as being too large. This value should be at least 16MB, and must be larger than the value of **max_script_size**. Setting innodb_lock_wait_timeout and innodb_log_file_size to larger values than the default is also recommended.

See the following example:

```
mysql> SHOW VARIABLES LIKE 'innodb_buffer_pool_size';
+-------------------------+------------+
| Variable_name           | Value      |
+-------------------------+------------+
| innodb_buffer_pool_size | 4294967296 |
+-------------------------+------------+
1 row in set (0.001 sec)


$cat my.cnf
...
[mysqld]
innodb_buffer_pool_size=4096M
innodb_log_file_size=64M
innodb_lock_wait_timeout=900
max_allowed_packet=16M
...

493 symb
```

Also, in MySQL versions prior to 5.7 the default row format was set to COMPACT which could cause some issues during an upgrade when creating tables. In more recent versions it was changed to DYNAMIC. The row format of a table determines how its rows are physically stored in pages and directly affects the performance of queries and DML operations. In very specific situations using a format other than DYNAMIC can lead to rows not fitting into pages and MySQL can throw an error during the creation of the table because of that. Therefore it is recommended to read carefully about the row format before creating your database tables if you are not using DYNAMIC by default, and consider setting that if your database version supports it. If the following InnoDB error shows up during an upgrade, the table can then be altered (may take some time) to set the row format to DYNAMIC in order to allow the conversion to proceed:

```
[Warning] InnoDB: Cannot add field ... in table ... because after adding it, the
row size is Y which is greater than maximum allowed size (X) for a record on index
leaf page.

174 symb
```

You can see what the default row format is by showing the innodb_default_row_format variable:

```
mysql> SHOW VARIABLES LIKE 'innodb_default_row_format';
+---------------------------+---------+
| Variable_name             | Value   |
+---------------------------+---------+
| innodb_default_row_format | dynamic |
+---------------------------+---------+
1 row in set (0.001 sec)
```

```
310 symb
```

You can also see how the tables are created by running the following command, where *db_name* is the name of your Slurm database (StorageLoc) set in your slurmdbd.conf:

```
mysql> SHOW TABLE STATUS IN db_name;

36 symb
```

Slurm Accounting Configuration After Build

For simplicity's sake, we are going to proceed under the assumption that you are running with the SlurmDBD. You can communicate with a storage plugin directly, but that offers minimal security.

Several Slurm configuration parameters must be set to support archiving information in SlurmDBD. SlurmDBD has a separate configuration file which is documented in a separate section. Note that you can write accounting information to SlurmDBD while job completion records are written to a text file or not maintained at all. If you don't set the configuration parameters that begin with "AccountingStorage" then accounting information will not be referenced or recorded.

- **AccountingStorageEnforce**: This option contains a comma separated list of options you may want to enforce. The valid options are any comma separated combination of

-
    - associations - This will prevent users from running jobs if their association is not in the database. This option will prevent users from accessing invalid accounts.
-
    - limits - This will enforce limits set on associations and qos'. By setting this option, the 'associations' option is automatically set. If a qos is used the limits will be enforced, but 'qos' described below is still needed if you want to enforce access to the qos.
-
    - nojobs - This will make it so no job information is stored in accounting. By setting this 'nosteps' is also set.
-
    - nosteps - This will make it so no step information is stored in accounting. Both nojobs and nosteps could be helpful in an environment where you want to use limits but don't really care about utilization.
-
    - qos - This will require all jobs to specify (either overtly or by default) a valid qos (Quality of Service). QOS values are defined for each association in the database. By setting this option, the 'associations' option is automatically set. If you want QOS limits to be enforced you need to use the 'limits' option.
-
    - safe - This will ensure a job will only be launched when using an association or qos that has a TRES-minutes limit set if the job will be able to run to completion. Without this option set, jobs will be launched as long as their usage hasn't reached the TRES-minutes limit which can lead to jobs being launched but then killed when the limit is reached. With the 'safe' option set, a job won't be killed due to limits, even if the limits are changed after a job was started and the association or qos violates the updated limits. By setting this option, both the 'associations' option and the 'limits' option are set automatically.

- - wckeys - This will prevent users from running jobs under a wckey that they don't have access to. By using this option, the 'associations' option is automatically set. The 'TrackWCKey' option is also set to true.

- **NOTE**: The association is a combination of cluster, account, user names and optional partition name. **Without AccountingStorageEnforce being set (the default behavior) jobs will be executed based upon policies configured in Slurm on each cluster.**

- **AccountingStorageExternalHost**: A comma separated list of external slurmdbds `(<host/ip>[:port][,...])` to register with. If no port is given, the AccountingStoragePort will be used. This allows clusters registered with the external slurmdbd to communicate with each other using the --cluster/-M client command options. The cluster will add itself to the external slurmdbd if it doesn't exist. If a non-external cluster already exists on the external slurmdbd, the slurmctld will ignore registering to the external slurmdbd.

- **AccountingStorageHost**: The name or address of the host where SlurmDBD executes

- **AccountingStoragePass**: If using SlurmDBD with a second MUNGE daemon, store the pathname of the named socket used by MUNGE to provide enterprise-wide authentication (i.e. /var/run/munge/moab.socket.2). Otherwise the default MUNGE daemon will be used.

- **AccountingStoragePort**: The network port that SlurmDBD accepts communication on.

- **AccountingStorageType**: Set to "accounting_storage/slurmdbd".

- **ClusterName**: Set to a unique name for each Slurm-managed cluster so that accounting records from each can be identified.

- **TrackWCKey**: Boolean. If you want to track wckeys (Workload Characterization Key) of users. A Wckey is an orthogonal way to do accounting against possibly unrelated accounts. When a job is run, use the --wckey option to specify a value and accounting records will be collected by this wckey.

SlurmDBD Configuration

SlurmDBD requires its own configuration file called "slurmdbd.conf". This file should be only on the computer where SlurmDBD executes and should only be readable by the user which executes SlurmDBD (e.g. "slurm"). This file should be protected from unauthorized access since it contains a database login name and password. See slurmdbd.conf(5) for a more complete description of the configuration parameters. Some of the more important parameters include:

- **AuthInfo**: If using SlurmDBD with a second MUNGE daemon, store the pathname of the named socket used by MUNGE to provide enterprise-wide. Otherwise the default MUNGE daemon will be used.
- **AuthType**: Define the authentication method for communications between Slurm components. A value of "auth/munge" is recommended.
- **DbdHost:** The name of the machine where the Slurm Database Daemon is executed. This should be a node name without the full domain name (e.g. "lx0001"). This defaults to localhost but should be supplied to avoid a warning message.
- **DbdPort**: The port number that the Slurm Database Daemon (slurmdbd) listens to for work. The default value is SLURMDBD_PORT as established at system build time. If none is explicitly specified, it will be set to 6819. This value must be equal to the AccountingStoragePort parameter in the slurm.conf file.

- **LogFile**: Fully qualified pathname of a file into which the Slurm Database Daemon's logs are written. The default value is none (performs logging via syslog).
- **PluginDir**: Identifies the places in which to look for Slurm plugins. This is a colon-separated list of directories, like the PATH environment variable. The default value is the prefix given at configure time + "/lib/slurm".
- **SlurmUser**: The name of the user that the slurmdbd daemon executes as. This user must exist on the machine executing the Slurm Database Daemon and have the same UID as the hosts on which slurmctld execute. For security purposes, a user other than "root" is recommended. The default value is "root". This name should also be the same SlurmUser on all clusters reporting to the SlurmDBD. NOTE: If this user is different from the one set for slurmctld and is not root, it must be added to accounting with AdminLevel=Admin and slurmctld must be restarted.
- **StorageHost**: Define the name of the host the database is running where we are going to store the data. Ideally this should be the host on which SlurmDBD executes, but could be a different machine.
- **StorageLoc**: Specifies the name of the database where accounting records are written. For databases the default database is slurm_acct_db. Note the name can not have a '/' in it or the default will be used.
- **StoragePass**: Define the password used to gain access to the database to store the job accounting data.
- **StoragePort**: Define the port on which the database is listening.
- **StorageType**: Define the accounting storage mechanism. The only acceptable value at present is "accounting_storage/mysql". The value "accounting_storage/mysql" indicates that accounting records should be written to a MySQL or MariaDB database specified by the StorageLoc parameter. This value must be specified.
- **StorageUser**: Define the name of the user we are going to connect to the database with to store the job accounting data.

SLURMDBD ARCHIVE AND PURGE

As time goes on, the slurm database can grow large enough that it is hard to manage. To maintain the database at a reasonable size, slurmdbd supports archiving and purging data based on its age. Purged data will be deleted from the database, but you can choose to archive the data as it is being purged. Archived data will be placed in flat files that can later be loaded into a slurmdbd by sacctmgr.

Archive and Purge options come in the form of `Archive${*}` and `Purge${*}After`. See slurmdbd.conf(5) for more details on the available configuration parameters.

The units for the purge options are important. For example: *PurgeJobsAfter=12months* will purge jobs more than 12 months old at the beginning of each month, while *PurgeJobsAfter=365days* will purge jobs older than 365 days old at the beginning of each day. This distinction can be useful for very active clusters, reducing the amount of data that needs to be purged at one time.

MySQL Configuration

NOTE: If you have an existing Slurm accounting database and plan to upgrade your database server to MariaDB 10.2.1 (or newer) from a pre-10.2.1 version or from any version of MySQL, please contact SchedMD for assistance.

While Slurm will create the database tables automatically you will need to make sure the StorageUser is given permissions in the MySQL or MariaDB database to do so. As the mysql user grant privileges to that user using a command such as:

```
GRANT ALL ON StorageLoc.* TO 'StorageUser'@'StorageHost';
(The ticks are needed)

85 symb
```

You need to be root to do this. Also in the info for password usage there is a line that starts with '->'. This a continuation prompt since the previous mysql statement did not end with a ';'. It assumes that you wish to input more info.

If you want Slurm to create the database itself, and any future databases, you can change your grant line to be . instead of StorageLoc.*

Live example:

```
mysql@snowflake:~$ mysql
Welcome to the MySQL monitor.Commands end with ; or \g.
Your MySQL connection id is 538
Server version: 5.0.51a-3ubuntu5.1 (Ubuntu)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> create user 'slurm'@'localhost' identified by 'password';
Query OK, 0 rows affected (0.00 sec)

mysql> grant all on slurm_acct_db.* TO 'slurm'@'localhost';
Query OK, 0 rows affected (0.00 sec)

463 symb
```

You may also need to do the same with the system name in order for mysql to work correctly:

```
mysql> grant all on slurm_acct_db.* TO 'slurm'@'system0';
Query OK, 0 rows affected (0.00 sec)
where 'system0' is the localhost or database storage host.

163 symb
```

or with a password...

```
mysql> grant all on slurm_acct_db.* TO 'slurm'@'localhost'
    -> identified by 'some_pass' with grant option;
Query OK, 0 rows affected (0.00 sec)
```

```
158 symb
```

The same is true in the case, you made to do the same with the system name:

```
mysql> grant all on slurm_acct_db.* TO 'slurm'@'system0'
    -> identified by 'some_pass' with grant option;
where 'system0' is the localhost or database storage host.

177 symb
```

Verify you have InnoDB support

```
 mysql> SHOW ENGINES;
```

Engine	Support	Comment	Transactions	XA	SavePoints
InnoDB	Default	Supports transactions, row-level locking, and foreign keys	YES	YES	YES

```
376 symb
```

Then create the database:

```
mysql> create database slurm_acct_db;

37 symb
```

This will grant user 'slurm' access to do what it needs to do on the local host or the storage host system. This must be done before the SlurmDBD will work properly. After you grant permission to the user 'slurm' in mysql then you can start SlurmDBD and the other Slurm daemons. You start SlurmDBD by typing its pathname '/usr/sbin/slurmdbd' or '/etc/init.d/slurmdbd start'. You can verify that SlurmDBD is running by typing 'ps aux | grep slurmdbd'.

If the SlurmDBD is not running you can use the -v option when you start SlurmDBD to get more detailed information. Starting the SlurmDBD in daemon mode with the '-D' option can also help in debugging so you don't have to go to the log to find the problem.

Archive Server

If ongoing access to Archived/Purged data is required at your site, it is possible to create an archive instance of slurmdbd. Data previously archived and purged from the production database can be loaded into the archive server, keeping the production database at a manageable size while making sure old records are still accessible.

The archive instance of slurmdbd should not be able to communicate with the production server. Ideally they would have separate instances of MySQL/MariaDB that they use to store their data. The Slurm controller (slurmctld) should never communicate with the archive slurmdbd.

When configuring an archive server, there are certain database entries that need to match the production server in order for the archived information to show up correctly. In order to make sure the unique identifiers match, mysqldump should be used to export the Association, QOS and TRES information. The command to export these tables should look like this, with the appropriate values substituted for `<slurm_user>`, `<db_name>`, and `<cluster>`:

```
mysqldump -u <slurm_user> -p <db_name> <cluster>_assoc_table qos_table tres_table
> slurm.sql

93 symb
```

While mysqldump should be used to transfer the information from these tables, it should not be used to transfer information that will be generated with the Archive/Purge process. If mysqldump is used to try to get the desired information, there will likely be a slight difference and when trying to load archive files later there will either be a gap in records or duplicate records that prevent the archive file from loading correctly.

Tools

There are a few tools available to work with accounting data, `sacct`, `sacctmgr`, and `sreport`. These tools all get or set data through the SlurmDBD daemon.

- **sacct** is used to generate accounting report for both running and completed jobs.
- **sacctmgr** is used to manage associations in the database: add or remove clusters, add or remove users, etc.
- **sreport** is used to generate various reports on usage collected over a given time period.

See the man pages for each command for more information.

Database Configuration

Accounting records are maintained based upon what we refer to as an *Association*, which consists of four elements: cluster, account, user names and an optional partition name. Use the *sacctmgr* command to create and manage these records.

NOTE: There is an order to set up accounting associations. You must define clusters before you add accounts and you must add accounts before you can add users.

For example, to add a cluster named "snowflake" to the database execute this line (**NOTE**: as of 20.02, slurmctld will add the cluster to the database upon start if it doesn't exist. Associations still need to be created after addition):

```
sacctmgr add cluster snowflake

30 symb
```

Add accounts "none" and "test" to cluster "snowflake" with an execute line of this sort:

```
sacctmgr add account none,test Cluster=snowflake \
Description="none" Organization="none"

94 symb
```

If you have more clusters you want to add these accounts, to you can either not specify a cluster, which will add the accounts to all clusters in the system, or comma separate the cluster names you want to add to in the cluster option. Note that multiple accounts can be added at the same time by comma separating the names. A *description* of the account and the organization to which it belongs must be specified. These terms can be used later to generate accounting reports. Accounts may be arranged in a hierarchical fashion. For example, accounts *chemistry* and physics may be children of the account science. The hierarchy may have an arbitrary depth. Just specify the *parent*='' option in the add account line to construct the hierarchy. For the example above execute

```
sacctmgr add account science \
Description="science accounts" Organization=science
sacctmgr add account chemistry,physics parent=science \
Description="physical sciences" Organization=science

206 symb
```

Add users to accounts using similar syntax. For example, to permit user *da* to execute jobs on all clusters with a default account of *test execute*:

```
sacctmgr add user brian Account=physics
sacctmgr add user da DefaultAccount=test

85 symb
```

If **AccountingStorageEnforce=associations** is configured in the slurm.conf of the cluster snowflake then user da would be allowed to run in account test and any other accounts added in the future. Any attempt to use other accounts will result in the job being aborted. Account test will be the default if he doesn't specify one in the job submission command.

Associations can also be created that are tied to specific partitions. When using the "add user" command of **sacctmgr** you can include the `Partition=<PartitionName>` option to create an association that is unique to other associations with the same Account and User.

Cluster Options

When either adding or modifying a cluster, these are the options available with sacctmgr:

```
Name= Cluster name

18 symb
```

Account Options

When either adding or modifying an account, the following sacctmgr options are available:

- **Cluster=** Only add this account to these clusters. The account is added to all defined clusters by default.
- **Description=** Description of the account. (Default is account name)
- **Name=** Name of account. Note the name must be unique and can not represent different bank accounts at different points in the account hierarchy
- **Organization=**Organization of the account. (Default is parent account unless parent account is root then organization is set to the account name.)
- **Parent=** Make this account a child of this other account (already added).

User Options

When either adding or modifying a user, the following sacctmgr options are available:

- Account= Account(s) to add user to
- AdminLevel= This field is used to allow a user to add accounting privileges to this user. Valid options are
- - None
- - Operator: can add, modify, and remove any database object (user, account, etc), and add other operators
    On a SlurmDBD served slurmctld these users can
- - - View information that is blocked to regular uses by a PrivateData flag
- - - Create/Alter/Delete Reservations
- - Admin: These users have the same level of privileges as an operator in the database. They can also alter anything on a served slurmctld as if they were the slurm user or root.
- Cluster= Only add to accounts on these clusters (default is all clusters)
- DefaultAccount= Default account for the user, used when no account is specified when a job is submitted. (Required on creation)
- DefaultWCKey= Default wckey for the user, used when no wckey is specified when a job is submitted. (Only used when tracking wckeys.)
- Name= User name
- NewName= Use to rename a user in the accounting database
- Partition= Name of Slurm partition this association applies to

Limit Enforcement

Various limits and limit enforcement are described in the Resource Limits web page.

To enable any limit enforcement you must at least have **AccountingStorageEnforce=limits** in your slurm.conf. Otherwise, even if you have limits set, they will not be enforced. Other options for AccountingStorageEnforce and the explanation for each are found on the Resource Limits document.

Modifying Entities

When modifying entities, you can specify many different options in SQL-like fashion, using key words like where and set. A typical execute line has the following form:

```
sacctmgr modify <entity> set <options> where <options>

54 symb
```

For example:

```
sacctmgr modify user set default=none where default=test

56 symb
```

will change all users with a default account of "test" to account "none". Once an entity has been added, modified or removed, the change is sent to the appropriate Slurm daemons and will be available for use instantly.\

Removing Entities

Removing entities using an execute line similar to the modify example above, but without the set options. For example, remove all users with a default account "test" using the following execute line:

```
sacctmgr remove user where default=test

39 symb
```

will remove all user records where the default account is "test".

```
sacctmgr remove user brian where account=physics

48 symb
```

will remove user "brian" from account "physics". If user "brian" has access to other accounts, those user records will remain.

Note: In most cases, removed entities are preserved in the slurm database, but flagged as deleted. If an entity has existed for less than 1 day, the entity will be removed completely. This is meant to clean up after typographical errors. Removing user associations or accounts, however, will cause slurmctld to lose track of usage data for that user/account.

Symbols

Total	Uncounted	Sum
35118	3336	31782