List of abbreviations used

API	Application programming interface	Интерфейс программирования приложения
TRES	Trackable Resources	Отслеживаемые ресурсы
GRES	Generic Resources	Обобщённые (без типа) ресурсы
QOS	Quality of service	Гарантированное качетсво обслуживания

Overview

This guide explains how to use Slurm burst buffer plugins. Where appropriate, it explains how these plugins work in order to give guidance about how to best use these plugins.

The Slurm burst buffer plugins call a script at different points during the lifetime of a job:

1. At job submission
2. While the job is pending after an estimated start time is established
3. Once the job has been scheduled but has not started running yet
4. Once the job has completed or been cancelled, but Slurm has not released resources for the job yet
5. Once the job has completed, and Slurm has released resources for the job

This script runs on the slurmctld node. These are the supported plugins:

- datawarp
- lua

Datawrap

This plugin provides hooks to Cray’s Datawarp APIs. Datawarp implements burst buffers, which are a shared high-speed storage resource. Slurm provides support for allocating these resources, staging files in, scheduling compute nodes for jobs using these resources, and staging files out. Burst buffers can also be used as temporary storage during a job's lifetime, without file staging. Another typical use case is for persistent storage, not associated with any specific job.

Lua

This plugin provides hooks to an API that is defined by a Lua script. This plugin was developed to provide system administrators with a way to do any task (not only file staging) at different points in a job’s life cycle. These tasks might include file staging, node maintenance, or any other task that is desired to run during one or more of the five job states listed above.

The burst buffer APIs will only be called for a job that specifically requests using them. The Job Submission Commands section explains how a job can request using the burst buffer APIs.

Configuration (for system administrators)

Common configuration

- To enable a burst buffer plugin, set BurstBufferType in slurm.conf. If it is not set, then no burst buffer plugin will be loaded. Only one burst buffer plugin may be specified.
- In slurm.conf, you may set DebugFlags=BurstBuffer for detailed logging from the burst buffer plugin. This will result in very verbose logging and is not intended for prolonged use in a production system, but this may be useful for debugging.
- TRES limits for burst buffers can be configured by association or QOS in the same way that TRES limits can be configured for nodes, CPUs, or any GRES. To make Slurm track burst buffer resources, add bb/datawarp (for the datawarp plugin) or bb/lua (for the lua plugin) to AccountingStorageTres in slurm.conf.
- The size of a job's burst buffer requirements can be used as a factor in setting the job priority as described in the multifactor priority document. The Burst Buffer Resources section explains how these resources are defined.
- Burst-buffer-specific configurations can be set in burst_buffer.conf. Configuration settings include things like which users may use burst buffers, timeouts, paths to burst buffer scripts, etc. See the burst_buffer.conf manual for more information.
- The JSON-C library must be installed in order to build Slurm’s burst_buffer/datawarp and burst_buffer/lua plugins, which must parse JSON format data. See Slurm's JSON installation information for details.

Datawrap

`slurm.conf`:

```
BurstBufferType=burst_buffer/datawarp
```

The datawarp plugin calls two scripts:

- dw_wlm_cli - the Slurm burst_buffer/datawarp plugin calls this script to perform burst buffer functions. It should have been provided by Cray. The location of this script is defined by GetSysState in burst_buffer.conf. A template of this script is provided with Slurm: `src/plugins/burst_buffer/datawarp/dw_wlm_cli`
- dwstat - the Slurm burst_buffer/datawarp plugin calls this script to get status information. It should have been provided by Cray. The location of this script is defined by GetSysStatus in burst_buffer.conf. A template of this script is provided with Slurm: `src/plugins/burst_buffer/datawarp/dwstat`

Lua

`slurm.conf`:

```
BurstBufferType=burst_buffer/lua
```

The lua plugin calls a single script which must be named burst_buffer.lua. This script needs to exist in the same directory as slurm.conf. The following functions are required to exist, although they may do nothing but

return success:

- slurm_bb_job_process
- slurm_bb_pools
- slurm_bb_job_teardown
- slurm_bb_setup
- slurm_bb_data_in
- slurm_bb_real_size
- slurm_bb_paths
- slurm_bb_pre_run
- slurm_bb_post_run
- slurm_bb_data_out
- slurm_bb_get_status A template of burst_buffer.lua is provided with Slurm: `etc/burst_buffer.lua.example`

This template documents many more details about the functions such as required parameters, when each function is called, return values for each function, and some simple examples.

Lua Implementation

This purpose of this section is to provide additional information about the Lua plugin to help system administrators who desire to implement the Lua API. The most important points in this section are:

- Some functions in burst_buffer.lua must run quickly and cannot be killed; the remaining functions are allowed to run for as long as needed and can be killed.
- A maximum of 512 copies of burst_buffer.lua are allowed to run concurrently in order to avoid exceeding system limits.

How does burst_buffer.lua run?

Lua scripts may either be run by themselves in a separate process via the `fork()` and `exec()` system calls, or they may be called via Lua's C API from within an existing process. One of the goals of the lua plugin was to avoid calling `fork()` from within `slurmctld` because it can severely harm performance of the `slurmctld`. The datawarp plugin calls `fork()` and `exec()` from `slurmctld` for every burst buffer API call, and this has been shown to severely harm `slurmctld` performance. Therefore, `slurmctld` calls burst_buffer.lua using Lua's C API instead of using `fork()`.

Some functions in burst_buffer.lua are allowed to run for a long time, but they may need to be killed if the job is cancelled, if `slurmctld` is restarted, or if they run for longer than the configured timeout in burst_buffer.conf. However, a call to a Lua script via Lua's C API cannot be killed from within the same process; only killing the entire process that called the Lua script can kill the Lua script.

To address this situation, burst_buffer.lua is called in two different ways:

- The slurm_bb_job_process, slurm_bb_pools and slurm_bb_paths functions are called from slurmctld. Because of the explanation above, a script running one of these functions cannot be killed. Since these functions are called while slurmctld holds some mutexes, it will be extremely harmful to slurmctld performance and responsiveness if they are slow. Because it is faster to call these functions directly than

to call fork() to create a new process, this was deemed an acceptable tradeoff. As a result, these functions cannot be killed.
- The remaining functions in burst_buffer.lua are able to run longer without adverse effects. These need to be able to be killed. These functions are called from a lightweight Slurm daemon called slurmscriptd. Whenever one of these functions needs to run, slurmctld tells slurmscriptd to run that function; slurmscriptd then calls fork() to create a new process, then calls the appropriate function. This avoids calling fork() from slurmctld while still providing a way to kill running copies of burst_buffer.lua when needed. As a result, these functions can be killed, and they will be killed if they run for longer than the appropriate timeout value as configured in burst_buffer.conf. The way in which each function is called is also documented in the burst_buffer.lua.example file.

Limitations

Each copy of the script that runs via slurmscriptd runs in a new process and a pipe (which is a file) is opened to read responses from the script. To avoid exceeding the open process or open file limits, running out of memory, or exceeding other system limitations, a maximum of 128 copies of the script are allowed to run concurrently per "stage", where the stages are stage-in, pre-run, stage-out, and teardown (see the Burst Buffer States section for more information on the burst buffer stages). This means that a maximum of 512 copies of burst_buffer.lua may run at one time. Without this limit, job throughput was lower and testing confirmed that the process open file limit could be exceeded.

WARNING: Do not install a signal handler in burst_buffer.lua because it is called directly from slurmctld. If slurmctld receives a signal, it could attempt to run the signal handler from burst_buffer.lua, even after a call to burst_buffer.lua is completed, which results in a crash.

Burst Buffer Resources

The burst buffer API may define burst buffer resource “pools” from which a job may request a certain amount of pool space. If a pool does not have sufficient space to fulfill a job’s request, that job will remain pending until the pool does have enough space. Once the pool has enough space, Slurm may begin stage-in for the job. When stage-in begins, Slurm subtracts the job’s requested space from the pool’s available space. When teardown completes, Slurm adds the job’s requested space back into the pool’s available space. The Job Submission Commands section explains how a job may request space from a pool. Pool space is a scalar quantity.

Datawrap

- Pools are defined by dw_wlm_cli, and represent bytes. This script prints a JSON-formatted string defining the pools to stdout.
- If a job does not request a pool, then the pool defined by DefaultPool in burst_buffer.conf will be used. If a job does not request a pool and DefaultPool is not defined, then the job will be rejected.

Lua

- Pools are optional in this plugin, and can represent anything.
- DefaultPool in burst_buffer.conf is not used in this plugin.

- Pools are defined by burst_buffer.lua in the function slurm_bb_pools. If pools are not desired, then this function should just return slurm.SUCCESS. If pools are desired, then this function should return two values: (1) slurm.SUCCESS, and (2) a JSON-formatted string defining the pools. An example is provided in burst_buffer.lua.example. The current valid fields in the JSON string are:
- - id - a string defining the name of the pool
- - quantity - a number defining the amount of space in the pool
- - granularity - a number defining the lowest resolution of space that may be allocated from this pool. If a job does not request a number that is a multiple of granularity, then the job's request will be rounded up to the nearest multiple of granularity. For example, if granularity equals 1000, then the smallest amount of space that may be allocated from this pool for a single job is 1000. If a job requests less than 1000 units from this pool, then the job's request will be rounded up to 1000.

Job Submission Commands

The normal mode of operation is for batch jobs to specify burst buffer requirements within the batch script. Commented batch script lines containing a specific directive (depending on which plugin is being used) will inform Slurm that it should run the burst buffer stages for that job. These lines will also describe the burst buffer requirements for the job.

The salloc and srun commands can specify burst buffer requirements with the --bb and --bbf options. This is described in the Command-line Job Options section.

All burst buffer directives should be specified in comments at the top of the batch script. They may be placed before, after, or interspersed with any #SBATCH directives. All burst buffer stages happen at specific points in the job's life cycle, as described in the Overview section; they do not happen during the job's execution. For example, all of the persistent burst buffer (used only by the datawarp plugin) creations and deletions happen before the job's compute portion happens. In a similar fashion, you can't run stage-in at various points in the script execution; burst buffer stage-in is performed before the job begins and stage-out is performed after the job completes.

For both plugins, a job may request a certain amount of space (size or capacity) from a burst buffer resource pool.

- A pool specification is simply a string that matches the name of the pool. For example: pool=pool1
- A capacity specification is a number indicating the amount of space required from the pool. A capacity specification can include a suffix of "N" (nodes), "K|KiB", "M|MiB", "G|GiB", "T|TiB", "P|PiB" (for powers of 1024) and "KB", "MB", "GB", "TB", "PB" (for powers of 1000). NOTE: Usually Slurm interprets KB, MB, GB, TB, PB, units as powers of 1024, but for Burst Buffers size specifications Slurm supports both IEC/SI formats. This is because the CRAY API supports both formats.

At job submission, Slurm performs basic directive validation and also runs a function in the burst buffer script. This function can perform validation of the directives used in the job script. If Slurm determines options are invalid, or if the burst buffer script returns an error, the job will be rejected and an error message will be returned directly to the user.

Note that unrecognized options may be ignored in order to support backward compatibility (i.e. a job submission would not fail in the case of an option recognized by some versions of Slurm, but not recognized

by other versions). If the job is accepted, but later fails (e.g. some problem staging files), the job will be held and its "Reason" field will be set to an error message provided by the underlying infrastructure.

Users may also request to be notified by email upon completion of burst buffer stage out using the --mail-type=stage_out or --mail-type=all option. The subject line of the email will be of this form:

```
SLURM Job_id=12 Name=my_app Staged Out, StageOut time 00:05:07
```

The following plugin subsections give additional information that is specific to each plugin and provide example job scripts. Command-line examples are given in the Command-line Job Options section.

Datawrap

The directive of `#DW` (for "DataWarp") is used for burst buffer directives when using the burst_buffer/datawarp plugin. Please reference Cray documentation for details about the DataWarp options. For DataWarp systems, the directive of #BB can be used to create or delete persistent burst buffer storage. NOTE: The #BB directive is used since the command is interpreted by Slurm and not by the Cray Datawarp software. This is discussed more in the Persistent Burst Buffer section.

For job-specific burst buffers, it is required to specify a burst buffer capacity. If the job does not specify capacity then the job will be rejected. A job may also specify the pool from which it wants resources; if the job does not specify a pool, then the pool specified by DefaultPool in burst_buffer.conf will be used (if configured).

The following job script requests burst buffer resources from the default pool and requests files to be staged in and staged out:

```
#!/bin/bash
#DW jobdw type=scratch capacity=1GB access_mode=striped,private pfs=/scratch
#DW stage_in type=file source=/tmp/a destination=/ss/file1
#DW stage_out type=file destination=/tmp/b source=/ss/file1
srun application.sh
```

Lua

The default directive for this plugin is #BB_LUA. The directive used by this plugin may be changed by setting the Directive option in burst_buffer.conf. Since the directive must always begin with a # sign (which starts a comment in a shell script) this option should specify only the string following the # sign. For example, if burst_buffer.conf contains the following:

```
Directive=BB_EXAMPLE
```

then the burst buffer directive will be #BB_EXAMPLE.

If the Directive option is not specified in burst_buffer.conf, then the default directive for this plugin (#BB_LUA) will be used.

Since this plugin was designed to be generic and flexible, this plugin only requires the directive to be given. If the directive is given, Slurm will run all burst buffer stages for the job.

Example of the minimum information required for all burst buffer stages to run for the job:

```
#!/bin/bash
#BB_LUA
srun application.sh
```

Because burst buffer pools are optional for this plugin (see the Burst Buffer Resources section), a job is not required to specify a pool or capacity. If pools are provided by the burst buffer API, then a job may request a pool and capacity:

```
#!/bin/bash
#BB_LUA pool=pool1 capacity=1K
srun application.sh
```

A job may choose whether or not to specify a pool. If a job does not specify a pool, then the job is still allowed to run and the burst buffer stages will still run for this job (as long as the burst buffer directive was given). If the job specifies a pool but that pool is not found, then the job is rejected.

The system administrator may validate burst buffer options in the slurm_bb_job_process function in burst_buffer.lua. This might include requiring a job to specify a pool or validating any additional options that the system administrator decides to implement.

Введение

Это рукводство показывает как использовать плагины пакетного буфера Slurm. Там, где это возможно, даётся разъяснение по работе дополнений, чтобы дать рекомендации по наилучшему их использованию.

Дополнение пакетного буфра Slurm вызывает скрипт на различных этапах жизненного цикла задачи:

1. При создании задачи
2. Пока задача находится в режиме ожидания и оценивается её примерное время выполнения
3. Когда задача попадает в очередь выполнения, но ещё не запущена на выполнение
4. Когда задача выполнилась или была отменена, но Slurm ещё не освободил занятые ей ресурсы
5. Когда задача завершена и Slurm освободил занятые ей ресурсы

Скрипт запускается на узле, где запущен демон slurmctld. Он поддерживает следующие дополнения:

- datawrap
- lua

Datawrap

Этот плагин обеспечивает обратные вызовы к API Cray's Datawrap. Datawrap имплементирует пакетные буферы, которые являюся общим высокоскоростным хранилищем ресурсов. Slurm обеспечивает поддержку выделения этих ресурсов, выделение файлов входных данных, бронирование вычислительных узлов для задачи, которая использует эти ресурсы и выделение файлов для выходных данных. Пакетные буферы также могут быть использованы как временное хранилище во время выполнения задачи, при этом не требуя выделения никаких файлов для записи и чтения. Другой типичный вариант использования - постоянное хранилище, не связанное с каким-либо конкретным заданием.

Lua

Это расширение обеспечивает обратные вызывы для API, определяемого Lua-скриптом. Этот плагин был разработан для того, чтобы предоставить системным администраторам возможность выполнять любую задачу (не только промежуточную обработку файлов) на разных этапах жизненного цикла задания. Эти задачи могут включать в себя промежуточную обработку файлов, обслуживание узла или любую другую задачу, которую желательно выполнять во время одного или нескольких из пяти состояний задачи, перечисленных выше.

API-интерфейсы пакетного буфера будут вызываться только для задания, которое специально запрашивает их использование. В разделе "Команды отправки задания" объясняется, как задание может быть запрошено с использованием API пакетного буфера.

Настройка (Для системных администраторов)

Общая конфигурция

- Чтобы включить плагин `burst buffer`, измените значение `Burst Buffer` в файле `slurm.conf`. Если значение пустое, то плагин `burst buffer` загружен не будет. Может быть указан только один модуль пакетного буфера.
- В `slurm.conf` вы можете установить `DEBUGFLAGS=Burst Buffer` для подробного ведения журнала с помощью плагина `burst buffer`. Эта приведет к очень подробному ведению журнала и не предназначена для длительного использования в производственной системе, но она может быть полезна для отладки.
- TRES пакетных буферов могут быть сконфигурированы с помощью ассоциации или QOS таким же образом, как количество отслеживаемых ресурсов может быть сконфигурировано для узлов, процессоров или любых GRES. Чтобы заставить Slurm отслеживать ресурсы пакетного буфера, добавьте `bb/datawarp` (для плагина datawarp) или `bb/lua` (для плагина lua) в секцию `AccountingStorageTres` файла `slurm.conf`.
- Требование к размеру задачи может использоваться в качестве фактора при установке приоритета задания, как описано в документе многофакторного приоритета. В разделе "Ресурсы пакетного буфера" объясняется, как определяются эти ресурсы.
- Конфигурации, специфичные для пакетного буфера, могут быть установлены в файле `burst_buffer.conf`. Параметры конфигурации включают в себя такие вещи, как то: какие пользователи могут использовать пакетные буферы, тайм-ауты, пути к пакетным буферным сценариям и т.д. Для получения подробной информции смотрите `man burst_buffer.conf`

- Для создания плагинов Slurm `burst_buffer/datawarp` и `burst_buffer/lua`, анализирующих данные в формате JSON должна быть установлена библиотека JSON-C. Подробности смотрите в информации об установке поддержки формата JSON в Slurm.

Datawrap

`slurm.conf`:

```
BurstBufferType=burst_buffer/datawarp
```

Datawrap плагин вызывает два скрипта:

- `dw_wlan_cli` - плагин Slurm `burst_buffer/datawarp` вызывает этот скрипт для выполнения функций пакетного буфера. Скрипт предоставляется компанией Cray. Местоположение этого скрипта определено параметром `Getsysstats` в файле `burst_buffer.conf`. Шаблон этого скрипта поставляется с Slurm: `src/plugins/burst_buffer/datawarp/dw_wlm_cli`
- dwstat - плагин Slurm burst_buffer/datawarp вызывает этот скрипт для получения информации о состоянии. Скрипт предоставляется компанией Cray. Местоположение этого скрипта определено параметром `Getsysstats` в файле `burst_buffer.conf`. Шаблон этого скрипта поставляется с Slurm: `src/plugins/burst_buffer/datawarp/dwstat`

LUA

`slurm.conf`:

```
BurstBufferType=burst_buffer/lua
```

Плагин lua вызывает один скрипт, который должен иметь имя `burst_buffer.lua`. Этот скрипт должен находиться в том же каталоге, что и файл `slurm.conf`. Для его работы необходимо существование следующих функций, которые могут не делать ничего, кроме возвращение успешного статуса:

- slurm_bb_job_process
- slurm_bb_pools
- slurm_bb_job_teardown
- slurm_bb_setup
- slurm_bb_data_in
- slurm_bb_real_size
- slurm_bb_paths
- slurm_bb_pre_run
- slurm_bb_post_run
- slurm_bb_data_out
- slurm_bb_get_status

Шаблон `burst_buffer.lua` создаётся Slurm'ом по пути `etc/burst_buffer.lua.example`

В этом шаблоне задокументировано гораздо больше подробностей о функциях, например: обязательные параметры, время вызова каждой функции, возвращаемые значения для каждой функции и несколько простых примеров.

Имплементация Lua

Цель этого раздела - предоставить дополнительную информацию о плагине Lua, чтобы помочь системным администраторам, желающим внедрить Lua API. Наиболее важными моментами в этом разделе являются:

- Некоторые функции в `burst_buffer.lua` должны выполняться быстро и не могут быть остановлены; остальным функциям разрешено выполняться столько времени, сколько необходимо, и они могут быть остановлены.
- Одновременно разрешается запускать не более 512 копий `burst_buffer.lua`, чтобы избежать превышения системных ограничений.

Как работает burst_buffer.lua

Lua-скрипты могут либо запускаться сами по себе в отдельном процессе с помощью системных вызовов `fork()` и `exec()`, либо они могут быть вызваны через C API Lua из существующего процесса. Одной из целей плагина lua было избежать вызова функции `fork()` из `slurmctld`, поскольку это может серьезно повлиять на производительность `slurmctld`. Плагин datawarp вызывает `fork()` и `exec()` из `slurmctld` для каждого вызова API пакетного буфера, и было показано, что это серьезно снижает производительность `slurmctld`. Следовательно, `slurmctld` вызывает `burst_buffer.lua` использует Lua's C API вместо использования `fork()`.

Некоторым функциям в `burst_buffer.lua` разрешено выполняться в течение длительного времени, но их, возможно, потребуется отключить, если задание отменено, если `slurmctld` перезапущен или если они выполняются дольше, чем заданный тайм-аут в burst_buffer.conf. Однако вызов Lua-скрипта через Lua C API не может быть остановлен внутри того же процесса; только завершение всего процесса, вызвавшего Lua-скрипт, может привести к завершению Lua-скрипта.

Чтобы устранить эту ситуацию, `burst_buffer.lua` вызывается двумя различными способами:

- Функции `slurm_bb_job_process`, `slurm_bb_pools` и `slurm_bb_paths` вызываются из `slurmctld`. Из-за приведенного выше объяснения скрипт, выполняющий одну из этих функций, не может быть уничтожен. Поскольку эти функции вызываются, когда `slurmctld` содержит некоторые мьютексы, их медленное выполнение будет крайне вредно для производительности `slurmctld` и быстродействия. Поскольку вызывать эти функции напрямую быстрее, чем вызывать `fork()` для создания нового процесса, это было сочтено приемлемым компромиссом. В результате эти функции не могут быть отключены.
- Остальные функции в `burst_buffer.lua` могут работать дольше без побочных эффектов. Их нужно уметь убивать. Эти функции вызываются из облегченного демона Slurm, называемого `slurm script d`. Всякий раз, когда требуется запустить одну из этих функций, `slurmctld` сообщает `slurm script d` запустить эту функцию; `slurm script d` затем вызывает `fork()` для создания нового процесса, а затем вызывает соответствующую функцию. Это позволяет избежать вызова `fork()` из `slurmctld`, в то же время предоставляя способ уничтожения запущенных копий `burst_buffer.lua` при необходимости. В результате эти функции могут быть отключены, и

они будут отключены, если будут выполняться дольше соответствующего значения тайм-аута, настроенного в `burst_buffer.configured`. Способ вызова каждой функции также задокументирован в файле `burst_buffer.lua.example`.

Ограничения

Каждая копия скрипта, запускаемая с помощью `slurm script d`, запускается в новом процессе, и канал (который представляет собой файл) открывается для чтения ответов из скрипта. Чтобы избежать превышения лимитов на открытый процесс или открытый файл, нехватки памяти или превышения других системных ограничений, для каждого "этапа" разрешается одновременный запуск не более 128 копий сценария, где этапами являются поэтапный ввод, предварительный запуск, поэтапный вывод и демонтаж (см. раздел Состояния пакетного буфера для получения дополнительной информации о стадиях пакетного буфера). Это означает, что одновременно может быть запущено максимум 512 копий `burst_buffer.lua`. Без этого ограничения пропускная способность задачи была ниже, и тестирование подтвердило, что лимит открытых файлов процесса мог быть превышен.

ПРЕДУПРЕЖДЕНИЕ: Не устанавливайте обработчик сигнала в `burst_buffer.lua`, потому что он вызывается непосредственно из `slurmctld`. Если `slurmctld` получит сигнал, он может попытаться запустить обработчик сигнала из `burst_buffer.lua` даже после завершения вызова `burst_buffer.lua`, что приведет к сбою.

Ресурсы пакетного буфера

API пакетного буфера может определять “пулы” ресурсов пакетного буфера, из которых задание может запрашивать определенный объем пространства пула. Если в пуле недостаточно места для выполнения запроса задания, это задание будет оставаться незавершенным до тех пор, пока в пуле не будет достаточно места. Как только в пуле будет достаточно места, Slurm может приступить к поэтапной подготовке к работе. Когда начинается этап ввода, Slurm вычитает запрошенное пространство задания из доступного пространства пула. Когда разборка завершается, Slurm добавляет запрошенное пространство задания обратно в доступное пространство пула. В разделе "Команды отправки задания" объясняется, как задание может запросить пространство из пула. Пространство пула - это скалярная величина.

Datawrap

- Пулы определяются `dw_wlm_cli` и представляют собой байты. Этот скрипт выводит строку в формате JSON, определяющую пулы, в стандартный вывод.
- Если задание не запрашивает пул, то будет использоваться пул, определенный по умолчанию в `burst_buffer.conf`. Если задание не запрашивает пул и пул по умолчанию не определен, то задание будет отклонено.

Lua

- Пулы в этом плагине необязательны и могут представлять все, что угодно.
- Пул по умолчанию в `burst_buffer.conf` в этом плагине не используется.
- Пулы определяются с помощью `burst_buffer.lua` в функции `slurm_bb_pools`. Если пулы нежелательны, то эта функция должна просто возвращать slurm.успех. Если требуются пулы, то

эта функция должна возвращать два значения: (1) slurm.SUCCESS, и (2) строка в формате JSON, определяющая пулы. Пример приведен в burst_buffer.lua.example. Текущими допустимыми полями в строке JSON являются:

- id - строка, определяющая имя пула
- quantity - число, определяющее объем пространства в пуле
- granularity (степень детализации) - число, определяющее наименьшее разрешение пространства, которое может быть выделено из этого пула. Если задание не запрашивает число, кратное степени детализации, то запрос задания будет округлен в большую сторону до ближайшего значения, кратного степени детализации. Например, если степень детализации равна 1000, то наименьший объем пространства, который может быть выделен из этого пула для одного задания, равен 1000. Если задание запрашивает менее 1000 единиц из этого пула, то запрос задания будет округлен до 1000.

Команды отправки заданий

Обычный режим работы заключается в том, что пакетные задания определяют требования к буферу пакетов в рамках пакетного сценария. Прокомментированные строки пакетного скрипта, содержащие определенную директиву (в зависимости от того, какой плагин используется), сообщат Slurm, что он должен запустить этапы пакетной буферизации для этого задания. В этих строках также будут описаны требования к пакетному буферу для задания.

Команды salloc и srun могут указывать требования к буферу пакетов с помощью параметров --bb и --bbf. Это описано в разделе Параметры задания командной строки.

Все директивы пакетного буфера должны быть указаны в комментариях в верхней части пакетного скрипта. Они могут быть размещены до, после или перемежаться с любыми директивами #SBATCH. Все этапы пакетного буферизации происходят в определенные моменты жизненного цикла задания, как описано в разделе "Обзор"; они не происходят во время выполнения задания. Например, все создания и удаления постоянного буфера пакетов (используемого только плагином datawarp) происходят до выполнения вычислительной части задания. Аналогичным образом, вы не можете запускать поэтапный ввод в различных точках выполнения скрипта; поэтапное включение пакетного буфера выполняется перед началом задания, а поэтапное отключение - после завершения задания.

Для обоих плагинов задание может запрашивать определенный объем пространства (размер или емкость) из пула ресурсов пакетного буфера.

- Спецификация пула - это просто строка, которая соответствует названию пула. Например: pool=pool1
- Спецификация емкости - это число, указывающее объем пространства, требуемый для пула. Спецификация емкости может включать суффикс "N" (узлы), "K|KiB", "M|MiB", "G|GiB", "T|TiB", "P|PiB" (для степеней 1024) и "KB", "MB", "ГБ", "ТБ", "ПБ" (для степеней 1000). ПРИМЕЧАНИЕ: Обычно Slurm интерпретирует единицы измерения KB, MB, GB, TB, PB как степени 1024, но для спецификаций размера пакетных буферов Slurm поддерживает оба формата IEC/SI. Это связано с тем, что CRAY API поддерживает оба формата.

При отправке задания Slurm выполняет базовую проверку директивы, а также запускает функцию в скрипте пакетного буферизации. Эта функция может выполнять проверку директив, используемых в

скрипте задания. Если Slurm определит, что параметры недопустимы, или если скрипт пакетного буфера вернет ошибку, задание будет отклонено, и сообщение об ошибке будет возвращено непосредственно пользователю.

Обратите внимание, что нераспознанные параметры могут игнорироваться для поддержки обратной совместимости (т.е. отправка задания не завершится неудачей в случае параметра, распознанного некоторыми версиями Slurm, но не распознанного другими версиями). Если задание принято, но позже завершается сбоем (например, некоторые промежуточные файлы проблем), задание будет отложено, а в поле "Причина" будет указано сообщение об ошибке, предоставленное базовой инфраструктурой.

Пользователи также могут запросить уведомление по электронной почте о завершении этапа буферизации пакетов, используя параметр `--mail-type=stage_out` или `--mail-type=all`. В теме электронного письма будет указана следующая форма:

```
SLURM Job_id=12 Name=my_app Staged Out, StageOut time 00:05:07
```

Следующие подразделы плагина содержат дополнительную информацию, специфичную для каждого плагина, и примеры сценариев заданий. Примеры командной строки приведены в разделе Параметры задания командной строки.

Datawrap

Директива `#DW` (для "DataWarp") используется для директив пакетного буфера при использовании `burst_buffer/data warpplugin`. Пожалуйста, обратитесь к документации Cray для получения подробной информации о параметрах DataWarp. Для систем DataWarp директива `#BB` может использоваться для создания или удаления постоянного буферного хранилища пакетов. ПРИМЕЧАНИЕ: Используется директива `#BB`, поскольку команда интерпретируется Slurm, а не программным обеспечением Cray Datawarp. Подробнее это обсуждается в разделе "Постоянный пакетный буфер".

Для пакетных буферов, специфичных для конкретного задания, требуется указать емкость пакетного буфера. Если в задании не указана пропускная способность, то задание будет отклонено. Задание также может указать пул, из которого ему требуются ресурсы; если в задании не указан пул, то будет использоваться пул, указанный DefaultPool в `burst_buffer.conf` (если он настроен).

Следующий сценарий задания запрашивает ресурсы пакетного буфера из пула по умолчанию и запрашивает файлы для поэтапного ввода и поэтапного вывода:

```
#!/bin/bash
#DW jobdw type=scratch capacity=1GB access_mode=striped,private pfs=/scratch
#DW stage_in type=file source=/tmp/a destination=/ss/file1
#DW stage_out type=file destination=/tmp/b source=/ss/file1
srun application.sh
```

Lua

Директива по умолчанию для этого плагина - `#BB_LUA`. Директива, используемая этим плагином, может быть изменена путем установки параметра `Directive` в `burst_buffer.conf`. Поскольку директива всегда должна начинаться со знака # (который является обозначением комментария в сценарии оболочки), этот параметр должен указывать только строку, следующую за знаком #. Например, если файл `burst_buffer.conf` содержит следующее:

```
Directive=BB_EXAMPLE
```

тогда директива пакетного буфера будет `#BB_EXAMPLE`.

Если параметр `Directive` не указан в `burst_buffer.conf`, то будет использоваться директива по умолчанию для этого плагина (#BB_LUA).

Поскольку этот плагин был разработан таким образом, чтобы быть универсальным и гибким, для этого плагина требуется только указать директиву. Если указана директива, Slurm запустит все этапы пакетного буферизации для задания.

Пример минимальной информации, необходимой для выполнения задания на всех этапах пакетного буферизации:

```
#!/bin/bash
#BB_LUA
srun application.sh
```

Поскольку пулы пакетных буферов являются необязательными для этого плагина (см. раздел Ресурсы пакетного буфера), задание не требуется для указания пула или емкости. Если пулы предоставляются API пакетного буфера, то задание может запросить пул и емкость:

```
#!/bin/bash
#BB_LUA pool=pool1 capacity=1K
srun application.sh
```

Задание может выбирать, указывать пул или нет. Если в задании не указан пул, то заданию по-прежнему разрешено выполняться, и этапы пакетного буферизации по-прежнему будут выполняться для этого задания (при условии, что была задана директива пакетного буферизации). Если в задании указан пул, но этот пул не найден, то задание отклоняется.

Системный администратор может проверить параметры пакетного буфера в функции `slurm_bb_job_process` в `burst_buffer.lua`. Это может включать требование задания указать пул или проверку любых дополнительных параметров, которые системный администратор решит реализовать.