Spalloc client

Release 1!7.0.0-a6

unknown

Jul 17, 2023

CONTENTS

Spalloc is a Python library and set of command-line programs for requesting SpiNNaker machines from a spalloc server.

CHAPTER

ONE

QUICK-START

Step 1: Install spalloc:

```
$ pip install spalloc
```

Step 2: Write a configuration file indicating your email address and the spalloc server's address (run `python -m spalloc.config` to discover what to call your config file on your machine):

```
[spalloc]
hostname = my_server
owner = jdh@cs.man.ac.uk
```

Step 3: Request a system using the command-line interface, e.g. a three-board machine:

```
$ spalloc 3
```

...or request one from Python...

```
>>> from spalloc_client import Job
>>> with Job(3) as j:
...     my_boot(j.hostname, j.width, j.height)
...     my_application(j.hostname)
```

Note: When a machine is allocated it is powered on but not booted: that is up to you. If Rig is installed on your system the `spalloc` commandline tool provides a `--boot` option which will boot the allocated machine for you.

Note: The dimensions of a machine may not be what you're used to and may change from allocation to allocation, even for the same number of boards.

Note: When you're finished with the boards you were allocated, pressing enter (or exiting the `with` block in the Python version) will automatically shut them down and allow them to be used by others.

CHAPTER

TWO

CONFIGURATION FILE FORMAT AND DEFAULTS

The spalloc command-line tool and Python library determine their default configuration options from a spalloc configuration file if present.

Note: Use of spalloc's configuration files is entirely optional as all configuration options may be presented as arguments to commands/methods at runtime.

By default, configuration files are read (in ascending order of priority) from a system-wide configuration directory (e.g. `/etc/xdg/spalloc`), user configuration file (e.g. `$HOME/.config/spalloc`) and finally the current working directory (in a file named `.spalloc`). The default search paths on your system can be discovered by running:

```
$ python -m spalloc.config
```

Config files use the Python `configparser` INI-format with a single section, `spalloc`, like so:

```
[spalloc]
hostname = localhost
owner = jdh@cs.man.ac.uk
```

Though most users will only wish to specify the `hostname` and `owner` options (as in the example above), the following enumerates the complete set of options available (and the default value).

hostname
: The hostname or IP address of the spalloc-server to connect to.

owner
: The name of the owner of created jobs. By convention the user's email address.

port
: The port used by the spalloc-server. (Default: 22244)

keepalive
: The keepalive interval, in seconds, to use when creating jobs. If the spalloc-server does not receive a keepalive command for this interval the job is automatically destroyed. May be set to None to disable this feature. (Default: 60.0)

reconnect_delay
: The time, in seconds, to wait between reconnection attempts to the server if disconnected. (Default 5.0)

timeout
: The time, in seconds, to wait before giving up waiting for a response from the server or None to wait forever. (Default 5.0)

machine
: The name of a specific machine on which to run all jobs or None to use any available machine. (Default: None)

`tags`
: The set of tags, comma seperated, to require a machine to have when allocating jobs. (Default: default)

`min_ratio`
: Require that when allocating a number of boards the allocation is at least as square as this aspect ratio. (Default: 0.333)

`max_dead_boards`
: The maximum number of dead boards which may be present in an allocated set of boards or None to allow any number of dead boards. (Default: 0)

`max_dead_links`
: The maximum number of dead links which may be present in an allocated set of boards or None to allow any number of dead links. (Default: None)

`require_torus`
: If True, require that an allocation have wrap-around links. This typically requires the allocation of a whole machine. If False, wrap-around links may or may-not be present in allocated machines. (Default: False)

CHAPTER

THREE

SPALLOC: ALLOCATE SPINNAKER MACHINES

A command-line utility for creating jobs.

Note: In the examples below, it is assumed that the spalloc server hostname and a suitable owner name have been specified in a `config` file.

3.1 Basic usage

By default, the `spalloc` command allocates a job according to the command-line specification and then waits for boards to be allocated and powered on.

Note: By default, allocated machines are powered on but not booted. If Rig is installed, `spalloc` provides a `--boot` option which also boots the allocated machine once it has been powered on. This dependency can be installed using the `[boot]` option at install time for spalloc.

The `spalloc` command can be called in one of the following styles though most users will probably only require the first two.

Invocation	Allocation
`spalloc`	A single SpiNN-5 board
`spalloc 5`	A machine with *at least* 5 boards
`spalloc 4 2`	A 4x2 *triad* machine.
`spalloc 3 4 0`	A single SpiNN-5 board at logical position (3, 4, 0)

A range of additional command-line arguments are available to control various aspects of Job allocation, run `spalloc --help` for a complete listing.

3.2 Wrapping other commands

The `spalloc` command can alternatively wrap an existing command, calling it once a SpiNNaker machine is allocated and cleaning up the job when the command exits:

```
$ spalloc 24 -c "rig-boot {} {w} {h} && python my_app.py {}"
```

The example above attempts to allocate a 24-board machine and, once allocated and powered on, calls the command above, with the arguments in curly braces substituted for details of the allocated machine.

The following substitutions are available:

Token	Substitution
`{}`	Chip 0, 0's hostname
`{hostname}`	Chip 0, 0's hostname
`{w}`	Width of the system (in chips)
`{width}`	Width of the system (in chips)
`{h}`	Height of the system (in chips)
`{height}`	Height of the system (in chips)
`{ethernet_ips}`	Filename of a CSV of Ethernet IPs
`{id}`	The job ID

3.3 Ethernet-connected chip hostname CSV Format

Hostnames for all Ethernet-connected SpiNNaker chips in a machine are provided in a CSV file with three columns:: x, y and hostname. The CSV file is newline (`\n`) delimited and the first row is a header.

3.4 Disconnecting and resuming jobs

Warning: This functionality is intended for advanced users only.

By default, when the `spalloc` command exits, the job will be destroyed and any allocated boards freed. This behaviour can be disabled with the `--no-destroy` argument, leaving the job allocated after the command exits.

Such a job may be 'resumed' by calling `spalloc` with the `--resume [JOB_ID]` option.

Note that by default, jobs require a 'keepalive' message to be sent to the server at a regular interval. While the `spalloc` command is running, these messages are sent automatically but after exiting the commands are no longer sent. Adding the `--keepalive -1` option when creating a job disables this.

CHAPTER

FOUR

SPALLOC-JOB: MANAGE AND RESET EXISTING JOBS AND THEIR BOARDS

Command-line administrative job management interface.

`spalloc-job` may be called with a job ID, or if no arguments supplied your currently running job is shown by default. Various actions may be taken and each is described below.

4.1 Displaying job information

By default, the command displays all known information about a job.

The `--watch` option may be added which will cause the output to be updated in real-time as a job's state changes. For example:

```
$ spalloc-job --watch
```

4.2 Controlling board power

The boards allocated to a job may be reset or powered on/off on demand (by anybody, at any time) by adding the `--power-on`, `--power-off` or `--reset` options. For example:

```
$ spalloc-job --reset
```

Note: This command blocks until the action is completed.

4.3 Listing board IP addresses

The hostnames of Ethernet-attached chips can be listed in CSV format by adding the –ethernet-ips argument:

```
$ spalloc-job --ethernet-ips
x,y,hostname
0,0,192.168.1.97
0,12,192.168.1.105
```

(continues on next page)

(continued from previous page)

```
4,8,192.168.1.129
4,20,192.168.1.137
8,4,192.168.1.161
8,16,192.168.1.169
```

4.4 Destroying/Cancelling Jobs

Jobs can be destroyed (by anybody, at any time) using the `--destroy` option which optionally accepts a human-readable explanation:

```
$ spalloc-job --destroy "Your job is taking too long..."
```

Warning: That this "super power" should be used carefully since the user may not be notified that their job was destroyed and the first sign of this will be their boards being powered down and re-partitioned ready for another user.

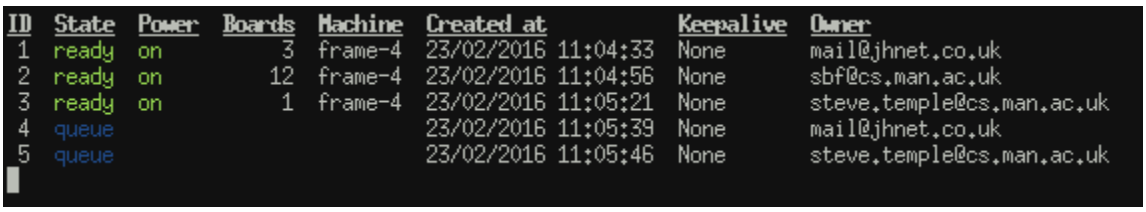
CHAPTER

FIVE

SPALLOC-PS: LIST ALL RUNNING JOBS

An administrative command-line process listing utility.

By default, the `spalloc-ps` command lists all running and queued jobs. For a real-time monitor of queued and running jobs, the `--watch` option may be added.



ID	State	Power	Boards	Machine	Created at	Keepalive	Owner
1	ready	on	3	frame-4	23/02/2016 11:04:33	None	mail@jhnet.co.uk
2	ready	on	12	frame-4	23/02/2016 11:04:56	None	sbf@cs.man.ac.uk
3	ready	on	1	frame-4	23/02/2016 11:05:21	None	steve.temple@cs.man.ac.uk
4	queue				23/02/2016 11:05:39	None	mail@jhnet.co.uk
5	queue				23/02/2016 11:05:46	None	steve.temple@cs.man.ac.uk

This list may be filtered by owner or machine with the `--owner` and `--machine` arguments.

CHAPTER

SIX

SPALLOC-MACHINE: LIST AVAILABLE MACHINES AND THEIR RUNNING JOBS

Command-line administrative machine management interface.

When called with no arguments the `spalloc-machine` command lists all available machines and a summary of their current load.

If a specific machine is given as an argument, the current allocation of jobs to machines is displayed:

Adding the `--detailed` option displays additional information about jobs running on a machine.

If the `--watch` option is given, the information displayed is updated in real-time.

CHAPTER

SEVEN

SPALLOC-WHERE-IS: QUERY THE SERVER FOR THE PHYSICAL/LOGICAL LOCATIONS OF BOARDS/CHIPS

Command-line tool to find out where a particular chip or board resides.

The `spalloc-where-is` command allows you to query boards by coordinate, by physical location, by chip or by job. In response to a query, a standard set of information is displayed as shown in the example below:

```
$ spalloc-where-is --job-chip 24 14, 3
                 Machine: my-machine
       Physical Location: Cabinet 2, Frame 4, Board 7
        Board Coordinate: (3, 4, 0)
Machine Chip Coordinates: (38, 51)
Coordinates within board: (2, 3)
         Job using board: 24
  Coordinates within job: (14, 3)
```

In this example we ask, 'where is chip (14, 3) in job 24'? We discover that:

- The chip is the machine named 'my-machine' on the board in cabinet 2, frame 4, board 7.
- This board's logical board coordinates are (3, 4, 0). These logical coordinates may be used to specifically request this board from Spalloc in the future.
- If 'my-machine' were booted as a single large machine, the chip we queried would be chip (38, 51). This may be useful for cross-referencing with diagrams produced by SpiNNer.
- The chip in question is chip (2, 3) its board. This may be useful when reporting faulty chips/replacing boards..
- The job currently running on the board has ID 24. Obviously in this example we already knew this but this may be useful when querying by board.
- Finally, we're told that the queried chip has the coordinates (14, 3) in the machine allocated to job 24. Again, this information may be more useful when querying by board.

To query by logical board coordinate:

```
spalloc-where-is --board MACHINE X Y Z
```

To query by physical board location:

```
spalloc-where-is --physical MACHINE CABINET FRAME BOARD
```

To query by chip coordinate (as if the machine were booted as one large machine):

```
spalloc-where-is --chip MACHINE X Y
```

To query by chip coordinate of chips allocated to a job:

```
spalloc-where-is --job-chip JOB_ID X Y
```

CHAPTER

EIGHT

PYTHON LIBRARY

Spalloc provides a pair of Python libraries which enable basic high- and low-level interaction with a spalloc server. The high-level *Job* interface makes the task of creating jobs (and keeping them alive) straight-forward but only facilitates basic job management functions such as resetting boards and getting their IP addresses. The low-level *ProtocolClient* provides an RPC-like interface to the spalloc server enabling any spalloc server command to be sent.

Note: These libraries are intentionally simplistic and may be unsuitable for very advanced applications. In such instances, users are encouraged to implement the spalloc server protocol in a manner better suited to their specific use-case.

8.1 High level interface (spalloc_client.Job)

class spalloc_client.**Job**(**args*, ***kwargs*)

A high-level interface for requesting and managing allocations of SpiNNaker boards.

Constructing a *Job* object connects to a spalloc-server and requests a number of SpiNNaker boards. See the *constructor* for details of the types of requests which may be made. The job object may then be used to monitor the state of the request, control the boards allocated and determine their IP addresses.

In its simplest form, a *Job* can be used as a context manager like so:

```
>>> from spalloc_client import Job
>>> with Job(6) as j:
...     my_boot(j.hostname, j.width, j.height)
...     my_application(j.hostname)
```

In this example a six-board machine is requested and the `with` context is entered once the allocation has been made and the allocated boards are fully powered on. When control leaves the block, the job is destroyed and the boards shut down by the server ready for another job.

For more fine-grained control, the same functionality is available via various methods:

```
>>> from spalloc_client import Job
>>> j = Job(6)
>>> j.wait_until_ready()
>>> my_boot(j.hostname, j.width, j.height)
>>> my_application(j.hostname)
>>> j.destroy()
```

Note: More complex applications may wish to log the following attributes of their job to support later debugging efforts:

- `job.id` – May be used to query the state of the job and find out its fate if cancelled or destroyed. The `spalloc-job` command can be used to discover the state/fate of the job and `spalloc-where-is` may be used to find out what boards problem chips reside on.
- `job.machine_name` and `job.boards` together give a complete record of the hardware used by the job. The `spalloc-where-is` command may be used to find out the physical locations of the boards used.

Job objects have the following attributes which describe the job and its allocated machines:

Attributes

job.id
[int or None] The job ID allocated by the server to the job.

job.state
[*JobState*] The current state of the job.

job.power
[bool or None] If boards have been allocated to the job, are they on (True) or off (False). None if no boards are allocated to the job.

job.reason
[str or None] If the job has been destroyed, gives the reason (which may be None), or None if the job has not been destroyed.

job.hostname
[str or None] The hostname of the SpiNNaker chip at (0, 0), or None if no boards have been allocated to the job.

job.connections
[{(x, y): hostname, ...} or None] The hostnames of all Ethernet-connected SpiNNaker chips, or None if no boards have been allocated to the job.

job.width
[int or None] The width of the SpiNNaker network in chips, or None if no boards have been allocated to the job.

job.height
[int or None] The height of the SpiNNaker network in chips, or None if no boards have been allocated to the job.

job.machine_name
[str or None] The name of the machine the boards are allocated in, or None if not yet allocated.

job.boards
[[[x, y, z], ...] or None] The logical coordinates allocated to the job, or None if not yet allocated.

__init__(**args*, ***kwargs*)

Request a SpiNNaker machine.

A *Job* is constructed in one of the following styles:

```
>>> # Any single (SpiNN-5) board
>>> Job()
>>> Job(1)
```

(continues on next page)

(continued from previous page)

```
>>> # Any machine with at least 4 boards
>>> Job(4)

>>> # Any 7-or-more board machine with an aspect ratio at least as
>>> # square as 1:2
>>> Job(7, min_ratio=0.5)

>>> # Any 4x5 triad segment of a machine (may or may-not be a
>>> # torus/full machine)
>>> Job(4, 5)

>>> # Any torus-connected (full machine) 4x2 machine
>>> Job(4, 2, require_torus=True)

>>> # Board x=3, y=2, z=1 on the machine named "m"
>>> Job(3, 2, 1, machine="m")

>>> # Keep using (and keeping-alive) an existing allocation
>>> Job(resume_job_id=123)
```

Once finished with a Job, the *destroy()* (or in unusual applications *Job.close()*) method must be called to destroy the job, close the connection to the server and terminate the background keep-alive thread. Alternatively, a Job may be used as a context manager which automatically calls *destroy()* on exiting the block:

```
>>> with Job() as j:
...     # ...for example...
...     my_boot(j.hostname, j.width, j.height)
...     my_application(j.hostname)
```

The following keyword-only parameters below are used both to specify the server details as well as the job requirements. Most parameters default to the values supplied in the local *config* file allowing usage as in the examples above.

Parameters

hostname
[str] **Required.** The name of the spalloc server to connect to. (Read from config file if not specified.)

port
[int] The port number of the spalloc server to connect to. (Read from config file if not specified.)

reconnect_delay
[float] Number of seconds between attempts to reconnect to the server. (Read from config file if not specified.)

timeout
[float or None] Timeout for waiting for replies from the server. If None, will keep trying forever. (Read from config file if not specified.)

config_filenames
[[str, ...]] If given must be a list of filenames to read configuration options from. If not

supplied, the default config file locations are searched. Set to an empty list to prevent using values from config files.

Other Parameters

resume_job_id
[int or None] If supplied, rather than creating a new job, take on an existing one, keeping it alive as required by the original job. If this argument is used, all other requirements are ignored.

owner
[str] **Required.** The name of the owner of the job. By convention this should be your email address. (Read from config file if not specified.)

keepalive
[float or None] The number of seconds after which the server may consider the job dead if this client cannot communicate with it. If None, no timeout will be used and the job will run until explicitly destroyed. Use with extreme caution. (Read from config file if not specified.)

machine
[str or None] Specify the name of a machine which this job must be executed on. If None, the first suitable machine available will be used, according to the tags selected below. Must be None when tags are given. (Read from config file if not specified.)

tags
[[str, ...] or None] The set of tags which any machine running this job must have. If None is supplied, only machines with the "default" tag will be used. If machine is given, this argument must be None. (Read from config file if not specified.)

min_ratio
[float] The aspect ratio (h/w) which the allocated region must be 'at least as square as'. Set to 0.0 for any allowable shape, 1.0 to be exactly square etc. Ignored when allocating single boards or specific rectangles of triads.

max_dead_boards
[int or None] The maximum number of broken or unreachable boards to allow in the allocated region. If None, any number of dead boards is permitted, as long as the board on the bottom-left corner is alive. (Read from config file if not specified.)

max_dead_links
[int or None] The maximum number of broken links allow in the allocated region. When require_torus is True this includes wrap-around links, otherwise peripheral links are not counted. If None, any number of broken links is allowed. (Read from config file if not specified.).

require_torus
[bool] If True, only allocate blocks with torus connectivity. In general this will only succeed for requests to allocate an entire machine. Must be False when allocating boards. (Read from config file if not specified.)

`__enter__()`

Convenience context manager for common case where a new job is to be created and then destroyed once some code has executed.

Waits for machine to be ready before the context enters and frees the allocation when the context exits.

Example:

```
>>> from spalloc_client import Job
>>> with Job(6) as j:
...     my_boot(j.hostname, j.width, j.height)
...     my_application(j.hostname)
```

destroy(*reason=None*)

Destroy the job and disconnect from the server.

Parameters

reason
[str or None] *Optional.* Gives a human-readable explanation for the destruction of the job.

close()

Disconnect from the server and stop keeping the job alive.

> **Warning:** This method does not free the resources allocated by the job but rather simply disconnects from the server and ceases sending keep-alive messages. Most applications should use *destroy()* instead.

set_power(*power*)

Turn the boards allocated to the job on or off.

Does nothing if the job has not yet been allocated any boards.

The *wait_until_ready()* method may be used to wait for the boards to fully turn on or off.

Parameters

power
[bool] True to power on the boards, False to power off. If the boards are already turned on, setting power to True will reset them.

reset()

Reset (power-cycle) the boards allocated to the job.

Does nothing if the job has not been allocated.

The *wait_until_ready()* method may be used to wait for the boards to fully turn on or off.

property state

The current state of the job.

property power

Are the boards powered/powering on or off?

property reason

For what reason was the job destroyed (if any and if destroyed).

property connections

The list of Ethernet connected chips and their IPs.

Returns

{(x, y): hostname, ...} or None

property hostname

The hostname of chip 0, 0 (or None if not allocated yet).

property width

The width of the allocated machine in chips (or None).

property height

The height of the allocated machine in chips (or None).

property machine_name

The name of the machine the job is allocated on (or None).

property boards

The coordinates of the boards allocated for the job (or None).

wait_for_state_change(*old_state*, *timeout=None*)

Block until the job's state changes from the supplied state.

Parameters

old_state
[*JobState*] The current state.

timeout
[float or None] The number of seconds to wait for a change before timing out. If None, wait forever.

Returns

JobState
The new state, or old state if timed out.

wait_until_ready(*timeout=None*)

Block until the job is allocated and ready.

Parameters

timeout
[float or None] The number of seconds to wait before timing out. If None, wait forever.

Raises

StateChangeTimeoutError
If the timeout expired before the ready state was entered.

JobDestroyedError
If the job was destroyed before becoming ready.

where_is_machine(*chip_x*, *chip_y*)

Locates and returns cabinet, frame, board for a given chip in a machine allocated to this job.

Parameters

- **chip_x** – chip x location
- **chip_y** – chip y location

Returns
tuple of (cabinet, frame, board)

__weakref__

list of weak references to the object (if defined)

class spalloc_client.**JobState**(*value*)

All the possible states that a job may be in.

unknown = 0

The job ID requested was not recognised.

queued = 1

The job is waiting in a queue for a suitable machine.

power = 2

The boards allocated to the job are currently being powered on or powered off.

ready = 3

The job has been allocated boards and the boards are not currently powering on or powering off.

destroyed = 4

The job has been destroyed.

exception spalloc_client.**JobDestroyedError**

Thrown when the job was destroyed while waiting for it to become ready.

exception spalloc_client.**StateChangeTimeoutError**

Thrown when a state change takes too long to occur.

8.2 Lower level interface (spalloc_client.ProtocolClient)

class spalloc_client.**ProtocolClient**(*hostname*, *port=22244*, *timeout=None*)

A simple (blocking) client implementation of the spalloc-server protocol.

This minimal implementation is intended to serve both simple applications and as an example implementation of the protocol for other applications. This implementation simply implements the protocol, presenting an RPC-like interface to the server. For a higher-level interface built on top of this client, see `spalloc.Job`.

Usage examples:

```
# Connect to a spalloc_server
with ProtocolClient("hostname") as c:
    # Call commands by name
    print(c.call("version"))  # '0.1.0'

    # Call commands as if they were methods
    print(c.version())  # '0.1.0'

    # Wait an event to be received
    print(c.wait_for_notification())  # {"jobs_changed": [1, 3]}

# Done!
```

__init__(*hostname*, *port=22244*, *timeout=None*)

Define a new connection.

> **Note:** Does not connect to the server until *connect()* is called.

Parameters

hostname
[str] The hostname of the server.

port
[str or int] The port to use (default: 22244).

`connect`(*timeout=None*)

(Re)connect to the server.

Raises

OSError, IOError
If a connection failure occurs.

`close`()

Disconnect from the server.

`call`(*name*, **args*, ***kwargs*)

Send a command to the server and return the reply.

Parameters

name
[str] The name of the command to send.

timeout
[float or None] The number of seconds to wait before timing out or None if this function should wait forever. (Default: None)

Returns

object
The object returned by the server.

Raises

ProtocolTimeoutError
If a timeout occurs.

ProtocolError
If the connection is unavailable or is closed.

`wait_for_notification`(*timeout=None*)

Return the next notification to arrive.

Parameters

name
[str] The name of the command to send.

timeout
[float or None] The number of seconds to wait before timing out or None if this function should try again forever.

If negative only responses already-received will be returned. If no responses are available, in this case the function does not raise a ProtocolTimeoutError but returns None instead.

Returns

object
The notification sent by the server.

Raises

ProtocolTimeoutError
If a timeout occurs.

ProtocolError
If the socket is unusable or becomes disconnected.

__weakref__

list of weak references to the object (if defined)

exception spalloc_client.**ProtocolTimeoutError**

Thrown upon a protocol-level timeout.

CHAPTER

NINE

INDICIES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

Symbols

B

C

D

H

J

M

P

Q

R

S

U

W