

CFD EXPERTS

Simulate the Future

WWW.CFDEXPERTS.NET



©2021 ANSYS, Inc.
All Rights Reserved.
Unauthorized use, distribution
or duplication is prohibited.

Running Fluent Using a Load Manager



ANSYS, Inc.
Southpointe
2600 Ansys Drive
Canonsburg, PA 15317
ansysinfo@ansys.com
<http://www.ansys.com>
(T) 724-746-3304
(F) 724-514-9494

Release 2021 R2
July 2021

ANSYS, Inc. and
Ansys Europe,
Ltd. are UL
registered ISO
9001:2015
companies.

Copyright and Trademark Information

© 2021 ANSYS, Inc. Unauthorized use, distribution or duplication is prohibited.

Ansys, Ansys Workbench, AUTODYN, CFX, FLUENT and any and all ANSYS, Inc. brand, product, service and feature names, logos and slogans are registered trademarks or trademarks of ANSYS, Inc. or its subsidiaries located in the United States or other countries. ICEM CFD is a trademark used by ANSYS, Inc. under license. CFX is a trademark of Sony Corporation in Japan. All other brand, product, service and feature names or trademarks are the property of their respective owners. FLEXIm and FLEXnet are trademarks of Flexera Software LLC.

Disclaimer Notice

THIS ANSYS SOFTWARE PRODUCT AND PROGRAM DOCUMENTATION INCLUDE TRADE SECRETS AND ARE CONFIDENTIAL AND PROPRIETARY PRODUCTS OF ANSYS, INC., ITS SUBSIDIARIES, OR LICENSORS. The software products and documentation are furnished by ANSYS, Inc., its subsidiaries, or affiliates under a software license agreement that contains provisions concerning non-disclosure, copying, length and nature of use, compliance with exporting laws, warranties, disclaimers, limitations of liability, and remedies, and other provisions. The software products and documentation may be used, disclosed, transferred, or copied only in accordance with the terms and conditions of that software license agreement.

ANSYS, Inc. and Ansys Europe, Ltd. are UL registered ISO 9001: 2015 companies.

U.S. Government Rights

For U.S. Government users, except as specifically granted by the ANSYS, Inc. software license agreement, the use, duplication, or disclosure by the United States Government is subject to restrictions stated in the ANSYS, Inc. software license agreement and FAR 12.212 (for non-DOD licenses).

Third-Party Software

See the [legal information](#) in the product help files for the complete Legal Notice for Ansys proprietary software and third-party software. If you are unable to access the Legal Notice, contact ANSYS, Inc.

Published in the U.S.A.

Table of Contents

| | |
|--|-------|
| 1. Running Fluent Under LSF | 7 |
| About This Part | ix |
| 1. Introduction | 11 |
| 1.1. Overview of Fluent and LSF Integration | 12 |
| 1.1.1. Requirements | 12 |
| 1.1.2. Optional Requirements | 12 |
| 1.1.3. Integration Components | 12 |
| 2. Checkpointing and Restarting | 13 |
| 2.1. Fluent Checkpoint Files | 13 |
| 2.2. Checkpoint Directories | 13 |
| 2.3. Checkpoint Trigger Files | 13 |
| 2.4. Restart Jobs | 14 |
| 3. Configuring LSF for Fluent | 15 |
| 4. Working with Fluent Jobs | 17 |
| 4.1. Submitting a Fluent Job from the Command Line | 17 |
| 4.2. Submitting a Fluent Job Using Fluent Launcher | 18 |
| 4.3. Using Custom LSF Scripts | 20 |
| 4.4. Manually Checkpointing Fluent Jobs | 21 |
| 4.5. Restarting Fluent Jobs | 21 |
| 4.6. Migrating Fluent Jobs | 21 |
| 4.7. Coupling LSF Job Submissions and ANSYS Licensing | 22 |
| 5. Fluent and LSF Examples | 25 |
| 5.1. Examples Without Checkpointing | 25 |
| 5.2. Examples with Checkpointing | 25 |
| 2. Running Fluent Under PBS Professional | 29 |
| About This Document | xxxi |
| 1. Introduction | 33 |
| 1.1. Overview of Running Fluent Jobs with PBS Professional | 33 |
| 1.1.1. Requirements | 33 |
| 1.1.2. Fluent and PBS Professional Communication | 33 |
| 2. Running Fluent Simulation under PBS Professional | 35 |
| 2.1. Using the Integrated PBS Professional Capability | 35 |
| 2.1.1. Overview | 35 |
| 2.1.2. Usage | 35 |
| 2.1.2.1. Submitting a Fluent Job from the Command Line | 36 |
| 2.1.2.2. Submitting a Fluent Job Using Fluent Launcher | 37 |
| 2.1.3. Examples | 39 |
| 2.1.4. Limitations | 40 |
| 2.2. Using Your Own Supplied Job Script | 40 |
| 2.3. Using Altair's Sample Script | 40 |
| 2.3.1. Configuration File Example | 41 |
| 2.3.2. Altair's Sample Script | 41 |
| 2.3.3. Submitting Altair's Sample Script | 42 |
| 2.3.4. Epilogue/Prologue | 43 |
| 2.4. Monitoring/Manipulating Jobs | 43 |
| 2.4.1. Monitoring the Progress of a Job | 43 |
| 2.4.2. Removing a Job from the Queue | 43 |
| 3. Running Fluent Under SGE | 45 |
| About This Document | xlvii |

- 1. Introduction** 49
 - 1.1. Overview of Fluent and SGE Integration 49
 - 1.1.1. Requirements 49
 - 1.1.2. Fluent and SGE Communication 50
 - 1.1.3. Checkpointing Directories 50
 - 1.1.4. Checkpointing Trigger Files 50
 - 1.1.5. Default File Location 50
- 2. Configuring SGE for Fluent** 51
 - 2.1. General Configuration 51
 - 2.2. Checkpoint Configuration 51
 - 2.3. Configuring Parallel Environments 52
 - 2.4. Default Request File 53
- 3. Running a Fluent Simulation under SGE** 55
 - 3.1. Submitting a Fluent Job from the Command Line 55
 - 3.2. Submitting a Fluent Job Using Fluent Launcher 57
 - 3.3. Using Custom SGE Scripts 60
- 4. Running Fluent Utility Scripts under SGE** 61
- 4. Running Fluent Under Slurm** 63
 - About This Document lxxv
 - 1. Introduction** 67
 - 1.1. Requirements for Running Fluent Jobs with Slurm 67
 - 2. Running Fluent Simulation under Slurm** 69
 - 2.1. Using the Integrated Slurm Capability 69
 - 2.1.1. Overview 69
 - 2.1.2. Usage 69
 - 2.1.2.1. Submitting a Fluent Job from the Command Line 70
 - 2.1.2.2. Submitting a Fluent Job Using Fluent Launcher 71
 - 2.1.3. Examples 73
 - 2.1.4. Limitations 74
 - 2.2. Using Your Own Supplied Job Script 74
 - 2.3. Epilogue/Prologue 74
 - 2.4. Monitoring/Manipulating Jobs 74
 - 2.4.1. Monitoring the Progress of a Job 75
 - 2.4.2. Removing a Job from the Queue 75

List of Figures

- 4.1. The Scheduler Tab of Fluent Launcher (Linux Version) 19
- 2.1. The Scheduler Tab of Fluent Launcher (Linux Version) 38
- 3.1. The **Scheduler** Tab of Fluent Launcher (Linux Version) 58
- 2.1. The Scheduler Tab of Fluent Launcher (Linux Version) 72

Part 1: Running Fluent Under LSF

About This Document (p. ix)

1. Introduction (p. 11)

2. Checkpointing and Restarting (p. 13)

3. Configuring LSF for Ansys Fluent (p. 15)

4. Working with Ansys Fluent Jobs (p. 17)

5. Ansys Fluent and LSF Examples (p. 25)

About This Part

This part provides general information about running Fluent under LSF. Examples have also been included, where available.

Information in this part is presented in the following chapters:

- [Introduction \(p. 11\)](#)
- [Checkpointing and Restarting \(p. 13\)](#)
- [Configuring LSF for Fluent \(p. 15\)](#)
- [Working with Fluent Jobs \(p. 17\)](#)
- [Fluent and LSF Examples \(p. 25\)](#)

This document is made available via the Ansys, Inc. website for your convenience. Contact Platform Computing Inc. (<http://www.platform.com/>) directly for support of their product.



Chapter 1: Introduction

Platform Computing's LSF software is a distributed computing resource management tool that you can use with either the serial or the parallel version of Ansys Fluent. This document provides general information about running Fluent under LSF, and is made available via the Ansys, Inc. website for your convenience. Contact Platform directly for support of their product.

Using LSF, Fluent simulations can take full advantage of LSF checkpointing (saving Fluent `.cas` and `.dat` files) and migration features. LSF is also integrated with various MPI communication libraries for distributed MPI processing, increasing the efficiency of the software and data processing.

Important:

Running Fluent under LSF is not supported on Windows.

Platform's Standard Edition is the foundation for all LSF products, it offers users load sharing and batch scheduling across distributed Linux and Windows computing environments. Platform's LSF Standard Edition provides the following functionality:

- comprehensive distributed resource management
 - provides dynamic load sharing services
 - allows for batch scheduling and resource management policies
- flexible queues and sharing control
 - prioritizes jobs
 - schedules jobs with load conditions
 - processes jobs with time windows
 - provides limits on the number of running jobs and job resource consumption
- fair-share scheduling of limited resources
 - manages shares for users and user groups
 - ensures fair sharing of limited computing resources
- maximum fault tolerance
 - provides batch service as long as one computer is active
 - ensures that no job is lost when the entire network goes down

- restarts jobs on other compute nodes when a computer goes down

For more information, see the following section:

[1.1. Overview of Fluent and LSF Integration](#)

1.1. Overview of Fluent and LSF Integration

For more information, see the following sections:

[1.1.1. Requirements](#)

[1.1.2. Optional Requirements](#)

[1.1.3. Integration Components](#)

1.1.1. Requirements

- LSF 10.1, available from Platform Computing at <http://www.platform.com/>
- Fluent 2021 R2

1.1.2. Optional Requirements

- The `echkpnt.fluent` and `erestart.fluent` binary files

These files are available from Platform Computing or Ansys, Inc., and permit Fluent checkpointing and restarting from within LSF.

- (For Linux) Hardware vendor-supplied MPI environment for network computing

1.1.3. Integration Components

The LSF components used by the Fluent integration are included in all versions of LSF packages that are able to be downloaded from the Platform ftp site (<ftp.platform.com>). If you are a current LSF customers, contact Platform support personnel for downloading instructions at support@platform.com. New LSF customers should contact the Platform sales department.

Chapter 2: Checkpointing and Restarting

LSF provides utilities to save (that is, checkpoint), and restart an application. The Fluent and LSF integration allows Fluent to take advantage of the checkpoint and restart features of LSF. At the end of each iteration, Fluent looks for the existence of a checkpoint or checkpoint-exit file. If Fluent detects the checkpoint file, it writes a case and data file, removes the checkpoint file, and continues iterating. If Fluent detects the checkpoint-exit file, it writes a case file and data file, then exits. LSF's `bchkpnt` utility can be used to create the checkpoint and checkpoint-exit files, thereby forcing Fluent to checkpoint itself, or checkpoint and terminate itself. In addition to writing a case file and data file, Fluent also creates a simple journal file with instructions to read the checkpointed case file and data file, and continues iterating. Fluent uses that journal file when restarted with LSF's `brstart` utility. For more details on checkpointing features and options within Fluent, see the Fluent [User's Guide](#).

The greatest benefit of the checkpoint facilities occurs when it is used on an automatic basis. By starting jobs with a periodic checkpoint, LSF automatically restarts any jobs that are lost due to host failure from the last checkpoint. This facility can dramatically reduce lost compute time, and also avoids the task of manually restarting failed jobs.

For more information, see the following sections:

- [2.1. Fluent Checkpoint Files](#)
- [2.2. Checkpoint Directories](#)
- [2.3. Checkpoint Trigger Files](#)
- [2.4. Restart Jobs](#)

2.1. Fluent Checkpoint Files

In order to allow you to checkpoint Fluent jobs using LSF, LSF supplies special versions of `echkpnt` and `erestart`. These Fluent checkpoint files are called `echkpnt.fluent` and `erestart.fluent`.

2.2. Checkpoint Directories

When you submit a checkpointing job, you specify a checkpoint directory. Before the job starts running, LSF sets the environment variable `LSB_CHKPNT_DIR`. The value of `LSB_CHKPNT_DIR` is a subdirectory of the checkpoint directory specified in the command line. This subdirectory is identified by the job ID and only contains files related to the submitted job.

2.3. Checkpoint Trigger Files

When you checkpoint a Fluent job, LSF creates a checkpoint trigger file (`check`) in the job subdirectory, which causes Fluent to checkpoint and continue running. A special option is used to create a different trigger file (`exit`), to cause Fluent to checkpoint and exit the job. Fluent uses the `LSB_CHKPNT_DIR` environment variable to determine the location of checkpoint trigger files. It checks the job subdirectory

periodically while running the job. Fluent does not perform any checkpointing unless it finds the LSF trigger file in the job subdirectory. Fluent removes the trigger file after checkpointing the job.

2.4. Restart Jobs

If a job is restarted, LSF attempts to restart the job with the `-restart` option appended to the original `fluent` command. Fluent uses the checkpointed data and case files to restart the process from that checkpoint point, rather than repeating the entire process.

Each time a job is restarted, it is assigned a new job ID, and a new job subdirectory is created in the checkpoint directory. Files in the checkpoint directory are never deleted by LSF, but you may choose to remove old files once the Fluent job is finished and the job history is no longer required.

Chapter 3: Configuring LSF for Fluent

LSF provides special versions of `echkpnt` and `erestart` called `echkpnt.fluent` and `erestart.fluent` to allow checkpointing with Fluent. You must make sure LSF uses these files instead of the standard versions.

To configure LSF 9.1.1 for Fluent:

- Copy the `echkpnt.fluent` and `erestart.fluent` files to the `$LSF_SERVERDIR` for each architecture that is desired.
- When submitting the job from the command line, include the `-a fluent` parameter when specifying the checkpoint information (see [Submitting a Fluent Job from the Command Line \(p. 17\)](#) for details).

Important:

Note that LSF includes an email notification utility that sends email notices to users when an LSF job has been completed. If a user submits a batch job to LSF and the email notification utility is enabled, LSF will distribute an email containing the output for the particular LSF job. When a Fluent job is run under LSF with the `-g` option, the email will also contain information from the Fluent console.

Chapter 4: Working with Fluent Jobs

Information in this chapter is provided in the following sections:

- 4.1. Submitting a Fluent Job from the Command Line
- 4.2. Submitting a Fluent Job Using Fluent Launcher
- 4.3. Using Custom LSF Scripts
- 4.4. Manually Checkpointing Fluent Jobs
- 4.5. Restarting Fluent Jobs
- 4.6. Migrating Fluent Jobs
- 4.7. Coupling LSF Job Submissions and ANSYS Licensing

4.1. Submitting a Fluent Job from the Command Line

The following shows how to submit a Fluent job from the Linux command line using LSF:

```
fluent <solver_version> [<Fluent_options>] -scheduler=lsf [-scheduler_queue=<queue>]
[-scheduler_opt=<opt>] [-gui_machine=<hostname>] [-scheduler_headnode=<head-
node>] [-scheduler_stderr=<err-file>] [-scheduler_stdout=<out-file>]
```

where

- `fluent` is the command that launches Fluent.
- `<solver_version>` specifies the dimensionality of the problem and the precision of the Fluent calculation (for example, `3d`, `2ddp`).
- `<Fluent_options>` can be added to specify the startup option(s) for Fluent, including the options for running Fluent in parallel. For more information, see the Fluent [User's Guide](#).
- `-scheduler=lsf` is added to the Fluent command to specify that you are running under LSF. This option causes Fluent to check for trigger files in the checkpoint directory if the environment variable `LSB_CHKPNT_DIR` is set.
- `-scheduler_queue=<queue>` sets the queue to `<queue>`.
- `-scheduler_opt=<opt>` enables an additional option `<opt>` that is relevant for LSF; see the LSF documentation for details. Note that you can include multiple instances of this option when you want to use more than one scheduler option.
- `-gui_machine=<hostname>` specifies that Cortex is run on a machine named `<hostname>`. This option may be necessary to avoid poor graphics performance when running Fluent under LSF.
- `-scheduler_headnode=<head-node>` allows you to specify that the scheduler job submission machine is `<head-node>` (the default is `localhost`).

- `-scheduler_stderr=<err-file>` sets the name / directory of the scheduler standard error file to `<err-file>`; by default it is saved as `fluent.<PID>.e` in the working directory, where `<PID>` is the process ID of the top-level Fluent startup script.
- `-scheduler_stdout=<out-file>` sets the name / directory of the scheduler standard output file to `<out-file>`; by default it is saved as `fluent.<PID>.o` in the working directory, where `<PID>` is the process ID of the top-level Fluent startup script.

Note that tight integration between LSF and the MPI is enabled by default for Intel MPI (the default) and Open MPI without requiring an additional option. Tight integration is not supported with the `-gui_machine=<hostname>` option.

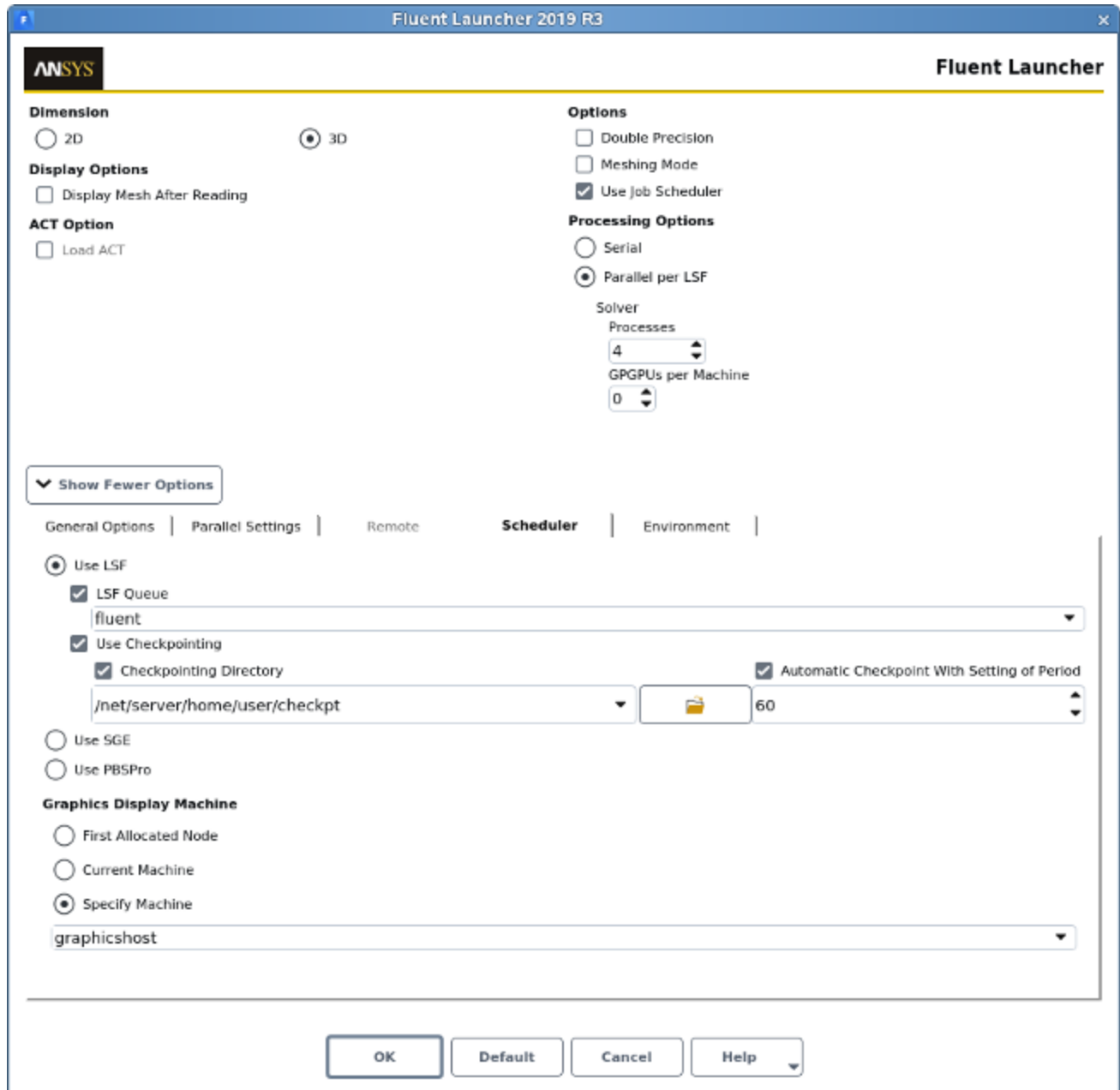
Important:

You must have the `DISPLAY` environment variable properly defined, otherwise the graphical user interface (GUI) will not operate correctly.

4.2. Submitting a Fluent Job Using Fluent Launcher

Fluent Launcher has graphical user input options that allow you to submit a Fluent job using LSF. Perform the following steps:

1. Open Fluent Launcher ([Figure 4.1: The Scheduler Tab of Fluent Launcher \(Linux Version\) \(p. 19\)](#)) by entering `fluent` without any arguments in the Linux command line.

Figure 4.1: The Scheduler Tab of Fluent Launcher (Linux Version)

2. Enable **Use Job Scheduler** under **General Options** tab.
3. Click the **Show More Options** button to expand Fluent Launcher.
4. Select the **Scheduler** tab.
 - a. Ensure that **Use LSF** is selected.
 - b. To specify a job queue, enable the **LSF queue** option and enter the queue name in the text box.
 - c. Enable the **Use Checkpointing** option to utilize LSF checkpointing. By default, the checkpointing directory will be the current working directory; you have the option of enabling **Checkpointing**

Directory and specifying a different directory, either by entering the name in the text box or by browsing to it.

You can specify that the checkpointing is done automatically at a set time interval by enabling the **Automatic Checkpoint with Setting of Period** option and entering the period (in minutes) in the text box; otherwise, checkpointing will not occur unless you call the `bchkpnt` command.

d. If you experience poor graphics performance when using LSF, you may be able to improve performance by changing the machine on which Cortex (the process that manages the graphical user interface and graphics) is running. The **Graphics Display Machine** list provides the following options:

- Select **First Allocated Node** if you want Cortex to run on the same machine as that used for compute node 0.

When the graphics display machine is set to the **First Allocated Node**, note that tight integration between LSF and the MPI is enabled by default for Intel MPI (the default) and Open MPI.

- Select **Current Machine** if you want Cortex to run on the same machine used to start Fluent Launcher.
- Select **Specify Machine** if you want Cortex to run on a specified machine, which you select from the drop-down list below.

5. Set up the other aspects of your Fluent simulation using the Fluent Launcher GUI items. For more information, see the Fluent [User's Guide](#).

Important:

You must have the `DISPLAY` environment variable properly defined, otherwise the graphical user interface (GUI) will not operate correctly.

Note:

Submitting your Fluent job from the command line provides some options that are not available in Fluent Launcher, such as specifying the scheduler job submission machine name or setting the scheduler standard error file or standard output file. For details, see [Submitting a Fluent Job from the Command Line](#) (p. 17).

4.3. Using Custom LSF Scripts

If you use custom LSF scripts instead of relying on the standard Fluent option (either the `-scheduler=lsf` option from the command line or the **Use LSF** option in Fluent Launcher, as described in the preceding sections), your environment variables related to the job scheduler will not be used unless you include the `-scheduler_custom_script` option with the Fluent options in your script.

4.4. Manually Checkpointing Fluent Jobs

You can checkpoint a batch job manually by using the `bchkpnt` command. Note that you can set up automatic checkpointing using Fluent Launcher as described previously. The syntax for the `bchkpnt` command is

```
bchkpnt [bchkpnt_options] [-k] <job_ID>
```

where

- *bchkpnt_options* are options for the job checkpointing. See the Platform LSF Reference guide for a complete list with descriptions.
- `-k` is the regular option to the `bchkpnt` command, and specifies checkpoint and exit. The job will be killed immediately after being checkpointed. When the job is restarted, it does not have to repeat any operations.
- *job_ID* is the job ID of the Fluent job, which is used to specify which job to checkpoint.

4.5. Restarting Fluent Jobs

Restarting a batch job requires the `brestart` command. The syntax for the `brestart` command is

```
brestart [bsub_options] <checkpoint_directory> <job_ID>
```

where

- *bsub_options* are options for the job restart. See the Platform LSF Reference guide for a complete list with descriptions.
- *checkpoint_directory* specifies the checkpoint directory where the job subdirectory is located.
- *job_ID* is the job ID of the Fluent job, and specifies which job to restart. At this point, the restarted job is assigned a new job ID, and the new job ID is used for checkpointing. The job ID changes each time the job is restarted.

4.6. Migrating Fluent Jobs

Migrating a Fluent job requires the `bmig` command. The syntax for the `bmig` command is

```
bmig [bmig_options] <job_ID>
```

where

- *bmig_options* are options for the job migration. See the Platform LSF Reference guide for a complete list with descriptions.
- *job_ID* is the job ID of the Fluent job, and specifies which job to checkpoint and restart on the migration target. At this point, the restarted job is assigned a new job ID, and the new job ID is used for checkpointing. The job ID changes each time the job is restarted.

4.7. Coupling LSF Job Submissions and ANSYS Licensing

You can couple your LSF job submission with **ANSYS** licensing by revising the LSF configuration files. Such a coupling will ensure that submitted jobs do not fail because licenses are not available. The jobs will be held in the queue until licenses are available. To set up such coupling, perform the following steps:

1. Copy the `elim` script from your **ANSYS** installation area to `$LSF_SERVERDIR`. The `elim` script is located in the following directory:

```
<path>/ansys_inc/v212/fluent/fluent21.2.0/multiport/mpi_wrapper/bin/
```

where `<path>` is the directory in which you have installed Fluent (for example, `/opt/apps/`).

2. Edit the copy of the `elim` script to add your license server and license feature details. The following is an example where `acfd_fluent`, `acfd_par_proc`, and `anshpc` are **ANSYS** solver and parallel license features:

```
$ENV{'ANSYSLMD_LICENSE_FILE'} = "1055@deval2"
my @features = qw(acfd_fluent acfd_par_proc anshpc);
```

3. Set the permissions to 755 and set `root` as the owner for the `elim` script.
4. Add all of your **ANSYS** solver license feature names and **ANSYS** parallel/HPC license feature names under `Resource` section in the file `lsf.shared`, which is located in `$LSF_ENVDIR` in your LSF installation area. The following is an example in which `acfd`, `acfd_fluent`, `acfd_solver`, and `acfd_fluent_solver` are the **ANSYS** solver license feature names and `anshpc`, `anshpc_pack`, and `acfd_par_proc` are the **ANSYS** parallel/HPC license feature names.

```
acfd Numeric 20 N (available ANSYS Fluent Solver licenses)
acfd_fluent Numeric 20 N (available ANSYS Fluent Solver licenses)
acfd_solver Numeric 20 N (available ANSYS Fluent Solver licenses)
acfd_fluent_solver Numeric 20 N (available ANSYS Fluent Solver licenses)
anshpc Numeric 20 N (available ANSYS Fluent Parallel licenses)
anshpc_pack Numeric 20 N (available ANSYS Fluent Parallel licenses)
acfd_par_proc Numeric 20 N (available ANSYS Fluent Parallel licenses)
```

5. Add all of your **ANSYS** solver license feature names and **ANSYS** parallel/HPC license feature names in the file `lsf.cluster.<cluster_name>` (where `<cluster_name>` is the name of the cluster), which is located in `$LSF_ENVDIR` in your LSF installation area. The following is an example in which `acfd`, `acfd_fluent`, `acfd_solver`, and `acfd_fluent_solver` are the **ANSYS** solver license feature names and `anshpc`, `anshpc_pack`, and `acfd_par_proc` are the **ANSYS** parallel/HPC license feature names.

```
# For LSF-ANSYS Licensing Coupling
Begin ResourceMap
RESOURCENAME LOCATION
acfd ([all])
acfd_fluent ([all])
acfd_solver ([all])
acfd_fluent_solver ([all])
anshpc ([all])
anshpc_pack ([all])
acfd_par_proc ([all])
End ResourceMap
```

6. Reconfigure the LSF daemons using the following commands, to specify that they reread their configuration. Note that you need administrator privileges to implement these changes.

```
lsadmin reconfig
```

```
badmin reconfig
```

7. Submit the Fluent job using the "`select[<solver_license_feature>] rusage[<parallel_license_feature>]`" option, as described in [Submitting a Fluent Job from the Command Line](#) (p. 17). In this option, `<solver_license_feature>` has the following form:

```
<serial_lic_name>>0
```

where `<serial_lic_name>` is the name of the serial **ANSYS** solver license feature name. Similarly, `<parallel_license_feature>` has the following form:

```
<parallel_lic_name>=<N>
```

where `<parallel_lic_name>` is the name of the **ANSYS** parallel/HPC license feature name, and `<N>` is the number of processes to use.

The previous descriptions are applicable when you have a single serial and/or parallel license feature. If you have multiple serial and/or parallel license features, you must add additional `<solver_license_feature>` and/or `<parallel_license_feature>` entries, separating them with `||`; additionally, you must enclose all of the `<solver_license_feature>` entries in a single pair of parentheses. The following is an example of submitting a Fluent job in which `acfd` and `acfd_fluent` are the **ANSYS** solver license feature names, `anshpc` and `acfd_par_proc` are the **ANSYS** parallel/HPC license feature names, and the number of processes to use is 4:

```
fluent 3d -t4 -scheduler=lsf -scheduler_opt='-R "select[(acfd>0 ||  
acfd_fluent>0)] rusage[anshpc=4 || acfd_par_proc=4]"'
```

Chapter 5: Fluent and LSF Examples

This chapter provides various examples of running Fluent and LSF.

[5.1. Examples Without Checkpointing](#)

[5.2. Examples with Checkpointing](#)

5.1. Examples Without Checkpointing

- Serial 3D Fluent interactive job under LSF

```
fluent 3d -scheduler=lsf
```

- Serial 3D Fluent batch job under LSF, which reads the journal file called `journal_file`

```
fluent 3d -g -i journal_file -scheduler=lsf
```

- Parallel 3D Fluent interactive job under LSF, on 4 CPUs

```
fluent 3d -t4 -scheduler=lsf
```

Important:

PAM is an extension of LSF that manages parallel processes by choosing the appropriate compute nodes and launching child processes. When using Fluent on Linux, PAM is not used to launch Fluent (so the `JOB_STARTER` argument of the LSF queue should not be set). Instead, LSF will set an environment variable that contains a list of `N` hosts, and Fluent will use this list to launch itself.

- Parallel 3D Fluent batch job under LSF, which uses 5 processes and reads the journal file called `journal_file`

```
fluent 3d -t5 -g -i journal_file -scheduler=lsf
```

5.2. Examples with Checkpointing

The examples that follow apply to both interactive and batch submissions. For brevity, only batch submissions are described. Usage of the LSF checkpoint and restart capabilities, requiring `echkpt` and `erestart`, are described as follows:

- Serial 3D Fluent batch job under LSF with checkpoint/restart

```
fluent 3d -g -i journal_file -scheduler=lsf -scheduler_opt='-k "  
/home/username 60"' -scheduler_opt='-a fluent'
```

- In this example, the LSF `-a fluent` specification identifies which `echkpnt/erestart` combination to use, `/home/username` is the checkpoint directory, and the duration between automatic checkpoints is 60 minutes.

The following commands can then be used:

- `bjobs -l <job_ID>`

→ This command returns the job information about `<job_ID>` in the LSF system.

- `bchkpnt <job_ID>`

→ This command forces Fluent to write a case file, a data file, and a restart journal file at the end of its current iteration.

→ The files are saved in a directory named `<checkpoint_directory>/<job_ID>`. The `<checkpoint_directory>` is defined through the `-scheduler_opt=` option in the original `fluent` command.

→ Fluent then continues to iterate.

- `bchkpnt -k <job_ID>`

→ This command forces Fluent to write a case file, a data file, and a restart journal file at the end of its current iteration.

→ The files are saved in a directory named `<checkpoint_directory>/<job_ID>` and then Fluent exits. The `<checkpoint_directory>` is defined through the `-scheduler_opt=` option in the original `fluent` command.

- `brestart <checkpoint_directory> <job_ID>`

→ This command starts a Fluent job using the latest case and data files in the `<checkpoint_directory>/<job_ID>` directory.

→ The restart journal file `<checkpoint_directory>/<job_ID>/#restart.inp` is used to instruct Fluent to read the latest case and data files in that directory and continue iterating.

- Parallel 3D Fluent batch job under LSF with checkpoint/restart, which specifies `/home/username` as the checkpoint directory, uses 4 processes, and reads a journal file called `journal_file`

```
fluent 3d -t4 -g -i journal_file -scheduler=lsf -scheduler_opt='-k "/home/username"' -scheduler_opt='-a fluent'
```

The following commands can then be used:

- `bjobs -l <job_ID>`

→ This command returns the job information about `<job_ID>` in the LSF system.

- `bchkpnt <job_ID>`

→ This command forces parallel Fluent to write a case file, a data file, and a restart journal file at the end of its current iteration.

- The files are saved in a directory named `<checkpoint_directory>/<job_ID>`. The `<checkpoint_directory>` is defined through the `-scheduler_opt=` option in the original `fluent` command.
- Parallel Fluent then continues to iterate.
- `bchkpnt -k <job_ID>`
 - This command forces parallel Fluent to write a case file, a data file, and a restart journal file at the end of its current iteration.
 - The files are saved in a directory named `<checkpoint_directory>/<job_ID>`. The `<checkpoint_directory>` is defined through the `-scheduler_opt=` option in the original `fluent` command.
 - Parallel Fluent then exits.
- `brestart <checkpoint_directory> <job_ID>`
 - This command starts a Fluent network parallel job using the latest case and data files in the `<checkpoint_directory>/<job_ID>` directory.
 - The restart journal file `<checkpoint_directory>/<job_ID>/#restart.inp` is used to instruct Fluent to read the latest case and data files in that directory and continue iterating.
 - The parallel job will be restarted using the same number of processes as that specified through the `-t<x>` option in the original `fluent` command (4 in the previous example).
- `bmig -m <host> 0`
 - This command checkpoints all jobs (indicated by 0 job ID) for the current user and moves them to host `<host>`.

Part 2: Running Fluent Under PBS Professional

About This Document (p. xxxi)

1. Introduction (p. 33)

2. Running an Ansys Fluent Simulation under PBS Professional (p. 35)

About This Document

This document provides general information about running Ansys Fluent under PBS Professional. Examples have also been included, where available.

Information in this document is presented in the following chapters:

- [Introduction \(p. 33\)](#)
- [Running Fluent Simulation under PBS Professional \(p. 35\)](#)

This document is made available via the Ansys, Inc. website for your convenience. Contact Altair Engineering, Inc. (<http://www.altair.com/>) directly for support of their product.

Chapter 1: Introduction

Altair PBS Professional is an open workload management tool for local and distributed environments. You can use PBS Professional when running Fluent simulations, and thereby control the number of jobs running and dynamically monitor the licenses. This document provides general information about running Fluent under PBS Professional, and is made available via the Ansys, Inc. website for your convenience. Contact Altair Engineering, Inc. (<http://www.altair.com/>) directly for support of their product.

For more information, see the following section:

[1.1. Overview of Running Fluent Jobs with PBS Professional](#)

1.1. Overview of Running Fluent Jobs with PBS Professional

For more information, see the following sections:

[1.1.1. Requirements](#)

[1.1.2. Fluent and PBS Professional Communication](#)

1.1.1. Requirements

- Standard
 - PBS Professional 18.2
 - Fluent 2021 R2

Important:

Running Fluent under PBS Professional is not supported on Windows.

- Optional
 - Checkpoint restart scripts must be obtained/written to use the checkpoint restart functionality

1.1.2. Fluent and PBS Professional Communication

- PBS Professional has the ability to send signals to the Fluent processes inside a PBS Professional job.
- Dynamic resource functionality provides a way to monitor the availability of software licenses. Refer to the PBS Professional Administrators Guide for more information on creating custom resources.

Chapter 2: Running Fluent Simulation under PBS Professional

For more information, see the following sections:

- [2.1. Using the Integrated PBS Professional Capability](#)
- [2.2. Using Your Own Supplied Job Script](#)
- [2.3. Using Altair's Sample Script](#)
- [2.4. Monitoring/Manipulating Jobs](#)

2.1. Using the Integrated PBS Professional Capability

For more information, see the following sections:

- [2.1.1. Overview](#)
- [2.1.2. Usage](#)
- [2.1.3. Examples](#)
- [2.1.4. Limitations](#)

2.1.1. Overview

One option of using PBS Professional with Fluent is to use the PBS Professional launching capability integrated directly into Fluent. In this mode, Fluent is started from the command line with the additional `-scheduler=pbs` argument. Fluent then takes responsibility for relaunching itself under PBS Professional. This has the following advantages:

- The command line usage is very similar to the non-RMS (resource management system) usage.
- You do not need to write a separate script.

The integrated PBS Professional capability is intended to simplify usage for the most common situations. If you desire more control over the PBS Professional `qsub` options for more complex situations or systems (or if you are using an older version of Fluent), you can always write or adapt a PBS Professional script that starts Fluent in the desired manner (see [Using Your Own Supplied Job Script \(p. 40\)](#) and [Using Altair's Sample Script \(p. 40\)](#) for details).

2.1.2. Usage

Information on usage is provided in the following sections:

- [2.1.2.1. Submitting a Fluent Job from the Command Line](#)
- [2.1.2.2. Submitting a Fluent Job Using Fluent Launcher](#)

2.1.2.1. Submitting a Fluent Job from the Command Line

The integrated PBS Professional capability can be activated by simply adding the `-scheduler=pbs` option when launching Fluent from the command line:

```
fluent <solver_version> [<Fluent_options>] -i <journal_file> -scheduler=pbs [-scheduler_queue=<queue>] [-scheduler_opt=<opt>] [-gui_machine=<hostname>] [-scheduler_tight_coupling]] [-scheduler_headnode=<head-node>] [-scheduler_stderr=<err-file>] [-scheduler_stdout=<out-file>]
```

where

- `fluent` is the command that launches Fluent.
- `<solver_version>` specifies the dimensionality of the problem and the precision of the Fluent calculation (for example, 3d, 2ddp).
- `<Fluent_options>` can be added to specify the startup option(s) for Fluent, including the options for running Fluent in parallel. For more information, see the Fluent [User's Guide](#).
- `-i <journal_file>` reads the specified journal file(s).
- `-scheduler=pbs` is added to the Fluent command to specify that you are running under PBS Professional.
- `-scheduler_queue=<queue>` sets the queue to `<queue>`.
- `-scheduler_opt=<opt>` enables an additional option `<opt>` that is relevant for PBS Professional; see the PBS Professional documentation for details. Note that you can include multiple instances of this option when you want to use more than one scheduler option.
- `-gui_machine=<hostname>` specifies that Cortex is run on a machine named `<hostname>`. This option may be necessary to avoid poor graphics performance when running Fluent under PBS Professional.
- `-scheduler_headnode=<head-node>` allows you to specify that the scheduler job submission machine is `<head-node>` (the default is `localhost`).
- `-scheduler_stderr=<err-file>` sets the name / directory of the scheduler standard error file to `<err-file>`; by default it is saved as `fluent.<PID>.e` in the working directory, where `<PID>` is the process ID of the top-level Fluent startup script.
- `-scheduler_stdout=<out-file>` sets the name / directory of the scheduler standard output file to `<out-file>`; by default it is saved as `fluent.<PID>.o` in the working directory, where `<PID>` is the process ID of the top-level Fluent startup script.
- `-scheduler_tight_coupling` enables tight integration between PBS Professional and the MPI. It is supported with Intel MPI (the default). This option is not supported with the `-gui_machine=<hostname>` option.

This syntax will start the Fluent job under PBS Professional using the `qsub` command in a batch manner. When resources are available, PBS Professional will start the job and return a job ID, usually in the form of `<job_ID>.<hostname>`. This job ID can then be used to query, control, or stop the

job using standard PBS Professional commands, such as `qstat` or `qdel`. The job will be run out of the current working directory.

Important:

You must have the `DISPLAY` environment variable properly defined, otherwise the graphical user interface (GUI) will not operate correctly.

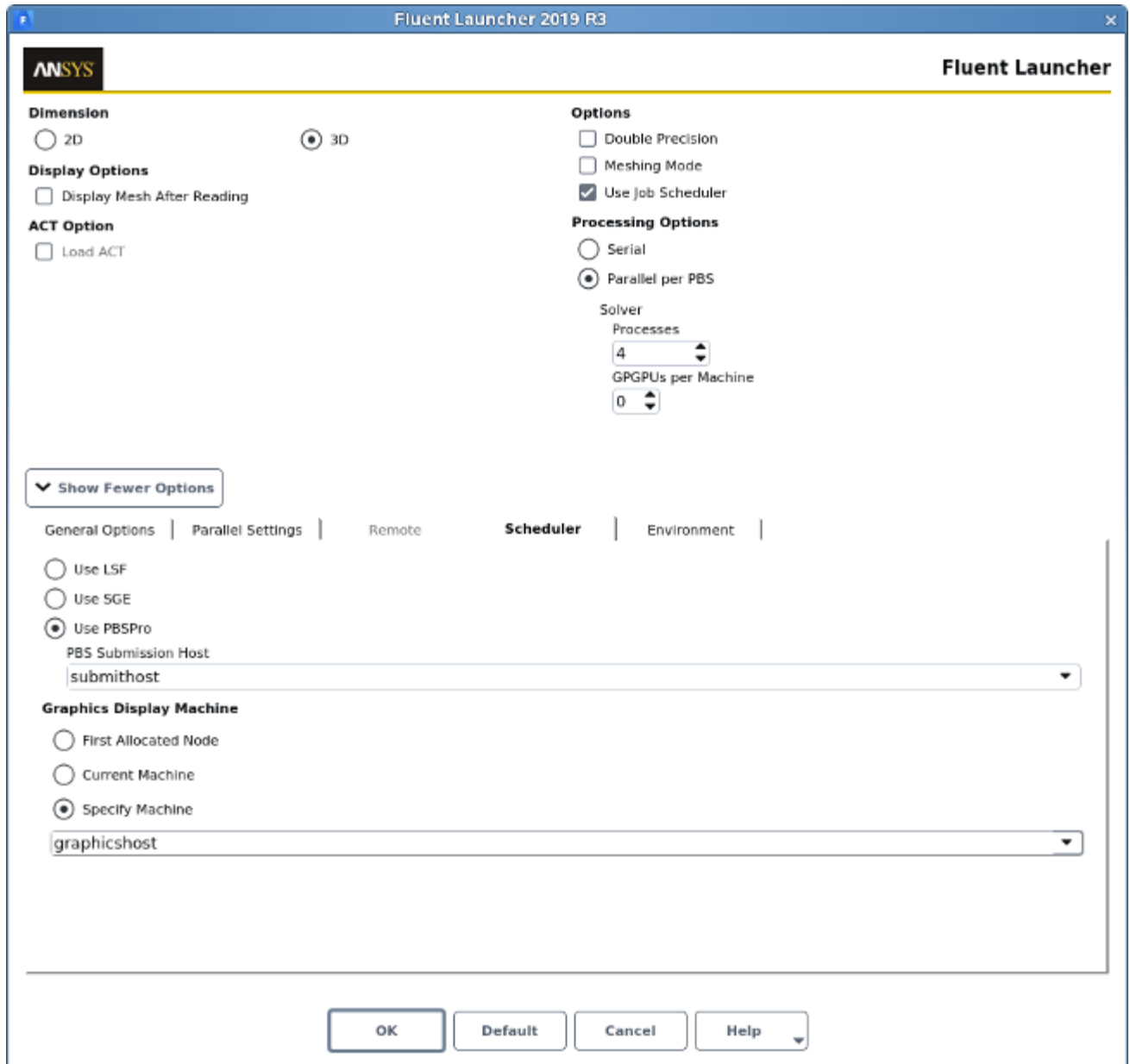
Note:

If you experience poor graphics performance when running Fluent under PBS Professional, you may be able to improve it by running Cortex on a specified machine through the use of the `-gui_machine=<hostname>` option.

2.1.2.2. Submitting a Fluent Job Using Fluent Launcher

Fluent Launcher has graphical user input options that allow you to submit a Fluent job using PBS Professional. Perform the following steps:

1. Open Fluent Launcher ([Figure 2.1: The Scheduler Tab of Fluent Launcher \(Linux Version\) \(p. 38\)](#)) by entering `fluent` without any arguments in the Linux command line.

Figure 2.1: The Scheduler Tab of Fluent Launcher (Linux Version)

2. Enable **Use Job Scheduler** under **General Options** tab.
3. Click the **Show More Options** button to expand Fluent Launcher.
4. Select the **Scheduler** tab.
 - a. Select **Use PBSPro**.
 - b. You can choose to make a selection from the **PBS Submission Host** drop-down list to specify the PBS Pro submission host name for submitting the job, if the machine you are using to run the launcher cannot submit jobs to PBS Pro.

- c. If you experience poor graphics performance when using PBS Professional, you may be able to improve performance by changing the machine on which Cortex (the process that manages the graphical user interface and graphics) is running. The **Graphics Display Machine** list provides the following options:

- Select **First Allocated Node** if you want Cortex to run on the same machine as that used for compute node 0.

Note that it is possible to enable a job-scheduler-supported native remote node access mechanism when the graphics display machine is set to the **First Allocated Node**. To do so, enable the **Tight Coupling** option. This tight integration is supported with Intel MPI (the default).

- Select **Current Machine** if you want Cortex to run on the same machine used to start Fluent Launcher.
 - Select **Specify Machine** if you want Cortex to run on a specified machine, which you select from the drop-down list below.
5. Set up the other aspects of your Fluent simulation using the Fluent Launcher GUI items. For more information, see the Fluent [User's Guide](#).

Important:

You must have the `DISPLAY` environment variable properly defined, otherwise the graphical user interface (GUI) will not operate correctly.

Note:

Submitting your Fluent job from the command line provides some options that are not available in Fluent Launcher, such as specifying the scheduler job submission machine name or setting the scheduler standard error file or standard output file. For details, see [Submitting a Fluent Job from the Command Line \(p. 36\)](#).

2.1.3. Examples

- Submit a parallel, 4-process job using a journal file `f15s3.jou`:

```
> fluent 3d -t4 -i f15s3.jou -scheduler=pbs
Relaunching fluent under PBSPro
134.les29
```

In the previous example, note that `134.les29` is returned from `qsub`. `134` is the job ID, while `les29` is the hostname on which the job was started.

- Check the status of the job:

```
> qstat -s

les29:
Job ID      Username Queue      Jobname      SessID NDS TSK  Req'd  Req'd  Elap
-----  -----  -----  -----  -----  ---  ---  ---    ---    ---
          Memory Time  S Time
```



```

134.les29      user1      workq      fluent      11958      4      4      --      --      R 00:00
Job run at Thu Jan 04 at 14:48

> qstat -f 134
Job Id: 134.les29
  Job_Name = fluent
  Job_Owner = user1@les29
<... additional status of jobID 134>

```

The first command in the previous example lists all of the jobs in the queue. The second command lists the detailed status about the given job.

After the job is complete, the job will no longer show up in the output of the `qstat` command. The results of the run will then be available in the scheduler standard output file.

2.1.4. Limitations

The integrated PBS Professional capability in Fluent 2021 R2 has the following limitations:

- The PBS Professional commands (such as `qsub`) must be in the users path.
- For parallel jobs, Fluent processes are placed on available compute resources using the `qsub` options `-l select=<N>`. (Optionally, for better control over the placement of processes, you can set the environment variable `SET_PBS_PPN` to a value of `<M>`, where `<M>` is the number of processes per node. Setting this variable will place Fluent processes using the option `-l nodes=<total_processes_requested>/<M>:ppn=<M>`. The value used for `<M>` has to be same for all nodes. If you desire more sophisticated placement, you may write separate PBS Professional scripts as described in [Using Your Own Supplied Job Script \(p. 40\)](#).)
- RMS-specific checkpointing is not available. PBS Professional only supports checkpointing via mechanisms that are specific to the operating system on SGI and Cray systems. The integrated PBS Professional capability is based on saving the process state, and is not based on the standard application restart files (for example, Fluent case and data files) on which the LSF and SGE checkpointing is based. Thus, if you need to checkpoint, you should checkpoint your jobs by periodically saving the Fluent data file via the journal file.

2.2. Using Your Own Supplied Job Script

PBS Professional can accept any custom shell script for running Fluent that you may already use. Note that this shell script may require modification to move files between machines in the cluster. The shell script should also endeavor to clean up after itself when execution completes/fails.

If you use custom PBS Professional scripts instead of relying on the standard Fluent option (either the `-scheduler=pbs` option from the command line or the **Use PBSPro** option in Fluent Launcher, as described in the preceding sections), your environment variables related to the job scheduler will not be used unless you include the `-scheduler_custom_script` option with the Fluent options in your script.

2.3. Using Altair's Sample Script

Altair provides a short sample script for running Fluent inside of PBS Professional. This script uses a configuration file to tell it several pieces of information that it needs (outlined in the section that follows).

Optionally, these pieces of information can be specified on the command line. The `fluent_args` option does not need to be used for specifying the number of CPUs in the default script. The script will address this for the user, based on their PBS Professional resource requests.

For more information, see the following sections:

[2.3.1. Configuration File Example](#)

[2.3.2. Altair's Sample Script](#)

[2.3.3. Submitting Altair's Sample Script](#)

[2.3.4. Epilogue/Prologue](#)

2.3.1. Configuration File Example

The sample script uses a configuration file called `pbs_fluent.conf` if no command line arguments are present. This configuration file should be present in the directory from which the jobs are submitted (which is also the directory in which the jobs are executed). The following is an example of what the content of `pbs_fluent.conf` can be:

```
input="example_small.flin"
case="Small-1.65m.cas"
fluent_args="3d -pinfiniband"
outfile="fluent_test.out"
mpp="true"
```

The following is an explanation of the parameters:

input

is the name of the input file.

case

is the name of the `.cas` file that the input file will utilize.

fluent_args

are extra Fluent arguments. As shown in the previous example, you can specify the interconnect by using the `-p<interconnect>` command. The available interconnects include `ethernet` (the default), `infiniband`, and `crayx`. The MPI is selected automatically, based on the specified interconnect.

outfile

is the name of the file to which the standard output will be sent.

mpp="true"

will tell the job script to execute the job across multiple processors.

2.3.2. Altair's Sample Script

Altair's Sample script is not intended to be a full solution for running Fluent jobs in PBS Professional, but rather a simple starting point. It runs the jobs out of the directory from which they are submitted

(PBS_O_WORKDIR). Care should be taken to submit jobs from locations that are going to be available on any node (perhaps via NFS).

```
#!/bin/sh
cd $PBS_O_WORKDIR

#We assume that if they didn't specify arguments then they should use the
#config file if [ "xx${input}${case}${mpp}${fluent_args}zz" = "xxzz" ]; then
if [ -f pbs_fluent.conf ]; then
    . pbs_fluent.conf
else
    printf "No command line arguments specified, "
    printf "and no configuration file found. Exiting \n"
fi
fi

#Set up the license information (Note: you need to substitute your own
#port and server in this command)
export AnsysLMD_License_FILE

#Augment the Fluent command line arguments case "$mpp" in
true)
    #MPI job execution scenario
    num_nodes='cat $PBS_NODEFILE | sort -u | wc -l'
    cpus='expr $num_nodes \* $NCPUS'
    #Default arguments for mpp jobs, these should be changed to suit your
    #needs.
    fluent_args="-t${cpus} $fluent_args -cnf=$PBS_NODEFILE"
    ;;
*)
    #SMP case
    #Default arguments for smp jobs, should be adjusted to suit your
    #needs.
    fluent_args="-t$NCPUS $fluent_args"
    ;;
esac
#Default arguments for all jobs
fluent_args="-ssh -g -i $input $fluent_args"

echo "----- Going to start a fluent job with the following settings:
Input: $input
Case: $case
Output: $outfile
Fluent arguments: $fluent_args"

#run the solver
/ansys_inc/v212/fluent/bin/fluent $fluent_args > $outfile
```

Note that for versions of Fluent prior to 12.0, the final line of the sample script should be changed to the following:

```
usr/apps/Fluent.Inc/bin/fluent $fluent_args > $outfile
```

2.3.3. Submitting Altair's Sample Script

- To submit the script with command line arguments:
 - `qsub -l <resource_requests> -v input=<input_file>,case=<case_file>,fluent_args=<fluent_arguments>,outfile=<output_file>[,mpp="true"] fluent-job.sh`
- To submit the script without command line arguments:
 - Edit `fluent_pbs.conf` to suit your needs

```
- qsub -l <resource_requests> fluent-job.sh
```

Note that the resources necessary for the job (that is, <resource_requests>) should be entered with the proper syntax. For more information about requesting resources, see the PBS Professional 7.1 Users Guide Section 4.3.

2.3.4. Epilogue/Prologue

PBS Professional provides the ability to script some actions immediately before the job starts and immediately after it ends (even if it is removed). The epilogue is a good place to put any best effort cleanup and stray process functionality that might be necessary to clean up after a job. For instance, one could include functionality to clean up the working/scratch area for a job if it is deleted before its completion. Fluent provides a way to get rid of job-related processes in the form of a shell script in the work directory (`cleanup-fluent-<host-pid>`). It could be useful to have the epilogue call this script to ensure all errant processes are cleaned up after a job completes.

2.4. Monitoring/Manipulating Jobs

For more information, see the following sections:

[2.4.1. Monitoring the Progress of a Job](#)

[2.4.2. Removing a Job from the Queue](#)

2.4.1. Monitoring the Progress of a Job

The `qstat` command is used to monitor the status of jobs in a PBS Professional queue. When no job ID is specified, the `qstat` command will display all jobs. If you use `qstat -an job_ID` you will receive information about a specific job. This information includes the location at which the job is executing, which will be listed under the `Job ID` column.

```
% qstat -an
server:
Job ID      Username Queue   Jobname   SessID NDS  TSK  Req'd  Req'd  Elap
-----
32.server  user    workq   fluent-job 15886  --   1    --    --   R 01:07
server/0
```

There are several job states to be aware of when using `qstat`, which are listed under the `S` column. The two main states you will see are `Q` and `R`. `Q` indicates that the job is waiting in the queue to run. At this point the scheduler has not found a suitable node or nodes to run the job. Once the scheduler has found a suitable area to run the job and has sent the job to run, its state will be set to `R`. Full details on the different job statuses reported by `qstat` can be found in the PBS Professional 7.1 Users Guide Section 6.1.1.

2.4.2. Removing a Job from the Queue

The `qdel <job_ID>` command will delete the job indicated by <job_ID>.

Part 3: Running Fluent Under SGE

About This Document (p. xlvii)

1. Introduction (p. 49)

2. Configuring SGE for Ansys Fluent (p. 51)

3. Running an Ansys Fluent Simulation under SGE (p. 55)

4. Running an Ansys Fluent Utility Scripts under SGE (p. 61)

About This Document

This document provides general information about running Fluent under Univa Grid Engine (UGE) and Sun Grid Engine software (SGE). Throughout this manual SGE is used to denote the Grid Engine software whether UGE or SGE. Examples are included, where available.

Information in this document is presented in the following chapters:

- [Introduction \(p. 49\)](#)
- [Configuring SGE for Fluent \(p. 51\)](#)
- [Running a Fluent Simulation under SGE \(p. 55\)](#)
- [Running Fluent Utility Scripts under SGE \(p. 61\)](#)

This document is made available via the Ansys, Inc. website for your convenience. Contact Oracle, Inc. (<http://www.oracle.com/>) directly for support of their product.

Chapter 1: Introduction

Univa (formerly Sun) Grid Engine (UGE/SGE) software is a distributed computing resource management tool that you can use with either the serial or the parallel version of Fluent. Throughout this manual SGE is used to denote the Grid Engine software whether UGE or SGE. This document provides general information about running Fluent under SGE, and is made available via the Ansys, Inc. website for your convenience. Contact Univa (<http://www.univa.com/>) directly for support of their product.

Fluent submits a process to the SGE software, then SGE selects the most suitable machine to process the Fluent simulation. You can configure SGE and select the criteria by which SGE determines the most suitable machine for the Fluent simulation.

Among many other features, running a Fluent simulation using SGE enables you to:

- Save the current status of the job (this is also known as checkpointing when the Fluent `.cas` and `.dat` files are saved)
- Migrate the simulation to another machine
- Restart the simulation on the same or another machine.

For more information, see the following section:

[1.1. Overview of Fluent and SGE Integration](#)

1.1. Overview of Fluent and SGE Integration

For more information, see the following sections:

[1.1.1. Requirements](#)

[1.1.2. Fluent and SGE Communication](#)

[1.1.3. Checkpointing Directories](#)

[1.1.4. Checkpointing Trigger Files](#)

[1.1.5. Default File Location](#)

1.1.1. Requirements

- Univa (Sun) Grid Engine software version 8.6, which is available online at <http://www.univa.com/>
- Fluent 2021 R2

Important:

Running Fluent under SGE is not supported on Windows.

1.1.2. Fluent and SGE Communication

Fluent and SGE communicate with each other through checkpointing and migration commands. To checkpoint, or save, Fluent simulations, SGE uses an executable file called `ckpt_command.fluent`. To migrate Fluent simulations to another machine, SGE uses another executable file called `migr_command.fluent`.

1.1.3. Checkpointing Directories

Fluent creates a checkpointing subdirectory, identified by the job ID. The checkpointing directory contains files related only to the submitted job.

1.1.4. Checkpointing Trigger Files

When a Fluent simulation must be checkpointed, SGE calls a command that creates a checkpoint trigger file (`check`) in the job subdirectory, which causes Fluent to checkpoint and continue running. If the job must be migrated, because of a machine crash or for some other reason, a different trigger file (`exit`) is created which causes Fluent to checkpoint and exit.

1.1.5. Default File Location

For Fluent 2021 R2, SGE-related files are installed by default in `path/ansys_inc/v212/fluent/fluent21.2.0/addons/sge`, where `path` is the Fluent installation directory. The files include the following:

- `ckpt_command.fluent`
- `migr_command.fluent`
- `sge_request`
- `kill-fluent`
- `sample_ckpt_obj`
- `sample_pe`

These files are described in the sections that follow.

Chapter 2: Configuring SGE for Fluent

SGE must be installed properly if checkpointing is needed or parallel Fluent is being run under SGE. The checkpoint queues must be configured first, and they must be configured by someone with manager or root privileges. The configuration can be performed either through the GUI `qmon` or the text command `qconf`.

For more information, see the following sections:

- 2.1. General Configuration
- 2.2. Checkpoint Configuration
- 2.3. Configuring Parallel Environments
- 2.4. Default Request File

2.1. General Configuration

Using the SGE graphical interface, the general configuration of SGE and Fluent requires the following:

- The **Shell Start Mode** must be set to either **unix_behavior** or **posix_compliant**.
- Under **Type**, the **checkpointing** option must be marked as true.

When running parallel Fluent simulations, the following options are also important:

- Under **Type**, the **parallel** option must be marked as true.
- The value of **slots** should be set to a value greater than 1.

2.2. Checkpoint Configuration

Checkpointing can be configured using the **min_cpu_interval** field. This field specifies the time interval between checkpoints. The value of **min_cpu_interval** should be a reasonable amount of time. Entering a value that is too low for **min_cpu_interval** results in frequent checkpointing operations, and writing `.cas` and `.dat` files can be computationally expensive.

SGE requires checkpointing objects to perform checkpointing operations. Fluent provides a sample checkpointing object called `sample_ckpt_obj`.

Checkpoint configuration also requires root or manager privileges. While creating new checkpointing objects for Fluent, keep the default values as given in the sample/default object provided by Fluent and change only the following values:

- queue list (`queue_list`)

The queue list should contain the queues that are able to be used as checkpoint objects.

- checkpointing and migration commands (`ckpt_command` and `migr_command`)

These values should only be changed when the executable files are not in the default location, in which case the full path should be specified. All the files (that is, `ckpt_command.fluent` and `migr_command.fluent`) should be located in a directory that is accessible from all machines where the Fluent simulation is running. When running Fluent 2021 R2, the default location for these files is `path/ansys_inc/v212/fluent/fluent21.2.0/addons/sge`, where *path* is the Fluent installation directory.

- checkpointing directory (`ckpt_dir`)

This value dictates where the checkpointing subdirectories are created, and hence users must have the correct permission to this directory. Also, this directory should be visible to all machines where the Fluent simulation is running. The default value is `NONE` where Fluent uses the current working directory as the checkpointing directory.

- checkpointing modes (`when`)

This value dictates when checkpoints are expected to be generated. Valid values of this parameter are composed of the letters `s`, `m`, and `x`, in any order:

- Including `s` causes a job to be checkpointed, aborted, and migrated when the corresponding SGE **Exceed** daemon is shut down.
- Including `m` results in the generation of checkpoints periodically at the `min_cpu_interval` interval defined by the queue (see `qconf`).
- Including `x` causes a job to be checkpointed, aborted, and migrated when a job is suspended.

Important:

The `m` mode must be set to permit interval checkpointing.

2.3. Configuring Parallel Environments

For submitting parallel jobs, SGE needs a parallel environment (PE) interface. Fluent provides a sample parallel environment interface called `fluent_pe`.

Parallel environment configuration requires root or manager privileges. Change only the following values when creating a new parallel environment for Fluent:

- queue list (`queue_list`)

This should contain all the queues where `qtype` has been set to `PARALLEL`.

- user/xuser lists (`user_list` and `xuser_lists`)

These contain the lists of users who are allowed or denied access to the parallel environment.

- Shut-down procedure invocation command (`stop_proc_args`)

This should be changed only if the `kill-fluent` executable is not in the default directory, in which case the full path to the file should be given and the path should be accessible from every machine.

- `slots` (`slots`)

This should be set to a large numerical value, indicating the maximum of slots that can be occupied by all the parallel environment jobs that are running.

Since Fluent uses `fluent_pe` as the default parallel environment, an administrator must define a parallel environment with this name.

2.4. Default Request File

As an alternative to specifying numerous command line options or arguments when you invoke SGE, you can provide a list of SGE options in a default request file. All default request options and arguments for SGE that are common to all users can be placed in this file.

A default request file should be set up when using SGE with Fluent. Fluent provides a sample request file called `sge_request`. To learn more about how to configure and utilize this resource, see the relevant documentation available at www.oracle.com.

Individual users can set their own default arguments and options in a private general request file called `.sge_request`, located in their `$HOME` directory. Private general request files override the options set by the global `sge_request` file.

Any settings found in either the global or private default request file can be overridden by specifying new options in the command line.

Chapter 3: Running a Fluent Simulation under SGE

Information in this chapter is divided into the following sections:

- 3.1. Submitting a Fluent Job from the Command Line
- 3.2. Submitting a Fluent Job Using Fluent Launcher
- 3.3. Using Custom SGE Scripts

3.1. Submitting a Fluent Job from the Command Line

When submitting a Fluent job from the Linux command line using SGE, you must include additional options for the Fluent command syntax. The command line syntax is as follows:

```
fluent <solver_version> [<Fluent_options>] -scheduler=sge [-scheduler_queue=<queue>]
[-scheduler_opt=<opt>] [-scheduler_pe=<pe>] [-gui_machine=<hostname>] [-scheduler_headnode=<head-node>]
[-scheduler_stderr=<err-file>] [-scheduler_stdout=<out-file>] [-scheduler_tight_coupling]
```

where:

<solver_version>

specifies the dimensionality of the problem and the precision of the Fluent calculation (for example, 3d, 2ddp).

<Fluent_options>

specify the start-up option(s) for Fluent, including the options for running Fluent in parallel. For more information, see the Fluent [User's Guide](#).

-scheduler=sge

(start-up option) instructs Fluent to run under SGE.

-scheduler_queue=<queue>

(start-up option) sets the queue to <queue>.

-scheduler_opt=<opt>

(start-up option) enables an additional option <opt> that is relevant for SGE; see the SGE documentation for details. Note that you can include multiple instances of this option when you want to use more than one scheduler option.

The following is an example of an action you might want to take through such an option: you may want to specify the checkpointing object and override the checkpointing option specified in the default request file. If this option is not specified in the command line and the default general request file contains no setting, then the Fluent simulation is unable to use checkpoints.

-scheduler_pe=<pe>

(start-up option) sets the parallel environment to <pe> when Fluent is run in parallel. This should be used only if the `-scheduler=sge` option is used.

The specified <pe> must be defined by an administrator. For more information about creating a parallel environment, refer to the `sample_pe` file that is located in the `/addons/sge` directory in your Fluent installation area.

If Fluent is run in parallel under SGE and the `-scheduler_pe=` option is not specified, by default it will attempt to utilize a parallel environment called `fluent_pe`. Note that `fluent_pe` must be defined by an administrator if you are to use this default parallel environment.

-gui_machine=<hostname>

(start-up option) specifies that Cortex is run on a machine named <hostname>. This option may be necessary to avoid poor graphics performance when running Fluent under SGE.

-scheduler_headnode=<head-node>

(start-up option) specifies that the scheduler job submission machine is <head-node> (the default is `localhost`).

-scheduler_stderr=<err-file>

(start-up option) sets the name / directory of the scheduler standard error file to <err-file>; by default it is saved as `fluent.<PID>.e` in the working directory, where <PID> is the process ID of the top-level Fluent startup script.

-scheduler_stdout=<out-file>

(start-up option) sets the name / directory of the scheduler standard output file to <out-file>; by default it is saved as `fluent.<PID>.o` in the working directory, where <PID> is the process ID of the top-level Fluent startup script.

-scheduler_tight_coupling

enables tight integration between SGE and the MPI. It is supported with Intel MPI (the default). This option is not supported with the `-gui_machine=<hostname>` option.

Important:

You must have the `DISPLAY` environment variable properly defined, otherwise the graphical user interface (GUI) will not operate correctly.

The following examples demonstrate some applications of the command line syntax:

- Serial 2D Fluent simulation running under SGE

```
fluent 2d -scheduler=sge
```

- Parallel 2D Fluent simulation running under SGE on 4 CPUs

```
fluent 2d -t4 -scheduler=sge
```

- Parallel 2D Fluent under SGE using the parallel environment `diff_pe`

```
fluent 2d -t4 -scheduler=sge -scheduler_pe=diff_pe
```

- Parallel 2D Fluent under SGE on 4 CPUs, using the parallel environment `diff_pe` and the queue `large`

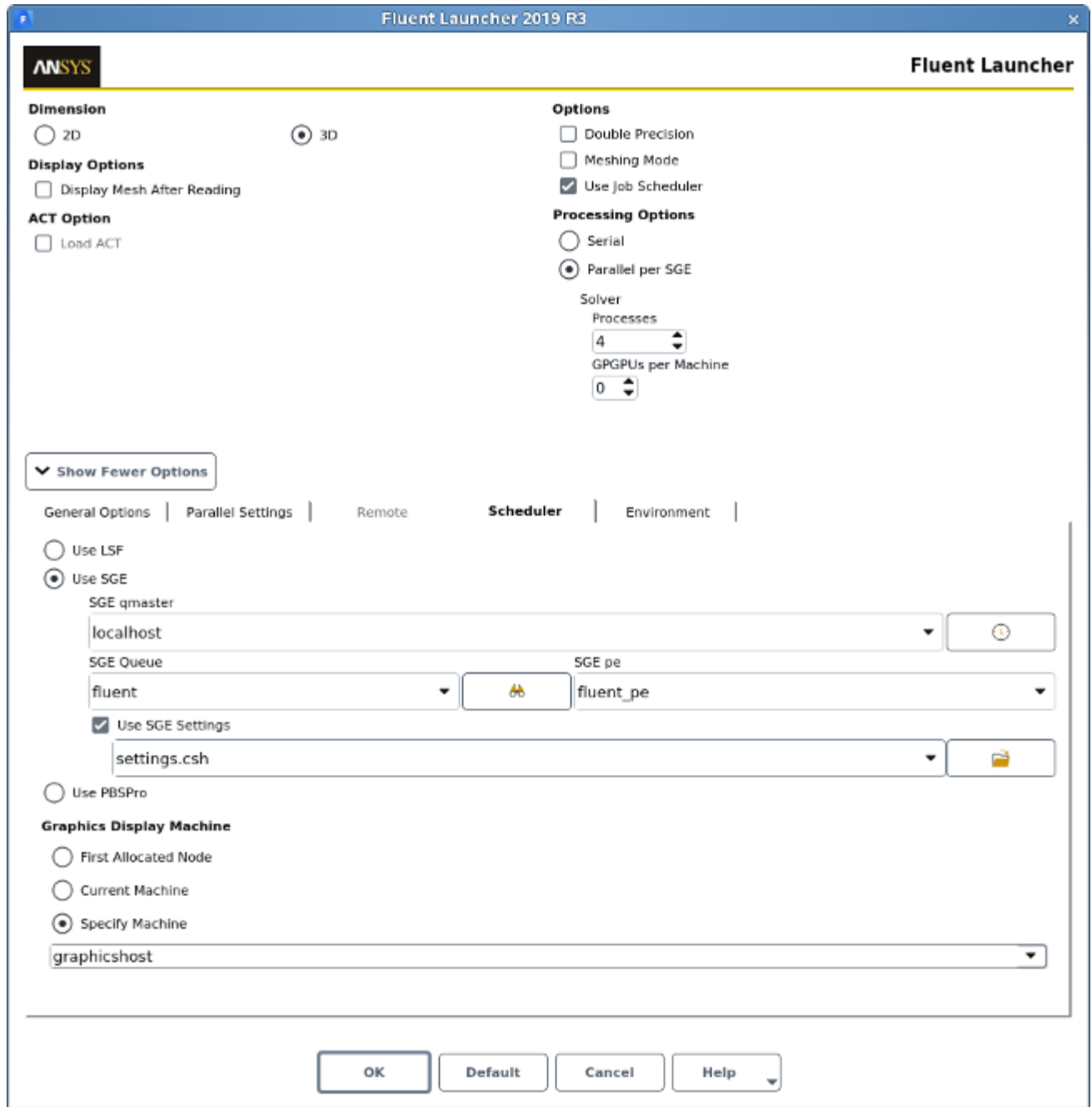
```
fluent2d -t4 -scheduler=sge -scheduler_queue=large -scheduler_pe=diff_pe
```

3.2. Submitting a Fluent Job Using Fluent Launcher



Fluent Launcher has graphical user input options that allow you to submit a Fluent job using SGE. Perform the following steps:

1. Open Fluent Launcher ([Figure 3.1: The Scheduler Tab of Fluent Launcher \(Linux Version\) \(p. 58\)](#)) by entering `fluent` without any arguments in the Linux command line.

Figure 3.1: The Scheduler Tab of Fluent Launcher (Linux Version)



2. Enable **Use Job Scheduler** under **Options**.
3. Click **Show More Options** to expand Fluent Launcher.
4. Click the **Scheduler** tab.
 - a. Select **Use SGE**.

- b. Enter the name of a node for **SGE qmaster**. SGE will allow this node to summon jobs. By default, **localhost** is specified for **SGE qmaster**. Note that the  button enables you to check the job status.
- c. You have the option of entering the name of a queue in which you want your Fluent job submitted for **SGE queue**. Note that you can use the  button to contact the **SGE qmaster** for a list of queues.
- d. If you are running a parallel simulation, you must enter the name of the parallel environment in which you want your Fluent job submitted for **SGE pe**. The parallel environment must be defined by an administrator. For more information about creating a parallel environment, refer to the `sample_pe` file that is located in the `/addons/sge` directory in your Fluent installation area.
- e. You can specify an SGE configuration file by enabling the **Use SGE settings** option. Then enter the name and location of the file in the text box or browse to the file.
- f. If you experience poor graphics performance when using SGE, you may be able to improve performance by changing the machine on which Cortex (the process that manages the graphical user interface and graphics) is running. The **Graphics Display Machine** list provides the following options:
- Select **First Allocated Node** if you want Cortex to run on the same machine as that used for compute node 0.
- Note that it is possible to enable a job-scheduler-supported native remote node access mechanism when the graphics display machine is set to the **First Allocated Node**. To do so, enable the **Tight Coupling** option. This tight integration is supported with Intel MPI (the default).
- Select **Current Machine** if you want Cortex to run on the same machine used to start Fluent Launcher.
 - Select **Specify Machine** if you want Cortex to run on a specified machine, which you select from the drop-down list below.

5. Set up the other aspects of your Fluent simulation using the Fluent Launcher GUI items. For more information, see the Fluent [User's Guide](#).

Important:

You must have the `DISPLAY` environment variable properly defined, otherwise the graphical user interface (GUI) will not operate correctly.

Note:

Submitting your Fluent job from the command line provides some options that are not available in Fluent Launcher, such as setting the scheduler standard error file or standard output file. For details, see [Submitting a Fluent Job from the Command Line](#) (p. 55).

3.3. Using Custom SGE Scripts

If you use custom SGE scripts instead of relying on the standard Fluent option (either the `-scheduler=sge` option from the command line or the **Use SGE** option in Fluent Launcher, as described in the preceding sections), your environment variables related to the job scheduler will not be used unless you include the `-scheduler_custom_script` option with the Fluent options in your script.

Chapter 4: Running Fluent Utility Scripts under SGE

You can run the Fluent utility scripts (for example, `fe2ram`, `fl42seg`, `partition`, `tconv`, `tmerge`) with the SGE load management system.

The command line syntax to launch a Fluent utility script under SGE is as follows:

```
utility <utility_name> [<utility_opts>] -sge [-sgeq <queue_name>] [-sgepe <parallel_env> <MIN_N>-<MAX_N>] <utility_
```

where

<utility_name>

is the name of the utility to be launched (for example, `fe2ram`, `fl42seg`, `partition`, `tconv`, `tmerge`).

<utility_opts>

are the options that are part of the syntax of the utility being launched. For more information about the options for the various utilities, see the Fluent [User's Guide](#).

-sge

(start-up option) instructs `<utility_name>` to run under SGE.

-sgeq <queue_name>

(start-up option) specifies the name of the queue.

-sgepe <parallel_env> <MIN_N>-<MAX_N>

(start-up option) specifies the parallel environment to be used when `<utility_name>` is run in parallel. This should be used only if the `-sge` option is used.

The specified `<parallel_env>` must be defined by an administrator. For more information about creating a parallel environment, refer to the `sample_pe` file that is located in the `/addons/sge` directory in your Fluent installation area.

The values for `<MIN_N>` and `<MAX_N>` specify the minimum and maximum number of compute nodes, respectively.

If `<utility_name>` is run in parallel under SGE and the `-sgepe` parameter is not specified, by default the utility will attempt to utilize a parallel environment called `utility_pe`. Note that `utility_pe` must be defined by an administrator if you are to use this default parallel environment. In such a case, `<MIN_N>` will be set to 1 and `<MAX_N>` will be set to the maximum number of requested compute nodes specified in the `<utility_opts>` (for example, `-t4`).

<utility_inputs>

are the fields that are part of the syntax of the utility being launched. For more information about the necessary fields for the various utilities, see the Fluent [User's Guide](#).

Important:

Note that neither checkpointing, restarting, nor migrating are available when using the utility scripts under SGE.

Part 4: Running Fluent Under Slurm

[About This Document \(p. lxxv\)](#)

[1. Introduction \(p. 67\)](#)

[2. Running an Ansys Fluent Simulation under Slurm \(p. 69\)](#)

About This Document

This document provides general information about running Ansys Fluent under Slurm. Examples have also been included, where available.

Information in this document is presented in the following chapters:

- [Introduction \(p. 67\)](#)
- [Running Fluent Simulation under Slurm \(p. 69\)](#)

This document is made available via the Ansys, Inc. website for your convenience.

Chapter 1: Introduction

Slurm is an open-source workload management tool for local and distributed environments. You can use Slurm when running Fluent simulations. This document provides general information about running Fluent under Slurm, and is made available via the Ansys, Inc. website for your convenience.

For more information, see the following section:

[1.1. Requirements for Running Fluent Jobs with Slurm](#)

1.1. Requirements for Running Fluent Jobs with Slurm

The following are the requirements for running Fluent jobs with Slurm:

- Standard
 - Slurm 20.11.3
 - Fluent 2021 R2

Important:

Running Fluent under Slurm is not supported on Windows.

Chapter 2: Running Fluent Simulation under Slurm

For more information, see the following sections:

- [2.1. Using the Integrated Slurm Capability](#)
- [2.2. Using Your Own Supplied Job Script](#)
- [2.3. Epilogue/Prologue](#)
- [2.4. Monitoring/Manipulating Jobs](#)

2.1. Using the Integrated Slurm Capability

For more information, see the following sections:

- [2.1.1. Overview](#)
- [2.1.2. Usage](#)
- [2.1.3. Examples](#)
- [2.1.4. Limitations](#)

2.1.1. Overview

One option of using Slurm with Fluent is to use the Slurm launching capability integrated directly into Fluent. In this mode, Fluent is started from the command line with the additional `-scheduler=slurm` argument. Fluent then takes responsibility for relaunching itself under Slurm. This has the following advantages:

- The command line usage is very similar to the non-RMS (resource management system) usage.
- You do not need to write a separate script.

The integrated Slurm capability is intended to simplify usage for the most common situations. If you desire more control over the Slurm `sbatch` options for more complex situations or systems (or if you are using an older version of Fluent), you can always write or adapt a Slurm script that starts Fluent in the desired manner (see [Using Your Own Supplied Job Script \(p. 74\)](#) for details).

2.1.2. Usage

Information on usage is provided in the following sections:

- [2.1.2.1. Submitting a Fluent Job from the Command Line](#)
- [2.1.2.2. Submitting a Fluent Job Using Fluent Launcher](#)

2.1.2.1. Submitting a Fluent Job from the Command Line

The integrated Slurm capability can be activated by simply adding the `-scheduler=slurm` option when launching Fluent from the command line:

```
fluent <solver_version> [<Fluent_options>] -i <journal_file> -scheduler=slurm [-scheduler_queue=<queue>] [-scheduler_account=<account>] [-scheduler_opt=<opt>] [-gui_machine=<hostname>] [-scheduler_headnode=<head-node>] [-scheduler_stderr=<err-file>] [-scheduler_stdout=<out-file>]
```

where

- `fluent` is the command that launches Fluent.
- `<solver_version>` specifies the dimensionality of the problem and the precision of the Fluent calculation (for example, 3d, 2ddp).
- `<Fluent_options>` can be added to specify the startup option(s) for Fluent, including the options for running Fluent in parallel. For more information, see the Fluent [User's Guide](#).
- `-i <journal_file>` reads the specified journal file(s).
- `-scheduler=slurm` is added to the Fluent command to specify that you are running under Slurm.
- `-scheduler_queue=<queue>` sets the Slurm partition to `<queue>`.
- `-scheduler_account=<account>` sets the Slurm account to `<account>`.
- `-scheduler_opt=<opt>` enables an additional option `<opt>` that is relevant for Slurm; see the Slurm documentation for details. Note that you can include multiple instances of this option when you want to use more than one scheduler option.
- `-gui_machine=<hostname>` specifies that Cortex is run on a machine named `<hostname>`. This option may be necessary to avoid poor graphics performance when running Fluent under Slurm.
- `-scheduler_headnode=<head-node>` allows you to specify that the scheduler job submission machine is `<head-node>` (the default is `localhost`).
- `-scheduler_stderr=<err-file>` sets the name / directory of the scheduler standard error file to `<err-file>`; by default it is saved as `fluent.<PID>.e` in the working directory, where `<PID>` is the process ID of the top-level Fluent startup script.
- `-scheduler_stdout=<out-file>` sets the name / directory of the scheduler standard output file to `<out-file>`; by default it is saved as `fluent.<PID>.o` in the working directory, where `<PID>` is the process ID of the top-level Fluent startup script.

This syntax will submit the Fluent job under Slurm using the `sbatch` command in a batch manner and return a job ID. This job ID can then be used to query, control, or stop the job using standard

Slurm commands, such as `squeue` or `scancel`. Slurm will start the job when resources are available. The job will be run out of the current working directory.

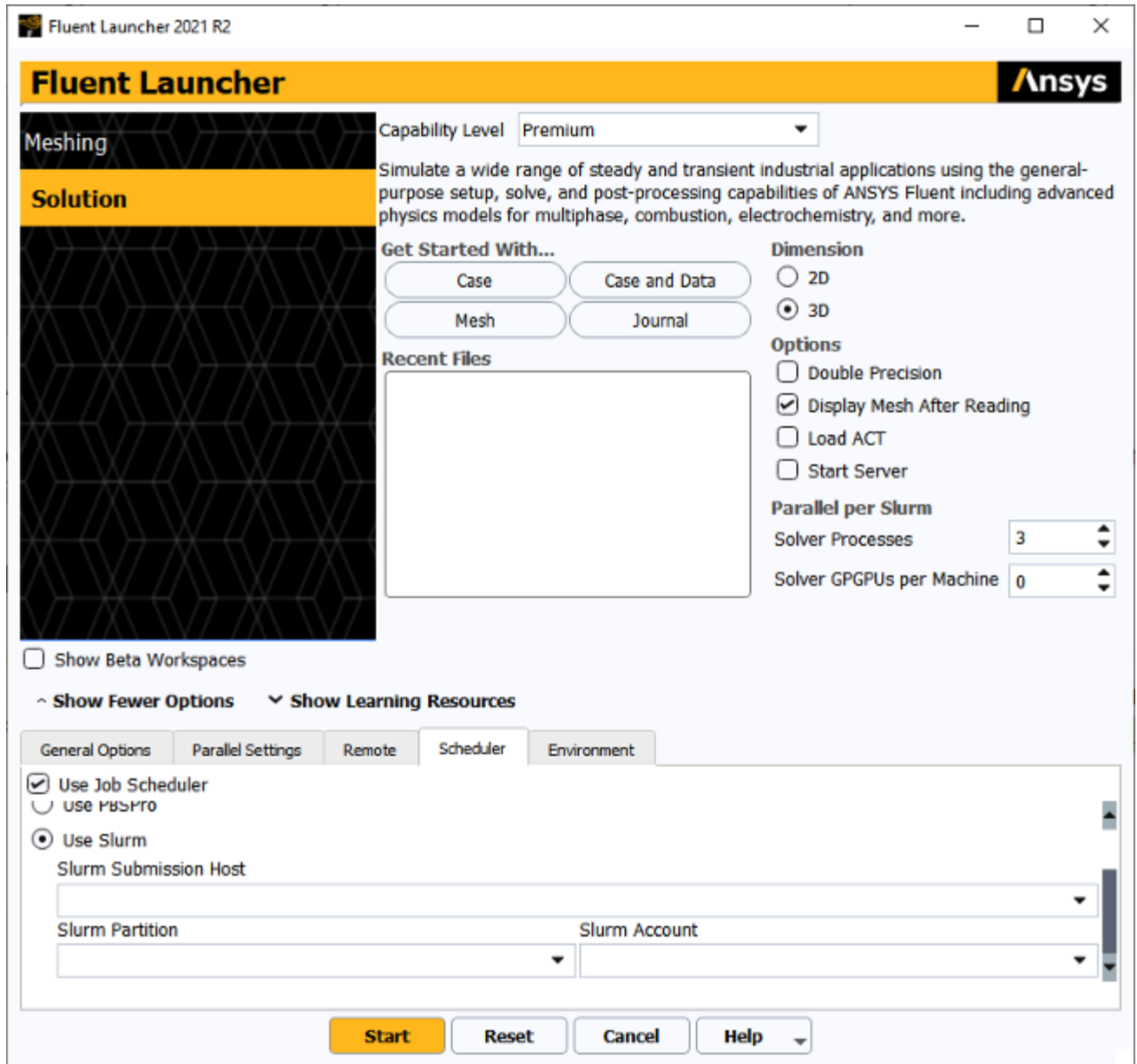
Note:

- You must have the `DISPLAY` environment variable properly defined, otherwise the graphical user interface (GUI) will not operate correctly.
- If you experience poor graphics performance when running Fluent under Slurm, you may be able to improve it by running Cortex on a specified machine through the use of the `-gui_machine=<hostname>` option.
- Dynamic spawning (that is, automatically spawning additional parallel node processes when switching from meshing mode to solution mode to achieve the requested number of total solution processes) is not allowed under Slurm, except when `-gui_machine=<hostname>` option is also used.
- Tight integration between Slurm and the MPI is enabled by default for Intel MPI (the default) without requiring an additional option. Tight integration is not supported with the `-gui_machine=<hostname>` option.

2.1.2.2. Submitting a Fluent Job Using Fluent Launcher

Fluent Launcher has graphical user input options that allow you to submit a Fluent job using Slurm. Perform the following steps:

1. Open Fluent Launcher ([Figure 2.1: The Scheduler Tab of Fluent Launcher \(Linux Version\) \(p. 72\)](#)) by entering `fluent` without any arguments in the Linux command line.

Figure 2.1: The Scheduler Tab of Fluent Launcher (Linux Version)

2. Enable **Use Job Scheduler** under **General Options** tab.
3. Click the **Show More Options** button to expand Fluent Launcher.
4. Select the **Scheduler** tab.
 - a. Select **Use Slurm**.
 - b. You can choose to make a selection from the **Slurm Submission Host** drop-down list to specify the Slurm submission host name for submitting the job, if the machine you are using to run the launcher cannot submit jobs to Slurm.
 - c. You can choose to make a selection from the **Slurm Partition** drop-down list to request a specific partition for the resource allocation.

- d. You can choose to make a selection from the **Slurm Account** drop-down list to specify the Slurm account.
- e. If you experience poor graphics performance when using Slurm, you may be able to improve performance by changing the machine on which Cortex (the process that manages the graphical user interface and graphics) is running. The **Graphics Display Machine** list provides the following options:
 - Select **First Allocated Node** if you want Cortex to run on the same machine as that used for compute node 0.
 - Select **Current Machine** if you want Cortex to run on the same machine used to start Fluent Launcher.
 - Select **Specify Machine** if you want Cortex to run on a specified machine, which you select from the drop-down list below.

Note:

Dynamic spawning (that is, automatically spawning additional parallel node processes when switching from meshing mode to solution mode to achieve the requested number of total solution processes) is not allowed under Slurm, except when you have selected **Current Machine** or **Specify Machine** from the **Graphics Display Machine** list.

Note:

Tight integration between Slurm and the MPI is enabled by default for Intel MPI (the default) only when **First Allocated Node** is selected from the **Graphics Display Machine** list.

5. Set up the other aspects of your Fluent simulation using the Fluent Launcher GUI items. For more information, see the Fluent [User's Guide](#).

Important:

You must have the `DISPLAY` environment variable properly defined, otherwise the graphical user interface (GUI) will not operate correctly.

Note:

Submitting your Fluent job from the command line provides some options that are not available in Fluent Launcher, such as setting the scheduler standard error file or standard output file. For details, see [Submitting a Fluent Job from the Command Line \(p. 70\)](#).

2.1.3. Examples

- Submit a parallel, 4-process job using a journal file `f15s3.jou`:

```
> fluent 3d -t4 -i fl5s3.jou -scheduler=slurm
Starting sbatch < user-scheduler-14239.slurm
/bin/sbatch
Submitted batch job 524
```

In the previous example, note that `sbatch` returns 524 as the job ID and `user-scheduler-14239.slurm` is name of the Slurm script written by Fluent for submitting this job.

- Check the status of the job:

```
> squeue -f 524
      JOBID PARTITION    NAME   USER  ST       TIME  NODES NODELIST(REASON)
       524  partname    fluent-t  user   R        45:23     2  compute[00-01]
```

The command in the previous example lists the status of the given job. The `squeue` command can be used to list all the jobs in the queue.

After the job is complete, the job will no longer show up in the output of the `squeue` command. The results of the run will then be available in the scheduler standard output file.

2.1.4. Limitations

The integrated Slurm capability in Fluent 2021 R2 has the following limitations:

- The Slurm commands (such as `sbatch`) must be in the users path.

2.2. Using Your Own Supplied Job Script

Slurm can accept any custom shell script for running Fluent that you may already use. Note that this shell script may require modification to move files between machines in the cluster. The shell script should also endeavor to clean up after itself when execution completes/fails.

If you use custom Slurm scripts instead of relying on the standard Fluent option (either the `-scheduler=slurm` option from the command line or the **Use Slurm** option in Fluent Launcher, as described in the preceding sections), your environment variables related to the job scheduler will not be used unless you include the `-scheduler_custom_script` option with the Fluent options in your script.

2.3. Epilogue/Prologue

Slurm provides the ability to script some actions immediately before the job starts and immediately after it ends (even if it is removed). The epilogue is a good place to put any best effort cleanup and stray process functionality that might be necessary to clean up after a job. For instance, one could include functionality to clean up the working/scratch area for a job if is deleted before its completion. Fluent provides a way to get rid of job-related processes in the form of a shell script in the work directory (`cleanup-fluent-<host-pid>`). It could be useful to have the epilogue call this script to ensure all errant processes are cleaned up after a job completes.

2.4. Monitoring/Manipulating Jobs

For more information, see the following sections:

2.4.1. Monitoring the Progress of a Job

2.4.2. Removing a Job from the Queue

2.4.1. Monitoring the Progress of a Job

The `squeue` command is used to monitor the status of jobs in a Slurm partition. When no job ID is specified, the `squeue` command will display all jobs. If you use `squeue -j job_ID` you will receive information about a specific job. This information includes the location at which the job is executing, which will be listed under the `NODELIST(REASON)` column.

```
> squeue -f 524
      JOBID PARTITION      NAME      USER ST      TIME  NODES NODELIST(REASON)
      524  partname    fluent-t    user  R      45:23    2  compute[00-01]
```

There are several job states to be aware of when using `squeue`, which are listed under the `ST` column. The two main states you will see are `PD` and `R`. `PD` indicates that the job is waiting in the queue to run. At this point the scheduler has not found a suitable node or nodes to run the job. Once the scheduler has found a suitable area to run the job and has sent the job to run, its state will be set to `R`.

2.4.2. Removing a Job from the Queue

The `scancel <job_ID>` command will delete the job indicated by `<job_ID>`.

