Skel User Manual

ADIOS 1.6.0
December 2013

SKEL USER MANUAL

Prepared for the
Office of Science
U.S. Department of Energy

Authors

J. Logan, N. Podhorszki, S. Klasky

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831-6070
managed by
UT-BATTELLE, LLC
for the
U.S. DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

Contents

List of Figures

Acknowledgments

This project is sponsored by ORNL, Georgia Tech, The Scientific Data Management Center (SDM) at Lawrence Berkeley National Laboratory, and the U.S. Department of Defense.

Chapter 1

Introduction

Skel is a tool used for the generation of I/O skeletal applications. Skel takes a high-level description of the I/O performed by an application, and generates a full benchmark that performs the described I/O. Compared to a scientific application, the generated benchmark is easy to build and run, and generally runs faster since it only performs I/O. Despite the simplicity of the generated code, it duplicates the I/O pattern of the target application, thus simplifying the process of understanding the I/O performance of an application.

Skel provides a simple mechanism for testing the performance of I/O operations that are relevant to applications of interest. This is of critical importance when evaluating new systems or new system configurations. It is equally useful for evaluating new I/O methods, or examining the effects of dirent parameters to existing I/O methods. Since the benchmarks produced by skel focus on the exact I/O patterns of applications, performance measurements obtained from those benchmarks will correlate highly with the performance of the actual applications.

This manual provides a detailed explanation of Skel, including the relevant file formats, documentation of skel commands, common usage of skel, and hints for porting skel to new platforms. Skel is young and still in development, so we expect there will be things that do not work perfectly, as well as things that have changed from the previous release. Please help us improve by letting us know when you encounter troubles. You can reach the skel developers by sending email to lot@ornl.gov.

Chapter 2

Using Skel

2.1 Requirements

Skel is bundled with ADIOS, and thus requires a system capable of building ADIOS on. In addition there are a couple of extra python packages that may be required to use all of the features of skel.

Skel has been tested to work with Python 2.7. Other versions of python may or may not work.

You will need Cheetah, which is free and open source software. Instructions for installing Cheetah may be found here: www.cheetahtemplate.org/docs/users_guide_html/users_guide.html This version of Skel has been tested with version 2.4.4 of Cheetah.

You will also need the PyYAML package. Instructions for downloading and installing PyYAML may be found here: http://pyyaml.org/wiki/PyYAML. This version of Skel has been tested with version 3.10 of PyYaml.

On Linux systems, you may be able to find these packages in your package manager's library, which may simplify the installation process.

2.2 Overview of Manual Benchmark Creation

Figure 2.1 shows the typical workflow of using skel to create a skeletal I/O benchmark. The example uses the GTS application, and thus the workflow begins with gts.xml, the XML descriptor from GTS. The *skel xml* command is used to create a second xml file, gts_skel.xml which will serve as the ADIOS xml descriptor for the skeletal application. Next, skel params is used to generate a parameters file. The generated parameters file is then edited by the user to guide the subsequent generation of the skeletal application.

At this point, the commands *skel source*, *skel makefile*, and *skel submit* may be used to generate the source files, Makefile, and submission scripts that comprise the skeletal application. With all of the components of the skeletal application created, it is now time to build the application, using *make*, and finally deploy the application to a directory from which it may be launched, using *make deploy*, which copies the gts_skel.xml file, executable files, and submission script. The user now has a ready to run I/O benchmark without having written any source code at all.

2.3 Detailed Example of Manual Benchmark Creation

In this section we will describe the steps used to create an I/O Kernel based on the GTS application. The ADIOS config file for GTS can be found in the examples directory.

The steps are

```
skel xml gts
skel params gts
```

Edit the parameters file, particularly the values of scalars that control array sizes, and the desired tests to run.

```
skel makefile gts
skel source gts
```

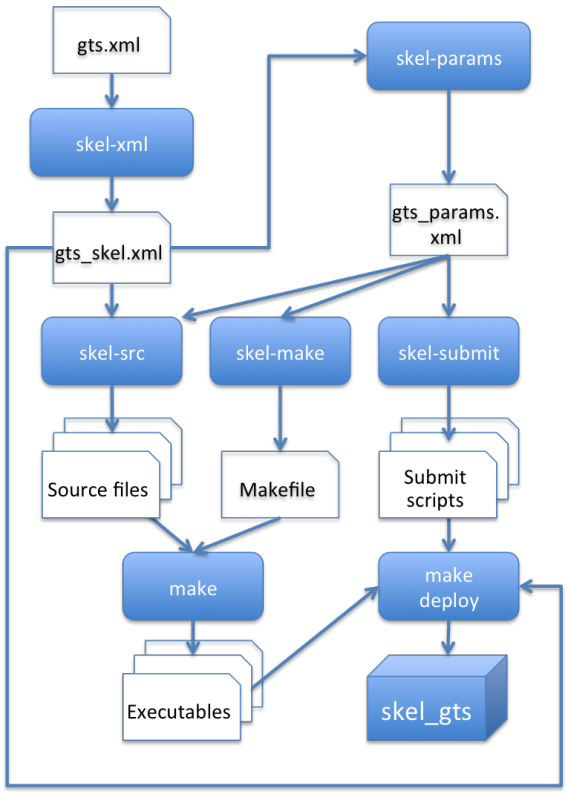

Figure 2.1: Skel Workflow

`skel submit gts` (optional)
`make`
`make deploy` (optional)

2.4 Recreating a Run Using Skel Replay (Experimental)

We have added a feature in this release that allows a benchmark to be created automatically based on an existing bp file. This replay feature eliminates the tedious nature of creating and tweaking a parameter file, instead taking that information directly from the bp file, settings file, or command line, in order to allow direct replay while simplifying the steps that a user must take.

To show the usage of the replay feature, assume we have a bp file, out.bp, that was produced by an application called myapp. Generating a benchmark based on out.bp is simple:

```
skel replay myapp -b out.bp
```

This single command generates an adios XML file, source code, a makefile, and a submission script, and finally executes the makefile, providing a ready to run benchmark.

2.5 Using Skel for a remote replay (Experimental)

In the case where you have a large bp file representing an I/O pattern you want to run on a different machine, it may be desirable to extract the metadata information from the bp file, rather than transferring the entire file to the remote machine. To accomplish this, simply use the skeldump command as follows:

```
skeldump out.bp > out.yaml
```

Skeldump creates a yaml file containing all of the information required to build the benchmark on the remote machine. After moving only the yaml file to the remote machine, the benchmark creation can be accomplished with:

```
skel replay myapp -y out.yaml
```

Chapter 3

Skel Command Reference

Most skel commands are of the form: skel subcommand project

3.1 Available Subcommands

skel install

Usage:

```
skel install
```

skel makefile

Generates a Makefile fit for building and deploying the skeletal application.

Requires <project>_skel.xml and <project>_params.xml

Usage:

```
skel makefile <project>
```

skel params

Generates a parameters file that can be customized by the user. Note that this command creates a file called <project> params.xml.default so as to avoid overwriting a customized parameters file. This means that the user should copy this file to <project> params.xml and edit before proceeding with code generation.

Requires <project>_skel.xml

Usage:

```
skel params <project>
```

skel replay

Generates a C or Fortran code that performs the I/O operations described by the given bp file or yaml file.

Usage:

```
skel replay <project> -y <yaml_file>
```

```
skel replay <project> -b <bp_file>
```

skel source

Generates a C or Fortran code that performs the I/O operations described by the XML descriptor and the parameters file.

Requires <project>_skel.xml and <project>_params.xml

Usage:

```
skel source <project>
```

skel submit

Generates a submission script for the skeletal application

Requires <project>_skel.xml and <project>_params.xml

Usage:

```
skel submit <project>
```

skel xml

Generates the <project>_skel.xml file.

Requires <project>.xml

Usage:

```
skel xml <project>
```

3.2 Skel Utilities

skeldump (experimental)

Extracts necessary metadata from a bp file to create a skeletal application. Metadata is produced in yaml format, and is sent to standard out.

Usage:

```
skeldump <bpfile> > <yamlfile>
```

Chapter 4

The Parameters File

Generation of a skeletal application requires some more information that is found in the ADIOS configuration file. To specify this additional information, the user must supply a Skel parameters file. The parameters file is an XML file which contains the elements described in the following section. Although it not overly dicult to construct your own parameters files, users may find it more convenient to use the *skel params* command to automatically generate a parameters file with default values, then simply edit the file to provide the desired configuration.

4.1 Elements

<skel-config>

Each parameters file must contain exactly one <skel-config> element as the only element at the root level of the document. The <skel-config> element should contain one <adios-group> element and one <batch> element, as described below.

Supported Attributes:

application The name of the application described by the document. This should correspond to the project used for the skel calls, and the name of the original XML file given to skel xml .

<adios-group>

The <adios-group> element is a child of the root <skel-config> element. It corresponds to the adios-group that will be written by the I/O skeletal application being generated. The <adios-group> element contains a collection of <scalar> and <array> elements corresponding to the variables described for the group in the ADIOS config file.

Supported Attributes:

name The name of the ADIOS group that this element describes

<scalar>

Represents a scalar variable that is described in the ADIOS descrip- tor. Often integer valued scalars will be used to determine the dimensions of arrays, thus providing appropriate values is essential for creating mean- ingful benchmarks.

Supported Attributes:

name The name of the scalar variable

type (Optional) The type of this variable in the generated code. Supplied for convenience by skel params .

value A value, of the proper type, that will be assigned to this scalar variable in the generated code.

<array>

Represents an array variable that is described in the ADIOS descrip- tor.

Supported Attributes:

name The name of the array variable

type (Optional) The type of this variable in the generated code. Supplied for convenience by *skel params*.

dimensions (Optional) A list of the scalar values that will be used to determine the dimensions of this array

fill-method Determines how the array memory will be initialized in the generated code

<batch>

Describes a set of tests to be performed by the generated I/O kernel. The <batch> element contains one or more <test> elements.

Supported Attributes:

name The name of the batch. Used to name the submission script and other elements of the batch job.

cores The number of MPI tasks that will be used for the skeletal application

walltime Amount of runtime requested by the generated submission script

<test>

Describes a single test to be performed.

Supported Attributes:

type The type of test to be performed. Currently only write is supported.

group The ADIOS group to be written

method The ADIOS write method to use for writing (i.e. POSIX). A full listing of available methods can be found in the ADIOS manual.

iterations The number of times to repeat this test

rm determines whether and when to remove the written output files. One of {pre, post, both, none}

Chapter 5

Yaml File Format

Skel supports various methods for specifying the high-level I/O description to be used for creating skeletal applications. One of these is the yaml file, described below. YAML is a data serialization language that is similar to JSON. YAML supports data abstractions including sequences and mappings, and allows these to be nested. General information about YAML is available at http://www.yaml.org/

The yaml format described here is the one that is produced by the `skeldump` utility, and which is accepted by the `skel replay` command, both described elsewhere in this manual.

At the top level of the yaml file, there are a series of mappings as follows:

lang specifies the target language, currently C and fortran are supported.

procs indicates the number of MPI tasks involved in the I/O operations.

group is the name of the ADIOS group in which to write the variables.

variables is a sequence of mappings representing the variables to be written

Each variable is represented by a nested mapping that consists of:

name The name of the variable

type The unit type of the variable, should correspond to a valid type in the target language

dims Either `scalar`, or a comma separated list of array dimensions

value (For scalars) a string representing the value to be assigned to this variable in the skeletal application code.

Chapter 6

Skel Settings File

Skel provides a collection of configurable options which are exposed in the Skel settings file. The settings file is located in the user's home directory at /.skel/settings. The settings file consists of single line entries of the form <name>=<value>. Blank lines and lines starting with # are ignored. Entries in the settings files are case sensitive, and lower case is typically used for all names and yes/no values.

6.1 Available Settings

deploy_dir This is the directory into which the compiled applications will be copied to be executed. This directory should be visible to the compute nodes.

submit_target This determines which submission script template will be used by `skel_submit`. Templates for titan and sith are included in this release.

sleep_before_open (yes/no) Determines whether a short sleep statement is inserted in the generated code before the adios open call. Default is no.

barrier_before_open (yes/no) Determines whether an `MPI_Barrier` call is inserted into the generated code before `adios_open`. Default is yes.

barrier_before_access (yes/no) Determines whether an `MPI_Barrier` call is inserted into the generated code before the sequence of `adios_write` calls. Default is no.

barrier_before_close (yes/no) Determines whether an `MPI_Barrier` call is inserted into the generated code before `adios_close`. Default is no.

barrier_after_close (yes/no) Determines whether an `MPI_Barrier` call is inserted into the generated code after `adios_close`. Default is no.

barrier_after_steps (yes/no) Determines whether a single `MPI_Barrier` call is inserted into the generated code after the final `adios_close`. Default is no.

use_adios_timing (yes/no) Determines whether a call is inserted to output detailed timing information collected by ADIOS. For this to work, your adios distribution must have been configured using `-enable-skel-timing`. Default is no.

Chapter 7

Low-Level Timing Mechanism

By default, applications generated by skel will produce a summary timing re- port, sending it to standard out. On most platforms, this will be captured in the output file produced by the job script. The summary report contains tex- tual information about the overall time taken by the various I/O operations. If more detail is desired, ADIOS has a mechanism for gathering low-level timing information for various events that occur within the ADIOS I/O calls.

7.1 Using the Low-Level Timing Mechanism

To use the low-level timing mechanism, you must use an ADIOS library that has been built with this low-level timing mechanism enabled. Simply build ADIOS as described in the ADIOS manual, inserting –enable-skel-timing in the configure command. We do not recommend enabling the low-level timing mechanism while running production codes. Once you have enabled low-level timing in ADIOS, you only need to enable generation of low-level timing calls in your Skel settings file. This is done by including the line: `use_adios_timing=yes` in your Skel settings file. This will cause the skel source command to include an additional call near the end of the generated skeletal application to output the detailed timing information that has been collected. The detailed timing information is written to a separate file using XML. This will work best if used to write a single iteration of a single group. The low-level timing mechanism provides detailed timing information for only some of the available write methods. As of this release, the supported methods are POSIX , MPI LUSTRE , and MPI AMR .

7.2 Extracting Timing Information

The XML file that is produced by ADIOS contains a large amount of measure- ment data, but it is somewhat unwieldy to work with directly. So, we have included an additional utility, skel extract.py , which allows data from the XML file to be exported as a CSV file that is simple to load using tools such as R or Matlab.

Chapter 8

Hints for Porting Skel

Skel has been developed and tested on only a small handful of platforms. While we expect most functionality will be portable to a wider range of machines, there are likely to be some issues arising when running skel on your system. There are a few hints in the following sections that may help you to get started. If further assistance is needed to get skel working, please contact lot@ornl.gov.

8.1 Makefiles

Assuming that they work properly, Makefiles generated by skel are quite conve- nient, as the skel user need not think about how to compile the code, but can simply type make. All of the details are taken care of by skel. We have tested skel on only a few systems at this point, and thus it is quite possible that the Makefile generated by skel may fail on some systems. Users familiar with make with a need to adjust some aspect of the generated makefiles should investigate /.skel/templates/Makefile.default.tpl. This template file is used by Skel to generate Makefiles, and can be adjusted to the needs of the user. The template syntax is simple, with `$$VAR_NAME$$` used to indicate template substitutions to be made by skel makefile. Again, if you run into trouble with this, please contact us as described above.

8.2 Submission Scripts

Similar to the Makefile generation, skel uses templates to generate submission scripts for the generated applications. The submission templates are also located in /.skel/templates/, and are named submit <target>.tpl, where <target> corresponds to the submit target defined in the user's settings file. To create a new submit target , simply copy one of the existing template files, and rename it with the desired submit target name. Then, adjust the submission syntax so that the generated files work properly with the submission mechanism on your system. Once again, if you run into trouble with this, please contact us as described above.

8.3 Cheetah templates

We are in the process of modifying the code generation system to use Cheetah. Cheetah provides a powerful and flexible template language that simplifies code generation for more complicated scenarios, and will make creating new generators less tedious. Currently only the `skel replay` subcommand uses Cheetah, but future releases should shift completely to Cheetah. Cheetah templates are included in the same directory with the skel templates, but use the .tmpl extension. For information about the syntax supported by Cheetah, see the Cheetah website: http://www.cheetahtemplate.org/