Big Data Research ••• (••••) •••–•••

Contents lists available at ScienceDirect

Big Data Research

www.elsevier.com/locate/bdr

Using the Launcher for Executing High Throughput Workloads ☆

Lucas A. Wilson

Texas Advanced Computing Center, The University of Texas at Austin, United States

ARTICLE INFO

Article history:
Received 27 May 2016
Received in revised form 24 February 2017
Accepted 3 April 2017
Available online xxxx

Keywords:
High throughput computing
Distributed computing
Parallel computing

ABSTRACT

For many scientific disciplines, the transition to using advanced cyberinfrastructure comes not out of a desire to use the most advanced or most powerful resources available, but because their current operational model is no longer sufficient to meet their computational needs. Many researchers begin their computations on their desktop or local workstation, only to discover that the time required to simulate their problem, analyze their instrument data, or score the multitude of entities that they want to would require far more time than they have available.

Launcher is a simple utility which enables the execution of high throughput computing workloads on managed HPC systems quickly and with as little effort as possible on the part of the user. Basic usage of the Launcher is straightforward, but Launcher provides several more advanced capabilities including use of Intel® Xeon Phi™ coprocessor cards and task binding support for multi-/many-core architectures. We step through the processes of setting up a basic Launcher job, including creating a job file, setting appropriate environment variables, and using scheduler integration. We also describe how to enable use of the Intel® Xeon Phi™ coprocessor cards, take advantage of Launcher's task binding system, and execute many parallel (OpenMP/MPI) applications at once.

1. Introduction

For many scientific disciplines, the transition to using advanced cyberinfrastructure comes not out of a desire to use the most advanced or most powerful resources available, but because their current operational model is no longer sufficient to meet their computational needs. Many researchers begin their computations on their desktop or local workstation, only to discover that the time required to simulate their problem, analyze their instrument data, or score the multitude of entities that they want to would require far more time than they have available.

At this point, when the task a researcher wishes to perform may take many months or years to complete, they will likely turn to one of the university, state, or national resources that are available to the scientific community. Many of these resources are tailored for problems where solutions cannot be found without performing frequent, high-speed data exchanges. These problems – focused primarily in physics, chemistry, and engineering – require frequent exchange of data in order to solve the underlying equations of state. For researchers with these types of problems the path toward effectively using advanced cyberinfrastructure may be difficult, but is well trodden.

For researchers in other domains, the problem that they face is not the scale of a single domain for which a global equation of state must be solved, but the multitude of small data sets which must be independently analyzed. In this computational model, called *throughput computing*, processes do not require frequent data exchange (they often require no data exchange), and can be properly executed in essentially any order.

2. Related approaches

While many efforts in the throughput-computing and workflow management space have resulted in fairly heavy-weight software solutions [8,11] intended to run either on dedicated hardware or in cloud-based architectures, there are few lightweight software solutions that can perform these low-communication workflows on existing HPC systems, which comprise the vast majority of the available computational cycles in large-scale installations.

In addition to workflow managers, emerging parallel scripting languages such as Swift [16] seek to provide improved capability for programmers to parallelize jobs across heterogeneous hardware infrastructures. However, the layer of abstraction required to make these current languages work across systems simultaneously creates an unnecessary level of complexity for users of homogeneous HPC clusters.

Many resource managers provide a similar mechanism for these problems called job arrays [4,17,15]. These job arrays allow the

☆ This article belongs to HPC Tutorial for Big Data.

E-mail address: lucaswilson@acm.org.

http://dx.doi.org/10.1016/j.bdr.2017.04.001
2214-5796/

user to create a single job out of many smaller, independent jobs, and run them on multiple nodes and processors. However, administrators for many shared HPC resources disable job array, because they provide a way to circumvent fair use limits placed on the system such as jobs-in-queue and dynamic submission restrictions.

Additionally, tools such as Apache YARN [12] and Mesos [6] are well suited to systems devoted to analytic processing using tools like Hadoop/Spark, which use the MapReduce [3] computational model. However, when the primary concern is running on shared cyberinfrastructure in the form of managed HPC systems, these particular resource brokering tools are not necessarily available, and may require significant re-engineering on the part of the user in order to be of use.

3. Why Launcher?

Launcher is a simple utility for executing the throughput computing workloads on managed advanced cyberinfrastructure systems [14,13]. *Managed* systems are those which are operated by a third party entity (e.g., university/state/national computing center) where the user has no input in or control of administrative decisions, and jobs are scheduled to execute using a resource manager such as Platform LSF [17], Oracle GridEngine [4], SLURM [15], OpenPBS [7], or IBM LoadLeveler [10]. These systems are typically tailored for physics and engineering problems, where a single program uses many resources coupled with data exchange, such as programs which make use of the Message Passing Interface (MPI). On these systems, execution of sequential or single-process (non-MPI) jobs may be discouraged or prevented altogether. Launcher provides a facility for running many sequential or single-process tasks bundled into a single multi-node job, which can be more efficiently scheduled on these managed systems.

Solving big data problems in many cases involves large, computationally intensive simulations or analyses within a parameter space. Launcher is intended for exactly these types of problems. Using Launcher to speed searches through parameter spaces can improve can not only help a researcher to generate the data necessary for further analysis, but speed up the analysis or preconditioning of data points before further analysis or reduction.

4. The basics

Throughout there will be examples of various scenarios where Launcher can be employed. The Launcher can be downloaded from GitHub at: https://github.com/TACC/launcher. Installation on a local system involves unpacking the tarball exporting the following environment variable:

```
export LAUNCHER_DIR=<directory containing
                     launcher files>
```

4.1. How Launcher works

Launcher consists of four major scripts which work together to run HTC workloads on managed HPC systems. The scripts are responsible for kick starting multi-node execution, managing multi-core execution, and executing and scheduling individual jobs:

- `paramrun`: Top-level script responsible for interfacing with the system's resource manager, ensuring appropriate environment variables are set, and kick-starting multi-node execution,
- `tskserver`: Python-based TCP/IP dynamic scheduling service,
- `init_launcher`: Script responsible for on-node process management for multi-core and many-core processors, and

```
./a.out $LAUNCHER_TSK_ID
./a.out $LAUNCHER_JID
echo $LAUNCHER_PPN
echo $LAUNCHER_NHOSTS
grep "bar" foo | wc -l > baz.o$LAUNCHER_JID
```

Listing 1: Examples of valid job file entries [14].

- `launcher`: Leaf script responsible for executing appropriate jobs from the job file, timing individual jobs, and providing stdout/stderr content.

Fig. 1 shows the hierarchy of shell scripts used by the launcher to efficiently execute across large numbers of multi-core and many-core servers, as well as the environment variables used/defined by the launcher which are available to the user (see Table 1).

4.2. Basic environment variables

Any of the provided variables in Table 1 can be referenced in the *job file*. The job file can contain any shell-executable commands, including referencing external shell scripts, using bash for loops and pipes, and stdin/stdout/stderr redirection. The job file can consist of any number of commands, placed one-per-line, so long as there are no blank lines in the file. Listing 1 shows several examples of valid job file entries. Launcher can be directed to this job file by exporting the `LAUNCHER_JOB_FILE` environment variable prior to calling `paramrun`.

4.3. "Hello, World!" – a simple launcher bundle

With all of the pieces of the launcher explained, we can now put together a complete launcher bundle. Launcher can execute on workstations and work groups of computers, as well as on HPC systems. For this example, we will build a bundle to be run on a cluster which is managed with the SLURM resource manager, although other resource managers are easily supported.

Step 1: create a job file

As a first step, we will create a job file which prints a variant of "Hello, World!" to the screen as many times as we wish. To start, we can create a text file (`helloworld`) which contains some number of copies of this line, remembering to remove any blank lines:

```
echo "Hello from $LAUNCHER_JID, task
$LAUNCHER_TSK_ID!"
```

For convenience, you can reference `extras/examples/helloworld` inside the Launcher source directory.

Step 2: run the bundle

Once we have built a job file, we can execute the bundle in a few different ways, depending on the system being used. On local machines, a launcher bundle can be directly executed from the command-line by setting `LAUNCHER_JOB_FILE` and invoking the `paramrun` script:

```
$ export LAUNCHER_JOB_FILE=helloworld
$ LAUNCHER_DIR/paramrun
```

On managed systems where a job scheduler is used, a batch submission script must first be build. When using SLURM, a simple batch script which will run the launcher on whatever nodes SLURM allocates for this particular batch job. An example job script, which assumes 4 cores per node and requests 4 nodes, is given in Listing 2.

Please cite this article in press as: L.A. Wilson, Using the Launcher for Executing High Throughput Workloads, Big Data Res. (2017), http://dx.doi.org/10.1016/j.bdr.2017.04.001