LPJS - Lightweight, Portable Job Scheduler

Contents

Chapter 1

Introduction

LPJS is a resource manager and job scheduler for running batch jobs on one or more computers. It can be used on a single machine in order to maximize utilization of CPUs and memory without oversubscribing the system, or on multiple networked computers organized as an HPC (high performance computing) cluster or HTC (high throughput computing) grid.

Unlike most similar tools, LPJS is designed to be small, simple, easy to install and configure, and easy to use. It provides an intuitive user interface, including menu-based operation for most common tasks.

This manual is aimed at the systems manager, covering installation and configuration of LPJS. For a user's guide, including general information on HPC clusters and HTC grids, see the Research Computing User's Guide at https://acadix.biz/publications.php.

Chapter 2

Installation

Installation should be performed using a package manager, such as FreeBSD ports or pkgsrc. We maintain a FreeBSD port for use on FreeBSD and Dragonfly BSD, and a pkgsrc package that should work on any other POSIX platform, including other BSDs, most Linux distributions, macOS, Solaris-based systems, etc. Other package managers may be supported by third parties. If you would like to add LPJS to your favorite package manager, see the instructions for packagers in the README at https://github.com/outpaddling/LPJS/.

See http://netbsd.org/~bacon/ for an introduction to installing pkgsrc.

LPJS uses **munge** (https://github.com/dun/munge) to encrypt and authenticate messages between nodes. Munge is installed automatically by the package manager when installing LPJS, and automatically configured by LPJS administration scripts.

Munge requires all nodes to have a shared munge key file, which is unique to your installation. It must be generated by you and distributed to all computers that are part of your cluster or grid. THE MUNGE KEY FILE MUST BE KEPT SECURE AT ALL TIMES ON ALL NODES. Use secure procedures to distribute it to all nodes, so that it is never visible to unauthorized users.

Chapter 3

Configuration

LPJS is designed to require minimal configuration. For example, compute node resources such as processors and memory are determined automatically, and need not be specified in configuration files.

Most configuration can be done entirely using **lpjs admin** (**man lpjs-admin**), a menu-driven admin tool. Simply run **lpjs-admin**, select an item from the menu, and answer the questions on the screen.

For the sake of understanding what **lpjs-admin** does, some basic information is provided below.

The head node requires a configuration file, which in its simplest form merely lists the complete host names (FQDNs) of the head node and each compute node, e.g.

```
head    myhead.mydomain
compute compute001.mydomain
compute compute002.mydomain
...
```

The FQDN (fully qualified domain name) must match the name reported by the **hostname** command, or the `gethostname()` standard library function. This is usually either listed in `/etc/hosts` or provided by *DNS* (*domain name service*).

Each compute node requires the same type of configuration file, but it need only list the head node. It can be the same configuration file used on the head node, in which case the compute node entries are ignored. You may wish to create a configuration file on the head node and simply distribute it.

Chapter 4

Starting Daemons

4.1 General Info

All nodes in the cluster or grid must be running **munged**, using the same munge key. The head node must also run **lpjs_dispatchd**, and all compute nodes must run **lpjs_compd**.

Appropriate services can be configured by running **lpjs-admin** on each node.

The head node can also serve as a compute node, though this is not generally recommended. The head node of most clusters and grids should remain lightly loaded so that it can respond promptly to events that occur such as new job submissions and job completions. The head node need not be a powerful machine. A laptop or low-end desktop machine will work just fine for a small cluster. Laptops are actually nice in that they have a built-in battery backup, so your head node at least is protected against power outages.

The head node on larger clusters need not be powerful, but should be highly reliable. We recommend a server with a mirrored boot disk, hot-swap disks, and possibly redundant power supplies, also hot-swap. With this hardware configuration, down time should be near zero. FreeBSD makes it easy to mirror a boot disk using any computer with two drives. However, a hardware RAID card will make it easier to swap out a bad disk than a ZFS RAIDZ, assuming the disks are hot-swap.

Jobs can be submitted from any node with the same version of LPJS and the shared munge key installed.

Note It need not even be running LPJS daemons, but it does require a configuration file listing the head node, and munge to authenticate requests.

Hence, other computers on the network can act as submit nodes, even if they are not part of the cluster/grid.

4.2 Daemons as a Service

The **lpjs_dispatchd** and **lpjs_compd** commands are normally run as a service, which automatically starts when the computer is rebooted. You can run **lpjs admin** and use the menus to configure a machine as a head node or compute node with the appropriate services enabled. This will require administrative rights on each computer in the cluster/grid.

4.3 Ad hoc Clusters and Grids

It is also possible to use LPJS without enabling services, even without having admin rights. Simply start the daemons manually by running **munged** and **lpjs_dispatchd** on the head node, and **munged** and **lpjs_compd** on each compute node. Note that if **lpjs_compd** is not running as root, the compute node will only be able to run jobs under the same user name running **lpjs_compd**.

The **lpjs ad-hoc** command displays a menu, allowing you to start and stop the appropriate daemons without having to know the precise commands.

Chapter 5

Network topology

A *cluster* is generally a collection of dedicated computers, all connected directly to the same private, often high speed network. In this sense, a cluster is a LAN (local area network). In many cases, special network technology, such as *Infiniband*, is used in place of, or in addition to, standard Ethernet. Infiniband and similar technologies offer much lower latency per message, and higher throughput.

A *grid* is conceptually like a cluster, in that it is used for distributed parallel computing. However, grids are more loosely coupled, often utilizing computers that are not on the same LAN. Hence, a grid is not as suitable for parallel computing that involves a lot of communication between processes, such as MPI (Message Passing Interface) distributed parallel programs.

LPJS is very flexible with network topology. The *only* requirement is that all nodes are able to connect to the head node. This means that a cluster or grid using LPJS can consist of other computers on the same LAN, computers in different buildings, virtual machines behind a *NAT* (*network address translation*) firewall, or cloud instances in a data center a thousand miles away. You must consider how each of these resources can effectively be used. Network latency and throughput may be quite poor for resources that are far away.

Chapter 6

File Sharing

Clusters normally have one or more file servers, so that jobs can run in a directory that is directly accessible from all nodes. This is the ideal situation, as input files are directly available to jobs, and output files from jobs can be written to their final location without needing to transfer them.

Note
At present, it appears to be impractical to use macOS for compute nodes with data on a file server. macOS has a security feature that prevents programs from accessing most directories unless the user explicitly grants permission via the graphical interface. In order for LPJS to access file servers as required for normal operation, the program **lpjs_compd** must be granted full disk access via System Settings, Privacy and Security. Otherwise, you may see "operation not permitted" errors in the log when trying to access NFS shares.
The major problem is that this is not a one-time setting. Each time LPJS is updated, full disk access is revoked, and the user must enable it via the graphical interface again.

Grids normally do not have file servers. In this case, it will be necessary for all nodes to have the ability to pull files from and push files to *somewhere*. Typically, this somewhere would be the submit node, or a server accessible for file transfers from the submit node and all compute nodes.

LPJS does not provide file transfer tools. There are numerous highly-evolved, general-purpose file transfer tools already available, so it is left to the systems manager and user to decide which one(s) to use. We recommend using **rsync** if possible, as it is highly portable and reliable, and minimizes the amount of data transferred when repeating a transfer.

Note All compute nodes must be able to perform a passwordless file transfers to the designated server, i.e. pulling files to, or pushing files from a compute node does not prompt the user for a password. This is generally accomplished by installing ssh keys on the submit node, which can be done by running **auto-ssh-authorize submit-host** from every compute node, as every user who will run jobs.

The **lpjs submit** command creates a marker file in the working directory on the submit host, named "lpjs-submit-host-name-shared-fs-marker" (replace "submit-host-name" with the FQDN of your submit node). If this file is not accessible to the compute node, then LPJS will take the necessary steps to create the temporary working directory and transfer it back to the submit node after the script terminates.

If the working directory (the directory from which the job is submitted on the submit node) is not accessible to the compute nodes (e.g. using NFS), then the user's script is responsible for downloading any required input files. Below is an example from `Test/fastq-trim.lpjs` in the LPJS Github repository.

Note Note that we used the `--copy-links` option with rsync, so that it copies files pointed to by symbolic links, rather than just recreating the symbolic link on the compute node. You must understand each situation and decide whether this is necessary.

```
# Marker file is created by "lpjs submit" so we can detect shared filesystems.
# If this file does not exist on the compute nodes, then the compute nodes
# must pull (download) the input files.
marker=lpjs-$LPJS_SUBMIT_HOST-shared-fs-marker
if [ ! -e $marker ]; then
    printf "$marker does not exist.  Using rsync to transfer files.\n"
    set -x
    printf "Fetching $LPJS_SUBMIT_HOST:$LPJS_WORKING_DIRECTORY/$infile\n"
    # Use --copy-links if a file on the submit node might be a symbolic
    # link pointing to something that it not also being pulled here
    rsync --copy-links ${LPJS_SUBMIT_HOST}:$LPJS_WORKING_DIRECTORY/$infile .
    set +x
else
    printf "$marker found.  No need to transfer files.\n"
fi
```

LPJS will, by default, transfer the contents of the temporary working directory back to the working directory on the submit node, using **rsync -av temp-working-dir/ submit-host:working-dir**. The "working-dir" above is the directory from which the job was submitted, and "temp-working-dir" is a job-specific temporary directory created by LPJS on the compute node. Following this transfer, the working directory on the submit node should contain the same output file as it would using a shared filesystem. Users can override the transfer command. command. See the Research Computing User Guide for details.

```
# If we downloaded the input file, remove it now to avoid wasting time
# transferring it back.  By default, LPJS transfers the entire temporary
# working directory to the submit node using rsync.
if [ ! -e $marker ]; then
    rm -f $infile
fi
```

Chapter 7

Advanced configuration

TBD