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 common tasks.

This manual is aimed at the systems manager, covering installation and configuration of LPJS. For a user's guide on LPJS use, 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 almost 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/.

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.

However, munge must be configured and enabled as a service as well. Munge requires all nodes to have a shared munge key file, which is unique to your installation. It must be generated by you (see the munge documentation) 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 specs such as available cores and memory are determined automatically, and need not be specified in configuration files.

Most configuration can be done entirely using **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.

Each compute node requires a similar 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.

Chapter 4

Starting Daemons

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

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

Note The head node can also serve as a compute node, though this is not generally recommended. The head node should remain lightly loaded so that it can respond in real time 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.

Jobs can be submitted from any node with the same version of LPJS and the shared munge key installed. It need not be running LPJS daemons, but it does require a configuration file point to 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.

4.1 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.

lpjs admin

4.2 Ad hoc Clusters and Grids

It is also possible to use LPJS without having admin access. 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 that submitted them.

The **lpjs ad-hoc** command displays a menu, allowing you to start and stop the apporopriate 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, 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 standard Ethernet, for the sake of much lower latency per message, and higher throughput. All nodes in a cluster typically have direct access to the same files on file servers using NFS, parallel file systems, or the like.

A *grid* is conceptually like a cluster, in that it is used for parallel computing. However, grids are more loosely coupled, often including computers that are not on the same LAN. Hence, a grid is not as good for parallel computing that involves a lot of communication between processes, such as MPI (Message Passing Interface) distributed parallel programs. Grid nodes typically do not share access to file servers, since this is inefficient and difficult to secure over the slower, wider networks used by grids.

One of the most important factors in network performance, besides the network hardware used, is the number of "hops" between two computers communicating with each other. Computers on the same LAN are 1 hop away from each other, meaning there is only one piece of network equipment between them. Computers a thousand miles from each other may by a dozen or more hops from either other. That is, packages of information must pass through many network devices such as switches and routers along the way. This significantly slows down communication when using protocols such as TCP/IP, which requires acknowledgment of correct transmission for each hop, before the next hop is attempted. While your PC's network interface is probably capable of 1,000 megabits per second, and you may approach this speed transferring data across a LAN, you may only see a few megabits per second when transferring data from another part of the continent.

LPJS is very flexible with network topology. The only requirement is that all nodes are able to contact 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, or cloud instances in a data center a thousand miles away. You must consider how each of these resources can effectively be used.

Clusters normally have some sort of file server, so that jobs run in a directory that is directly accessible from all nodes. This is the ideal situation, as inputs are available to jobs, and outputs from jobs are available without needing to transfer files.

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 file server accessible 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 jobs are rerun and some input or output is the same as before.

If the working directory is not on a filesystem shared by the submit node and compute nodes, then the user's script is responsible for downloading any required input files. LPJS will, by default, transfer the entire temporary working directory back to the submit node, using **rsync -av temp-working-dir submit-host:working-dir**, where "working-dir" is the directory from which the job was submitted. 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 does not exist on the compute node, then **lpjs_compd** and **chaperone** will take the necessary steps to create a temporary working directory and transfer it back to the submit node after the script terminates.

Note All compute nodes must be able to perform a passwordless **ssh** login to the submit node in order for this file transfer to succeed. This requires 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.

Chapter 6

Advanced configuration

TBD