EXTOLL: EASY X-INTERFACE TO LOADLEVELER FOR PARALLEL JOBS ON THE NIST IBM SP2

KARIN A. REMINGTON

MATHEMATICAL AND COMPUTATIONAL SCIENCES DIVISION, NIST

Note: *Extoll is based on the utility X-llcreate (released in May 1995), and incorporates changes made to the local parallel software environment since that time.*

1. Introduction. Extoll is a graphical user interface (GUI) for preparing parallel batch jobs for submission to the LoadLeveler system. The purpose of this utility is to provide a convenient and robust way to generate a LoadLeveler command file matched to a specific parallel program, without requiring the user to manipulate a complicated script for setting up the parallel environment remotely.

Since both PVM and MPI generally require the user to play an interactive role in the set-up of the parallel environment, a job submitted through LoadLeveler must mimic the steps a user would take in an interactive environment. This is accomplished via shell commands, which are invoked when the LoadLeveler job becomes active on the queue. The Extoll utility generates a appropriate LoadLeveler command file to invoke the environment control tool (PVM or POE), set the correct number of processors, copy files to/from the local node address space, and start the user's MPMD or SPMD program(s) with the correct arguments and/or input redirection. The generated command file is also "customizable", by allowing for user provided setup and cleanup scripts to handle data and output file management. By providing easy, modular customization, the utility encourages users to continually leverage system supported command files, thus avoiding programming delays when system software is upgraded.

With the functionally of the system maintained and supported scripts, the "point and click" style X-interface, and the provision for customized setup and cleanup, Extoll has been designed to lighten the burden of submitting parallel jobs to batch queues on the NIST IBM SP2. For users without X-windows access, extoll can be used without the X-interface. In this "line oriented" mode, the user is prompted by a series of questions about their program. After the information is collected, a LoadLeveler command file is generated.

2. Getting started with Extoll.

2.1. Setting up your environment. To use the Extoll interface, you will need to set your DISPLAY environment variable on "danube", and enable permission for "danube" to open windows on your local display. To accomplish this, you may use the following commands:

On danube: (entered at prompt, or placed in .cshrc file)

```
setenv DISPLAY yourdesktop:0.0
```

On your desktop machine:

```
xhost +danube.nist.gov
```

Because PVM depends on environment variables set in your .cshrc file to determine its path and architecture type, you may require some modifications to this file if you plan to use PVM. Normally, you could simply set the proper environment variables once and for all in your .cshrc file by adding the following commands:

```
setenv PVM_ROOT /usr/local/apps/pvm3
setenv PVM_ARCH RS6K
```

However, if you would like to use the high-speed switch implementation of PVM (SP2MPI), or would like to alternate between using the standard ethernet version and the switch version, the variable settings in your .cshrc file need to be more creatively managed. The following snippet can be included in your .cshrc file to set these variables in a appropriate fashion so that Extoll can properly control the PVM environment you select:

```
0  unsetenv PVM_ROOT
1  unsetenv PVM_ARCH
2  if ( $?LL_JOB ) then
3     source $HOME/.llpvmstub
4  else
5     setenv PVM_ROOT /usr/local/apps/pvm3
6     setenv PVM_ARCH RS6K
7  endif
```

The file ".llpvmstub" (line 3) sets the PVM_ROOT and PVM_ARCH variables according to your LoadLeveler command script, and will be copied into your HOME area as it is needed. For non-LoadLeveler jobs, the default settings for PVM_ROOT and PVM_ARCH (lines 5 and 6) can be edited as desired.

2.2. Invoking Extoll from the executable directory. This utility should be run from the directory where your *executables* are stored. If you do not call extoll from that directory, use the "cd" button at the bottom of the Extoll window to navigate to your executable directory.

2.3. Mouse buttons and keyboard entry . Extoll accepts input primarily in the form mouse button clicks over menus or windows and typed keyboard input in entry fields.

- The left mouse button can be used to "press" Extoll buttons. It can also be used to select (highlight) text for pasting to other locations.
- The right mouse button, if depressed over an entry field, will paste selected text into that field. The right mouse button is also used to kill a persistent subwindow (for example, a "Help" window).
- The center mouse button (or left and right buttons simultaneously on a two-button mouse) is used to scroll to the end of entry fields if the entered data extends beyond the entry field window.

2.4. Entering your program information into Extoll . As you begin, you should be prepared with basic information regarding your program. Carefully read through each item in the main Extoll window to determine the specific information you will need. See Fig. 5.1 (p. 8) for a sample view of the main window.

After becoming familiar with the queries, go through the window from top to bottom responding to each query as it applies to the program you wish to submit to LoadLeveler. Note that certain entries will not apply under all circumstances, and will be disabled if they are not required. In addition, default values are provided for all fields except the name of the executable program itself.

Most of the queries and buttons should be self explanatory. Additional help is available on the main action buttons (at the bottom of the Extoll window), and some advanced features including customized file space management.

2.5. Saving your entries and submitting a job. After your have entered the required information, you may submit the job to the LoadLeveler batch system, or create/update a command file based on this input without immediately submitting the job. The buttons at the bottom of the Extoll window labeled "Submit Job" and "Create *.cmd File" perform these functions. The current entries in the Extoll window may also be saved for later use by pressing the "Save Settings" button. These settings are saved to the file .extollrc in your executable directory, and will be used as default values for any subsequent runs of extoll. Note that the contents of the .extollrc will be ignored if the current version of extoll does not match the version which created the .extollrc file.

3. Action buttons.

3.1. Submit Job: Submit the program to the LoadLeveler queue. After completing entry of your program and job characteristics, you may submit the job to the LoadLeveler system by pressing the "Submit Job" button at the bottom of the Extoll window. You will be prompted for whether you would like to generate a new *.cmd file. If you decline, the current *.cmd file will be used, and recent modifications in the Extoll window will not take effect.

3.2. Create *.cmd File: Create a "customized" LoadLeveler command file. After completing entry of your program and job characteristics, you may create a job specific LoadLeveler command file by pressing the "Create *.cmd File" button at the bottom of the Extoll window.

The file generated will be named "<program>.cmd", where <program> is the name of the (main) executable you have specified. If a file with that name already exists in the current directory, you will be prompted for overwrite permission. If you do not select for this file to be overwritten, a temporary file will instead be created with the filename "<program>.cmd.tmp". After creation, you are of course free to change the names of these files. However, the "Submit" facility of Extoll submits only the file named "<program>.cmd".

The generated "<program>.cmd" file contains basic LoadLeveler keyword setting, and exports environment variables for use with customized user setup and cleanup scripts.

3.3. Save Settings: Restarting without re-entering information. If your Extoll session is interrupted, you may save the settings you have entered for later use. The "Save Settings" button at the bottom of the Extoll window creates (or overwrites) the file ".extollrc" in the current directory. Extoll searches for this file upon start-up, and if it is present, the values saved will be read in and displayed when the Extoll window is created.

3.4. cd: Change the executable directory. It is *vital* that the Extoll working directory be set to the directory where your executables reside. If you invoke extoll from that directory, this will happen automatically. If you invoked extoll from some other directory, you must change its working directory to match your executable directory. This should be the first thing you do upon starting up extoll!

There are two ways to change the working directory of Extoll. One way is to quit Extoll and reinvoke it in the proper directory. An alternative is to use the “cd” button at the bottom of the Extoll window. If you change the working directory in this way, any files created by Extoll (<program>.cmd, .extollrc, etc) will be located in the new working directory.

Note: Global vs. Local cd: The main “cd” button will make a “global” change of directory in the Extoll window. The buttons associated with file entries (such as the ”executables” button under the “Main (manager)” program entry) will present lists of candidate entries based on this working directory. If you should require input files from a different directory, you should make a “local” directory change, with the “local cd” menu item which can be reached by pressing the “data files” button associated with the entry field for input files.

4. Selecting input files. The entry field for input files (that is, files explicitly opened by name in your program) will accept multiple filenames. You may enter your input files by typing directly into the entry field, or, as an alternative, the “data files” menu will allow you to navigate to your input files, bring up a list of files, select files with the mouse, and drop the selected files into the entry field.

The “navigate” button will bring up a list of files in the currently selected directory. You may select an input file from this list by highlighting it with the left mouse button and then clicking on ”Ok”, or you may navigate to another directory by double clicking on the desired directory. Repeat the navigation step for each input file. The entry window can be scrolled by “grabbing” the text with the center mouse button and “dragging” it across the window.

The select input files will be copied “flat” to the working directory. That is, subdirectory structure will NOT be preserved. If your programs have I/O which assumes a particular file structure, you will need to set up the appropriate directories with a customized script. See Section 5.

5. Using custom setup and cleanup scripts . By default, when each of the generic scripts runs, a local directory /tmp/$USER is created on each of the nodes. This directory serves as the “working directory” for the main and secondary executable programs. Under the default conditions, executable programs, a redirected standard input file (if required), and any files which are declared to be opened explicitly by name in your program are copied to this temporary directory. At the end of the job, this directory is removed from each of the nodes.

What if your program requires a more complicated temporary directory structure than the flat /tmp/$USER model, or, if you wish to save node output files to your permanent filespace before the temporary directory is removed? Because it is impossible to predict individual requirements, Extoll allows for optional customized user ”setup” and ”cleanup” scripts. These scripts can be written in the shell language of your choice, and have access to the environment variables of the command file which calls them.

5.1. Effective working directory during your job. To construct your custom scripts, you need to be aware of the effective working directory when each script is called:

custom setup: The effective working directory is the directory in your **$HOME** area where your executables are located. This directory is specified by the LoadLeveler keyword “ initialdir” in your “<program>.cmd” file.

program execution: The effective working directory is the directory “/tmp/$USER”. Any required custom directory structure should be setup relative to this path. For example, if the main or secondary program opens the file “program.input”, it will look for the file “/tmp/$USER/program.input”. If the program(s) opens the file “output/program.output”, the custom setup must create the directory “/tmp/$USER/output”.

custom cleanup: The effective working directory is still the directory /tmp/$USER. You can copy to the original directory in your home space using the environment variable **$INITIALDIR**.

5.2. Environment variables within custom scripts. Your custom scripts have access to the following environment variables describing your job:

Variable:	Value:
USER	‘whoami‘
CLUSTER	4-digit batch process id for your job
HOST	master process host (grand1, for example)
INITIALDIR	initial working directory of the LoadLeveler job
$CLUSTER.hostlist	(PVM) contains the hostlist generated for your job
PVM_PROCESSOR_LIST	list of all the nodes available to your job
MPI_PROCESSOR_LIST	

5.3. Sample custom scripts.

A sample ksh custom setup script:

```
#!/bin/ksh
#---------------------------------------------------------------------
#  File:     user.setup
#  Purpose:  Copy input files to the PVM nodes in the processor list
#---------------------------------------------------------------------
echo "${0##*/} ($HOST): The setup script is running in :" `pwd`
echo "${0##*/}: The setup script has the following variables set:"
echo "${0##*/}: LL_JOB: $LL_JOB"
echo "${0##*/}: CLUSTER: $CLUSTER"
echo "${0##*/}: PVM_PROCESSOR_LIST: $PVM_PROCESSOR_LIST"

# count the number of nodes in the processor list:
(( nprocs = 0 ))
for node in $PVM_PROCESSOR_LIST
do
  (( nprocs = nprocs + 1 ))
done

echo "${0##*/}: Number of available PVM nodes: $nprocs"

#
#  Copy some portion of the task queue input to each node:
#
for node in $PVM_PROCESSOR_LIST
do
  echo "${0##*/}: Here is where input is copied to node $node..."
done

echo "${0##*/}: Finishing the user.setup script."
```

A sample ksh custom cleanup script:

```
#!/bin/ksh
#----------------------------------------------------------------------
#  File:     user.cleanup
#  Purpose:  Copy back a file of results from the master PVM process
#            node directory, and worker output files from each node
#            to the shared filespace under the user's $HOME directory.
#----------------------------------------------------------------------

#  Copying back results file from master PVM process....
cp $TMPDIR/manager.results $INITIALDIR/manager.results

for host in $hostlist
do

#  The node output file ``nodeoutput" from each host is copied back to an
#  output subdirectory, with an extension that indicates which node created
#  the outputfile.

   rcp $host:$TMPDIR/nodeoutput $INITIALDIR/output/nodeoutput.$host
done
```

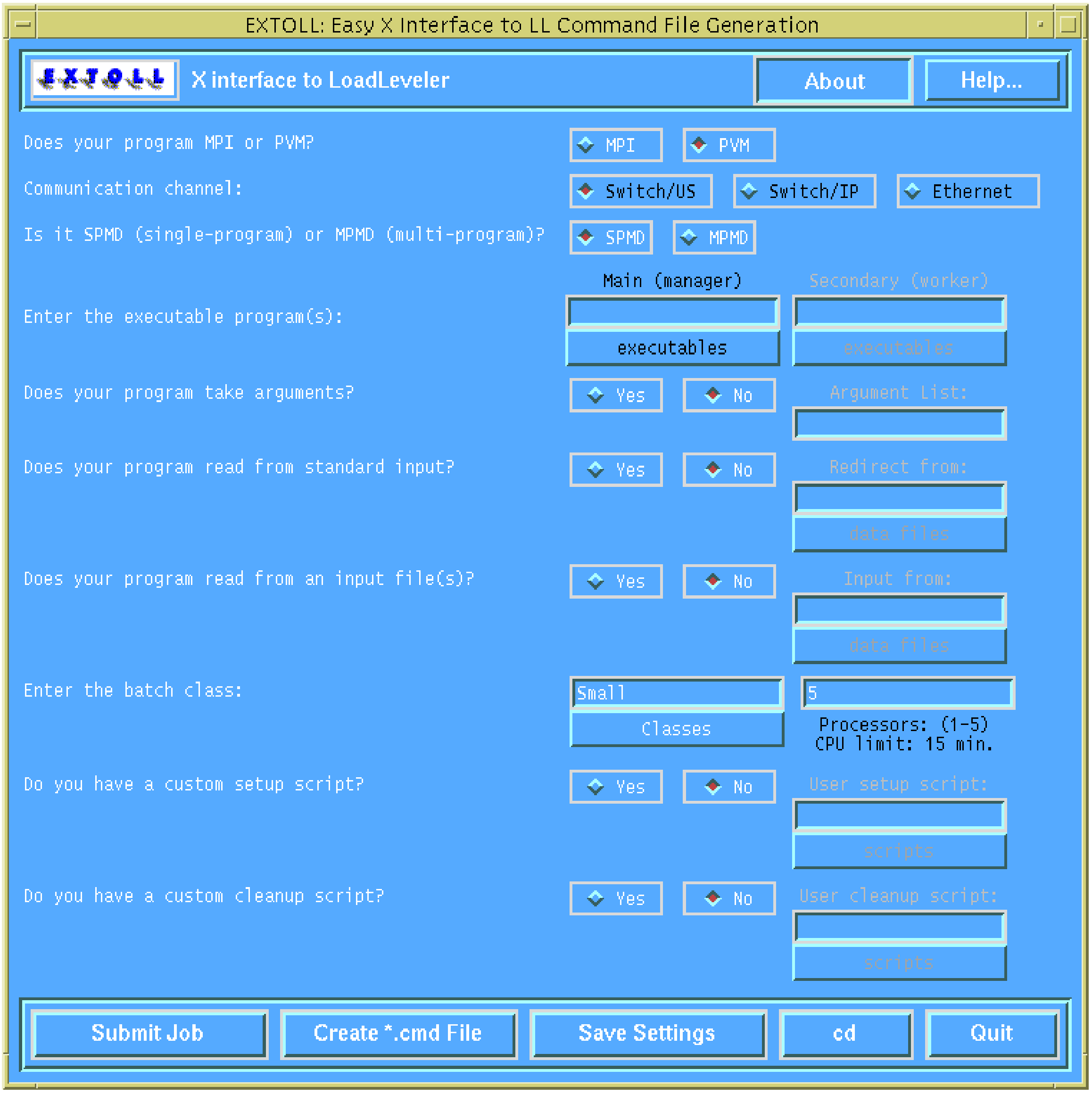

FIG. 5.1. *Sample view of the Extoll interface*