LoadLeveler

Workload Management for Parallel and Distributed Computing Environments

Anton A. Prenneis, Jr.

prenneis@vnet.ibm.com

IBM RISC System/6000 Division

POWERparallel Systems

Poughkeepsie, NY 12601, USA

October 11, 1996

LoadLeveler

- Product Highlights

- Functionality

  - Building Blocks of LoadLeveler
  - What happens when I submit a serial job?
  - What happens when I submit a parallel job?
  - Interactive Session Support (ISS)

- Design Features of LoadLeveler 1.3

- Future Directions

Product Highlights

A Brief History...

- LoadLeveler is based on the Condor scheduler from the University of Wisconsin.

- IBM Research obtained the public domain Condor source code and made significant modifications, including user-based priority scheduling, NFS and AFS support, and a Motif-based GUI.

- LoadLeveler is now a fully licensed IBM program product, with complete product documentation and service support.

Some Key Benefits...

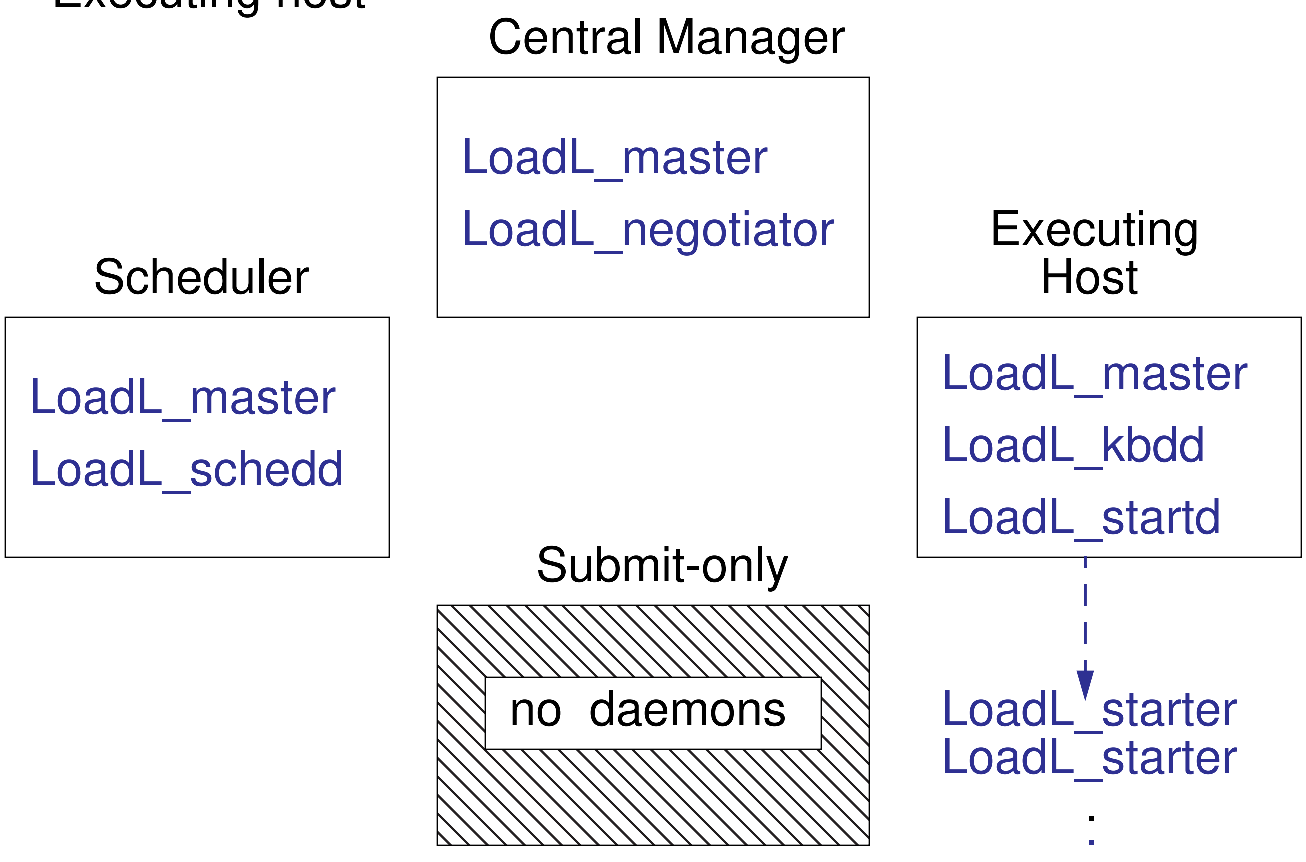
- Treat heterogeneous clusters as a single managed resource.
- Balance batch and interactive workload.
- Capture unused workstation capacity.
- Implement flexible job scheduling policies.
- Establish resource limits for users, job classes and machines.
- Collect comprehensive accounting information.
- Allow users limited administrative capability.
- Scalable to hundreds of nodes.

Building Blocks

Machines

There are four basic **machine types**:

- Central Manager
- Submit-only
- Scheduler
- Executing host



Default machine configurations are defined in a cluster's global LoadL_config file. Individual machine characteristics can be further customized in the global LoadL_admin file and in each machine's LoadL_config.local file.

Building Blocks

Jobs

There are four basic **job types**:

- Single-step
- Multi-step
- Multi-step with dependencies
- Parallel

Jobs are submitted to LoadLeveler in the form of **job command files**. Job command file expressions are specified as shell comments, which gives the user the option of submitting a job command file that is itself an shell script to be executed.

e.g.

```
#!/bin/ksh
# @ executable = myprogram
# @ step_name = step1
# @ initialdir = /afs/mycell/u/myhome
# @ output = $(executable).out
# @ error = $(executable).err
# @ notify_user = myself@myhost.ibm.com
# @ requirements = (Arch == "R6000")
# @ class = small
# @ queue
# @ dependency = (step1 == 0)
# @ executable = mycleanup
# @ queue
```

Job classes and **requirements** are used to help the central manager match jobs with appropriate machines.

Building Blocks

LoadL_admin File

Characteristics of *machines*, *job classes*, *users* and *user groups* can be customized by the administrator via **stanzas** in the central LoadL_admin file. Stanza formats are as follows:

```
label: type = machine
       alias = machine_name
       central_manager = true | false | alt
       cpu_speed_scale = true | false
       master_node_exclusive = true | false
       max_jobs_scheduled = integer
       name_server = list
       pvm_root = pathname
       rm_host = true | false
       schedd_host = true | false
       speed = number
       submit_only = true | false

label: type = class
       admin = list
       class_comment = "string"
       exclude_groups = list
            .
            .
            .
       job_cpu_limit = hardlimit,softlimit
       cpu_limit = hardlimit,softlimit
       wall_clock_limit = hardlimit,softlimit
```

Building Blocks

LoadL_admin File (cont.'d)

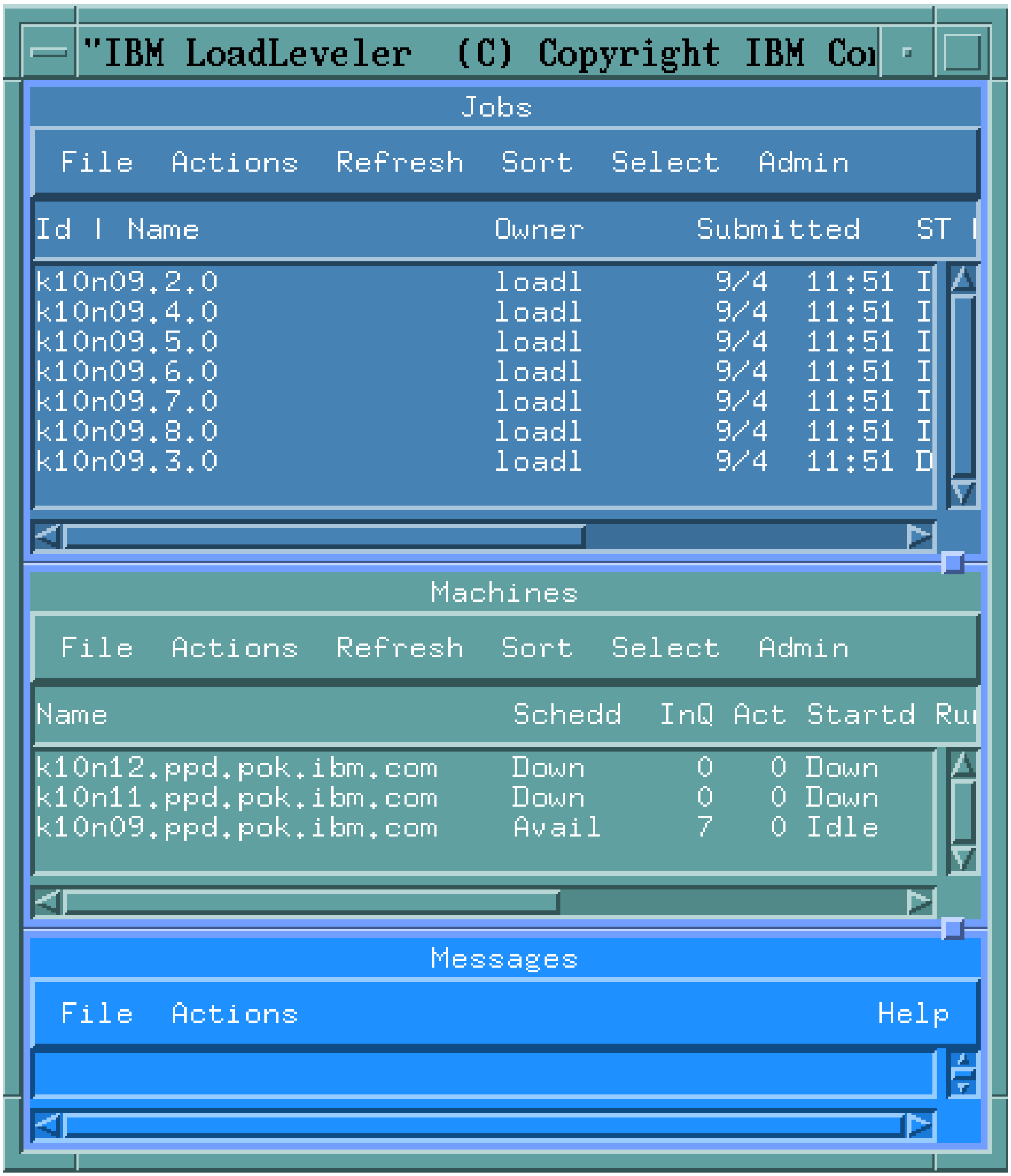
```
label: type = user
       account = list
       default_class = list
       default_group = group_name
       maxidle = integer
       maxjobs = integer
       maxqueued = integer
       max_processors = integer
       priority = integer


label: type = group
       admin = list
       exclude_users = list
       include_users = list
       maxidle = integer
       maxjobs = integer
       maxqueued = integer
       max_processors = integer
       priority = integer
```

Note the *admin list* that can be specified for the class and group stanzas. This allows the LoadLeveler administrator to give one or more users the authority to *hold*, *release* and *cancel* jobs of a certain class or jobs that were submitted by users in a certain group.

Building Blocks

The Graphical User Interface



Building Blocks

Accounting

- Accounting for Serial and Parallel Jobs

- All processes for a serial job step and all tasks of a parallel job are tracked and their resource usage is reported cumulatively.

- Accounting data from all nodes in the cluster can be gathered from a single node via the **llacctmrg** command.

- The **llsummary** command can be used to generate reports of varied detail, and custom reports can be generated using LoadLeveler’s **Accounting API**.

- Interval accounting can be gathered on *running jobs* with the **-x** option of the **llq** command.

Building Blocks

Checkpointing

- *User Initiated Checkpointing*

  Provided with the **ckpt()** API.

- *System Initiated Checkpointing*

  Checkpoints are taken by LoadLeveler at intervals, defined by the administrator via MIN_CKPT_INTERVAL and MAX_CKPT_INTERVAL.

- Both types of checkpointing require user applications to be linked with LoadLeveler’s **checkpointing libraries**.

Building Blocks

User Commands

llcancel - cancels one or more jobs in the job queue

llclass - returns status information about job classes

llhold - places jobs on hold and releases held jobs

llq - returns information about jobs in the job queue

llprio - prioritizes a user’s jobs against each other

llstatus - returns information about machines in the cluster

llsubmit - submits jobs to LoadLeveler

Building Blocks

Administrator Commands

llctl - command suite which allows the administrator to:

- start, stop, recycle, reconfigure, purge machines
- drain/resume schedd's, startd's and job classes
- flush/resume schedd's and startd's
- suspend jobs
- capture accounting data on machines

llfavorjob - raises or lowers job priorities

llfavoruser - raises or lowers user priorities

llacctmrg - collects accounting data from all nodes in cluster

llsummary - generates accounting reports

Building Blocks

Application Programming Interfaces

- Accounting
  - Account Validation
  - Report Generation

- User Initiated Checkpointing

- Job Submit and Monitor

- Parallel Job Enablement

- Query Jobs and Nodes

- Scheduling

Administrator and User Exits

- **.llrc**

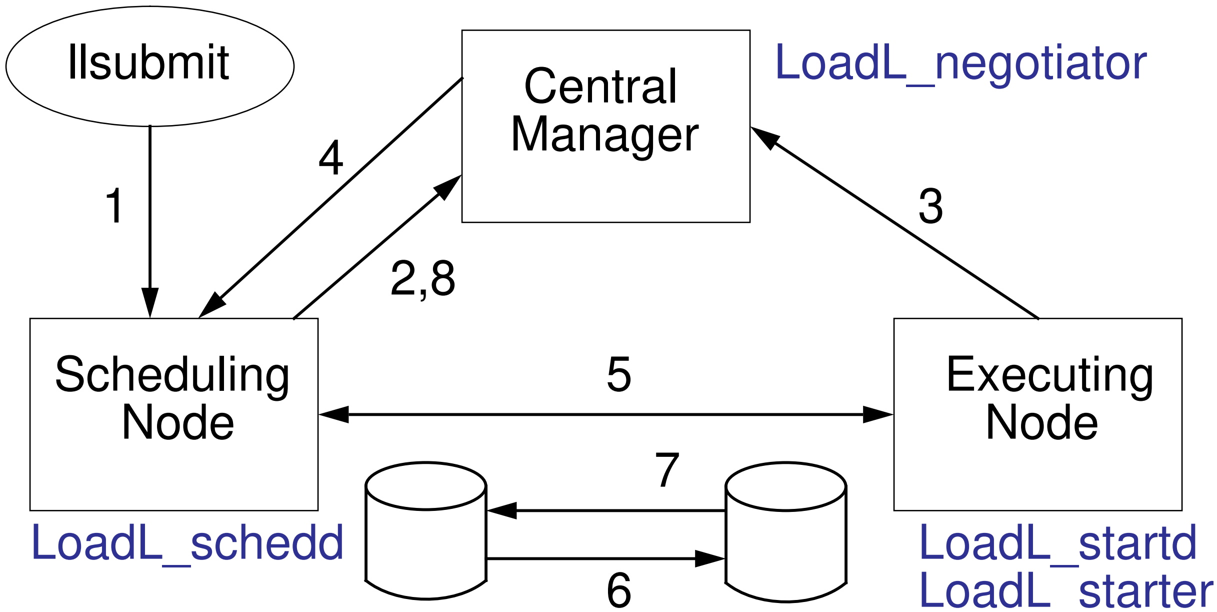
- Job Prolog and Epilog

Building Blocks

High Availability

- Alternate Central Managers
- Persistent Job State Information is stored on both scheduling and executing machines.
- Backup ISS Monitors
- HACMP (*High Availability Cluster Multi-Processing*) can be also be used to provide redundancy for cluster nodes. See ITSO Redbook *SG24-4742-00*.

Serial Job Scheduling



What happens when I submit a serial job?

1. User submits job which is queued locally on a scheduling node by its LoadL_schedd.
2. LoadL_schedd sends job requirements to the central manager, where the LoadL_negotiator queues the job in the LoadLeveler job queue.
3. A LoadL_startd reports machine info. compatible with job.
4. Central manager informs scheduling node that the executing node is available.
5. LoadL_startd on executing node spawns a LoadL_starter process, which the LoadL_schedd will connect to.
6. Executable is passed from scheduling node to executing node, which stores it on disk and executes it.
7. LoadL_starter reports job status to startd and schedd.
8. LoadL_schedd reports job status to central manager.

Parallel Job Scheduling

What happens when I submit a parallel job?

The Negotiator uses **temporary reservation** to select nodes to run a parallel job on:

- Scans available machine list for nodes with resources matching job requirements.
- If a sufficient number of nodes is not available, reserves as many as it can.
- Continues to reserve new nodes meeting job requirements as they become available.
- Dispatches the job when required number of nodes have been reserved. The **resource manager** can then be used by the master node for actual node allocation.
- Limits the amount of time nodes are held in reserve ("NEGOTIATOR_PARALLEL_HOLD"). After this time elapses:
  - All reserved nodes are released.
  - Parallel job is placed on a **deferred** queue.
  - Parallel job reenters the LoadLeveler queue after "NEGOTIATOR_PARALLEL_DEFER" has elapsed.
  - Parallel job reenters queue with original system priority.

Parallel Job Scheduling

LoadLeveler Parallel Job Support

Valid Version Combinations on the SP

Product	**Compatible Versions**		
LoadLeveler	1.2.0	1.2.1 , 1.3.0	1.3.0
AIX	3.2.5	4.1.3 , 4.1.4	4.1.4
PSSP	1.2.0	2.1.0	2.2.0
PE	1.2.1	2.1.0	2.2.0
PVM	3.3.0	3.3.0	3.3.11+ *
PVMe	1.3.1	2.1.0	2.2.0

* ***Compile PVM 3.3.8 - 3.3.10 programs for the RS/6000 when they are targetted to run on SPs running LoadLeveler.***

- *Suspend*, *Flush* and *Checkpointing* are presently unsupported for parallel jobs.

- *PVM/PVMe Dynamic Allocation and De-allocation* of parallel nodes is presently unsupported.

- **Parallel APIs** are provided for use with other parallel programming environments.

Interactive Session Support

How does ISS balance login sessions?

- ISS manages one or more **ISS Pools**. Each pool has a unique hostname **alias** that is known to the nameserver (e.g. isspool1.kgn.ibm.com).

- ISS runs **issmonitor** on the nameserver machine**.**

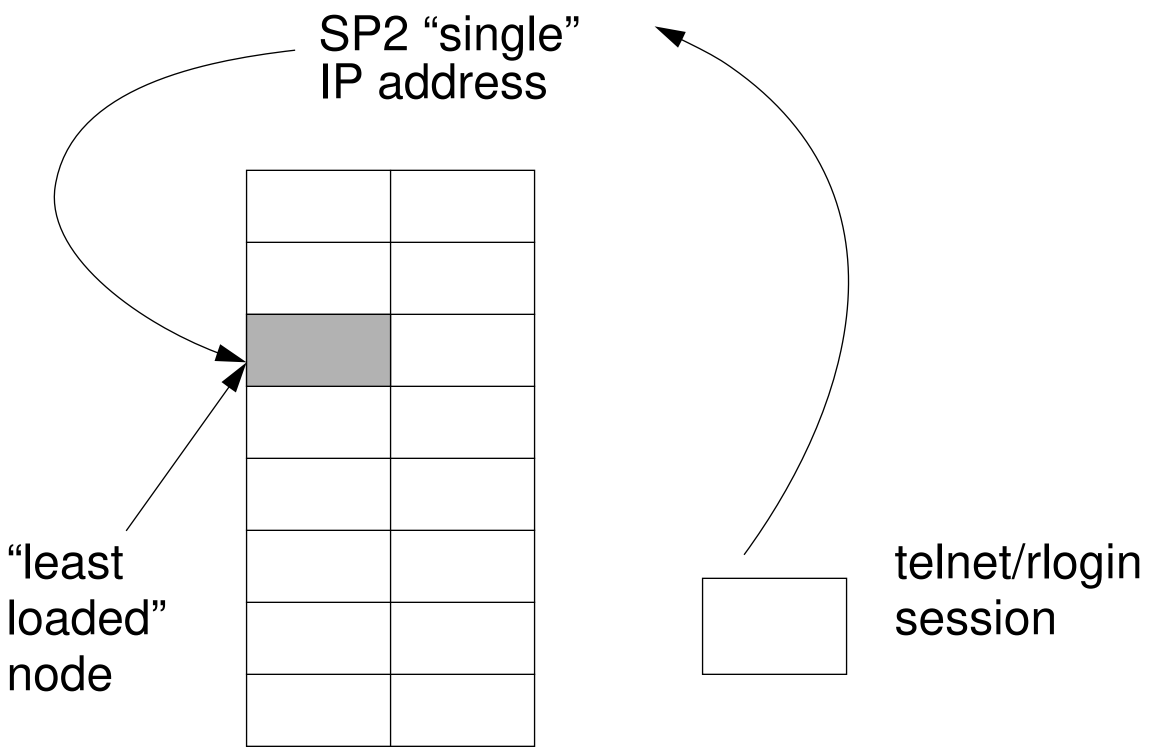
- At a pool's **polling interval**, issmonitor will tell the named daemon to update the pool's alias with the IP address of the least loaded node.

- When an ISS user opens a connection to the pool's alias via rlogin or telnet, the named resolves the alias to the new IP address.

- One of three **metrics** can be chosen to determine machine load:
  - LoadLeveler metric
  - Custom metric
  - Roundrobin metric

Interactive Session Support

Interactive Login Session



User initiating an interactive session on SP2 telnets to a “single name” for SP2 defined by administrator

ISS interacts with the standard nameserver in resolving the single name to the IP address of the “least loaded” SP2 node

Telnet completes with user logged into “least loaded” node

Interactive Session Support

ISS Configuration File

(/usr/lpp/LoadL/iss/iss_config)

- NAME - Defines the pool's hostname alias.
- SERVERS - Specifies the list of machines in the pool.
- METRIC - Specifies the pool's metric, either:
  - LOADLEVELER
  - ROUNDROBIN
  - CUSTOM
- POLICY - Used with the custom metric to indicate whether the metric function will be minimized or maximized.
  - MIN - appropriate, for example, for number of processes, ‘ps -ef | wc -l‘
  - MAX - appropriate, for example, for CPU idle time ‘vmstat | grep [0-9] | cut -c63-64‘

LoadLeveler 1.3

Design Improvements

- Daemon Framework
- GUI Enhancements
- Wall Clock Limit Support
- EASY and the Scheduling and Query APIs
- Drain by Class Function
- More Fairness Built Into Priority Calculations
- New Scheduling Controls
- ISS 1.3 Features
- Non-AIX Platform Support and Migration Considerations
- Documentation Resource List

Daemon Framework

LoadL_negotiator / LoadL_collector Merge -> LoadL_negotiator

- Job information used to make scheduling decisions is more up to date with a one daemon central manager architecture.

Daemon Threading

- Improved command response and job scheduling decisions from LoadL_negotiator.
- Quicker and more reliable processing from LoadL_master, LoadL_schedd and LoadL_startd.

Daemon Class Framework

- Startd, Schedd, Master and Negotiator daemons use common C++ classes, simplifying software testing and enabling code reuse.
- Transaction class enables more reliable daemon communications. Failed transactions are retried with those deemed critical being retried forever.

GUI Improvements

- Customizable menu pulldowns
  - Field lengths customizable
  - User defined pulldown menus

- Primary and secondary sorting available for *Jobs* and *Machines* windows

- Wildcard support in selection fields

  (e.g. Type “**ll***” to list machine names starting with ll.)

- Sorting and selecting actions against a group

  (Group column can now be displayed in the *Jobs* window.)

- Specification of multiple inputs

  (e.g. View jobs submitted from machines A and B.)

Wall Clock Limit

- A new limit to control the execution time of a job in terms of seconds spent running on a machine, rather than cpu time.
- A new keyword *wall_clock_limit* specifies the maximum elapsed time that a job can run.
- Can be specified in class stanza or in job command file.
- Syntax:

  wall_clock_limit = *hardlimit*,*softlimit*

***New data fields in “llq -x -l” and in the mail message*:**

- *dispatch_time* - the time the negotiator dispatched the job
- *job_start_time* - the time the starter fork()/exec()’d the job

Scheduling and Query APIs

- Provided to allow customers to use their own scheduling algorithm(s) while taking advantage of LoadLeveler's ability to manage resources in a cluster.

- LoadLeveler collects job and machine data and the customer's scheduling system requests the data from the LoadL_negotiator. The customer's scheduling system then tells LoadLeveler which jobs to run on which machines.

- The customer's scheduling system can be identified in the LoadL_config file.

- Used with great success on up to 300+ node parallel jobs with EASY (*Extensible Argonne Scheduling sYstem*) on the Cornell Theory Center's 512-node SP2 system.

Scheduling and Query APIs

EASY

- Deterministic queueing algorithm
- First-in-first-out
- Wall clock limit required
- *Backfill*
- More information about EASY:

  http://info.mcs.anl.gov/Projects/sp/scheduler/scheduler.html
- More information about EASY at Cornell:

  http://www.tc.cornell.edu/er/media/1996/easy-II.html

Scheduling and Query APIs

API Functions

- *ll_get_jobs()* - Returns job status on all jobs currently in the LoadLeveler job queue

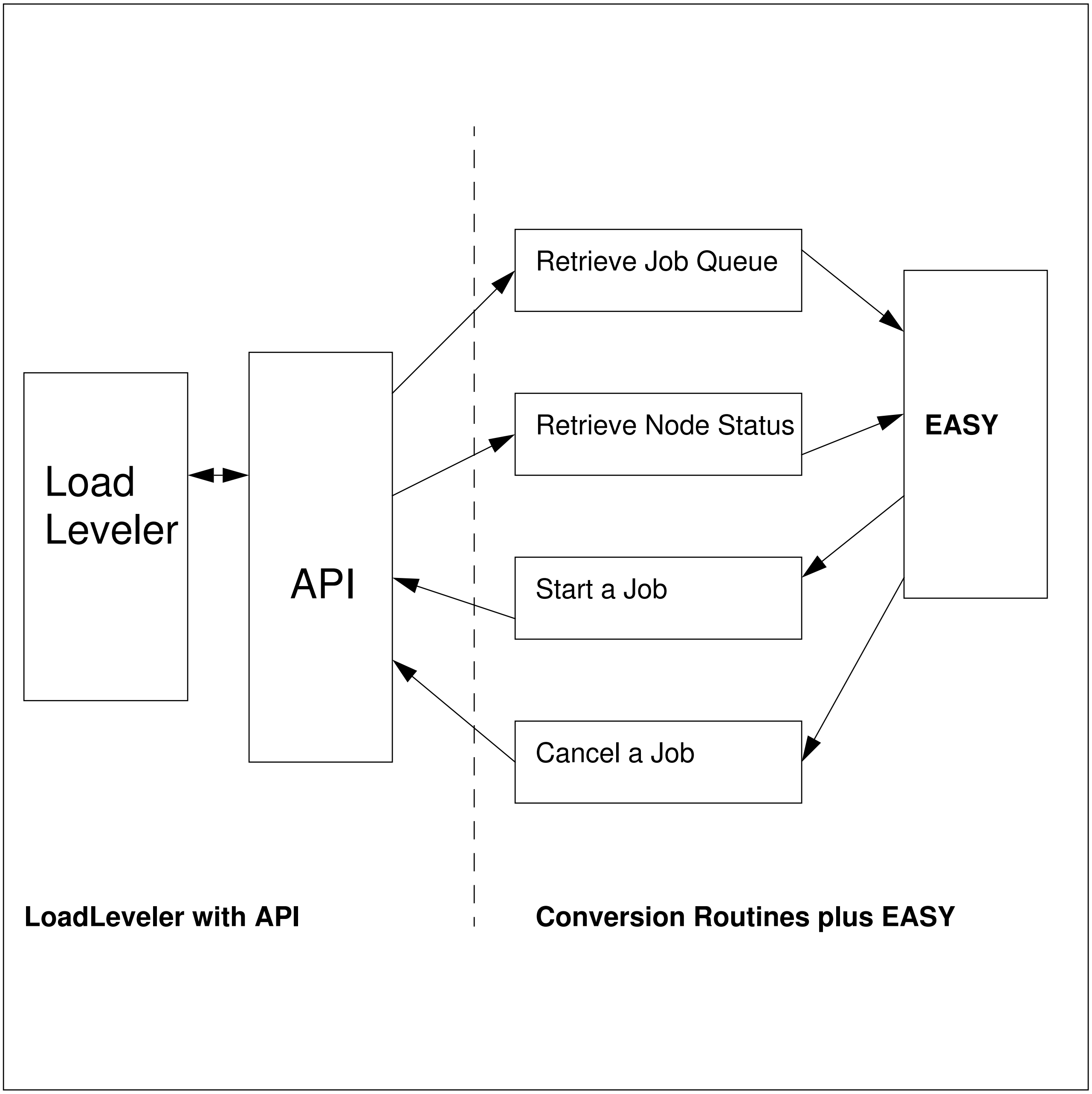
- *ll_get_nodes()* - Returns node information on all machines currently in the LoadLeveler machine list including such data as *loadavg*, *adapters*, and *memory*

- *ll_start_job()* - Tells LoadLeveler to start a specific job on a set of nodes

- *ll_terminate_job()* - Terminates an idle or running job

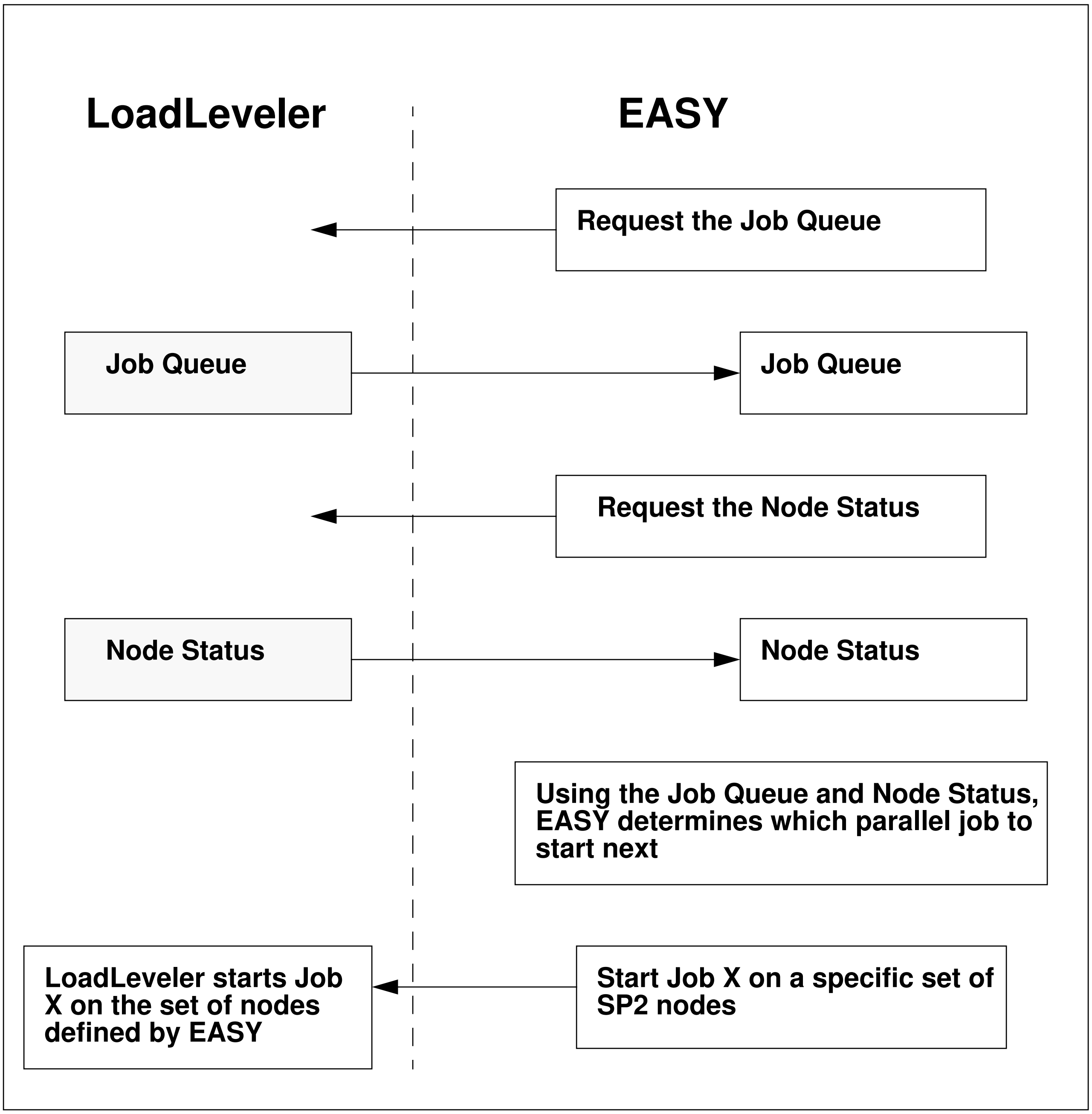
Scheduling and Query APIs

API Functions



Scheduling and Query APIs

Job Scheduling Flow



Scheduling and Query APIs

Benefits

- Customers can make their own scheduling decisions.

- Provides a base for future enhancements which include allowing installations to:

  - Switch scheduling algorithms depending on time of the day,
  - Run different algorithms on subsets of nodes in a LoadLeveler cluster.

- EASY scheduler only needs to deal with a customer's scheduling decisions, leaving the task of constantly maintaining cluster status to LoadLeveler.

- LoadLeveler becomes more flexible, allowing customers to implement *any* scheduling algorithm without having to wait for the IBM developers to build it into the LoadLeveler system.

Drain by Class Function

llctl ... drain [schedd | startd [<classlist> | allclasses]]

- Prior to 1.3, administrators could drain the LoadL_schedd and the LoadL_startd, allowing them to complete all their current work while refusing to accept new work.

- Now the administrator has the additional ability to *drain a job class* on all LoadL_startd's in the cluster, or on just a subset of LoadL_startd's.

- This provides the benefit of shutting down LoadLeveler for particular job classes while still scheduling new work from other job classes to run.

- The major reason an administrator might want to do this is to assist load balancing by draining certain job classes at certain times of day. Another reason might be to assist in problem determination related to a particular job class.

Scheduling Fairness

New SYSPRIO Parameters

- UserQueuedJobs -

  total of *idle* and *running* jobs queued for a user

- UserTotalJobs -

  total number of jobs in any state queued for a user

- GroupQueuedJobs -

  total of *idle* and *running* jobs queued for a user group

- GroupTotalJobs -

  total number of jobs in any state queued for a user group

Now the queue priority of a job can be adjusted “real time” in proportion to the amount of resources being used by the owner of that job, or collectively by the users in the owner’s user group.

New Scheduling Controls

Job Queue Control Keywords

Two LoadL_admin file keywords have been introduced to help ease the scheduling decision load on the LoadL_negotiator and to insure that one user will not be able to monopolize a majority of cluster resources:

- maxidle (can be specified for a *user* or a *user group*) -

  Maximum number of jobs that can be *Idle* in the job queue. Any additional jobs will remain in *NotQueued* (**NQ**) state, meaning that the LoadL_negotiator will not consider them for dispatching until the number of Idle jobs drops below the maxidle value.

- maxjobs (can be specified for a *user*, *user group* or *class*) -

  Maximum number of jobs allowed to be running at any given time.

Additional Improvements

- The VACATE expression has been improved for parallel jobs so that a parallel job that is vacated will be returned to the LoadL_negotiator for rescheduling.

- *Job status queries* are now more informative. *llq -s* now provides more comprehensive information about the reasons a job might stay in *Idle*, *Deferred*, *Hold* or *NQ* state.

ISS 1.3 Features

ISS 1.3 Enhancements

- Custom Metric Improvements

  - Load Capping
  - issagent
  - Sample Custom Metrics

- Health Checks

ISS 1.3 Features

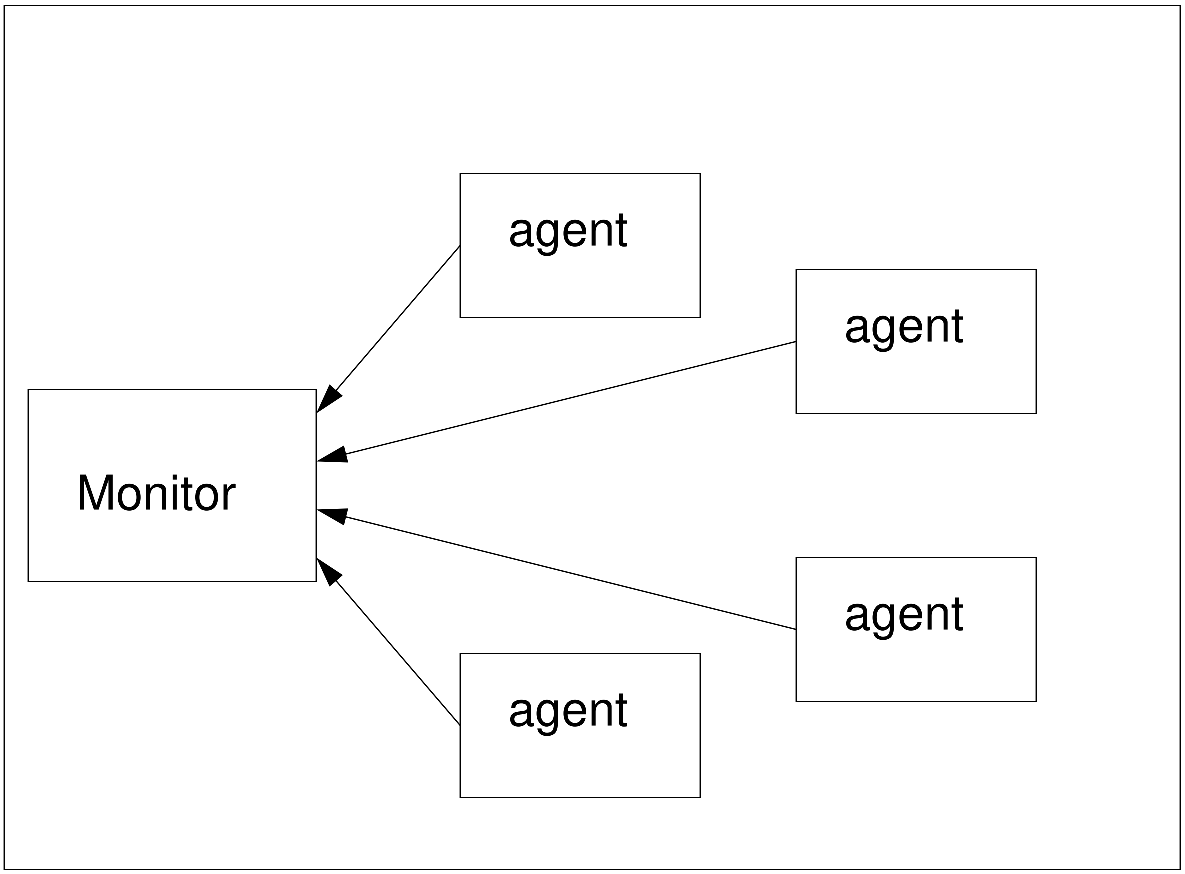
ISS Load Capping

- New Configuration File Keyword - LIMIT:
    - Applies a load cap to the servers in a CUSTOM pool.
    - If policy is MIN, limit is interpreted as an upper limit.
    - If policy is MAX, limit is interpreted as a lower limit.
    - Overflow server can be optionally defined to handle the extra load.
    - POOL_ALARM is triggered if overflow server is not defined.

      (The ISS **alarm facility** can be used with any metric to notify the administrator when a pool runs out of servers.)

ISS 1.3 Features

issagent



- *issagent* runs on each server in a custom pool, detecting changes in load and notifying the *issmonitor*.
- Can be set to **continuous** to reduce network traffic
- Eliminates of the use of *rsh*

ISS 1.3 Features

Sample Custom Metrics

- The number of users logged-on

  ‘w | wc -l‘

- The number of processes running

  ‘ps -ef | wc -l‘

- The load average

  ‘rup localhost | cut -c53-56‘

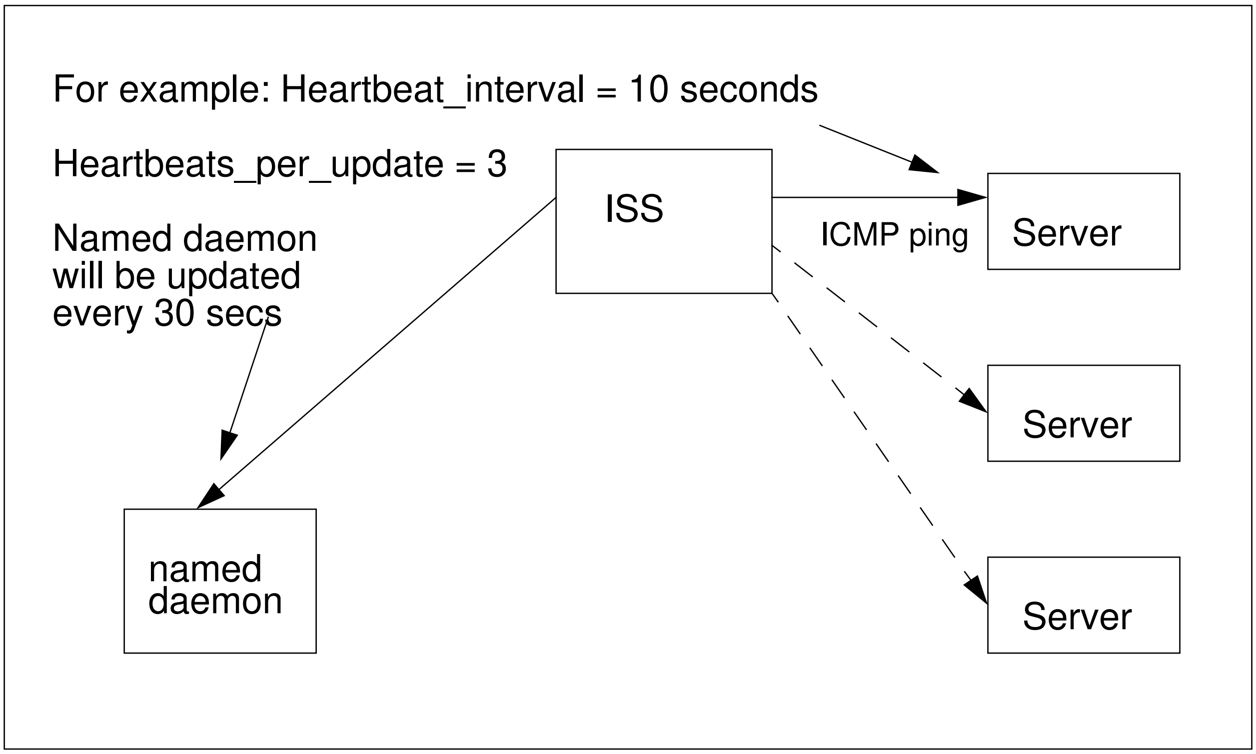
- The number of users connected via telnet

  ‘ps -ef | grep telnetd | grep -v grep | wc -l‘

- The number of running instances of a certain application

ISS 1.3 Features

Health Checks



HEARTBEAT_INTERVAL - indicates the time in seconds which ISS allows to elapse before it checks the well-being of the server

HEARTBEATS_PER_UPDATE - indicates the number of heartbeats after which ISS will determine whether a named daemon update is required

Non-AIX Platforms

LoadLeveler 1.2.1 is available on:

HP-UX 9.0.x (Product #5765-287)

SGI-IRIX 5.x (Product #5765-228)

Sun Solaris 2.x (Product #5765-227)

SunOS 4.1.x (Product #5765-227)

LoadLeveler 1.3.0 is currently only available in *submit-only* form on the following non-AIX platforms:

HP-UX 9.0.x and 10.x

SGI-IRIX 5.x and 6.x

Sun Solaris 2.x

(*SunOS is not supported beyond release 1.2.1.*)

Full LoadLeveler 1.3 function for Sun, HP and SGI will be available 4Q96.

You may obtain the above submit-only binaries by anonymous ftp to ***lscftp.kgn.ibm.com***, changing directories to

/pub/pps/loadleveler/SUBMIT_ONLY

and then changing to either the **V1.2.1** or **V1.3.0** subdirectories.

Non-AIX Platforms

Important Notes

- Checkpointing not supported on Solaris 2.3 or SGI-IRIX 5.2
- HP Apollo 9000 Series 700 only supported (not series 800)
- HP-UX only supports two resource limits: *job_cpu_limit* and *hard file_limit*. This is due to the lack of full *setrlimit()* support in HP-UX 9.0.x.
- AFS support is available for AIX only.
- IPC (*Interprocess Communication*) must be available on any Sun to run ISS.

Migration Considerations

- LoadLeveler 1.3 is available for AIX 4.1+ **only**.

- LoadLeveler 1.3 and 1.2.x are **not** compatible.

- LoadLeveler 1.3 submit-only binaries are presently **not** available for AIX 3.2.5 systems.

- **lldbconvert** is available to reformat local job queue files from 1.2.x format to 1.3 format on scheduling machines. This allows the administrator to migrate a cluster to 1.3 without having to flush the 1.2.x job queue.

LoadLeveler 1.3 will be available on AIX 4.2 in 4Q96.

Documentation List

- LoadLeveler's User and Administrator Guides have been combined into "***Using and Administering LoadLeveler***", document number ***SC23-3989-00***.

  Can also be browsed, searched and downloaded at:

  http://www.austin.ibm.com/software/Pubs/sp_books/ga41.html#ll

  Can also be ftp'd from ***lscftp.kgn.ibm.com****:*

  - ftp as anonymous.
  - User your email address as your password.
  - cd to ***/pub/pps/loadleveler.***

- The POWERparallel User Group

  http://spud-web.tc.cornell.edu/HyperNews/get/SPUserGroup.html

  http://spud-web.tc.cornell.edu/HyperNews/get/LL.html ***(newsgroup)***

- IBM RS/6000 Division Home Page

  http://www.rs6000.ibm.com

- IBM's Internal Newsgroup: ***ibm.ibmunix.loadlev*** from a UNIX newsreader, or ***LOADLEV FORUM*** on the ***IBMUNIX*** conference disk on VM.

Documentation List

- ***sp-discussion*** listserv:

  Send a note to *Majordomo@mcs.anl.gov* containing "*subscribe sp-discussion <email-address> <name>*" in the text.

- Maui High Performance Computing Center (MHPCC):

  Home Page:

  http://www.mhpcc.edu/

  LoadLeveler Tutorial:

  http://www.mhpcc.edu/training/workshop/html/loadleveler/LoadLeveler.html

- ITSO Redbook, "***Implementing High Availability on RISC/6000 SP***", documentation number ***SG24-4742-00***.

Note: LoadLeveler is supported under

AIX/6000 Family Support.

Future Directions

Fourth Quarter 1996

Full LoadLeveler 1.3 functionality for non-AIX platforms

Full LoadLeveler 1.3 functionality for AIX 4.2

ISS Enhancements for Scalable Web Serving

Early 1997

DFS/DCE Support

Next Major Version

(A partial list which may be expanded and revised...)

Parallel Scheduling Improvements

User Initiated Parallel Checkpoint/Restart

SMP Exploitation

Enhanced Resource Management

Continued GUI and Usability Improvements