Technical Overview

Table of Contents

1.0	SmartSuspend Overview	3
2.0	How SmartSuspend Works	3
3.0	Job Suspension Using SmartSuspend	4
3.1	Job pre-emption example	4
3.2	Practical, reliable and immediate job suspension	5
4.0	Using SmartSuspend™ with your applications	6
5.0	Common Use Cases	6
3.1	Pre-empt lower priority work without work loss	6
5.2	Manage expensive software licenses	7
5.3	Reduce or remove dedicated interactive user slots	7
5.4	Reclaim interactive slots during off hours	7
5.5	Eliminate micromanagement of project job queues	8
6.0	SmartSuspend Installation	8
7.0	SmartSuspend Deployment	9
7.1	Simple Deployment	9
7.2	Integration with Platform LSF	9
7.2.1	Configuring LSF to Use SmartSuspend	9
7.2.2	Controlling Jobs Using SmartSuspend	11
7.3	Integration with Other Job Queuing Systems	11
8.0	About Jaryba	11

1.0 SmartSuspend Overview

Jaryba SmartSuspend (SSR) is a grid workload management solution that eexecuting jobs to be reliably suspenderesumed at will. While suspended, the jhardware (CPU and memory) and license resources are released, making thosresources available to other jobs. As a user space technology, SSR achieves this without any modification to the underlying operating system (OS) or the applications under management. Licenses, memorCPU are cleanly reacquired when a job isresumed.

SmartSuspend Key Features:

- Application neutral suspension and resumption of running workload
- Seamless integration with popular queuing systems such as Platform LSF
- Releases hardware resources (CPU and memory) and software licenses upon suspension
- Virttually no performance overhead
- Supports both serial and parallel applications
- Enhances existing features of queuing system to increase utilization and flexibility
- No OS or application modification required

Customers use SmartSuspend for a variety of reasons:

- **Pre-empt without work loss**

Pre-empt a lower priority job without losing any work to date, and safely resume from where it left off when all higher priority work has completed.

- **Manage expensive software licenses**

Suspended jobs release their software licenses which become immediately available for other workload. It is no longer necessary to hold licenses in reserve – all licenses can be immediately re-assigned – allowing all licenses to be in use without penalty.

- **Reclaim interactive slots for off-hours workload**

Using SmartSuspend, interactive slots that would otherwise remain idle during off-hours can be reclaimed to run other grid workload, with the guarantee that these slots are immediately available when needed by the interactive user.

- **Reduce or remove the need for interactive slots**

SmartSuspend's immediacy enables classic interactive workload to be executed as needed without reserving grid resources for interactive users.

- **Eliminate micromanagement of project workload**

SmartSuspend enables "sponge" queues that will complete work on a best-effort basis. Project teams can consume daily tokens for the very important work, and sponge available capacity for their other workload without consuming a project token.

2.0 How SmartSuspend Works

Jaryba SmartSuspend leverages Jaryba's system call abstraction layer technology to seamlessly integrate between an application and the underlying operating system.

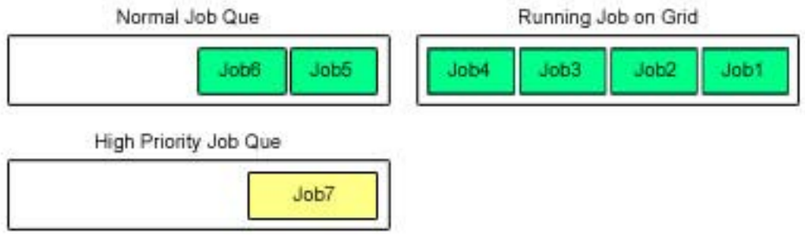
The SmartSuspendTM library intercepts all process and thread creation calls to build the set of threads and processes to suspend or resume. SmartSuspend manages both serial and parallel jobs, tracking all threads and processes on all machines used by the job.

3.0 Job Suspension Using SmartSuspend

3.1. Job pre-emption example

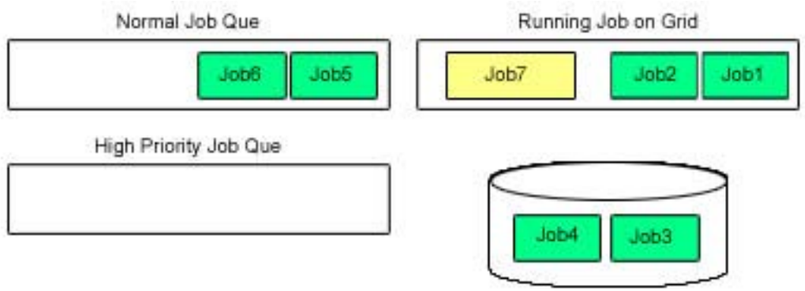
The following diagram represents a grid with jobs 1-4 dispatched to and running on nodes, with jobs 5 and 6 waiting in the normal priority job queue for resources to become available.

A new high priority job (#7) is added to the high priority queue that is configured to pre-empt the normal priority queue.



Normally one or more running jobs would be terminated to make space for the higher priority job. For example, both jobs 3 and 4 may have to be terminated to make way for job 7. The termination results in lost work – both jobs 3 and 4 have to be restarted from the beginning.

Instead, with SmartSuspend, both jobs 3 and 4 can be suspended and saved to disk to temporarily make way for job 7. Any licenses used by jobs 3 and 4 are returned to the license server.



When job 7 completes, jobs 3 and 4 are resumed from where they left off:

3.2. Practical, reliable and immediate job suspension

Many operating systems have the inherent ability to suspend a running process and, subsequently, to resume the process. Why then not simply fashion a solution around this ability to achieve job suspension control? Three reasons: practicality, reliability and immediacy.

- Practicality:

When a job is suspended using a stop signal, it is merely deprived of CPU resources. The job still occupies memory, which may be swapped out to disk if swapping is enabled. Eventually, swap space will be exhausted and the kernel's out-of-memory killer feature randomly selects a job to be killed.

This is not a practical solution – a successfully suspended job could be killed because a subsequent job uses lots of memory. It is also common to have no swap space configured, especially with the advent of bladed systems.

- Reliability:

Any process that is making a system call at the point when a stop signal is received must be able to handle the returned error code from the operating system call, recognize it as a stop/resume combination and then repeat the operating system call, adjusting the parameters to take into account work that has already been accomplished.

To be reliable, this error handling must exist around every single operating system call, in both the program and the libraries that it uses, and in the other programs that are invoked during the job. In reality, most applications have poor error handling, certainly not to the level of sophistication required to address deliberate operating system stop and resume signals.

- Immediacy:

Even though a job is suspended with an operating system stop signal, checked-out licenses continue to be held by the suspended job. A workaround enables licenses to be forcibly checked back into the license server. However, this approach is subject to a time penalty imposed by the vendor's license daemon to stop license misuse; typically the penalties are 20+ minutes before the licenses can be reused.

Therefore suspending a job using a stop signal and then forcibly checking the licenses back in is not an immediate solution – the scheduler must wait until the licenses are available until the preempting job can be dispatched to the grid.

SmartSuspend addresses these issues in three way with a more sophisticated approach to produce a practical, reliable and immediate solution to suspend and subsequently resume jobs:

a. Memory is written to a named storage device, avoiding reliance on swap space. Typically, parallel jobs can have all suspended memory written to the same job specific directory.

b. All system calls are seamlessly protected against suspension and resumption, with SmartSuspend automatically making the adjustments prior to repeating the system call.

c. Licenses are immediately returned to the license server upon job suspension, so that the scheduler

can immediately dispatch the next job without waiting for the license timeout penalty.

4.0 Using SmartSuspend™ with your applications

SmartSuspend leverages Jaryba's core user mode technology to seamlessly insert a library between

an application and the underlying operating system.

The SmartSuspend library intercepts all process and thread creation calls to build the set of threads and processes to suspend or resume. SmartSuspend manages both serial and parallel jobs, tracking all threads and processes on all machines used by the job.

To achieve license suspension, SmartSuspend intercepts calls made to the FLEXnet licensing server. This interception is performed regardless of the manner in which FLEXnet software is integrated into the application. This interception does not manipulate license counts, but instead sets the environment such that the licenses are immediately checked back into the license server when the application is suspended.

SmartSuspend uses a simple job loader called ssrcmd that precedes the job's normal invocation command. If a job is normally started in the following manner:

program <program options>

Then it is started under SmartSuspend using the following command:

ssrcmd <ssr options> -- program <program options>

SmartSuspend is queuing system neutral. It is delivered with tight integration scripts for many common queuing systems, with additional integration scripts under development. These integration scripts can be adapted on site to match customer specific environments and use cases.

5.0 Common Use Cases

SmartSuspend is used a variety of different ways to increase grid and license utilization. Common use cases for SmartSuspend are described below.

3.1. Pre-empt lower priority work without work loss

The grid scheduler dispatches the next highest priority job from the waiting queues. It is not aware of future higher priority work and thus makes the best decision with the information at hand. When a subsequent higher priority job is submitted, there are two options: either queue the high priority work until a slot becomes available, or pre-empt (kill) the lower priority job to make way. Neither option is optimal – either the high priority work is delayed or the work accomplished to date by the

low priority job is lost.

SmartSuspend adds a third option that enables immediate dispatch of the high priority job without losing the low priority job's work to date. Depending on the grid's dynamic mix, SmartSuspend improves grid utilization by 10% or greater.

SmartSuspend enables the lower priority job to be suspended – put into a comatose state – such that hardware resources (CPU, memory, etc.) and licenses are made available to run the higher priority job. The low priority job is resumed – awoken – when no more high priority jobs require the hardware and license resources. SmartSuspend seamlessly integrates with popular queuing systems such as LSF and SGE to provide this value with little or no end-user impact except to achieve better grid throughput.

5.2. Manage expensive software licenses

Software licenses can be a significant portion of a grid's cost – it is not unusual to have licenses that cost hundreds of thousands of dollars per seat, per year. In these cases the cost of software licenses easily eclipses the cost of the hardware that they run on. Grid scheduling policies must strike a balance between using the licenses as much as possible, while still having licenses available for higher priority work.

Keeping licenses in reserve for specific projects or high priority queues means that they often remain unused. Unused expensive licenses waste money, putting pressure on the grid manager to purchase additional licenses to satisfy the demands of other users.

The ability of SmartSuspend to immediately release license resources enables grids to speculatively use each and every license with the knowledge that a license can be instantly "borrowed back" for project or priority need. This increases the license usage and thus increases the ROI on each license. When grid managers do request additional licenses, it is with the knowledge that the current licenses are fully utilized.

The queuing system's ability to schedule license tokens can be used to identify when there is license contention and then select a lower priority job to suspend.

5.3. Reduce or remove dedicated interactive user slots

Typically the resources available to a grid are used both by interactive users and batch jobs. Interactive slots enable engineers to design solutions to meet business requirements. Often interactive slots are used only sporadically throughout the day, although they tie up dedicated hardware and license resources that cannot be reused for other workload.

SmartSuspend provides a different approach to satisfying sporadic needs for interactive sessions. It guarantees very quick access to the resources, minimizing engineer downtime while productively using the previously wasted time.

This productivity gain is accomplished by creating an "immediate run" queue that preempts one or more other queues. As soon as the engineer submits a job to the immediate run queue, the queuing system selects a lower priority job to suspend, runs the engineer's immediate job and then resumes the original job. The queuing system's existing controls and priority schemes select the job to be temporarily suspended – this can be as simple as always selecting the shortest running job, or using a more complex scheme to fair share the interruptions to all running jobs.

5.4. Reclaim interactive slots during off hours

While SmartSuspend can remove most of the demand for dedicated interactive slots, inevitably some interactive slots will remain, especially during a project's active design phase. Often interactive slots

are reserved over a span of days, consuming the dedicated resources even while the engineer is sleeping.

Because SmartSuspend seamlessly interacts with popular queuing systems, it has the ability to monitor for idle interactive slots during off hours. The idle interactive slot jobs are suspended to make way for other queued grid workload. The queuing system is configured not to dispatch new work to these slots within a few hours before the start of the work day. In-progress grid workload completes naturally and then the interactive workload is resumed, with the engineer unaware that resources were reclaimed during the off hours.

Detection of idle interactive slots can be fairly sophisticated using simple operating system commands. For example, there is a difference between a slot in which an interactive design tool has been left idle, and a slot used to run an overnight simulation to be ready for the morning. Idle detection includes idle terminal time and lack of significant consumption of CPU resources.

The reclamation of interactive slots is achieved through the queuing system's configuration, and with grid specific programs to detect idle slots. SmartSuspend facilitates this by seamlessly releasing and subsequently reclaiming hardware and license resources used by the idle interactive slot.

5.5. Eliminate micromanagement of project job queues

Job run tokens are often used in shared grids to ensure a controlled balance of grid resources for each project. As each job is dispatched to the worker node(s) a token is consumed. In a heavily used grid, this ensures that all projects have guaranteed access to their allocated resources. This requires project teams to micromanage their submission rate – they don't want to consume all tokens early in the day and then have no rights to grid capacity later in the day. Micromanaging job submission, dispatching and cancellation are time consuming tasks and require human intervention.

Because SmartSuspend is able to immediately suspend a running job, the need for human micromanagement of job queues is removed. Grid utilization is significantly increased while guaranteeing the project level SLAs, and removes the expensive human from the trivia of managing individual jobs. The key is to sponge spare capacity from the grid while immediately making room for any project job that has a scheduling token.

SmartSuspend accomplishes this efficiency through seamless integration with the queuing system. A lower priority "sponge" queue is created; any job executed from that queue does not consume a project token. A normal priority queue is used for jobs that consume a project token when the job is dispatched. Whenever a job runs from the sponge queue, it is making use of idle resources. When a normal job is ready to run, a sponge job is suspended to allow the normal job to run.

The project team can choose which queue to use at submission time – whether it is guaranteed to run using a project token, or will run only when there are spare resources. Project tokens are therefore used on the project's most important work, with other work being accomplished on a best effort basis. The project team can move jobs between the queues as required – for example towards the end of the day to promote jobs from the sponge queue to the guaranteed queue to consume remaining tokens.

6.0 SmartSuspend Installation

SmartSuspend can be installed to accommodate common grid designs:

- Installed on each node. This enables grid nodes to be booted from a single "golden image" such that every node has exactly the same OS.
- Installed on a central file system and mounted on each node.
- A combination of the above two methods.

7.0 SmartSuspend Deployment

SmartSuspend frees a suspended job's memory by writing it to disk. Either local or shared central storage can be used for memory storage. The product does not rely upon a node's swap space, which may not exist or may be overwhelmed by the number of running jobs.

7.1. Simple Deployment

Installation of SmartSuspend takes just a few minutes and once the software is installed on each node in a cluster, there are a few simple configuration steps:

- Set up a directory to which SmartSuspend can store job status. For parallel jobs or jobs involving multiple nodes, the directory should reside on a shared file system.
- Set up a location to which to save the suspended job's memory. This can be either a local or remote disk.
- Set up integration with the queuing system of choice.

Once set up, users can issue the normal queuing system commands. For example, they can use Platform LSF commands (bstop, bresume) to suspend and resume jobs; these commands invoke the underlying SmartSuspend command (ssrcmd).

The ssrcmd command is the only command provided by SmartSuspend. It uses different options to perform suspend and resume operations. A user can call this command directly, although it is typically not necessary because SmartSuspend can be seamlessly integrated with popular job queuing systems.

7.2. Integration with Platform LSF

Platform LSF has built-in hooks to suspend and resume a job. SmartSuspend has been seamlessly integrated and tested with LSF to allow users to take advantage of the job preemption capability without re-engineering their existing LSF job management infrastructure.

The built-in LSF suspend/resume behavior is to send SIGSTOP/SIGCONT signals to running jobs. The SmartSuspend integration substitutes these signals with the SmartSuspend ssrcmd command.

7.2.1. Configuring LSF to Use SmartSuspend

Platform LSF provides a preemptive scheduling feature that allows pending high-priority jobs to take resources away from running jobs of a lower priority. To keep the integration as simple and transparent as possible, SmartSuspend takes advantage of the LSF queue-based preemptive scheduling model, but uses SmartSuspend technology for handling the underlying job suspension and resumption operations.

Jaryba provides a separate wrapper script, ssrlsf.sh, that integrates between Platform LSF and the SmartSuspend ssrcmd command. This wrapper script is called from LSF when suspending, resuming or killing running jobs.

The administrator creates job queues by setting the proper parameters in the LSF queue configuration files:

- Pre-emptive—Jobs in this queue can preempt jobs in any pre-emptable queue of lower priority.
- Pre-emptable—Jobs in this queue can be preempted by jobs from any pre-emptive queue.

The following is an example of a queue definition for a high priority (pre-emptive) queue and a low priority (pre-emptable) queue:

Begin Queue

QUEUE_NAME = hipri

DESCRIPTION = For HIGH Priority Jobs

JOB_CONTROLS = SUSPEND[<path_to_lsf_integration>/ssrlsf.sh -s] |

RESUME[<path_to_lsf_integration>/ssrlsf.sh -r] |

TERMINATE[<path_to_lsf_integration>/ssrlsf.sh -k]

PREEMPTION = PREEMPTIVE[lowpri]

RERUNNABLE = YES

CHKPNT = <chkpnt_dir>

PRIORITY = 100

NICE = 20

End Queue

Begin Queue

QUEUE_NAME = lowpri

DESCRIPTION = For LOW Priority Jobs

JOB_CONTROLS = SUSPEND[<path_to_lsf_integration>/ssrlsf.sh -s] |

RESUME[<path_to_lsf_integration>/ssrlsf.sh -r] |

TERMINATE[<path_to_lsf_integration>/ssrlsf.sh -k]

PREEMPTION = PREEMPTABLE[hipri]

RERUNNABLE = YES

CHKPNT = <chkpnt_dir>

PRIORITY = 50

NICE = 20

End Queue

Where

- *<chkpnt_dir>* specifies a base directory where SmartSuspend will create the subdirectories (under <chkpnt_dir>/<LSF_JOBID>/ ssr_stat>) required for storing job state communication data and process IDs (PIDs), in addition to the LSF job output and log files. Specify a location on a shared file system.
- *<path_to_lsf_integration>* specifies the path where the LSF integration wrapper script (ssrlsf.sh) is installed. By default, it is installed in /usr/share/smart-suspend, but it can also be installed on a shared file system.

A user can then use normal job submission commands to submit jobs to either one of the two queues, based on the job's priority or SLA.

7.2.2. Controlling Jobs Using SmartSuspend

Once the job queues are set up using SmartSuspend, their behaviors are the same as normal LSF queues working with suspend/resume. A user can also suspend and resume jobs manually using LSF commands such as bstop and bresume, which will invoke calls to the SmartSuspend command with corresponding options instead of LSF's default actions.

LSF users typically use a bsub script to submit their jobs. They must modify the existing script, prepending the Jaryba LSF integration wrapper script ssrlsf.sh with any arguments before the application name:

ssrlsf.sh -- <application_name> <application_arguments>

After that, users can submit a SmartSuspend enabled job to the pre-emptable queue using:

 # bsub –q lowpri < script.bsub

Jobs will be suspended and resumed or will run to completion automatically based on their priorities. However, administrators can manipulate SmartSuspend enabled jobs manually using normal LSF commands.

To suspend a job using SmartSuspend:

 # bstop <job_id>

To resume a suspended job:

 # bresume <job_id>

To kill a job:

 # bkill <job_id>

7.3. Integration with Other Job Queuing Systems

The simple command line user interface included with SmartSuspend lends itself to easy integration with most any job queuing system. The more sophisticated the job suspension and pre-emption support of the queuing system, the more transparent the integration.

8.0 About Jaryba

Jaryba, LLC. is a leading provider of software that maximizes the utilization of hardware and software assets in data centers while maintaining application quality of service.

Jaryba products are based on unique application-centric variations on virtualization that are transparently interposed between the application and operating system. This allows the products to be easily deployed in existing data centers or cloud environments and generates a rapid return on investment.

The company is headquartered in Reno, Nevada.

Contact:

P.O. Box 19064

Reno, Nevada 89511

US: 775-393-9055

International: +1 775-384-8439

sales@jaryba.com