Improving Crash Analysis by Increasing Throughput of Large-Scale Simulations

Dale I. Dunlap and Shawn Freeman
Platform Computing

Abstract

Numerical simulation is an important tool used by engineers to design and develop safe automobiles. As engineers study larger and more complex models, demand for computational throughput increases. Grids allow a company to utilize its existing hardware investment to build a cost-effective platform for simulating automotive crash testing.

This paper will discuss how grid technology can substantially increase computational throughput of large-scale parallel simulations without having to upgrade the existing compute infrastructure. This leads to significant payback since customers can complete more work, while also deferring capital and operational costs.

The Challenge Presented by Large Scale Simulations

While decision makers expect significant contributions from CAE, earlier in the product development program, an ever-increasing barrage of new or improved requirements threatens to erode any gains made by buying more and faster hardware.

Large-scale simulations are especially sensitive to lack of resources, and conflicts can arise when multiple projects are run in the same environment, without any coordination of resources provided.

Large-scale simulations can have one or more of the following characteristics:

- Multiple jobs – collections of jobs used to study various permutations (i.e. DOE, stochastic, etc.).
- Heavy memory and disk requirements – simulations that require substantial amounts of available memory and disk space to complete.
- Long duration – simulation jobs that typically take a long time (CPU and wall clock) to complete.

This goal of this paper is provide some suggestions on how an organization will be able to complete more work while using the existing infrastructure. By using the existing resources, it is possible to save a substantial amount of capital and operational expenditures while meeting the business requirements for higher quality, no recalls, and faster time to market.

Large scale: Multiple jobs

One of the most significant contributions that CAE has made to the product development process is the ability to study several different permutations of the same basic design (virtual prototyping). This is something that could not be done practically in the physical world due the expensive and time-consuming nature of physical prototypes.

Traditionally, an engineer would have to rely on past experience and gut feelings to decide which combinations of parameters would be tested and which ones would not. For example, the engineer might decide to perform a test using the lightest weight vehicle body and largest engine to determine the worst-case crash scenario. However, other engine and weight combinations would not be tested.

With CAE, all such combinations can be simulated fairly easily. However, running these models still requires significant compute resources, which may be scarce in an already over-utilized data center. Instead of performing hundreds of DOE runs or thousands of stochastic runs, the engineer selects a subset to perform the analysis on. While this is better than what could have been achieved through physical testing, it is still not optimal.

With increased demand for quality products with no recalls, running all possible cases is the only solution.

Large scale: Heavy memory and disk

Continuously larger models are being developed for crash analysis, CFD, NVH, and other disciplines. The desire to create accurate simulations as close to the real world as possible is driving engineers to include increasingly more detailed representations of the parts, environment, and physics involved.

For example, structural and fluid coupling in crash occupant models is now starting to become mainstream. While this functionality has been available in some form for the past ten years, it was too expensive (CPU and memory) to perform such analyses outside of an R&D environment.

New emphasis and interest in out-of-position occupants, and small adults and children interacting with airbags during the early deployment stages, has driven the need to use more accurate modeling techniques for airbags.

Furthermore, the very nature of the mathematics used in h-version Finite Element Analysis also dictates that an exact solution requires the use of infinitely small elements. Therefore, as a model approaches the ideal representation of the real world, the number of simultaneous equations to be solved goes up, along with memory and disk requirements.

Large scale: Long duration

A typical full vehicle crash simulation can take several days to complete just 100ms of simulation time. Faster hardware and parallelization have helped to bring the total time down

over the years. However, new requirements and expectations have blunted the impact that new technology has had on overall throughput.

Some of these requirements are:

- Demand for improved fidelity when compared to physical testing.
    a. Finer granularity when modeling of components.
    b. Use of new features (fluid/structures interaction).
    c. Heavier reliance on automatic generation (meshless welds, contacts, etc.).
- New scenarios introduced by consumer-oriented testing. For example:
    a. EuroNCAP – initiative to reduce pedestrian injury, in car to pedestrian impacts.
    b. NCAP – frontal and side impact ratings published and used heavily for advertising.
    c. IIHS – very public 40mph offset barrier and new (2003) side impact testing.
- New and/or improved scenarios introduced by government regulation. For example:
    a. FMVSS208 – 5th female and 6 year old dummies for out-of-position.
    b. FMVSS214 – quasi-static side impact testing.
    c. FMVSS201 – interior head impact requirements.
- Desire to include dummy occupants in models where previously only structural results were required.
- Desire to use biofidelic human representations, instead of dummy models, in crash simulations.

Making use of parallel features in LS-DYNA can greatly reduce the amount of wall clock time required to complete a simulation. The challenge is finding and reserving an appropriate set of machines with sufficient resources free to run the job.

In all three cases, a perceived scarcity of compute resources has a direct impact on the simulation engineer's ability to provide timely and complete results. In some cases, the solution set may be reduced or the model simplified, or some other compromise is made.

This is counter to the goals of nearly every manufacturing company. Specifically, manufacturers are relying more heavily on CAE to predict all possible outcomes, and to design a top quality product at a price that is accepted by consumers and profitable for manufacturers. To this end, CAE must be utilized to its fullest capability to reduce expensive testing, late stage engineering changes, and damaging product recalls.

The Solution – Clustering

In most MCAE organizations, there are workstations and compute servers.

Solution for HPC

The compute servers are usually dedicated to a specific group of solvers and run 24/7 in the corporate data center. Platform LSF HPC is specifically designed to address the needs and requirements of the high performance computing environment.

Platform LSF HPC provides intelligent scheduling for parallel and serial workload, providing the capability of solving large, grand challenge problems while utilizing the available computing resources at maximum capacity. Platform LSF HPC takes full advantage of high performance network interconnects available on clustered systems and supercomputers.

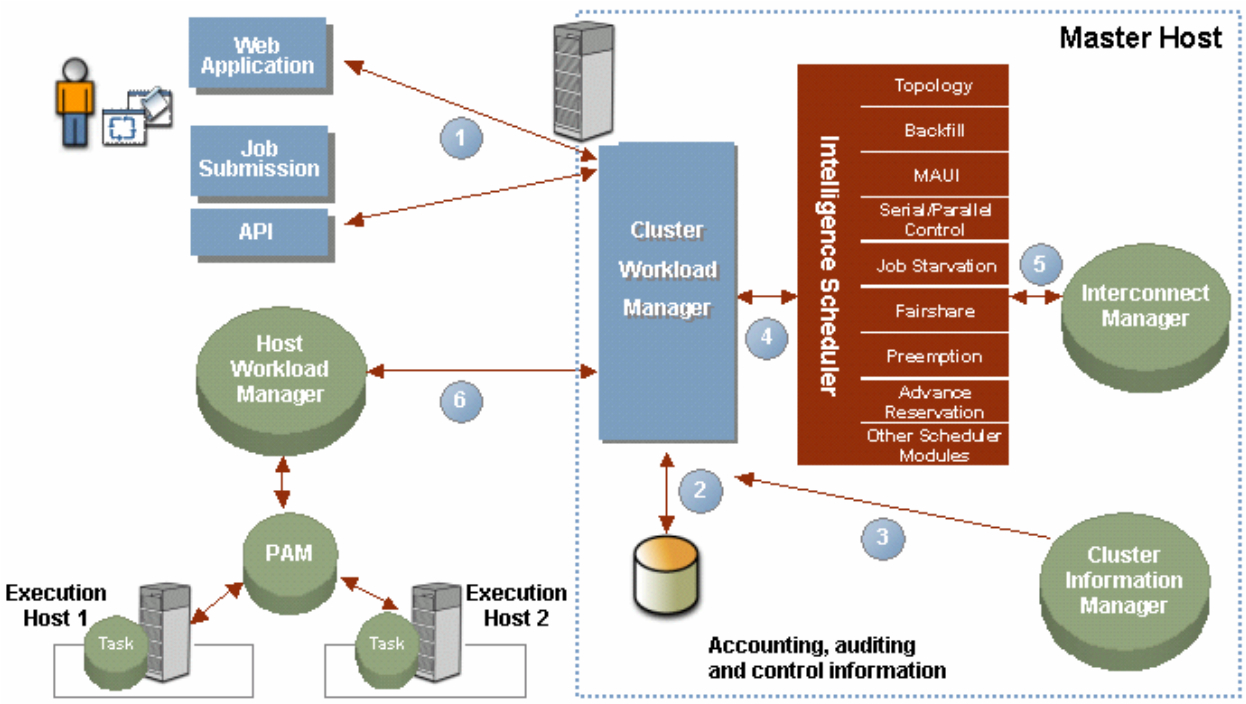

Figure 1 - HPC Topology

A typical HPC environment will provide several queues for targeted job submission. Some queues may provide strict run time or memory limits, while others may be tailored for parallel jobs.

Once appropriate queue and host definitions have been created, it is a simple matter to submit jobs to the cluster for running:

Example 1 – bsub job submission:

```
$ bsub -q normal myjob1
```

Example 1 shows a very simple command for a user to submit a job to the "normal" queue. If a user wishes to submit a DOE or stochastic study to the queue, this can also be accomplished with a single command:

Example 2 – bsub job submission for an array of jobs:

```
$ bsub -q normal < myjobs.scr
```

Where, "myjobs.scr" might look like this:

```
#BSUB -J lsdyna[1-100]
#BSUB -n 2
#BSUB -o jobdoe%I.stdout
#BSUB -e jobdoe%I.stderr


/usr/bin/lsdyna970 i=jobdoe${LSB_JOBINDEX}
```

Example 2 utilizes a job array to submit 100 jobs into the "normal" queue. All 100 jobs are managed under a single job id, making tracking and management as easy as handling a single job.

A single user that is running multiple jobs (DOE, stochastic, etc.) can easily monopolize the cluster, preventing the fair consumption of resources by others. To help prevent this, Platform LSF HPC can place limits on individual or groups of users as well as utilize special scheduling algorithms. Limits on users can be become cumbersome to manage for any medium to large organization and often time using the fairshare scheduler is more than sufficient to solve this problem.

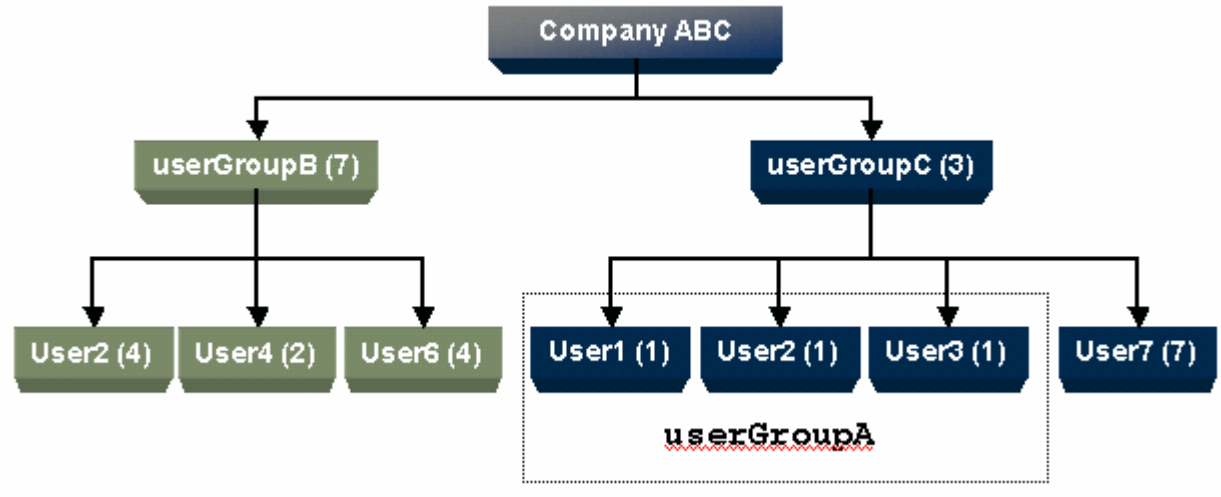

Figure 2 - Fairshare Hierarchy

Example 3 – Enabling the fairshare scheduler:

```
<lsb.queues>
Begin Queue
# Define a unique queue name
QUEUE_NAME = normal
# Give a meaningful description of the queue
DESCRIPTION = Queue used for medium length jobs
# Who are the administrators for this queue
ADMINISTRATORS = ddunlap sfreeman
# Priority relative to other queues
PRIORITY = 90
# Set threshold of 5 minutes for automatic job migration
MIG = 5
# Which hosts can be used
HOSTS = all
# Which users can use this queue
USERS = all
# Define equal Fairshare scheduling for all users
FAIRSHARE = USER_SHARES[[default,1]]
End Queue
```

Example 3 gives a very simple example of using queue-level fairshare scheduling. The fairshare priority determines whether a job from a specific user will be dispatched to run or skipped in favor of another user's job. The fairshare priority is a function of user shares, number of jobs running, and history of recently run jobs. In this example, all users have an equal share. It is possible to bias the priority by assign non-equal shares based on individual users or user groups for a single queue or cluster wide.

Example 4 – Biasing fairshare to favor the local user group (DepartmentA) over DepartmentB:

```
<lsb.users>
Begin UserGroup
GROUP_NAME  GROUP_MEMBER  USER_SHARES
DeptA  (ddunlap user3 user5 user7)  ()
DeptB  (sfreeman user4 user6)  ()
End UserGroup

<lsb.queues>
Begin Queue
# Define a unique queue name
QUEUE_NAME = normal
# Give a meaningful description of the queue
DESCRIPTION = Queue used for medium length jobs
# Who are the administrators for this queue
ADMINISTRATORS = ddunlap sfreeman
# Priority relative to other queues
PRIORITY = 90
```

```
# Set threshold of 5 minutes for automatic job migration
MIG = 5
# Which hosts can be used
HOSTS = all
# Which users can use this queue
USERS = DeptA DeptB
# Define equal Fairshare scheduling for all users
FAIRSHARE = USER_SHARES[[default,1]  [DeptA,10]  [DeptB,5]]
End Queue
```

In example 4, users from DeptA are given preference over users from DeptB. Both DeptA and DeptB are given preference over all other users. It is possible to assign individual shares, in the lsb.users file, and build a hierarchical structure to sway preference towards individual users within groups. By providing the maximum level of flexibility, Platform LSF HPC preserves the compute resource owner's ability to maximize local throughput while still sharing free cycles with others.

When submitting large memory jobs, it is desirable to execute on a host with sufficient memory and swap. Otherwise, the job may fail due to a lack of resources and have to be restarted on another host. This loss of work is not only an inconvenience, but can cost organizations real money in the form of time and productivity lost.

Example 5 - Large memory jobs may require that thresholds on available memory and swap be defined:

```
<lsb.hosts>
Begin Host
HOST_NAME  it  r1m       mem  swp  tmp
default    15  0.6/1.6   512  1024 2048
End Host
```

In example 5, we have defined limits on available (free) memory, swap, and scratch space of 512Mb, 1Gb, and 2Gb respectively. If a machine in the cluster, does not meet these requirements, a job will not be dispatched to it.

It is also possible to create a queue and apply these requirements at the queue level.

Example 6 - Thresholds on available memory and swap defined on the queue:

```
<lsb.queues>
Begin Queue
# Define a unique queue name
QUEUE_NAME = large_mem
# Give a meaningful description of the queue
DESCRIPTION = Queue used for large memory jobs
# Who are the administrators for this queue
ADMINISTRATORS = ddunlap sfreeman
# Priority relative to other queues
```

```
PRIORITY = 90
# Which hosts can be used
HOSTS = all
# Define memory, swap, and scratch start/stop requirements
MEM = 512/64
SWP = 1024/128
TMP = 2048/128
End Queue
```

In example 6, we have defined limits on available (free) memory, swap, and scratch space of 512Mb, 1Gb, and 2Gb respectively. The queue will not select a host for job dispatch if it does not meet the requirements given. Additionally, the queue has defined stop limits of 64Mb, 128Mb, and 128Mb for memory, swap, and scratch respectively.

Often, long running jobs take advantage of parallel execution to reduce total wall clock run time. In cases where jobs are parallel, it is possible to reserve resources such that the jobs don't sit in the queue endlessly waiting for the appropriate resources to become available. However, once a resource is reserved, that resource will sit idle until the parallel job actually starts. To scavenge these idle resources, one can create a back-fill queue.

Example 7 – Back-fill queue definition:

```
<lsb.queues>
Begin Queue
QUEUE_NAME = back_fill
DESCRIPTION = backfill queue to utilize reserved resources
for parallel jobs
PRIORITY = 40
BACKFILL = Y
RUN_LIMIT = 20 240
SLOT_RESERVE = MAX_RESERVE_TIME[200]
End Queue
```

Such a backfill queue definition will allow jobs that are submitted with a run limit of less than 240 minutes to be executed on resources that are reserved for a pending parallel job.

Example 8 – Submitting a job to the "back_fill" queue:

```
$ bsub –W 10 –q back_fill myjob2
$ bsub –W 222 –q back_fill myjob3
$ bsub –q back_fill myjob4
```

In example 8, "myjobs2" is submitted with a run limit of 10 minutes. If the job takes longer than 10 minutes, it will be killed by the queue to ensure that the pending parallel has the resources promised to it. "myjob3" is submitted with a run limit of 222 minutes, while "myjob4" uses the default queue specified run limit of 20 minutes. Any job with a specified run limit longer than 240 minutes will be rejected by the queue.

Solution for workstations

While clustering dedicated compute servers together to provide a single, powerful virtual super-computer has become relatively common; most companies still have large numbers of powerful CAD/CAE workstations sitting idle 75% of the time.

Clever engineers will find a way to selectively use a portion of these machines for test runs or short “overnighters”, but the typical simulation engineer will only have access to a small number of local workstations. While the engineer competes with others for these precious “untapped” computed resources, nearly 4 times as many CAD workstations sit untouched because the engineer has no access to them.

Often, powerful desktop workstations sit idle only because there is no easy, politically correct way to coalesce these machines into a manageable compute resource. As a result, more jobs are sent to the HPC data center, filling up queues and increasing overall time required to push a single job through the system, and local machines are unwittingly over-taxed with “overnighters”.

Using Platform LSF HPC, a company can harness their powerful desktop workstations by grouping them into a cluster. This cluster can be used for large MPP jobs, without any performance impact on the desktop user. Since these clusters can be managed at the departmental level, political boundaries remain intact and each owner maintains full control over their computers.

To accomplish this, several options are available:

- Limit the amount of CPU, memory, swap, and/or any other resource such that a machine is not overloaded.
- Create windows, during which execution can occur, that disallow execution of jobs during working hours.
- Set reasonable thresholds for jobs to be suspended and/or migrated to another machine.
- Create a multi-cluster environment in which each department has overall control of their compute resources (local cluster) while still sharing their resources with others throughout the organization.

Pooling design and engineering workstations into a local (departmental) cluster brings a few additional considerations to be accounted for.

Most workstation users would not mind having their workstation crunch large jobs overnight or on weekends, as long as the jobs do not interfere with their productive working hours. To keep the machine free from jobs during working hours, a queue can be defined with a run window. A run window defines a time when the queue is “open for business”. Outside of that time window, the queue may accept jobs, but will not execute them.

It is important to target solvers that can be check pointed and migrated when establishing a workstation-based cluster. LS-DYNA is an excellent example of such a solver. Should a window close during a solution, the job can be check pointed and migrated to another host, or set of hosts, without having to start from the beginning.

Example 9 - Create a run window to only allow jobs Monday through Friday at night and through the entire weekend:

```
<lsb.queues>
Begin Queue
# Define a unique queue name
QUEUE_NAME = off_hours
# Give a meaningful description of the queue
DESCRIPTION = Queue used for nightly and weekend work
# Who are the administrators for this queue
ADMINISTRATORS = ddunlap sfreeman
# Priority relative to other queues
PRIORITY = 90
# During what times are jobs allowed to run in this queue
RUN_WINDOW = (18:00-6:00  5:18:00-1:6:00)
# Which hosts can be used
HOSTS = all
End Queue
```

The above example defines a queue definition that will allow jobs to run only at nights (Monday through Friday) and through the entire weekend. There are no restrictions on job size, type, or when the job can be submitted. Jobs submitted to the queue when the run window is closed will simply pend until the window opens. Jobs that are running when the window closes will be suspended until the window opens again.

In the event that a user is using or logs into a host, during the open time period of the run window it is possible to migrate the job to another host. This will free resources on that host to help ensure that the interactive user has the fullest availability of the host resources. To accomplish this, simply add a few more lines to the queue and host definition files.

Example 10 – Run window, with automatic migration:

```
<lsb.queues>
     Begin Queue
     # Define a unique queue name
     QUEUE_NAME = off_hours
     # Give a meaningful description of the queue
     DESCRIPTION = Queue used for nightly and weekend work
     # Who are the administrators for this queue
     ADMINISTRATORS = ddunlap sfreeman
     # Priority relative to other queues
     PRIORITY = 90
     # During what times are jobs allowed to run in this queue
     RUN_WINDOW = (18:00-6:00  5:18:00-1:6:00)
     # Set threshold of 15 minutes for automatic job migration
     MIG = 15
     # Which hosts can be used
     HOSTS = all
```

```
End Queue

<lsb.hosts>
Begin Host
HOST_NAME      it   r1m
default        15   0.6/1.6
End Host
```

In example 10, we allow jobs to be suspended by the system for 15 minutes (MIG = 15 in lsb.queues). After that time has passed, such jobs will be migrated to another host. We have also limited the hosts by requiring that the machine be idle (no active keyboard or mouse activity) for at least 15 minutes before starting a job ("it" of 15), and that the 1 minute run queue length (kernel queue length) be less than 0.6 ("r1m" of 0.6). Jobs will be stopped if r1m hits 1.6 or higher (this indicates the machine is busy). Placing these limits at the host level helps to ensure that a machine will be responsive to an interactive user, even during times when the run window is open.

While HPC computers will likely have high-speed inter-connects (i.e. Myrinet), workstations clusters will likely have more conventional inter-connects and domains separated by switches. In such cases, it is desirable to ensure that selection of hosts for parallel jobs remain on the same domain to minimize the impact of network latency on the message passing portion of the process.

Grouping hosts by locality can be accomplished through the use of the host groups.

Example 11 – Grouping hosts by locality.

```
<lsb.hosts>
Begin HostGroup
GROUP_NAME  GROUP_MEMBER
groupA      (hostA hostD)
groupB      (hostF hostK groupA)
groupC      (!)
End HostGroup
```

In this example, three host groups are created. The groups are described below:
GroupA – includes "hostA" and "hostD"
GroupB – includes "hostF", "hostK", and all the hosts from "groupA"
GroupC – will be dynamically configured using an "egroup".

The "egroup" is especially useful for clusters where hosts may come or go from the cluster, or where a machine may be moved from one subnet to another arbitrarily. A site specific egroup can determine the locality based on the organization's topology and provide that information to Platform LSF HPC.

Solution for the organization

At this point, we have accomplished creating a cluster out of a collection of desktop workstations and also making sure that those workstations would give a preference to interactive users.

Once a department implements a cluster utilizing local workstations, it becomes possible for inter-departmental resource sharing through Platform LSF MultiCluster.

Platform LSF MultiCluster extends an organization's reach to share resources beyond a single Platform LSF HPC cluster to span geographical locations. With Platform LSF MultiCluster, local ownership and control is maintained ensuring priority access to any local cluster while providing global access across an enterprise grid. Organizations using Platform LSF MultiCluster complete workload processing faster with increased computing power, enhancing productivity and speeding time to results.

Platform LSF MultiCluster supports two models for resource sharing:

1. Resource leasing – resources of a remote cluster are leased to a local cluster, becoming part of the local clusters virtual computer.
2. Job forwarding – queues on the local cluster may forward jobs to a remote cluster for execution.

Resource leasing requires that the provider, often the HPC, cluster export hosts to the consumer cluster(s). This allows consumer clusters to provision additional resources from the provider on an as needed basis. The leased compute resources show up in the consumer cluster as a regular part of the cluster. The local user need not be aware that the compute resources are not actually local to the cluster.

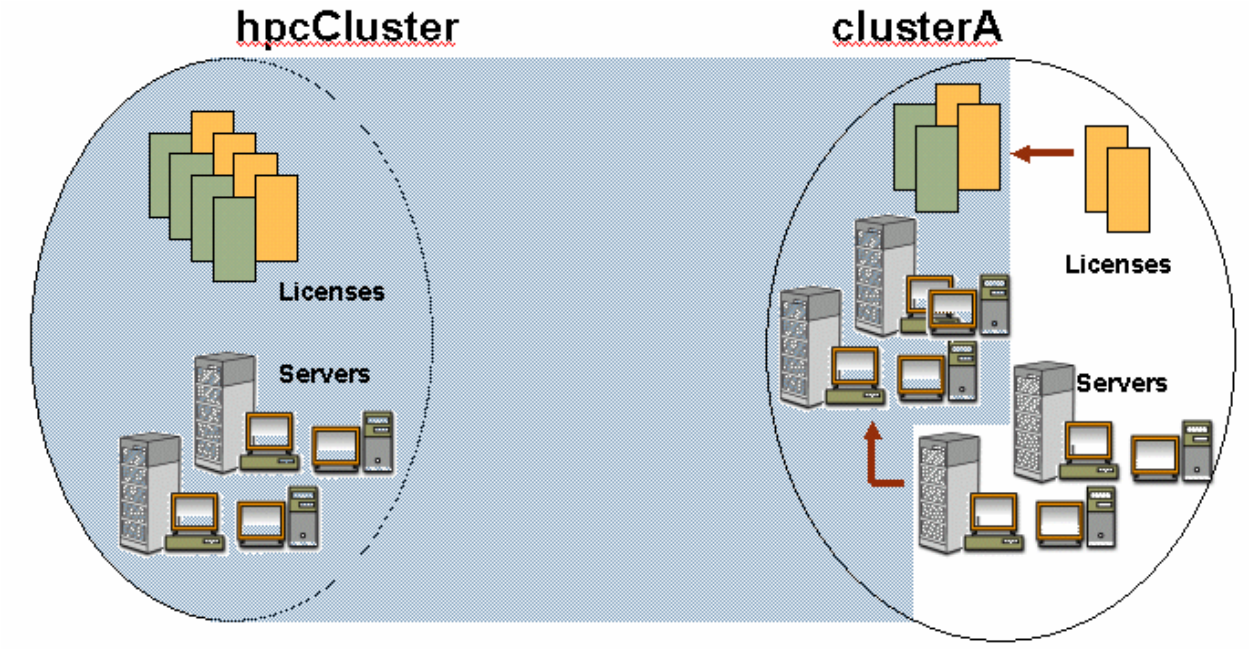

Figure 3 - Resource Leasing Model

Example 12 – Export hosts from the HPC to clusterA and clusterB (workstation clusters for DepartmentA and DepartmentB respectively):

```
<lsb.resources>
Begin HostExport
PER_HOST = ibm01 ibm02
SLOTS = 4
DISTRIBUTION = [clusterA, 2] [clusterB, 1]
End HostExport

<lsb.queues>
Begin Queue
# Define a unique queue name
QUEUE_NAME = large_mem
# Give a meaningful description of the queue
DESCRIPTION = Queue used for large memory jobs
# Who are the administrators for this queue
ADMINISTRATORS = ddunlap sfreeman
# Priority relative to other queues
PRIORITY = 90
# During what times are jobs allowed to run in this queue
RUN_WINDOW = (18:00-6:00  5:18:00-1:6:00)
# Which hosts can be used
HOSTS = all all@hpcCluster
# Define memory, swap, and scratch start/stop requirements
MEM = 512/64
SWP = 1024/128
TMP = 2048/128
End Queue
```

In the preceding example, the HPC exports two hosts, ibm01 and ibm02, to clusterA and clusterB. Note that clusterA has been 2 shares to clusterB's 1. The "large_mem" queue, from clusterA, includes the hosts exported from the hpcCluster as well as all local hosts.

The job forwarding provided the ability to spill jobs over to another cluster as the local cluster becomes busy. For example, when the workstation window closes, it is possible to migrate the jobs to another cluster for completion, rather than having them pend on the workstation waiting for the window to reopen.

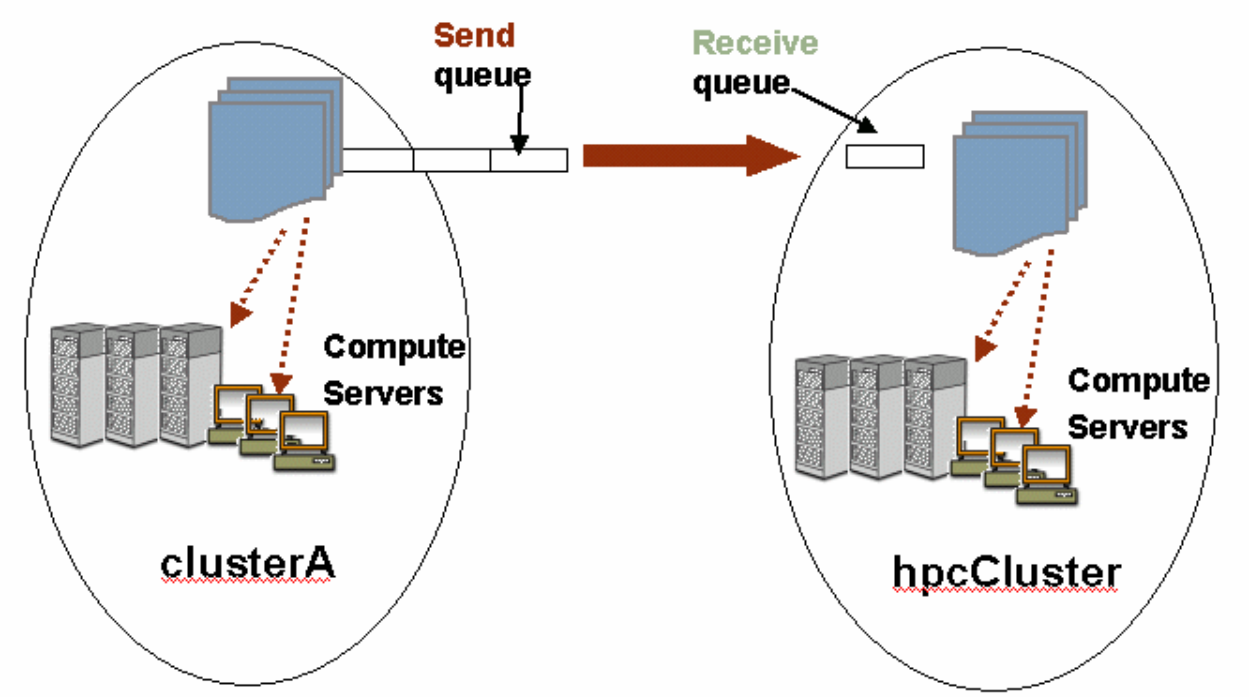

Figure 4 - Job Forwarding Model

The job forwarding model is equally simple to setup:

Example 13 – Forward jobs from clusterA to hpcCluster:

```
<lsb.queues>
Begin Queue
# Define a unique queue name
QUEUE_NAME = large_mem_recv
# Give a meaningful description of the queue
DESCRIPTION = Queue used for large memory jobs
# Who are the administrators for this queue
ADMINISTRATORS = ddunlap
# Priority relative to other queues
PRIORITY = 90
# Which clusters can this queue receive jobs from
RCVJOBS_FROM = clusterA
# Define memory, swap, and scratch start/stop requirements
MEM = 512/64
SWP = 1024/128
TMP = 2048/128
End Queue

<lsb.queues>
Begin Queue
# Define a unique queue name
```

```
QUEUE_NAME = large_mem_snd
# Give a meaningful description of the queue
DESCRIPTION = Queue used for large memory jobs
# Who are the administrators for this queue
ADMINISTRATORS = sfreeman
# Priority relative to other queues
PRIORITY = 90
# Which hosts can be used
HOSTS = none
# Send jobs where
SNDJOBS_TO = hpcCluster
# Define memory, swap, and scratch start/stop requirements
MEM = 512/64
SWP = 1024/128
TMP = 2048/128
End Queue
```

Example 13 establishes a receiving queue on hpcCluster and sending queue on clusterA. The queue on clusterA is configured to only forward jobs to HPC, and not run any locally.

Job forwarding and resource leasing can work both ways (the local cluster can receive jobs and export resources). There are few limits on how an organization can creatively leverage the functionality of the Platform LSF MultiCluster environment.

Interactive uses for Platform LSF HPC

Additionally, it is possible to leverage the HPC and workstation clusters for tasks other than batch jobs. Some of the advantages of doing so include:

- Resource accounting
- License scheduling
- User and/or host limits

Using Platform LSF HPC, interactive jobs can be run through the batch system with little or no noticeable impact on the end user. To accomplish this, we only need to define a special "interactive" queue for receiving such jobs.

Example 14 – Interactive queue definition:

<lsb.queues>

```
Begin Queue
QUEUE_NAME = interactive
DESCRIPTION = queue for tracking interactive jobs
PRIORITY = 150
INTERACTIVE = ONLY
NEW_JOB_SCHED_DELAY = 0
HOSTS = all
End Queue
```

In example 14, we have defined an interactive queue. The NEW_JOB_SCHED_DELAY is the key to making the queue responsive to the user commands by scheduling the job immediately.

Example 15 – Submit a job to the interactive queue:

```
$ bsub –q interactive –I  gimp
$ bsub –q interactive –I –m "`hostname`+5 others" hm
```

The first bsub command, in example 14, submits interactive Gimp session. The job will be started on the best host available in the cluster. It will not be obvious to the end user that the job is running through Platform LSF HPC, even though the job may be actually running on a different host than the host on which the user is sitting. In some cases, due to licensing or performance issues, it may be desirable to prefer the localhost to other hosts in the queue. The second bsub command shows an example of this. Here, an interactive hm session is started, with a preference (+5) for the localhost, as determined by the "hostname" command.

Conclusion

The solution is a phased process that starts with the HPC, extends to the engineering workstations, and then ties the organization together through Platform LSF MultiCluster. Each cluster is owned and managed locally, but has the ability to share and borrow resources from other clusters within the organization.

By using a 3-phased approach, it is possible to maximize the utilization of the existing infrastructure incrementally. Each phase of implementation improves the organizations ability to meet current and future computing needs, without substantial additional investment.