Paper 168-27

Using SAS® in a Distributed Computing Environment

Chris Smith, Platform Computing, San Jose, CA

ABSTRACT

The Platform LSF® software is a software solution which allows organizations to take advantage of their distributed computing resources. By aggregating these resources into a "virtual supercomputer", the LSF software can be used in conjunction with SAS software to provide various benefits to users of the computing environment. Benefits include: quicker time to solution by enabling the use of distributed processors and by controlling access to limited computer resources, a single user interface to heterogeneous computer systems, centralized resource provisioning using queue priorities and job preemption, and a reliable job scheduling environment enabled by calendar-based and inter-job dependencies.

An example of how to apply distributed computing principles to SAS software processing will be discussed in the context of web server log analysis using the SAS Institute Webhound™ software. A job flow using the Platform LSF software to manage this processing will be developed, and the results of running this flow sequentially, and then in parallel on a cluster of machines, will be presented. By way of this example, it is clear that the LSF software enables shorter time to results, and provides a more reliable execution environment.

INTRODUCTION – THE PROBLEM

Traditionally, the use of SAS software has been fairly "host focused". That is, users either run SAS software on a dedicated machine, such as a Windows NT® desktop, or are used to logging into a UNIX compute server and running SAS software programs through their login session. While this model works well for the development cycle of a SAS software program, when the activity is very interactive, it is an inefficient way of running the production workload through to solution. On a Windows NT desktop, for instance, the system might take a very long time to run the solution, interrupting any other user activities, or requiring a dedicated machine to run production programs. On a time-shared system, running the SAS software program might negatively impact the performance of the system, on which the OS is trying to balance the needs of all the logged in users. Furthermore, if many users are running many SAS software programs, the performance of the system can be degraded by paging and swapping such that all processing ends up running for much longer than in a dedicated mode.

Ultimately, the time to solution is impacted, in turn impacting the productivity of the users, and ultimately the organization. This is especially onerous since there is usually an abundance of computing resources in the organization, available on the network, with no way for the users to identify and access them. For instance, what if the Windows NT user referenced above could use the desktop of their colleague who was out of the office for the day. This way, while the production SAS software code was running, the user could continue to do their other work. Similarly, what if the user of the timesharing system logged in to another compute server that was much less loaded than the current host they were logged into. This is actually very common in many organizations where a particular machine is so well recognized by name by the users that they don't know the other machines that might be available to them.

Taking advantage of these distributed resources can be very difficult. Users need to know what computers are on the network; they need to know whether those machines are already fully used; they need to be able to move their processing to the machine. The way to combat this complexity is to use a software layer that can collect these resources together, providing a single interface, regardless of the location of the resource. The Platform LSF software provides this capability by providing: a central repository of real time load information about the machines in the environment, a single interface for submitting workload (or "jobs") to the LSF software which can then execute them transparently (to the user) on remote computers, and a central scheduling system which can make sure that machines are not overloaded and that important processing gets the resources needed in order to minimize the time to results.

PLATFORM LSF SOFTWARE FOR DISTRIBUTED COMPUTING

The LSF software solution provides features for solving the problem of managing and sharing the workload among a cluster of networked computing systems. The LSF software simplifies the use of distributed systems through the centralization of system resource information, by centrally matching resources to jobs and controlling the jobs during runtime, by providing a mechanism for transparently executing tasks remotely, and by providing a reliable, fault-tolerant execution environment.

CENTRAL STORE OF RESOURCE INFORMATION

Platform Computing's Load Information Manger (LIM) technology provides a centralized view of all system resources. All hosts in the cluster (regardless of system type or operating system) report their current "load" to a "master" LIM service that is dynamically elected from the pool of systems in the cluster. This provides one repository of state information for all the nodes in the system. By querying the master, you can get a snapshot of the health of all your systems, and the current load conditions on each.

LIM tracks resources such as run queue length, CPU utilization, system paging rate, io transfer rate, free memory and swap among other useful indices. Sites can also extend the LIM to monitor resources that are of importance to computing at the site. Examples of site-specific indices might be the amount of local "scratch" space, the availability of floating software licenses, or the amount of bandwidth available over a network connection.

LIM's central store of information can be used to make task placement decisions based on current load conditions, or based on the type of system which is required (for example, a particular type of job will only run on my HP-UX systems). This load information is also used by the central scheduling component of the LSF software.

CENTRALIZED CONTROL AND PROVISIONING OF DISTRIBUTED RESOURCES

In the LSF software, a central scheduler plays the role of matching up computing supply to the workload demand. This ensures that systems are fully utilized; yet not oversubscribed. The scheduler relies on the LIM to provide the runtime load information necessary to achieve this balance.

Since you no longer are responsible for making job placement decisions, and since the scheduler primarily places jobs based on resource requirements, the use of a scheduler solves the problem of the "host-centric" view of computing resources, which was mentioned in the introduction, by providing you a "resource based" view of computing resources. The scheduler promotes this view by translating job resource requirements into the list of machines that can support the requirement. For example, you might submit a job that does a lot of sorting, and thus requires a large amount of memory. The scheduler might noticed that host X is less loaded in terms of cpu, but has less memory available than host Y (which is only running some jobs), and thus dispatches the job to host Y since it better supports the resource requirement.

In addition to resource-based scheduling, the LSF software scheduling policies can implement business or organizational

policy. Typically, a particular project (perhaps close to deadline) has a requirement to use more of the computing resources in order to meet the deadline. The Platform LSF software provides a job priority mechanism (through queue priority) so that high-priority work gets access to the resources before low-priority work. This is complemented by a preemption engine that can preempt low priority jobs in order to free up resources for higher priority work.

The scheduler's central repository of jobs allows for a rich set of inter-job dependencies between jobs in a "job flow". Jobs can be scheduled based on the finish (successful or unsuccessful) of a previous job, or can be scheduled based on calendar requirements, or events such as file existence or file age. This functionality, coupled with the resource-based and fault-tolerant execution has been identified as an important part of the processing in a data warehousing environment, and thus the LSF JobScheduler software is now bundled with the SAS/Warehouse Administrator® software as the job scheduling engine of choice.

FACILITY FOR TRANSPARENT REMOTE JOB EXECUTION

In order to make it easier for users to run jobs remotely, without needing to worry about all the details of executing on a remote host, the LSF software attempts to virtualize the distribution.

One interface is used for accessing all systems in the cluster (the 'bsub' command interface or equivalent graphical user interface), and in order to ensure that jobs run similarly on the remote host, the job submission environment is replicated in the execution environment. An "interactive batch" execution mode is also supported, allowing you to perform interactive tasks such as program development or data visualization, while being able to take advantage of distributed resources.

RELIABLE AND FAULT-TOLERANT ENVIRONMENT

The LSF software's central scheduler will be automatically started on the host that is running the master LIM. The scheduler can be run on any of the nodes in the cluster without loss of state through the use of a job event log. Thus if the scheduler host crashes, or is brought down for maintenance (more common), the master LIM is elected to run on another host in the cluster, the scheduler is automatically started on this host, and the scheduling and runtime state of jobs is not lost. In fact, jobs continue to run with no interruption.

If jobs fail due to transient failures (host runs out of memory, swap or temporary space, or a program can't access a particular file from a particular node) it can be automatically requeued to run again on a different machine, thus routing around system failures. Using inter-job dependencies you can even take corrective action in the case of a particular type of failure. This is very important for processing which must occur such as the batch processing of data warehouse ETL jobs overnight.

DISTRIBUTED PROCESSING OF WEB LOGS

The primary use of the LSF software in production is to reach a shorter time to results through distributed computing. This manifests itself in many different ways. You might leverage idle compute cycles on workstations in your organization in order to process the same amount of data in less time. Alternatively, distributed computing might allow you to process more data in the same amount of time. By instituting the use of distributed computing within your organization, you also enable incremental scalability. That is, as your processing needs grow, you can incrementally add computing resource to your cluster (i.e. buy another server) as you incrementally grow the amount of data that you are processing. In order to illustrate the use of the LSF software to reduce the time to results an example of web log analysis using SAS Institute's Webhound software will be presented.

Why Webhound? No example is as good as a real world example, and using Webhound as an example illustrates two points:

- First, Webhound can be considered an "off-the-shelf" software package. That is, minimal SAS programming was necessary to achieve benefits from distribution. An equally valid example would be to take a particular data processing problem which could be solved through a SAS program, and modify the program such that it implemented a "divide and conquer" approach to the problem (using multiple instances of SAS software), and distribute those instances. But, using Webhound shows how to exploit the natural parallelism found in many off-the-shelf software programs by using the LSF software's distributed queuing function.
- Second, since I'm not a statistician and don't use SAS to solve the problems encountered in my day to day work, I would rather illustrate a problem to be solved which has some real meaning to myself and to many other companies. The example shows the effectiveness of combining the SAS Webhound and Platform LSF software to cost-effectively process large amounts of web log data.

FROM DATA TO INFORMATION

Most companies today have a presence on the web and recognize that their web site is a major interface point with their customers (both potential and existing). Many of these companies invest significant resources in the infrastructure, content and human resources necessary to maintain their web site, and would like to know how much return on their investment they are getting from customer exposure to their web site. As such, marketing departments want to analyze the browsing patterns on their web sites to figure out which parts work and which parts don't, so that the maintenance investment can be focused accordingly.

The desired information is held within the log files of the web servers, which track the individual hits on the web site, but is not useable for analysis in its raw form. By using the Webhound software, the data in these web logs can be turned into information through reports of the activities on the web site. This includes information such as which pages are most popular, how long a user typically browses the site, and what is the typical sequence of pages a user visits before exiting the site. The generation of the reports from the web log data is a fairly compute intensive process; the larger the web log file, or the processing of multiple web log files, can take on the order of hours (see the timing for the sequential run in the results section).

The example is going to model a company which has to process multiple, independent log files. Independent log files means that each log file corresponds to a different set of reports, as opposed to multiple log files that are aggregated to generate one set of reports. This model is not uncommon. A company might have multiple web servers and domains which they need to process, either because the company has different sites (or sections of sites) for different business units within the company (one site for the corporate web site, one for the main product line, one for technical support, etc). Alternatively, the company might be a web hosting company that needs to provide reports for each individual customer (a customer corresponding to a domain, and thus a separate log file).

THE COMPUTING ENVIRONMENT

The environment used to run the LSF and Webhound software was made up of 7 different machines with the following characteristics and roles:

- One single CPU (1GHz Pentium III – 256Mb of RAM) Linux 2.4.17 machine serving home directories via NFS. Web logs were stored in my home directory.
- One 4 CPU (700 MHz Pentium III – 1Gb of RAM) Linux 2.2.16 machine serving a "webhound" file volume. All web marts, reports and temporary files during Webhound processing were stored on this node. This system is more I/O balanced than the home directory server (it has SCSI rather than IDE drives, and the 4 processors means there would be no nfsd processes waiting while 4 nodes were doing I/O).
- One 2 CPU (700 MHz Pentium III – 1Gb of RAM) Linux 2.2.16 machine serving as the LSF "master" host. That is, the master LIM and the scheduler were running on this node.

- One single CPU (440 MHz Ultra 10 – 1Gb of RAM) Solaris 8 machine. This was the fastest machine at running the Webhound software, and was thus used for the sequential processing timing run.
- One dual CPU (180 MHz HP J-class – 1Gb of RAM) HP-UX 11i (64 bit) machine. Due to the two processors and the large amount of RAM, this machine almost matched the Ultra 10 for speed. Unfortunately, the 100 megabit ethernet adapter would not work, so it was restricted to 10 megabit ethernet. Due to the I/O requirements, this meant I couldn't run two instances of the Webhound at once on this machine.
- Two single CPU (360 MHz Ultra 5 – 256Mb of RAM) Solaris 7 and Solaris 8 machines. These two machines generally could process log files at about 2/3 the speed of the Ultra 10 machine.

LSF 4.1 was installed on 6 of the 7 machines (all machines except for the machine serving home directories). A special LSF queue was set up to run the Webhound jobs in, and was used to set a default resource requirement which indicated that jobs should run on hosts which had Webhound installed (a boolean resource). The hosts were configured to only run one LSF job at a time.

SAS Webhound was installed on the three Solaris machines and the HP-UX machine. The default `sasv8.cfg` file was changed to set '`memsize`' to 256M on the 1Gb machines and to 128M on the Ultra 5 machines. In addition, because of the use of NFS (and the unreliable locking), '`filelocks`' was set to 'none' in all `sasv8.cfg` files.

DATA PARTITIONING

Typically, the most difficult step in developing a distributed version of a previously sequential processing activity is in splitting up the input data such that the data can be processed independently. In some situations there might be some kind of aggregation step at the end, but not in all cases. Many computing activities turn out to be "parameter space studies" where the same computation is performed on multiple sets of inputs. Both the input data and the results are independent of each other, so the processing is a natural candidate for distributed processing. For example, when using a program which implements the BLAST algorithm for genetic sequence analysis, each sequence to be compared to the database is the input, and the match or no match answer is the result. Both input and output are independent of other inputs and outputs, so many BLAST searches can be done in parallel with no interaction.

Since the web logs that will be processed in my test represent independent web marts, the processing of these web logs can be done in parallel and on different nodes.

If, on the other hand, my multiple web logs were just subsets (say based on week) of the same web server, I could only use one Webhound run to process the data at a time, since all processes would be modifying the same web mart.[1]

When using LSF, it's not required to make the partitions the same size. As long as the number of partitions is larger than the amount of processing resource needed to process the logs, the LSF software will "fill in" any gaps and keep the compute cluster busy.

There are eight log files that will be processed for this example, each representing a different web site, and thus a different set of reports to be generated. The log files for each site are listed in Table 1 along with their size.

Site name	Log file	File size (in Mb)
ex010605	ex010605.log	106
ex010701	ex010701.log	107
ex010702	ex010702.log	123
ex010703	ex010703.log	84
ex010704	ex010704.log	77
ex010705	ex010705.log	33
ex010801	ex010801.log	39
ex010802	ex010802.log	49

Table 1

Another important question to answer when creating data partitions is how to make the data available on any of the potential compute nodes. Since the scheduler will decide where a particular job will run, and this is not known *a priori*, placing the datasets on local disks on every node is not feasible, especially as cluster sizes get large (Platform Computing has quite a few customers with cluster sizes of over 1000 nodes). It is also desirable to have the results accessible from whichever node you will analyze the results on (perhaps your own workstation).

There are generally two approaches to making data available: using a shared file system, and staging data in and out of the machine. Staging data is preferable when dealing with very large sets of data that might be passed over multiple times during processing, or if the accesses to a shared file system are very latent due to many clients for a network file share. The down side to staging is that your program must provide more logic for dealing with the transfer of input and output files. The Platform LSF software provides some mechanisms for performing stage in and stage out activities through file transfer options, and pre and post job execution steps.

Shared file systems provide a much more flexible mechanism for making data available. In the case of my web log file processing, NFS file systems are being used so that the processing steps for one log file can be performed on different nodes if desired. With four NFS clients, the load on the network and file server did not seem to impact the runtime of the Webhound software. A test was run using the dataset for site "ex010705" to compare the run time of the extract, load, and report phases when the "`stayarea`" (the persistent files of the web mart) and "`scratcharea`" were on different combinations of local and remote file systems. The results are shown in Table 2.

`stayarea`	`scratcharea`	Run time (MM:SS)
local	local	41:10
NFS	local	42:56
NFS	NFS	39:34

Table 2

It's interesting to note that the run time was less when both file systems were located on NFS file servers. It shows that the run times were probably more affected by warm caches and incidental system load than in any significant way by the location of the file system.

If the number of nodes in the cluster kept increasing, there would be a certain point at which the file server or network bandwidth would become saturated. At this point, other data sharing mechanism would probably be necessary. File staging for one, or some combination of file servers whose volumes were shared from some kind of SAN environment.

JOB FLOW

Once the data has been partitioned, and the sharing strategy decided upon, consideration needs to be made for breaking up the processing into the tasks to be scheduled. This generally maps almost one-to-one with the data partitions, such that each data partition is processed using one LSF job. There are cases though, when breaking the job into processing steps is necessary or desirable.

For example, perhaps one step of the processing only runs "well"

[1] The Webhound software does provide a capability for performing some processing steps in parallel, but the discussion of those capabilities is not within the scope of this paper. It is worth mentioning that the settings used for the test made use of three report processes rather than the default of five. This was done to reduce the IO load on the file server.

on a subset of nodes in your cluster because of a need for large amounts of memory. Either you can run the entire job on only those nodes with enough memory, thus not making use of all the processors available, or you can subdivide the job such that the parts that need more memory can be scheduled to the subset of nodes with larger memory configurations.

The generation of Webhound reports from each log file is done in four different steps done in sequence:

- A web mart configuration step (which I call "config") in which the directory structure is set up and some parameters for the web mart are changed. This configuration step was necessary since the web marts were not configured using the Webhound administrative GUI, so the parameters needed to be set in a SAS software program which called the %WEBHOUND macro to assign the libraries, and then used a DATA step to modify the configuration parameters for the web mart.
- The "extract" step that reads in the web log file and performs some preliminary analysis. This step results in the generation of some temporary data sets that are then read by the "load" step. The SAS software program line to perform the extract step is:
  ```
  %webhound(data_store=/path/to/mart,
  program=extract);
  ```
- The "load" step that takes the data from the temporary data sets, and incorporates the data into the web mart's detail and summary tables. The SAS software program line to perform the load step is:
  ```
  %webhound(data_store=/path/to/mart,
  program=load);
  ```
- The "report" step that generates the static and drillable reports from the detail and summary tables. In general, I found that the report step took 75% of the processing time, and ran more quickly on the Ultra 10 and HP machines. The SAS software program line to perform the report step is:
  ```
  %webhound(data_store=/path/to/mart,
  program=report);
  ```

The log files for the example were processed in three different ways in order to compare different approaches. First, all eight log files were processed in sequence. Eight individual SAS software programs (one for each log file) were prepared that contained the configuration code at the beginning, and then called the %WEBHOUND macro to perform the extract, load, and report steps in sequence. Each of these SAS programs was then called in sequence from a Bourne shell script (called `process_all_marts_seq.sh`). Second, each log file was processed by one Webhound job that performed all four steps within one LSF job. A third approach was to submit each step of the Webhound process as individual jobs, using the inter-job dependencies in the LSF software to preserve the ordering. For this case there were four SAS software programs coded for each log file. The first program (`do_config.sas`) performed the configuration step. The second, third and fourth programs (`do_extract.sas`, `do_load.sas`, and `do_report.sas`) performed the extract, load, and report steps respectively. These three programs essentially were one-line programs that called the %WEBHOUND macro. The purpose of trying both the second and third approaches was to see if the scheduler might be able to provide better throughput with the larger number of component jobs.

RUNNING THE FLOW

In order to properly record the timing information for each job mix, all jobs (including the sequential run) were run through the LSF software. The LSF software provides detailed accounting records that could be used to determine how long the entire job mix took to run. The exception to this rule was the determination of the timing of the sequential run. In this case, the timing was taken from the SAS software log file, so that the latency of the batch system would not be taken into consideration (that is, I wanted timing information which was consistent with running the `process_all_marts_seq.sh` script from the shell command line.

For timing the jobs running across the cluster in parallel the elapsed time between the submission of the first job and the completion of the last job to finish was considered to be the timing for the entire mix. This timing include some latency incurred by the batch queuing behaviour of the LSF software, but this latency was included in the timing because I was timing how long until the results were ready, rather than merely timing the run time of the Webhound software.

Each mode of execution (sequential, job per log file, and job flow made up of four jobs per log file) was run at least twice to make sure that some transient system problem or some other transient system load didn't affect the timings. After each run, the resultant web mart directory and scratch directory were removed in order to create the web mart from scratch each time. There are two comparisons for which results can be examined. The first is a comparison of the sequential run, the "batch" run (one job per log file), and the "flow" run (four jobs per log file). Both the batch run and the flow run in this case are done using all four compute hosts in the cluster. The timing results of 2 runs for each mode are shown in Table 3.

Mode	Run number	Run time (in min)	Average run time (in min)
sequential	1	404	400.5
	2	397	
batch	1	145	145.5
	2	146	
flow	1	153	149.0
	2	145	

Table 3

It seems somewhat obvious, but by using four compute hosts rather than one, there is a 2.75 times speedup in the processing of the eight log files. What is most notable about the result is that there was very little time expended in preparing the Webhound software to run in the distributed mode. The four nodes were also workstation class nodes of the type that many organizations have in abundance, and the file servers were also not very expensive classes of machines.

Another interesting result shown in Table 3 is that there was virtually no difference in run time between running the job mix in batch and running it in flow mode. In fact, running the job mix using job flows appeared to have worse performance, probably due to the batch overhead incurred for each separate job.

You might wonder why there was only a 2.75 times speedup between the sequential and batch cases. This is due to the fact that the machines used to incrementally scale the computing capacity of the cluster were not as powerful (in terms of processing power and memory capacity) as was the machine used to run the sequential job. For instance, when processing the log file `ex010705.log`, the Ultra 10 machine could complete the job in 39 minutes and 34 seconds, while the HP machine could complete the job in 43 minutes and 37 seconds. They are close due to the fact that the HP had two processors that were exploited by the fact that the Webhound software was run with three report processes. The Ultra 5 machines, on the other hand, were quite a bit slower. In a couple of the flow runs the Ultra 10 machine performed the report phase of the `ex010703.log` in 45 minutes and 31 seconds, as opposed to the Ultra 5 which took 63 minutes and 19 seconds. This difference means that the Ultra 5 machine take about 40% longer than the Ultra 10 to process the same report phase (which dominated the processing).

This leads us to the second comparison that can be made between the run times. It is useful to compare the incremental speedup by adding compute resources incrementally to the cluster. The speedup in running the processing in sequence on one node (the fastest node), to running the processing on the two fastest nodes in the cluster (the Ultra 10 and the HP), to running on three nodes, to running the on all four nodes in the cluster. Table 4 shows the average run time for these four cases, and the results are graphed in Figure 1.

Number of processing nodes	Average run time (in min)	Speedup (1 node run time / run time)
1	400.5	1
2	239	1.67
3	194	2.06
4	145.5	2.75

Table 4

Note that the speedup is not linear, but this can be attributed to the different configurations of the workstations making up the cluster.

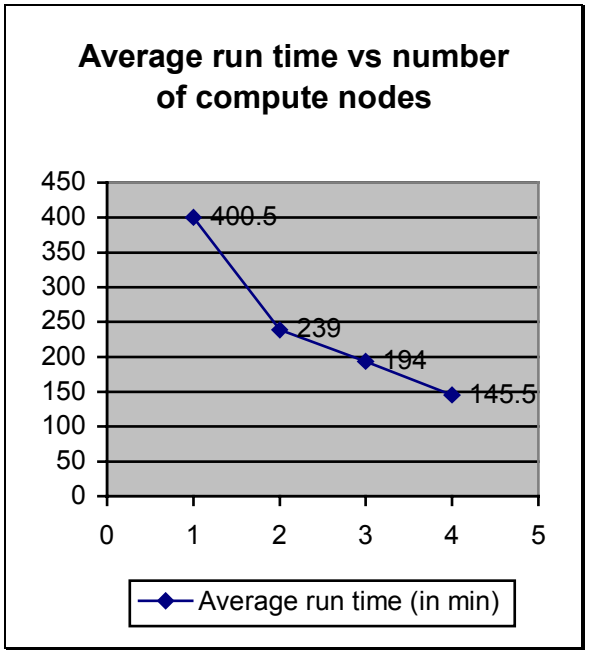

Figure 1

CONCLUSION

Many organizations have an abundance of computing resources available, but cannot make use of them due to the lack of some kind of management layer that can allocate workload to free resources. This is exacerbated by a "host-centric" approach where individuals manually place workload on familiar, yet overused hosts.

The Platform LSF software provides a layer that manages the complexity of matching supply to demand, and that provides transparent access to these networked computing resources, allowing users to think in terms of resource requirements rather than hosts.

An example of the use of the LSF software for distributed processing was presented which used the SAS Webhound software to process web server log files into reports of web site metrics. It was shown that with a very simple computing environment made up of a cluster of machines using shared file systems, and with no changes to Webhound or to the web log files, multiple web log files were processed in a shorter time using the LSF software than by manually running the processes in sequence on a single node.

ACKNOWLEDGMENTS

I would like to thank the following people for their help during the preparation of this paper: Steven Wartofsky and Tricia Dudley of the SAS Institute for their help in getting me copies of the Webhound software, and the assistance in getting it running; Darren Healey and Vartex Nicolian of Platform Computing for their help getting me some real web log files to process; plus Jennifer Georgas, Sylvia Smellie, and Ian Lumb of Platform Computing for their help with paper revision.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Chris Smith
Platform Computing
25 Metro Dr., Ste. 100
San Jose, CA USA 95110
Telephone: 408-392-4938
Fax: 408-392-4905
Email: csmith@platform.com

Platform™
The Bottom Line in Distributed Computing