J90 SuperCluster - A Progress Report

Bruno Loepfe, Computing Services, ETH, Zurich Switzerland
Dr. Svend Knudsen, Computing Services, ETH, Zurich Switzerland
Dr. Rudy Wilopo, SGI/CRI, Switzerland

ABSTRACT: *SGI/CRI and ETHZ have reached an agreement to jointly implement a cluster. The project* SuperCluster Development/Operation *is part of the agreement A* SuperCluster *should give the user a* Single System View *and provide extra value beyond just providing the computing capacity.*

The project SuperCluster J90

History

The project *SuperCluster J90* is part of a co-operation agreement between SGI/CRI and ETHZ. The goal of this project is to implement a Single System View (SSV) on a cluster of computers. When users work on a cluster equipped with SSV, they should get the look-and-feel of working on a single system. From the implementation point of view, SSV is a collection of services provided by the machines in the cluster. As far as possible these services should not be bound to a particular machine. Of course, certain services rely on a particular piece of hardware. Tape-services, for example, need access to some kind of tape-drives; but in most cases not all machines in a cluster are directly connected. Apart from dependencies like these, the cluster should not rely on a particular machine providing any given service.

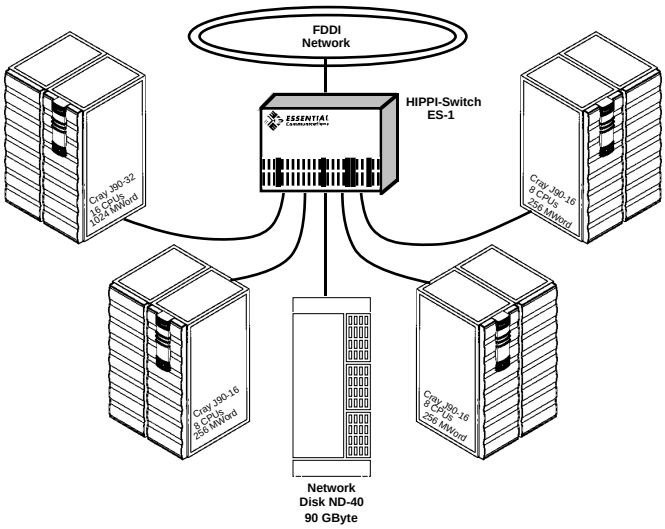

Figure 1: Cluster with 4 J90, HIPPI-switch and network-disk

The advantages for the users are twofold. First: they don't need to care which machine in a cluster they should address in order to use a particular service. Second: hardware or software failures don't lead to a loss of all services like on a single machine. Instead, the SuperCluster should be able to continue uninterrupted operations for unaffected services. After a hopefully short amount of time to recover a lost service, the SuperCluster should provide full functionality again, although perhaps with a certain loss of performance.

Single System Image

Single-System-Image (SSI) is a superset of SSV. The former provides additional functionality like process migration. As a consequence, SSI is restricted to homogeneous clusters, while with a few restrictions, SSV can be operated on heterogeneous clusters as well..

Basics of Single System View

We identified the following basic topics for SSV:

- clusterwide transparent file access
- job-distribution (batch and interactive)
- administration / accounting / operations

Of course some of these topics are strongly correlated. For example: should you encounter problems with file access during operations, proper emergency procedures are needed to fix them.

Clusterwide Transparent File-Access

What we intended to do

For reasons of transparency and resiliency we decided to provide clusterwide file-access with the Shared Filesystem (SFS), cached by the Distributed Filesystem (DFS). On a dedicated system running one copy of our I/O-benchmark, we saw 75% or more of the performance of an NC1-filesystem. It turned out, however, that this combination doesn't perfom well on a loaded system. We ran 12 copies of the I/O-program

simultaneously (on one machine as well as on different machines) and found a satisfactory transfer-rate of 10-40 MBytes/sec (depending on the size of the I/O-requests). However, this is the plain transfer-rate only. If we take into account the time for opening and closing the files as well, the average transfer-rate drops to unacceptable 15 KBytes/sec in our tests. In addition we found that it takes up to 7 minutes to delete a single file. Consequently, we abandoned SFS and started to look for alternatives.

Alternatives

Our HIPPI-disk gives us the unique capability of mounting its file-systems on any machine in the cluster. Of course, a particular file-system must be mounted on one machine only. It is our decision, on which machine we mount a particular filesystem. Therefore, we are able to distribute the filesystems over all machines in the cluster and even to change this distribution dynamically. We just need a way to export the locally mounted filesystems to other machines in the cluster. The choices for exporting are NFS and DFS. We decided to use DFS because of its local caching capabilities and its ease of moving exporters to other machines. Resiliency, however, will suffer compared to SFS: we will lose all jobs/sessions in the cluster which try to access a particular filesystem throughout the duration of its move. SGI/CRI is aware of this problem and there are some ideas to solve it.

Actual state

The clusterwide file access is still implemented via NFS from one central exporter. The next step will be to distribute the NFS-exporters over all machines. For the switch to DFS we have to wait until DCE-ticket-forwarding is available in NQE. Independent of the method for accessing files, DMF is available on all machines. We replaced the commands *dmget* and *dmput* with our own versions. These commands simply contact the machine exporting the filesystem, perform the operation there, and wait for the local DMF-daemon to finish the operation. This, of course, requires that machines exporting DMF-filesystems be equipped with tape-drives. Additionally, the layout of the filesystems has to be identical on all machines in the cluster, but this is already mandatory for clusterwide file-access. The reason for not running distributed DMF is that it would require an SFS-based filesystem for storing its databases.

Job-Distribution (batch and interactive)

What we intended to do

We thought this would be the easiest of all topics: we didn't intend to do anything at all. Interactive sessions don't contribute considerably to our overall load, so we don't really care about distributing them. For batch-jobs, we had thought we only needed to start NQE to benefit from load-balancing. We discovered, however, that for the purposes of the SuperCluster, load-balancing via NLB is not sufficient. The SuperCluster is a collection of services running on different machines providing a Single-System-View to its users. NLB is basically a linear combination of thresholds and other values. We found that most of the time identifying the hosts providing a particular service requires more flexibility than linear functions are able to provide.

What we did instead

NQE gives you the possibility of writing your own scheduler. In this way the entire process of making decisions is under your control. Additionally, the scheduler is written in a programming language. Therefore you have far more functions available than just linear ones. The documentation, however, is still in an early state. So SGI/CRI sent two NQE-experts to Zurich for a week to help us design our own scheduler. They left us a framework strongly based on what we told them we would like to have. For example: we moved all the checking and enforcing of job limits, as well as the inter-queue-scheduling and intra-queue-scheduling, from the local NQS-queues to the NQE-scheduler. Another example: the default-scheduler of NQE would accept a job with the *-a*-parameter and schedule it for an appropriate LWS. There, the job would wait in the local queue until its execution time. We let this job wait in the scheduler instead, so that the scheduling decision is based on the state of the cluster at the time the job is intended to start, instead of at the time the job was submitted. Our scheduler isn't finished yet; the task is considerably more difficult than we expected.

Actual state

While we are waiting for a version of NQE with DCE-ticket-forwarding, we are slowly upgrading our 70+ remote-NQS-stations with NQE. As soon as the load-balancing is ready, we will start to distribute the NFS-exporters over the cluster. It makes little sense to do this before load-balancing is running. It would just slow down the access to filesystems residing on other machines.

Furthermore, we are supposed to measure the quality of the scheduler. Here I have to confess that we don't have the slightest idea how this should or could be done. At this point it's unclear, what an appropriate metric should look like.

Another unsolved problem concerns MPI/PVM and NQE. The spawning of processes from a job to other machines defeats the purpose of the NQE-scheduler. We therefore need a way to have this creation of remote-processes under control of NQE.

Administration / Accounting / Operations

We didn't do anything about accounting. Much to our own surprise, we spent a lot of time on administration and even more so on operations.

Administration

Here we faced a very boring situation: whenever users were forced to change the password for their accounts, either they did it on the production machine and had the system staff copying the UDB to the other machines, or they changed the password on each of the four machines. In both cases they also had to change the DCE-password. Meanwhile we provided a program named *cpasswd*. This changes the password on all machines in the cluster, as well as that for the DCE-account. Should any machine in the cluster be unready to accept a change, this change will be buffered on the machine which

issued the command. During the boot-process, each machine asks all other machines whether there are any buffered changes pending. This way we ensure the consistency of the passwords for all acounts everywhere in the cluster including DCE.

Operations

Most of the time we spent concerned operations. To give some examples:

- as mentioned before, we intend to distribute our filesystem-exporters over the cluster. Not all machines are equipped with tape-drives, however. So how do we dump/restore filesystems exported from machines not equipped with tape-drives ?
- for some reason a particular machine has to be taken out of operation. Of course, we don't want to lose the filesystems it has exported. How do we move these filesystems and their exporters from one machine to another without losing too many jobs during this process ?
- one of the machines is equipped with four tape-drives. How do we connect these drives to another machine if the previous tape-server goes down ?

The first example was relatively easy to solve: we extended our backup procedure with the ability to find the exporter for a given filesystem. It takes a list of filesystems, figures out which machines to ask for a dump, and manages the VSN-numbers of the cartridges. One sub-problem isn't solved yet: the program *dump* stores a timestamp for each filesystem it dumps in the file */etc/dumpdates*. Whenever we need to move an exporters, we therefore need to move the corresponding contents of */etc/dumpdates*.

The program is currently based on sockets, but we will very likely rewrite it using MPI. The handling of communications in MPI is much easier than using sockets.

The other two examples mentioned above, like most of the problems in operations, fall into the following general category:

- How do we gracefully move a service from the machine originally providing it to another machine, and how do we move it back later ?

Unfortunately we found in most cases: this is impossible, at least when you insist in the word **gracefully**. Neither NFS nor DFS on Cray machines provide means to move exporters to other machines during operations. In the case of NFS you might even need to kill jobs on a machine if you have to re-import a filesystem from a new exporter. With DFS you are at least able to move exporters without being hindered by *Stale File Handles*, but the procedure is quite complicated. DFS provides the possibility to move exporters gracefully. However, this requires the use of the Enhanced LFS for DCE/DFS which is not implemented for Cray machines.

Another example is NQE: its master resides on a particular machine but no provisions have been made to move it gracefully to another. We managed to find a way to move it, but it requires that the NQE-database resides on its own filesystem (in our case located on the Hippi-disk) so that we are able to mount this filesystem on another machine. The tape-drives are even worse: the only way we could find to connect them automatically to another machine is a two-way SCSI-switch. The worst case happens when we need to move the tape-server. We then also need to move DMF, as well as the exported filesystems, which are typically DMF-filesystems. We did it once manually, step by step. The final goal, however, is to have this move triggered by our external watchdog, should it detect the loss of the tape-server. This is not implemented yet.

Wish-List

Several of the problems mentioned before led us to the following list of wishes to SGI/CRI:

1. make sure services are not bound to a particular machine
2. provide better integration of the various SGI/CRI-software-packages.

This list is fortunately very short. Nevertheless these two points are essential for SuperClusters as well as for regular clusters. The motivation for wish number 1 has been included previously. It is possible to achieve this goal but you have to take care of it very early in the design phase of a software-package. Our clusterwide password-changer, for example, is not bound to a particular machine. It doesn't need special hardware, so it can run everywhere. It has been designed to deliver this functionality.

The reasons for number 2 are the various problems we have encountered so far. For example: none of the software packages is able to cooperate with DCE. This is a real problem for NQE and MPT: both lack the forwarding of DCE-tickets to their child-processes. MPT and NQE don't cooperate either: the creation of new processes by MPT is not under control of NQE and accordingly the scheduler of NQE is unable to consider new processes when making its decisions.

Conclusion

Based on our experiences so far we think that SSV is an excellent vehicle for combining computers into a throughput-delivering device. By writing an NQE-scheduler, one has sufficient control over the process of distributing jobs to take full advantage of all the features the machines in the cluster provide. These features can be enhanced dynamically by adding new services (hardware/software) with little impact on the users.