SuperCluster SV1: The Next Step

Bruno Loepfe, Computing Services ETHZ, Zurich, Switzerland
Dr. Olivier Byrde, Cray Inc.

ABSTRACT: *In the first two phases of the project* SuperCluster SV1*, we dealt primarily with the user aspects of a SuperCluster. Now in phase three, the main focus is operational aspects. Three subjects will be presented based on our experience: NQE versus LSF, clusterwide dump/restore, and clusterwide resiliency features.*

The project SuperCluster SV1

History

The project *SuperCluster SV1* is part of a co-operation agreement between SGI/Cray 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 to access 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.

The advantages for the users are twofold. First: they do not need to care which machine in a cluster they should address in order to use a particular service. Second: hardware or software failures do not 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 special case of SSV that 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.

Phase 1

In phase 1 of this project we identified and implemented the basic topics for SSV:

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

In this phase we found out that operational aspects and administration would each need their own phase.

Phase 2

In phase 2 we started to deal with operational aspects. The basic question in this phase was:

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

Although we already started with phase 3, this phase is not finished yet.

Phase 3

Currently most of our work belongs to phase 3: operational aspects and administration. In this paper we will discuss three topics, chosen from a wider range of problems.

NQE versus LSF

General remarks

Network Queueing Environment (NQE) and Load Sharing Facility (LSF) are both available on Cray systems. We do not intend to describe too many implementation details of each batch system. For the purpose of this paper it is sufficient

to know that both accept jobs from users, execute them, and deliver job output back to the users.

Both batch systems consist of a set of co-operating collectors and daemons, spread over all machines participating in the cluster.

In the following discussion we describe a couple of advantages/disadvantages of LSF version 3.2.2 for UNICOS and NQE version 3.3.0.15 for UNICOS. As soon as new/ improved versions of these systems appear, some of the points mentioned hereafter may become obsolete. In addition, the lists below are in no way exhaustive and are strongly biased towards the way ETHZ runs its SuperCluster.

Positive aspects of LSF

Probably the two most important positive aspects of LSF are its scalability and the possibility to move its master daemon to another machine. In fact, if the LSF master daemon ever goes out of service, intentionally or as a consequence of a crash, LSF will detect it and automatically recover from this loss by designating a new master daemon from the remaining ones.

LSF has implemented its queues at the daemon level, as opposed to NQE where the queues reside on the execution machines. This way LSF is able to exercise a much tighter control over the jobs in its queues. This in turn enables it to provide a feature called *preemptive queues*, which effectively allows a new job to stop an already running job and to take over its resources.

LSF has DCE, PVM and MPI support built in. For jobs using PVM or MPI, LSF provides a special submission command which ensures that these jobs and the allocation and distribution of their subprocesses stay entirely under control of LSF.

Positive aspects of NQE

NQE contains a scheduler, written in Tcl. This scheduler is a collection of algorithms for bookkeeping and making decisions. Since Tcl is an interpreted language, the scheduler can easily be adapted to fit any particular scheduling needs of a site, even while NQE is running.

NQE features freely definable job attributes. Of course the scheduler will need some adaption in order to react properly. But a site is completely free to define any attributes it needs.

All spooling for a job is local to the execution machine. This means a job gets the full I/O performance for spooled I/O, no network traffic is generated.

Negative aspects of LSF

Version 3.2.2 of LSF runs under UNICOS, but does not yet support some of the important features of UNICOS. Among other problems we found that jobs created by LSF always get unlimited resources, despite any limits the user has set during the job submission. Default limits are not provided.

A less important problem is the statistics generated by LSF. For example: the memory usage of a job is reported in Mbytes, instead of Mwords, and it is incorrect.

The administrator should take a lot of care when configuring LSF. For example: the spooling directory of an LSF job is *$HOME/.lsbatch*, which in a cluster most likely is imported by the execution machine from a file server. In order to avoid heavy network traffic for jobs generating a lot of spooled output, the administrator should preallocate this directory for all users, so that it points to a public local spool area on each machine in the cluster.

The only way to influence scheduling decision in LSF seems to be the set of parameters of the submission command. The administrator has very few possibilities to directly access and influence the built-in scheduling algorithms.

Negative aspects of NQE

Several features offered by LSF are missing in NQE and have to be implemented in the scheduler using job attributes. Among these are for example the routing of jobs requiring licenses for commercial packages to the appropriate machine(s), as well as the serializing of a set of jobs, where a particular job is not allowed to start before certain other jobs have finished.

NQE does not provide any support for PVM and MPI jobs. This means that new subprocesses, remote as well as local, are not under the control of NQE.

Despite its name, Network Queueing Environment, the master daemon of NQE resides on a particular machine of the cluster and is not movable. Should this machine need to leave the cluster, be it for maintenance or due to a crash, the operations staff will have an overly hard time to restart the master daemon on another machine in the cluster. And this has to be done manually, no provisions are made to assist the staff during this process. In order to allow this move, it is necessary to take great care during the initial setup of NQE. Unfortunately the documentation does not give any information on this topic.

The most important negative aspect of NQE, however, is its lack of scalability. The underlying database can run on a single CPU only. According to our experience, this CPU reaches saturation at a rate of about one request per second to the database. A request can be a job submission, a status request from a user, or various internal events and state changes. Unfortunately we found that it is possible to reach this limit with a cluster of two machines already. While NQE is an excellent batch system for single machines like T90, T3E

and single J90/SV1, its current implementation will not be able to handle clusters of more than four machines with lots of users and the traditional mix of small, medium and big jobs.

Clusterwide dump/restore

The problem

Figure 1 shows a stripped-down picture of the cluster machines at ETHZ. The two machines on the left side, P1 and P2, form the production cluster. The two machines on the right side, T1 and T2, form the test cluster. The big GigaRing, connecting all four machines, allows to join both subclusters and to form a bigger cluster. Typically this is done for scaling tests. Usually the machine P2 acts as tape and file server. If necessary, the tape drives can be moved to P1, the filesystems can be moved to any machine in the cluster.

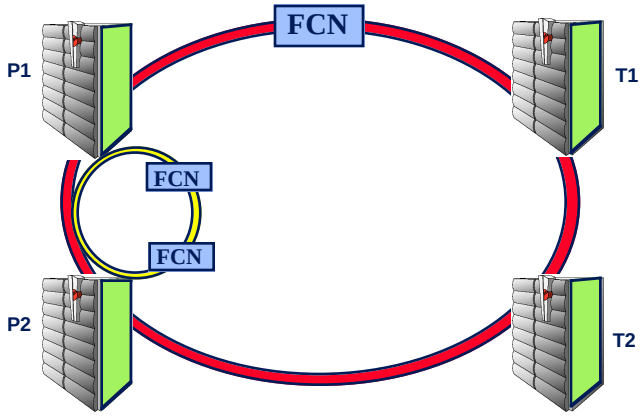

Figure 1: SuperCluster at ETHZ

Since only P2 is equipped with tape drives, we were able to take tape dumps on that machine only. Of course we would have liked to take dumps from other machines as well, for example for storing the actual OS and its configuration from each machine.

The command *rdump* is supposed to deliver the functionality of taking a dump and storing it on tapes on a remote machine. Under UNICOS, however, we could not get it to work. It seems that *rdump* tries to access the tape devices directly, whereas under UNICOS the tape daemon hides them and therefore they are not directly accessible under normal circumstances.

Another way of taking dumps and store them on tapes on a different machine is to store the dump on a network medium, like an NFS exported filesystem, and copy it to tapes later. First, this requires sufficient free space, which cannot always be guaranteed. Second, we found that NFS achieves transfer rates in the order of 10 - 12 MB/sec over GigaRing, which is not sufficient for practical purposes.

The requirements

Due to the above limitations, we decided to create a utility that would fulfill our needs. According to the SuperCluster concept, the new command should be able to:

- run on any machine in the cluster, regardless of the location of the tape or file server
- find the tape server without user intervention
- find the file server and exporting protocol (NFS/DFS) without user intervention
- convert NFS/DFS directories to disk devices, so that the user does not need to consult a disk layout before starting a dump
- mimic the original *dump* as close as possible

The programs cdump/crestore

With the exception of recognizing the DFS protocol and DFS directories, the new program *cdump* delivers the functionality mentioned above. Now it is possible to log on any machine in the cluster, start *cdump*, and have any filesystem (local or remote) dumped to either a file on any machine in the cluster, or to tapes on the tape server. Benchmarks under non-dedicated conditions showed a transfer rate of 50 - 120 MB/sec. This number shows a variation too big to be really meaningful. But it shows that the network transfers are not really a bottleneck.

In the trivial case where the tape devices reside on the same machine as the filesystem to be dumped, *cdump* uses the original *dump* to do the work. In this case the overhead of *cdump* compared to *dump* is neglectible.

crestore, the inverse operation of *cdump*, is not implemented yet. We found it very difficult to implement the interactive mode of *restore*. So far we have not been able to come up with a more efficient solution than reading the dump file twice. But reading dumps twice will cause a major overhead; in particular for large tape dumps this overhead is not acceptable. We still hope to find a better way to implement this feature, but maybe we will have to drop it.

Clusterwide resiliency features

Because of the multiplicity of its elements, a SuperCluster is more likely to be the subject of a component failure than a conventional, monolithic system. On the other hand, the inherent redundancy of the SuperCluster makes it very robust; an isolated failure will generally not bring the whole cluster down.

The term *resiliency* denotes the ability of the SuperCluster to sustain such a component failure, and possibly to recover from it. In the present context, we will consider only the

situation where the component is a node; i.e., a whole SV1 cabinet. Indeed, those situations where the faulty component is a CPU or memory module are normally handled (more or less gracefully) by the operating system running on the machine where the event occured.

At the SuperCluster level, two scenarios are of particular interest: node maintenance and power failure. First, when hardware or software maintenance is required, the administrator should be able to intervene on any node of the SuperCluster with minimal impact on the production environment. Second, in the unlikely case of a power failure on one node, the SuperCluster should be able to survive that event, detect and isolate the faulty node, and continue to operate normally until that node is functional again.

Both scenarios have been tested in a real production environment using the SuperCluster at ETHZ. The two production systems and the two test systems were all connected together via the common GigaRing, although for obvious reasons the actual tests (reboot, power off) were performed on the test systems only. (It must be noted, however, that traffic over GigaRing between the production and the test systems was virtually nonexistent.)

In the first test, one node was brought down to single-user mode and then rebooted. The node was automatically folded out of the GigaRing (that is normal shutdown procedure) and then manually folded back in. This whole operation had absolutely no effect on the other three nodes on the GigaRing, which continued to operate normally.

In the second test, one node was turned off via the master circuit breaker, thus simulating a power failure. Not surprisingly, the effect on that node was dramatic: all processes and running applications were terminated, all disk and network activity was abruptly stopped, causing certain data loss. However, the GigaRing controller was able to detect the loss of that node and successfully folded it out. The effect on the other three nodes was minimal; only those data being exchanged with the first node were lost. The node was then powered back on and rebooted, from which point the situation is similar to that described in the first test.

These tests have shown that a temporary failure can be detected and sustained by the SuperCluster without much impact on its normal operation. However, the problem becomes more complex if the failure persists over a longer period of time and if the faulty node is handling global services such as NFS, NQE or DMF. In this case, a special procedure is required to migrate these services to another node in the SuperCluster. How this could be achieved as gracefully as possible, was the main topic in phase 2 of this project SuperCluster SV1.

Conclusion

In the framework of a co-operation agreement between SGI/Cray and ETHZ we have studied concepts and implementation of various aspects of Single System View (SSV) for clusters. Based on the experience gained during the three phases of this project, we think that SSV is an excellent vehicle for combining computers into a throughput-delivering device for all people involved: users, operations staff and system administrators.