Trust Separation on the Cray XC40 using PBS Pro

Sam Clarke
Met Office
Fitzroy Road
Exeter, UK
Email: sam.clarke@metoffice.gov.uk

***Abstract*—As the UK's national weather agency, the Met Office has a requirement to produce regular, timely weather forecasts. As a major centre for climate and weather research, it has a need to provide access to large-scale supercomputing resources to users from within the organisation. It also provides a supercomputer facility for academic partners inside the UK, and to international collaborators. Each of these user categories has a different set of availability requirements and requires a different level of access.**

This paper describes the steps taken to create an HPC facility that separates these different requirements using soft partitions created by the batch system. We detail our initial experiences with cgroup containers and our use of custom PBS hooks to partition the Lustre name space. We summarise some of the problems observed during implementation, comment on the scalability of the solution and outline possible future enhancements.

***Keywords*-Scheduling; PBS Pro**

I. Introduction

As a leading centre for weather and climate science research, the Met Office supercomputer installation supports a large scientific community. Much of this work is focused on developing new features of the various geophysical models used by the organisation for weather forecasting and climate research.

In addition to research priority work, the Met Office also has a requirement to run large, time-critical operational forecasts. Each forecast consists of a complex suite of dependent jobs, many of which interact with external systems in order to source observations and to transfer resultant forecasts to customers.

The Met Office has strong links with a number of external scientific organisations both nationally and internationally. There are particularly strong links with the Natural Environment Research Council (NERC), the non-governmental body responsible for supporting environmental research in the UK, and since 2009 the Met Office and NERC have been involved in the MONSoonN project which involves the provision of a series of dedicated supercomputers for joint research partnership[1].

The Met Office also works with a number of other national meteorological centres who have chosen to join the Unified Model Partnership, using a shared model to support their own forecasting and climate prediction services[2].

Each of these three separate categories of work — operational forecasting, internal research, and external collaboration — requires a different set of features from the supercomputer and needs a different level of access to both the data held on the system and to other facilities within the Office.

Given the overlap between these groups and the need to dynamically change their relative shares of computational resources, it is not practical to use static partitioning. This is especially true where some research groups may have a requirement to run large and short simulations to test scalability while others need to run small but long-running scientific studies.

In addition to the dedicated MONSooN facility, the Met Office has two significant Cray XC40 systems dedicated to operational weather forecasting, climate research, and weather science. Access to these systems is restricted to internal users. It has recently acquired a third Cray system intended to replace the existing MONSooN hardware whilst also supporting operational and research work on the same system.

In order to maximise the flexibility of the new system the Met Office has chosen to use a dynamic partitioning scheme to divide the XCS system rather than rely on hardware partitioning. This makes it possible to divide the entire 6,600 node Cray XC40 between the three different groups of users mentioned above using the facilities provided by PBS Pro and the Cray Linux Environment.

II. Design Decisions

During the initial design phase, it became clear that the standard Met Office system architecture used to configure the existing XC40 systems contained a number of resources which were shared by all users of the system. These included the external login nodes, the Lustre file system, and all the XC40 nodes.

The design analysis also revealed a number of common software resources, such as the Unified Model code base, which were required by a large number of users but which had previously needed to be manually synchronised between internal supercomputers and those used for collaboration. It also indicated other areas, such as user account management,

where administrative effort was being duplicated between internal and external systems.

Once these constraints had been identified, it became clear that the shared resources would need to be separated into different zones of trust based on their point of origin.

Users in the collaboration group, most of whom are based outside the Met Office and who access the system via a proxy from the internet, need the least level of trust. Their level of file system access can be limited to their own data areas and read-only access to the shared code base directories. They should not have access to internal Met Office systems.

Users in the research group, all of whom are based in the Met Office and have direct access to the system, have access to their own files, to those in the collaboration zone, and to the shared code base. They are allowed limited access to specific Met Office systems, such as the observations store, required to allow them to pursue their research interests.

Users in the operational group, composed of an extremely limited subset of Met Office users, only need access to their own files. They have a need to be able to access some internal systems, such as the observations store. They also have a requirement to run jobs at particular times and to ensure that competition for resources with other jobs, including other operational jobs, cannot deny them access to the resources they require.

III. Identification of Node Roles

The Cray system is composed of four different types of node directly visible to end users of the system. Of these nodes only the external login nodes are accessed interactively by the users; all other nodes, internal to the XC40 mainframe, are accessed in batch.

A. External Login Nodes

Each of the existing Cray systems has been configured with an active-active pair of external login nodes. These provide general interactive user services and act as a point of entry to the PBS batch system.

Having decided to divide the system into three separate trust zones, the number of external login nodes connected to the XCS was increased to six with two nodes per zone. Each of these hosts all the users at a particular trust level, providing a physical separation between processes at different levels. Each category of external login is connected to a specific network, with in- and out-bound access to each of these networks restricted by firewall rules and routing settings.

As all new PBS work must enter the system through one of the external login nodes, these points of origin can be used to identify the trust level of each job. And since the point of origin is resolved on the PBS server and not the client where the `qsub` command was run, it is not possible for the user to change the apparent point of origin.

Since each pair of login nodes is statically assigned to a particular trust zone, it is possible to configure the node with the required bind mounts to the trust zone specific portions at boot time once both Lustre file systems have been mounted.

B. XC40 MAMU Nodes

Much of the work run on the Met Office supercomputers requires the ability to run serial jobs in parallel with compute jobs. These serial tasks, which typically involve archiving, file management, post-processing, and compilations, account for the majority of the jobs run on the system. To support this work, the XCS system has been configured with a hundred MAMU nodes.[1]

On the existing internal systems, where all work is run at the same trust level, the MAMU nodes are available to all users. A small number of these nodes have been marked with an operation resource to separate operational tasks from general user work. There are no additional constraints on the resources available to individual jobs on these nodes; rather it is assumed that these jobs are well behaved and will not exceed the CPU and memory resources they have requested via PBS.

In order to implement trust levels, it is important to assign work submitted at different levels to different groups of nodes. It was also apparent that the current *laissez faire* approach to resource management was not sustainable and a stronger method was needed to ensure process separation.

As all MAMU nodes have access to the Aries high speed network, it is important to limit communication between them and other nodes in the system. This ensures that jobs running on MAMU nodes at different trust levels cannot communicate with each other in a way that might allow them to bridge trust zones.

C. XC40 MOM Nodes

Each of the Cray systems installed at the Met Office has been configured with 10 MOM nodes.[2] These execute the serial portions of the PBS job scripts and host the `aprun` commands used to launch parallel executables on to the set of compute nodes allocated to the job PBS.

As with the MAMU nodes, it is important to assign work submitted at different levels of trust to a different group of MOM nodes. Similarly it is important to prevent the MOM nodes from communicating with nodes at different trust levels across the network. Unlike the MAMU nodes, the MOM nodes must be free to communicate with the compute nodes in order to execute applications on the compute nodes.

[1]MAMU nodes are re-purposed compute nodes that have been booted with a service node CLE image. The nodes are defined as serial in PBS by setting the `vntype` resource to `cray_serial`

[2]These are re-purposed compute nodes booted with a service node image and distinguished in PBS by a `vntype` of `cray_login`

D. XC40 Compute Nodes

When a job asks for compute nodes from PBS, the actual request is satisfied by ALPS which returns a list of nodes that the job is allowed to use. These nodes are allocated exclusively to each job. This means that there is no need to explicitly partition the compute nodes — which comprise the vast bulk of the 6,600 nodes on the system — so they do not need to be statically assigned to a particular trust zone.

When a compute node is assigned to a job at a particular trust level, it must be reconfigured to allow access to the correct file systems for the trust level. And whenever a job completes, the trust-zone-specific file system bindings must be removed and the compute node must be returned to its original un-configured state ready for its next job.

IV. Implementation

The implementation of the trust zone model can be divided up in to separate stages, with the first encompassing the configuration of the user environment and Lustre, and the next concerning the changes to the batch system required to tailor each job and each job's executing environment to its requested trust zone.

A. User Set-up

Once the decision had been made to create three different zones of trust, each supercomputer user was assigned to a particular zone based on their usage of the existing systems. Each user was assigned a secondary group which matched their trust level. Where an internal Met Office user was also involved in work on a MONSooN project, they were assigned to both the research and the collaboration group.

Users within the collaboration zone were also allocated to a group matching each of the research projects they were involved with. This group allows them to place access controls on shared project data. It can also be used by the batch software to determine whether a user is allowed to charge compute resources to a particular project.

Where users and groups had previously been defined using static files on the small MONSooN system, a decision was made to use LDAP to provide identification and authentication services on XCS. This was done using the SSSD software on the XC40 and external nodes, and standard LDAP features on the Sonexion storage.

B. Lustre File Systems

Both the MONSooN and XCS systems have been configured with two Lustre file systems. The first of these file systems is relatively small and is used to hold user home directories and configuration files. The second file system is much larger and is used to hold model output data. Much of the model data is transient, with most of it transferred to an external archive in parallel with each model run.

In order to separate the system into different trust levels, it is necessary to prevent users at one level from accessing files at another. Due to hardware constraints and for performance reasons, it was necessary to achieve this separation without the need to create additional file systems.

Lustre services on the system are provided by two Sonexion 2000 appliances. These support the standard POSIX discretionary access controls but do not provide any mandatory controls at the file system level which might be used to prevent a file owner from exposing their data to users at different levels of trust. However, we found that by combining standard discretionary controls with bind mounts, we were able to create a facility which provided an approximation of mandatory access controls.

In order to limit the visibility of the file systems, we first created a set of directories for each trust level. The root of each of these directory trees was given an extremely limited set of permissions and ownership was set to the superuser. This effectively prevents normal users from accessing anything under the directory tree using the Lustre mount point.

Underneath each of the restricted directories, we created a normal directory structure. For example, under the home directory tree for the research trust zone, we created directories for each user with the necessary permissions to allow them to access the contents as usual.

On both the Cray XC40 and the external nodes, we mounted the Lustre file systems as normal. Depending on the partitioning scheme and the node type, we then run a series of mount commands to bind the restricted locations in the file system to a user accessible directory.

For example, on an external login node which has been marked as a member of the research trust zone, we might bind `/small/research/home` to `/home`. The permissions on the `/home` directory allow the user access to the contents whilst also hiding the details of the trust zone directory from the user. This has the added benefit of providing the user with a consistent set of directory paths regardless of the trust zone.

A similar method can be used to configure static bindings on the user-accessible nodes of the XC40, such as the MAMU and MOM nodes, where the trust zone of a particular node is not likely to change.

Although the same method of bind mounts can be used on the Cray compute nodes to provide access to a specific sub-tree of the Lustre file system, these bindings cannot be applied statically. Rather, an appropriate set of bind mounts must be created on a set of nodes before PBS passes control to a job and removed again when each job completes.

V. PBS Implementation

With the exception of the external login nodes, all access to the Cray XC40 is via PBS Pro and this makes it possible to delegate much of the implementation of the trust zone model to the workload manager.

PBS contains a number of hook points which can be used to trigger code at particular points in the job life cycle. These

can be used to modify each job as it is submitted to ensure that its requested resources and attributes adhere to local site policies, and to update the job as required. Hooks can also be used to trigger events on the executing node prior to the start of and immediately after each job, making it possible to tailor the environment for a specific job.

A. Identification of Resources

The PBS server supports the creation of arbitrary string resources. These can be used to assign a value to one or more nodes and similar values can be requested by each job as it is submitted. When these resources are added to the PBS scheduler, they impose a constraint which prevents jobs which have requested the resource from being executed until a node with a matching resource becomes available.

In order to implement basic separation of trust levels, a string resource was created to hold the zone value. This resource was then assigned to each of the MAMU and MOM nodes, effectively dividing each into three pools based on the value of the resource string.

B. Job Submission Changes

Whenever a new job is submitted to the system, it is necessary to associate it with a particular trust zone. This ensures that it will be passed to the correct set of MOM or MAMU nodes. For reasons of security, it is important that it is not possible for the user to set the trust zone. This can be achieved using a PBS submit hook.

When a job is passed to the submit hook, it examines the value of the `requestor_host` attribute as defined by the PBS server. If this value matches the name of the one of the external login nodes, the hook consults a configuration file which maps the name of each external node to a trust zone. This ensures that jobs submitted from outside the system belong to the correct zone.

Where the value of `requestor_host` matches an internal node such as a MOM or MAMU node, PBS checks the value of the trust zone resource assigned to the host and reapplies it to the job. This ensures that jobs submitted from within the system retain their parent trust zone whilst also allowing internal nodes to be transferred from one zone to another by the simple expedient of draining the node and changing its zone attribute.

Under normal circumstances, where a job has requested Cray compute nodes, PBS allows it to run on any MOM node and does not impose any resource constraints. This does not match the trust zone model, where jobs from different trust levels must be assigned to different sets of MOM nodes.

Consequently, the trust zone submit hook also allocates an additional chunk of resources to each job which asks for Cray compute nodes. This chunk requests a single CPU, a small amount of memory, and, more importantly, specifies trust zone resource. This last ensures that the job can only be assigned to a MOM node at the right trust level.

C. Non-Compute Node Jobs

Jobs which do not require Cray compute nodes will be scheduled for execution on the MAMU nodes with a `vntype` of `cray_serial` and where the trust zone resource matches the zone requested by the job. Because the MAMU nodes are statically partitioned and the file system bind mounts had already been configured, the job can be scheduled without any further intervention from PBS.

D. Compute Node Jobs

Jobs which require compute nodes must wait for sufficient resources to become available. Because the compute nodes are not labelled with a trust zone, any compute node can be used to satisfy the requirement. Once sufficient free nodes are available, the job is passed to a MOM node in the appropriate trust zone for execution.

Prior to the start of each compute node job, a PBS hook is run on the allocated MOM node to prepare the set of assigned compute nodes for the job. The hook obtains the list of compute nodes from the server and uses the `pcmd` command available through ALPS to bind the Lustre file systems to the appropriate mount points for the job and to apply network configuration settings to reset the gateway node used to direct traffic out of the system.

Once the job is complete and all the user processes have exited, but before the job has left the MOM node, the same hook runs and reverses the various changes applied by the set-up hook. This makes multiple attempts to unbind the Lustre file system to ensure that it is not possible for a subsequent job to view files at the wrong trust level.

While experiences with the compute node configuration hook on our development system were successful and initial tests on the larger XCS appeared to be successful, serious performance problems were encountered as soon as we attempted to run benchmarking jobs larger than a few hundred nodes.

When we profiled the hook, we found that almost all the time was being spent in mount operations, with each bind mount carried out in its own `pcmd`. When this was changed to merge the mount operations into a single `pcmd` performance improved substantially, with the time taken to launch a machine-sized job dropping from over five minutes to approximately 30 seconds.

E. MOM Node Resources and Placement Sets

Following the implementation of the trust zone mode, a problem was encountered which caused almost every cycle of the PBS scheduler to abort with a calendaring error. This caused the system to drain and the only jobs run during this period were those that spanned multiple cabinets. Investigations in combination with Cray and Altair revealed that this problem was caused by an error in the placement set configuration on the system.

As originally configured, each two-cabinet group of electrically connected nodes had been assigned a `pair` string resource in PBS. This had been added to the `node_group_key` setting in the server, causing PBS to attempt to allocate jobs to nodes with the same `pair` value to encourage tasks to be placed on physically adjacent nodes — a behaviour that noticeably improved performance.

With the introduction of the trust zone configuration, each job was also assigned a chunk of resource on a MOM node. But, as originally configured, the MOM nodes were not defined with a `pair` resource. Consequently the PBS server could not satisfy the placement constraint for all jobs which were small enough to fit in a two-cabinet group, causing the scheduler backfill calendar to fail. Jobs which were too large to fit in any of the `pair` placement sets were implicitly assigned to the universal set which explains why they were able to be scheduled.

Once the root cause of the problem was identified, it was resolved by removing and recreating the `pair` resource as a `string_array`. This redefinition of the resource made it possible to make each MOM node a member of every placement set. This ensured that whatever the placement set selected for the compute nodes, the server was always able to locate a MOM node with a matching placement string, preventing the scheduler from failing to calendar the request because of an unsatisfied resource requirement.

F. PBS Conclusions

We have now been running with our PBS trust zone configuration for some months and our experiences have been extremely positive. Having resolved the problems with placement set configuration and having improved the scalability of the compute node set-up hook, the system has performed reliably and our end users have reported very little inconvenience as a result of the new operating model.

VI. CGroups and Resource Separation

The separation of user work into separate trust zones is sufficient to prevent processes at different trust levels from interacting with each other. This provides some degree of protection from inadvertent denials of service caused by jobs at the same level. However, it does not provide any separation between processes owned by jobs at the same level of trust running on the same MOM node or MAMU node.

By default PBS does not apply any additional constraints on jobs once they have been allocated to a node. This means that it is possible for a job to request a single CPU from PBS and then to make use of all the CPU resources on the node. As a result it is possible for users at the same level of trust and running on the same node to compete for access with other tasks on the same node. Where these tasks are time critical, as is the case for operational forecast work, it is possible for these delays to result in a denial of service.

In order to isolate tasks from different jobs from each other when running on the same node, it is necessary to configure PBS to run each job in its own cgroup. The cgroup acts as a container, limiting the processes running within it to a subset of the available processors and memory based on the resources requested in the PBS job submission. Where a job attempts to use more than the requested number of CPUs, its processes are forced to time-slice; and where a process attempts to exceed the memory limit of the group, it is forceably killed.

Cgroups have been integrated with PBS via an internal hook script which runs on the MOM node or MAMU node prior to the start of each job to create the container. The same hook runs again at the end of each job and removes the container if all user processes have exited.

A number of attempts have been made to enable cgroup support on the Cray, starting prior to the implementation of the trust model with PBS 12.2 and progressing as far as PBS 13.0.406. Each of these releases has been problematic and so far none has proved sufficiently stable for full production use.

A. Inability to Release Cgroups

Initial experiences with an experimental version of the cgroup hook provided with PBS 12.2 were largely unsuccessful. The hook was able to create the cgroup and the job scheduled to run in it worked as expected, with resources being constrained to its requested resources.

Problems were observed with the cgroup clean-up, where stale job processes occasionally prevented the hook from removing the group. This caused an accumulation of stale groups on executing nodes, eventually causing jobs to fail to start correctly. This behaviour was invisible to the PBS scheduler and resulted in many jobs being assigned to the same problematic node, causing a large amount of work to be placed on hold.

This problem was resolved by Altair in a subsequent release of the hook which moved orphaned job containers which could not be removed, allowing them to be cleaned up by a periodic hook once all the job processes had been exited.

B. CGroup VNode Time-outs

Following an upgrade to PBS 13.0.401, an attempt was made to enable cgroups on a subset of the system by changing the hook configuration file to ignore all nodes except those with a custom `vntype`. This configuration worked successfully on our small test platform but produced very inconsistent results when run on our larger production systems.

An analysis of the problem revealed that the code used to determine the `vntype` of the current node posted a query to the PBS server every time a cgroup was created. Due to the load on the server the node type code was timing out before

receiving a response, causing it to return a `None` value to the hook which, because this failed to match the exempt list, resulted in the hook being enabled on the wrong set of nodes.

This consistency problem was partly avoided on the XCS system by creating an empty exempt list and enabling the hook on all node types. While this solution was not suitable for production use, it did allow the cgroup hook to be tested during system acceptance.

The underlying fault was resolved in PBS 13.0.406 when the hook was updated to cache the `vntype` of the current node in the persistent `/var` file system, removing a great deal of unnecessary load on the server daemon.

C. CGroup Time-out Condition

Once the cgroup hook was enabled on XCS, a further problem was observed with containers not being removed correctly at the end of every job. As before, this resulted in an accumulation of stale containers on the nodes, eventually resulting in a large number of job failures as the PBS scheduler repeatedly tried to schedule jobs to nodes without any available resources.

An examination of the PBS daemon logs on the executing nodes revealed a correlation between stale cgroups and hook alarm time-out events. In some cases these alarms had occurred in the cgroup hook itself, but in most cases the time-outs were found to have occurred in one of the local site epilogue hooks.

An inspection of the implementation of PBS hooks in the PBS Pro open source code base provided a useful insight into the cause of the problem. This showed that each script associated with a given chain of hooks was run in sequence, so that the failure of any particular script would cause any remaining scripts in the sequence to be abandoned. Thus an alarm event in one of the site epilogues, by default ordered ahead of Altair's hook, was sufficient to prevent the cgroup from running.

This problem was largely avoided by increasing the time-outs of the hook scripts and changing the order of Altair's cgroup hook to -1[3] ensuring the hook ran before any of the site hooks.

D. Cgroups and File Cache Memory

When a cgroup memory limit is enabled any process which attempts to exceed the maximum memory available to the container is killed with a signal. When determining the total amount of memory used by processes in the container, cgroups includes that used for file caching[3].

Although file cache memory is dropped to free up resources for large computational tasks, limiting it in this way can cause IO-intensive operations to fail unexpectedly. These failures result in a distinctive trace-back in the system logs, indicating that a process has failed while attempting to acquire additional page cache.

For example, when a file copy is run in a cgroup with a memory limit that is too low to buffer the file and the IO stream has not been opened in direct mode, the process will be terminated by the out-of-memory killer when it reaches the limit. For single files this problem can be avoided by enabling direct IO but this greatly reduces the performance and does not help with cases where the failure is due to a recursive copy of a directory tree.

This problem was first noticed when cgroups were enabled on MOM nodes on XCS where default memory was capped at a gigabyte to encourage users to run post-processing work on the shared nodes. These limits resulted in intermittent failures of sequences of model runs, where large files needed to be copied at the end of each run ready for the start of the next model in sequence.

This problem was resolved by the simple expedient of raising the memory limits on the MOM nodes to 1.5 gigabytes. This was sufficient to prevent jobs from failing whilst also being small enough to allow all 72 CPUs on the node to be used.

E. CGroup and PBS Memory Settings

The cgroup hook, as provided by Altair in PBS 13.0, was initially configured to prevent the hook script from attempting to allocate more than 90 per cent of physically configured memory on the node. But when a node is provisioned by PBS, its maximum available memory is set to 100 per cent of the physically configured memory.

Consequently it is possible for a job to request all the available memory on a node and for the PBS scheduler to assign the job to a MAMU node which has advertised that it has sufficient free memory to run the job. When the job starts on the node, PBS is unable to create the cgroup because it is constrained to 90 per cent of memory. It treats this failure as a run-time error and places the job in a held state.

This problem can be resolved by reducing the amount of available memory advertised by each MAMU node to match the amount of memory available using cgroup containers. This ensures that work will only ever be scheduled on a node if it can fit within the resources available at run-time.

This behaviour has changed slightly at PBS 13.0.406 where it is now possible to specify an absolute memory limit in gigabytes rather than having to specify a percentage of the available memory. This simplifies the task of reserving memory for operating system tasks on the MAMU nodes.

F. CGroups and Hyperthread Settings

Following the upgrade to PBS 13.0.406, it was noticed that the performance of certain serial jobs on the MAMU nodes had degraded significantly. Investigations showed that these problems were limited to CPU-intensive jobs requesting access to all the physical CPUs on a node.

[3]Internal PBS hooks ordering is allowed to go negative. User Hooks are limited to values from 0 to 1023

When the allocation of resources to these jobs was investigated, it was found that the cgroup hook had incorrectly assigned all the tasks to a single NUMA socket on the node rather than spreading across the two available sockets. Consequently tasks which had previously been placed on separate physical cores were now being placed on different hyperthreads on the same core.

An examination of the cgroup hook suggested that the problem had occurred because the CPUs were being detected by examining each NUMA node in turn. In the previous version of the hook, the CPUs had been sorted by ID to ensure physical cores were placed ahead of hyperthreads. In the revised hook, which added partial support for hyperthreading, the CPUs were returned in order of discovery, causing tasks to be assigned a mix of cores and hyperthreads.

This problem was worked around by modifying the cgroup hook to explicitly ignore hyperthreads.

G. CGroup Run-time Error Handling

Prior to PBS 13.0.406, run-time errors when creating a cgroup at the start of a job were handled by placing the affected job in a held state. Because the node remained in service, this typically resulted in another being assigned to the same node and also being placed on hold. This resulted in occasions where a fault on a single node caused many hundreds of jobs to fail and be placed in a held state.

With the advent of PBS 13.0.406, Altair changed the way run-time errors were handled to prevent large numbers of jobs from failing by taking the affected node off-line when it was unable to create or remove a cgroup container. This behaviour was coupled with a change to the periodic clean-up portion of the cgroup hook which places the node back on-line once any transient problems — especially those caused by failed cgroup removal — have been resolved.

This change works well on the MAMU nodes but works less well when a cgroup error occurs on a MOM node.

The implementation of PBS on the Cray is such that the compute nodes, which make up the majority of the system, are defined as virtual nodes owned by the MOM nodes. Consequently, when a MOM node is taken off-line, a large number of the compute vnodes are also placed off-line. As a result, it is possible for a cgroup handling fault on a MOM node to cause the compute resource to become unavailable.

This problem has been raised with Altair but no immediate work around has yet been found.

H. Cgroup Conclusions

It is apparent that, although cgroups offer a powerful tool for managing the resources available to jobs on specific nodes, the facility is still immature and problematic. Where a problem occurs at run-time, the actions taken by the cgroup hook — such as placing a job on hold or, in the latest version of the hook, taking a node off-line — have the potential to cause large numbers of jobs to become stuck or to cause large parts of the system to be taken out of service.

As a result of the problems seen on the XCS system, the Met Office has chosen not to implement cgroups on its operational systems. However, we continue to be interested in using cgroups and have been working closely with Altair to provide feedback on the hook, trialling new versions of the hook prior to formal upgrades of PBS.

VII. Summary

As can be seen from this paper, the PBS trust labelling scheme makes it possible to ensure that user jobs at different levels are always assigned to different serial or MOM nodes. The use of dynamic compute node partitioning ensures that resources can be created or removed as needed to fulfil individual job requirements.

It is further possible to isolate the processes and resources assigned to jobs at the same trust level using cgroup integration with PBS. This effectively dedicates resources to each job and prevents one user from denying resources to another user on the same non-compute node.

The use of cgroups presents a number of additional challenges and the facility is not yet sufficiently mature to be used on systems where completely reliable behaviour is required. Depending on the nature of the workload and, especially, the amount of IO performed by compute jobs on MOM nodes, users jobs may fail with unexpected memory errors.

References

[1] Met Office and NERC joint supercomputer system (MONSooN). [Online]. Available: http://www.metoffice.gov.uk/research/collaboration/jwcrp/monsoon-hpc

[2] Unified Model Partnership. [Online]. Available: http://www.metoffice.gov.uk/research/collaboration/um-partnership

[3] Memory cgroup Documentation (Kernel Archives). [Online]. Available: https://www.kernel.org/doc/Documentation/cgroup-v1/memory.txt

[4] *PBS Professional 13.0 Administrator's Guide*, 2015.

[5] *PBS Professional 13.0 Reference Guide*, 2015.