Systems-level Configuration and Customisation of Hybrid Cray XC30

Roberto Aielli, Sadaf Alam, Vincenzo Annaloro, Nicola Bianchi, Massimo Benini,
Colin McMurtrie, Timothy Robinson, and Fabio Verzolli
CSCS – Swiss National Supercomputing Centre
Lugano, Switzerland
Email: {aielli,alam,annaloro,nbianchi,benini,cmurtrie,robinson,fverzell}@cscs.ch

***Abstract*—In November 2013 the Swiss National Supercomputing Centre (CSCS) upgraded the 12 cabinet Cray XC30 system, Piz Daint, to 28 cabinets. Dual-socket Intel Xeon nodes were replaced with hybrid nodes comprising one Intel Xeon E5-2670 CPU and one Nvidia K20X GPU. The new design resulted in several extensions to the system operating and management environment, in addition to user driven customisation. These include the integration of elements from the Tesla Deployment Kit (TDK) for Node Health Check (NHC) tests and the Nvidia Management Library (NVML). Cray extended the Resource Usage Reporting (RUR) tool to incorporate GPU usage statistics. Likewise, the Power Monitoring Database (PMDB) incorporated GPU power and energy usage data. Furthermore, custom configurations are introduced to the Slurm job scheduling system to support different GPU operating modes. In collaboration with Cray, we assessed the Cluster Compatibility Mode (CCM) with Slurm, which in turn allows for additional GPU usage scenarios, which are currently under investigation. Piz Daint is currently the only hybrid XC30 system in production. To support robust operations we invested in the development of: 1) an holistic regression suite that tests the sanity of various aspects of the system, ranging from the development environment to the system hardware; 2) a methodology for screening the live system for complex transient issues, which are likely to develop at scale.**

***Keywords*-Hybrid Cray XC30; Nvidia GPU; System management; PMDB; RUR; Regression; Tesla Deployment Kit (TDK); Health monitoring**

I. Introduction

Piz Daint, the first Petascale hybrid Cray XC30 system, has been deployed at the Swiss National Supercomputing Centre (CSCS) during the last quarter of 2013. The system has a number of unique features, including hybrid node daughter cards (Intel Xeon CPU and Nvidia Tesla GPU), a fully provisioned dragonfly interconnect for 28 cabinets, an adaptive programming and execution environment and an energy-aware system monitoring and diagnostics infrastructure. This report provides an overview of the system administration tools and configurations that have been extended for the hybrid Cray XC30 platform. Moreover, we present solutions that are developed at CSCS to support robust operations of a unique system at scale.

Piz Daint was installed in two phases. During the first installation phase (Phase I), multi-core only nodes were installed in 12 cabinets [1]. The goal of the Phase I installation was to capture the requirements of user applications with respect to earlier Cray platforms. Evaluation and comparison of the Aries interconnect in the Cray XC30 and the Gemini interconnect in the Cray XE6 and Cray XK7 confirmed that applications can exploit the high bandwidth interconnect. During Phase I, system management and monitoring interfaces were evaluated on the Cray XK7 system. Each node of a Cray XK7 system contains an AMD Interlagos processor and Nvidia Tesla K20X accelerator. During the final installation phase (Phase II), hybrid multi-core nodes with the Nvidia Tesla K20X devices replaced multi-core nodes. Furthermore, additional optical cables are added in order to implement a fully provisioned dragonfly network on 28 cabinets (i.e. 14 electrical groups). The resulting system is the fastest supercomputing platform in Europe (according to the Top500 list released in November 2013) and the most energy efficient PetaScale system (Green500 list, November 2013).

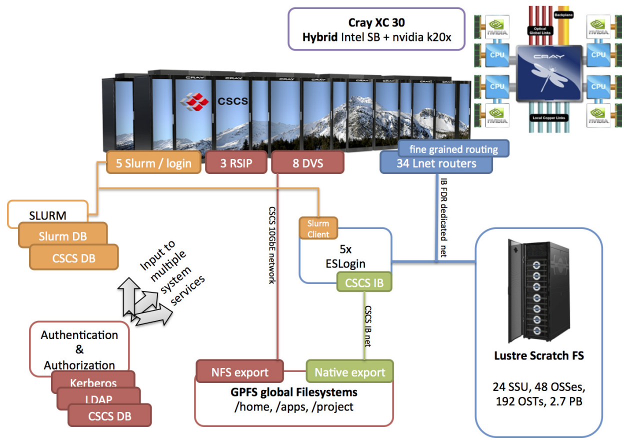


Figure 1. Piz Daint and its supporting ecosystem including the scratch and external file systems, resource management and accounting setup.

Figure 1 gives a high level overview of Piz Daint and its supporting environment at CSCS, which includes an internal parallel scratch file system (Sonexion1600), external login nodes, external and site-wide file systems, a resource manager (Slurm) and CSCS database accounting as well as authentication subsystems.

In order to support the additional compute capability of the Phase II system, several components of the supporting ecosystem (e.g., the scratch file system, the number of service node and LNET routers) were augmented; details will be provided in the subsequent sections. At the same time the new node design required extension and customization of the system management and operating environment: this involved the integration of elements from the Tesla Deployment Kit (TDK), for Node Health Check (NHC) tests, and the Nvidia Management Library (NVML) [2]. Cray extended the Resource Utilization Reporting (RUR) tool to incorporate GPU usage statistics. Likewise, the Power Monitoring Database (PMDB) incorporated GPU power and energy usage data [3]. Furthermore, custom configurations were introduced to the Slurm job scheduling system to support different GPU operating modes [4]. In collaboration with Cray we assessed Cluster Compatibility Mode (CCM) [5] in conjunction with Slurm, which in turn allows for additional GPU usage scenarios that are currently under investigation. At the time of writing, Piz Daint is the only hybrid XC30 system in production. To support robust operations we invested in the development of:

1) A holistic regression suite that tests the sanity of various aspects of the system, ranging from the development environment to the system hardware;
2) A methodology for screening the live system for complex transient issues, which are likely to develop at scale.

The organization of the paper is as follows: Section II provides system configuration details and compares and contrasts Phase I and Phase II configurations. Section III lists the key extensions to the system management tools and interfaces: Tesla Deployment Kit (TDK), analysis of logs, Resource Usage Reporting (RUR), Power Management Database (PMDB) and counters, GPU operations modes, and Cluster Compatibility Mode (CCM) extension to Slurm. Details on the CSCS regression suite and motivation are provided in Section IV. A case study will be presented in Section V that highlights how complex problems – for example, transient at-scale issues at the driver level – can be investigated and verified. A summary of features and plans for extensions is provided in Section VI.

II. System Configuration

The main system is composed of 5,272 compute nodes, each with a single Intel Xeon Sandy Bridge 8-core processor, an Nvidia Tesla K20X GPU and 32 GBytes of memory. The hybrid blade layout is shown in Figure 2. There are 26 service blades (which equates to 52 service nodes) and these are distributed to optimally distribute parallel file system traffic. The scratch file system is hosted by a Sonexion1600 storage appliance comprised of 24 Scalable Storage Units (SSUs) and one Metadata Management Unit (MMU). The file system is built around Lustre Server v2.1+ and is connected to the XC30 service nodes via a dedicated FDR InfiniBand fabric to 34 Service Nodes within the XC30. There are 5 external Login nodes (esLogin) and we use the Slurm scheduling system. For pre-installation and testing, a 16-compute node, self-contained TDS system was available with its own Sonexion1600 storage and external login node.

Tables I to IV provide an overview of system configuration details at each of the two distinct phases of the system. Table I summarises the compute node characteristics and as can be seen the Phase II system delivers over 10x double-precision floating-point performance. Furthermore, the per node compute capability and memory bandwidth is increased by a factor of 4.

Table II compares the network characteristics. The Phase II system has a fully provisioned optical network and as

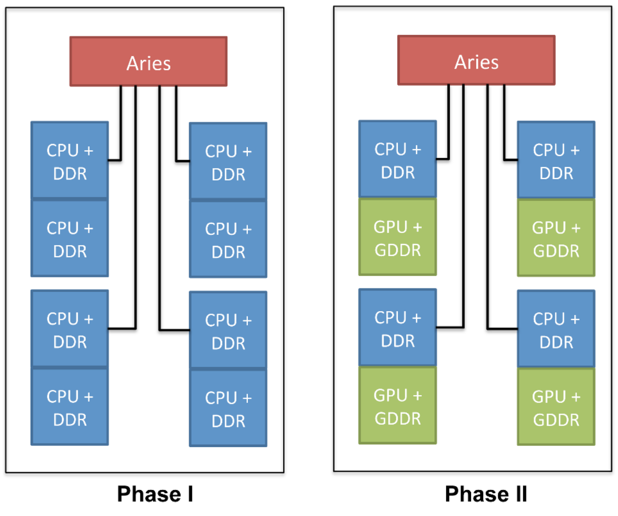


Figure 2. Piz Daint compute blade design in the two phases of installation. Two types of memory are available in Phase II: CPU DDR3-1600 (32 GBytes per node) and GPU GDDR5 memory.

	Phase I	Phase II
Number of compute nodes	2,256	5,272
Peak system DP performance	0.75 Pflops	7.8 PFlops
CPU cores/sockets per node (Intel Xeon E5-2670)	16/2	8/1
DDR3-1600 memory per node	32 GBytes	32 GBytes
GPU SMX/devices per node (Nvidia K20x)	--	14/1
GDDR5 memory per node	--	6 GB (ECC off)
DP Gflops per node	332.8 Gflops (CPU)	166.4 Gflops (CPU) 1311 Gflops (GPU)
DDR3-1600 bandwidth per node	102.4 GB/s	51.2 GB/s
GDDR5 bandwidth per node	--	250 GB/s (ECC off)
Network & I/O interface	16x PCIe 3.0	16x PCIe 3.0
Network injection bandwidth per node	~ 10 GB/s	~ 10 GB/s

Table I
Comparison of node characteristics during the two phases of installation.

	Phase I	Phase II
Number of cabinets	12	28
Number of groups	6	14
Number of Aries network & router chips	576	1344
Number of optical ports (max)	1440	3360
Number of optical ports (connected)	360	3276
Number of optical cables	180	1638
Bandwidth of optical cables	6750 GB/s	61425 GB/s
Bisection bandwidth	4050 GB/s	33075 GB/s
Point to point bandwidth	8.5-10 GB/s	8.5-10 GB/s
Global bandwidth per compute node	3 GB/s	11.6 GB/s

Table II
COMPARISON OF HIGH-SPEED DRAGONFLY NETWORK CHARACTERISTICS DURING THE TWO PHASES OF INSTALLATION.

	Phase I	Phase II
Storage capacity (user scratch)	1.1 PBytes	2.6 PBytes
Storage units (Sonexion SSU)	10	24
Storage servers (OSS)	20	48
Storage targets (OST)	80	192
File system bandwidth (max)	60 GB/s	144 GB/s
Metadata unit (MDU)	1	1
LNET routing nodes	12	34
DVS servers (external file systems)	4	8
Total service nodes	24	52
SLURM/login servers	4	5
RSIP servers (CCM mode)	--	3

Table III
COMPARISON OF SYSTEM ADMIN AND STORAGE CHARACTERISTICS DURING THE TWO PHASES OF INSTALLATION.

a result, the per-node global bandwidth and the bisection bandwidth of the system are increased by factors of 4 and 8, respectively.

System administration components of the system are listed in Table III together with the storage configuration. With the additional storage capacity there is a relative increase in the aggregate storage bandwidth. Similarly, additional service nodes are included to support additional servers for specific operations, for example, DVS, Slurm and login. RSIP services are added for Slurm CCM, which will be explained in the next section.

Table IV lists some of the features of the programming and execution environment. These are included here because they impact to some extent the configuration of certain system management and monitoring tools, especially the Slurm job scheduler. For instance, the support of GPU operating modes required customization of Slurm. Moreover, RUR and CCM are only introduced in Phase II.

	Phase I	Phase II
Cray Linux Environment	5.0	5.1UP02
Programming environments	CCE, GNU, Intel, PGI	CCE, GNU, Intel, PGI
CUDA SDK	--	5.5
CUDA tools	--	nvcc, cuda-gdb, cuda-memcheck, nvprof, nvvp
MPI 3.0	Supported	Supported
GPU-2-GPU MPI	--	Supported
OpenACC compilers	--	Cray & PGI
Debugger (DDT)	MPI & OpenMP	MPI, OpenMP, CUDA & OpenACC
Vampir and scorep performance tools	MPI & OpenMP	MPI, OpenMP, CUDA & OpenACC
Resource Usage Reporting	--	Available
Cluster compatibility mode	--	Available

Table IV
COMPARISON OF PROGRAMMING AND EXECUTION ENVIRONMENTS CHARACTERISTICS DURING THE TWO PHASES OF INSTALLATION.

III. EXTENSIONS TO THE SYSTEM TOOLS AND INTERFACES

As mentioned in the previous section, system management and monitoring interfaces, and some diagnostics tools, have been extended to support a robust operation of the hybrid Cray XC30 platform. Moreover, in order to improve the productivity of the end users of the hybrid system, custom configurations have been introduced to the Slurm scheduling environment to allow for different GPU operating modes and to fully support the Nvidia CUDA SDK. Note that Cray's ALPS scheduling interface is quite robust for mapping MPI processes and OpenMP threads on multi-core platforms. However ALPS currently does not contain similar extensions for the hybrid platform.

A side-by-side comparison of system tools and interfaces are shown in Figure 3. Some new features, for example, RUR and PMDB, are not specific to the GPU. During the first phase of installation, these features were unavailable on the Cray XC30 platform.

A. Integration of the Tesla Deployment Kit (TDK)

The Tesla Deployment Kit (TDK), which has recently been renamed the GPU Deployment Kit, is composed of a set of tools and APIs that enable users and/or system administrators to control and configure GPU devices. There are two main components of TDK: `nvidia-healthmon` and Nvidia Management Library (NVML) API.

`nvidia-healthmon` is a system administrator tool for detecting and troubleshooting common problems affecting Nvidia Tesla GPUs in a high performance computing (HPC) environment, i.e., a GPGPU-accelerated cluster. Hence, the `nvidia-healthmon` contains limited hardware diagnostic capabilities and focuses on software and system configuration issues and is designed to:

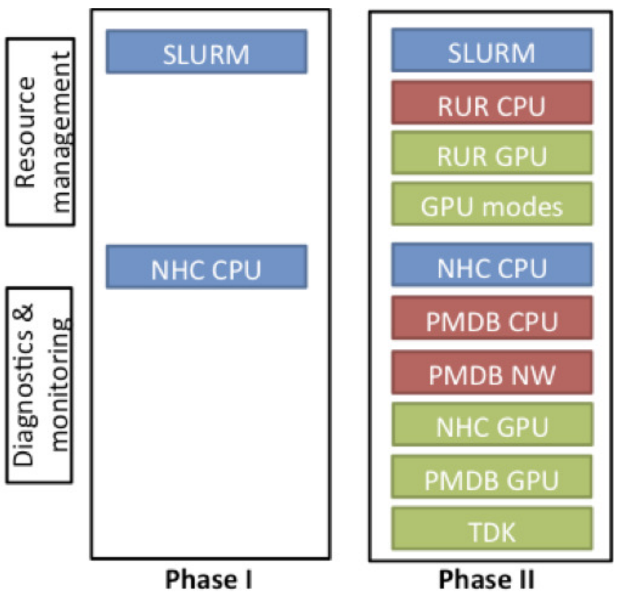


Figure 3. System monitoring and diagnostics stacks for two installation phases. Red items show that the features are available for multi-core or CPU components. Green items are specific to the GPU devices.

- Discover common problems that affect a GPU's ability to run a compute job including:
  - Software configuration issues;
  - System configuration issues;
  - System assembly issues, like loose cables;
  - A limited number of hardware issues.
- Provide troubleshooting help;
- Easily integrate into Cluster Scheduler and Cluster Management applications;
- Reduce downtime and failed GPU jobs.

Currently `nvidia-healthmon` cannot detect all known GPU issues that can be uncovered by the Nvidia hardware field diagnostics (`fielddiag`) tests. Moreover, it is a completely passive tool in the sense that it cannot offer a resolution to a known problem or fix it.

Nonetheless, the `nvidia-healthmon` tool was incorporated as part of the Cray Node Health Check (NHC) prologue. For this integration into the NHC configuration we wrote a simple shell script that calls the `nvidia-healthmon` command with its configuration file. The NHC script was written following Cray's guidelines ([6]), and uses the following files:

- Configuration file: `/etc/opt/cray/nodehealth/nodehealth.conf`
- Custom script: `/apps/daint/system/nodehealth/NHC_nvidia-healthmon.sh`

A sample output from the `nvidia-healthmon` test is shown in Figure 4.

```
Loading Config: SUCCESS
Global Tests
Black-Listed Drivers: SUCCESS
Load NVML: SUCCESS
NVML Sanity: SUCCESS
...
NVML Sanity: SUCCESS
InfoROM: SUCCESS
Multi-GPU InfoROM: SKIPPED
ECC: SUCCESS
PCIe Maximum Link Generation: SUCCESS
PCIe Maximum Link Width: SUCCESS
CUDA Sanity: SUCCESS
PCI Bandwidth: SUCCESS
Memory: SUCCESS
...
```

Figure 4. Sample output from `nvidia-healthmon`.

The Nvidia Management Library (NVML) is a C-based programming interface for monitoring and managing various states within Nvidia GPU devices. It is intended to be a platform for building third-party applications and is also the underlying library for the Nvidia-supported `nvidia-smi` tool.

The NVML API is divided into five categories:

- Support Methods:
  - Initialization and Cleanup
- Query Methods:
  - System Queries
  - Device Queries
  - Unit Queries
- Control Methods:
  - Device Commands
  - Unit Commands
- Event Handling Methods:
  - Event Handling
- Error Reporting Methods:
  - Error Reporting

The `nvidia-smi` has a set of privileged and non-privileged commands, but the majority of query methods are available to the users. However, some control methods, particularly those that can change the unit or device configuration or operating modes, can only be used in the privileged mode. Conveniently a couple of privileged-mode options can be enabled via the Slurm epilogue and prologue extensions. This will be explained in the next section.

NVML query methods can be used through the API or using a python interface called pyNVML [7]. On the hybrid Cray XC30 platform, GPU accounting data, namely the GPU usage and GPU memory usage, is reported in Cray's Resource Usage and Reporting (RUR) tool, and this too is explained in the subsequent section. Users can also query additional information about the GPU devices such as ECC setting, clock speed, and so on.

B. *Enabling Various GPU Operating modes*

Users often like to request a number of privileged commands. For example, OpenCL applications may want to request the default operating mode to allow for multiple MPI tasks per GPU. This is available to CUDA and OpenACC applications in a setting called Multi Process Mode (MPS). Another instance is clock frequency boost. One application, namely GROMACS, showed a speedup of 10-15% by increasing the clock frequency of the GPU, which is a privileged-mode option.

We have enabled a small subset of these options via the Slurm constraint mechanism. The steps involved as described as follows:

1) `/opt/slurm/default/etc/slurm.conf`
   a) Include a definition of the slurm control daemon (`slurmctld`) prologue and epilogue scripts thus: `PrologSlurmctld=/opt/slurm/default/etc/prologslurmctld.sh` `EpilogSlurmctld=/opt/slurm/default/etc/epilogslurmctld.sh`
   b) Include a definition of the "Feature" supported by Slurm: `Feature="UNKNOWN,gpumodedefault,aclock,startx"` `Gres=gpu_mem:6144,gpu:1`
2) The `prologSlurmctld` script interprets the requested "Features" and, usually via `nvidia-smi` command, sets the desired mode of the GPU.

A user can request a predefined feature in a job script. Resources are allocated to the user according to the constraints and returned to the system in default mode. For example, by default, all GPU devices are set in the "Exclusive Operating Mode". However when the user of an OpenCL application specifies `-C gpumodedefault` in their job script, the GPU devices assigned to the users job are configured in the privileged "GPU Mode Default" mode. After job execution the device is reconfigured to the default "Exclusive Operating Mode."

A similar mechanism has been adopted for boosting the clock frequency of the GPU. The following options are used within the `prologSlurmctld` script:

- Show Supported Clock Frequencies:
  – `nvidia-smi -q -d SUPPORTED_CLOCKS`
- Set Memory and Graphics Clock Frequency:
  – `nvidia-smi -ac <MEM clock, Graphics clock>`
- Show current mode:
  – `nvidia-smi -q -d CLOCK`
- Reset all clocks:
  – `nvidia-smi -rac`
- Allow non-root to change clocks:
  – `nvidia-smi -acp 0`

C. *Resource Usage and Reporting (RUR)*

Cray's Resource Utilization Reporting (RUR) is a tool for gathering statistics on how system resources are being used by applications. RUR is enabled on Piz Daint. With the default setting, outputs are recorded in `~/rur.jobid`.

Users can customize the output in a number of ways. Firstly, the RUR output can be redirected to a user-defined location as specified the redirect file `~/.rur/user_output_redirect`.

The contents of this file must be a single line that specifies the absolute path (or relative path within the user's `$HOME`) to the directory where the RUR output is to be written. If the redirect file does not exist, or if it points to a path that does not exist or to which the user does not have write permission, then the output is written to `$HOME`. Users who do not wish to collect RUR output data can simply set the redirect path to `/dev/null`.

Additionally, the user may override the default report type by specifying a valid report type in `~/.rur/user_output/report_type`. Valid report types are apid, jobid, or single, resulting in the user's RUR data being written to one file per application, one file per job, or a single file, respectively. If the report type file is empty or contains an invalid type, the default report type, as defined in the configuration file, is created.

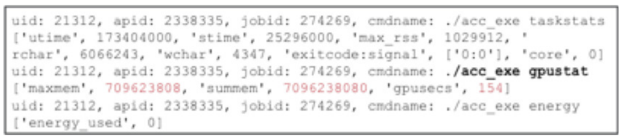
The default output of RUR after a job completes shows the `taskstats`, `gpustat` and `energy` information. For each one of these entries, the following data is listed:

- user ID (`uid`)
- ALPS ID of the job (`apid`)
- Slurm job ID (`jobid`)
- name of the executable (`cmdname`)

The `taskstats` record provides basic process accounting similar to that provided by Unix process accounting or `getrusage`. This includes the system and user CPU time, maximum memory used, and the amount of file input and output from the application. These values are sums across all nodes, except for the memory used, which is the maximum value across all nodes.

The `gpustat` record provides utilization statistics for Nvidia GPUs on Cray systems. It reports both the GPU compute time and the memory used summed across all nodes as well as the maximum GPU memory used by the application across all nodes.

The `energy` record prints the total energy used, measured in Joules, by all nodes participating in the running of



```
uid: 21312, apid: 2338335, jobid: 274269, cmdname: ./acc_exe taskstats
['utime', 173404000, 'stime', 25296000, 'max_rss', 1029912, '
rchar', 6066243, 'wchar', 4347, 'exitcode:signal', ['0:0'], 'core', 0]
uid: 21312, apid: 2338335, jobid: 274269, cmdname: ./acc_exe gpustat
['maxmem', 709623808, 'summem', 7096238080, 'gpusecs', 154]
uid: 21312, apid: 2338335, jobid: 274269, cmdname: ./acc_exe energy
['energy_used', 0]
```

Figure 5. RUR sample output for a GPU enabled application on Piz Daint.

the job. Note that the figure does not include the Aries ASIC or cabinet blowers.

Figure 5 illustrates a sample output of a GPU accelerated application called `acc_exe`. Two GPU statistics are highlighted: the sum of memory across all GPU devices that are used for the execution of the code and the maximum memory per GPU device. These data were in turn gathered by NVML GPU accounting interface.

D. Power Management Database (PMDB) and `pm_counters`

System power usage is available in the PMDB where data can be queried at multiple levels. There are cabinet level sensors, blade level sensors and node level sensors. Node level information is also available in `/sys/cray/pm_counters`:

- `accel_energy`
- `accel_power_cap`
- `power`
- `accel_power`
- `energy`
- `power_cap`

Details of CSCS's PMDB configuration are presented in a CUG 2014 publication [8]. Note that multi-core only Cray XC30 platforms do not have any "accel" counters.

E. CCM Extensions to Slurm

On Piz Daint, Slurm has been extended to support Cluster Compatibility Mode (CCM). Conventionally, CCM is enabled at Cray sites to run ISV applications, but our motivation is different – namely, to facilitate the use of the graphical profiling tool Nvidia Visual Profiler (`nvvp`), which can not run natively through the ALPS `aprun` command. With CCM, a user is able to launch the application by logging into the compute node with X11 forwarding.

CCM is tightly coupled to the workload management system. We are using a patch for the Slurm version 2.5.4, which is written by Bright Computing in collaboration with Cray. Enabling CCM was done following Cray's procedures (documented in [9]) with the optional step regarding the Realm Specific Internet Protocol (RSIP) configuration. This step is necessary to allow the compute nodes to reach the Kerberos and LDAP servers which sit on the CSCS datacentre network (external to the system) for user authentication and authorisation.

Enabling CCM involves the following files:

```
> salloc -N 1 -p ccm
> module load ccm
> export PBS_JOBID=$SLURM_JOBID
> ccmlogin -V
> hostname
nid02542
> module load craype-accel-nvidia35
> nvvp &
```

Figure 6. Launching `nvvp` on a compute node of Piz Daint.

1) `/etc/ldap.conf` in the shared root (default)
2) `/etc/nsswitch.conf` in the shared root (default)

Three CCM RSIP servers are configured to support this mode on the Piz Daint system. There are some outstanding bugs that need to be resolved, especially for multiple MPI tasks.

Step-by-step instructions for launching `nvvp` are shown in Figure 6.

With the CCM solution, the entire Nvidia CUDA SDK is available to the users of Piz Daint.

IV. Holistic Regression Suite

The Cray Node Health Check (NHC) and `nvidia-healthmon` tools are useful for the early detection of a small subset of already known hardware issues. These tools, however, provide information only at the level of the node, and thus cannot provide a good assessment of the sanity of the system as a whole. For this reason, CSCS has developed a regression suite for Piz Daint that provides an overview of the system health over a much broader range of metrics, including both the hardware and the software configurations. The regression suite consists of a range of unit/component tests alongside user applications. The full regression suite has been added as a final step in the regular monthly maintenance workflow, and its modular design means that specific tests can also be run while the machine is in production.

The regression suite enables us to provide feedback to Cray on pre-release versions of the CLE and PE as well as intermediate patches. The regression suite is capable of reporting on functional as well as performance bugs on the entire system as well as individual nodes.

As novel GPU usage scenarios are explored on the system in the development of CPU/GPU-hybrid codes, we have found that faults can be missed by the NHC, even with the `nvidia-healthmon` extended version which is implemented on Piz Daint. Recently, for example, an application failure was reported by a user that exposed an issue with the functioning of MPS. Although the node was sick, all vendor-provided tests (e.g. `nvidia-healthmon`) showed the node as healthy, because the vendor provided test suites did not have checks for MPS functionality. We quickly implemented a test that exposed the bug – based on the `simpleMPI.cc` code from Nvidia – and the test is now included in the regression suite. The regression suite can be called with an optional argument `--check-node=nidid` which will run all GPU sanity-related tests on a specific (read: suspect) node.

The regression suite (or subsets of the suite that check specific functionality) is launched at the conclusion of monthly maintenance sessions, after unexpected or expected downtimes, after programming environment updates, or during production time to check for suspected hardware or software faults. Some tests require the use of the batch system and to this end the regression suite checks for the existence of a Slurm reservation and will submit the test jobs to that reservation if it exists and is active. Some of the tests in fact *require* a specific Slurm reservation and will be skipped automatically if one is not present.

Each test has been classified as one of the following types:

- Configuration and Functionality ("it is there and meets some functionality requirements"), or
- Performance ("It meets some performance criterion/a")

Each test has also been categorized as one or more of:

- Application Performance (AP)
- System Performance (SP)
- System Management (SM)
- File systems/Storage (FS)
- Developer Environment (DE)
- Networking/Connectivity (NC)

These labels are mainly for information purposes, to help ensure the tests are providing adequate coverage of all system and software components.

The ease with which failures can be identified and debugged is a key design element for the regression suite, and apart from the tests that check for regression in application performance – where a number of tightly or loosely-coupled system components are being tested at the same time (batch system, file system mount points, node performance, network performance, Lustre, and so on) – the tests attempt as much as possible to test individual components in isolation. We wished to avoid situations where a test is marked simply as a "fail" because one component has failed inside a long and complicated workflow: a failed test that compiles a code, runs the executable in a batch job, and checks the performance obtained compared a reference is likely to be less informative than three separate tests that test each component separately (an additional test may be necessary to check the workflow itself, however).

In time, the tests will be categorized also in terms of their criticality, such that decisions can be made more easily about whether the system can be returned to users or not. The levels are still to be defined, but we envisage something along the lines of:

- Critical ("Can not be returned to users . . . ")
- Major ("Machine is usable, but . . . ")
- Minor or Purely Cosmetic

The classifications of criticality are important because they should determine whether or not a machine can be returned to the user community. To this end we are developing a set of processes that define how we should react to a given failure or failures in the regression suite. In some cases, system administrators will be able to determine easily the cause of a test failure and correct it immediately. In other cases they will need to communicate failures to staff responsible for application development, tuning and support, and after analysis, a consensus can be reached on whether or not it is safe to return the system into production.

In Figure 7, we show a partial output from a run of the regression suite launched from an internal login node of Piz Daint.

A. System Sanity

The first test in the regression suite simply parses the output from a system sanity script that is run on the SMW. This test checks the presence and functionality of critical system components such as Slurm, various file systems, load averages and disk space, amongst others (see `Test 1000` in Figure 7).

B. Programming Environment Regression

There are approximately 50 tests that assess the configuration and functionality of the programming environment on Piz Daint. Some of these tests are purely configuration-based: for example, we have tests that check that each programming environment can compile simple hello world-type codes, another test that checks if modules relating to architecture-specific targets provide the expected functionality (e.g., the generation of AVX instructions when the `craype-sandybridge` module is loaded), and another test that checks that symbols are resolved by the correct libraries according to which modules are loaded (e.g., calls to DGEMM are resolved by the accelerated LibSci library if the `craype-accel-nvidia35` module is loaded). Others check the performance obtained by various elements of the programming environment (e.g., numerical libraries) and compare them to known references: one such test checks the performance of LibSci's DGEMM over a range of matrix sizes and compares the results to reference values, both for accelerated and CPU-only versions of the library.

The regression suite has already identified half a dozen or more bugs related to the configuration of the programming environment, which have been reported to Cray. Some examples of these were:

- Serial codes failing to compile with `PgEnv-gnu` and dynamic linking
- `module unload` followed by `module load` not mimicking `module switch` behaviour
- Problems with Intel MKL configuration

```
robinson@daint03:~/daint-regression> ./run_regression
 Regression test started by robinson
 ===> start date  Wed Apr 30 13:37:55 CEST 2014
 Running on daint internal login ( daint03 )
 Regression testing output will appear in:
 /users/robinson/daint-regression/output/30-04-2014_13-37-55/idaint

 Test 1000: Check system health and sanity
 =============================================================================
   | Checking checkSystem.out was last modified within last 2 hours   [ OK ]
   | Checking the load average                                        [ OK ]
   | Checking for zombie processes                                    [ OK ]
   | Checking ntpd process                                            [ OK ]
   | Checking free space in /                                         [ OK ]
   | Checking free space in /snv                                      [ OK ]
   | Checking free space in /rr                                       [ OK ]
   | Checking if /apps is mounted on all nodes                        [ OK ]
   | Checking if /users is mounted on all nodes                       [ OK ]
   | Checking if $SCRATCH is mounted on all nodes                     [ OK ]
   | Checking the service nodes                                       [ OK ]
   | Checking the DVS nodes                                           [ OK ]
   | Checking lustre                                                  [ OK ]
   | Checking lnet router nodes                                       [ OK ]
   | Checking login nodes                                             [ OK ]
   | Checking down nodes                                              [ OK ]
   | Checking scratch usage                                           [ OK ]
   | Checking slurm frontend nodes                                    [ OK ]
   | Checking slurm partition                                         [ OK ]
 | Check system health and sanity                                   [ FAILED ]

 Test 5040: Modules loaded at login match for ext and int login nodes
 =============================================================================
 | Modules loaded at login match for ext and int login nodes        [ PASSED ]

 Test 5041: Modules in PrgEnv-intel match for ext and int login nodes
 =============================================================================
 | Modules in PrgEnv-intel match for ext and int login nodes        [ PASSED ]

 Test 5042: Modules in PrgEnv-gnu match for ext and int login nodes
 =============================================================================
 | Modules in PrgEnv-gnu match for ext and int login nodes          [ PASSED ]

 Test 5043: Modules in PrgEnv-pgi match for ext and int login nodes
 =============================================================================
 | Modules in PrgEnv-pgi match for ext and int login nodes          [ PASSED ]

 Test 5003: Ensure PrgEnv-cray is loaded by default
 =============================================================================
 | Ensure PrgEnv-cray is loaded by default                          [ PASSED ]

 Test 5020: Functionality of module unload PrgEnv-*; module load PrgEnv-*
 =============================================================================
 | Functionality of module unload PrgEnv-*; module load PrgEnv-*    [ PASSED ]

 Test 5010: Time for compiling hello world in PrgEnv-cray
 =============================================================================
 | Time for compiling hello world in PrgEnv-cray                    [ PASSED ]

 Test 5011: Time for compiling hello world in PrgEnv-intel
 =============================================================================
 | Time for compiling hello world in PrgEnv-intel                   [ PASSED ]

 Test 5012: Time for compiling hello world in PrgEnv-gnu
 =============================================================================
 | Time for compiling hello world in PrgEnv-gnu                     [ PASSED ]

 Test 5013: Time for compiling hello world in PrgEnv-pgi
 =============================================================================
 | Time for compiling hello world in PrgEnv-pgi                     [ PASSED ]

 Test 5004: Compile hello world in PrgEnv-cray
 =============================================================================
 | Compile hello world in PrgEnv-cray                               [ PASSED ]

 Test 5005: Compile hello world in PrgEnv-intel
 =============================================================================
 | Compile hello world in PrgEnv-intel                              [ PASSED ]

 Test 5006: Compile hello world in PrgEnv-gnu
 =============================================================================
 | Compile hello world in PrgEnv-gnu                                [ PASSED ]

 Test 5007: Compile hello world in PrgEnv-pgi
 =============================================================================
 | Compile hello world in PrgEnv-pgi                                [ PASSED ]

 Test 5009: craype-accel-nvidia35 resolves to libsci_acc in PrgEnv-gnu
 =============================================================================
 | craype-accel-nvidia35 resolves to libsci_acc in PrgEnv-gnu       [ PASSED ]

 Test 5014: Compiler generates AVX instructions PrgEnv-cray
 =============================================================================
   | craype-sandybridge module loaded                                 [ OK ]
   | Compiler flags for AVX found                                     [ OK ]
   | Compilation                                                      [ OK ]
   | AVX instructions generated in assembly code                      [ OK ]
 | Compiler generates AVX instructions PrgEnv-cray                  [ PASSED ]
```

Figure 7. Partial output from the regression suite created for Piz Daint.

- Undefined references to Trilinos symbols when linking dynamically

C. System Performance Regression

We have written a series of tests to check the floating-point performance on the node (with and without the GPU) for each and every node in the system. We intend to add tests that measure the interconnect performance and I/O subsystem in the near future.

D. Applications Performance Regression

The regression suite currently includes three scientific software applications (CP2K, NAMD, and GROMACS). In each case, we run both the CPU-only and the GPU-enabled versions, which can help debugging problems when tests fail. We intend to include additional software applications with time.

V. Case Study: Identification & Verification of Complex, Transient Bugs

GPU devices in a system bring a complex ecosystem, particularly so the GPU driver software. As mentioned before, this driver software allows for the configuration and manipulation of the GPU devices including its connection to the CPU in user and privileged modes. Users can write applications using the driver API. Like any piece of software, the driver can have bugs, which can be extremely difficult to identify, especially if the issues are transient and only occur at scale. Thus, on Piz Daint, since it has nearly 5,300 nodes, transient bugs are hard to detect and difficult to verify because it is a unique system at this scale. The system is in production and therefore cannot be taken offline for detecting bugs or isolating faulty components. We therefore had to devise a solution that allows us to screen and produce results on a live system.

We set the following design criteria for the screening solution:

- Full system coverage
- Statistics gathering
- Analysis of logs
- Simple interface for users
- Reporting
- Flexibility

In designing the solution, we exploited the Slurm scheduling system for creating special reservations, Logstash and custom scripts to produce reports. Logstash is an Open Source log management tool that allows logs collection in a single place in order to easily store, index and analyze log entries [10]. Logging on Piz Daint is managed with LLM (Lightweight Log Management). Logs belonging to different nodes (syslog and console logs) are collected on SMW node. Not all logs are handled by LLM (i.e. Slurm logs) so agents are defined on relevant nodes in order to have desired logs sent to Logstash.

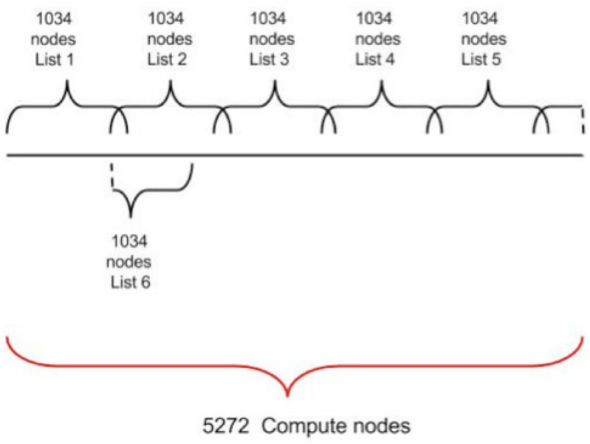


Figure 8. Overlapping reservations used for screening of Piz Daint using application runs of 1,024 nodes.

Using Slurm, we can create reservations that rotate around the system with some ghost nodes for buffering and overlapping. Output from the system logs are gathered in logstash and triggers can be placed for identifying known errors. Finally, the Slurm database is analyzed using custom scripts to ensure adequate screening has been performed on all available compute nodes.

Here we provide a brief overview of a concrete example of where we used our design to verify a driver patch. For this case study we used a production level application called DCA++ ([11]), which was being used on the system by a non-privileged (i.e. standard) user. The benefit of the test procedure is that the user could carry on using the application in the normal way and at the same time we could exercise the whole system over a period of time without the need for extended maintenance windows. In this way we did not use any compute cycles outside the production User Program.

The procedure involves the creation of ghost reservations to:

1) Provide a single reservation name for user so they do not have to generated multiple job scripts while the reservation rotates around the system (Slurm does not allow for two reservations with identical names);
2) Allow for buffering and overlapping of reservations.

Figure 8 shows a graphical depiction of the screening process that was customized for DCA++. The application ran on 1,024 nodes because the frequency of errors was higher at that scale. Ten extra nodes are assigned to each reservation for buffering purposes (in the case that nodes fail during job execution these buffer nodes allow the job to restart).

Two reservations are created at the beginning of the configuration (“dcapp” and “ghost”); “dcapp” is the reservation where jobs will actually run starting with the set of nodes in List1; “ghost” is the reservation where “dcapp” will be updated into, with the set of nodes in List2.

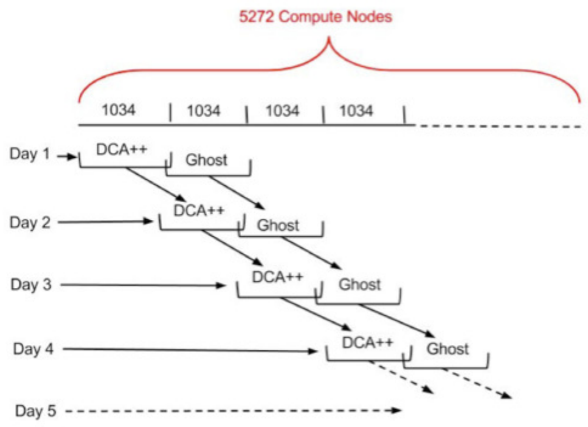


Figure 9. Rotating reservation mechanism allows the overlapping reservations to move across the system over time.

```
daint01:  # scontrol show res
ReservationName=dcapp StartTime=26 Jan 09:10
EndTime=Tomorr 07:34 Duration=1122:23:44
Nodes=nid0[0004-0191,0196-0383,0388-0451,
0456-0767,0772-0835,0840-1057] NodeCnt=1034
CoreCnt=16544 Features=(null)
PartitionName=(null)
Flags=MAINT,OVERLAP,IGNORE_JOBS,SPEC_NODES
Users=(null) Accounts=csstaff,s299
Licenses=(null) State=ACTIVE
ReservationName=ghost StartTime=Tomorr 07:34
EndTime=Sat 07:34 Duration=100:00:00
Nodes=nid0[1047-1151,1156-1535,1540-1919,
1924-1987,1992-2096] NodeCnt=1034
CoreCnt=16544 Features=(null)
PartitionName=(null)
Flags=MAINT,OVERLAP,IGNORE_JOBS,SPEC_NODES
Users=(null) Accounts=csstaff,s299
Licenses=(null) State=INACTIVE
```

Figure 10. Slurm reservations “dcapp” and “ghost”.

By means of a script the “dcapp” reservation was updated daily and switched to the “ghost” reservation using a new set of nodes (List1 $\rightarrow$ List2, List2 $\rightarrow$ List3, List3 $\rightarrow$ List4, List4 $\rightarrow$ List5, List5 $\rightarrow$ List6, List6 $\rightarrow$ List1, …) and “ghost” will be updated daily with a new set nodes as well. This process is represented graphically in Figure 9.

Figure 10 shows the “dcapp” and “ghost” reservations setup within Slurm.

The switch of “dcapp” to “ghost” and the update of “ghost” reservation is made by a script called by an entry in the crontab:

```
34 7 * * * /ufs/slurm/priorities/bin/
```

Table V
AGGREGATE DCA++ TEST TIME ACCUMULATED USING THE TEST PROCEDURE.

Number of Nodes	DCA++ Test Time (hr)
4062	>48
1037	>24
169	<24
4	0

```
create_dcapp_res.sh >>
/ufs/slurm/priorities/bin/day.log
```

Using this process we screened 1,024 compute nodes at a time in 24-hour cycles. Table V gives a summary of the screening cycle for primarily 1,024 nodes runs. As can be seen all bar 173 nodes ran the code for more than 24 hours.

VI. SUMMARY AND FUTURE PLANS

The first hybrid Cray XC30 system incorporates unique features in its system administration, management, monitoring and accounting environments. CSCS has introduced extensions to the Slurm scheduling system to allow for GPU operating modes that are needed by end users of the system. Furthermore, for supporting robust operations, CSCS has developed an extensive regression suite and a system screening mechanism which have been detailed in this paper. In future, we plan on continuing our collaboration with Cray and Nvidia to improve coverage of their system management and diagnostics tools. CSCS will continue to invest in the regression suite so as to improve on early detection and diagnosis. We are currently investigating additional GPU operating modes and features, specifically for OpenGL, to facilitate interactive visiualization using Piz Daint compute nodes.

ACKNOWLEDGMENT

The work reported in this paper would have not been possible without input and support from Cray, Nvidia and CSCS staff.

REFERENCES

[1] N. Bianchi, C. McMurtrie, and S. Alam, "CRAY XC30 Installation A System Level Overview," in *Proceedings of the Cray User Group Conference*, 2013.

[2] "GPU Deployment Kit." [Online]. Available: https://developer.nvidia.com/gpu-deployment-kit

[3] "The Power Management Database (PMDB)." [Online]. Available: http://docs.cray.com/

[4] G. Renker, N. Stringfellow, K. Howard, S. Alam, and S. Trofinoff, "Deploying Slurm on XT, XE, and Future Cray Systems," in *Proceedings of the Cray User Group Conference*, 2011.

[5] "Using Cluster Compatibility Mode (CCM) in CLE." [Online]. Available: http://docs.cray.com/

[6] Cray, "Writing a Node Health Checker (NHC) Plugin Test," *Cray Internal Document*, vol. S-0023-5002, 2013.

[7] "pyNVML." [Online]. Available: http://pythonhosted.org/nvidia-ml-py/

[8] G. Fourestey, B. Cumming, and L. Gilly, "First Experiences With Validating and Using the Cray Power Management Database Tool," in *Proceedings of the Cray User Group Conference*, 2014.

[9] Cray, "Managing System Software for the Cray Linux Environment," *Cray Internal Document*, vol. http://docs.cray.com/books/S-2393-51//S-2393-51.pdf, 2012.

[10] "Logstash: Open Source Log Management Tool." [Online]. Available: http://logstash.net/

[11] P. Staar, T. A. Maier, R. Solca, G. Fourestey, M. S. Summers, and T. C. Schulthess, "Taking a Quantum Leap in Time to Solution for Simulations of High-Tc Superconductors," in *Proceedings of the Supercomputing Conference SC13*, 2013.