Using Open XDMoD for accounting analytics on the Cray XC supercomputer

Thomas Lorenzen
Danish Meteorological Institute
tl@dmi.dk

Jason Coverston
Cray
jcovers@cray.com

***Abstract*—As supercomputers grow and accommodate more users and projects, the system utilization and accounting log files grow as well, often beyond easy native comprehension and thus requiring a flexible graphical tool for accounting analytics. This presentation will depict the joint effort of the Danish Meteorological Institute, DMI, and Cray to adapt Open XDMoD, http://open.xdmod.org, to the DMI Cray XC supercomputer. Extensions to the Cray RUR framework that monitor metrics of particular relevance to the site have been embedded into Open XDMoD and will be presented as well. This will show Open XDMoD to be a flexible tool for use with the Cray XC supercomputer, with strong potential for extending metrics ingestion and graphical presentation in numerous ways.**

***Keywords*—Accounting analytics, RUR, PBS Pro, Open XDMoD.**

I. INTRODUCTION AND MOTIVATION

Supercomputers, even those of modest size, are a significant investment, and thus stakeholders are in continuous need of ensuring proper return on investment. As supercomputers become increasingly complex with several specialized components, there is also elevated need to ensure that each of these are used appropriately. Additionally, stakeholders are most often in need of simplified graphical views as opposed to long tabular forms filled with numbers. Thus, a supercomputing facility must today be accompanied by an appropriate graphical analytics engine, which can analyze usage profiles to both oversee the system utilization and to quickly pinpoint possible suboptimal usage scenarios.

At the Danish Meteorological Institute, DMI, the supercomputer is the central nerve in the production of atmosphere, ocean, and climate forecasts. However, the supercomputer is still only a part of a larger production chain which includes pre- and post-processing on numerous peripheral servers, all of which are equally important in ensuring timely delivery of products. For seamless usage analysis and capacity planning across the production line, it is therefore important to fully integrate such a tool, which is agnostic to the different Workload Management tools in use across the different platforms.

Together, Cray and DMI surveyed the internet for such analytics tools and found Open XDMoD [1]. This merited further exploration, in particular due to it supporting back-end accounting logs for several Workload Managers, including the possibility of adapting to local customizations of accounting data output produced from such Workload Managers. This latter part appeared most appealing in connection with the presentation Enhanced Job Accounting with PBS Works and RUR [2], which discusses integrating RUR [3] statistics into accounting logs via the hooks framework of PBS Professional [4]. This paper outlines how DMI together with Cray has prototyped extending the gathering of RUR statistics and the seamless inclusion of these metrics into the accounting logs of PBS Professional for subsequent visualization and analysis with Open XDMoD as a graphical front end.

II. THE SITE AND SYSTEM

The Danish Meteorological Institute, DMI, is the national meteorological service responsible for forecasting the evolution of weather, ocean, and climate for the Kingdom of Denmark. As part of this obligation, DMI is using supercomputers to execute the numerical models governing these evolutions in order to produce the best possible forecasts for society.

In preparation for a future of tight collaboration between the national meteorological services of the Nordics, DMI entered into a partnership with its Icelandic sister organization IMO, the Icelandic Meteorological Office. The DMI supercomputer is located on the premises of the Icelandic Meteorological Office with IMO staff having responsibility for site infrastructure. Thus, the weather forecasts for the Kingdom of Denmark have been successfully produced from Iceland since spring 2016.

To ensure timely production and resiliency, DMI is operating two identical XC systems, one for production and one for research and development. Both systems share Sonexion and Netapp storage systems with full and transparent failover capability. Network communication is secured by means of alternate paths between the DMI site (location of forecasters) and the IMO site (location of the supercomputer).

The design of each individual XC system is configured to reflect the production nature of the model suites, which is

characterized by a chain of activities. Some chains are clear candidates for the compute nodes and some are more general purpose and limitedly parallel in nature and are therefore well suited for repurposed compute nodes, also called MAMU [5] nodes. At the site, MAMU nodes are named GPC (general purpose computing) nodes to be distinguished from the traditional compute nodes named HPC (high performance computing) nodes. Henceforth, this paper will use the terms traditional compute nodes and repurposed compute nodes to refer to HPC and GPC/MAMU nodes, respectively.

Both XC systems are scheduled and managed with PBS Professional, ensuring proper resource allocation for both production and research and development activities according to priorities set by management. Scheduling and resource management of the node types outlined above is accomplished on the queue level by means of PBS Pro vntype resource.

III. THE NEEDS AND TOOLS FOR LEVERAGE

The supercomputing tool is first and foremost a production tool, but due to resiliency requirements, there is a surplus of resources available for research and development. In this context, development focuses on continuous refinement of the operational forecast models, whereas research includes climate simulations. All activities must be monitored continuously to assess that project priorities are met and also that resource usage is in expected proportion to these priorities.

The need for a graphical tool to analyze system usage is not limited to making pie charts for system usage across projects. Nor is such a tool needed to dive into application tuning for every activity. The need is somewhere in the middle, where suspicious outliers for selected resources can be identified and caught for further analysis. The reason being is that users, who are typically researchers, cannot generally be expected to be experts at optimizing code and setup. The system analysts supply documentation and written guidelines including best practices, but establishing and maintaining these resources take time and do not guarantee they are read and adhered to. Suboptimal choices for code setup and execution are seen periodically, therefore a tool is needed to catch the outliers, which can have adverse effects on system utilization.

Thus, an overall need was identified for continuous monitoring and retrospective analysis of system usage on different levels of detail. This paper focuses on two items **(a)** and **(b)** below, which have been identified as requiring attention for pinpointing suboptimal user activity.

(a) The storage subsystems are shared resources, at least when it comes to bandwidth between compute and storage, but also when it comes to metadata activity. The Sonexion OSS resources are separated between production and research and development via lustre pools (the Sonexion system still has the MDS as a shared resource). Thus, a single research activity containing a suboptimal IO profile can influence all activities, including those production in nature, potentially slowing down their execution due to shared resource bottlenecks. Catching such detrimental activities in the act is not always possible and thus it is important to seek a way to retrospectively identify such activities.

(b) The nature of repurposed compute nodes is to serve multiple batch jobs at one time, each normally serial in nature, which makes for minimal waste of resources as compared to using traditional compute nodes, which are each allocated exclusively to one single job. However, the use of repurposed compute nodes is new at the site and puts the demand on users to be aware of which type of node is most adequate for their activity. It is important to continuously inform and guide users to use the repurposed compute nodes when appropriate.

The tools for metrics gathering and analysis will be identified and described below as will the assessment of mitigating the two problem types outlined above. Much of this will be about taking off-the-shelf tools and modifying them to local needs, followed by using them in a series of actions. This will be done in the following sections, sections IV through VII, ultimately leading to a revisit of the above items **(a)** and **(b)** at the end of section VII. A brief overview of the tools and their intended use is as follows:

A. Using RUR for extracting compute node metrics, inclusive of non standard metrics (section IV)

The paper will primarily focus on collecting metrics from traditional compute nodes. This node type is exclusively allocated to one single job and is easily identifiable. However, in light of the itemized topics above, customized RUR plugins need to be developed. Thoughts for how to extend the work to repurposed compute nodes will be outlined in the summary section.

B. Executing RUR from PBS Pro hooks to collect metrics on a per batch job basis (section V)

Traditional compute nodes are scheduled via ALPS, with users allocating these resources at batch job submission time. Thus, the execution of RUR on a per application basis is less relevant than the execution of RUR in the prologue and epilogue of each batch job. The PBS Pro hooks framework is used.

C. Using PBS Pro hooks to embed collected metrics into accounting logs to be read by Open XDMoD (section VI)

Resource utilization is logged in the accounting logs at job completion, but the metrics collected by means of RUR is not natively recognized by PBS Pro. However, PBS Pro contains the possibility of defining local custom resources, which then can be assigned values via the hooks framework. A mechanism of assigning RUR metrics to local custom resources will be exploited right before they are written to the PBS accounting log.

D. Adjust Open XDMoD to be able to visualize and analyze non standard localized metrics (section VII)

Just as with PBS Pro, Open XDMoD can be customized to recognize special non standard resources from the PBS Pro accounting logs. Thus, Open XDMoD can be extended to seamlessly ingest such special metrics, which can then be

analyzed.

IV. THE BACK-END ADAPTATIONS – USING RUR WITH CUSTOM METRICS

The RUR framework contains the ability to query traditional compute nodes for different types of metrics and statistics, including metrics such as energy consumption and memory usage profiles. A base RUR installation contains a number of ready-to-use plugins, while at the same time serving as foundation for local development of site specific RUR plugins. This has been convenient for the two problem types **(a)** and **(b)** outlined above, which are not easily addressable by the supplied ready-to-use plugins from Cray. In the context of this paper, the simple plugin for querying energy consumption has been fundamental as it adapts rather naturally to the two problem types at hand.

For the case of logging, if an activity makes unexpected and excessive use of IO subsystem resources, for instance by performing numerous metadata operations (e.g. continuously opening and closing the same file for small scale writing of data), `/proc/fs/lustre/llite/<lus-cli-id>/stats` is an obvious candidate to probe at job start and at job end. The format of `/proc/fs/lustre/llite/<lus-cli-id>/stats` is described in [6] and essentially contains a number of counters for bookkeeping and many different lustre client activities, including number of bytes read and written and number of files opened and closed since starting the node as a lustre client. Sampling of selected counters can be done with a customized RUR stage plugin for each compute node allocated to a batch job both at job start and end. The differences between the counters are calculated and aggregated for all the allocated compute nodes in the post RUR plugin. For reference, please consult sample code snippets 1 and 2, which are easily fitted into the same flow as seen in the RUR energy plugin.

Sample code 1 : Lustre metrics sampling, RUR stage

```
with open(''.join(glob.glob("/proc/fs/lustre/llite/*/stats")), 'r')
as f:
  for line in f.readlines():
    if re.match('read_bytes|write_bytes',line.strip().split()[0]):
      data[line.strip().split()[0]] = int(line.strip().split()[6])
      continue
    if
re.match('open|close|getattr|getxattr$',line.strip().split()[0]):
      data[line.strip().split()[0]] = int(line.strip().split()[1])
      continue
```

Sample code 2 : Lustre metrics aggregation, post RUR stage

```
with open(inputfile, "r") as f:
  for line in f:
    linedata = rur_unwrap_post_data(line)
    input = json.loads(linedata)
    for a in ['read_bytes', 'write_bytes', 'open', 'close']:
      output[a] += input[a]
```

Additionally, `/proc/stat` is an obvious candidate to probe at job start and end to determine if the allocated compute nodes are being utilized appropriately, i.e. using the entire node rather than one single core. The format of `/proc/stat` is described in [7] and lists the top counters for user, kernel, iowait, and idle states for each core. Other states are present as well, but they are not relevant to the compute nodes, only counters for states user, kernel, iowait, and idle are accounted for. Use of hyperthreading is not forced upon and thus the counters for these cores not included, thereby only sampling the first half of the listed cores of `/proc/stat`. Sampling of counters from `/proc/stat` can be done at the beginning and end of a customized RUR stage plugin for each compute node allocated to a batch job. The RUR post plugin calculates the differences between the counters for each of the user, system, iowait, and idle states, and those are aggregated for all allocated compute nodes of the job. Ultimately, these aggregated totals for each state are then compared to the sum of totals across all four states. This leads to four percentage numbers, summing up to 100%, and indicates how much of the job has spent in user, kernel, iowait, and idle mode, respectively. Jobs with high idle percentages are then suspected to be serial jobs, thus requiring further inspection as potential candidates for repurposed compute nodes. For reference, please consult sample code snippets 3 and 4, which are again easily fitted into the same flow as seen in the RUR energy plugin.

Sample code 3 : Processor metrics sampling, RUR stage

```
with open("/proc/stat", 'r') as f:
  cpu, user, nice, system, idle, iowait, irq =
f.readline().strip().split()
  for i in range(nppu):
    cpu, user, nice, system, idle, iowait, irq =
f.readline().strip().split()
    data['user'] += int(user)
    data['system'] += int(system)
    data['idle'] += int(idle)
    data['iowait'] += int(iowait)
```

Sample code 4 : Processor metrics aggregation, post RUR stage

```
with open(inputfile, "r") as f:
  for line in f:
    linedata = rur_unwrap_post_data(line)
    input = json.loads(linedata)
    output['nodes'] += 1
    totaltic=0.0
    for a in ['user', 'system', 'idle', 'iowait', 'irq',
'softirq']:
      totaltic += input[a]
    for a in ['user', 'system', 'idle', 'iowait', 'irq',
'softirq']:
      output[a] += input[a]/totaltic
  for a in ['user', 'system', 'idle', 'iowait', 'irq', 'softirq']:
    output[a] = 100*output[a]/output['nodes']
```

V. THE BACK-END ADAPTATIONS – EXECUTING RUR VIA PBS HOOKS

Having defined customized RUR plugins for collecting site specific metrics, now we show the execution strategy. The target nodes for RUR are the traditional compute nodes. Thus, RUR is closely affiliated with ALPS and is executed in the prologue and epilogue of aprun. However, the target goal for RUR is for it to be executed in the prologue and epilogue of PBS Pro. The main reason for this is that users reserve compute node access via PBS Pro and work submitted typically contains several aprun invocations, interspersed with serialized code executing on the MOM node associated to the job. Retrospective usage analysis becomes too fragmented if it is done for every aprun invocation and the time spent in serialized code executing on the associated front-end MOM node will also remain unanalyzed. Fortunately, the execution of RUR on a per batch job basis is possible by making use of the `-j` option to `rur_prologue.py` and `rur_epilogue.py`. Using the `-j` option with the batch job identifier as the argument will result in RUR output loggings identifiable with fields `apid: 0` and `jobid: <job-id>`.

More recent versions of PBS Pro have introduced the concept of hooks, which allow administrators to run custom Python code during numerous stages of the batch job lifecycle.

For example, a job's requested resources can be validated, modified, or amended via a `queuejob` hook prior to entering a queue, or a signal can be sent to a user via a `runjob` hook right before a job is sent to an execution host. PBS Pro hooks are a replacement for the traditional prologue and epilogue scripts via the `execjob_prologue` and `execjob_epilogue` hooks. Therefore, `rur_prologue.py` and `rur_epilogue.py` are run from within these hooks. All relevant structures and attributes of PBS Pro are available to the hook code via import of the PBS module

```
import pbs
job=pbs.event().job
```

One caveat is ensuring that RUR is only engaged for jobs allocating traditional compute nodes. This is accomplished by checking for the `arch` resource and engaging RUR if its value is `XT` as the following snippet shows.

```
If (str(job.Resource_List.arch) == 'XT'):
  rurcmd = '/path/to/rur_{pro,epi}logue.py' + ' -j ' + job.id
  os.system(rurcmd)
  pbs.event().accept()
```

For future data gathering centered around repurposed compute nodes, a similar probing attempt can be made to ensure metrics are collected only on these types of nodes.

VI. The back-end adaptations – Embedding RUR metrics into PBS accounting logs

Having established the execution of RUR on a per batch job basis, the final step is to have the RUR data embedded into the PBS Pro accounting log. This is the foundation for the graphical analytics tool. RUR outputs one file per batch job in json format, which is then casted into the form used in the PBS accounting logs, preferably on the fly, after the post stage of RUR has finished at the end of the batch job.

PBS Pro supports the definition of custom resources, and values can be added to these at will via the PBS Pro hooks framework. The following PBS Pro qmgr command defines the custom resources for the metrics gathered by RUR described above.

```
create resource read_bytes,write_bytes,open,close, type=long
create resource idle,user,system,iowait type=float
```

Within an `execjob_epilogue` PBS hook, the RUR output is parsed and assigned to these custom resources as simply as the following snippet.

```
for dmistats_metric in ["read_bytes", "write_bytes", "open",
"close", "idle", "user", "system", "iowait"]:
  job.resources_used[dmistats_metric] =
rur_data[jobid]['dmistats'][dmistats_metric]
```

One caveat to be aware of is the execution order of hooks in the case that a particular trigger point contains more than one hook, as is now the case for the `execjob_epilogue` trigger point. For this, the `order` hook attribute can be set to control the order of execution of several hooks. Hooks with a low value `order` hook attribute are executed before those with higher values.

VII. The front-end adaptations – Graphing and analyzing the PBS accounting logs

The former three sections resulted in extending PBS Pro accounting logs, which - for jobs using traditional compute nodes - contain custom metrics collected and aggregated from the compute nodes allocated to the job. These accounting logs will be the bread and butter for the front-end presentation along with Open XDMoD. Open XDMoD is a php based web application utilizing MySQL as the storage back end. It is presumably able to run on all newer linux distributions; however, only CentOS and Ubuntu seem to be actually tested by developers. Previous versions of Open XDMoD were supported on Ubuntu 14.04, and the latest version fortunately also outlines a recipe for how to establish a working Open XDMoD version on Ubuntu 16.04, thereby running on the latest Ubuntu LTS version. The experiences gained for the results presented in this paper stem from the latest Ubuntu LTS version being the base.

Just as with RUR and PBS Pro, Open XDMoD will need to be extended with the knowledge of the metrics, which are not part of standard PBS Pro accounting log lines. It has taken a bit of trial and error to identify all spots requiring adaptation and the code changes will not be detailed in this paper, since the changes, albeit simple, are somewhat verbose. Instead, the following paragraphs identify the parts of the Open XDMoD code that required adjustments. The authors can of course be consulted for further details.

Accounting data shall be shredded (parsed) and ingested (written) into the MySQL databases and tables via Open XDMoD commands. The receiving MySQL tables were modified for the custom metrics. A file named db/migrations/cray/cray.sql was created containing appropriate ALTER TABLE SQL commands to add columns to the MySQL tables, correlating to the addition of the custom metrics.

Each workload manager has its own format of accounting logs. The present case of adding custom metrics for shredding will be described accordingly. For this, a new file `Craypbs.php` beneath `classes/OpenXdmod/Shredder` was defined with inspiration from `Pbs.php` in the same directory, but with adaptation to the Cray version of PBS Pro containing the amendment of custom metrics.

For ingesting the shredded data, the file `classes/OpenXdmod/Ingestor/*/Jobs.php` was adjusted to include the custom metrics.

Finally, for each custom metric, the files `Average<metricname>.php` and `Total<metricname>.php` were created beneath `classes/DataWarehouse/Query/Jobs/Statistics`. These contain appropriate descriptions of the metric and any possible processing thereof prior to visualization, such as converting the ingested data from seconds to hours or from bytes to megabytes .

After adapting the code base and installing Open XDMoD, the accounting data can be shredded and subsequently ingested into the storage back end with

commands `xdmod-shredder -v -r <resource> -f craypbs -d /path/tl/<resource>/acctfile` and `xdmod-ingestor -v`, respectively. Doing this shredding and ingesting on a daily basis by means of remote file copying from the XC system is assigned to cron. With a recent work week's worth of accounting data, it is now time to introduce the web interface of Open XDMoD.

The Open XDMoD web interface has a number of tabs at the top for creating both dashboards and reports. The help button in the upper right corner, which is context sensitive, provides help in accordance to which part of the Open XDMoD interface is presently being used. This paper will not explore all the functionality, but rather will show what can be visualized. For this, go to the Usage tab, which can be considered the first place to dig around and get experience before defining dashboards and reports. Activating the Usage tab uncovers a list of metrics to be selected, seen in the lower left part of the web interface. When a metric is selected, the visualization will appear in the lower right part of the web interface, and will reflect the chosen display options, which can subsequently be adjusted to taste. Clicking into the graphics presents further options for drilling down into the displayed statistics. This allows quick identification of which user and job type show an unusual usage pattern with respect to the selected metric. Many other use cases are possible, but with just these simple handles it is possible to assess the items **(a)** and **(b)** from section III.

(a) : As per the graphics in Figures 1 and 2, assessments about abnormal lustre metadata activity can be made by first selecting the tab for total count of lustre file open operations, for example. This will create a graph of the total number of such operations on a per day basis. Changing to a bar chart via the Display button, followed by clicking in the graph, presents a drop down menu. There, selecting to order by PI, Principal Investigator (here equivalent to the unix group) leads to the graphics in Figure 1. Clicking on a bar in the chart will present other attributes and allows the user to further drill down for this selected PI. This ultimately leads to the graphics in Figure 2, where the count of lustre open file operations metric has been drilled down to the node level via PI -> User -> Node count. This interestingly shows that for this user, his or her jobs which allocate thirty-two nodes (blue bars) are very much lustre metadata intensive. When approaching the user, as he or she presumably may not know in detail which of his or her jobs could be lustre metadata intensive, being able to narrow down the options will certainly be helpful for quickly identifying the culprit. Also very interesting are the bars with the second highest lustre metadata operation count (red bars), which appear to be only single core jobs and may be better suited for repurposed compute nodes.

Figure 1 : Starting to look at lustre files being opened

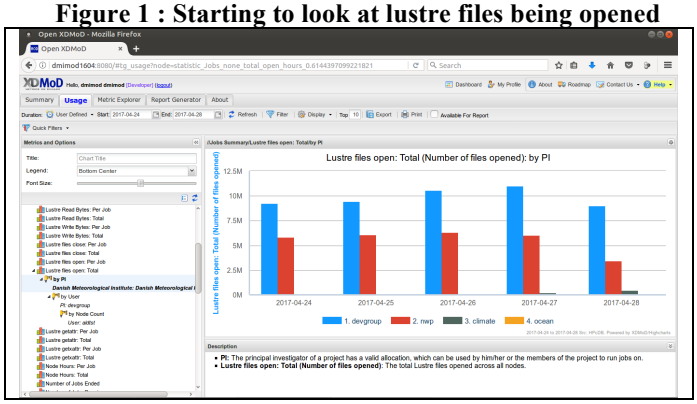
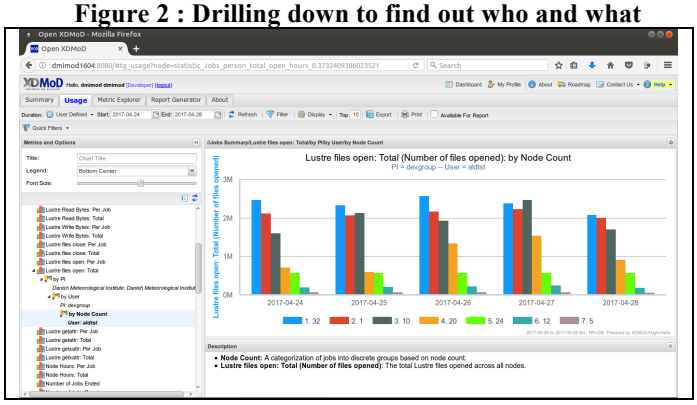


Figure 2 : Drilling down to find out who and what



(b) : As per the graphics in Figures 3 and 4, one can see suspiciously low processor utilization. Selecting the Usage tab for total amount of idle processor time will create a graphic of the total amount of idle processor time on a per day basis. Change the graph to a pie chart to see a summarization over the full time period instead of visualizing statistics per day. This can be achieved via the Display button, followed by clicking in the graph to activate a drop down menu and selecting to order by PI, Principal Investigator (here equivalent to the unix group). The results are shown in Figure 3. Clicking on a section of the pie chart will present other attributes and allow the user to drill down into this selected PI. This ultimately leads to the graphics in Figure 4, where the amount of processor idle time metric has been drilled down as PI -> User -> Node count. This confirms for this user, whose jobs were top rank in allocated processor idle time, that the majority of this resource waste is most probably due to jobs mistakenly being placed on the traditional compute nodes when they should have been placed on the repurposed compute nodes.

Figure 3 : Starting to look at idle processors

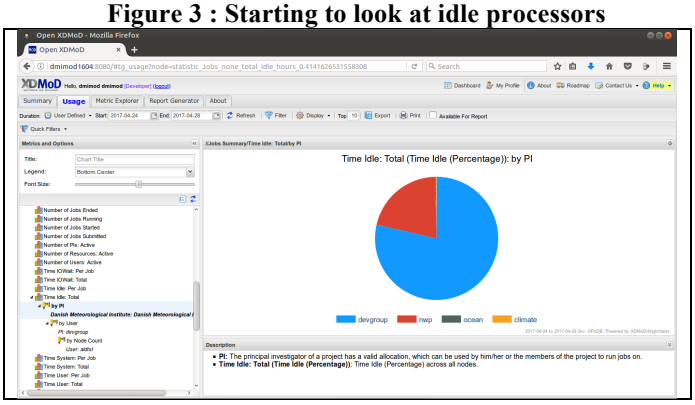
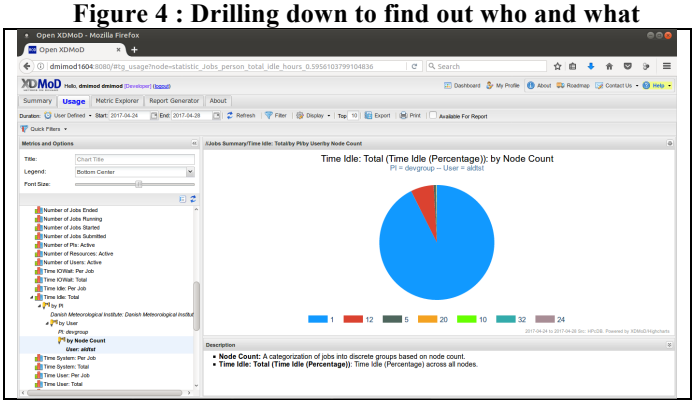


Figure 4 : Drilling down to find out who and what



VIII. First Experiences and Further Outlook

The presented examples are mostly to be considered as initial experiences, but they seem promising as an intuitive way to dive into the accounting statistics, either by a novice to the system details who only cares for overall system usage information, or by the systems analyst who wants to dive deeper into how a particular resource is being used or misused and by whom. The level of detail has been restricted to batch job granularity, and thus does not go into details of each individual ALPS aprun invocation in the case that there are several inside of the same batch job. An immediate thought for improvement would be to allow for both batch job identifiers and ALPS identifiers to be made part of the workload manager accounting and subsequent presentation inside of Open XDMoD.

The examples provided in this paper are centered only on the traditional compute nodes of the XC system, but repurposed compute nodes have turned out to be a vital resource and complement to ensure users have the right tool available for the right job. However, in its present form, RUR is tightly connected to ALPS and is thus not adaptable to repurposed compute nodes. Furthermore, repurposed compute nodes are not exclusively allocated to a single job, so monitoring system counters like those in `/proc/fs/lustre/llite/*/stats` or `/proc/stat` does not easily translate to the use of resources of a particular job executing on a repurposed compute node. An idea to be investigated is that of cgroups [8], a hook control mechanism inside of PBS Pro which has recently become sufficiently mature to be relied upon. Under the control of cgroups, information about resource allocation and use for such jobs can be found in `/sys/fs/cgroup/*/pbspro/<job-id>`. Querying `/sys/fs/cgroup/*/pbspro/<job-id>` of repurposed compute nodes from within PBS Pro `execjob_prologue` and `execjob_epilogue` hooks could possibly provide metrics for repurposed compute node job analogous to those presented above for jobs targeted to traditional compute nodes. By making the same types of adaptations that were made for customized RUR accounting metrics, these results would then easily make their way into Open XDMoD.

The above examples just scratch the surface of the mountain of possibilities, but they show that Open XDMoD is easily capable of adapting to locally desired metrics. Additionally, the XC system, enhanced with custom metrics from RUR and potentially also metrics collected from the cgroups framework, can seamlessly make its way into being used in Open XDMoD. Furthermore, Open XDMoD is capable of ingesting accounting data from a plethora of different workload managers, making it a capable swiss army knife tool for data usage analysis and capacity planning across the full production line and at both the management and the systems analyst levels. Thus, in this world of data, it seems that the biggest challenge will not be to make data available in a somewhat *consistent* form across diverse compute platforms, but instead to provide "the *right* information, at the *right* time, in the *right* place, in the *right* way to the *right* person" [9].

Acknowledgment

Thomas Lorenzen and Jason Coverston thank their respective employers for being given the possibility to collaborate on this project and share its outcome with the user community.

References

[1] http://open.xdmod.org.

[2] Scott Suchyta, Enhanced Job Accounting with PBS Works and Cray RUR, https://cug.org/proceedings/cug2014_proceedings/includes/files/pap198-file2.pdf.

[3] CRAY CLE XC System Administration Guide S-2393-5204xc, section "Resource Utilization Reporting".

[4] Altair PBS Professional 13.1 Administrator's Guide, section "Hooks".

[5] CRAY CLE XC System Administration Guide S-2393-5204xc, section "Service node MAMU".

[6] http://doc.lustre.org/lustre_manual.xhtml, "Lustre Software Release 2.x Operations Manual", "Lustre Parameters".

[7] https://www.kernel.org/doc/Documentation/filesystems/proc.txt.

[8] https://www.kernel.org/doc/Documentation/cgroup-

v1/cgroups.txt.

[9] Ashley Barker, The Complexity of Arriving at Useful Reports to Aid in the Successful Operation of an HPC Center https://cug.org/proceedings/cug2013_proceedings/includes/files/protected/pap193-file2.pdf.