Challenges of Workload Analysis on Large HPC Systems; A Case Study on NCSA Blue Waters

Joseph P. White
jpwhite4@buffalo.edu
Center for Computational Research
University at Buffalo, SUNY
Buffalo, NY

Martins Innus
minnus@buffalo.edu
Center for Computational Research
University at Buffalo, SUNY
Buffalo, NY

Matthew D. Jones
jonesm@buffalo.edu
Center for Computational Research
University at Buffalo, SUNY
Buffalo, NY

Robert L. DeLeon
rldeleon@buffalo.edu
CCR

Nikolay Simakov
nikolays@buffalo.edu
CCR

Jeffrey T. Palmer
jtpalmer@buffalo.edu
CCR

Steven M. Gallo
smgallo@buffalo.edu
CCR

Thomas R. Furlani
furlani@buffalo.edu
CCR

Michael Showerman
mung@illinois.edu
National Center for Supercomputing Applications (NCSA)

Robert Brunner
rbrunner@illinois.edu
NCSA

Andriy Kot
andriy@illinois.edu
NCSA

Gregory Bauer
gbauer@illinois.edu
NCSA

Brett Bode
brett@illinois.edu
NCSA

Jeremy Enos
jenos@illinois.edu
NCSA

William Kramer
wtkramer@illinois.edu
NCSA

ABSTRACT

Blue Waters [4] is a petascale-level supercomputer whose mission is to greatly accelerate insight to the most challenging computational and data analysis problems. We performed a detailed workload analysis of Blue Waters [8] using Open XDMoD [10]. The analysis used approximately 35,000 node hours to process the roughly 95 TB of input data from over 4.5M jobs that ran on Blue Waters during the period that was studied (April 1, 2013 - September 30, 2016).

This paper describes the work that was done to collate, process and analyze the data that was collected on Blue Waters, the design decisions that were made, tools that we created and the various software engineering problems that we encountered and solved. In particular, we describe the challenges to data processing unique to Blue Waters engendered by the extremely large jobs that it typically executed.

CCS CONCEPTS

•General and reference →Performance; •Software and its engineering →Software design engineering;

PEARC17, New Orleans, LA, USA
© 2017 ACM. 978-1-4503-5272-7/17/07…$15.00
DOI: http://dx.doi.org/10.1145/3093338.3093348

KEYWORDS

Measurement techniques, Modeling techniques, Performance attributes, Reliability, availability, and serviceability.

1 INTRODUCTION

Blue Waters [4] is a petascale-level supercomputer whose mission is to enable the U.S. scientific and research community to solve “grand challenge” problems that are orders of magnitude more complex than can be carried out on other high performance computing systems. Given the important and unique role that Blue Waters plays in the U.S. research portfolio, it is important to have a detailed understanding of its workload in order to guide performance optimization both at the software and system configuration level as well as inform architectural design decisions. Furthermore, understanding the computing requirements of the Blue Water’s workload (memory access, IO, communication, etc.), which is comprised of some of the most computationally demanding scientific problems, will help guide changes in future computing architectures, especially at the leading edge. With these objectives in mind, we carried out a detailed workload analysis of Blue Waters [8].

The workload analysis used various data, such as compute node performance metrics [11], that had been previously collected on Blue Waters. These node-level data were converted to job-level data which were then analyzed using Open XDMoD [10]. The data conversion itself was a challenging computational problem — requiring more than 35,000 node hours (over 1.1 million core hours) on Blue Waters. We analyzed roughly 95 TB of input data from over 4.5M jobs that ran on Blue Waters during the period of our analysis

(April 1, 2013 - September 30, 2016) spanning the beginning of Full Service Operations for Blue Waters to the recent past. In the process, approximately 250 TB of temporary data across 100M files were generated. The output data were subsequently stored in MongoDB and a MySQL datawarehouse to allow rapid searching, analysis and display in Open XDMoD. A workflow pipeline was established so that data from all future Blue Waters jobs will be automatically ingested into the Open XDMoD datawarehouse, making future analyses much easier. It is important to note that a significant challenge of the analysis was our requirement to process a backlog of more than three years of performance data as opposed to having it ingested daily by Open XDMoD, which is more than capable of processing the daily volume of Blue Water's data.

2 BACKGROUND

In this section we describe the source data that was available on Blue Waters for the workload analysis. We also list the tools that were used to process the data and discuss the reason for our tool choices.

2.1 Source Data

The workload analysis relied on data that had previously been collected on Blue Waters. This included accounting logs, metrics from the compute nodes (such as OS and driver metrics and hardware performance counters), application launch information, I/O metrics from instrumented software and library usage data. It also used information about the project allocations, such as the field of science. The various data sources are summarized below.

2.1.1 System Accounting Logs. Job information was provided by the Torque resource manager, which operates in coordination with the Cray ALPS placement scheduler and the MOAB job scheduler. Log entries are created when a job on Blue Waters passes through various states of execution. This includes job submission, changes of scheduler status (queued, eligible to run, start/running, etc.), and completion. The log entry for job completion includes various information about the job including the exit status, the project allocation and assigned compute nodes.

The log data were in text files with one log file per day. Each file was approximately 10 MB in size.

2.1.2 Compute node metrics. The Blue Waters system is a Cray XE/XK hybrid machine composed of AMD 6276 "Interlagos" processors all connected by the Cray Gemini torus interconnect. There are 22,640 Cray XE6 compute nodes. The 4,228 Cray XK7 compute nodes also include NVIDIA GK110 (K20X) "Kepler" GPUs.

The Lightweight Distributed Metric Service (LDMS) is software that provides data collection and transport of metrics from the compute nodes [1]. The implementation of LDMS on Blue Waters is described in detail in [11]. The LDMS data collection provides information on several subsystems of the compute nodes including load average, memory usage, limited filesystem data transfers and network utilization. Data are collected every minute on every compute node. For the XK nodes, GPU utilization and GPU memory utilization are also collected.

The CPUs on the compute nodes have programmable hardware performance counters. These counters are configured to count various events within the CPU such as the number of clock ticks, the number of instructions retired and the number of floating point operations performed. The hardware counters are accessed via machine-specific registers (MSRs). Throughout this document we use the abbreviation MSR to refer to the information read from the hardware performance counters on the compute nodes.

LDMS data was exported from the data store in CSV-format files with one file per day. Each file contained data for all the compute nodes, but due to the way the data were exported, the order of the compute node data between different files was not deterministic. The file sizes varied, but were each approximately 56 GB.

The MSR data was stored in CSV-format files with each file containing the hardware performance counter measurements from approximately one quarter of the compute nodes covering an approximately 45 day period. The file sizes varied, but were each approximately 900 GB.

2.1.3 I/O metrics. Darshan [5] is a low overhead HPC I/O characterization tool that is designed to capture an accurate picture of application MPI I/O behavior, including properties such as patterns of access within files. It is implemented as a set of user space libraries. These libraries require no source code modification and can be added transparently during the link phase of MPI compiler scripts. Darshan provides very detailed reports for file-system access for each executable linked with Darshan support.

Darshan output files were stored in the Darshan binary log format with one file per job. The log files were stored in a nested directory structure with one directory per job. The job directories were stored in month directories.

2.1.4 Library information. ALTD and XALT tools [2] provide tracking capabilities for utilization of statically and dynamically linked libraries. XALT is a more recent enhancement of the ALTD software. ALTD and XALT allow one to analyze which libraries and modules were used by each job.

The ALTD and XALT data were stored in a MySQL database.

2.1.5 Application information. The ALPS `aprun` command is used to launch applications on the Blue Waters compute nodes. The `aprun` logs provide the primary means to determine what application(s) was executed during the job. In most cases, this information includes the executable path, the number of nodes requested and the requested layout of the tasks on the cores of the nodes.

The `aprun` logs were collected and stored in a relational database. For the workload analysis, we used a CSV-format dump of the data that contained the job id, aprun command line, aprun unique identifier and the timestamps of the start and end of the execution of each command. The entire `aprun` data set was stored in a 1 GB gzip compressed file.

2.2 Analysis tools

The primary software used for the analysis was the comprehensive HPC system management tool Open XDMoD [10]. Open XDMoD is designed to be able to store, analyze and display information for hundreds of millions of HPC jobs. For example, the XDMoD instance that contains the accounting data for the NSF funded XSEDE program has approximately 150 million job records for jobs that consumed over 11 billion core hours [6]. An important capability of Open XDMoD, particularly for the workload analysis, is its ability to analyze detailed job level performance data. The default install includes support for comprehensive statistics on: number and type of computational jobs run, resources (computation, memory, disk, network, etc.) consumed, job wait times, and quality of service. The web interface is intuitive, allowing one to chart various metrics and interactively drill down to access additional related information.

Another key feature of Open XDMoD is its ability to combine and compare data across multiple data sources. For example, it provides easy tools to be able to break down resource usage by the field of science of a job's allocation and analyze the performance data for jobs by various categories.

The Open XDMoD software suite provides tools for processing, analysis and display of job level performance data, and uses existing tools to collect the raw data. The recommended data collection software is Performance Co-Pilot (PCP), which is an open source toolkit designed for monitoring and managing system-level performance. However, there is no requirement to use PCP, for example, the XSEDE implementation of XDMoD uses data collected by the open source `TACC_Stats` software [7] it also supports the ingestion of performance data from Cray's RUR tool [3] and Ganglia [9].

Open XDMoD has a flexible configuration system, which makes it straightforward to modify the data schema to add new metrics. This capability was useful for the workload analysis as it allowed us easily to add required data filters and dimensions. For example, we wanted to be able to filter data based on the type of the compute node that the job was assigned or by the concurrency type.

3 IMPLEMENTATION

This section gives an overview of the design and implementation of the data processing workflow that produced job level information from the raw source data.

The main Open XDMoD software was installed on a dedicated server, which was not part of Blue Waters. The install included the Open XDMoD portal software, web-server, the MySQL database that contained the Open XDMoD datawarehouse and the MongoDB document database. The Open XDMoD server had IP network connectivity to the Blue Waters compute nodes and the web-server was accessible to the Internet. The various data conversion and summarization software was installed in the `/projects` filesystem on Blue Waters. The majority of the data processing was run in batch jobs on Blue Waters.

There were two plausible designs for the data processing architecture. We could have modified the Open XDMoD summarization software to process the CSV format metric data directly, or we could convert the CSV data into PCP format archives and then use the existing summarization software with no modifications. We chose the latter option, that is, convert the existing metric data into PCP format archives and process them using the existing Open XDMoD software suite. The main reasons for this decision were that it allowed us to use the existing, production-tested release software, we already had experience with writing conversion scripts from a different project where we converted metric data from Ganglia to PCP and we anticipated that it would require less developer effort. The main disadvantage of this approach was that it necessitated the creation of a large numbers of temporary files (between three and six files per compute node per day). These files were stored on the `/scratch` Lustre filesystem on Blue Waters. Lustre was designed for large sequential I/O workloads and typically performs very well with large files but does not perform well with many small files. Another disadvantage is that it required extra time for the conversion step to run before the summarization step could start. This slowed down the software development process somewhat, but had only a minor impact once the dataflow was up and running.

The overall data processing workflow was as follows:

(1) Ingestion of the accounting data into Open XDMoD.
(2) Conversion of the compute node metric data from CSV format to the PCP archive format.
(3) Summarization of the node-level PCP archive data to create job-level records that contain overall statistics of the performance metrics collected during each job.
(4) Appending additional data to the job summary records, such as application information and I/O metrics.
(5) Ingestion of the job-level summary record data into Open XDMoD.

Each of these steps is described in more detail below.

3.1 Accounting data ingestion

Open XDMoD has native support for the Torque accounting log format so the standard ingest procedure was used. The accounting data were loaded into the database by the `xdmod-shredder` script and the data are post-processed by the `xdmod-ingestor` script.

The `xdmod-shredder` and `xdmod-ingestor` scripts store all of the job data in memory before it is loaded into the database. The consequence of this is that there is a limit on the number of jobs that can be ingested for each run of the script. This upper limit is defined by the memory available on the Open XDMoD server. To bulk load the historical data we created a simple shell script that ran the shredder and ingestor scripts multiple times, each processing one day's worth of data at a time.

3.2 Data format conversion

The metric data format conversion workflow is shown in Figure 1 below. A python script `ldms2pcp` was written to convert the CSV files from the LDMS collector into PCP archive files. The `ldms2pcp` script parsed each input file and wrote one PCP archive for each compute node containing one day of data. The `ldms2pcp` script also wrote metadata about each created PCP archive to the XDMoD datawarehouse[1]. This metadata includes the timestamps of the first and last record in the archive and its location on the filesystem. This metadata is used subsequently in the summarization step.

The LDMS collection was configured so that all nodes had a common metric set which was a superset of all metrics available on the compute nodes and service nodes [11]. The values are set to zero for nodes where the raw data does not exist. The Open XDMoD summarization software, however, does distinguish between the

[1]A utility `indexarchives` is provided with the Open XDMoD software to generate the archive metadata. It traverses the filesystem and records the metadata from existing PCP archives. Incorporating the metadata generation into the `ldms2pcp` eliminated the need to run this I/O intensive process.

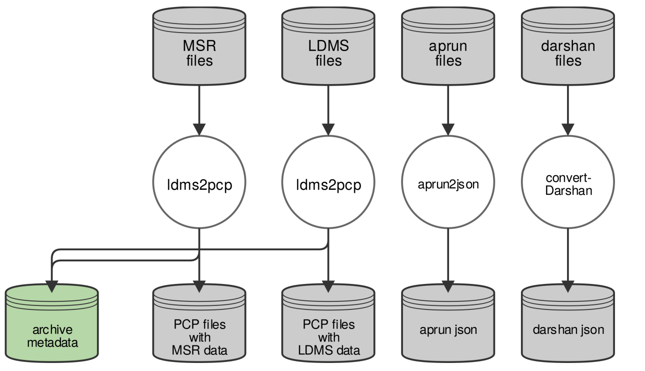

Figure 1: The Data flow diagram for the data conversion step. The gray shading indicates data stored in files on the filesystem (Lustre). Green shading indicates data stored in a relational database (MySQL). Note that the various configuration files for the `ldms2pcp` script are not shown here.

absence of a metric and a metric with value of zero. The `ldms2pcp` software therefore had to have a database with the node capabilities so that it would output PCP archives with the correct content.

The MSR data was stored in separate CSV files from the other LDMS metric files. The `ldms2pcp` script was also used to convert this data into PCP archives. We did not need to merge the two PCP archives during the conversion process because the summarization software automatically merges all of the PCP archives for each job.

The `ldms2pcp` script was designed to be able to handle data from an HPC system that has a variety of performance data sources and that changes over time. The bulk of the script is an engine that takes input data in CSV format and converts it to PCP archives according to a set of rules. These rules are defined in a configuration file, where each set of rules can have an associated date range. This was important in analyzing the Blue Waters data since the LDMS and MSR collection parameters changed several times over the four years of operation. The `ldms2pcp` code was able to automatically choose the correct set of rules based on the timestamps of the metrics to be converted.

This script was run in HPC batch jobs on Blue Waters XE nodes. Overall, the data format conversion consumed 10,000 node hours and generated approximately 250 TB of PCP archives spread over 100M files. Due to the size and number of the files it was necessary to find any strategy that would increase the performance of this processing step. We investigated parallel I/O patterns and found that parallel reads of the raw LDMS files did not help performance. However, we did find a significant gain by writing the PCP archives in parallel. We were able to utilize all 32 cores in an XE node during the conversion and write steps of this process.

The `aprun` data and Darshan data were each converted to files in json format by dedicated scripts. The Darshan conversion script was written in python and called the `darshan-parser` software to read the binary-format Darshan output files into a json format. The `aprun-parser` script also was written in python. The `aprun-parser` script generated multiple json files with 100,000 jobs per file.

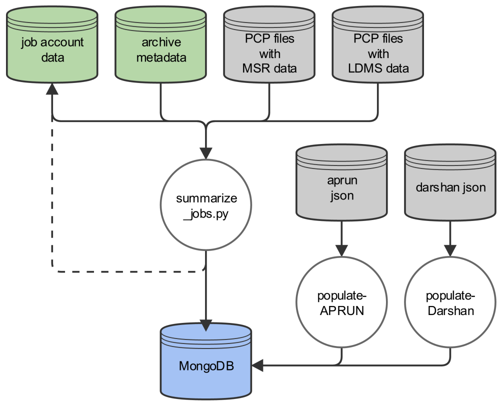

Figure 2: The data flow for the summarization step. The gray shading indicates data stored in files on the filesystem (Lustre). Green shading indicates data stored in a relational database (MySQL). Blue shading indicates data stored in a document database (MongoDB). Note the various configuration files for the scripts are not shown in this diagram.

3.3 Summarization Step

The Open XDMoD performance data summarization software converts the compute node-level metric data into job-level metric data. The software has a plugin-based architecture and generates a wide range of statistics on the various job-level metrics, generates time-series summaries and runs anomaly detection algorithms. Figure 2 shows the data flow diagram for this step. The `summarize_jobs.py` script queries the Open XDMoD datawarehouse to get a list of jobs to summarize and then uses the node-level data (in PCP archive format) to create job level summary documents that are stored in the document database. The script also stores metadata about the job summarization progress in the Open XDMoD datawarehouse (shown as the dotted line in the figure). This metadata is used to track which jobs have been processed.

The original design was to use the existing summarization code without modification. However, the semantics of the LDMS data were sufficiently different to require some modifications to the summarization code. The summarization code had support for processing the per-numa node memory metrics, but the per-compute node memory metrics had been recorded on Blue Waters. The MSR data recorded the raw values from the 48 bit wide hardware counters. However, the summarization code was written to process 64 bit wide counters[2]. The operating system CPU usage metrics were not collected on Blue Waters, however the Open XDMoD software assumes that they are present.

Fortunately the summarization software was designed with a plugin-based architecture, so it was easy to add new components to process the new metrics. We added a memory usage plugin that used the node-level memory metrics and a CPU usage plugin that used the hardware counter clock tick metric to estimate the CPU

[2]The PCP hardware counter data collector uses the Linux kernel perf events interface which handles the 48 bit counter overflow within the kernel and reports the data in a 64 bit wide counter.

usage. We also added plugins to process the load average metrics and the metrics from the Cray Gemini interconnect.

The code to convert metrics from 48 bit to 64 bit counters was implemented in the summarization software framework. The new functionality was configurable and allowed the metrics to be range converted to be specified in the configuration file.

We also made some other changes to the summarization software framework to make best use of the Blue Waters batch environment. We added full support for running the software in parallel using the python MPI bindings mpi4py. This allowed us to submit multi-node batch jobs to summarize large numbers of jobs at a time. We found that running on 8-12 nodes at a time to process 1 month worth of jobs was an ideal job size. The summarization step generated the 4.5M job records that were stored in the MongoDB and required 20,000 node hours. We added several new configuration options that allow more fine-grained control over which jobs are processed. One example of this is restricting the jobs to be processed by their size. We discuss job size and processing constraints in section 4.1 below.

3.4 Additional data

The default Open XDMoD data flow uses information about the running processes that is gathered by PCP on the compute nodes. This information was not recorded on Blue Waters, so instead we used the information from the aprun and Darshan logs. Rather than modify the summarization software, we opted to append the aprun and Darshan log data to the job summary documents using a separate script.

The aprun and Darshan processing scripts update the job documents in the document database with the aprun and Darshan information. Each script will only update an existing job record therefore the script should be run after the summarization script execution has completed and the job record created. The scripts will never delete data and therefore may be safely run multiple times with the same input.

The Open XDMoD ingestion software for the summary documents marks each job document when it is ingested into the Open XDMoD datawarehouse. This marker is checked by the ingestion software so that it does not ingest the same data again on subsequent invocations. The aprun and Darshan processing scripts cleared this marker for job documents that were modified. This ensured that the ingestion script would pull in the correct updated data on the next run.

3.5 Performance data ingestion

Before we started the workload analysis project, we identified a set of metrics to include in the analysis. Many of these were already supported in the main Open XDMoD release, but some were not. We chose to add the missing information as new statistics and dimensions to the Open XDMoD instance. The performance data realm in Open XDMoD uses a set of configuration files to define the statistics and dimensions and a mapping file for each different HPC resource that defines how the source data (from the job summary document in MongoDB) is transformed into the representation of the data to be loaded into the datawarehouse.

The new dimensions included:

- A "Job Type" dimension that categorizes jobs based on the concurrency type (MPI, threaded or serial);
- An "Active Cores" dimension that categorizes jobs based on the number of CPU cores that were active during the job;
- A "Node Type" dimension that categorizes jobs based on the type of the compute node that the job was assigned (XE or XK);
- An "Application Science Area" dimension that categorizes jobs based on the science area of the application executed.

We also added an additional job size dimension based on the node count of the jobs (the default uses the job core count) since Blue Waters allocates full nodes to a job. All of our analyses were done with respect to node count as opposed to core count. Another extra dimension categorizes jobs based on the amount of data transferred to the filesystem and over the interconnect.

The additional statistics included:

- An "Effective CPU Usage" statistic which gives the CPU usage of the cores that were active each the job;
- A "Load Average" statistic which gives the mean value of the system load average during each job;
- A "GPU usage" statistic which gives the value of the GPU usage reported by the GPU driver.

We also added new memory statistics that reported the memory usage per node in addition to the existing memory usage per core value.

We modified the appropriate configuration files and dataset mapping file and then used the standard procedure to ingest the data into the Open XDMoD instance.

3.6 Data coverage

Figure 3 and Table 1 present an overview of the availability of source data. aprun data coverage is fairly complete for the life of the Blue Waters system, both in terms of node hours and number of jobs run. This allowed for a fairly comprehensive analysis of the applications being run on the system.

As seen in Figure 3, Darshan data was available for a fairly small percentage of jobs and much of it concentrated during a relatively short time period at the end of 2015. This limited the amount of analysis that could be done on individual I/O operations performed by the applications since the Darshan data was the only source of discrete I/O operations. The LDMS data provided a time series representation of aggregate I/O.

Compute node performance metrics were collected starting approximately one year after the Blue Waters system went into service. Excluding this initial time period, coverage for LDMS data is fairly complete. Data is missing sporadically for a variety of reasons including downtimes and collection errors. Information about the GPUs on the XK nodes is available for a shorter time. There are two main reasons for this: GPU data collection was not implemented until January 2015 and there was a gap in data availability at the end of 2015 due to a graphics driver issue.

MSR data exists for roughly 12 months of the analysis period. As the MSR data provides information on the state of the CPU performance counters, analysis that requires these metrics could not

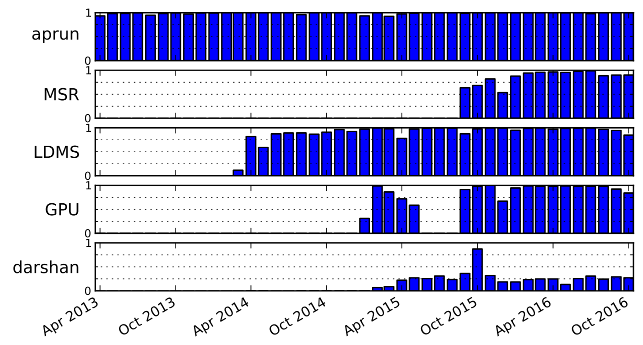

Figure 3: The ratio of node hours with data to total node hours for jobs running on Blue Waters over time.

be performed for the time period where this was not available. This includes, CPU utilization, concurrency, and operations performed.

4 DISCUSSION

The Open XDMoD software was designed to scale to large numbers of jobs and we had no difficulty ingesting the 4.6 million job records. However, it was originally only designed to process data from HPC resources with thousands of compute nodes. We encountered some scalability problems when processing the job data from Blue Waters, which had jobs with an order of magnitude more compute nodes than was previously processed. We also spent a large amount of time diagnosing and fixing issues with corrupt and invalid input data. In this section, we discuss some of the problems that we encountered in the project and how we overcame them.

4.1 Scalability problems

We encountered several issues caused by jobs with large numbers of compute nodes (>20,000 nodes). We also ran into problems when we tried to increase the size of our HPC batch jobs that performed the data conversion and summarization steps.

Some of the job-size related issues had very simple solutions. For example, some jobs were so large that the column in the database that stored the node list data from the accounting logs was not large enough to hold the data[3]. The simple fix was to change the column type to be able to store more data. The node list parsing algorithm in the `xdmod-shredder` command exhibited relatively poor performance processing long node lists. We made a small improvement to the string parsing but did not spend much developer effort on the problem, since the bulk accounting data ingest was a single run that completed overnight. Another minor issue was that job summary record size was large enough that the nodejs based ingestion process hit the maximum heap size when we ran bulk ingests of large numbers of jobs. This issue was mitigated by increasing the heap limit for the v8 engine in nodejs.

One more complex problem was seen in the job summary ingestion process (ETL process). The ETL process had an optimization where multiple insert queries were batched together, which results in a significant performance improvement especially for small queries. The algorithm batched all dimension inserts together for each job. This caused a problem for the jobs with very large numbers of nodes as the batched inserts exceeded the maximum datagram size in the MySQL driver. We changed the software to split the batch inserts into to smaller chunks so that large jobs could be inserted with minimal overhead.

Increasing the number of parallel processes in the HPC batch jobs lead to some minor problems. The summarization software created one connection to the MySQL database per process. As we scaled up the number of processes we hit the limit of the maximum allowed simultaneous connections on the database. The simple fix in this case was to increase the threshold.

There was a small subset of jobs that had many thousands of `aprun` calls and some individual jobs had over 10,000 individual job steps. This caused problems in parsing and storing the large amounts of `aprun` and Darshan data generated by these types of jobs. The first issue was due to data storage since the data generated for each job was greater than the 16 MB document size limit in the MongoDB database. Overcoming this problem would have required a re-architecting of the database so that we could store and fetch records over multiple documents. The second issue was that of processing time. The Darshan data was stored in a gzipped binary format with one file per step. Each of these files needed to be individually opened and parsed. This process quickly became I/O bound and likely would not have been able to be completed if we attempted to parse all of the job steps. In light of these issues, the number of steps that were populated with Darshan and `aprun` data was limited to 100. Note that this artificial truncation of data did not impact the workload analysis, since the jobs with large numbers of `aprun` commands were typically parameter sweeps where the same command is executed multiple times with different input data.

In addition, due to the large numbers of files being processed, we discovered and fixed several bugs in the underlying software used as part of the workflow. Very early on we found that processing thousands of files would lead to our analysis crashing due to an exhaustion of file descriptors. This was tracked down to a file descriptor leak in the underlying PCP libraries. This problem was fixed and submitted back to the PCP maintainers.

We made improvements to the speed of the aggregation queries for bulk ingest of large amounts of job performance data. The summary information displayed in Open XDMoD must be generated whenever new job data is added. There is a database query that is run to determine the time periods where the data must be regenerated. The original query was chosen to be optimal when a small number of jobs are added to the database. This is ideal for the main use case where new job data are added daily, however it has very poor performance when multiple years of data are bulk loaded. We changed the code to run a different, much faster, query when large number of jobs are bulk ingested.

A subset of the jobs had to be handled separately because of their size. We found that the jobs that ran on more than 20,000 compute nodes could not be summarized by the normal batch process because of the memory usage and the amount of time needed to read the PCP archive data from the parallel filesystem. Since there were only a few tens of jobs in this category, we chose to process them using one summarization task per compute node so that there was

[3]The unparsed node list data from the accounting log is stored in staging tables in the Open XDMoD datawarehouse before being parsed and stored in tables in normalized form.

Table 1: Data coverage estimates.

Metric	Total Jobs	Jobs Covered	% Jobs	Total Hours	Hours Covered	% Hours
aprun	4,646,659	4,410,785	95	547,809,848	543,197,000	99
Darshan	4,646,659	657,422	14	547,809,848	75,169,985	14
LDMS	4,646,659	3,304,285	71	547,809.848	382,280,591	70
MSR	4,646,659	2,032,068	44	547,809,848	153,723,022	28

sufficient memory available and to minimize I/O contention. These summarization tasks still took 90 hours to complete, much longer that the default 48 hour job time limit.

4.2 I/O issues

We also encountered scaling problems with I/O infrastructure. The PCP archive format comprises at least three files: an index file, a metadata file and one or more files that contain the metric data. I/O problems were caused by opening many files with relatively small amounts of data per file. This problem affects all implementations, but is exacerbated by the large jobs.

We modified the PCP library to reduce the number of I/O system calls needed to open an archive. This modification uses the fact that the archive layout is known because it was generated by the `ldms2pcp` script and we know the exact name of the archive to process. This optimization is not generally applicable to an arbitrary PCP archive since in general the PCP software stack tries to deal with a variety of archives names to prevent filename collisions and issues with archive size.

4.3 Data quality issues

Monitoring an HPC system is a complex problem involving multiple interacting software and hardware systems. Since it is impossible to foresee all possible contingencies when the software is originally written and there will always be occasional hardware failures, data problems abound.

One method to diagnose problems with the data is to check whether the values are physically realistic. For example, the maximum clock tick rate for the CPUs is known so any measurements that report a significantly larger value is suspect.

We observed that the accounting logs contained job records with impossible, or contradictory data. For example, there were jobs where the difference between the recorded start and end times was many months. There were also jobs with impossible start and end times (in the 1970s). The `xdmod-shredder` software was modified to log these kinds of records. The default behavior for the Torque shredder script is to use the elapsed time between the start and end timestamps as the job duration. We observed unrealistic job duration values. The software was modified so that if there was a large discrepancy between the job duration as measured by the difference between start and end and the reported job walltime then the job walltime value was used.

We also found job records in the accounting logs that were missing the account field. To handle this case, we made a configuration change to Open XDMoD to use the contents of the group field to populate the account information if the account field was not present.

The LDMS files had several issues with data corruption. Being basically a CSV format, an error that occurs while storing the data can cause many issues during later parsing. We found truncated lines, binary data in what should be string fields, and impossible timestamps in the time fields. The solution in these cases was to skip the corrupt or unparsable lines.

As part of our data validation step, we found several cases with impossible or improbable data, even though it appeared to be parsed correctly. The first of these was in the Lustre storage metrics. It was found that for a time period, the metrics as reported by the Lustre data collector were either physically impossible or did not match the data reported by the Darshan data when it was available. This was discovered to be a bug in the Lustre collection infrastructure. In these cases the Lustre data was ignored. Also, some outlier values from the Cray Gemini interconnect metrics recorded values that were physically impossible. Again, these values were discarded.

We found `aprun` records where the timestamps in the data were very different from the timestamps in the accounting data. Some of these discrepancies were explained because there was a brief period where the job scheduler software was updated and there was an overlap in the job ids between the old and new instances which meant that two different jobs had the same job id.

There were some periods where the node names changed — this data was not reliably recorded at the time, which meant that we had no easy way of determining the node type.

4.4 Miscellaneous issues

The summarization software uses PCP's `pmlogextract` command to select records from the PCP archives. The original software simply forked a subprocess to execute the command. The `mpi4py` implementation on Blue Waters does not support the `system()` call so we added the ability to launch the `pmlogextract` code without forking a sub-process.

The application identification algorithm in Open XDMoD used a simple glob-based pattern matching algorithm to classify jobs based on the executable name. On a typical academic HPC system we are able to use the executable name to categorize approximately 60% of jobs (by node hours). The majority of the uncategorized jobs are user written custom software and scripts. Whereas on Blue Waters we were able to categorize 92% of jobs (by node hours). The main reason for this high categorization ratio is that the Blue Waters staff have good knowledge of the software that their users run. We did find, however that the glob-based pattern matching was inadequate and so we made several improvements including adding regular expression support and allowing matching on the full executable path.

During the data analysis part we identified and fixed several minor bugs in the Open XDMoD software itself. For example, there was an issue with displaying the measurement units in the Job Viewer's detailed metric pane. The various fixes were incorporated into the Open XDMoD software and will be available in the next release.

A typical install of the performance collection on the compute nodes includes adding hooks in the job prologue and epilogue scripts to record metrics as well as making periodic measurements. The metric data collection on Blue Waters only recorded the periodic measurement every 60 seconds therefore very short jobs typically did not have sufficient data to run the summarization analysis. The absence of data for short jobs was not a major concern since the short job usage was negligible compared to the overall machine usage (jobs shorter than 30 seconds used approximately 70 thousand node hours out of over 5 billion).

The very large jobs toThe very large jobs took so long to process that the database connection timed out. This revealed a bug in the software in that did not attempt to reconnect after a timeout, which we fixed.

5 CONCLUSIONS

The workload analysis of Blue Waters was highly successful. It accomplished the short term goal of understanding the Blue Waters workload and the longer term goal of establishing a robust instance of Open XDMoD to monitor Blue Waters in the future. It also provided a great deal of insight into HPC monitoring on a large scale that has been the subject of this paper. The total number of jobs on Blue Waters, while large, is comparable to other HPC systems and well within the capabilities of Open XDMoD to process and present in an intuitive manner. The problems that were encountered with this project were largely due to three main issues: the requirement to process a multi-year backlog of data in a short space of time, missing and erroneous source data, and a large number of compute nodes and the subsequent size of the jobs.

The performance data from Blue Waters was in a format that was not supported by Open XDMoD. However, the Open XDMoD data processing infrastructure was designed to be able to handle multiple different data formats and data schemas. We were able easily to modify the Open XDMoD configuration to handle the different input data. The ability to ingest, analyze and display: LDMS, `aprun`, and Darshan data added new functionality to XDMoD and is a positive outcome for the XDMoD infrastructure in general. We expect that other users of Open XDMoD will be able to make use of this functionality. In addition, many of the bug fixes and new features that were added during the project will be incorporated into the next Open XDMoD release. This includes the enhancements to the application identification algorithm as well as the various performance improvements.

Finding errors in the source data is expected anytime you are trying to analyze an extremely large number of metrics collected by a diverse software stack. Dealing with these errors adds robustness to XDMoD and also provides system administrators information on better data collection policies.

Finally, while Open XDMoD has proven to be well suited to handle a large number of jobs to analyze, this project was a good test of the infrastructure and lead to many gains in demonstrating and improving the scalability of the whole system. By leveraging parallel processing in the summarization stage and implementing algorithmic improvements in the internal pipeline, we were able to increase the efficiency of the end to end process.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation under grant number ACI 1650758 (Workload Analysis of Blue Waters) and ACI 1445806 (XD Net Metrics Service for High Performance Computing).

REFERENCES

[1] Anthony Agelastos, Benjamin Allan, Jim Brandt, Paul Cassella, Jeremy Enos, Joshi Fullop, Ann Gentile, Steve Monk, Nichamon Naksinehaboon, Jeff Ogden, Mahesh Rajan, Michael Showerman, Joel Stevenson, Narate Taerat, and Tom Tucker. 2014. The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '14)*. IEEE Press, Piscataway, NJ, USA, 154–165. DOI: http://dx.doi.org/10.1109/SC.2014.18

[2] Kapil Agrawal, Mark R. Fahey, Robert McLay, and Doug James. 2014. User Environment Tracking and Problem Detection with XALT. In *Proceedings of the First International Workshop on HPC User Support Tools (HUST '14)*. IEEE Press, Piscataway, NJ, USA, 32–40. DOI: http://dx.doi.org/10.1109/HUST.2014.6

[3] Andrew Barry. 2013. Resource Utilization Reporting. In *Proceedings of the Cray User Group conference*. Cray User Group, Inc., Oak Ridge, TN. https://cug.org/proceedings/cug2013_proceedings/includes/files/pap103.pdf

[4] Brett Bode, Michelle Butler, Thom Dunning, Torsten Hoefler, William Kramer, William Gropp, and Wen-mei Hwu. 2013. The Blue Waters Super-System for Super-Science. In *Contemporary High Performance Computing*. Chapman and Hall/CRC, Boca Raton, FL, USA, 339–366. DOI: http://dx.doi.org/10.1201/b14677-16

[5] Phillip Carns, Robert Latham, Robert Ross, Kamil Iskra, Samuel Lang, and Katherine Riley. 2009. 24/7 Characterization of petascale I/O workloads. In *2009 IEEE International Conference on Cluster Computing and Workshops*. IEEE, Piscataway, NJ, USA, 1–10. DOI: http://dx.doi.org/10.1109/CLUSTR.2009.5289150

[6] University at Buffalo Center for Computational Research. 2017. XD Metrics on Demand (XDMoD). https://xdmod.ccr.buffalo.edu. (2017).

[7] Todd Evans, William L. Barth, James C. Browne, Robert L. DeLeon, Thomas R. Furlani, Steven M. Gallo, Matthew D. Jones, and Abani K. Patra. 2014. Comprehensive Resource Use Monitoring for HPC Systems with TACC_Stats. In *Proceedings of the First International Workshop on HPC User Support Tools (HUST '14)*. IEEE Press, Piscataway, NJ, USA, 13–21. DOI: http://dx.doi.org/10.1109/HUST.2014.7

[8] Matthew D. Jones Jones, Joseph P. White, Martins Innus, Robert L. DeLeon, Nikolay Simakov, Jeffrey T. Palmer, Steven M. Gallo, Thomas R. Furlani, Michael Showerman, Robert Brunner, Andriy Kot, Gregory Bauer, Brett Bode, Jeremy Enos, and William Kramer. 2017. Workload Analysis of Blue Waters. *ArXiv e-prints* (March 2017). arXiv:1703.00924

[9] Matthew L Massie, Brent N Chun, and David E Culler. 2004. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Comput.* 30, 7 (2004), 817 – 840. DOI: http://dx.doi.org/10.1016/j.parco.2004.04.001

[10] Jeffrey T. Palmer, Steven M. Gallo, Thomas R. Furlani, Matthew D. Jones, Robert L. DeLeon, Joseph P. White, Nikolay Simakov, Abani K. Patra, Jeanette M. Sperhac, Thomas Yearke, Ryan Rathsam, Martins Innus, Cynthia D. Cornelius, James C. Browne, William L. Barth, and Richard T. Evans. 2015. Open XDMoD: A tool for the comprehensive management of high-performance computing resources. *Computing in Science and Engineering* 17, 4 (2015), 52–62. DOI: http://dx.doi.org/10.1109/MCSE.2015.68

[11] Michael Showerman, Jeremy Enos, Joseph Fullop, Paul Cassella, Nichamon Naksinehaboon, Narate Taerat, Thomas Tucker, James Brandt, Ann Gentile, and Benjamin Allan. 2014. Large Scale System Monitoring and Analysis on Blue Waters using OVIS. In *Proceedings of the Cray User Group conference*. Cray User Group, Inc., Oak Ridge, TN. https://cug.org/proceedings/cug2014_proceedings/includes/files/pap156.pdf