Parallel I/O Characterisation Based on Server-Side Performance Counters

Salem El Sayed*, Matthias Bolten†, Dirk Pleiter* and Wolfgang Frings*
*Jülich Supercomputing Centre, Forschungszentrum Jülich, 52425 Jülich, Germany
†Institut für Mathematik, Universität Kassel, 34132 Kassel, Germany

***Abstract*—Provisioning of high I/O capabilities for high-end HPC architectures is generally considered a challenge. A good understanding of the characteristics of the utilisation of modern I/O systems can help address the increasing performance gap between I/O and computation. In this paper we present results from an analysis of server-side performance counters that had been collected for multiple years on a parallel file system attached to a peta-scale Blue Gene/P system. We developed a set of general performance characterisation metrics, which we applied to this large dataset.**

I. Introduction

As high-end high-performance computing is advancing from petascale to exascale, there are various challenges which apply in particular to the capabilities of the I/O sub-system. For instance, the US Department of Energy lists maintaining balanced systems as one of the major hurdles [1] . One such balance concerns the ratio of I/O bandwidth versus instruction throughput [2]. Another hurdle is the growing need for coping with run-time errors, e.g. by means of check-point/restart. This results in growing demands for I/O operations.

While there are technological options available to grow performance of the storage sub-system at a similar rate as, e.g., compute capabilities grow, affordability of the resulting architectures becomes a problem. Among the strategies for mitigating this problem, the use of hierarchical storage architectures is one of the most promising. Within such a hierarchy, fast I/O layers with limited storage capacity are introduced, which are based on expensive technologies like high-endurance non-volatile memories. High-capacity layers will continue to be based on more traditional technologies like hard drives. Trading costs for complexity and the architectural decisions to be made in the context of such hierarchical architectures do, however, require a good understanding of the characteristics of the I/O load for current high-end HPC systems.

Server-side performance counters enable collection of information that allow to characterise the I/O load. Characterisation involves the application of a set of criteria, which we present in this paper. Using server-side performance counters has the advantage that the amount of created data does not become prohibitively large even in case of big systems comprising hundreds of I/O servers and of long-term monitoring covering a multi-year lifetime of such a system. On the downside, monitoring data is collected at file system level or below, where some information available at application level is lost or difficult to access.

We apply our approach to collecting server-side performance counters and evaluating this data using a set of characterisation criteria to a petascale Blue Gene/P system, which was installed at Jülich Supercomputing Centre from 2007 to 2012. For this system GPFS was used as a parallel file system and server-side performance counters were collected on the Blue Gene/P I/O forwarding nodes.

This article makes the following contributions:

1) We present an approach to monitor the I/O workload of a massively-parallel HPC system by means of server-side performance counters.
2) To evaluate this performance data we introduce a set of general work-load characterisation metrics.
3) This approach is applied to analyse the performance counters that had been collected on a petascale Blue Gene/P system over a period of approximately 19 months.

After summarizing related work in section II we present our methodology in section III. In section IV we present a selection of criteria, which we developed to characterise the I/O load. Before we present our results from applying this methodology in section VI we provide background information on the system, to which we applied our methodology, in section V. We end by drawing our conclusions in section VII.

II. Related Work

The observation that compute and communication performance is increasing rapidly, while I/O performance remains performance limited, triggered early attempts on I/O performance characterisation. Miller et al. [3] performed in the early 90s an analysis of traces of a Cray Y-MP system for a variety of user codes. They were to our knowledge the first to report on bursty I/O behaviour. Reed published together with different collaborators several papers related to I/O workload characterisation [4], [5], [6]. Their focus was on analysing the characteristics of specific applications, while in this paper an approach to characterise the load for a system is attempted, which is used by many different applications over a long period of time. The closest we found to this approach is the Charisma project [7], [8], [9], which focused on file access patterns with the target to improve file system performance.

Due to the growing interest in analysing the I/O behaviour, different frameworks have been developed. Most of these frameworks have in common that they are performed at client-side by intercepting user calls of I/O functions. Already Miller

et al. [3] used this approach and modified standard system libraries. The problem of aggregating data efficiently while minimizing performance impact was solved by forwarding trace information in bunches via available system monitoring channels. In [4] an extension of the performance environment Pablo is presented, which allows to capture the parameters of application I/O calls. Pablo included the feature of performing different types of in-situ data reductions to reduce the amount of generated tracing data. Also the currently most popular I/O characterisation tool Darshan [10] is based on intercepting user calls. In case of MPI I/O the PMPI profiling interface is used, while calls to POSIX routines are manipulated at linker level.

Little has been published based on server-side performance monitoring only, exploiting for example the monitoring capabilities of parallel file systems like GPFS or Lustre (for the latter see, e.g., [11]). Yang Liu et al. [12], use server-side traces to isolate the I/O signature of applications. A novel approach was taken by the SIOX project [13] as it allows to cover parallel I/O both, on client and server side. SIOX is not only designed for performance monitoring but also for automatic steering of the I/O system based on performance data.

III. Methodology

In the following we assume an I/O sub-system, which allows to regularly log at least the following 6 performance counters: amount of data read and written, the number of read and write operations, the number of file open and close operations. Parallel file systems like GPFS provide the required feature. We denote the logging interval with Δt. On the one hand, Δt must be small compared to typical job execution times and, on the other hand, sufficiently large such that the amount of performance data remains manageable even if the overall monitoring period becomes long, e.g. several years, and the number of monitored I/O servers is large. Minimizing the amount of monitoring data also reduces the risk of adverse performance impacts due to system monitoring.

After completing the raw data taking, this data needs to be pre-processed for different reasons. Part of this pre-preprocessing is specific to the considered I/O architecture. Examples are the possible need of coping with lost data sets or the reset of performance counters at different points in time. Here we only report on one pre-processing step, which is inherently required for our approach. The reason is that we cannot assume the logging of the I/O servers to be synchronised. While all servers are expected to write-out performance counters periodically in the same periodic intervals Δt, these will happen at different points of time. We here assume a relatively small clock drift between I/O servers compared to the time interval Δt, for example the I/O nodes of the Blue Gene/P system employ the Network Time Protocol (NTP) for synchronisation [14]. Let us consider an observable v, which is measured on a given server at discrete points in time $t_i = t_0 + i\,\Delta t$, i.e. $v_i = v(t = t_i)$. The start time t_0 is possibly different for all considered servers. We now introduce an artificial time grid with a granularity $\Delta\tilde{t} \ll \Delta t$ as well as a universal start time $\tilde{t}_0$, which is chosen to be the same for all servers. We then determine the performance counter values for this new time grid by linear interpolation:

$$\tilde{v}_k = v_i + \frac{(\tilde{t}_0 + k\Delta\tilde{t}) - (t_0 + i\Delta t)}{\Delta t}\,(v_{i+1} - v_i)\,, \qquad (1)$$

where $(t_0 + i\Delta t) \le (\tilde{t}_0 + k\Delta\tilde{t}) \le [t_0 + (i+1)\Delta t]$.

A next step concerns linking of I/O performance data and job information. The goal is to be able to reconstruct, e.g., the total amount of data read or written by a specific job. This may involve another interpolation step to determine the performance counter values at the time the job started and ended. We solved this technical problem by choosing $\Delta\tilde{t}$ sufficiently small (Here we choose $\Delta\tilde{t} = 1\,\mathrm{s}$). For some systems it may not be possible to perform this step as multiple jobs might access the same server and thus prevent linking this data.

The list of jobs that is considered also requires filtering. We decided to consider only jobs that ran for at least 1 hour, as a short job residence time likely indicates erroneous behaviour. Furthermore, extended job duration means more performance counter logging intervals, which makes application of some characterisation metrics more reliable.

Once the raw data has been pre-processed and filtered a set of characterisation criteria can be applied to this data. We present details on selected criteria in the next section.

As a final step, it is recommended to foresee a validation of the framework used for collecting performance counters, pre-processing the raw data, linking I/O performance data and job data as well as applying the characterisation criteria. For this purpose we implemented a micro-benchmark performing sequential write and read operations. This was kept simple on purpose in order to create a load with known characteristics and thus allows for comparison of expected and obtained evaluation results.

IV. Characterisation criteria

In this section we present a selected subset of the I/O characterisation criteria [15]. Let us start by introducing a set of basic quantities, which we use for our formalism. We are interested in performing a characterisation on a per job basis, where the job starts and ends at time t_{start} and t_{end}, respectively. The quantities

$$D_r(l, s, t) \quad \text{and} \quad D_w(l, s, t)\,, \qquad (2)$$

denote the number of read or write operations of length l in bytes that arrive at server s during the time interval $[t_{\mathrm{start}}, t]$ (with $t_{\mathrm{start}} \le t \le t_{\mathrm{end}}$). In the following we use $x = \{r, w\}$ with r or w changing the quantity to read or write respectively.

Below we will make use of the helper quantity

$$\delta(s, t, \Delta t) = \begin{cases} 1 & \text{if } \sum_l l\,[D_x(l, s, t + \Delta t) - D_x(l, s, t)] > c\,, \\ 0 & \text{otherwise}\,, \end{cases} \qquad (3)$$

where $x = \{r, w\}$ and $c \ge 0$ is a threshold parameter.

Using this definition, the helper quantity $\delta(s,t,\Delta t)$ has the value 1 if more than c bytes were moved during the time interval $[t, t+\Delta t]$ and 0 otherwise.

a) Aggregate I/O volumes:: Using the quantities introduced in Eq. (2) we define the aggregate amount of data read or written by a job as

$$N_x = \sum_l \sum_{s \in S} l\, D_x(l, s, t_{\mathrm{end}})\,, \tag{4}$$

where $x = \{r, w\}$ and S the set of I/O servers used by the given job. Here we assume that an I/O server is exclusively used by a single job, which is the case for the I/O forwarding nodes on Blue Gene/P.

b) Bandwidth:: As we perform only a relatively coarse grained collection of performance counters, we can only determine a bandwidth averaged over the sampling time interval Δt as follows:

$$B_x(s,t) = \frac{1}{\Delta t} \sum_l l\, [D_x(l, s, t+\Delta t) - D_x(l, s, t)] \tag{5}$$

with $x = \{r, w\}$. Note that the maximum bandwidth exploited by an application over a shorter period of time might be much higher.

c) Read/Write IOPS:: The throughput of I/O operations can be defined in a very similar way as the bandwidth:

$$\Gamma_x(s,t) = \frac{1}{\Delta t} \sum_l [D_x(l, s, t+\Delta t) - D_x(l, s, t)] \tag{6}$$

d) I/O intensity:: Another way to characterise the I/O behaviour of applications is to measure the number of time intervals during which I/O is performed and put this into relation to the total number of time intervals. Let us introduce the quantity $H(t, \Delta t) = 1$ if any $\delta(s, t, \Delta t) > 0$, otherwise $H(t, \Delta t) = 0$. This quantity indicates whether during a time interval $[t, t+\Delta t]$ a significant number of I/O operations have been executed by any of the servers. Significant means that the I/O volume exceeds the threshold parameter c as defined in Eq. (3). We now define the I/O intensity I of a job as

$$I = \frac{\Delta t \sum_{i=0}^{n} H(t_i, \Delta t)}{t_{\mathrm{end}} - t_{\mathrm{start}}}\,, \tag{7}$$

where $t_i = t_{\mathrm{start}} + i\Delta t$ and $t_{\mathrm{start}} \le t_i \le t_{\mathrm{end}}$ for $i = 0, \ldots n$. The I/O intensity is defined such that $0 \le I \le 1$. $I = 1$ indicates that the application is performing continuous read or write operations above threshold.

e) Burstiness parameter:: The bursty I/O behaviour of applications running on HPC systems has been reported by several authors [16]. However, to our knowledge, no quantitative metrics for measuring the extend of the I/O bursty behaviour have been reported so far. We propose a burstiness parameter ρ which is defined as

$$\rho = \begin{cases} 1 - \tanh(l_{\mathrm{IO}}/l_{\mathrm{noIO}}) & \text{if } l_{\mathrm{noIO}} > 0\,, \\ 0 & \text{otherwise}\,, \end{cases} \tag{8}$$

where l_{IO} and l_{noIO} is the average number of consecutive time intervals Δt for which $H = 1$ and $H = 0$, respectively. Like in case of I/O intensity, using $tanh$ we defined the burstiness parameter to be bound to the interval $[0, 1]$. If a short period of I/O, i.e. l_{IO} is small, is followed by a long period without I/O, i.e. l_{noIO} becomes large, then we expect ρ to be close to 1.

f) Parallel I/O intensity:: Finally, we introduce a metric that quantifies the level of parallelism during I/O. Let us first introduce the quantity $\pi(t, \Delta t) = (\sum_s \delta(s, t, \Delta t))/|S|$, where $|S|$ is the number of I/O servers involved. If for a given time interval all these servers read or write data beyond threshold then $\pi = 1$. We define the unnormalised parallel intensity as $\Pi = [\sum_i \pi(t_{\mathrm{start}} + i\Delta t, \Delta t)] \,/\, [\sum_i \delta(t_{\mathrm{start}} + i\Delta t, \Delta t)]$ and the normalised parallel I/O intensity as

$$P = \frac{|S|\,\Pi - 1}{|S| - 1}\,. \tag{9}$$

If, whenever I/O beyond threshold is performed, always all servers are involved then $P = 1$. In the other extreme case where only a single server is used for all I/O, we have $P = 0$. The parallel I/O here refers to real time parallelism.

V. I/O Sub-system Background

We applied our methodology to a large-scale Blue Gene/P called JUGENE, which was installed at Jülich Supercomputing Centre [17]. This system, which was available for users from 2007 to 2012, comprised 72 racks with 1024 compute nodes each. The compute nodes were connected via the global collective network, which had a tree topology, to I/O nodes. The compute node to I/O node ratio was chosen to be 128:1, i.e. 8 I/O nodes per rack, except for one rack where 32 I/O nodes were installed. The total number of I/O nodes was therefore 600. The 10GE interface of each of the I/O nodes as well as the external storage servers were all connected to a central Ethernet switch fabric. For an overview on the I/O relevant part of the architecture, see Fig. 1.

While the compute nodes were operated with a lightweight kernel called Compute Node Kernel (CNK), the I/O nodes were operated with Linux. The parallel file system used for accessing the external storage servers was mounted on the I/O nodes only. A function shipping mechanism was used to forward any I/O request of the compute to the I/O nodes.

JUGENE was connected to a parallel storage service. Under benchmarking conditions a performance of up to 66 GByte/s has been observed [18], which we consider the maximum measured performance of the system.

As a parallel file system IBM's General Parallel File System (GPFS) was used [19]. It was setup using Network Shared Disk (NSD) servers running on the central storage servers, to which a large number of disks were attached. This central storage is accessed by the NSD clients running on the I/O nodes. A high bandwidth is achieved for large, contiguous I/O by striping across many NSD servers, i.e. across many disks, storage servers, and network links. Highest bandwidth was observed using a file system block size of 4 MiByte [18].

GPFS features monitoring capabilities that allows to collect statistics by each NSD client as shown in Fig. 1. These

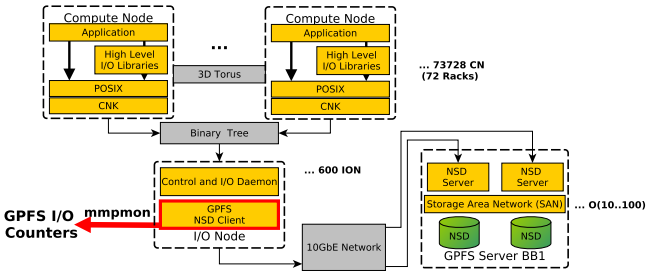

Fig. 1. Location of GPFS I/O counters in JUGENE I/O stack, adapted from [18].

counters have been written by each of the 600 I/O nodes in regular intervals $\Delta t = 120\,\mathrm{s}$ to disk. The number of counters was kept to a minimum. In Table I we list all used counters and how they relate to the observables defined in the previous section.

TABLE I
GPFS PERFORMANCE COUNTERS USED FOR THIS WORK.

Counter	Description	Observable
_br_	Total number of bytes read	$\sum_l l\, D_r(l,s,t)$
_bw_	Total number of bytes written	$\sum_l l\, D_w(l,s,t)$
rdc	Number of read requests	$\sum_l D_r(l,s,t)$
_wc_	Number of write requests	$\sum_l D_w(l,s,t)$
_oc_	Count of `open()` call requests	$F_{\mathrm{open}}(s,t)$
_cc_	Number of `close()` call requests	$F_{\mathrm{close}}(s,t)$

VI. Selected Results

In the following we report on a subset of I/O characterisation metrics, which we applied to about 0.17 million jobs that had ran for more than an hour on the Blue Gene/P system described in the previous section. The complete analysis of all I/O criteria and more detailed descriptions can be found in [15].

a) Aggregate I/O volumes:
The analysed jobs read a total of 14.7 PiByte and wrote a total of 11.7 PiByte. In Fig. 2 and Fig. 3 we offer different statistical views on the data.

In the central area (a) we provide a scatter plot, which allows to identify possible relations between the amount of read or written data and maximum read or write bandwidth. It is plotted as a heat map to take also the number of jobs into account which did read or write a very similar amount of data, while having a similar maximum read or write bandwidth.

We furthermore plotted histograms that show the probability of a particular amount of data being read (N_r) Fig. 2-(b) or written (N_w) Fig. 3-(b) by a job. As histograms may suffer from artefacts due to the need of data binning, we also plotted the cumulative distribution function. From these plots one can derive the maximum number of Bytes read by a single job, which is 109.5 TiByte. Interestingly, most of jobs read a relatively small amount of data, as 80% of the analysed jobs read only 12.7 GiByte or less. Almost all data is read by a relatively small number of jobs: 20% of analysed jobs are responsible for 97.6% of the total read volume. The maximum amount of data written by a single job is smaller, namely 22.3 TiByte. Also in case of writing we observe that most of the jobs perform a relatively small amount of I/O: 80% of the jobs write only 15.3 GiByte or less. Similarly to read, write has 20% of analysed jobs responsible for 97.7% of the total written volume.

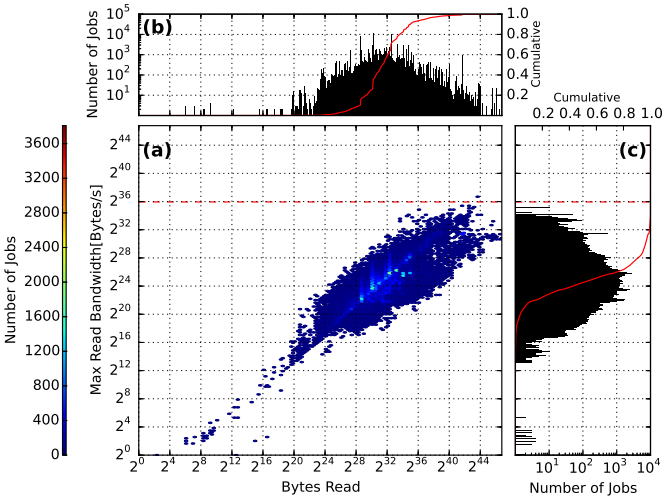

Fig. 2. Aggregate I/O volumes against maximum bandwidth of read for analysed jobs: (a) Scatter plot, (b) distribution of bytes read and (c) distribution of maximum read bandwidth.

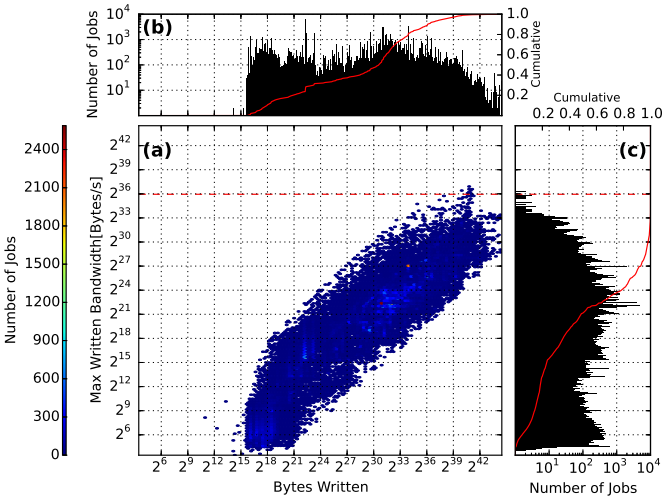

Fig. 3. Aggregate I/O volumes against maximum bandwidth of write for analysed jobs: (a) Scatter plot, (b) distribution of bytes written and (c) distribution of maximum write bandwidth.

b) Maximum bandwidth:
Here we focus on the maximum average bandwidth at which data was read or written by a job. Bandwidth is always averaged over the performance counter sampling period Δt. The statistical analysis is shown in Fig. 2 and Fig. 3 in relation to aggregate I/O volume, where we indicated the maximum measured performance of the system by red dashed lines.

For read a single job has exceeded the maximum bandwidth observed in benchmarks using 73728 compute nodes and 576 I/O nodes, while for write it was exceeded by 10 jobs using either 4096 or 8192 and 32 or 64 I/O nodes. A possible reason for this particular high bandwidth is buffering on the I/O nodes, which is more likely for write than for read.

In both Fig. 2 and Fig. 3 it is observed that the bandwidth and aggregate I/O volume distributions appear similar, which indicates that jobs performing more I/O utilize more of the available bandwidth. This can be observed in the heat maps (Fig. 2-(a) and Fig. 3-(a)) which show a clustering behaviour of jobs on the diagonal.

Most jobs experienced a low maximum bandwidth, with 80% of jobs staying below 84 MiByte/s for read and 19 MiByte/s for write. This is likely not due to bad I/O performance, but rather due to the fact that most jobs (see above) read or write only relatively small amounts of data. Even if these jobs would read or write 15 GiByte of data at maximum speed during a sampling interval of $\Delta t = 120\,\mathrm{s}$ the maximum observed bandwidth would be 128 MiByte/s. The read bandwidth appears to be slightly higher than write.

c) Maximum I/O operation throughput:
Next let us consider the throughput of I/O operations. We observe both, maximum read and write I/O Operations Per Second (IOPS) per job averaged over the sampling period Δt to be fairly low. While the highest maximum read IOPS is around 1.2×10^6, 80% of jobs did not exceed a read IOPS of 1.3×10^3. For write operations we observe a slightly lower maximum throughput, with the maximum being 7.3×10^5 and 80% of the jobs not even exceeding 6.0×10^1.

d) I/O intensity and burstiness:
Tab. II shows the maximum I/O intensity (I) of 80% of analysed jobs for read and write using different threshold values. Although the I/O intensity appears high for write at 0 Byte threshold, it drops as expected when the threshold is increased. Meanwhile, read has a relatively low I/O intensity even at 0 Byte, with 80% of the jobs having an I/O intensity below 0.28. As the threshold increases both read and write settle to lower I/O intensities. Nonetheless, a few jobs still achieve a relatively high I/O intensity for read, write or both even at a threshold of 1 MiByte.

TABLE II
80% OF ANALYSED JOBS ARE EQUAL OR BELOW THESE VALUES FOR I/O INTENSITY, BURSTINESS AND PARALLEL I/O INTENSITY, USING VARIOUS THRESHOLDS c.

Threshold c	0 Byte r / w	128 KiByte r / w	1 MiByte r / w
I/O intensity (I)	0.28 / 1.0	0.15 / 0.34	0.05 / 0.12
Burstiness (ρ)	0.99 / 0.0	0.99 / 1.0	1.0 / 1.0
Parallel I/O intensity (P)	0.91 / 1.0	0.88 / 0.28	0.84 / 0.27

Additionally, Tab. II shows the maximum burstiness (ρ) for 80% of the analysed jobs for read and write using different threshold (c) values. As observed, for 0 Byte threshold write exhibits almost no burstiness, but then increases to almost 1.0 when the threshold is increased. Meanwhile, read exhibits a high burstiness for most of the jobs even at 0 Byte threshold. Nonetheless, at 1 MiByte threshold some jobs exhibit a low burstiness for either read, write or both.

The analysis results seem to indicate that most jobs have a low I/O intensity and exhibit high burstiness.

e) Parallel I/O intensity:
Tab. II shows the maximum of read and write parallel I/O intensity (P) for 80% of the analysed jobs. As seen, write has a high parallel I/O intensity only for 0 Byte threshold and decreases to 0.27 or below for 80% of jobs. In comparison, read has a relatively high parallel I/O intensity which decreases very little when increasing the threshold, where 80% of jobs are below 0.84 for 1 MiByte threshold.

While parallel I/O intensity is higher for read than for write, the reverse is true for I/O intensity. This indicates that while write occurs more frequently, concurrent read is more common among analysed jobs. However, we suspect that many applications perform task local read, that occurs on multiple I/O nodes concurrently. Meanwhile, both read and write exhibit a high burstiness.

VII. Summary and Conclusions

Server-side I/O performance counters enable monitoring of the I/O subsystem of large-scale HPC systems without affecting its performance. To characterise the system's I/O load we applied a small set of criteria to I/O performance statistics collected over 19 months. This allowed us to evaluate about 0.17 million jobs. The I/O behaviour exhibited by these jobs is observed to vary widely. Our main observations are:

- Given the overall capacity of the storage system supporting JUGENE, the transient data hitting the external storage system was indicated by the analysis to be relatively small.
- Most jobs were found to have a low I/O intensity.
- Small I/O requests are found to dominate the I/O, more so for write than for read.
- The analysis demonstrated the bursty I/O behaviour already observed in earlier efforts of characterising I/O behaviour.

The methodology once implemented and established allows for regular monitoring of an I/O subsystem with relatively low overhead, while the generated information and statistics can be extremely valuable for maintaining and improving the system as well as for designing and preparing for future I/O architectures. Using the collected dataset and the conclusions drawn here we have already began to investigate changes to existing I/O architectures and their impact on the I/O performance [20].

GPFS performance counter monitoring has meanwhile been enabled for all large-scale systems at Jülich Supercomputing Centre and has been integrated into the system monitoring tool LLview [21]. We plan to also integrate the application characterisation metrics presented in this paper.

VIII. Acknowledgements

This work has been performed in the context of the Exascale Innovation Center at Jülich Supercomputing Center. It was partially funded by the state of North Rhine-Westfalia ("Anschubfinanzierung zum Aufbau des Exascale Innovation Center (EIC)"). We acknowledge furthermore the help of Michael Hennecke and Willi Homberg for collecting and managing JUGENE's I/O performance data.

References

[1] Department of Energy, "Department of Energy exascale strategy," 2013.

[2] J. Kunkel, M. Kuhn, and T. Ludwig, "Exascale storage systems – an analytical study of expenses," *Supercomputing frontiers and innovations*, vol. 1, no. 1, 2014.

[3] E. L. Miller and R. H. Katz, "Input/output behavior of supercomputing applications," in *Proceedings of the 1991 ACM/IEEE Conference on Supercomputing*, ser. Supercomputing '91. New York, NY, USA: ACM, 1991, pp. 567–576. [Online]. Available: http://doi.acm.org/10.1145/125826.126133

[4] P. Crandall, R. Aydt, A. Chien, and D. Reed, "Input/output characteristics of scalable parallel applications," in *Supercomputing, 1995. Proceedings of the IEEE/ACM SC95 Conference*, 1995, pp. 59–59.

[5] E. Smirni and D. A. Reed, *Computer Performance Evaluation Modelling Techniques and Tools: 9th International Conference St. Malo, France, June 3–6, 1997 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, ch. Workload characterization of input/output intensive parallel applications, pp. 169–180.

[6] E. Smirni and D. Reed, "Lessons from characterizing the input/output behavior of parallel scientific applications," *Performance Evaluation*, vol. 33, no. 1, pp. 27 – 44, 1998, tools for Performance Evaluation.

[7] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. Ellis, and M. Best, "File-access characteristics of parallel scientific workloads," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 7, no. 10, pp. 1075–1089, October 1996.

[8] A. Purakayastha, C. Ellis, D. Kotz, N. Nieuwejaar, and M. Best, "Characterizing parallel file-access patterns on a large-scale multiprocessor," in *Parallel Processing Symposium, 1995. Proceedings., 9th International*, April 1995, pp. 165–172.

[9] D. Kotz and N. Nieuwejaar, "File-system workload on a scientific multiprocessor," *Parallel Distributed Technology: Systems Applications, IEEE*, vol. 3, no. 1, pp. 51–60, Spring 1995.

[10] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley, "24/7 characterization of petascale i/o workloads," in *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, Aug 2009, pp. 1–10.

[11] A. Uselton and D. Ushizima, "Poster: I/O workload analysis with server-side data collection," in *Proceedings of the 2011 Companion on High Performance Computing Networking, Storage and Analysis Companion*, ser. SC '11 Companion. New York, NY, USA: ACM, 2011, pp. 33–34.

[12] Y. Liu, R. Gunasekaran, X. Ma, and S. S. Vazhkudai, "Automatic identification of application i/o signatures from noisy server-side traces," in *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 14)*. Santa Clara, CA: USENIX, 2014, pp. 213–228. [Online]. Available: https://www.usenix.org/conference/fast14/technical-sessions/presentation/liu

[13] J. M. Kunkel, M. Zimmer, N. Hübbe, A. Aguilera, H. Mickler, X. Wang, A. Chut, T. Bönisch, J. Lüttgau, R. Michel, and J. Weging, "The SIOX architecture - coupling automatic monitoring and optimization of parallel I/O," in *Supercomputing*, ser. Lecture Notes in Computer Science, J. Kunkel, T. Ludwig, and H. Meuer, Eds. Springer International Publishing, 2014, vol. 8488, pp. 245–260.

[14] C. Sosa and B. Knudson, "IBM system Blue Gene solution: Blue Gene/P application development," Aug 2009, redbook SG24-7287-03. [Online]. Available: https://www.redbooks.ibm.com/redbooks/pdfs/sg247287.pdf

[15] S. El Sayed, "Analysis of i/o requirements of scientific applications," Ph.D. dissertation, Bergische Universität Wuppertal, 2015, to be published.

[16] N. Liu, J. Cope, P. Carns, C. Carothers, R. Ross, G. Grider, A. Crume, and C. Maltzahn, "On the role of burst buffers in leadership-class storage systems," in *Mass Storage Systems and Technologies (MSST), 2012 IEEE 28th Symposium on*, April 2012, pp. 1–11.

[17] N. Attig *et al.*, "Contemporary high performance computing: From petascale toward exascale." Chapman & Hall/CRC, 2013, ch. Blue Gene/P: JUGENE.

[18] W. Frings and M. Hennecke, "A system level view of petascale i/o on ibm blue gene/p," *Computer Science - Research and Development*, vol. 26, no. 3-4, pp. 275–283, 2011.

[19] D. Quintero *et al.*, "IBM Spectrum Scale (formerly GPFS)," 2015. [Online]. Available: http://www.redbooks.ibm.com/Redbooks.nsf/RedbookAbstracts/sg248254.html

[20] S. El Sayed, M. Bolten, and D. Pleiter, "Parallel i/o architecture modelling based on file system counters," to appear in High Performance Computing, LNCS, vol. 9945. Springer, 2016, ISC High Performance 2016 International Workshops ExaComm, E-MuCoCoS, HPC-IODC, IXPUG, IWOPH, P3MA, VHPC, WOPSSS Frankfurt, Germany.

[21] Jülich Research Centre. (2015, Dec.) Llview: Graphical monitoring of batch system controlled cluster. Visited on Sept. 2016. [Online]. Available: http://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/LLview/_node.html