Vol.2013-ARC-207 No.11
Vol.2013-HPC-142 No.11
2013/12/16

Environment Matters: How Competition for I/O among Applications Degrades their Performance

Chih-Song Kuo[1,2,a)] Akihiro Nomura[1,b)] Satoshi Matsuoka[1,c)] Aamer Shah[2,3,d)]
Felix Wolf[2,3,e)] Ilya Zhukov[4,f)]

Abstract: In modern supercomputer architectures, thousands of compute nodes can share a dedicated file I/O server. Recent research has shown I/O can become the performance bottleneck due to large number of simultaneous I/O accesses, in particular, file writes. While methods to tackle I/O performance issues for parallel applications have been extensively studied, not much is known about how co-located I/O-intensive applications interfere each other's performance. In this study, we present a novel approach that uses a lightweight profiling tool to capture concurrent applications' competing behaviors. By comparing all application I/O performance profiles during their overlapped runtime, we are able to figure out the underlying reason to unbalanced write throughput when applications of different sizes run side by side, which is then proved to be useful in building performance models in a concurrent execution environment.

1. Introduction

The steep ratio between computing capability and storage bandwidth in modern HPC clusters has caused a great challenge of achieving satisfactory I/O performance to computational science application designers, and this trend has been ever becoming more severe due to storage resource sharing by rapid-increasing number of nodes. In response to such challenges, extensive research has been made in characterizing and improving overall I/O performance of parallel programs [1][2][3][4]. Despite the successfulness of these studies, they implicitly ignore possible external interferences occured within the same execution environment, although in fact it is very common to have several parallel applications co-located in modern supercomputers and sharing the same file server. In this setting, when two or more applications generate intensive file read/write traffic simultaneously, they are competing for the limited I/O bandwidth. Recent research [5][6] has shown that competition of two concurrent parallel applications can almost halve the I/O performance in contrast to a standalone execution, indicating the severity of application interference in file accesses.

Existing tools for analyzing I/O performance either adopt a tracing approach, or the profiling approach [1]. While the former allows for a thorough characterization, it usually hinders an application by introducing perceptible overhead particularly at extreme scale. On the contrary, the latter approach is able to capture application performance more silently in that it generates less data at runtime, while at the same time turns the possibility of fine-grained analysis aside. In our previous research [5], we leveraged the advantages from both approaches and developed a profiling tool called Light Weight Monitoring Module (LWM^2). With a mean overhead of less than 1%, LWM^2 logs predefined performance counters for each globally synchronized 4-second intervals, thus making reconciling time slice profiles among concurrent-executing applications possible.

This study was done in a small-sized 18-node cluster with an NFS server acting the I/O node, simulating a popular configuration in industry and numerous research labs. The experiments are now planned at world leading scale supercomputing facilities like TSUBAME 2.5 utilizing advanced parallel file server such as Lustre FS. As for the application, Intel MPI Benchmark for IO (IMB-IO) [7] is chosen due to its transparent behavior.

We execute a dual of applications with the same I/O write volume yet possibly different number of processes concurrently. Our result shows running two IMB-IO at the same time can degrade write throughput up to 65%. When two unequal-sized applications are executed side by side, we identified significant imbalance in both runtime and write throughput. Further analysis targeting I/O performance counters during the overlapped execution time indicates the root cause to be a uniform distribution of I/O network bandwidth among all processes. Based on the observations, we present a formula relating standalone runtime to con-

[1] GSIC, Tokyo Institute of Technology, Meguro, Tokyo 152–8550, Japan
[2] Department of Computer Science, RWTH Aachen University, 52062 Aachen, Germany
[3] Laboratory for Parallel Programming, German Research School for Simulation Sciences, 52425 Juelich, Germany
[4] Juelich Supercomputing Centre, Forschungszentrum Juelich, 52425 Juelich, Germany
[a)] chih-song.kuo@rwth-aachen.de
[b)] nomura.a.ac@m.titech.ac.jp
[c)] matsu@is.titech.ac.jp
[d)] a.shah@grs-sim.de
[e)] f.wolf@grs-sim.de
[f)] i.zhukov@fz-juelich.de
* This paper is an IPSJ SIG Technical Report, neither refereed nor peer-reviewed.

current runtime considering the number of application processes.

The paper is organized as follows: in the next section we describe our experiment set-up in more details. In Section 3, we reproduce the result of downgraded application performance in concurrent executions. Performance evaluations of two co-located I/O intensive applications are summarized in Section 4 with an attempt to concurrent runtime modelling, which is then followed by a review of related research. In Section 6, we draw the conclusions and make a few points on possible future directions.

2. Experiment Set-up

2.1 Platform

All experiments were conducted on a small-scale cluster, raccoon, in Tokyo Institute of Technology. The computing facility has 18 compute nodes, each equipped with one Intel(R) Core(TM) i7-3930K 3.20GHz processor having 6 physical cores capable of running 12 hardware threads in HyperThreading mode, and with 16GiB DDR3 memory. The compute nodes are connected by a combination of DDR/FDR Infiniband in a flat tree topology. As for the choice of operating systems and MPI implementations, Scientific Linux 6.1 and OpenMPI 1.6.3 are used. During the experiment time, the whole system was dedicated for this study, thus reducing possible interferences from other user or system activities.

The working directory of all compute nodes is located on a separate NFS server connected by Gigabit Ethernet, indicating a theoretical peak bandwidth around 120MiB/s. The NFS server employs a RAID 6 hard disk storage capable of 8-way striping. A simple experiment with the *dd* command yields an empirical peak local file-write performance around 1GiB/s. In order to separate possible influences from the RAID storage, all experiments are repeated on a *tmpfs* format ram disk forked on the NFS server. The ram disk provides 1.5GiB/s write bandwidth. In the followings, all the experiments mentioned assume hard disk as the write target, unless otherwise stated.

2.2 Profiling tool

The Light-Weight Monitoring Module (LWM^2) [5] is a profiling tool developed at German Research School for Simulation Sciences. It uses a combination of application sampling and direct instrumentation through interposition wrappers for profiling, leading to low overhead during measurements. It supports MPI and CUDA based parallel applications, while multithreading is handled at the level of POSIX threads, hence covering most of the OpenMP implementations. MPI I/O and POSIX-I/O are also captured. Moreover, the user has the option to enable logging hardware counters via PAPI [8].

Along with creating a complete profile of an application, LWM^2 divides the application execution into fix-sized time-slices and creates separate profiles for them, as shown in Figure 1. The time-slice boundaries are aligned with the system clock, resulting in synchronized time-slices across applications. This gives LWM^2 the unique ability to allow performance analysis across applications by comparing the applications' time-sliced profiles.

In this study, LWM^2 is primarily used to collect the number

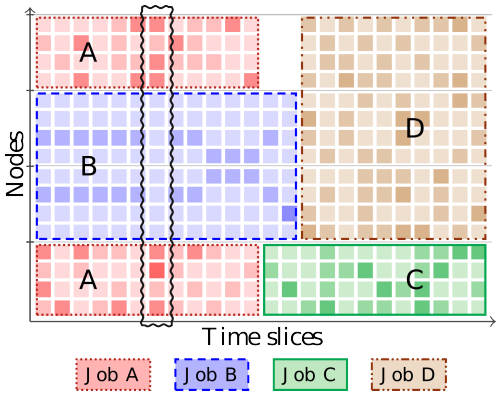

Fig. 1 The LWM^2 profiler is able to log performance counters every 4 seconds for all processes residing on the same cluster.

of bytes written per time slice. When multiple applications are co-located on the cluster, their interferences in I/O accesses are therefore reflected to the fluctuations of the throughput, which enables deeper investigation of I/O resource contention behaviors. Moreover, we found LWM^2 helpful in separating the start-up and ending phases from the whole execution, which have the potential to take up to several minutes on extreme-scale clusters and in turn distort the application throughput if its calculation is based on the total runtime.

2.3 Application

As a starting point of measuring concurrent execution performance, IMB-IO, an executable from the Intel MPI Benchmarks suite release 3.2.4 [7] , is chosen to be our target given its transparent behavior. We fix the file size at 16MiB per application regardless of the number of processes used. To mitigate run-time variance, the write operation is repeated 1000 times per run.

The aggregate execution mode, defined by the Intel MPI Benchmarks as putting assurance of file transfer completion [*1] only after all iterations, is chosen in this study. The decision is made on that most data written by extreme scale application are not read by the program itself but by post-mortem analysis software, and hence file consistency does not need to be frequently assured [2].

We picked two write benchmarks, P_Write_Priv and P_Write_Indv, with the difference that the former runs in one-file-per-process mode, and the latter executes in single-shared-file mode. We limited our study to write operations as they are proved to be more frequent than reads[9] in computational science applications.

The one-file-per-process mode (private mode) has its literal meaning of letting each process operate on its own file. This approach is straightforward in that little additional process synchronization needs to be managed. However the large amount of files does not scale well on certain file systems. Moreover, because the number of files generated highly depends on the number of processes employed, certain difficulties are introduced to post-processing. On the other hand, in the single-shared-file mode (shared mode), all processes manipulate one file with individual file pointers so that there is no worry about metadata. To ease the

[*1] Assurance of file transfer completion is defined as a sequence of MPI_File_Synch, MPI_Barrier, and MPI_File_Synch.

effort of file pointer management, sophisticated parallel I/O APIs like MPI-IO are developed to provide programmers a high-level access to the single-shared-file scheme. In the followings, we use the term parallel file-writing mode to include the two modes.

2.4 Experiment design

We define an experiment to be a choice of one or more applications with a specific configuration the application(s) has to execute. A configuration includes the number of processes, the parallel file-writing mode, the set of compute nodes for each application, hard disk or ram disk used for writing data, and other possible environment variables. Accordingly, a run is an execution of a selected experiment. To further prevent performance fluctuations, each experiment is repeated 5 runs. When the experiments are carried out, we explicitly turn off all system services to the best of our knowledge, and make sure no other user is accessing the system at the same time. Furthermore, we replicated all experiments on a different day to ensure a reproducible result. Finally, for those very few suspectible runs that seem to be affected by unknown jitters, we drop an observation if its throughput bias is higher than 1.5 times of the standard deviation.

We explicitly choose the number of processes an application can have to be 1, 12, 36, and 60. In each run, up to 12 processes can reside in one node. To make comparisons, we first execute the application in standalone mode, meaning no other application running at the same time on any compute node. Then we pick a pair of number of processes (p, q) from $\{1, 12, 36, 60\} \times \{1, 12, 36, 60\}$ for two concurrent-executing applications employing p, q processes, respectively, and launch the dual simultaneously on disjoint compute nodes side by side. This yields 16 combinations in total. In order to mitigate potential influences from operating system noises, the combinations are executed in a round-robin fashion.

3. I/O Performance Bottleneck

Before digging into the concurrent execution performance, we first investigate I/O capability by comparing standalone application throughput to theoretical peak bandwidth. A scalability test is performed in contemplation of the role of the number of processes in parallel writes. Our observation on two parallel file-writing modes yields different conclusions.

3.1 Private (One-File-Per-Process Mode) Mode

In the private mode, write throughput is typically very close to the theoretical bandwidth of the I/O network, Gigabit Ethernet, which appears to be the I/O performance bottleneck in our hardware configuration. Best cases mostly have a throughput slightly above 100MiB/s. Moreover, the I/O performance scales well with the number of processes in that the 12, 36, 60-process experiment all produce similar throughputs except the 1-process application, which is around 80MiB/s. Runs with ram disk serving the target provides 5% slight higher throughput than with hard disk, confirming that the Gigabit Ethernet is dominating the I/O bandwidth. Table 1 summarizes the average write throughput for each configuration.

Table 1 Write throughput (in MiB/s) of IMB-IO run in standalone in private mode with different number of processes.

	1 proc	12 proc	36 proc	60proc
Hard disk	82.6	95.5	95.0	93.5
Ram disk	93.7	98.0	100.4	103.3

3.2 Shared (Single-Shared-File) Mode

In contrast to the private mode, relatively poor scalability is observed in shared mode when writes are made to hard disks. As shown in Table 2, starting at 80MiB/s with one process, the throughput gradually drops to 55MiB/s with 60 processes. However, replicating the same experiments on the ram disk produces a much more stable throughput around 100MiB/s. Contrasting the two results, it is clear that the hard disk-based RAID 6 storage is limiting I/O throughput in a way that is invisible from the hardware setting. We presume the reason to be increased randomize accesses. Although the root cause requires further investigation, we can at least conclude an NFS plus hard disk-driven RAID storage is not good at handling parallel writes to single shared file at scale.

Table 2 Write throughput (in MiB/s) of IMB-IO run in standalone in shared mode with different number of processes.

	1 proc	12 proc	36 proc	60proc
Hard disk	83.8	78.5	73.8	55.4
Ram disk	93.7	93.7	93.6	95.1

4. Concurrent Performance Analysis

4.1 Competition of identical applications

Typical concurrent execution computing environments host tens to hundreds of applications at a time. In order to ease the challenge in analyzing cross-interference from all applications, we begin our experiment by running two identical instances of IMB-IO side by side.

In the example that each application employs 60 processes writing in private mode, we observed 45% drop in I/O throughput for both executables compared to their standalone performances, which in turn causes the two applications to spend 80% extra time to finish their executions. When we run the same configuration in the shared mode, degradation in write throughput becomes more severe such that the value decreases by more than 65%. As a result, concurrent execution of two IMB-IO takes 2.8× runtime, which is even more than twice of standalone runs. Figure 2 illustrates the example with a line chart.

The experiment is repeated by replacing the number of processes to be 1, 12, and 36 instead. The runtime and throughput values are summarized in Figure 3 and Figure 4 . We observed analogous results with different application sizes. Moreover, it is worth mentioning that an application in a concurrent run provides comparable throughputs to its sibling, i.e. no sign of imbalance is detected.

Our results confirm the findings in [5][6], with an addition that the degree of performance drop is also tightly related to the parallel file-writing mode. Specifically, with our file server configuration, I/O performance drops more than 50% in the shared mode, which makes parallel execution of several applications employing such writing mode an uneconomic choice.

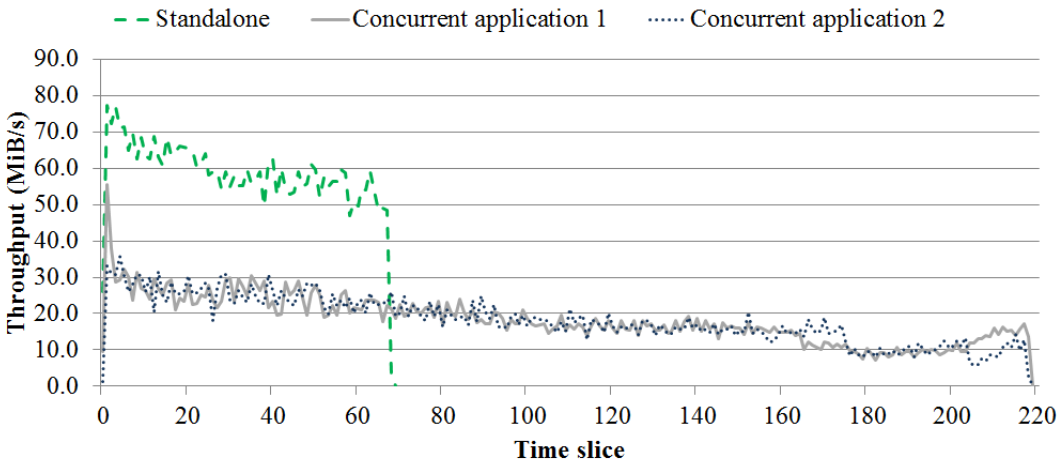

Fig. 2 Comparison of write throughput of an 60-process IMB-IO run in standalone against a pair of concurrent-executing IMB-IOs each employing 60 processes. This diagram shows that in shared-mode, running two I/O-intensive parallel programs can lead to more than 200% run time dilation.

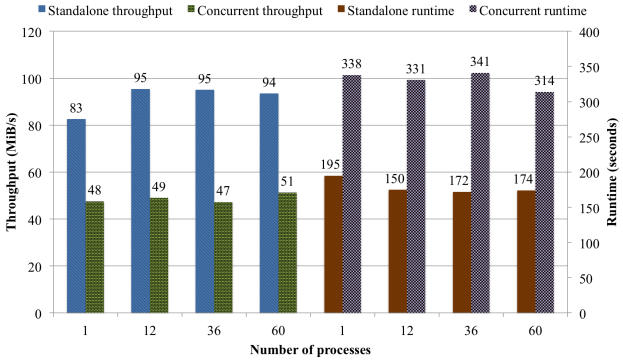

Fig. 3 Comparison of write throughput and runtime of standalone run against concurrent run when the private mode is chosen.

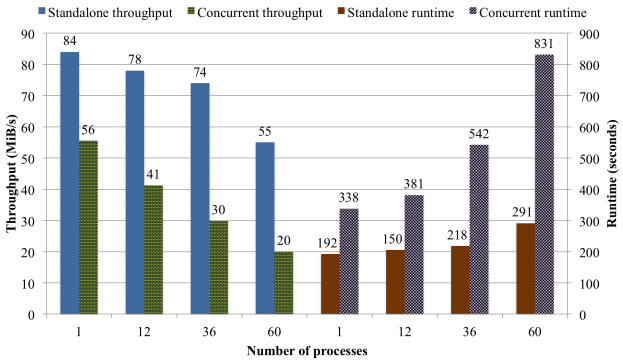

Fig. 4 Comparison of write throughput and runtime of standalone run against concurrent run when the shared mode is chosen.

4.2 Competition of two size-varying applications

In the followings, the size of an application is defined to be the number of processes used while other factors stay constant. In section 3, we already showed the number of processes has only slight impact on write throughput when an application is run in standalone in private mode. However, this information is not sufficient to claim the size of application plays no role in I/O performance when an application is executed in an environment with other applications' presence. To simulate such environment while maintaining in-depth analysis feasible, we intentionally execute two size-varying applications simultaneously. Be noted that the file size is fixed for all applications, meaning a process in a small application has to write more bytes, and vice versa.

The unequal competitions end up with acknowledging that the large application is capable of taking higher Eithernet bandwidth, and hence consumes significantly shorter runtime. We also found the throughput fluctuates more during the overlapped duration. A time slice trace of the throughput is visualized in Figure 5 where the large application is executed in the middle of the small application's total runtime.

Take an example from an experiment where a 12-process application runs concurrently with a 60-process application in private mode, we observed 78.8MiB/s and 50.4MiB/s write throughput, respectively. We found this trend holds for almost all experiments only with a few exceptions run in the shared mode, where the application employing only one process has a high possibility to produce larger throughput compared to its large-sized neighborhood. Moreover, there is an about 50% possibility to get almost equal throughput for two IMB-IOs employing (36, 60) processess. Actually, such phenomenon can be explained with our earlier scalability test in shared mode that larger applications tend to suffer more from increased randomized accesses.

Although the few abnormal behaviors in shared mode are clarified, it is still difficult to find the mathematical relationship between application throughput and the number of processes just by examining the difference in throughput and runtime, even for the private mode runs. To break the black box of I/O resource sharing in a concurrent executing environment, we took a closer look into the number of bytes written per time slice captured by LWM^2. Mapping this data of two applications onto one plane, we found the small application has its runtime divided into two phases, namely the overlapped and the standalone phase, with largely increased bytes written per time slice in the latter one, which indicates the throughput we get for the small application is weighted-averaged, therefore showing a need to find out the real throughput in the overlapped execution period.

With the help of LWM^2 , we were able to sum up the bytes written of the small application during the large application's runtime.

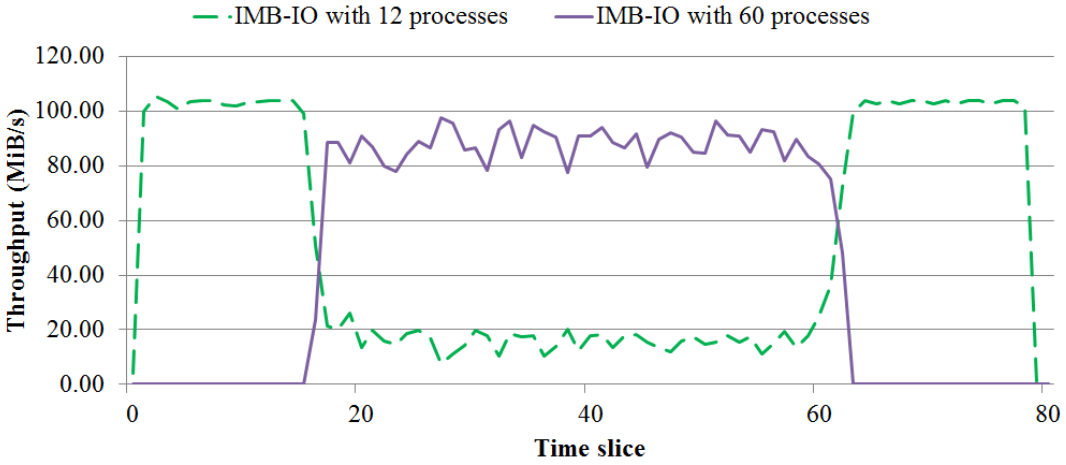

Fig. 5 Write throughput of overlapped execution of one IMB-IO with 12 processes and another one with 60 processes. Note that the performance fluctuates more in the overlapped runtime.

To make such comparison more compact, we further exclude a few ending time slices of the large application in which at least one process has finished writing. In a more precise way, we only calculate the throughput in the active overlapping region, defined as the longest continuous duration when all processes of the two applications write more than zero bytes. The recalculation yields much lower throughput for the small application, as shown in Table 3 together with the overall throughput. In the aforementioned example, we get 83.1MiB/s and 19.0MiB/s for the 60-process and 12-process application, respectively.

Investigating this further, we found that in the private mode, runs where the number of processes of each application is greater than or equal to 12 tend to have the ratio of recalculated throughputs close to the ratio of the number of processes. To explain such findings, we presume each process is given a fair share of I/O network bandwidth. However, as for the cases containing a sequential executing instance, the one-process application tends to be able to take the bandwidth that a 12-process application would get, implying a possibility that the fairness policy is addressed to compute nodes instead, which requires further investigation.

Table 3 Write throughput (in MiB/s) of selected runs where two IMB-IO namely (P, Q) executed concurrently with number of processes (p, q).

		Overall			Overlap		
Mode	(p, q)	P	Q	Ratio	P	Q	Ratio
Private	(1, 12)	47.3	51.8	1:1.1	45.0	52.5	1:1.2
	(1, 36)	46.8	74.8	1:1.6	23.6	77.0	1:3.3
	(1, 60)	46.8	82.0	1:1.8	16.6	87.1	1:5.3
	(12, 36)	48.3	69.2	1:1.4	24.7	71.6	1:2.9
	(12, 60)	50.4	78.8	1:1.6	19.0	83.1	1:4.4
	(36, 60)	51.5	59.9	1:1.5	41.4	64.9	1:1.6
Shared	(1, 12)	49.1	45.1	1:0.9	49.6	43.1	1:0.9
	(1, 36)	43.3	51.5	1:1.2	33.8	54.7	1:1.6
	(1, 60)	37.9	45.6	1:1.2	25.5	50.6	1:2.0
	(12, 36)	38.3	50.8	1:1.3	24.0	54.4	1:2.3
	(12, 60)	35.3	47.8	1:1.4	16.1	53.7	1:3.3
	(36, 60)	23.9	23.7	1:1.0	24.4	33.3	1:1.4

To further confirm our hypothesis that all active processes employed by multiprocessing applications are equal in I/O network, we calculate the average bytes written during the active overlapping region for each process. The result turns out to be a uniform distribution with very small relative standard deviation, defined as standard deviation divided by mean, of only 5.8% in private mode and 20.1% in shared mode. The observation is consistent to [10] in that the biggest application can monopolize the I/O while for the rest the I/O resource is allocated proportionally to the number of processes.

It is worth mentioning that in the shared mode, the throughput does not correlate so tightly to the number of process as in the private mode. Relative large standard deviation among bytes written per process is recognized. Nevertheless, the tendency that the larger applications relation can take more I/O network bandwidth does not change so much.

4.3 Runtime Modeling of Concurrent Application Executions

In the followings, we demonstrate a case where LWM^2 can be useful in modeling application runtime in an I/O resource-sharing environment where network between compute and storage nodes is the major I/O performance bottleneck. We continue the settings in the last section where we found each process gets a fair share relative to the peak I/O bandwidth.

Given two possibly different-sized I/O-dominated applications L and S where each application produces the same amount of I/O write volume in total, and assume their number of processes to be np_L, np_S with $np_L \geq np_S$, standalone runtime to be t_L^{st}, t_S^{st}, respectively. If L and S are executed simultaneously with 100% time overlapping, then the concurrent runtime of L and S, namely t_L^{co}, t_S^{co}, can be derived by the followings.

$$\begin{aligned} t_L^{co} &= \frac{total\_bytes\_written_L}{bandwidth\_allocated_L} \\ &= \frac{bandwidth \times t_L^{st}}{bandwidth \times \frac{np_L}{np_S + np_L}} \\ &= t_L^{st} \times \frac{np_L + np_S}{np_L} \end{aligned} \tag{1}$$

$$t_S^{co} = t_L^{co} + \frac{total\_bytes\_written_S - bytes\_already\_written_S}{bandwidth}$$

$$= t_L^{co} + \frac{total\_bytes\_written_S - \frac{np_S}{np_L+np_S} \times bandwidth \times t_L^{co}}{bandwidth}$$
$$= \frac{total\_bytes\_written_S}{bandwidth} + t_S^{co} \times (1 - \frac{np_S}{np_L + np_S})$$
$$= t_S^{st} + t_L^{st} \times \frac{np_L + np_S}{np_L} \times \frac{np_L}{np_L + np_S}$$
$$= t_S^{st} + t_L^{st} \quad (2)$$

Conversely, we can also predict the standalone runtime from the concurrent runtime.

$$t_L^{st} = t_L^{co} \times \frac{np_L}{np_L + np_S} \quad (3)$$
$$t_S^{st} = t_S^{co} - t_L^{co} \times \frac{np_L}{np_L + np_S} \quad (4)$$

The model is tested against all runs and proved capable of making runtime prediction with a relative error (RE), defined as $\frac{value\ predicted - value\ observed}{value\ observed} \times 100\%$, less than 12% in private mode with the exception of runs comprising of sequential application(s). The model exhibits worse predicability to executions in shared mode due to the additional influences from the hard disk, despite the bias in certain configurations is still low. We visualize the model's predicability along with the observed values in Figure 6 and Figure 7. Overall speaking, it can be referred from the model that when the bandwidth of I/O interconnect is below the bandwidth of the storage device itself, the largest application has a high chance of monopolizing the I/O subsystem and hence substantially delaying other co-existing small applications.

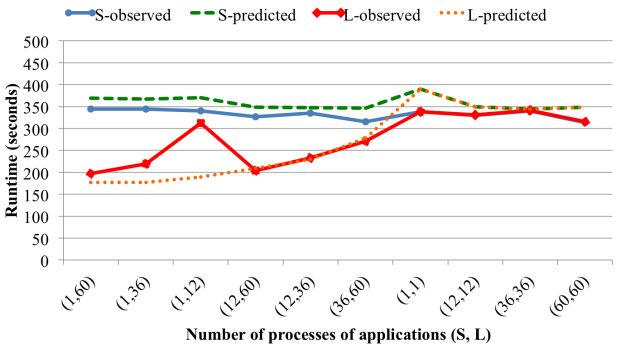

Fig. 6 Predicted and observed concurrent runtime of a pair of IMB-IO (S, L) run in private mode employing (np_S, np_L) processes.

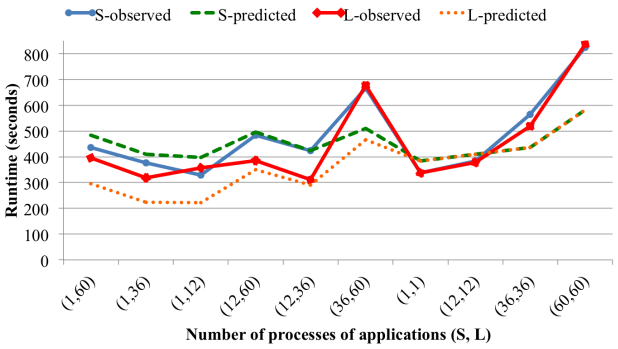

Fig. 7 Predicted and observed concurrent runtime of a pair of IMB-IO (S, L) run in shared mode employing (np_S, np_L) processes.

5. Related Work

Skinner et al. [11] attributed parallel program performance jitter to five causes: (1) Resource contention within compute nodes; (2) MPI synchronization overhead caused by inter-process communication; (3) Kernel process scheduling due to operating system activities; (4) Cross-application contention; (5) System activity outside the node. In particular, using a statistical approach, I/O performance variability is proved to be highly correlated with I/O performance based on the trace of two extreme-scale scientific applications [12]. Furthermore, measurable I/O variability is classified as internal interference and external interference [6], which differ in that the former happens within an application and the latter occurs among applications, which are not restricted to petascale simulation codes but also include analysis or visualization programs.

I/O resource contention can appear in both physical storage and I/O network. Lofstead et. al. [6] reported halved throughput due to external interference in an environment that two applications run concurrently, which we also confirmed in our previous study [5]. A more detailed observation on Intrepid concluded an application's I/O performance to scale with the the process count [10]. Xie et. al [13] carried out a study about I/O resource competing in a different way to reveal how burst-size, write-sharing, stripped-wirte, and the number of storage nodes relate to the I/O bandwidth allocated to an application executing in an I/O resource-sharing environment. In terms of network, evidence shows that the presence of contention can increase message latencies signicantly with increasing number of hops messages travel [14]. Lastly, in contrast to all aforementioned studies targeting spcialized parallel I/O storage architectures, some work exist to monitor and characterize NFS server workloads in the industry [15][16], although the usage pattern in an enterprise environment differs from HPC clusters in that enterprise programs are usually not massively-parallelized and have lower read/write volume.

On the perspective of parallel application I/O charactertization and measurement tools, several have been provided by the HPC community. General-purpose modules include VAMPIR [17], TAU [18], Scalasca [19], IPM [20], Perfsuite [21]. However, they mostly lack the ability to generate synchronized time slice profiles, or introduce too much overhead to the application. For I/O operations in particular, LANL-Trace [22], IOT [23], and Darshan [1] exist to help developers tackle I/O performance issues. Among them, Darshan is capable of generating profiles with negligible overhead, and had been deployed on Intrepid for two months [24], which produced several important findings about I/O access characteristics in an HPC setting. However, Darshan cannot log cross-process correlation and temporal information, and thus is still not suitable for capturing application I/O interferences.

6. Conclusions and Future Work

In this study, we presented an approach using the LWM^2 profiler to understand competition for I/O among applications. In our experiment environment where an NFS server connected by Gigabit Ethernet acts as the storage node, we confirmed an up

to 65% drop in write throughput in a setting that two IMB-IO instances execute concurrently.

When applications of different sizes are executed at the same time, we found the I/O bandwidth allocated to an application is approximately proportional to the number of processes, suggesting the larger the application is, the higher the throughput is. However, at the same time, we also observed that in shared mode, such trend can sometimes be reversed because a large number of processes can hinder write throughput possibly due to increased random access to hard disks.

Throughout our analysis, we demonstrated LWM^2 's usefulness in recording process time slice profiles. This feature not only enables investigating process load balance over execution time, but also makes comparing application performances in their overlapped runtime possible, which allows for opening the black box of a typical concurrent execution, resource-sharing computing environment that the HPC community has not known much about. At the time of writing this paper, LWM^2 is likely the only tool capable of doing this job with low runtime overhead and little extra human effort.

To extend our result to more practical applications, we are currently planning larger experiments on extreme scale clusters such as TSUBAME 2.5, JUQUEEN, which employ Lustre I/O as their storage system. I/O benchmarks which simulate real application behaviors such as IOR, NAS-BTIO will be used in addition to IMB-IO in order to understand real world application interferences.

References

[1] Carns, P., Latham, R., Ross, R., Iskra, K., Lang, S. and Riley, K.: 24/7 Characterization of petascale I/O workloads, *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, pp. 1–10 (online), DOI: 10.1109/CLUSTR.2009.5289150 (2009).

[2] Dorier, M., Antoniu, G., Cappello, F., Snir, M. and Orf, L.: Damaris: How to Efficiently Leverage Multicore Parallelism to Achieve Scalable, Jitter-free I/O, *CLUSTER - IEEE International Conference on Cluster Computing*, Beijing, Chine, IEEE, (online), available from ⟨http://hal.inria.fr/hal-00715252⟩ (2012).

[3] Kassick, R., Boito, F., Diener, M., Navaux, P., Denneulin, Y., Schepke, C., Maillard, N., Osthoff, C., Grunmann, P., Dias, P. and Panetta, J.: Trace-Based Visualization as a Tool to Understand Applications' I/O Performance in Multi-core Machines, *Architecture and Multi-Core Applications (WAMCA), 2011 Second Workshop on*, pp. 5–11 (online), DOI: 10.1109/WAMCA.2011.12 (2011).

[4] Byna, S., Chen, Y., Sun, X.-H., Thakur, R. and Gropp, W.: Parallel I/O Prefetching Using MPI File Caching and I/O Signatures, *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, Piscataway, NJ, USA, IEEE Press, pp. 44:1–44:12 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1413370.1413415⟩ (2008).

[5] Shah, A., Wolf, F., Zhumatiy, S. and Voevodin, V.: Capturing inter-application interference on clusters, *Proc. of IEEE International Conference on Cluster Computing (Cluster 2013), Bloomington, IN, USA*, IEEE Computer Society (2013).

[6] Lofstead, J., Zheng, F., Liu, Q., Klasky, S., Oldfield, R., Kordenbrock, T., Schwan, K. and Wolf, M.: Managing Variability in the IO Performance of Petascale Storage Systems, *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*, pp. 1–12 (online), DOI: 10.1109/SC.2010.32 (2010).

[7] Intel Corporation: Intel MPI Benchmark User Guide and Methodology Description Version 3.2.4, (online), available from ⟨http://software.intel.com/sites/products/documentation/hpc/ics/imb/32/IMB_Users_Guide/IMB_UsersGuide.pdf⟩ (accessed 2013-11-18).

[8] Mucci, P. J., Browne, S., Deane, C. and Ho, G.: PAPI: A Portable Interface to Hardware Performance Counters, *In Proceedings of the Department of Defense HPCMP Users Group Conference*, pp. 7–10 (1999).

[9] Kim, Y., Gunasekaran, R., Shipman, G., Dillow, D., Zhang, Z. and Settlemyer, B.: Workload characterization of a leadership class storage cluster, *Petascale Data Storage Workshop (PDSW), 2010 5th*, pp. 1–5 (online), DOI: 10.1109/PDSW.2010.5668066 (2010).

[10] Lang, S., Carns, P., Latham, R., Ross, R., Harms, K. and Allcock, W.: I/O performance challenges at leadership scale, *High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on*, pp. 1–12 (online), DOI: 10.1145/1654059.1654100 (2009).

[11] Skinner, D. and Kramer, W.: Understanding the causes of performance variability in HPC workloads, *Workload Characterization Symposium, 2005. Proceedings of the IEEE International*, pp. 137–149 (online), DOI: 10.1109/IISWC.2005.1526010 (2005).

[12] Uselton, A., Howison, M., Wright, N., Skinner, D., Keen, N., Shalf, J., Karavanic, K. and Oliker, L.: Parallel I/O performance: From events to ensembles, *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pp. 1–11 (online), DOI: 10.1109/IPDPS.2010.5470424 (2010).

[13] Xie, B., Chase, J., Dillow, D., Drokin, O., Klasky, S., Oral, S. and Podhorszki, N.: Characterizing output bottlenecks in a supercomputer, *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, Los Alamitos, CA, USA, IEEE Computer Society Press, pp. 8:1–8:11 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2388996.2389007⟩ (2012).

[14] Bhatele, A. and Kale, L. V.: Quantifying Network Contention on Large Parallel Machines, *Parallel Processing Letters*, Vol. 19, No. 04, pp. 553–572 (online), DOI: 10.1142/S0129626409000419 (2009).

[15] Leung, A. W., Pasupathy, S., Goodson, G. and Miller, E. L.: Measurement and Analysis of Large-scale Network File System Workloads, *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, ATC'08, Berkeley, CA, USA, USENIX Association, pp. 213–226 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1404014.1404030⟩ (2008).

[16] Anderson, E.: Capture, Conversion, and Analysis of an Intense NFS Workload, *Proccedings of the 7th Conference on File and Storage Technologies*, FAST '09, Berkeley, CA, USA, USENIX Association, pp. 139–152 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1525908.1525919⟩ (2009).

[17] Nagel, W. E., Arnold, A., Weber, M., Hoppe, H.-C. and Solchenbach, K.: VAMPIR: Visualization and Analysis of MPI Resources, *Supercomputer*, Vol. 12, pp. 69–80 (1996).

[18] Shende, S. S. and Malony, A. D.: The Tau Parallel Performance System, *Int. J. High Perform. Comput. Appl.*, Vol. 20, No. 2, pp. 287–311 (online), DOI: 10.1177/1094342006064482 (2006).

[19] Geimer, M., Wolf, F., Wylie, B. J. N., Ábrahám, E., Becker, D. and Mohr, B.: The Scalasca Performance Toolset Architecture, *Concurr. Comput. : Pract. Exper.*, Vol. 22, No. 6, pp. 702–719 (online), DOI: 10.1002/cpe.v22:6 (2010).

[20] Fuerlinger, K., Wright, N. and Skinner, D.: Effective Performance Measurement at Petascale Using IPM, *Parallel and Distributed Systems (ICPADS), 2010 IEEE 16th International Conference on*, pp. 373–380 (online), DOI: 10.1109/ICPADS.2010.16 (2010).

[21] Kufrin, R.: Perfsuite: An Accessible, Open Source Performance Analysis Environment for Linux, *In Proc. of the Linux Cluster Conference, Chapel* (2005).

[22] Los Alamos National Laboratory: LANL-Trace, (online), available from ⟨http://institute.lanl.gov/data/software/#lanl-trace⟩ (accessed 2013-11-18).

[23] Roth, P. C.: Characterizing the I/O Behavior of Scientific Applications on the Cray XT, *Proceedings of the 2Nd International Workshop on Petascale Data Storage: Held in Conjunction with Supercomputing '07*, PDSW '07, New York, NY, USA, ACM, pp. 50–55 (online), DOI: 10.1145/1374596.1374609 (2007).

[24] Carns, P., Harms, K., Allcock, W., Bacon, C., Lang, S., Latham, R. and Ross, R.: Understanding and improving computational science storage access through continuous characterization, *Mass Storage Systems and Technologies (MSST), 2011 IEEE 27th Symposium on*, pp. 1–14 (online), DOI: 10.1109/MSST.2011.5937212 (2011).