Rethinking Node Allocation Strategy for Data-intensive Applications in Consideration of Spatially Bursty I/O

Jie Yu
National University of Defense Technology
College of Computer
Changsha, China
yujie@nudt.edu.cn

Guangming Liu
National University of Defense Technology
College of Computer
Changsha, China
liugm@nscc-tj.gov.cn

Xin Liu
National University of Defense Technology
College of Computer
Changsha, China
liuxin@nscc-tj.gov.cn

Wenrui Dong
National University of Defense Technology
College of Computer
Changsha, China
dongwr@nscc-tj.gov.cn

Xiaoyong Li
National University of Defense Technology
College of Meteorology and Oceanology
Changsha, China

Yusheng Liu
National University of Defense Technology
College of Computer
Changsha, China
liuys@nscc-tj.gov.cn

ABSTRACT

Job scheduling in HPC systems by default allocate adjacent compute nodes for jobs for lower communication overhead. However, it is no longer applicable to data-intensive jobs running on systems with I/O forwarding layer, where each I/O node performs I/O on behalf of a subset of compute nodes in the vicinity. Under the default node allocation strategy a job's nodes are located close to each other and thus it only uses a limited number of I/O nodes. Since the I/O activities of jobs are bursty, at any moment only a minority of jobs in the system are busy processing I/O. Consequently, the bursty I/O traffic in the system is also concentrated in space, making the load on I/O nodes highly unbalanced. In this paper, we use the job logs and I/O traces collected from Tianhe-1A to quantitatively analyze the two causes of spatially bursty I/O, including uneven I/O traffic of job's processes and uneven distribution of job's nodes. Based on the analysis we propose a node allocation strategy that takes account of processes' different amounts of I/O traffic, so that the I/O traffic can be processed by more I/O nodes more evenly. Our evaluations on Tianhe-1A with synthetic benchmarks and realistic applications show that the proposed strategy can further exploit the potential of I/O forwarding layer and promote the I/O performance.

CCS CONCEPTS

• **Computer systems organization → Architectures**;

ICS '18, June 12–15, 2018, Beijing, China
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5783-8/18/06...$15.00
https://doi.org/10.1145/3205289.3205305

KEYWORDS

Spatially Bursty I/O, Job Scheduling, Node Allocation, I/O Forwarding, Information Entropy

1 INTRODUCTION

The ever-growing complexity and scale of scientific and engineering applications have driven high performance computing (HPC) systems to extreme level of compute capability. Take the latest Top 1 system Sunway TaihuLight [17] as an example, it has more than ten millions of compute cores and reaches 93 petaflops of Linpack performance. The scale of next-generation HPC systems will grow by orders of magnitude within a few years. The underlying storage system becomes more and more incompetent to handle the tremendous amount of concurrent I/O from the extreme scale of compute cores.

In order to alleviate the I/O pressure on storage systems, many large-scale HPC systems such as BlueGene/Q [1], Cray XC [3] and Tianhe-2 [10] insert an I/O forwarding layer between compute nodes and storage system, as shown in Figure 1. Each I/O node serves a subset of compute nodes in the vicinity and forwards their I/O requests to the storage system. Since the ratio of compute nodes to I/O nodes varies from 8:1 to 64:1 [2], the number of clients served by storage system reduces from tens of thousands (i.e. the number of compute nodes) to hundreds (i.e. the number of I/O nodes). I/O forwarding layer makes it possible to scale current storage systems to the extreme level of concurrency.

However, some novel characteristics appear in the HPC I/O stack after inserting I/O forwarding layer. Without I/O forwarding layer, all storage nodes can be accessed by all compute nodes. The complete sharing of storage resources reduces the possibility of load imbalance. After inserting I/O forwarding layer, I/O nodes replace the storage nodes and become the storage resources of compute nodes. However, each I/O node can only be exclusively utilized by a subset of compute nodes. I/O nodes become separated aggregation

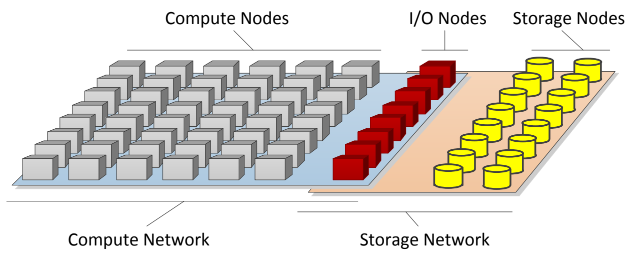


Figure 1: Architecture of HPC systems with I/O forwarding layer.

points in the I/O paths from compute nodes to storage, resulting in the potential to be load unbalanced. In particular, the spatially bursty I/O characteristic exhibited in HPC systems exacerbates the load imbalance problem.

Much work [4, 8, 12] on I/O characterization has uncovered bursty I/O characteristic in HPC systems. Bursty I/O means the I/O demands can be tremendously high for short periods of time. A survey on Intrepid shows that, for 98% of the time, the I/O subsystems are utilized less than 33% [4]. We collect and analyze the I/O traces on a top HPC system Tianhe-1A and find that the I/O traffic is not only bursty in time, but also bursty in space. That is, the enormous amount of I/O traffic not only exists in short periods of time, but also comes only from a minority of compute nodes located close to each other. Since adjacent compute nodes probably share the same I/O node, the immense amount of bursty I/O traffic can only be forwarded by a minority of I/O nodes. The caused spatially bursty I/O poses two problems. Firstly, it makes the load on I/O nodes very unbalanced. The I/O nodes serving I/O-intensive compute nodes are heavily loaded and can easily become the bottleneck while other I/O nodes have little workload to handle. Secondly, since only a small number of I/O nodes access data at any moment, the concurrent performance of parallel file system can not be fully exploited. Thus, even if the small number of I/O nodes are not overloaded, their I/O performance is limited.

In this paper, we quantitatively analyze the two main causes of spatially bursty I/O. The first cause is the intrinsic characteristic of applications that the amount of I/O traffic varies with different compute nodes. We find that 25.7% multi-nodes jobs use only one node to perform most of their I/O and 38.1% of the rest jobs have uneven I/O traffic distribution on their nodes. The second cause is the node allocation strategy of Slurm, which preferentially maps jobs to adjacent compute nodes. We find that the compute nodes of 70.4% jobs are concentrated on a minimal number of cabinets. In the jobs that run on multiple cabinets, 50.7% jobs' nodes are unevenly distributed on their cabinets.

Based on above analysis, we propose a node allocation strategy that takes the spatially bursty I/O into account. We extract I/O characteristics of jobs from realistic job logs and I/O traces and use the historical jobs' node traffic distribution to guide the node allocation of a new job so that its I/O traffic can be distributed on more I/O nodes more evenly. Our evaluations on Tianhe-1A with synthetic benchmarks and realistic applications show that the proposed strategy can further exploit the potential of I/O forwarding layer and significantly promote the performance.

2 JOB LOGS AND I/O TRACES ON TIANHE-1A

Tianhe-1A was the No. 1 system in TOP500 list in November 2010. There are 64 compute nodes in each of its cabinet. Cabinets are connected through 384-port switches via fat tree topology. As the first system of Tianhe series, it has no I/O forwarding infrastructure. Even so, it does not prevent us from using its logs and traces to investigate the impacts of spatially bursty I/O on the I/O forwarding layer. Because the I/O forwarding layer is completely transparent to applications and job scheduling system, it has absolutely no affect on the logs and I/O trace of jobs. With or without I/O forwarding layer, jobs are scheduled and mapped in the same way, and applications leave the same I/O traces on the compute nodes. Therefore, we can make a simple assumption that 64 compute nodes in a cabinet share an I/O node when analyzing the logs and traces on Tianhe-1A.

HPC systems use job scheduling system to efficiently deliver computing power to applications. Slurm is a widely used job scheduling system in HPC community. We collect the Slurm job logs on Tianhe-1A over three years (from January, 2015 to January, 2018), including the start and end time of jobs, their number and names of allocated nodes, etc.

The I/O traces on Tianhe-1A are collected with Zabbix, an enterprise open source monitoring software. We deployed Zabbix daemons on 5677 compute nodes to read their status in the `/proc` file system, including Lustre client read and write I/O traffic, network traffic, CPU and memory usage, etc. We set the data capture interval to 10 seconds in order to obtain as fine traces as possible while avoiding significant performance impacts. We collect traces on compute nodes over 24 days (from December 11, 2017 to January 5, 2018).

Since jobs exclusively use their allocated compute nodes on Tianhe-1A, we can get to know a job's I/O behavior in granularity of compute node by combining its start time, end time and allocated nodes with the I/O traces on compute nodes. To the best of our knowledge, job's I/O traffic distribution on compute nodes has not been well studied yet. Therefore, we have found some interesting new results, which will be elaborated in the rest of this paper.

3 IDENTIFYING SPATIALLY BURSTY I/O

3.1 Uneven I/O Traffic of Job Processes

3.1.1 Rank 0 I/O. Developers of large-scale applications use thousands of processes working in parallel to solve complex scientific problems or undertake large engineering simulations. Although the tasks performed by the processes are likely to be highly relevant, their I/O demands vary. An extreme case is rank 0 I/O pattern [9]. In applications that exhibit rank 0 I/O pattern, only one process performs I/O although applications may be running on thousands of them. It is a common I/O pattern in scientific applications. Applications utilizing non-parallel high-level I/O libraries (e.g. serial HDF5 and NetCDF) have to read input data and write output data with the root process. Applications that need to reduce the data located in all the processes also exhibit rank 0 I/O pattern, because the reduction (e.g. summing or sorting) must eventually be completed on one process and the data will be written out by that process.

We analyze the I/O traces of all jobs that run on multiple compute nodes, and investigate whether rank 0 I/O pattern is prevalent among them. Since we only obtain the total I/O traffic in a node, the results reveal node 0 other than rank 0 I/O pattern. However, the statistics of whether I/O traffic is concentrated in only one compute node still have significance for identifying spatially bursty I/O. We calculate the proportion of each node's I/O traffic to the job's total traffic. If a node's I/O traffic amount exceeds 95%, the job is considered to exhibit rank 0 I/O pattern. The statistics show that 25.7% multi-nodes jobs use only one node to perform most of the I/O.

When applications read or write large amounts of data, their rank 0 I/O patterns make almost all the data traffic flow through only one compute node. Even if an application is running on many compute nodes and multiple I/O nodes are used to forward its I/O requests, there is only one I/O nodes processing the tremendous amount of concentrated I/O traffic. The load on I/O nodes is highly unbalanced and one of the I/O nodes can easily become the bottleneck.

3.1.2 All rank I/O. Compared with rank 0 I/O, all rank I/O is a more intuitive way to access data since all processes perform I/O for their own needs. However, even if all processes issue I/O requests, their amount of I/O traffic can be uneven, which can still cause spatially bursty I/O.

We developed an information entropy-based mathematical model to quantify the unevenness of I/O traffic. Information entropy [15] is defined as the average amount of information of a random variable, and also refers to the disorder or uncertainty of a probability distribution. Shannon defined the entropy H of a discrete random variable X with possible values $\{x_1, ..., x_N\}$ and probability mass function $P(X)$ as

$$H(X) = -\sum_{i=1}^{N} P(x_i) \log P(x_i). \tag{1}$$

The definition determines that the lower-probability value contributes larger information entropy. It is consistent with common sense since what is probable to happen (e.g. the sun rises in the east) does not bring us much information. Since information entropy is monotonic, if X is more deterministic, it contains less information and has smaller entropy. When X is more random, it contains more information and has larger entropy. A variable is more random means that the probability distribution of its possible values is evener. A uniform distribution U is the most random distribution since all its possible values have equal probability. It has the largest entropy of

$$H(U) = -\sum_{i=1}^{N} \frac{1}{N} \log \frac{1}{N} = \log N. \tag{2}$$

Therefore, information entropy can be used as a measure of how even a probability distribution is. When a distribution is more even, then its information entropy is closer to the entropy of uniform distribution, i.e. the maximal entropy. When a distribution is more concentrated, its entropy is closer to the minimal entropy of 0. If we treat the amount of I/O traffic of a job's N nodes as a discrete distribution of length N, the informational entropy of the traffic distribution measures the evenness of the I/O traffic.

Since what we want to measure is the unevenness of the I/O traffic distribution, we can define the unevenness degree (UD) as

$$UD(X) = H(U) - H(X) = \log N - H(X), \tag{3}$$

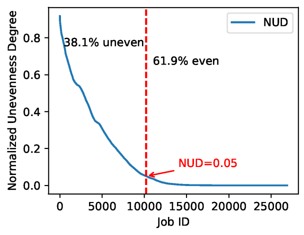
where X is an I/O traffic distribution of a job running on N compute nodes, and U is a uniform distribution of the same length as X. The more uneven the traffic distribution X, the closer $H(X)$ is to 0 and the closer $UD(X)$ is to the maximal entropy of $\log N$. In fact, the same result can be derived if we calculate the Kullback-Leibler divergence from X to U. Since Kullback-Leibler divergence is a widely used measure of how a probability distribution diverges from another, our definition of UD is mathematically correct.

In order to make jobs of different sizes have unevenness degrees in $[0, 1]$, we define the normalized unevenness degree (NUD) as

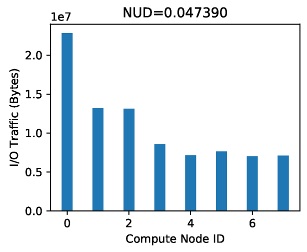
$$NUD(X) = \frac{UD(X)}{H(U)} = 1 - \frac{H(X)}{\log N}. \tag{4}$$

By definition, NUD is monotonic for distributions of the same length. In other words, for different jobs that are allocated with the same number of nodes, the larger the NUDs, the more uneven the traffic distributions. However, for jobs of different numbers of nodes, their NUDs are not comparable because of the inherent difficulty of defining unevenness of distributions of different lengths. For example, it is difficult to determine which of the two probability distributions, $\{0.4, 0.3, 0.3\}$ and $\{0.3, 0.3, 0.2, 0.2\}$, is more uneven. Its accurate definition is beyond the scope of this paper.

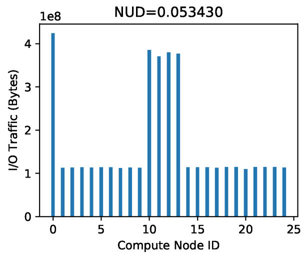
We calculate the NUDs of all jobs exhibiting all rank I/O pattern. Their values are shown in Figure 2(a). In order to understand more intuitively how uneven a NUD value is, we select 3 distributions of different lengths that have NUD around 0.05 and draw their traffic distribution in Figure 2(b)(c)(d). As can be seen from the figure, the I/O traffic distributions of these jobs are all very uneven. Since NUD is monotonic in jobs with the same number of nodes, a job is more uneven than the example jobs in the figure if it runs on the same number of nodes and has a larger NUD. If we use 0.05 as a NUD threshold to determine whether a job's



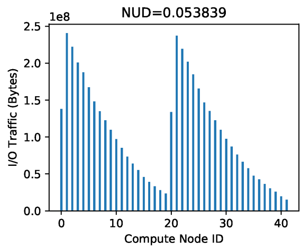
(a) The NUDs of all jobs.



(b) Traffic distribution of a job running on 8 nodes.



(c) Traffic distribution of a job running on 25 nodes.



(d) Traffic distribution of a job running on 42 nodes.

Figure 2: Uneven I/O traffic distributions of jobs exhibiting all rank I/O pattern.

traffic distribution is uneven, then 38.1% jobs' distributions are more uneven than the example jobs.

When the I/O traffic distribution of a job is uneven, even if the job's node are located in multiple cabinets and multiple I/O nodes are used to forward its I/O, the I/O traffic can still concentrated in a small minority of the I/O nodes. Thence, in addition to the rank 0 I/O pattern, the uneven I/O traffic distribution of processes is another important reason for causing spatially bursty I/O.

3.2 Uneven Distribution of Job Nodes

The scheduling system selects the most appropriate compute nodes to assign to a job based on its specific requirements (e.g. number of nodes, memory capacity). However, it is possible that the assigned nodes are unevenly distributed on multiple cabinets. In this case, multiple I/O nodes are used by a job, but the number of compute nodes served by each I/O node is not the same. Even if the job's I/O traffic is evenly distributed on all its compute nodes, the uneven distribution of compute nodes on cabinets can still cause spatially bursty I/O.

Tianhe-1A uses Slurm to schedule jobs and manage resources. Slurm's default strategy of node selection for jobs is to consider the nodes as a one-dimensional array. Jobs are allocated with compute nodes on a best-fit algorithm based on the number of consecutive nodes. For larger jobs, this minimizes the number of sets of consecutive nodes allocated to the job. Slurm also provides several plugins to support various kinds of network topology, including torus and fat tree network. In fat tree network of Tianhe-1A, the plugin attempts to place the nodes allocated to a job in the least number of leaf switches, so that most of the communication takes places within the switches. The best-fit strategy and the fat tree plugin make jobs be allocated with large chunks of nodes located close to each other, wherefore the nodes tend to be grouped in a smaller number of cabinets and use less I/O nodes.

Under the default node allocation strategy of Slurm, for a N nodes job, Slurm tends to centrally place its nodes on $\lceil N/64 \rceil$ cabinets, which is the minimal number of cabinets the job's nodes can be located on. However, in a production system, a large number of concurrent jobs compete for the limited number of node resources at the same time, so a job's actual number of cabinets may be larger than the minimal number. We investigate the node distribution of all jobs on Tianhe-1A over 3 years and find that the nodes of 70.4% multi-nodes jobs are only located in the minimal number of cabinets. Even if the job's nodes are distributed on more cabinets due to node competition, the numbers of nodes on different cabinets can be different. Similar to analyzing the I/O traffic distribution of job processes, we use NUD to quantify the unevenness of node distributions of multi-cabinets jobs. The results are shown in Figure 3. Similarly, if 0.05 is used as a NUD threshold to determine whether a distribution is uneven, then the node distribution of 50.7% jobs are uneven.

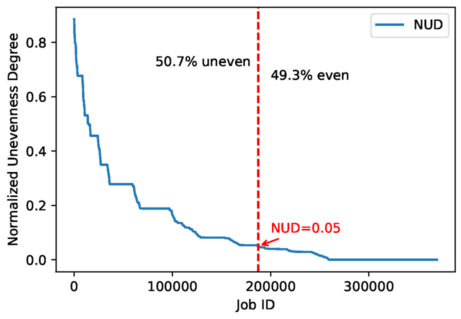
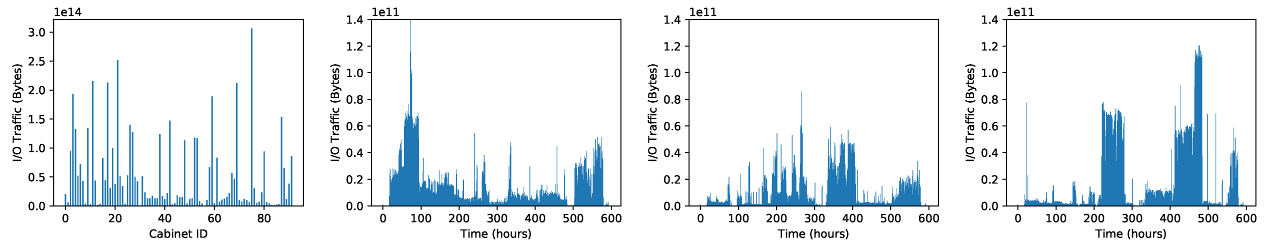


Figure 3: Uneven node distribution on cabinets.

It causes spatially bursty I/O in the system that jobs' nodes are unevenly distributed in a small number of cabinets. In production environment multiple jobs sharing cabinets will help balance the cabinet load. However, the I/O demands of jobs are tremendously high only for short periods of time. At any moment only a minority of jobs in the system access immense amounts of data and thus only a small number of I/O nodes are heavily loaded. Therefore, the load on I/O nodes are highly unbalanced and the I/O forwarding layer is seriously underutilized.



(a) The amounts of I/O traffic of all cabinets. (b) I/O traffic on cabinet 70. (c) I/O traffic on cabinet 71. (d) I/O traffic on cabinet 72.

Figure 4: Spatially bursty I/O on Tianhe-1A.

3.3 Spatially Bursty I/O on Tianhe-1A

Due to uneven I/O traffic distribution on job processes and uneven node distribution on cabinets, the I/O on HPC systems tends to concentrate on a minority of cabinets. We collect and analyze the I/O traffic of 5677 compute nodes located on 92 cabinets of Tianhe-1A, and observe evident spatially bursty I/O.

Figure 4(a) shows the total amount of I/O traffic handled by each cabinet. We can see that different cabinets have processed very different amounts of I/O traffic. If there is an I/O node connected to each cabinet, some I/O nodes would be overloaded while others have little workload to handle. We select 3 adjacent cabinets and draw their I/O traffic over time in Figure 4(b)(c)(d). As can be seen, bursty I/O pattern exists on all cabinets. The traffic peaks appear only for short periods of time. In addition, the arrival time of I/O bursts on different cabinets is not the same. At some moment when an I/O node is busy processing large numbers of I/O requests, there is little I/O load on other I/O nodes. Thus, at any moment only a minority of I/O nodes in the system share the responsibility of processing huge amount of I/O. The spatially bursty I/O makes the I/O forwarding layer unable to achieve its full capability.

4 OPTIMIZED NODE ALLOCATION STRATEGY

4.1 Node Allocation Criteria

According to above analysis, the node allocation strategy of Slurm is one of the major causes of spatially bursty I/O. Slurm's default strategy does not take jobs' different I/O characteristics into consideration when assigning compute nodes to them. In this paper, we propose to optimize the node allocation for data-intensive jobs, so that more I/O nodes can be utilized more evenly. Specifically, the following three criteria are proposed to guide their node allocation.

Nodes should be placed on more cabinets. If all the nodes of a job are centrally located in a single cabinet, all their I/O traffic must be forwarded by the same I/O node and thus makes the I/O node overloaded. If the job's nodes are located in more cabinets, more I/O nodes will be used to share the excessive I/O traffic. Not only I/O nodes are more load balanced, jobs can also benefit from the higher parallel performance of more I/O nodes and achieve better I/O efficiency.

Node distribution on multiple cabinets should be as even as possible. According to the analysis in Section 3.2, even if the job's nodes are located in more than one cabinet, the numbers of nodes in different cabinets can be very uneven, resulting in spatially bursty I/O. Placing the nodes evenly on multiple cabinets can effectively balance loads on I/O nodes and improve I/O forwarding efficiency.

I/O traffic distribution on multiple cabinets should be as even as possible. According to the analysis in Section 3.1.2, different processes of a job have different I/O demands, resulting in uneven I/O traffic on compute nodes. Even if the nodes are evenly distributed among multiple cabinets, the I/O amounts of different cabinets are still uneven. Therefore, we must take into account the amounts of I/O traffic of nodes when distributing them to multiple cabinets, so that the I/O traffic of all cabinets can be more balanced.

4.2 Optimizable Jobs

Not all jobs can be optimized by rearranging their node distribution. Specifically, only jobs with following I/O characteristics can benefit from our optimization.

4.2.1 Data-intensive Jobs. Only data-intensive jobs can cause significant spatially bursty I/O and only their I/O efficiencies can be greatly impacted by unbalanced I/O nodes. Therefore, our approach can only promote the performance of data-intensive jobs by placing their nodes on more cabinets more evenly. On the contrary, if applying the optimized strategy to non-data-intensive jobs, it may increase their inter-node communication overhead and impact their performance.

We analyze the I/O traffic of all time intervals in the job's I/O trace. If the traffic of a time interval exceeds a certain threshold, the interval is counted as the I/O time of the job. However, measuring job I/O activity based solely on I/O traffic is one-sided. A job that is busy accessing data with a large amount of small I/O is also data-intensive although it has no large amount of I/O traffic. Therefore, we also count a time interval whose I/O count exceeds a certain threshold as the I/O time of a job. The ratio of a job's I/O time to its running time can be used to determine whether it is data-intensive.

4.2.2 Jobs with All Rank I/O Pattern.

For a job that exhibits rank 0 I/O pattern, its I/O traffic is concentrated on one of its nodes. Therefore, no matter how Slurm places its nodes, all I/O requests are forwarded to only one I/O node, making that I/O node overloaded. Only jobs that exhibit all rank I/O pattern can benefit from a more even I/O traffic distribution on I/O nodes. We calculate the proportion of the I/O traffic of a job's all nodes to the job's total traffic. When the traffic proportion of a node exceeds a certain threshold, the job is considered to exhibit rank 0 I/O pattern.

4.2.3 Jobs with Low Communication Traffic.

Slurm by default places a job's nodes on as fewer cabinets as possible to minimize the network overhead. In order to apportion the concentrated I/O traffic to more I/O nodes, our approach tries to distribute a job's nodes on more cabinets more evenly, which will inevitably increase the network overhead. If a job has a large amount of inter-node communication, it may not benefit from the optimization.

We subtract the remote I/O traffic from the total network traffic to obtain the communication traffic and calculate the ratio of a job's communication traffic to remote I/O traffic. If the ratio exceeds a certain threshold, the job is considered to be unsuitable for our optimized strategy.

4.3 I/O Characteristic Consistency

Users of HPC systems often run the same application multiple times with different parameters or different data to get the best results [14]. For the sake of clarity, in this paper, we call the multiple runs of the same application its jobs. Thence different jobs of the same application may exhibit different runtime characteristics. The prerequisite for using the job logs of an application to guide the node assignment of its new jobs is that the I/O characteristics are consistent across all its jobs. To this end, we investigate the I/O characteristic consistency across jobs.

We need to determine what kind of jobs belong to the same application. Jobs belonging to the same application should not only have the same job name and username (i.e. submitter) but also have the same numbers of running nodes. The main reason is that we need to balance the I/O traffic on I/O nodes based on the node traffic distributions of historical jobs. If their numbers of nodes are different, there is no guidance value in historical jobs.

An application either can meet the three requirements described in Section 4.2 or cannot benefit from our approach. Therefore, it is required that all the three I/O characteristics are consistent across an application's all jobs. Furthermore, we can not balance the traffic on I/O nodes for a newly submitted job unless it has similar traffic distribution on its nodes with its application's historical jobs. Therefore, it is also required that the node traffic distributions of an application's all jobs are similar.

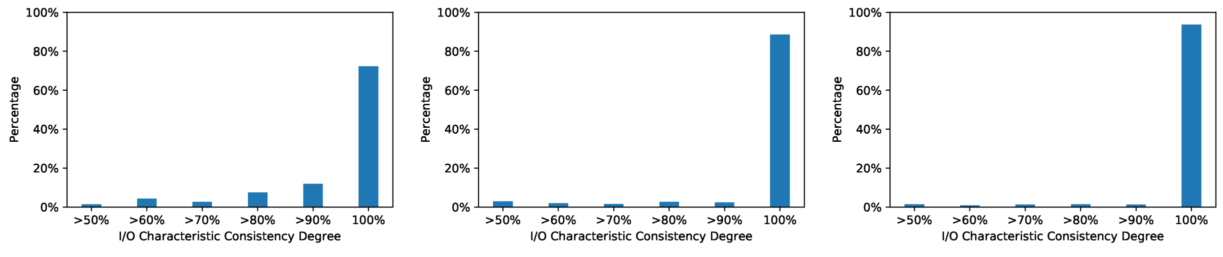
For the first three I/O characteristics, we can calculate whether a job is data-intensive, whether it exhibits all rank I/O, and whether it has low communication traffic with their respective thresholds. They all have only two possible values (yes or no), thus the percent of the 2 values are both between 50% and 100%. We use the larger percent as a measurement of the consistency degree of an application's characteristic. If the consistency degree is small, such as 50%, it means that half jobs of the application have different I/O characteristic with the other half, thus the characteristic is very inconsistent. If the consistency degree is large, such as 100%, it means that all jobs have the same characteristic, thus the characteristic is very consistent in this application. We calculate the I/O characteristic consistency degrees of all applications that have more than 5 jobs. The results are shown in Figure 5. As can be seen from the figure, for all the three I/O characteristics, the vast majority of applications have consistency degrees of 100%. Therefore, they are all very consistent.

For the fourth I/O characteristic whether the jobs' node traffic distributions of an application are similar, we use Euclidean distance to measure their divergence. Assuming an application running on N nodes has M jobs, we treat the M distributions of length N as M points in an N-dimensional space. The Euclidean distance between two points measures the divergence of distributions they represent. We use the average of C_M^2 distances between each two of the M points to represent the divergence of M jobs' node traffic distributions. We calculate the traffic distribution divergences of all applications running on more than 2 nodes. The results are shown in Figure 6(a). In order to more intuitively understand the divergence a value indicates, we select three applications running on two nodes and draw the heat map of their jobs' node traffic distributions in Figure 6(b)(c)(d). As can be seen, when the value is larger than 0.3, the traffic distributions tend to be different. If we choose 0.3 as the threshold, the jobs of about 70.6% applications have similar node traffic distributions.

To sum up, although multiple runs of an application have different parameters or data, for most applications the four I/O characteristics are consistent across their jobs. It is workable to use job logs of an application to guide the node allocation of its new jobs. Even for the 29.4% applications whose jobs have different node traffic distributions and cannot provide distribution reference to their new jobs, if they satisfy the first three requirements of I/O characteristics described in Section 4.2, we can still optimize their new jobs by simply distributing their nodes on more cabinets.

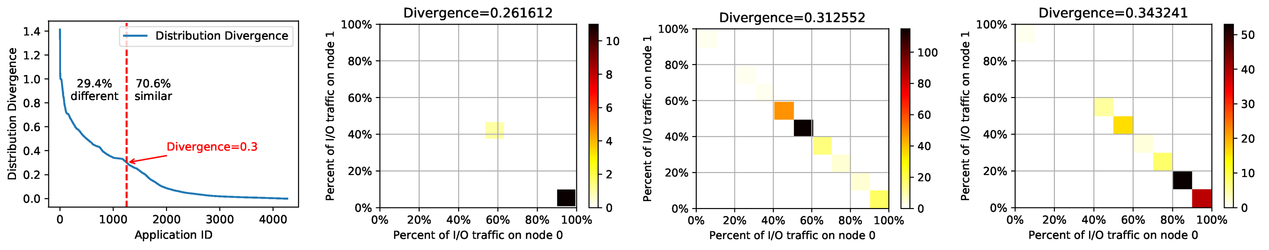
4.4 Implementation of Slurm Plugin

The Slurm scheduler plugin leverages several helper plugins (e.g. network topology-aware plugins) to select appropriate nodes for a job. We implement our node allocation algorithm as a helper plugin and use it to select most appropriate nodes for data-intensive jobs. We analyze the job logs and I/O traces within a time window (e.g. 1 month)



(a) The characteristic whether an application is data-intensive. (b) The characteristic whether an application exhibits all rank I/O pattern. (c) The characteristic whether an application has low ratio of communication traffic to I/O traffic.

Figure 5: The I/O characteristic consistency degrees of all applications.



(a) The divergence values of all applications. (b) The traffic distributions of an application's all jobs. (c) The traffic distributions of an application's all jobs. (d) The traffic distributions of an application's all jobs.

Figure 6: The I/O characteristic whether the node traffic distributions of an application's jobs are similar.

and classify all the jobs into different applications according to their username, job name and number of allocated nodes. Then we determine whether an application can benefit from the optimized strategy by extracting their three I/O characteristics described in Section 4.2. If an application satisfies all the requirements, it is considered to be optimizable and is recorded in the knowledge base. When a new job submission arrives, the plugin look it up in the knowledge base. If it matches an application, then the job will be optimized by our node allocation strategy. For an optimizable application that satisfies the three requirements, we further determine whether all its jobs have similar node traffic distributions. If they are not similar, we will optimize the application's new jobs only by distributing their nodes on more cabinets. If they are similar, the application's average node traffic distribution is further recorded in the knowledge base, which will be later used to distribute the I/O traffic of its new job on its cabinets more evenly.

The core of our node allocation algorithm is to divide a job's nodes into several partitions based on their amounts of I/O traffic and make the total traffic amounts of all partitions as even as possible. Different partitions of nodes will be located on different cabinets so that more I/O nodes will be utilized more evenly to handle the job's I/O traffic. It is a typical equal sum partition problem, except that we have a slightly more relaxed goal that the sums of all partitions only should be about equal. However, the relaxed goal leads to a more complex algorithm because we have to calculate and compare the I/O traffic unevenness degrees of all possible partition plans. Fortunately, there are only thousands of compute nodes and hundreds of cabinets in an HPC system. The problem scale is so small that we can find the optimal partition plan with breadth-first search (BFS) at a very small cost. The algorithm takes the node traffic distribution and the number of expected partitions as inputs. We preselect multiple appropriate expected numbers of partitions based on the number of nodes and run the algorithm with them. After all the partition plans are calculated, we choose a feasible plan based on the available nodes in the system. If there are no appropriate available nodes, then we fall back to the default node allocation algorithm of Slurm.

5 PERFORMANCE EVALUATION

5.1 Testbed

The evaluations are conducted on Tianhe-1A in National Supercomputer Centre in Tianjin. Its compute node has 2 Intel Xeon X5670 CPU and 12 cores. The Lustre storage system we used has 64 object storage servers (OSTs), each of which has a peak bandwidth of around 200 MB/s. Files are striped across all 64 OSTs with the stripe size of 1 MB.

Since there is no I/O forwarding layer on Tianhe-1A, we let compute nodes act as I/O nodes by installing IOFSL [2] software on them. IOFSL is a general-purpose I/O forwarding software designed by ANL and now has been widely used in academia for research purpose. On Tianhe-1A, compute nodes on the same cabinet communicate over cabinet back boards. Compute nodes on different cabinets communicate over 384-port switches. Since I/O nodes are always located close to their compute nodes to minimize network overhead, we locate 1 I/O node on each cabinet, and every 4 compute nodes share an I/O node on the same cabinet. We use at most 8 I/O nodes and 32 compute nodes in the experiments. In order not to impact the production system, we run Slurm with our optimized plugin on a private server. The private Slurm only manages the compute nodes in our experiments and only process our job submissions.

5.2 Synthetic Benchmark

5.2.1 Overall Performance. We develop a synthetic parallel benchmark that its process i accesses $i/\sum_1^N i$ of total amount of data. We test its performance under Slurm's default node allocation strategy, strategy of even node distribution and strategy of even traffic distribution with varying number of processes. Each compute node runs 1 process. The total amount of processed data is 1 TB in each test. The node traffic distribution of the benchmark is manually recorded in the knowledge base in advance.

Figure 7 shows the test results. Since the default strategy allocates nodes on the same cabinet for the benchmark, all the I/O traffic flows through only 1 I/O node, making it heavily loaded. As a result, the performance of default strategy is significantly worse than the optimized strategies. Moreover, its performance soon reaches the limit as the number of compute nodes grows. On the contrary, the two optimized strategies have scalable performances as the number of nodes grows because more I/O nodes are utilized.

The results indicate that even if an application's jobs have very different node traffic distributions and cannot be optimized by the even traffic strategy, its performance can still be promoted significantly by utilizing more I/O nodes. One might think that the significant performance improvements are brought by the ideal experimental environment that the extra I/O nodes we used are completely idle. In fact, the same is mostly true in realistic production environments. As shown in Figure 4, our investigation on Tianhe-1A unveils that the I/O bursts of different cabinets arrive at different time. It is likely that when a cabinet are busying processing a job's I/O burst, other cabinets are only lightly loaded.

Even though the compute nodes are evenly distributed on all cabinets and more I/O nodes are utilized, the load on I/O nodes is still unbalanced. The workload on the busiest I/O node is 2.6× to 12.2× of the idlest I/O node. By allocating nodes according to their different I/O demands, better load balancing can be achieved on the I/O nodes. Therefore, the performance of the even traffic distribution strategy is much better than the strategy of even node distribution. For example, the speedup can reach up to 2 in the test with 32 nodes.

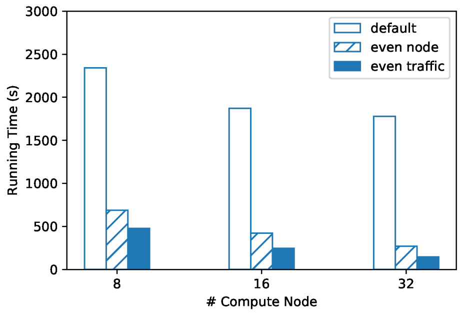


Figure 7: Overall performance of Slurm's default strategy, strategy of even node distribution and strategy of even traffic distribution.

5.2.2 Performance in Terms of the Amount of Communication Traffic. Distributing compute nodes on more cabinets will inevitably increase the overheads of inter-node communication. In order to study such performance impacts, we develop a synthetic benchmark based on the benchmark we used in the last section. In this benchmark, the processes will communicate with each other in an all-to-all manner after accessing data. The all-to-all communication is conducted with an MPI routine `MPI_Alltoall()`. Considering that the network bandwidth is better with larger transfer size, we set the transfer size in the benchmark to a very small value of 32 bytes, so that the increased network overhead of our optimized strategy is as significant as possible. The benchmark runs on 32 compute nodes with different ratios of communication traffic of a single process to the total I/O traffic of the benchmark. The test results are shown in Figure 8.

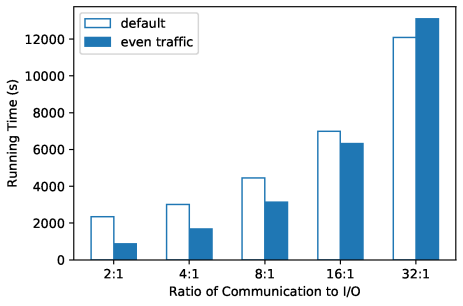


Figure 8: Performance in terms of the ratio of communication traffic of a single process to the total I/O traffic of the benchmark.

Under the default strategy, the communication is more efficient than the optimized strategy since all communication happens within the cabinet. However, the I/O bandwidth difference of the two strategies is much larger than their network bandwidth difference. As a result, the performance of default strategy cannot catch up with the optimized strategy until the ratio of communication to I/O is larger than 16:1.

To be noted that the ratio is calculated with the communication traffic of a single process. If calculating with the total communication traffic of the benchmark's all processes, the ratio would be vastly larger. The results reveal that only applications with communication demands far in excess of their I/O demands cannot benefit from our optimized strategy.

5.3 Seismic Exploration Application

Seismic exploration is an important means to search oil and gas resources before drilling and has been widely used in geological prospecting, geological research, and crustal research. Seismic exploration applications are typical data-intensive applications on Tianhe-1A [11]. We use a seismic exploration application called Gather to evaluate our optimization. When processing every region, each of the Gather processes first reads in shot files, and then extracts and calculates data to obtain the subsurface image. The root process collects the subsurface images from all the processes for reduction and then writes the result to the storage system. It can be seen that Gather application exhibits all rank I/O pattern when reading data, and exhibits rank 0 I/O pattern when writing data. We test the performance of Gather on 8, 16, 32 compute nodes respectively. The results are shown in Figure 9.

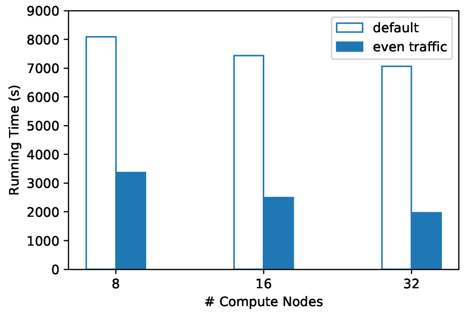


Figure 9: Performance of Gather application.

The results indicate that the performance of optimized strategy outperforms the default strategy in all compute node scales. Although it uses only one process to write results, the size of output data is very small compared to the input data. Therefore, Gather application mainly exhibits all rank I/O pattern, which can be accelerated by the optimized strategy. In addition, since more I/O nodes are utilized under the optimized strategy, the I/O forwarding capability is much higher and less likely to reach the limit, resulting better scalability of Gather application.

5.4 WRF

The Weather Research and Forecasting (WRF) Model is a numerical weather prediction system that has been widely used in both atmospheric research and operational forecast. In a typical run, WRF application reads in the input data, conducts weather simulation, writes checkpoints during execution, and writes the simulation results in the end. Its all processes perform I/O operations if PnetCDF library is enabled. Therefore, it is a typical data-intensive application exhibiting all rank I/O pattern. In the test we run WRF on 8, 16, 32 compute nodes respectively.

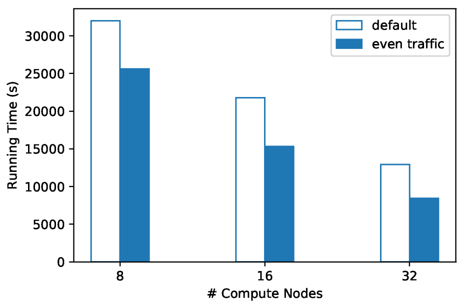


Figure 10: Performance of WRF application.

As can been seen from the results in Figure 10, as more compute nodes are used to undertake simulation, the performances of both strategies are getting better. However, the optimized strategy outperforms the default strategy in all numbers of compute nodes because more I/O nodes are utilized more evenly. Note that, the reason why their performance difference isn't getting significantly larger as the number of nodes grows is that the time-consuming simulation makes the benefits of utilizing more I/O nodes not so relevant.

6 RELATED WORK

A large body of research on node allocation strategy of large-scale HPC systems has been devoted to reducing the network overhead for communication between compute nodes. Slurm has adopted several network topology-aware plugins to place job's nodes close in the network. Researchers propose to further take dynamic network topology information and the over-subscription of links into account [16]. Above work optimizes node allocation solely based on network topology. Georgiou et al. [5] introduce a new method that takes account of both the topology and application communication pattern. However, they do not delve into the communication pattern differences among an application's multiple runs. In this paper, we extract job I/O characteristics from realistic job logs and I/O traces and validate their consistency before using them to guide the node allocation of new jobs.

There are some studies that are focused on node allocation strategy taking account of the job's I/O. In systems that compute nodes are equipped with local storage, a job's running time is significantly affected by whether it is co-located with its files. Therefore, Zhang et al. [19] propose to extend gang scheduling by adding different levels of I/O awareness. Qin et al. [13] propose an I/O-aware load balancing schemes for heterogeneous clusters that are comprised of a variety of nodes with different disk I/O speed. Recently Herbein et al. [7] propose batch job scheduling techniques that reduce I/O contention by modeling the available bandwidth of links between each level of the storage architecture. These approaches differ from ours in that, they try to effectively map jobs on nodes with different I/O capabilities, while we try to eliminate the I/O capability difference of I/O nodes by balancing the load on them.

Some research is also on balancing the load on I/O nodes. Cray DataWarp [6] is a fast data staging area comprised of burst nodes with fast storage. DataWarp strips data across multiple nodes for better load balance. Yu et al. [18] propose a cross-layer I/O coordination between compute nodes and I/O nodes. Each compute node accesses data from multiple I/O nodes instead of only one so that the load on the I/O nodes can be more balanced. This approach's drawback is that each job can only balance the load on the I/O nodes it uses. Our approach balance the load on all I/O nodes in the system by distributing jobs' I/O traffic on more cabinets more evenly.

7 CONCLUSION

Based on the analysis of I/O traces collected from Tianhe-1A, we find that the tremendous amount of I/O traffic handled by storage systems not only exists in short periods of time, but also comes from a minority of compute nodes that located close to each other. The caused spatially bursty I/O makes the load on I/O nodes highly unbalanced. We quantitatively analyze the two main reasons of spatially bursty I/O. The first reason is the uneven traffic distribution on processes caused by job's intrinsic characteristics. We find that 25.7% multi-nodes jobs only use one node to perform most of their I/O, and 38.1% of the rest jobs have uneven I/O traffic on their nodes. The second reason is the uneven node distribution on cabinets caused by Slurm's node allocation strategy. We find that 50.7% multi-cabinets jobs have uneven node distribution on cabinets. We propose a node allocation strategy that takes account of job's I/O traffic distribution on nodes, so that more I/O nodes can be used by jobs more evenly. The evaluations on Tianhe-1A show that our approach can further exploit the potential of I/O forwarding layer and significantly promote the performance of multiple synthetic benchmarks and realistic applications.

In the future work, we plan to study an alternative approach to address the load imbalance problem by changing the I/O node to compute node mapping and redirecting I/O traffic to remote I/O nodes.

ACKNOWLEDGMENTS

This work was supported by the National Key Research and Development Program of China (Grant No. 2016YFB0200402 and 2017YFB0202204), National Natural Science Foundation of China (Grant No. 61502511), and the China National Special Fund for Public Welfare (Grant No. GYHY 201306003).

REFERENCES

[1] Boyle P A. 2012. The Bluegene/Q Supercomputer. *PoS* 020 (2012).

[2] Nawab Ali, Philip Carns, Kamil Iskra, Dries Kimpe, Samuel Lang, Robert Latham, Robert Ross, Lee Ward, and P. Sadayappan. 2009. Scalable I/O forwarding framework for high-performance computing systems. In *IEEE International Conference on CLUSTER Computing and Workshops*. 1–10.

[3] Katie Antypas, Nicholas Wright, Nicholas P Cardo, Allison Andrews, and Matthew Cordery. 2014. Cori: a cray xc pre-exascale system for nersc. *Cray User Group Proceedings. Cray* (2014).

[4] Philip Carns, Kevin Harms, William Allcock, Charles Bacon, Samuel Lang, Robert Latham, and Robert Ross. 2011. Understanding and improving computational science storage access through continuous characterization. *ACM Transactions on Storage (TOS)* 7, 3 (2011), 8.

[5] Yiannis Georgiou, Emmanuel Jeannot, Guillaume Mercier, Adle Villiermet, Yiannis Georgiou, Emmanuel Jeannot, Guillaume Mercier, Adle Villiermet, Yiannis Georgiou, and Emmanuel Jeannot. 2017. Topology-aware job mapping. *International Journal of High Performance Computing Applications* (2017), 109434201772706.

[6] Landsteiner B Henseler D and Petesch D. 2016. Architecture and Design of Cray DataWarp. In *Proc. Cray Users' Group Technical Conference (CUG)*.

[7] Stephen Herbein, H. Ahn Dong, Don Lipari, Thomas R. W. Scogland, Marc Stearman, Mark Grondona, Jim Garlick, Becky Springmeyer, and Michela Taufer. 2016. Scalable I/O-Aware Job Scheduling for Burst Buffer Enabled HPC Clusters. In *ACM International Symposium on High-Performance Parallel and Distributed Computing*. 69–80.

[8] Youngjae Kim, Raghul Gunasekaran, Galen M Shipman, David A Dillow, Zhe Zhang, and Bradley W Settlemyer. 2010. Workload characterization of a leadership class storage cluster. In *Petascale Data Storage Workshop (PDSW), 2010 5th*. IEEE, 1–5.

[9] Quincey Koziol et al. 2014. *High performance parallel I/O*. CRC Press.

[10] Xiangke Liao, Liquan Xiao, Canqun Yang, and Yutong Lu. 2014. MilkyWay-2 supercomputer: system and application. *Frontiers of Computer Science Selected Publications from Chinese Universities* 8, 3 (2014), 345–356.

[11] Xin Liu, Yu-tong Lu, Jie Yu, Peng-fei Wang, Jie-ting Wu, and Ying Lu. 2017. ONFS: a hierarchical hybrid file system based on memory, SSD, and HDD for high performance computers. *Frontiers of Information Technology & Electronic Engineering* 18, 12 (2017), 1940–1971.

[12] Yang Liu, Raghul Gunasekaran, Xiaosong Ma, and Sudharshan S. Vazhkudai. 2016. Server-Side Log Data Analytics for I/O Workload Characterization and Coordination on Large Shared Storage Systems. In *International Conference for High PERFORMANCE Computing, Networking, Storage and Analysis*. 70.

[13] Xiao Qin, Hong Jiang, Adam Manzanares, Xiaojun Ruan, and Shu Yin. 2009. Dynamic load balancing for I/O-intensive applications on clusters. *Acm Transactions on Storage* 5, 3 (2009), 1–38.

[14] Stephan Schlagkamp, Rafael Ferreira da Silva, William Allcock, Ewa Deelman, and Uwe Schwiegelshohn. 2016. Consecutive job submission behavior at Mira supercomputer. In *Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing*. ACM, 93–96.

[15] Claude E Shannon. 2001. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review* 5, 1 (2001), 3–55.

[16] H. Subramoni, D. Bureddy, K. Kandalla, K. Schulz, B. Barth, J. Perkins, M. Arnold, and D. K. Panda. 2014. Design of network topology aware scheduling services for large InfiniBand clusters. In *IEEE International Conference on CLUSTER Computing*. 1–8.

[17] TOP500. 2017. TOP500 Supercomputer Sites. http://www.top500.org. (2017).

[18] Jie Yu, Guangming Liu, Xiaoyong Li, Wenrui Dong, and Qiong Li. 2017. Cross-layer coordination in the I/O software stack of extreme-scale systems. *Concurrency & Computation Practice & Experience* (2017).

[19] Yanyong Zhang, Antony Yang, Anand Sivasubramaniam, and Jose Moreira. 2007. Gang Scheduling Extensions for I/O Intensive Workloads. In *Job Scheduling Strategies for Parallel Processing, International Workshop, Jsspp 2003, Seattle, Wa, Usa, June 24, 2003, Revised Papers*. 183–207.