The Mystery of the Failing Jobs: Insights from Operational Data from Two University-Wide Computing Systems

Rakesh Kumar[1,2], Saurabh Jha[3], Ashraf Mahgoub[2], Rajesh Kalyanam[2], Stephen L Harrell[2], Xiaohui Carol Song[2], Zbigniew Kalbarczyk[3], William T Kramer[3], Ravishankar K Iyer[3], Saurabh Bagchi[2]

1: Microsoft, rakku@microsoft.com; 2: Purdue University, {amahgoub, rkalyana, slh, cxsong, sbagchi}@purdue.edu;
3: University of Illinois at Urbana-Champaign, {sjha8, kalbarcz, wtkramer, rkiyer}@illinois.edu

***Abstract*—Node downtime and failed jobs in a computing cluster translate into wasted resources and user dissatisfaction. Therefore understanding why nodes and jobs fail in HPC clusters is essential. This paper provides analyses of node and job failures in two university-wide computing clusters at two Tier I US research universities. We analyzed approximately 3.0M job execution data of System A and 2.2M of System B with data sources coming from accounting logs, resource usage for all primary local and remote resources (memory, IO, network), and node failure data. We observe different kinds of correlations of failures with resource usages and propose a job failure prediction model to trigger event-driven checkpointing and avoid wasted work. Additionally, we present user history based resource usage and runtime prediction models. These models have the potential to avoid system related issues such as contention, and improve quality of service such as lower mean queue time, if their predictions are used to make a more informed scheduling decision. As a proof of concept, we simulate an easy backfill scheduler to use predictions of one of these models, i.e., runtime and show the improvements in terms of lower mean queue time. Arising out of these observations, we provide generalizable insights for cluster management to improve reliability, such as, for some execution environments local contention dominates, while for others system-wide contention dominates.**

***Index Terms*—HPC, Production failure data, Data analytics, Compute clusters**

I. Introduction

"The phoenix must burn to emerge."

Janet Fitch

Large-scale high performance computing (HPC) systems have become common in academic, industrial, and government for compute-intensive applications, including large-scale parallel applications. These HPC systems solve problems that would take millennia on personal computers, but managing such large shared resources can be challenging and requires administrators to balance requirements from a diverse set of users. Large, focused organizations can afford to buy centralized resources, and choose to manage and operate it at academic organizations through a central IT organization. These are funded by federal funding agencies (like the National Science Foundation in the US) and individual researchers write grant proposals to get access to compute time on these systems. Examples of such systems include Comet at the University of California San Diego, Blue Waters at the University of Illinois at Urbana-Champaign, and Frontera at the University of Texas at Austin.

TABLE I: Summary of data analyzed (all production jobs) for the two university-wide clusters. The percentages in parentheses refer to the raw counts and node seconds. Sharing allows multiple jobs to run on the same node.

Computing Cluster		*System A*	*System B*
Duration		Mar 2015-Jun 2017	Feb-June 2017
# jobs		2,908k	2,219k
shared	# single	1,125k (38.7%, 15.8%)	-
	# multi	28k (1.0%, 1.9%)	-
	total	1,153k (39.7%, 17.7%)	-
non-shared	# single	1,348k (46.3%, 18.4%)	1,640k (73.9%, 5.4%)
	# multi	407k (14.0%, 63.9%)	580k (26.1%, 94.6 %)
	total	1,755k (60.3%, 82.3%)	2,219k (100%)
# unique users		617	467

Another trend in many universities' IT acquisition is the adoption of the *community cluster model*. Here, research groups buy assets (nodes and other hardware) in a central computing cluster, which is then assembled and managed by the central IT organization. These clusters have flexible usage policies, such that partners in a community cluster have ready access to the capacity they purchase, but they can use more resources when other groups' nodes are unused. This allows for opportunistic use for the end users and higher resource utilization for the cluster managers. System administrators take care of security patches, software installation and upgrades, and hardware repair, as well as space and cooling requirements. The community cluster model has become a foundation of the research cyber-infrastructure at many universities. For example, Purdue has run such a program since 2006 with 10 generations of clusters to date and in 2018 they provided 431M CPU hours. This model is also being successfully used at the Universities of Rochester, Delaware, and Texas at Austin.

This paper studies the reliability of jobs that run on two clusters that follow the two operational models introduced above. Our analyses are based on two centrally managed computing clusters called *System A* and *System B*, at Purdue University and University of Illinois at Urbana-Champaign

978-1-7281-5809-9/20/$31.00 ©2020 IEEE
DOI 10.1109/DSN48063.2020.00034

TABLE II: Key observations and recommendations for the two university-wide computing clusters, **System A** *and* **System B.** *Wherever possible, we separate the recommendation for cluster provider* **(P)** *and user* **(U).**

Observations	Recommendations
O1: Although *System A* has a local IO capacity of 100MB/s, local IO-related failure rate starts rising with utilization as low as 3-6MB/s. Similarly, the remote IO capacity of *System B* is 1.1TB/s while remote IO-related failures are observed with a utilization of only 46MB/s for a given job.[**Fig 4, 5**]	**R1**: **(P)** With in-depth analysis, we notice that the majority of the jobs are sending random access requests instead of sequential. Expectedly, we do observe the local IO threshold in case of shared jobs (3 MB/s) to be less when compared with non-shared jobs (6 MB/s) for *System A*. Accordingly, both local storage and network file system reaches saturation with much less utilization than expected. Online monitoring of IO utilization is required to identify contention thresholds and hence take proactive remedy decisions.
O2: Similar to prior studies, a linearly increasing relationship is observed between job failure rates and job runtime durations in *System B*. However, *System A* shows the exact opposite trend. [**Fig 6**]	**R2**: **(P)** Our analysis shows that the majority of failed jobs in *System A* are due to newly submitted jobs that fail due to startup issues (like making huge system resources demand). On the other hand, users of *System B* submit codes that are more mature and the long-running jobs are exposed to more faults in space, time or both. For *System B*, we recommend using job runtime duration as a feature while predicting job failure probabilities, and hence identify optimal checkpointing frequencies (as we propose in section VI-A) **(U)** For *System A*, system admins can encourage new users to test their jobs on test environments and overcome any startup issues before running on the full-scale cluster.
O3: A significant portion of jobs fail with out-of-memory exceptions, even in cases where free memory of more than 10GB is available on a node (*System A*). Moreover, the memory utilization is less than 65% of the available capacity in 50% of the failed jobs [**Sec V-A**]	**R3**: **(P)** (i) Use user history based memory usage prediction model (Sec. V-A) while making scheduling decisions for the shared jobs. (ii) Monitor and start taking preemptive measures when the application gets close to the memory capacity [43].
O4: Although *System B* is 43X larger in scale compared to *System A*, system issues are responsible for over 53% of the failed jobs in *System A*, while it is only 4% for *System B*. Moreover, sharing a node does *not* increase the fraction of jobs failing due to system issues. Additionally, both *System A* and *System B* have significant user-related failures (33% and 48% of all failures). [**Table IV**]	**R4**: **(P)** (i) This analysis measures quantitatively how beneficial it is to use resiliency features in HPC (such as *System B*) and dedicated system administrators for the particular cluster. (ii) Monitor resource usages of the jobs and take proactive actions when resource exhaustion is being approached such as in [50], [61]. (iii) Use failure prediction model like ours (Sec. VI-A) to dynamically change checkpointing frequency with failure probability to complement current optimal periodic checkpointing techniques [15], [47]. **(U)** User-related job failures: (i) Use static analysis tools for checking errors in job submission scripts, environment setup, and user code [9], [53], [81]. (ii) Use small scale testing *e.g.*, using containers before submitting to a large cluster.
O5: Contention for remote resources with executing elements outside the node is dominant in non-shared environment, while the contention with other jobs executing on the same node is dominant for a shared environment. [**Sec V-B, V-C, V-D**]	**R5**: **(P)** (i) Use user-based resource usage prediction while making scheduling decisions (Sec V-C,V-D). (ii) Adopt resource isolation technologies (such as containers) for shared environment, to reduce failures due to local contention. **(U)** (i) Use dynamic reconfiguration of applications based on current resource availability [50], such as reconfiguring the number of threads or network timeout.
O6: A significant fraction of total compute resources are used by jobs that hit against the walltime for both *System A* (33%) and *System B* (43%). [**Table IV**]	**R6**: Significant loss of work can happen when the program is terminated upon hitting walltime. **(P)** (i) Like System B, provide extra cycles to enable an application to take a checkpoint when job termination is signaled due to walltime. (ii) For long-running jobs, perform checkpointing with dynamically varying frequency, say using our failure prediction model (Sec. VI-A).

respectively. The details of the source data are provided in Table I. For *System A*, which comprises 4,640 cores and 1,160 Xeon Phi acceleators, we consider a total of 3.0M jobs over a period of 28 months (March 2015–June 2017). We have released the entire data used in the analyses in this paper into an open source repository as part of an ongoing NSF project [10]. For *System B*, which consists of 396,000 CPU cores and 4,229 GPU accelerators, we analyze about 2.2M jobs over a period of 5 months (February–June 2017) of which approximately 26% are multi-node jobs, including some which are very large in execution scale reaching up to 358,400 cores. A subset of the data for *System B* is available at [38]. This dataset represents the most comprehensive one in terms of variety of data sources analyzed publicly to date for failure characteristics, whether of a university compute infrastructure or otherwise. The paper performs 3 different categories of analysis—examination of failure root causes through job exit status codes, likelihood of failure with resource usage for both local (memory, local IO) and remote (remote IO, network) resources, and effect of job runtime on failures. We use these analyses to drive two actionable decisions—changing the checkpoint frequency and scheduling jobs through backfilling.

New insights and old insights in new environments. The analyses in this paper shed new light relative to prior studies of system usage and failures in large-scale computing clusters in the following ways. We present in Table II, the key observations and the implications for administering central compute clusters with general-purpose needs. *First*, our paper

looks at two acquisition and operation models for research computing clusters at two large universities. The heterogeneity of jobs and the expertise level of the users together with the relatively smaller size of the IT system administration staff for maintaining such clusters have important implications for job reliability. For example, the continuous node reachability in this environment is lower than reported in prior studies of focused, dedicated computing clusters, such as, US Department of Energy-run supercomputing clusters [11], [67] or highly instrumented and highly managed cloud clusters [64], [78]. *Second*, we categorize jobs into 5 different categories based on their exit codes. A number of prior works have done similar categorization into 3 categories [8], [21]. We go one step further and split failed jobs category into 3 further categories, i.e., user-related, system-related or user/system-related (or indeterminate) failures (Sec. IV). This is important because the mitigations are likely to be different for these categories. *Third*, we consider fine-grained system usage data for the different resources, local to a node as well as remote, and identify their implication for job failures. In some cases, we see increased job failure rates due to local contention (such as, for memory on a node) while in some cases we see the effect of congestion for remote resources (such as, for networking bandwidth and parallel file system for non-shared jobs in *System A*). For some cases, there is no correlation found (such as, memory-related failures in multi-node jobs). Taken in totality, our analyses indicate which job categories (single-node vs multi-node, shared vs non-shared) put contention on which kinds of resources, and correspondingly at what quantitative level, to the point of increasing job failure rates. This can directly feed back into the acquisition and upgrade decisions made by IT staff. More coarse-grained data and analysis, such as, aggregate job failure rate [67], coarse-grained resource utilization metrics [12], [64], or the effect simply of the execution time of the job on its failure probability [14], [19], cannot shed such detailed light. Additionally, we use user historical usage information to predict resource usages of jobs currently in queue, *i.e.*, even before a job starts executing. We show how the prediction helps a backfilling-based scheduler to improve cluster utilization. Finally, we build a failure prediction model based on resource usages, which triggers checkpointing when the likelihood of failure is high.

The paper is structured as follows. Section II provides details of the two systems while Section III describes the data sources. In Section IV, we analyze job failure categories and in Section V, we analyze the impact on job failures of resource usages, and present resource usage prediction models. Finally, Section VI shows the applications of failure prediction and runtime prediction models. We then discuss threats to validity, related work and conclude the paper.

II. System Details

Table III provides system specification of two university-based HPC systems. *System A* is hosted at *Purdue University* and *System B* is hosted at *University of Illinois at Urbana-Champaign*.

TABLE III: System Details

Unit	*System A*	*System B*
Compute	580 nodes, 64GB/node, Xeon E52670 + 2x Xeon Phi/node, ECC-protected CPU & Xeon Phi memory	22,636XE (CPU only) and 4,228XK (CPU+GPU) nodes, 64GB/node, AMD 6276 Interlagos, NVIDIA GK110, Chipkill-protected CPU memory modules, ECC-protected GPU memory modules
Local IO	500 GB (SATA), 100MB/s	Not present
Network IO	1.4PB, 23GB/s, Data Disks: RAID 6, Index Disks: RAID 1+0, OSS : Active-Active HA pair, MDS: Active-Passive HA pair	26.4PB, 1.1TB/s, Data Disks: RAID 6, Index Disks: RAID 1+0, OSS : Active-Active HA pair, MDS: Active-Passive HA pair
Network	5GB/s, Fat Tree, Infiniband Forward Error Correction (FEC)	9.6GB/s, Cray Gemini 3D Torus [7], Packet: 16-bit packet CRC, links: adaptive load balancing, routers: quiesce and reroute

Job Submission System: In *System A*, jobs can be flagged by their submitters as shared or non-shared; the former means that the job can be executed together with other jobs on the same node. In *System B*, all jobs execute in non-shared mode, without any co-location with another job on the same node. *System B* puts a 48-hour walltime restriction, whereas *System A* has a restriction of 336 hours. A job termination status (exit code) is captured by the TORQUE [71] log in both systems, while for *System B*, ALPS [39] also captures the exit code of each application within a job.

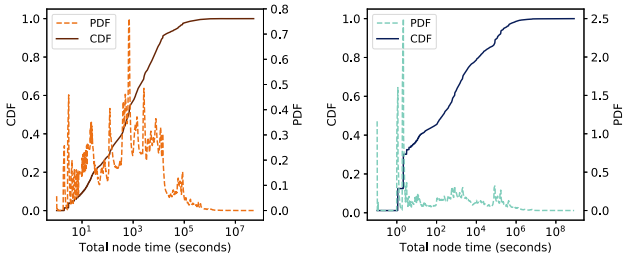
Job Characteristics: This section compares and contrasts the job characteristics of *System A* and *System B* in terms of i) job node-seconds and ii) job size.

1) **Job Node-seconds:** It is the product of the number of nodes and the wallclock time (in seconds) for which the job executes. On both *System A* and *System B*, 50% of the jobs run for less than $\sim 10^3$ node-seconds (i.e., less than 16 minutes), however on *System B* the jobs run up to $\sim 10^9$ node-seconds i.e., more than 1 year (refer Fig. 1a).
2) **Job Size:** Most of the jobs on *System A* and *System B* are single-node jobs, 85% and 74% respectively. However, these jobs contribute just 34% (*System A*) and 5% (*System B*) by node-seconds. Furthermore, scale of jobs submitted on *System B* is significantly larger than *System A* where more than 20% of the jobs (by node-seconds) execute on 2K nodes or more (refer Fig. 1b).

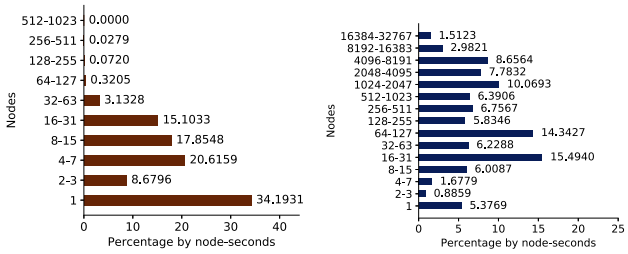
III. Description of Data

The data used in this paper falls under three broad categories: job scheduler accounting logs, job-level resource utilization logs, and node-level health monitoring logs.

Job Accounting: Both *System A* and *System B* use TORQUE [71]. These records contain the event being recorded (e.g., queuing, job start, job end), corresponding timestamps, the submitting user or group, and resources requested and used. For *System A*, we processed the raw TORQUE logs,



(a) Comparing job node-seconds (#nodes × wallclock time)



(b) Comparing percentage by node-seconds

Fig. 1: Characteristics of jobs on* System A *(in brown, left in each subfigure) and* System B *(in blue, right in each subfigure).

however *System B* uses Integrated System Console (ISC) [28] to parse and store the job records and its associated metrics (performance and failure) in its database.

Resource Utilization Stats: *System A* uses TACC Stats [23] and *System B* uses light-weight distributed metric service (LDMS) [6] for collecting resource utilization values. TACC stat on *System A* is configured to collect data for each node at 5-minute granularity, whereas LDMS is configured at 1-minute granularity on *System B*.

Node Failure Reports: *System A* maintains a record of planned system outages, reboots, or alerts on unreachable nodes from the Sensu [5] and Nagios [4] monitoring frameworks. *System B* uses ISC [28] for recording in-depth information about each job and node of the system.

IV. Job Categories based on Exit Statuses

In this analysis, we use the exit code information to find the probable job failure cause and categorize them based on the approach used in LogDiver [51]. On a TORQUE-based system, a job upon termination returns an exit code in the range -11 to 271. A successful job has exit status of 0 and any other exit status can be either walltime or denotes unsuccessful job termination, with each exit status value representing a different error.

TABLE IV: Job categories based on exit codes. Percentages in brackets are based on the total node-seconds

		Environment & Job Type				
		shared		non-shared		overall
	Category	single	multi	single	multi	
System A	Success	93.1%	87.6%	87.6%	61.8 %	86.1% (48.4%)
	System	2.7%	6.5%	6.5%	8.8 %	5.3% (4.0%)
	User	1.6%	2.2%	3.5%	7.2%	3.3%(12.9%)
	User/System	0.6%	0.2%	0.4%	6.1%	1.3% (1.3%)
	Walltime	2.0%	3.5%	2.0%	16.1%	4.0% (33.4%)
	Total	1,125k	28k	1,348k	407k	2,908k
System B	Success	-	-	91.6%	64.0%	84.4% (44.4 %)
	System	-	-	0.10%	1.0%	0.3% (1.4%)
	User	-	-	3.8%	3.0%	3.6% (2.7%)
	User/System	-	-	1.2%	0.8%	3.6% (8.0%)
	Walltime	-	-	3.7%	20.4%	8.0% (43.4%)
	Total	-	-	1,640k	579k	2,219k

Exit reasons are classified into the following categories: (i) *Success*, for applications completing without any errors, (ii) *Walltime*, for applications not completing within the allocated wall clocktime, (iii) *User*, for applications that fail due to issues that originate from the submitter of the job or the developer of the code. These include mis-configuration of job script or compilation/execution environment setup, job user action (such as a control-C signal), command errors, missing module/file/directory, and wrong permissions, (iv) *System*, where an application is terminated due to system hardware or software errors, and (v) *User/System*, when it is not possible to disambiguate whether the error occurred due to system or user issues, *e.g.*, SIGTERM signal can be issued both by user (through asssertions) or scheduler (on failing health check). A job exiting due to walltime does not necessarily mean loss of production hours. Most of the jobs (especially large-scale jobs) depend on checkpoint-restart mechanisms to start from previously checkpointed state. On *System B*, developers can trap the kill signal issued by the scheduler on expiry of requested walltime and write a checkpoint file before exiting. Since, there is no information available to us about application-level checkpointing events, we ignore walltime jobs from the rest of the study as we cannot disambiguate jobs that wrote a checkpoint before being terminated (good case and little work is lost) from the jobs that did not.

Table IV shows the category distribution for both the systems for different execution environments (shared or non-shared) and job types (single or multi). On *System A*, a significant number of jobs failed due to system related errors (5.3%) whereas most common failure category on *System B* is user (3.6%). Only 61.8% (*System A*) and 64.0% (*System B*) of the multi-node jobs in non-shared environment completed successfully. On *System A* a higher 87.6% of the *shared* multi-node jobs completed successfully but their number is too small (1.0% overall) to draw any conclusion. Walltime category jobs contributed to 33% and 43% of total compute hours for *System A* and *System B* respectively. We find empirically through a sub-sampled set that most of these jobs checkpoint frequently. However, even here, there is possible loss of computation due to the time gap between the last checkpoint and program termination by the scheduler.

To investigate further, we also studied the node downtime and uptime distribution using the node failure reports maintained by admins of these systems. We observed *System B* has lower continuous downtime (95th percentile value: 24 hours for *System A* vs 20 hours for *System B*) and higher continuous uptime (95th percentile value: 31 days for *System A* vs 92 days for *System B*).

Implications for System Design: *We find that on the smaller scale system (*System A*), job failures caused by system issues are more frequent (53% for* System A *vs 4% for* System B *of all failures). This is because* System B *is more reliable*

by design. It uses expensive Cray HPC solutions and has better resiliency features compared to System A *(such as use of chipkill-enabled memory modules over ECC). Additionally, unlike* System A*,* System B *has dedicated maintenance staff who monitor and manage system on a daily basis.*

V. Effect of Resource Usages on Job Failures

The section studies the influence of resource usage on job failure due to system errors. We consider the 5 primary kinds of resources, both local and remote, namely, memory, local and remote IO, network, and job node-seconds. We calculate failure rate after removing all debug as well as walltimed jobs. We remove the walltimed jobs because these are not considered failed jobs and debug jobs are not representative of production workloads. We define **job failure rate** as the fraction of jobs that fail due to system-related issues. Here we focus on system-related issues to reveal any deficiencies in the underlying system architecture. Examples are: insufficient main memory per node, unsuitable file system, slow remote file system, or insufficient network speed. The purpose of the analysis is to highlight which modules in the system architecture is causing job failures and hence requires administrative modifications. User-related errors happen due to factors that have no discernible pattern, such as, correctness bugs or misconfigurations in the user code. Hence, we do not consider job failures due to user or user/system issues.

Tail-usage: *It is the total amount of resource consumed (or rate of resource consumption for I/O and network) by the job in the* last *measurement window before failure. For* System A *measurement window is five minutes whereas for* System B *the measurement window is one minute.*

Errors in high-speed HPC systems quickly propagate through the system, hence resource utilization values shortly before application failure provide a better understanding of the failure reason than resource utilization throughout the application run. We empirically validate that the job failure rate is correlated with the tail resource utilization rather than aggregate resource utilization (if there is any relation between that resource and failures). Therefore, in this section, all analyses are conducted using tail resource utilization values.

Since resource usages can vary widely across jobs, for all the analyses in this section, we first define equal-sized bins across the range of given resource usages. Jobs are grouped based on the bin's resource usage ranges and then the failure rate of each bin is calculated as the fraction of jobs that failed in that bin. For all analyses here, we only consider data points or bins that have 100 or more jobs, in order to maintain the statistical significance of the results. Additionally, job count is included on the right y-axis to indicate the relative confidence level of all the data points. Job count is equal to the number of jobs considered while computing the failure rate of a given bin. Higher the number of jobs, more accurate the failure rate is. For resources such as local I/O, network I/O and network, the monitoring tool collects the read (receive for network) and write (transmit for network) data separately. We derive the total I/O rate of a node by aggregating these read and write rates. Since the rate range can vary anywhere from 0 to a very high value (23GB/s and 1.1 TB/s for network file system I/O for *System A* and *System B* respectively), we map these rates to a log, base 10, scale.

We do two-sided, t-test based hypothesis testing for all correlation results in this section. The null hypothesis is of the form "Job failure rate is *not* correlated with resource usage of resource X". So if the null hypothesis is rejected, then we can conclude that resource usage of resource X is indeed implicated with job failures. The results of all the hypothesis tests are given in Table V. When the null hypothesis is rejected, in some cases, the failure rate is positively correlated while in others, it is negatively correlated. We remove plots for inconclusive results to save space. Each plot has at the top right a mark designating positive correlation ("+"), negative correlation ("-"), or no statistically significant correlation ("0"). We do not present the analysis for multi-node shared jobs for *System A* since their number is too small to draw any statistically significant conclusions. Wherever applicable, we model the failure rate plot using the best-fit statistical distribution and report the R^2 value. Even where R^2 is low, if the hypothesis testing is significant, then the effect is validated.

Prediction Models: Each section also includes resource usage prediction models for *System A* (combined results for both shared and non-shared), where the measure being predicted is the average resource usage per node during the lifetime of the job. We are omitting the results for *System B* due to space constraints. Similar to observations in prior works [68], [69], [75]), jobs submitted by the same user tend to show strong patterns. Accordingly, future resource usages for a certain user's job can be predicted by profiling previously submitted jobs by that user. The prediction models presented in this section basically are of 4 different kinds: (i) ***Last (L)*** - a naïve model which estimates the resource usage as the resource usage of last finished job of a given user, (ii) ***Average (A)*** - model which estimates the resource usage as the average of resource usages of last n finished jobs of a given user, (iii) ***Median (M)*** - model which estimates the resource usage as the median of resource usages of last n finished jobs of a given user, and (iv) ***Maximum Cosine Similarity (MCS)*** - model which estimates the resource usage as the resource usage of job which is most similar to the current job from the same user. For cosine similarity computation, we use 5 different attributes of a job such as jobname (1 if it is same as current job's jobname else 0), queue (1 if it is same as current job's queue else 0), number of nodes requested (normalized in the range [0,1]), walltime requested (normalized in the range [0,1]) and difference in submit time (normalized in the range [0,1]). We define **history length** as the number of last n finished jobs of a user to consider while doing prediction. For the last three kinds of models i.e., Average, Median and MCS, we find optimal history length on the training dataset (70% of total) and present the results (refer Table VI) with that optimal history length used in the model applied to the test dataset (30% of total). For the results, the best history length is added after the model name, thus M19 means the Median model with

TABLE V: Hypotheses results containing the Pearson correlation coefficients and their corresponding p-values. Null Hypothesis ($\alpha > 0.01$): Failure rate is not correlated with resource usage of resource X. All rejected null hypotheses are in either green or red. Green represents positive correlation while red represents negative correlation

	System A			*System B*		Ref.
	non-shared		shared	non-shared		
	single	multi	single	single	multi	
H_0^1: Memory	0.83, 1.7e-28	0.17, 0.4	0.84, 7.2e-32	0.57, 3.2e-9	0.13, 0.2	V-A
H_0^2: Local I/O	-0.41, 2.0e-4	0.12, 0.4	0.15, 0.2	-	-	V-B
H_0^3: Network I/O	-0.55, 3.3e-19	-0.57, 2.6e-11	0.42, 1.8e-7	-0.07, 0.4	0.45, 2.6e-6	V-C
H_0^4: Network	-0.31, 7.0e-7	0.11, 0.1	0.40, 1.0e-9	-0.21, 0.04	-0.56, 2.5e-9	V-D
H_0^5: Node-seconds	-0.36, 9.8e-5	-0.25, 0.01	-0.31, 1.1e-3	0.04, 0.6	0.42, 2.1e-5	V-E

TABLE VI: Median absolute percentage error (MAPE) of resource usage and runtime predictors for* System A *on the test set. Best history length as determined on the training set is given in parenthesis. Absolute percentage error, $APE = \frac{Actual - Predicted}{Actual} \times 100$.

Prediction	Last	Average	Median	MCS
Memory	3.0%	3.0% (1)	2.4% (19)	**2.2%** (20)
Local IO	31.9%	33.1% (2)	**23.1%** (19)	29.5% (10)
Network IO	31.8%	31.8% (1)	26.6% (17)	**22.3%** (13)
Network	17.7%	17.7% (1)	13.5% (17)	**13.0%** (11)
Runtime	16.7%	16.7% (1)	**13.3%** (9)	14.5% (16)

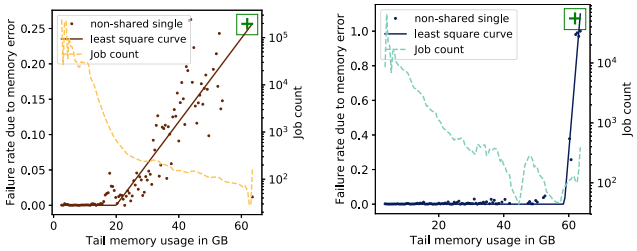
history length of 19.

A. Memory

1) Relation of job failure with memory usage:: Memory related errors are common among failed jobs. In case of memory, we know that exit code 137 corresponds to OOM (out of memory) error. Therefore, we study the likelihood of failure due to OOM error for different ranges of memory use. Here the job failure rate is the fraction of jobs that fail with OOM error. By memory use, we capture the entire memory used on the node including use by system-level processes and all user-level processes (corresponding to multiple user jobs if being run in a shared environment). Fig. 2 shows the failure distribution in non-shared environments while Table V shows the results of the hypothesis testing.

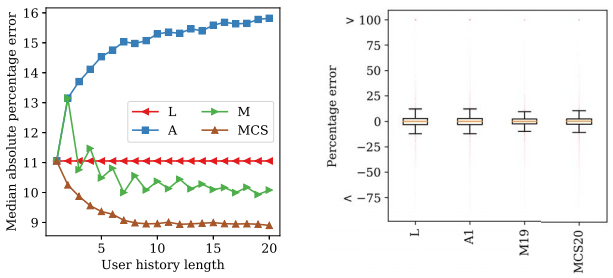
We observe that the failure due to memory error distribution is positively correlated with the tail memory usage of non-shared single and shared single jobs (plot similar to Fig. 2a) for *System A* as well as single jobs for *System B*. Recollect that on *System B*, all jobs run in non-shared mode. While the positive correlation is expected, it is quite surprising to see the likelihood of failure increasing even when the available memory is more than half of the total node memory capacity for *System A*. *System B* exhibits the expected behavior where the failure rate is flat till close to node memory capacity and then jumps to 1. Neither *System A* nor *System B*'s non-shared multi node jobs exhibit any such positive correlation and due to higher *p*-value the null hypothesis cannot be rejected.

There are two root causes for this OOM problem. First, users sometime mistakenly provide upper bound for their memory limit. Second, when some heavy memory usage applications reach close to the upper bound, they go into a "death spiral" whereby they cannot free memory while writing out the memory to disk. This phenomenon has been reported



(a) *System A* non-shared single (b) *System B* non-shared single

Fig. 2: Failure rate vs tail memory usages.



(a) Different models performance with different history lengths on the training set (b) Percentage error distribution for different models on test set with best history length as per training set.

***Fig. 3: Memory usage prediction for* System A**

previously with the OOM killer in Linux [20]. The least square fit in Fig. 2 corresponds to function $f = 0$ for $x <= a$ else $f = b(x - a)$ where f is failure rate and x is memory usage in GB. The best fit curves have R^2 value of 0.72 and 0.97 for Fig. 2a and Fig. 2b.

Implications for System Design: *Job failure rate caused by OOM error increases with increasing tail-memory utilization, but the increase starts to happen much earlier than the node memory capacity in* System A. *Application of data mining would help to determine when a job should be pre-empted and moved to a larger node.*

2) Prediction of memory usage based on user profile:: We have seen above that memory usage is positively correlated with job failure rate. While one approach to avoid such failure is to continuously monitor memory utilization and take proactive action whenever memory utilization is high, a better approach is to predict memory utilization of a job even before it starts executing. Then a scheduler can decide in advance where to schedule a job in case of heterogeneous memory cluster or it can make a more informed decision on which all jobs to schedule together on a node in case sharing is enabled. Figure 3 shows the results of different predictors (explained in the last paragraph of Sec V) for *System A*. Here, jobs can be shared or non-shared. We observe that any predictor performs equally well. Median absolute percentage error (MAPE) of all predictors are less than 12% (for at least one history length). Among these, Maximum Cosine Similarity (MCS) outperforms others for any selected history length.

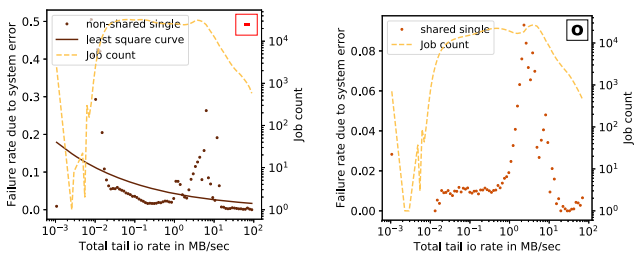
B. Local IO

1) Relation of job failure with local IO usage:: In this section, we conducted an analysis to conclude if total local

IO on a node impacts job failure likelihood. Fig. 4 shows the results for *System A*. *System B* has no local storage (always uses NFS for IO). Fig. 4a corresponding to non-shared single jobs for *System A*, shows an overall negative correlation, which initially appears counter-intuitive. The explanation is that any IO related issues usually restrict IO usage and hence jobs fail with lower IO rate. On the other hand an IO heavy job unimpeded by any system-related issues can perform the required IO operations at much higher rate and completes successfully. This analysis reveals that a good number jobs are failing due to systems issues such as disk bottlenecks or faulty drivers that restrict IO rate. We also observe a peak in the failure rate around 6 MB/s which is much lower than the specified IO limit of available local discs (100 MB/s) for *System A*. The least square fit in Fig. 4a corresponds to function $f = ae^{bx}$ where f is failure rate and x is resource usage in log scale. For fitted curve, $a = 0.04, b = -0.48$ with R^2 value of 0.16. Even though we cannot reject the null hypothesis for shared single jobs of *System A*, a peak in the failure rate, similar to the one for non-shared single jobs, is observed around 3 MB/s in Fig. 4b. We see a dip in the job failure rate to the right end of the Fig. 4b. This higher IO above the local IO operating limit is due to caching which is counted in the IO rate by the performance stats. The high failure rate around 6 MB/s for non-shared single and 3 MB/s for shared single is because random access rates are usually much slower (can be more than 100X slower [1], [2]) than sequential access rates and such accesses are more common in shared.

Implications for System Design: *This analysis can be used to identify system issues in the local IO. A pattern like the peak in the job failure rate can help in estimating the operational IO rate limit for local storage, which can be much smaller than the rated capacity. Thus, while provisioning the IO subsystem, one should consider the prevalence of random IO (rather than sequential IO) in the applications of value and benchmark the IO system to determine this lower rate for random IO.*

2) Prediction of local IO usage based on user profile:: The above analysis shows failure rate is high (peaks in Fig. 4) with higher IO usage and the peak in case of shared occurs earlier than non-shared. The difference in behavior between shared and non-shared is primarily due to randomness in access which is more common in case of shared. Hence, if we can predict IO requirements of a job in advance, this randomness can be minimized by scheduling only shared jobs with complementary local IO requirements. So, we explore user historical information based different local IO usage prediction models. Table VI shows the performance of different models on the test dataset. We observe the median predictor with $MAPE = 23.1\%$, outperforms others.



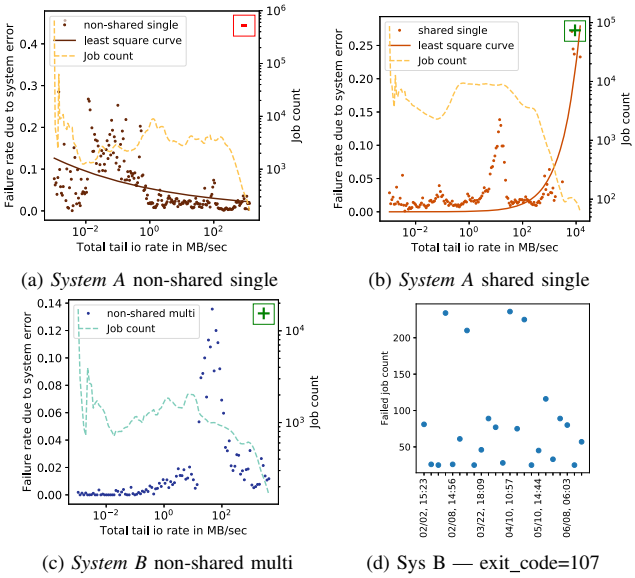
(a) *System A* non-shared single (b) *System A* shared single

Fig. 4: Failure rate vs total tail local IO rates

C. Network File System

1) Relation of job failure with remote IO usage:: This section studies job failure correlation with total remote storage usage rate. Fig. 5 shows the results for *System A* and *System B*—the metric is the I/O rate per node. The least square curves for *System A* correspond to function $f = ae^{bx}$ where f is failure rate and x is resource usage in log scale. Hypothesis testing gives statistically significant negative correlation with respect to total tail remote usage rate for non-shared single (Fig. 5a) and non-shared multi jobs (figure similar as Fig. 5a) for *System A*. These results can be explained by the fact that during congestion for remote storage server or with poor network connectivity between computational and storage nodes, jobs fail with low I/O usage rate. However, for shared single jobs for *System A*, the overall failure likelihood increases as total remote I/O rate increases as shown in Fig. 5b. Recollect that the resource usage in shared mode is the aggregated resource usage across *all* jobs running on the node at the given time. Here failure is triggered by interference among multiple jobs contending for the remote storage service. At a high level, the contention for remote resources with executing elements outside the node is predominant in non-shared environment, while the contention with other jobs executing on that same node is dominant for a shared environment. Therefore there is a negative correlation of failure rate in the non-shared environment while a positive correlation in the shared environment.

In case of *System B*, no correlation is observed between failure rate and total remote I/O usage rate for single-node



(a) *System A* non-shared single (b) *System A* shared single

(c) *System B* non-shared multi (d) Sys B — exit_code=107

Fig. 5: Failure rate vs total tail network file system I/O rates

jobs. Moreover, the failure rate distribution is flat for the whole remote I/O range. This is expected for *System B* since the peak read/write rate by a single node is 9.6 GB/s (limited by the network interface card capacity) whereas the peak bandwidth supported by the file system is much higher at 1.1 TB/sec. However, for multi-node jobs, we observe that the failure rate is positively correlated with tail usage rate with a sharp peak close to 46 MB/sec (Fig. 5c). It is difficult to explain why contention in the NFS happens precisely at this traffic volume going out of one node. However, we hypothesize that at this volume, aggregated over all the nodes, there is contention at the NFS. Though its rated capacity is 1.1 TB/s, its actual operating limit is much lower due to random access (as shown for local I/O). On investigating deeper and analyzing all the failed jobs around the peak, we find that most of the jobs in that peak failed with an exit code 107. Exit code 107 is returned when 'transport endpoint is not connected' indicating jobs are not able to connect to the NFS. Tellingly, 33% of all system-related job failures across the study period are due to unreachability of remote file system.

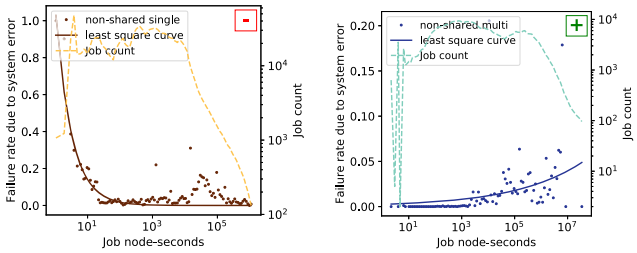
To understand the failures caused by unreachability of remote file system and its system-wide impact, all jobs with exit code 107 are clustered using the DBSCAN algorithm [22] with parameters ϵ=300s and min job failure count=25. ϵ is the maximum radius of the neighborhood from point p, where p represents a failed job. A total of 26 clusters are found using this approach and are shown in Fig. 5d along with the total number of failures that belong to the cluster. The median time between occurrence of these burst failures (i.e., more than 25 jobs failing within a duration of 300s) is 3.3 days. A remote file system may become unreachable due to: (i) remote file system failure (2 clusters), (ii) network failure (4 clusters) and, (iii) congestion in the network and NFS (remaining 20).

Implications for System Design: *Bandwidth to the parallel file system continues to be a problem for large-scale systems. Where it is not possible to increase that bandwidth, it is needed to carefully monitor its net usage and to stagger the IO requests from different applications at times of contention.*

2) Prediction of remote IO usage based on user profile:: The above analysis shows that contention at remote storage increases job failure likelihood. Additionally, contention with other jobs executing on that same node increases failure likelihood for a shared environment. Hence, if the remote IO requirement of a job can be predicted, it can enable a better scheduling strategy such as scheduling jobs with high remote IO at different times. Table VI shows the results of different IO usage predictors. We observe the Maximum Cosine Similarity model with $MAPE = 22.3\%$, outperforms others.

D. Network

1) Relation of job failure with network usage:: We omitted all plots of this section to save space. No significant correlation is found for non-shared multi-node jobs of *System A* and (non-shared) single-node jobs of *System B* (refer Table V). However, non-shared single jobs for *System A* and multi jobs for *System B* are negatively correlated with tail I/O usage rate.



(a) *System A* non-shared single (b) *System B* non-shared multi

Fig. 6: Failure rate vs node-seconds.

As discussed earlier, negative correlation implies system issues such as contention or poor network connectivity is causing jobs to fail with low tail network usage. Similar to remote I/O, we observe overall positive correlation for shared single-node jobs for *System A*.

2) Prediction of network usage based on user profile:: Network usage prediction model can help in minimizing the contention in network. Hence, here we explore different network usage prediction models. Table VI shows the performance of 4 different kinds of models on the test dataset. All models have promising $MAPE$ of $< 18\%$. Among these, MCS based model has the best performance of 13.0%.

E. Job Node-seconds

1) Relation of job failure with job node-seconds:: This section analyzes the effect of the total node-seconds of a job (defined in Sec.II) on the job failure rate. Prior studies have found positive correlation of failure rate with the total execution time of an application [14], [19]. For example, [19] found that for extreme scale Blue Waters applications, both of CPU and CPU+GPU kind, there was a linearly increasing relationship (in log-log scale) of the probability of application failure and the application node hours. Here we analyze to see if a similar relationship holds for *System A* and *System B*.

We find that for *System A*, the relation has a negative slope for all three categories of jobs—non-shared single (Fig. 6a), non-shared multi, and shared single (negative correlation with distribution similar to Fig. 6a for both non-shared multi and shared single—plot omitted to save space). This seemingly counter-intuitive relation can be explained by the observation that many novice users submit jobs to *System A*. The allocation model is that any faculty member who purchases even one asset in the system can authorize researchers from her group to execute on the cluster. Therefore, many poorly written jobs get submitted, which on startup make huge demands on system resources (such as loading huge datasets into memory) and fail quickly. Hence, the high failure rate for low job execution times. On the other hand, jobs that have executed for a while are unlikely to run into such problems.

The trend for *System B* is the opposite. Here, long-running jobs do put pressure on the system resources and hence have a higher likelihood of failing due to system issues. Here the codes are more mature and the long-running jobs are exposed to more faults in space or time or both. Interestingly, a significant fraction of all failed jobs fail with job execution

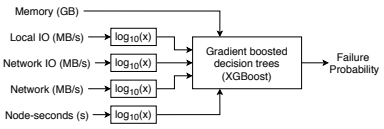


Fig. 7: ML model for predicting impending failure of a job.

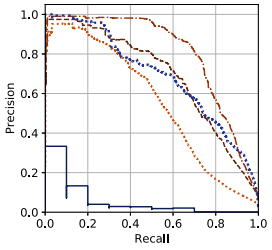


Fig. 8: PR curves

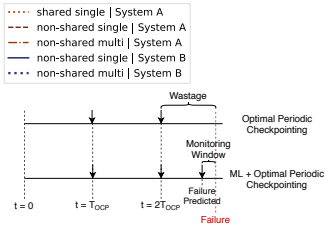


Fig. 9: Periodic vs ML + Periodic

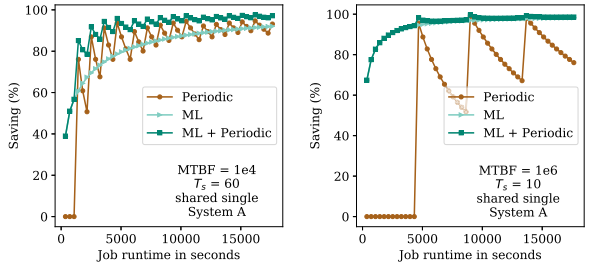


Fig. 10: [Base case: Periodic] ML model + periodic checkpointing savings as a percentage of the total time that would have been wasted in the baseline case due to job failure and no checkpointing. Mean time between failures (MTBF in seconds) and time to save information at a checkpoint (T_s in seconds) used to compute optimal checkpointing frequency are given in the bottom right of each plot. Jobs with runtimes up to 5 hours are considered.

time less than 1 minute—34% for *System A* and 45% for *System B*. These cause inefficient cluster usage due to the constant overhead of scheduling, initiating, and terminating execution.

Implications for System Design: *With mature demanding codes, the job failure rate goes up with larger jobs. However, with naïve usage models, the failure rate is high for short jobs.*

2) Prediction of job runtime based on user profile: While a runtime prediction model cannot help in preventing a job failure, it can improve the quality of scheduling significantly (as we show in Section VI-B). We observe that users overestimate their runtime (more than 9X on median) leading to longer queue time since the scheduler takes requested walltime as input. Hence, we explore different runtime prediction models here. Table VI shows the results of the different models on the test dataset. All predictors have $MAPE$ less than 17% with the median predictor outperforming others.

VI. Application of the Analyses

A. Predicting Job Failures

The previous section shows that the job failure rate is correlated with usage for certain resources. The question we pose ourselves is, can we predict impending failures to take mitigation actions, such as, taking a checkpoint and migrating

TABLE VII: Symbols used for saving calculation

Symbol: Description
$\mathbf{x}$: Total number of failures — $\mathbf{y}$: Number of failures predicted
$\mathbf{P}$: Precision of the model — $\mathbf{R}$: Recall of the model
$\mathbf{T_f}$: Mean time between node failure (MTBF)
$\mathbf{T_s}$: Time to save information at a checkpoint
$\mathbf{T_w}$: $\sqrt{T_s T_f}$ [82] (Work time between checkpoints)
$\mathbf{T_{OCP}}$: $T_s + T_w$ (Optimal checkpointing period)
$\mathbf{T_r}$: Runtime of the job
$\mathbf{N_{OCP}}$: $\lfloor T_r / T_{OCP} \rfloor$ (Number of periodic checkpoints taken)
$\mathbf{S_{OCP}}$: Saving by the optimal periodic checkpointing method
$\mathbf{S_{ML}}$: Saving by the ML method
$\mathbf{S_T}$: Saving by the ML + periodic method

the process. This motivates us to explore different ML models to predict job failure using the different resource usages for a job as input features. We divide our dataset into training and test sets in the ratio 7:3. After trying different ML models such as linear regression, logistic regression, and decision trees, we find that gradient-boosted decision tree performs the best based on recall and precision scores. The output is converted to a binary decision using a threshold, whether the job will fail or not. These gradient-boosted decision tree based models have been implemented using the `XGBoost` package in Python (refer Figure 7). Fig. 8 shows the precision-recall curves for different execution environments and job types generated by varying the threshold. We observe that model performance corresponding to the non-shared single job types of *System B* is not usable, expectedly since the number of jobs belonging to failure category due to system issues is insignificant (0.1%, refer Table IV). Additionally, precision scores of multi node jobs are better than that of single node jobs. This we suspect is because the diversity in the single node jobs is significantly higher than in the multi node jobs.

Our job failure prediction model can reduce resource wastage on these systems by triggering a checkpoint when failure is imminent. The application registers a callback which is invoked by our system upon this event. Since the failure prediction model is not perfect, our ML model by itself may not give the optimal savings. So, we recommend to combine our ML model with the optimal periodic checkpointing method such as one given by Young [82] or Vaidya [76] to obtain the optimal savings. In our implementation, combination means taking a checkpoint whenever either method suggests taking a checkpoint—there are of course other ways of combining the two methods, which we do not explore in this work.

Fig. 9 shows the schemes of optimal periodic checkpointing and ML + optimal periodic checkpointing methods. We define savings as the time that would have been wasted in the baseline case due to job failure and no checkpointing (and that is saved due to the checkpointing) minus the overhead of checkpointing, expressed as a percentage of the total execution time of the job. In the case of periodic checkpointing, whenever a failure occurs, the amount of work done between the last checkpoint to the failure is lost. Our ML-based checkpointing will minimize this wastage by forcing a checkpoint whenever it predicts a failure and the probability is above a threshold. Recall that monitoring happens periodically (5 minutes for

TABLE VIII: [Base case: Periodic] Normalized area under the curve of Figure 10 (normalized with respect to jobs with no wastage execution due to failures). We present results with $T_S = 60s$ a reasonable value for real production jobs. For jobs with higher T_S values, we recommend to use data compression techniques such as the one by Islam [34] (which reduced large-scale application checkpointing overhead to less than 60s). With SSD, this overhead can be further reduced to $T_S = 10s$ [3].

	System		Periodic	ML	ML+Periodic
MTBF=1e4, T_S=60 sec	A	shared single	0.81	0.82	0.91
		non-shared single		0.89	0.94
		non-shared multi		0.90	0.95
	B	non-shared multi		0.91	0.94
MTBF=1e5, T_S=60 sec	A	shared single	0.66	0.82	0.88
		non-shared single		0.89	0.92
		non-shared multi		0.90	0.93
	B	non-shared multi		0.91	0.93
MTBF=1e6, T_S=60 sec	A	shared single	0.30	0.82	0.84
		non-shared single		0.89	0.90
		non-shared multi		0.90	0.91
	B	non-shared multi		0.91	0.92
MTBF=1e6, T_S=10 sec	A	shared single	0.60	0.95	0.95
		non-shared single		0.97	0.97
		non-shared multi		0.97	0.98
	B	non-shared multi		0.97	0.98

System A and 1 minute for *System B*). Therefore the wastage is reduced to the window from the last monitoring (and attendant prediction) to the actual failure. Our next analysis shows how much saving can be achieved using different checkpointing methods. Refer Table VII for all symbols used in the analysis.

1) **Periodic**: $S_{OCP} = \frac{N_{OCP}T_w}{T_r} * 100$
2) **ML**: True positive = $Py = Rx$ or $y = Rx/P$.
   Total time in checkpointing, $T_c = T_s y$.
   $S_{ML} = \frac{RxT_r - T_s y}{xT_r} * 100 = R\big(1 - \frac{T_s}{PT_r}\big) * 100$
3) **ML+periodic**: Number of failures not detected by ML model, $N_{FN} = x - Rx$.
   $S_T = \frac{RxT_r - T_s y - N_{OCP}RxT_s + N_{OCP}N_{FN}T_w}{xT_r} * 100$
   $= \big(R\big(1 - \frac{T_s}{PT_r} - \frac{N_{OCP}T_{OCP}}{T_r}\big) + \frac{N_{OCP}T_w}{T_r}\big) * 100$

Using the derivations of S_{OCP}, S_{ML} and S_T, we now calculate the maximum net savings over all possible (precision, recall) pairs for different MTBF and time to take a checkpoint i.e., T_s values. The results for shared single jobs of *System A* are shown in Fig. 10 (plots are similar for other job types and *System B* and are omitted to save space). However, for comparison, we do present the results in terms of the normalized area under these curves in Table VIII. Based on these results, ML + periodic checkpointing method outperforms the base optimal checkpointing method by between 12.3% (unreliable system with MTBF =1e4, Ts=60s) and 2X (reliable system with MTBF = 1e6 and Ts=60s). Additionally, we observe the savings achieved by the optimal checkpointing method in case of failure decreases as a system becomes more reliable (MTBF increases from $1e4$ to $1e6$). This is because the optimal checkpointing period increases and hence the amount of lost work increases. This limitation is overcome by integrating it with the ML model.

B. Predicting Job Runtime

In this section, we investigate the use case of our resource usage prediction models (Sec. V) and show how these can significantly improve the efficiency of shared computing clusters. Many scheduling techniques rely on job runtime estimates to minimize job waiting times. For example, many current batch job schedulers use *Backfilling* [26], [75] whereby the scheduler prefers smaller jobs to jump to the head of the queue provided they do not delay previously queued jobs. Therefore, accurate job runtime estimates are essential for *Backfilling* to be effective in reducing overall jobs wait time. We apply job runtime prediction to both systems A & B and use the *pyss* [41] simulator, which implements *Backfilling* scheduling (among others). Figure 11 shows the average job wait time for *System A* and *System B* for a trace of 15K jobs. We draw several insights:

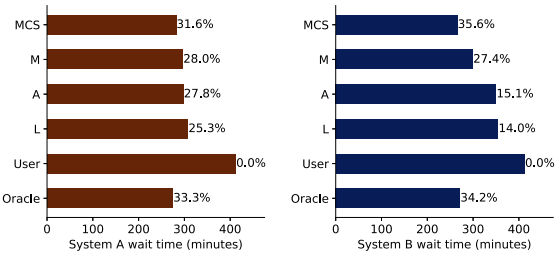


Fig. 11: Overall Job Wait time for* System A *and* System B *using different runtime predictors. The values over the bars show improvements over user estimated runtime

1) Users tend to overestimate the runtimes of their jobs. This causes the scheduler to consider fewer jobs for *Backfilling* and hence increases the overall wait time.
2) The MCS model performs the best among our 4 prediction models, achieving an overall wait time that is within 5% of oracle for *System A* and 1% for *System B*.
3) Between our 4 models, model L performs the worst. However, despite the simplicity of the model, it still performs 25% better than user estimated runtimes for *System A* and 14% better for *System B*.

VII. Threats to Validity

Inferring failure from exit statuses Our failure analysis relies on exit status either from a job (*System A*) or from individual applications within a job (*System B*). While there is a guideline provided by TORQUE for exit status [59], there is no compulsion for a job script or application developer to follow the guideline. However, it is generally found by others [13], [60] and us here that application developers of popular applications (which contribute most data points in our dataset) follow the guideline. The job script developer usually takes the exit code of the last executable in the script and returns that and thus benefits from the standardization of exit status.

I/O Caching and Utilization. We determine the I/O utilization using the `block` and `llite` counters. *Not* all 'read_bytes' and 'write_bytes' values recorded through these counters result in fetching of data from local or remote file system as the data may be cached in memory. However, it is shown to be a good estimator of I/O load for the file system [6].

VIII. Related Work

Failure analysis of large-scale computing systems [18], [19], [23], [31], [54] has been done in the past on many scopes of analysis, such as job-level, application-level, node-level, or system-level. These works focus on hardware reliability rather than job failures. In contrast, Mitra *et al.* [54] performed investigations of jobs submitted to a university cluster to understand their primary failure modes. However, our work focuses on finding the effect of resource usage on reliability of jobs in community clusters. In a closely related work [19], authors conducted job failure characterization study on more than 5 million HPC job runs. However, the analysis was limited to one system and did not consider resource utilization characteristics for job failure study. Our approach concretely shows that resource utilization characteristics by a job play an important role in determining job success rate as it captures both the impact of error/failure in the system as well as interference among jobs. In a recent work [8], the authors studied workload diversity in different clusters. They also categorize jobs into different categories such as successful, timeouts, and unsuccessful. However, unlike this work and several others [12], [21], we delve deeper to determine whether the job failure is due to user or system-related issues. Orthogonal to our analysis, there have been many studies that have focused on performance optimization [35]–[37], [62] or to detect inefficient applications [23], [54], [70], [77] on large-scale clusters. Several node failure [16], [27], [29], [30], [79], and job failure [12], [24], [33], [46] prediction models have been proposed in the past as well. A few prior works such as [58] have performed single-bit error prediction for GPU as well. Our ML failure prediction model belongs to the job failure category. We do not claim novelty in our prediction model, instead show the utility of this simple ML model by integrating it with an existing checkpointing method. A number of works have already explored various optimal checkpointing strategies. For instance, [74] explores checkpointing strategies based on temporal locality of failures. Our proposed checkpointing strategy however is based on job failure prediction.

Related works for job runtime and resource usage prediction can be categorized into 2 categories. i) **Black box prediction:** The kind of prediction where the characteristic of job is unknown. The features used for prediction can be either the fields from the job submission script [44], [68], [69], [72], [75], [80] or the current resource utilization of the job [25], [40], [66]. A recent study [80] uses the parameters used by users during job submission as features to NN to predict runtime and IO rate. While proposed model just uses theses parameter to estimate runtime, the model needs retraining every 100 submitted jobs making it harder to scale for large HPC systems (having thousands of nodes) where more than 100s of jobs are submitted every second. Our proposed predictor also belong to the former class of this category. ii) **White box prediction:** These prediction models utilizes the characteristics of jobs to predict the runtime or resource usage. In some cases the characteristics of a job are known learnt by the algorithm [52], [56], [63] or derived from a particular implementation [45], [57], [65]. In other cases the characteristics are derived from job executing data on known inputs [32], [42], [73], or by performing offline profiling of different jobs and predicting the performance for new unseen jobs [48], [49].

IX. Conclusion And Open Challenges

We have analyzed extensive system usage and failure data from two centrally administered computing clusters at two Tier 1 US Research universities. Our dataset comprises a total of 3.0M and 2.2M jobs from the two clusters, which represents the richest data source to be analyzed in the literature. The two clusters vary significantly in their scale, hardware resiliency characteristics, and systems management practices shedding a comparative light on failure characteristics. Through analysis of different kinds of data—node failures and recovery, job failures, and job resource usages —we bring out important insights into how the clusters behave and implications for how they can be managed more effectively. Our hope is that this study will trigger other studies by researchers using data from their local organization's compute clusters and we look forward to seeing which insights will be shared and which will be distinct. We have also publicly released the dataset on which the analyses are based. Some of our key findings are: (i) A significant fraction of jobs hit against the walltime (33% and 43% for the two systems) and therefore event-driven application-level checkpointing is needed to avoid lost work. (ii) Resource pressure affects different types of jobs differently. Sometime job failure rate is positively correlated with resource usage (such as local memory), while sometime it is negatively correlated with remote resource pressure (such as network usage). These have different implications for how resource contention should be handled. (iii) User historical resource usages can be used to predict resource usages of jobs current in queue. These models can be directly integrated with the existing schedulers to increase its effectiveness. (iv) Job failures due to resource contention can be observed with levels of utilization that is much less the available capacity. This is observed with memory, local IO, and remote IO as well.

Open challenges: We lay out two open challenges to the technical community motivated by the observations from our two production systems. *First*, current optimal checkpointing estimation methods take only hardware reliability (such as MTBF) into account. This paper integrated it with job failure likelihood information. However, there is still scope for improvement. A better method is to consider in addition the rate of job progress [55]. *Second*, current contention-aware schedulers [17] need to profile a job first to estimate job's interference and latency-sensitivity. This is a major limitation for clusters where majority of jobs are short running. This paper presents user history-based resource usage predictions, which can be used to predict a job's profile that then should feed into the contention-aware scheduler.

"Giving up is the only sure way to fail."

Gena Showalter

REFERENCES

[1] Lies, Damn Lies And SSD Benchmark Test Result. https://www.seagate.com/tech-insights/lies-damn-lies-and-ssd-benchmark-master-ti/.

[2] Random vs Sequential. https://blog.open-e.com/random-vs-sequential-explained/.

[3] UserBenchmark. https://www.userbenchmark.com/.

[4] Nagios. https://www.nagios.com, 2018.

[5] Sensu. http://www.sonian.com/cloud-monitoring-sensu/, 2018.

[6] Anthony Agelastos, Benjamin Allan, Jim Brandt, Paul Cassella, Jeremy Enos, Joshi Fullop, Ann Gentile, Steve Monk, Nichamon Naksinehaboon, Jeff Ogden, et al. The lightweight distributed metric service: a scalable infrastructure for continuous monitoring of large scale computing systems and applications. In *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*, pages 154–165. IEEE, 2014.

[7] Robert Alverson, Duncan Roweth, and Larry Kaplan. The gemini system interconnect. In *2010 18th IEEE Symposium on High Performance Interconnects*, pages 83–87. IEEE, 2010.

[8] George Amvrosiadis, Jun Woo Park, Gregory R. Ganger, Garth A. Gibson, Elisabeth Baseman, and Nathan DeBardeleben. On the diversity of cluster workloads and its impact on research results. In *USENIX Annual Technical Conference*, 2018.

[9] Mona Attariyan and Jason Flinn. Automating configuration troubleshooting with dynamic information flow analysis. In *OSDI*, volume 10, pages 1–14, 2010.

[10] Saurabh Bagchi, Rakesh Kumar, Rajesh Kalyanam, Stephen Harrell, Carolyn A Ellis, and Carol Song. Fresco: Open source data repository for computational usage and failures (http://www.purdue.edu/fresco), Oct 2019.

[11] Franck Cappello, Al Geist, William Gropp, Sanjay Kale, Bill Kramer, and Marc Snir. Toward exascale resilience: 2014 update. *Supercomputing frontiers and innovations*, 1(1):5–28, 2014.

[12] Xin Chen, Charng-Da Lu, and Karthik Pattabiraman. Failure analysis of jobs in compute clouds: A google cluster case study. In *Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on*, pages 167–177. IEEE, 2014.

[13] Mendel Cooper. Advanced Bash-Scripting Guide. http://tldp.org/LDP/abs/html/exitcodes.html, 2014.

[14] Domenico Cotroneo, Roberto Natella, Roberto Pietrantuono, and Stefano Russo. A survey of software aging and rejuvenation studies. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 10(1):8, 2014.

[15] J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Gener. Comput. Syst.*, 22(3):303–312, February 2006.

[16] A. Das, F. Mueller, P. Hargrove, E. Roman, and S. Baden. Doomsday: Predicting which node will fail when on supercomputers. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 108–121, Nov 2018.

[17] Christina Delimitrou and Christos Kozyrakis. Paragon: Qos-aware scheduling for heterogeneous datacenters. In *ACM SIGPLAN Notices*, volume 48, pages 77–88. ACM, 2013.

[18] Catello Di Martino, Zbigniew Kalbarczyk, Ravishankar K Iyer, Fabio Baccanico, Joseph Fullop, and William Kramer. Lessons learned from the analysis of system failures at petascale: The case of Blue Waters. In *Dependable Systems and Networks (DSN), 2014 44th Annual IEEE/IFIP International Conference on*, pages 610–621. IEEE, 2014.

[19] Catello Di Martino, William Kramer, Zbigniew Kalbarczyk, and Ravishankar Iyer. Measuring and understanding extreme-scale application resilience: A field study of 5,000,000 hpc application runs. In *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*, pages 25–36. IEEE, 2015.

[20] THE GEEK DIARY. What is Out-of-Memory (OOM) Killer in Linux (Causes, Troubleshooting, Mitigation). https://www.thegeekdiary.com/what-is-out-of-memory-oom-killer-in-linux-causes-troubleshooting-mitigation/.

[21] N. El-Sayed, H. Zhu, and B. Schroeder. Learning from failure across multiple clusters: A trace-driven approach to understanding, predicting, and mitigating job terminations. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 1333–1344, June 2017.

[22] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[23] Todd Evans, William L Barth, James C Browne, Robert L DeLeon, Thomas R Furlani, Steven M Gallo, Matthew D Jones, and Abani K Patra. Comprehensive resource use monitoring for hpc systems with tacc stats. In *Proceedings of the First International Workshop on HPC User Support Tools*, pages 13–21. IEEE Press, 2014.

[24] H. Fadishei, H. Saadatfar, and H. Deldari. Job failure prediction in grid environment based on workload characteristics. In *2009 14th International CSI Computer Conference*, pages 329–334, Oct 2009.

[25] Fahimeh Farahnakian, Pasi Liljeberg, and Juha Plosila. Lircup: Linear regression based cpu usage prediction algorithm for live migration of virtual machines in data centers. In *2013 39th Euromicro Conference on Software Engineering and Advanced Applications*, pages 357–364. IEEE, 2013.

[26] Dror G Feitelson and Ahuva Mu'alem Weil. Utilization and predictability in scheduling the ibm sp2 with backfilling. In *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing*, pages 542–546. IEEE, 1998.

[27] S. Fu and C. Xu. Exploring event correlation for failure prediction in coalitions of clusters. In *SC '07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, pages 1–12, Nov 2007.

[28] Joshi Fullop et al. A diagnostic utility for analyzing periods of degraded job performance. In *Proc. Cray User Group*, 2014.

[29] A. Gainaru, F. Cappello, M. Snir, and W. Kramer. Fault prediction under the microscope: A closer look into hpc systems. In *SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–11, Nov 2012.

[30] Ana Gainaru, Franck Cappello, Marc Snir, and William Kramer. Failure prediction for hpc systems and applications: Current situation and open issues. *Int. J. High Perform. Comput. Appl.*, 27(3):273–282, August 2013.

[31] Saurabh Gupta, Tirthak Patel, Christian Engelmann, and Devesh Tiwari. Failures in large scale systems: Long-term measurement, analysis, and implications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '17, pages 44:1–44:12, New York, NY, USA, 2017. ACM.

[32] Ling Huang, Jinzhu Jia, Bin Yu, Byung-Gon Chun, Petros Maniatis, and Mayur Naik. Predicting execution time of computer programs using sparse polynomial regression. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 1*, NIPS'10, pages 883–891, USA, 2010. Curran Associates Inc.

[33] T. Islam and D. Manivannan. Predicting application failure in cloud: A machine learning approach. In *2017 IEEE International Conference on Cognitive Computing (ICCC)*, pages 24–31, June 2017.

[34] Tanzima Zerin Islam, Kathryn Mohror, Saurabh Bagchi, Adam Moody, Bronis R De Supinski, and Rudolf Eigenmann. Mcrengine: a scalable checkpointing system using data-aware aggregation and compression. *Scientific Programming*, 21(3-4):149–163, 2013.

[35] S. Jha, V. Formicola, C. D. Martino, M. Dalton, W. T. Kramer, Z. Kalbarczyk, and R. K. Iyer. Resiliency of hpc interconnects: A case study of interconnect failures and recovery in blue waters. *IEEE Transactions on Dependable and Secure Computing*, 15(6):915–930, 2018.

[36] S. Jha, A. Patke, J. Brandt, A. Gentile, M. Showerman, E. Roman, , Z. Kalbarczyk, W. T. Kramer, and R. Iyer. A study of network congestion in two supercomputing high-speed interconnects. In *2019 IEEE 26th Annual Symposium on High-Performance Interconnects (HOTI)*, Aug 2019.

[37] Saurabh Jha, Archit Patke, Jim Brandt, Ann Gentile, Benjamin Lim, Mike Showerman, Greg Bauer, Larry Kaplan, Zbigniew Kalbarczyk, William Kramer, and Ravi Iyer. Measuring congestion in high-performance datacenter interconnects. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 37–57, Santa Clara, CA, February 2020. USENIX Association.

[38] Saurabh Jha, Archit Patke, Mike Showerman, Jeremy Enos, Greg Bauer, Zbigniew Kalbarczyk, Ravishankar Iyer, and William Kramer. Monet - blue waters network dataset (https://doi.org/10.13012/B2IDB-2921318_V1), 2019.

[39] M. Karo, R. Lagerstrom, M. Kohnke, and C. Albing. The application level placement scheduler. In *Cray User Group - CUG*, 2008.

[40] Yiannos Kryftis, Constandinos X Mavromoustakis, George Mastorakis, Evangelos Pallis, Jordi Mongay Batalla, Joel JPC Rodrigues, Ciprian

Dobre, and Georgios Kormentzas. Resource usage prediction algorithms for optimal selection of multimedia content delivery methods. In *2015 IEEE international conference on communications (ICC)*, pages 5903–5909. IEEE, 2015.

[41] The Hebrew University Experimental Systems Lab.

[42] H. Li, Y. Wu, Y. Chen, C. Wang, and Y. Huang. Application execution time prediction for effective cpu provisioning in virtualization environment. *IEEE Transactions on Parallel and Distributed Systems*, 28(11):3074–3088, Nov 2017.

[43] Jack Li, Calton Pu, Yuan Chen, Vanish Talwar, and Dejan Milojicic. Improving preemptive scheduling with application-transparent checkpointing in shared clusters. pages 222–234, 11 2015.

[44] Xiuqiao Li, Nan Qi, Yuanyuan He, and Bill McMillan. Practical resource usage prediction method for large memory jobs in hpc clusters. In *Asian Conference on Supercomputing Frontiers*, pages 1–18. Springer, 2019.

[45] Y. Ling, F. Liu, Y. Qiu, and J. Zhao. Prediction of total execution time for mapreduce applications. In *2016 Sixth International Conference on Information Science and Technology (ICIST)*, pages 341–345, May 2016.

[46] C. Liu, J. Han, Y. Shang, C. Liu, B. Cheng, and J. Chen. Predicting of job failure in compute cloud based on online extreme learning machine: A comparative study. *IEEE Access*, 5:9359–9368, 2017.

[47] Yudan Liu, R. Nassar, C. Leangsuksun, N. Naksinehaboon, M. Paun, and S. L. Scott. An optimal checkpoint/restart model for a large scale high performance computing system. In *2008 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–9, April 2008.

[48] Ashraf Mahgoub, Paul Wood, Sachandhan Ganesh, Subrata Mitra, Wolfgang Gerlach, Travis Harrison, Folker Meyer, Ananth Grama, Saurabh Bagchi, and Somali Chaterji. Rafiki: a middleware for parameter tuning of nosql datastores for dynamic metagenomics workloads. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, pages 28–40, 2017.

[49] Ashraf Mahgoub, Paul Wood, Alexander Medoff, Subrata Mitra, Folker Meyer, Somali Chaterji, and Saurabh Bagchi. {SOPHIA}: Online reconfiguration of clustered nosql databases for time-varying workloads. In *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, pages 223–240, 2019.

[50] Amiya K Maji, Subrata Mitra, Bowen Zhou, Saurabh Bagchi, and Akshat Verma. Mitigating interference in cloud services by middleware reconfiguration. In *Proceedings of the 15th International Middleware Conference*, pages 277–288. ACM, 2014.

[51] Catello Di Martino, Saurabh Jha, William Kramer, Zbigniew Kalbarczyk, and Ravishankar K Iyer. Logdiver: a tool for measuring resilience of extreme-scale systems and applications. In *Proceedings of the 5th Workshop on Fault Tolerance for HPC at eXtreme Scale*, pages 11–18. ACM, 2015.

[52] Andréa Matsunaga and José AB Fortes. On the use of machine learning to predict the time and resources consumed by applications. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 495–504. IEEE Computer Society, 2010.

[53] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014.

[54] Subrata Mitra, Suhas Javagal, Amiya K Maji, Todd Gamblin, Adam Moody, Stephen Harrell, and Saurabh Bagchi. A study of failures in community clusters: The case of conte. In *Software Reliability Engineering Workshops (ISSREW), 2016 IEEE International Symposium on*, pages 189–196. IEEE, 2016.

[55] Subrata Mitra, Ignacio Laguna, Dong H Ahn, Saurabh Bagchi, Martin Schulz, and Todd Gamblin. Accurate application progress analysis for large-scale parallel debugging. In *Proceedings of the ACM Symposium on Programming Language Design and Implementation (PLDI)*, volume 49, pages 193–203. ACM, 2014.

[56] Tudor Miu and Paolo Missier. Predicting the execution time of workflow activities based on their input features. In *Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, SCC '12, pages 64–72, Washington, DC, USA, 2012. IEEE Computer Society.

[57] Sara Mustafa, Iman Elghandour, and Mohamed A. Ismail. A machine learning approach for predicting execution time of spark jobs. *Alexandria Engineering Journal*, 57(4):3767 – 3778, 2018.

[58] B. Nie, J. Xue, S. Gupta, T. Patel, C. Engelmann, E. Smirni, and D. Tiwari. Machine learning models for gpu error prediction in a large scale hpc system. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 95–106, June 2018.

[59] University of Michigan. Interpreting Torque Return Codes. https://arc-ts.umich.edu/software/torque/return-codes/, 2018.

[60] Stack Overflow. Are there any standard exit status codes in Linux? https://stackoverflow.com/questions/1101957/are-there-any-standard-exit-status-codes-in-linux, 2012.

[61] KyoungSoo Park and Vivek S Pai. Comon: a mostly-scalable monitoring system for planetlab. *ACM SIGOPS Operating Systems Review*, 40(1):65–74, 2006.

[62] Fabrizio Petrini, Darren J Kerbyson, and Scott Pakin. The case of the missing supercomputer performance: Achieving optimal performance on the 8,192 processors of asci q. In *SC'03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, pages 55–55. IEEE, 2003.

[63] I. Pietri, G. Juve, E. Deelman, and R. Sakellariou. A performance model to estimate execution time of scientific workflows on the cloud. In *2014 9th Workshop on Workflows in Support of Large-Scale Science*, pages 11–19, Nov 2014.

[64] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, page 7. ACM, 2012.

[65] Nikzad Babaii Rizvandi, Javid Taheri, Reza Moraveji, and Albert Y Zomaya. On modelling and prediction of total cpu usage for applications in mapreduce environments. In *International conference on Algorithms and architectures for parallel processing*, pages 414–427. Springer, 2012.

[66] Florian Schmidt, Mathias Niepert, and Felipe Huici. Representation learning for resource usage prediction. *arXiv preprint arXiv:1802.00673*, 2018.

[67] Bianca Schroeder and Garth Gibson. A large-scale study of failures in high-performance computing systems. *IEEE Transactions on Dependable and Secure Computing*, 7(4):337–350, 2010.

[68] Warren Smith, Ian Foster, and Valerie Taylor. Predicting application run times with historical information. *Journal of Parallel and Distributed Computing*, 64(9):1007 – 1016, 2004.

[69] Ozan Sonmez, Nezih Yigitbasi, Alexandru Iosup, and Dick Epema. Trace-based evaluation of job runtime and queue wait time predictions in grids. In *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing*, HPDC '09, pages 111–120, New York, NY, USA, 2009. ACM.

[70] Niyazi Sorkunlu, Varun Chandola, and Abani K. Patra. Tracking system behaviour from resource usage data. *CoRR*, abs/1705.10756, 2017.

[71] Garrick Staples. Torque resource manager. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

[72] Taraneh Taghavi, Maria Lupetini, and Yaron Kretchmer. Compute job memory recommender system using machine learning. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 609–616. ACM, 2016.

[73] S. Tahvili, M. Saadatmand, M. Bohlin, W. Afzal, and S. H. Ameerjan. Towards execution time prediction for manual test cases from test specification. In *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 421–425, Aug 2017.

[74] D. Tiwari, S. Gupta, and S. S. Vazhkudai. Lazy checkpointing: Exploiting temporal locality in failures to mitigate checkpointing overheads on extreme-scale systems. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 25–36, June 2014.

[75] Dan Tsafrir, Yoav Etsion, and Dror G. Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Trans. Parallel Distrib. Syst.*, 18(6):789–803, June 2007.

[76] Nitin H. Vaidya. Impact of checkpoint latency on overhead ratio of a checkpointing scheme. *IEEE Transactions on Computers*, 46(8):942–947, 1997.

[77] Davide Del Vento, Thomas Engel, Siddhartha S. Ghosh, David L. Hart, Rory Kelly, Si Liu, and Richard Valent. System-level monitoring of floating-point performance to improve effective system utilization. In *State of the Practice Reports*, SC '11, pages 5:1–5:6, New York, NY, USA, 2011. ACM.

[78] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at google with borg. In *Proceedings of the Tenth European Conference on Computer Systems*, page 18. ACM, 2015.

[79] M. Wu, X. Sun, and H. Jin. Performance under failures of high-end computing. In *SC '07: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, pages 1–11, Nov 2007.

[80] Michael R. Wyatt, II, Stephen Herbein, Todd Gamblin, Adam Moody, Dong H. Ahn, and Michela Taufer. Prionn: Predicting runtime and io using neural networks. In *Proceedings of the 47th International Conference on Parallel Processing*, ICPP 2018, pages 46:1–46:12, New York, NY, USA, 2018. ACM.

[81] Tianyin Xu, Xinxin Jin, Peng Huang, Yuanyuan Zhou, Shan Lu, Long Jin, and Shankar Pasupathy. Early detection of configuration errors to reduce failure damage. In *OSDI*, pages 619–634, 2016.

[82] John W Young. A first order approximation to the optimum checkpoint interval. *Communications of the ACM*, 17(9):530–531, 1974.