THE MYSTERY OF THE FAILING JOBS: INSIGHTS FROM OPERATIONAL DATA FROM TWO UNIVERSITY-WIDE COMPUTING SYSTEMS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Rakesh Kumar

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

August 2019

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Prof. Saurabh Bagchi, Chair
School of Electrical and Computer Engineering

Prof. Jan P. Allebach
School of Electrical and Computer Engineering

Prof. Somali Chaterji
Agricultural and Biological Engineering

Prof. Felix X. Lin
School of Electrical and Computer Engineering

Approved by:

Dr. Dimitrios Peroulis, Head of the Graduate Program
School of Electrical and Computer Engineering

ACKNOWLEDGMENTS

This thesis would not have been possible without the contribution of many people. I thank all of them who made this possible and because of whom I have had a memorable graduate research experience.

I want to express my gratitude to my advisor Prof. Saurabh Bagchi, for giving me this opportunity to work on the research project and the freedom to experiment with new ideas and explore new possibilities. I thank him for the inspiration, innovative ideas, constant support, and encouragement throughout the project.

My heartfelt thanks to Ashraf Mahgoub and Natat Sombuntham, my colleagues at Purdue, and Saurabh Jha from the University of Illinois at Urbana-Champaign whose contributions to this project have been indispensable. I thank them for helping me with various analyses in the project. I sincerely thank Prof. Ravishankar K. Iyer, Prof. Zbigniew T. Kalbarczyk and Prof. William T. Kramer from the University of Illinois at Urbana-Champaign for their invaluable inputs and timely help and advice. My sincere thanks to Rajesh Kalyanam, Stephen L. Harrell and Dr. Carol X. Song, from Information Technology at Purdue (ITaP), for helping us get access to the various data sets in spite of their busy schedules.

I am also grateful to all my advisory committee members, Prof. Jan P. Allebach, Prof. Somali Chaterji, and Prof. Felix X. Lin, for their various forms of help and support during my graduate study. I appreciate the financial support from NSF that funded the project discussed in this dissertation.

Finally, I would like to thank my parents, my family, and my friends. They have been my constant support and strength in this process. Without their help, it would not have been possible to reach this goal.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Kumar, Rakesh MS, Purdue University, August 2019. The Mystery of the Failing Jobs: Insights from Operational Data from Two University-Wide Computing Systems. Major Professor: Saurabh Bagchi.

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. 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.

1. INTRODUCTION

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, and Frontera at the University of Texas at Austin. In United States, these systems are funded through National Science Foundation. In this model, academicians across the country write research project proposals and submit to NSF for getting hours on the system. NSF, using a panel of research scholars, competetively selects and allocates system hours. University hosting the system gets a marginal share of system hours which is also given out to researchers at the home university competetively.

Another trend in many universities' IT acquisition is the adoption of the *community cluster model*. Often, no single research group can maintain its own cluster—the hardware and administrative costs would be too high. In this model, 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 uti-

lization 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. Centralizing this expertise cuts costs and allows disparate groups to buy large machines with competitive prices for volume purchases. Most importantly, researchers can focus on their research while leaving the task of managing the compute infrastructure to the IT wing of the university. 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 2017 they provided 302M CPU hours. This model is also being successfully used at the Universities of Rochester, Delaware, and Texas at Austin.

In this paper, we study the end reliability of the 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 two US Research I universities[1]. The *System A* cluster formed the main research computing platform for the university U1 at the time of its deployment. The details of the source data are provided in Table 1.1. 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). In this set, a majority (83%) are single node jobs, which means they use less than 16 cores. Of the single node jobs, approximately 39% are shared, meaning multiple jobs from different users share that node. 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 [1]. 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. This dataset represents the largest one that has been analyzed publicly to date for failure characteristics, whether of a university compute infrastructure or otherwise.

[1]We will henceforth refer to the universities as *Research U1* and *Research U2* respectively. All names are anonymized to respect the double blind policy.

Table 1.1.: Summary of data analyzed for the two university-wide clusters at Research U1 and Research U2. The percentages in parentheses refer to the raw counts and node hours.

Computing Cluster		***System A***	***System B***
Duration		Mar 2015-Jun 2017	Feb-Jun 2017
# jobs		2,968k	2,231k
shared	# debug	0.137k (0.0%, 0.0%)	-
	# single	1,125k (37.9%, 15.8%)	-
	# multi	28k (1.0%, 1.9%)	-
	total	1,153k (38.9%, 17.7%)	-
non-shared	# debug	61k (2.0%, 0.1%)	12k (0.5%,0.0%)
	# single	1,348k (45.4%, 18.3%)	1,640k (73.5%, 5.3%)
	# multi	407k (13.7%, 63.9%)	580k (26%, 94.3 %)
	total	1,815k (61.1%, 82.3%)	2,231k (100%)
# unique users		617	467

1.1 Categories of Analysis

This paper presents the results of 4 different categories of analysis. In the first, we show the breakdown of node failures into different categories and reason about the corresponding up times and recovery times. In the second, we consider the failures of individual jobs and reason about their root cause through examination of their exit codes. The third analysis sheds light on the relation between resource usage and job failure rates. We consider the 5 primary kinds of resources—memory on a node, local IO, remote IO to the parallel file system, network, and runtime of a job (in terms of node-seconds of execution). Finally, we present a job failure prediction model which can help minimize resource wastage corresponding to job failures due to system related issues. For all the analyses, we consider only production jobs, i.e., eliminating jobs that are submitted for debugging purposes.

1.2 Key Results and Observations

We present the following key results, the observations, and the implications for administering central compute clusters with general-purpose needs. Wherever possible, we separate the recommendation for cluster provider **(P)** and user **(U)**.

- **O1**: Multi-node jobs consume the majority of computational time even though their overall percentage by count is significantly less compared to single node jobs. Their percentage by total node hours is 66% for *System A* and 95% for *System B* while by total number, it is 15% for *System A* and 26% for *System B*.

  R1: **(P)** While making important decisions like required network capacity per node, procurement practice should consider the multi-node case as it represents the majority of node-seconds time in the system. For example, per node network use in case of multi-node applications is 3X on median than non-shared single-node case (1.8 vs 0.6 in MB/s) for System A while for System B, it is 8X (1.5 vs 0.18 in MB/s).

  Refer: Table 1.1

- **O2**: Differences in hardware and software resiliency techniques can affect the continuous node reachability of systems. The 95-th percentile data point for *System B* is 92 days (2.2X) compared to 42 days for *System A*.

  R2: **(P)** *System B* is more reliable by design partly because it uses expensive hardware solutions such as chipkill memory and has more automated mechanisms for failure logging, monitoring, and root cause analysis (such as CrayPAT [2], LogDiver [3]). Unlike *System B*, *System A* does not keep a record of previous node failure root causes and mechanism of recovery. Such records can help faster root cause identification and reduce downtime.

  Refer: Sec 4.1

- **O3**: Jobs may fail due to system-related or user related errors. Their relative frequencies differ in the two systems. System issues are responsible for over 53% of the failed jobs in System A, while they only cause 4% of the failed jobs in system B. Also, we notice that sharing a node among multiple jobs does not increase the fraction of jobs that fail due to system issues (56% with shared-single vs 57% with non-shared single). Both System A and System B have significant user-related failures (33% and 48% of all failures).

  R3: **(P)** Reliability in System A will increase with online monitoring of resource usages of the jobs and detecting when resource exhaustion is being approached such as in [4, 5]. Failure prediction model similar to the one described in Sec. 6 can help minimize resource wastage by triggering a checkpoint when failure is imminent. Moreover, exit code-based categorization can be used to identify job failure category. This can help both user and system owner identify if a failure is due to user fault or system issues (like resource exhaustion). **(U)** User-related job failures can be minimized by using static analysis tools for checking job scripts, environment setup, and user code as in [6–8] together with user training. We also believe it will help if users compile and execute their jobs in containerized environments, which simulate the cluster but at a smaller scale.

  Refer: Table 4.3

- **O4**: 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%). Walltime is either the upper bound for the execution time specified by the job owner or a system-wide upper bound: 336 hours (*System A*) and 48 hours (*System B*). When a job has executed for the walltime, it is typically killed.

  R4: Significant loss of work can happen when the program is terminated upon hitting walltime. **(P)** Like System B, future systems should provide extra cycles to enable an application to take a checkpoint when job termination is signaled

due to walltime. **(U)** To avoid the loss of resources, jobs should perform application level checkpointing with reasonable frequency [9, 10]. Various optimal checkpointing estimation techniques [9, 11] do not take into account a dynamically changing failure probability. A failure prediction model like ours in Sec. 6 can help minimize checkpointing overhead by dynamically reducing checkpointing frequency when failure likelihood is low. Users can also use any available tooling to better estimate walltimes [12].

Refer: Table 4.3

- **O5**: Probability of job failure due to Out-of-Memory (OOM) error increases with increasing memory utilization even when greater than 10 GB memory is free on a node for *System A*. *System B* has a flat failure rate curve with a jump close to the node memory limit.

  R5: There are two root causes for this problem. First, users sometime mistakenly provide upper bound for the memory their job will require. 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 previously with the OOM killer in Linux [13]. **(P)** Provide tooling support to users to estimate upper bound of memory usage [14]. Use user history based memory usage prediction model [15] while making scheduling decisions for the shared jobs. Monitor and start taking preemptive measures when the application gets close to the memory capacity [16].

  Refer: Sec 5.1

- **O6**: With respect to remote IO and network resources, *System A* shared single jobs have positive correlation of failure rate with resource usage while non-shared jobs have either negative correlation or no correlation. We also see similar behavior in *System B* with network usage (all jobs in *System B* are non-shared).

R6: Local contention is dominant for shared jobs while global contention is dominant for non-shared. **(P)** Reliability for shared jobs can be improved by deploying contention-aware schedulers [17, 18]. Additionally, since monitoring all resources for all applications could add significant overhead, our results indicate to monitor local resources for shared jobs and global resources for non-shared jobs. For shared environment, we also recommend the admins to adopt resource isolation technologies (such as containers [19]), which can reduce failures due to local contention. **(U)** User can reduce her job failure likelihood by using tools for dynamic reconfiguration of applications based on current resource availability [5], such as reconfiguring the number of threads or network timeout.

Refer: Sec 5.2, 5.3, 5.4

- **O7**: In several instances (35%) updating one of the 20 most widely used libraries on *System A* causes job reliability to degrade.

  R6: Prior studies already suggest to update libraries on the compute infrastructure after testing for unintended effects. Additionally, the system can allow the user to specify which older versions can be used for their jobs to succeed. This way, with the help of a method similar to one used in this paper 7 to estimate library reliability, the system can decide to use the most reliable version of the library for that user. Furthermore, use containers to minimize impact on other jobs [8].

  Refer: Sec 7

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. *First*, this 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 [20,21] or highly instrumented and highly managed cloud clusters [22, 23]. *Second*, 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 due to local contention (such as, for memory on a node) while in some cases we see the effect of 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, for memory for 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 [20, 24], coarse-grained resource utilization metrics [22], or the effect simply of the execution time of the job on its failure probability [25, 26], cannot shed such detailed light.

1.3 Why is it Challenging to Achieve High Reliability

Now let us consider the factors that make it challenging to ensure high reliability for these shared computing clusters and the jobs that run on them.

1.3.1 Diversity of Users and Applications

The user base is composed of students, professors, and research scholars coming from a diverse set of backgrounds, such as computer science, biology, physics, linguistics, and having skills and understanding of computer systems that range from novice to experts. These users execute many different types of applications. Some run a large number of short single-core jobs (such as, parameter sweep for a short simula-

tion), and others execute long-running jobs using many hundreds of cores (such as constructing the spatial configuration of lowest energy proteins). The jobs span the spectrum of resource usage in terms of all local resources (memory, local IO) as well as remote resources (remote IO, remote file system, network bandwidth). The job scripts that are used to submit the jobs to the cluster wrap different applications and are written in free form text. This makes it not amenable to automated parsing and extracting relevant information such as what applications are included in the script.

1.3.2 Diversity of Execution Environments

It is challenging to create execution environments that are optimized for this wide diversity of users and applications. The execution environment encompasses the totality of hardware configurations (*e.g.*, number of cores on a node, amount of memory on a node, the network backplane, etc.) and software configurations. Preinstalling applications and associated libraries is often the only viable solution as the OS versions of packages are often very out of date and not usable as HPC codes often depend on particular versions of libraries. However, this significantly increases recurring work for the cluster administrators in updating these software packages and ensuring that dependencies between them are preserved during any update process.

1.3.3 Understaffing of HPC Departments

Most HPC centers at universities are chronically understaffed, even when the overall IT department is not. To give some representative examples, user support team at TACC@UT Austin has only ∼20-25 people supporting 10K users, RCAC@Purdue has ∼20 technical staff supporting 12 central compute clusters. Thus, the user support system embraces automation and workload analytics to increase management efficiency.

The rest of the paper is structured as follows. In Section 2, we provide details of the two systems while in Section 3 we describe the data sources. In Section 4,

we analyze node and job failure categories and in Section 5, we analyze the impact on job failures of resource usages. Finally, Section 6 presents a job failure prediction model. We then discuss threats to validity, related work and conclude the paper.

2. SYSTEM DETAILS

Table 2.1 provides system specification summary of two university-based HPC System. *System A* is hosted at *U1* and *System B* is hosted at *U2*. The two systems vary significantly in their scale. *System A* comprises 580 nodes of 16-core Intel Xeon E5-2670 processors and Xeon Phi accelerators, while *System B* has a total of 22,640 CPU-only nodes and 4,228 CPU + GPU nodes. The network and IO bandwidth resources also scale accordingly, with the total compute power of *System A* as 1.34 PetaFlops and *System B* as 13.34 PetaFlops.

Table 2.1.: System Detail

Unit	**Spec**	***System A***	***System B***
Compute	Node	580	22,636 XE, 4,228 XK
	Memory	64 GB/node	64 GB/node
	Processor	Xeon E52670 + 2x Xeon Phi/node	AMD 6276 Interlagos, NVIDIA GK110
	Resilience	ECC-protected CPU & Xeon Phi memory	Chipkill-protected CPU memory modules, ECC-protected GPU memory modules
Local I/O	I/O bandwidth	100MB/s	Not present
	Capacity	500 GB	Not present
	Type	SATA	Not present
Network I/O	I/O bandwidth	23GB/s	1.1 TB/s
	Capacity	1.4 PB	26.4 PB
	Resilience	Disks: RAID 6, Index Disks: RAID 1+ 0, OSS : Active-Active HA pair, MDS: Active-Passive HA pair	Data Disks: RAID 6, Index Disks: RAID 1+ 0, OSS : Active-Active HA pair, MDS: Active-Passive HA pair
Network	Peak node inj.	40Gb/s	76.8 Gb/s
	Topology	Fat Tree	Cray Gemini 3D Torus [27]
	Resilience	Infiniband Forward Error Correction (FEC)	Packet: 16-bit packet CRC, links: adaptive load balancing, routers: quiesce and reroute

2.1 Architecture Details

U1 Community Cluster, *System A*, consists of nodes with two 8-core Intel Xeon E5-2670 processors, two Xeon Phi accelerator cards and 64GB of memory. The 580 nodes of *System A* are connected through an FDR Infiniband network at 40Gb/s in a fat-tree topology. Users are provided with a shared home directory as well as a share for widely-used applications; both are available from each node via NFS at approximately 2GB/s. These filesystems are accessed using IP over IB. The parallel filesystem used by the jobs is a Lustre 2.4 [28] installation that can sustain up to 23GB/s and is connected via the above-mentioned IB network. All nodes run RedHat Enterprise Linux (RHEL) [29] major version 6 and are regularly upgraded to the latest minor version, currently 6.9. Along with default RHEL libraries, the environment also provides many other important pre-installed libraries and applications which can be loaded when needed. The nodes are administrated using RedHat Kickstart installers integrated with Puppet configuration management software.

System B is an Open-Science massively parallel processing (MPP) system at *U2*. It is composed of 288 Cray liquid-cooled cabinets hosting 22,640 XE (CPU-only nodes) nodes and 4,228 XK (CPU + GPU) nodes running Cray Linux Environment (CLE) OS. CLE OS includes Cray's customized version of the SUSE Linux Enterprise Server (SLES) 11 Service Pack 3 (SP3) operating system. Each cabinet is further composed of three chassis, and each chassis contains eight blades. Each blade is composed of four compute nodes (XE or XK) and two network switches (Gemini ASICs). A Cray Gemini [27] is a 3D torus-based high-speed interconnection network that connects network nodes within *System B*. The users have access to three Lustre-based remote distributed filesystem –home, project and scratch, provide room for over 24 PB of data with a combined 1.1 TB/s IO rate.

2.2 Job Submission System

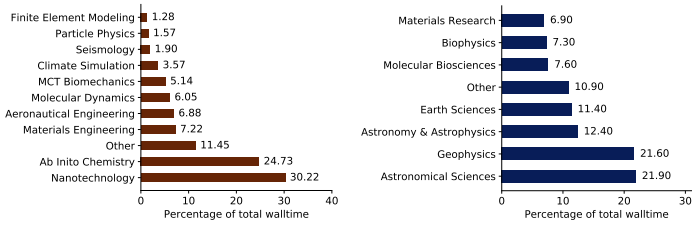
Both *System A* and *System B* use Torque [30] for resource management and batch scheduling. On *System B*, users can pack multiple application execution runs spanning one or more compute nodes. These applications are then placed on to the compute nodes with required modules and environment settings using Cray Application Level Placement Scheduler (ALPS) [31]. Cray ALPS is also responsible for monitoring of node- and applications-health. Users on *System A* can also submit a job packing multiple applications, each spanning single or multiple nodes. However, accounting is done at a job level rather than application level. On *System A*, each faculty who has purchased nodes in the cluster gets an "owner queue" that is the size of their node purchase in cores. On the other hand, users have to write NSF-grant or University-grant proposals to get node-hours on *System B*. Users can run jobs on one or more nodes on *System B* as long as they do not run out of their allocated node-hours.

The job submission scripts can be configured by the user to set number of parameters such as: i) number of nodes, ii) the number of cores, iii) job mapping on system, and iv) the system walltime (i.e., requested clock time for the job). Jobs are submitted to the scheduler and queued for batch run. 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 by another job on the same node. *System B* puts a 48-hour walltime restriction, whereas *System A* has a restriction of 336 hours. During the job run the stdout and stderr of the submission script or application are written to disk and copied to the users working directory after job completion. A job termination status is captured by the TORQUE log in both systems, while for *System B*, ALPS also captures the exit code of each application within a job.

2.3 Job Characteristics

In this section, we compare and contrast the job characteristics of *System A* and *System B* in terms of i) job type, ii) job size, iii) job duration, and iv) job node seconds. *System A* supports 2 kinds of execution environment for jobs i.e. shared and non-shared. Sharing means the node is shared among multiple jobs while non-shared denotes the entire node is used by a single node. *System B* always runs jobs in non-shared environment.

2.3.1 Job Type



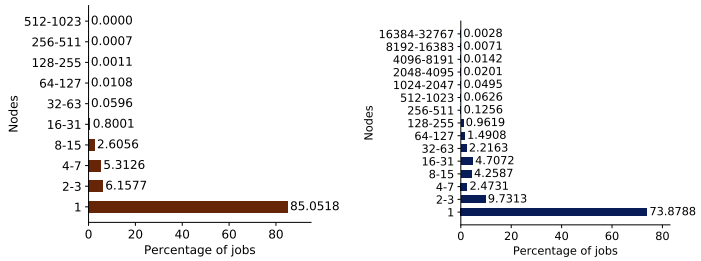
(a) Comparing wallclock time of scientific application domains

Fig. 2.1.: Characterisitcs of jobs running on *System A* (in brown, left in each subfigure) and *System B* (in green, right in each subfigure)

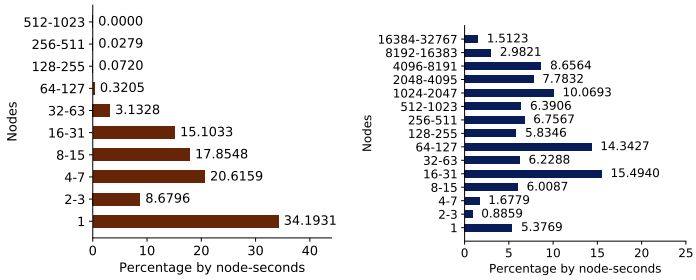
The workloads on *System A* and *System B* predominantly consist of scientific applications but distinct scientific disciplines are covered by the two systems. The most prolific scientific fields researched on *System A* and *System B* are summarized in Figure 2.1(a) in terms of node-hours. The top scientific discipline for *System A* was found to be 'Nanotechnology' (30.7%) whereas it was 'Astronomical Sciences' (21.9%) on *System B*.

Definition 2.3.1 Node-seconds: *is the product of the number of nodes and the wallclock time (in seconds) used by a job. The metric captures the scale of the job's execution across space and time.*

2.3.2 Job Size



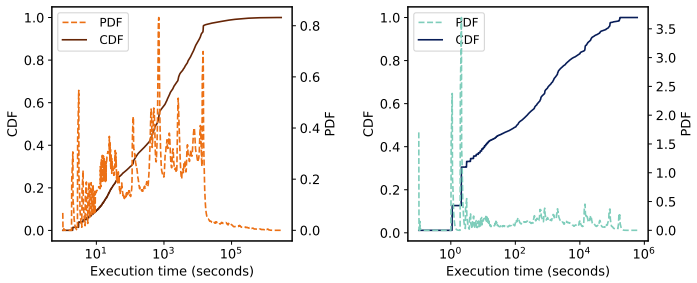
(a) Comparing job sizes in terms of number of nodes



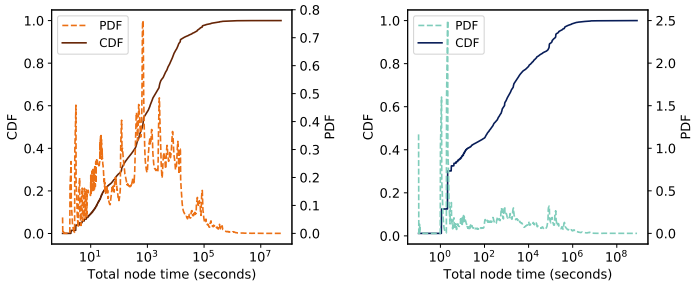
(b) Comparing percentage by node-seconds

Fig. 2.2.: Characterisitcs of jobs running on *System A* (in brown, left in each subfigure) and *System B* (in green, right in each subfigure)

Figure 2.2(b) shows a bar plot of percentage by node-seconds (see Def. 2.3.1). Most of the jobs on both *System A* and *System B* are single-node jobs, 87% and 73% respectively. However, these jobs contribute just 38% (*System A*) and 5% (*System B*) by node-seconds. If we look at the total contribution of multi-node jobs, they contribute to 62% and 94% of the total node-hours on *System A* and *System B* respectively even though the percentage of such jobs are much smaller (13% for *System A* and 26% for *System B*) (Table 1.1). Furthermore, scale of jobs submitted on *System B* is significantly larger than *System A* (refer Fig. 2.2(b)). Percentage distribution by job count for *System A* and *System B* is provided in Figure 2.2(a).



(a) Comparing job wallclock execution time



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

Fig. 2.3.: Characterisitcs of jobs running on *System A* (in brown, left in each subfigure) and *System B* (in green, right in each subfigure)

2.3.3 Job Duration and Job Node-seconds

Figure 2.3(a) shows CDF/PDF distribution of job wall clock execution time for both *System A* and *System B*. On both *System A* and *System B*, 50% of the jobs run for less than $\sim 10^3$ seconds, however on *System B* the jobs run for as long as $\sim 10^9$ node-seconds. Figure 2.3(b) shows the CDF/PDF distribution of node-seconds of the jobs.

2.4 System Management Practices

System A is housed in a data center facility on the University U1 campus. It is housed in water-cooled racks, which used rear-door heat exchangers to dissipate heat. The operation of the system is performed with heavy reliance on a homegrown automated configuration management system using Puppet. Warewolf NHC tools are

integrated into the Torque resource manager and Sensu is used to monitor the overall health of the cluster. Splunk is also used to find and correlate disperate log messages to help diagnose complex problems. The hardware repair staff consists of part-time undergraduate students working under the direction of full-time staff.

System B is housed at a dedicated supercomputing facility equipped with specialized cooling cabinets and redundant power supply and has dedicated maintenance staff at the facility. Further, *System B* employs dedicated system admins from Cray and University U2 at an offsite location who monitor and manage system on a daily basis using automated tools such as ISC, LDMS, CrayPAT, LogDiver, etc. Furthermore, *System B* employs a dedicated application team (consisting of 10 members) that monitor application performance and failures, as well as help big customers of the system to develop and run their code on the system.

3. DESCRIPTION OF DATA

Our analysis of job failure leverages data at various granularities from various system monitoring resources. By analyzing the node, resource, and software utilization in failed jobs; failure can potentially be tied to addressable issues with both job specifications and node health. In this section, we briefly describe these various data resources, their collection and post-processing (if any) methods.

The data used in the analysis reported in this paper falls under four broad categories: job scheduler accounting logs, job-level resource utilization logs, node-level health monitoring logs, and finally node-level software library specifications. Possible explanations for job failure can be obtained by chaining together information from these four sources. In particular, the job scheduler logs for a failed job point to the error condition as well as the nodes that were employed for the job; both job-level resource utilization logs and node-level health logs can be analyzed for what resources on what nodes were used in the failing job. Next we describe each of the four data sources and their processing steps.

Table 3.1.: Dataset

Data Source	***System A***	***System B***
Duration	Mar 2015- Jun 2017	Feb-Jun 2017
Job accounting	Torque logs	Torque logs (stored in ISC [32]) [31.0 GB]
Performance accounting	TACC stats	LDMS logs ([33]) [10 TB]
Node-failure reports*	Kickstand DB (Jan - Dec 2015)	Stored in ISC [60 MB]
Liblist	lsof	-

* Node-failure reports consists of syslogs and human-annotated reports

3.1 Job Accounting

Job accounting logs are obtained from TORQUE for both *System A* and *System B*. TORQUE maintains a record of a batch job as it moves through the job submission queue and eventual execution on the cluster. 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. TORQUE accounting logs contain these records organized by date; each line in an accounting log file corresponds to a single event record for a job processed on that date. The accounting logs employ a fixed format for these records:

```
<date> <timestamp>;<job event>;<jobID.node>;[<stats>]
```

where `[<stats>]` is an array of `<key>=<value>` pairs where the respective keys can depend on the job event being recorded. For instance, a record of a queuing event would only contain `queue=debug`; while a job start event would include the `qtime`, `start`, and `exec_host` keys representing the time the job was queued, the time the job started running, and, the actual nodes the job was executed on, respectively. For *System A*, we processed the raw TORQUE logs, however *System B* uses Integrated System Console (ISC) [32] to parse and store the job records and its associated metrics (performance and failure) in its database.

3.2 Resource Utilization Stats

System A uses TACC Stats [34] and *System B* uses light-weight distributed metric service (LDMS) [33] for collecting resource utilization values. Both are widely used, low-overhead infrastructure for collecting system-wide performance data at regular intervals. 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*. This represents a pragmatic operational choice balancing the overhead incurred with the fidelity of the monitoring data.

3.3 Node Failure Reports

In our analyses, we have collected all node-related failure reports from corresponding management teams of *System A* and *System B*. Management team of *System A* uses Kickstand, a database that contains information about the system infrastructure and error/failure events. These event records can point to planned system outages, reboots, or alerts on unreachable nodes from the Sensu [35] and Nagios [36] monitoring frameworks. The management team of *System B* uses ISC [32] for recording in-depth information about each job and node of the system. We extract only node-events (such as 'admindown', 'up', 'down', 'suspect', and 'unknown') from the databse for getting node-failure reports. This information can then further be used to mark system-wide outages (see Def. 3.3.1). For *System A*, this information is only available for the year 2015.

Definition 3.3.1 System-wide Outage (SWO): *is declared when more than 25% of the nodes fail either due to scheduled maintenance or catastrophic failure (such as power failure or file system failure).*

3.4 Liblist

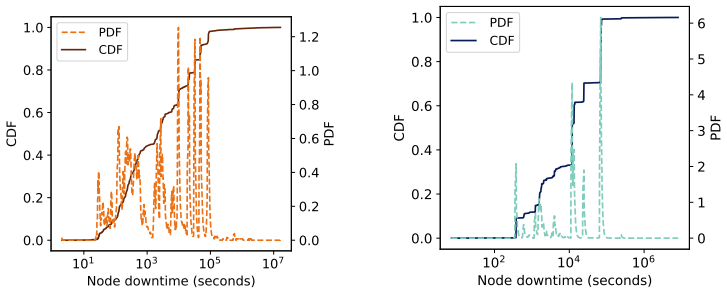
This data is only available for *System A*. A job's failure can be related to the use of a deprecated or buggy software library. Such scenarios can be analyzed by studying the versions of shared libraries employed during a job's execution. This data is obtained by collating periodic snapshots of the shared libraries being utilized by a job. These shared libraries (that are currently in use) are identified using the *lsof* command on the cluster nodes during the job's execution. For our dataset, this data item is collected on each node at a frequency of once every 30 minutes. The library lists are then organized by the username, jobID, date of execution and cluster node. The shared libraries can be further divided into standard Linux operating system libraries and scientific libraries from a specific module loaded in the job script. This division is based on the path to the shared library; those originating in */apps* are

typically scientific libraries, while those loaded from the standard paths, */usr/lib* or */lib* or */usr/lib64* or */lib64* can be considered to be standard Linux libraries.

4. ANALYSIS OF NODE AND JOB FAILURES

In this section, we first analyze node downtime distribution and categorize the causes for node downtime. Then we analyze the causes for job failure based on the job exit status captured in the accounting logs. As before, we remove all debug jobs before doing this analysis.

4.1 Node Downtime Distribution



(a) *System A* (b) *System B*

Fig. 4.1.: Node downtime distribution

In this subsection, we analyze the distribution of node uptime and downtime duration. A node may become unreachable due to permanent failure, i.e. fail-stop failure (such as memory or processor failure), or transient failure (e.g., network rerouting or operating system hang due to high load). Permanent failure requires human operator intervention for repair (e.g., 1-7 days for replacing nodes) whereas transient failures are automatically repaired by the system relatively quickly (e.g., rerouting takes 1-15 minutes). For both *System A* and *System B*, node unreachability or reachability is detected via heartbeats and recorded in a database that we used to characterize the uptime and the downtime duration.

Fig. 4.1 shows the time duration it takes for a node to be repaired *i.e.*, its "offline-online" or downtime duration. A node may become unreachable in a transient manner and become reachable without human intervention, or that may involve human intervention. A node may become unreachable due to the node itself failing, a network service on the node failing, or the network infrastructure to the node failing. Again, the word "failing" may denote either a permanent (*i.e.*, requiring human intervention) or a transient failure. Fig. 4.1(b) shows this result. Due to wide-range of recovery/repair time, we use logarithm scale to study node downtime. We observe that the 50-th percentile value is 0.6 hours for *System A* and 3.3 hours for *System B* while the 95-th percentile value is 24 hours for *System A* and 20 hours for *System B*. For *System A*, this means that for 50% of the cases, the node becomes reachable within 0.6 hours, while in 95% of the cases, it becomes reachable within 24 hours. The data is skewed by the various system wide outage (SWO) events that affect a significant number of nodes in the cluster. A sample of these SWOs is shown in Table 4.1. There are a total of 13 SWOs for *System A* and 4 for *System B* during the duration of the study (1 year and 5 months respectively). These events show up as the pdf peaks in Fig. 4.1.

Table 4.1.: Sample of system wide outages (SWOs). These affect a significant number of nodes and skew the node uptime/downtime statistics.

System	**# Nodes**	**Date (Total time)**	**Outage type**
System A	100%	2015-02-05 (9 hrs)	Maintenance
	100%	2015-01-20 (24 hrs)	Filesystem
	100%	2015-11-04 (10 hrs)	Unknown
	100%	2015-11-28 (6 hrs)	Filesystem
System B	100%	2017-02-11 (4.5 hrs)	Site power interruption
	31.1%	2017-03-25 (2 hrs)	CLE/Lustre
	100%	2017-03-28(25 hrs)	Water Facility
	100%	2017-06-29 (6 hrs)	Emergency Maintenance

Next, we look at the time duration from when a node becomes reachable to when it becomes unreachable, the "online-offline" or uptime duration. Overall, we observe

that for the 50-th percentile data point, the value is 4 days for *System A* and 31.3 days for *System B*, while the 95-th percentile point is 42 days and 92 days respectively.

To understand the causes of node downtime, we analyzed the system logs in consultation with *System B* management staff. Table 4.2 provides a breakdown of node failure reasons for *System B* only[1]. We removed all SWO events for this analysis as they would have skewed this result. For example, a single SWO in *System B* will have more than 6.7K node events compared to 1K overall when we remove all SWOs. As many as 41.2% of non-SWO related node-failures were caused by processor failures whereas cooling accounts for 27.2% failures.

Table 4.2.: Breakdown of Node Failure Reasons in *System B*

Category	**Percentage**
Processor	41%
Cooling	27.2%
CLE/Linux Kernel	13.1%
Memory	8.9%
CLE/Lustre	4.7%
Power Supply	1.2%
Other	3.8%
Total failures	1,058

Implications for System Design: *Higher continuous uptime of nodes in* System B *can be attributed to better resiliency features compared to* System A *(such as use of chipkill-enabled memory modules over ECC) and use of vendor-provided monitoring tools (such as Cray node-health checker). Additionally, unreachability for system A is higher on median compared to system B, and the reason we suspect is longer time needed by admins to reproduce the failure and identify the root cause. Better monitoring and logging reduces the required human effort and lead to lower unreachable time.*

[1] *System A* does not maintain node failure root cause information.

4.2 Job Categories Based on Exit Statuses

Here we use the exit code information from TORQUE logs (*System A* and *System B*) and ALPS logs (for *System B*) to find the probable job failure cause and categorize them based on the approach used in LogDiver [37] used previously on thhe Blue Waters supercomputer [26]. 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 where each exit status represents a different error. Exit reasons are classified into the following categories: i) *'success'*, for applications completing without any errors/failures, ii) *'walltime'*, for applications not completing within the allocated wall clocktime, iii) *'user'*, job failures that are caused by 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'*, when an application is terminated due to system hardware and software errors and failures, and v) *'user/system'*, when an application is terminated for reasons that can be related to either user or system events and it is not possible to disambiguate, 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 scheduler on expiry of requested walltime) and write checkpoint to the file system before exiting. It may not always be possible to checkpoint at the time of walltime exit with the given window, as it may take several minutes to hours to build a consistent checkpoint [38, 39]. 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 which wrote a checkpoint before being terminated (good case and little work is lost) from the jobs which did not (undesirable case with work lost).

Table 4.3.: Job categories based on exit codes. Percentages in brackets are based on the total node hours

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

Table 4.3 shows the categories distribution for both the systems. We have divided the jobs based on execution environment (shared or non-shared) and job type (single node or multi-node). On *System A*, a significant number of jobs failed due to system related errors (5.1%) whereas most common failure category on *System B* is user (3.6%). A significant fraction multi-node jobs on *System A* (16.1%) and *System B* (20.4%) ran till expiry of requested walltime in non-shared environment. Only 61.8% (on *System A*) and 64.0% (on *System B*) of the multi-node jobs in non-shared environment completed successfully. However, on *System A* 87.6% of the shared multi-node jobs completed successfully. Higher success rate may to due to lesser computing requirement as these jobs contributed just 1.8% of the total node hours of *System A*.

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 a good number of these jobs checkpoint frequently. However, even in this case, there is possible loss of computational time between last checkpoint and program termination by the scheduler. The checkpointing interval should be determined based on the MTBF metric and must be calculated on a regular basis to account for runtime conditions like resource exhaustion and node ageing [40].

Implications for System Design: *Contrary to common belief, 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). Therefore, system software even at the scale of* System A *needs careful monitoring and upgrades that take into account dependencies. Both systems have non-trivial number of failures caused by user-related issues (33% on* System A *and 48% on* System B *of all failures). We find through inspection of a sub-sampled set that many errors can be avoided if software infrastructure statically checks for common mis-configurations (such as if the files referenced in the user-code are present in the file system, similar to the approaches presented in [6, 7]). We also believe it will help if users compile and execute their jobs in containerized environments, which simulate the environment as the cluster, but at a smaller scale. Thus reducing failures due to user issues.*

5. EFFECT OF RESOURCE USAGES ON JOB FAILURES

The section studies the influence of resource usage on probability of job failure

Table 5.1.: Hypotheses results containing the Pearson correlation coefficients and their corresponding p-values. All accepted hypotheses are in either green or red. Green represents positive correlation while red represents negative correlation

Null Hypothesis: $\alpha > 0.01$ Failure rate is not correlated with	*System A*			*System B*		Reference
	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	5.1
H_0^2: Local I/O	-0.41, 2.0e-4	0.12, 0.4	0.15, 0.2	-	-	5.2
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	5.3
$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	5.4
H_0^5: Node-seconds	-0.36, 9.8e-5	-0.25, 0.01	-0.31, 1.1e-2	0.04, 0.6	0.42, 2.1e-05	5.5

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. The CPU cores are never oversubscribed and therefore do not contribute to job failures. As before, we calculate failure rate after removing all debug as well as walltimed jobs. We define **job failure rate** as the fraction of jobs that fail due to system-related issues. Our focus on system-related issues, as opposed to user-related issues, is due to the fact that here we are analyzing the impact of *system-provided* resources: memory, file system local I/O as well as remote parallel file system I/O and network usage. 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.

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 much more effectively than resource utilization value captured throughout the application run. The job failure rate was statistically found

to be highly correlated to tail resource utilization compared to max, mean, or median resource utilization. Moreover, the correlation between job failure rate and resource utilization gets weaker as we move away from the tail toward the start. The quantitative evidence for this is shown in **Appendix**. Therefore, in this section, all analyses are conducted using tail resource utilization values.

Definition 5.0.1 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. For* System A *measurement window is five minutes whereas for* System B *the measurement window is one minute.*

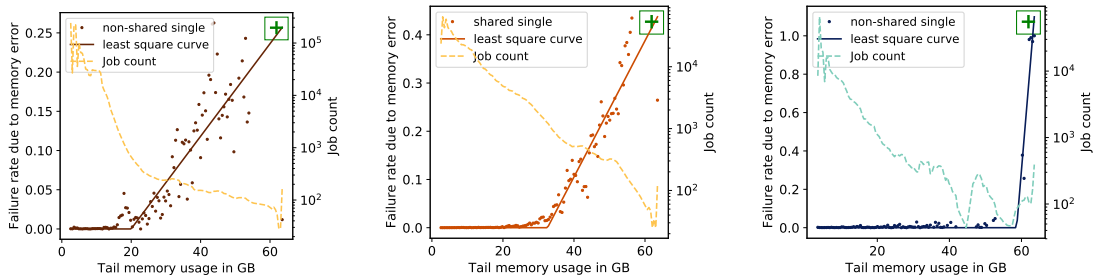
Since resource usages can vary widely across jobs, for all the analyses in this section, we first define equal-sized bins across the range of given resource usages. Jobs are grouped based on the bin's resource usage ranges and then the failure rate of each bin is calculated as the fraction of jobs that 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. 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 hypothesis testing for all 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 5.1. When the null hypothesis is rejected, in some cases, the failure rate is positively correlated while in some others, it is negatively correlated.

We remove plots for inconclusive results to save space. Each figure with a 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 not significant, if the hypothesis testing is significant, then the effect is validated.

5.1 Memory

Null Hypothesis H_0^1: *Job failure rate is not correlated with high memory utilization.*



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

Fig. 5.1.: Failure rate distribution vs tail memory usages.

Table 5.2.: Estimated least square fit for failure rate (f) vs memory usage (x). Parameters are in the form a, b where $f = 0$ for $x \leq a$ else $f = b(x - a)$.

Category	Reference	Parameter	R-square
System A non-shared single	Fig. 5.1(a)	20.6, 6.2e-3	0.72
System A shared single	Fig. 5.1(b)	32.6, 1.4e-2	0.93
System B non-shared single	Fig. 5.1(c)	58.5, 2.1e-1	0.97

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 range of memory usages.

Here the job failure rate is the fraction of jobs that fail with OOM error. By memory usage, 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. 5.1 shows the distribution of failure due to OOM error for different amounts of node memory consumption in shared and non-shared environments. The results of the hypothesis testing are shown in Table 5.1.

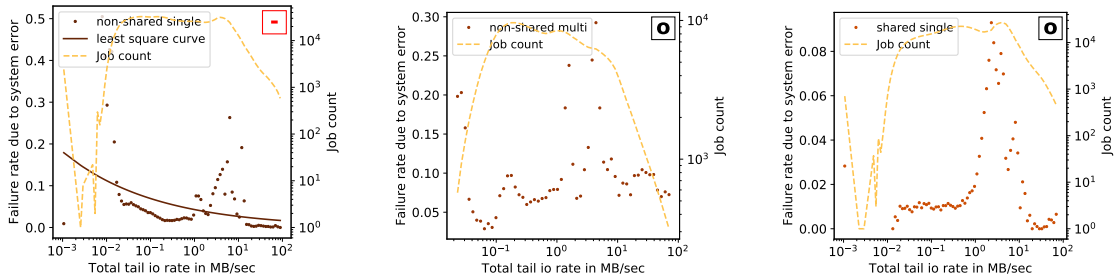
We observe that the failure due to memory error distribution is positively correlated with the tail memory usage of non-shared single and shared single jobs for *System A* as well as single jobs for *System B*. Recollect that on *System B*, all jobs run in non-shared mode. While the positive correlation is expected, it is quite surprising to see the likelihood of failure increasing even when the available free memory is more than half of the total node memory capacity for *System A*. In shared environment, this memory usage is the aggregated memory usages of the jobs running during the tail interval of a given job. *System B* exhibits the expected behavior where the failure rate is flat till close to node memory capacity and then jumps quickly 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 was accepted.

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 previously with the OOM killer in Linux [13]. (Table 5.2).

Implications for System Design: *The analysis of* System A *job failures shows that job failure rate caused by OOM error increases with increasing tail-memory utilization. Application of data mining would help to determine when, in terms of its memory utilization, a job should be pre-empted and moved to a larger node.*

5.2 Local I/O

Null Hypothesis H_0^2: *Job failure rate is not correlated with local I/O rate.*



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

Fig. 5.2.: Failure rate distribution vs total tail local I/O rates

In this section, we conducted an analysis to conclude if total I/O on a node impacts job failure likelihood. Fig. 5.2 shows the results for *System A*. *System B* has no local storage and always uses NFS for I/O; hence, this analysis is not applicable for *System B*.

Fig. 5.2(a) corresponding to non-shared single jobs for *System A*, shows an overall negative correlation which initially appears counter-intuitive. Any I/O related issues usually restrict I/O usage and hence jobs fail with lower I/O rate. On the other hand an I/O heavy job unimpeded by any system-related issues can perform the required I/O operations at much higher rate. This analysis reveals that a good number jobs are failing due to systems issues such as disk failure or faulty drivers that restrict I/O usages. We also observe a peak in the failure rate around 6 MB/s which is much lower than the specified operating I/O limit of available local discs (100 MB/s) at nodes of *System A*. The least square fit in Fig. 5.2(a) 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. The result is inconclusive (null hypothesis accepted with p-value of 0.4) for non-shared multi jobs for *System A*.

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. 5.2(c). In case of shared environment, this total I/O rate is the aggregated I/O rate of all the jobs running at the node during a given time. So as the total I/O rate increases, interference between multiple jobs requiring I/O service increases and that results in the higher failure rate. We see a dip in the job failure rate to the right end of the Fig. 5.2(c). This higher I/O above the local I/O operating limit is due to caching which is captured as I/O 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 [41, 42]) than sequential/specified access rates and such accesses are more common in shared.

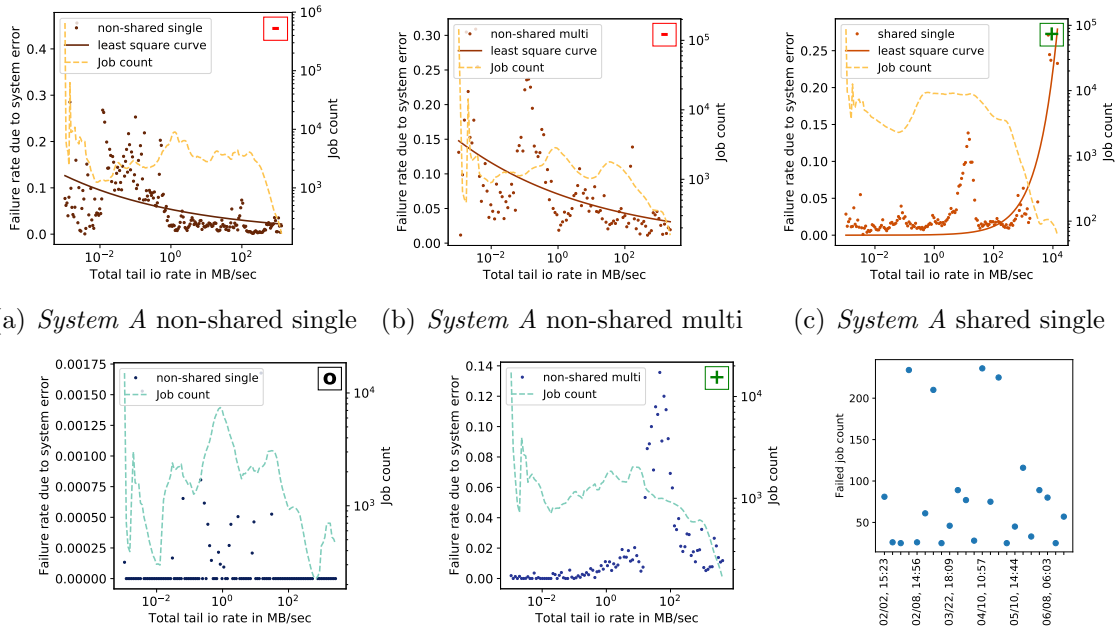
Implications for System Design: *This kind of analysis can be useful for identifying contention issues in the local I/O. A pattern like the peak in the job failure rate can help in estimating the actual I/O usage bound for local storage which can be much smaller than the rated capacity. For instance, the specified I/O limit is 100 MB/s for* System A *local storage, however, we observe higher failure rate even at 6 MB/s and 3 MB/s in case of non-shared single and shared single jobs. For shared environment, we recommend the admins of the system to adapt resource isolation technologies (such as Docker [19]). This can reduce cascading failures due to local contention.*

5.3 Network File System

Null Hypothesis H_0^3**:** *Job failure rate is not correlated with network file system usage rate.*

Table 5.3.: Estimated least square fit for failure rate (f) vs network file system usage rate (x) in log scale. Parameters are in the form a, b where $f = ae^{bx}$.

Category	**Reference**	**Parameters**	R^2
System A non-shared single	Fig. 5.3(a)	0.05, -0.29	0.26
System A non-shared multi	Fig. 5.3(b)	0.07, -0.25	0.32
System A shared single	Fig. 5.3(c)	6.4e-4, 1.46	0.38



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

(d) *System B* non-shared single (e) *System B* non-shared multi (f) *System B*; exit code 107

Fig. 5.3.: Failure rate distribution vs total tail network file system I/O rates

This section studies job failure correlation with total remote storage usage rate. Fig. 5.3 shows the results for *System A* and *System B* and the corresponding parameters of least square curves are given in Table 5.3. Hypothesis testing gives statistically significant negative correlation with respect to total tail remote usage rate for non-shared single and non-shared multi jobs for *System A* in Fig. 5.3(a) and Fig. 5.3(b). These results can both be explained by the fact that during congestion at the remote storage server, jobs fail with low I/O usage rate. Bad network is another factor responsible for job failure with low tail I/O 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. 5.3(c). Remember again that the resource usage in a shared mode is the aggregated resource usage across *all* jobs running on the node at the given time. Here the 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 paramount for a shared environment. The metric being measured (total remote I/O) is *per node* and therefore we see a negative correlation of failure rate in the non-shared environment while we see a positive correlation in the shared environment.

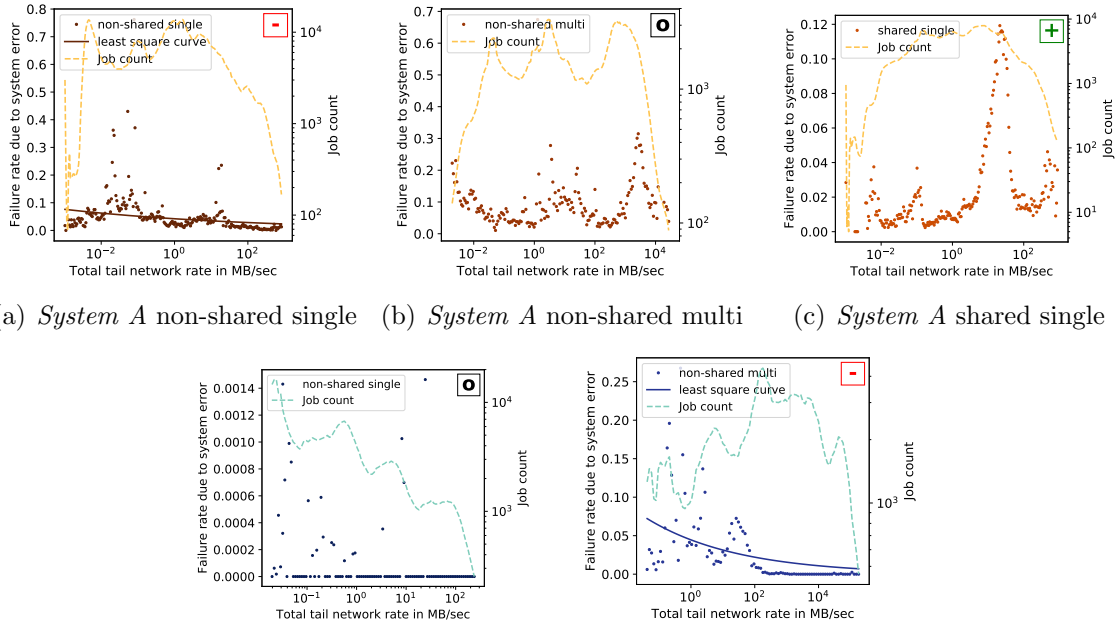
In case of *System B*, no correlation is observed between failure rate and total remote I/O usage rate for single-node jobs (Fig. 5.3(d)). Moreover, the failure rate distribution is flat for the whole remote I/O range. This is expected for *System B* since the peak read/write rate by a single node is 76.8 Gb/sec (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 [46*25k(nodes)∼1.1TB/sec] (Fig. 5.3(e)). On investigating deeper and analyzing all the failed job 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 network file system. 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 DBSCAN algorithm [43] 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 with exit code 107. A total of 26 clusters are found using this approach and are shown in Fig. 5.3(f) along with the total number of failures that belong to the cluster (on y-axis). The median time between occurrence of these burst failures (i.e., more than 25 jobs failing within a short 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 file system (remaining clusters). The effect of network congestion on job failure is described later in Section 5.4.

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 net usage of this resource and to stagger the I/O requests from different applications at times of contention.*

5.4 Network

Null Hypothesis H_0^4: *Job failure rate is not correlated with network usage rate.*



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

(d) *System B* non-shared single (e) *System B* non-shared multi

Fig. 5.4.: Failure rate distribution for different ranges of total tail network rates.

Table 5.4.: Estimated least square fit for failure rate (f) vs network usage rate (x) in log scale. Parameters are in the form a, b where $f = ae^{bx}$.

Category	Reference	Parameter	R-square
System A non-shared single	Fig. 5.4(a)	0.04, -0.20	0.06
System B shared single	Fig. 5.4(e)	0.04, -0.35	0.26

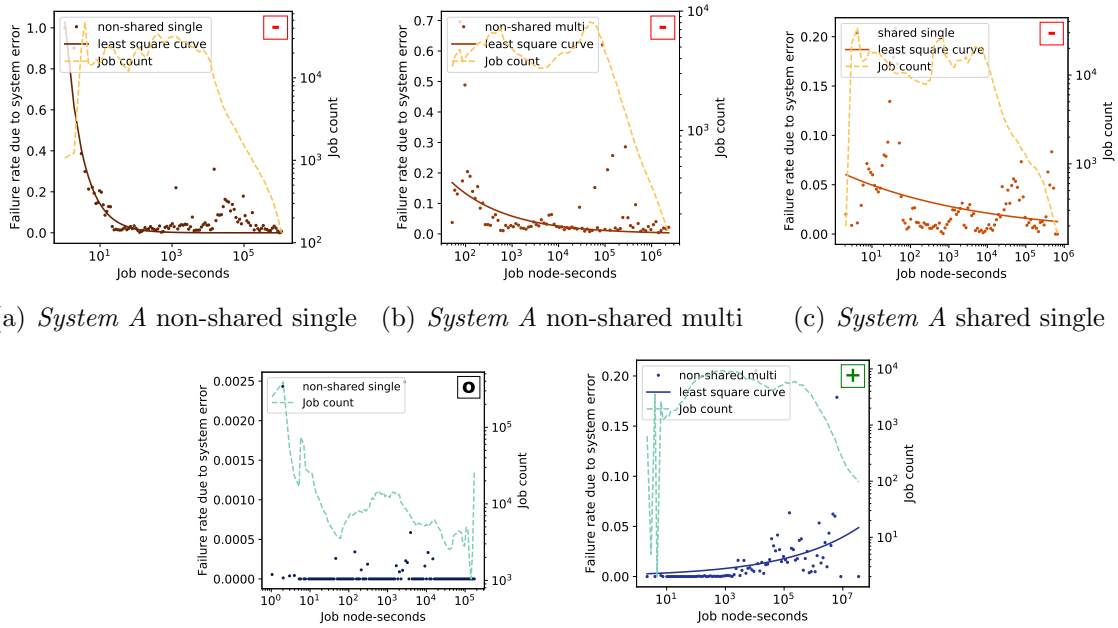
This section analyzes the failure rate distribution for different ranges of total network rates at nodes. For *System A*, remote I/O traffic goes through the same infrastructure as network and so the network usage includes the remote I/O usage plus the network traffic of multi-node jobs. Fig. 5.4 shows the results for *System A* and *System B*. No significant correlation is found for non-shared multi-node jobs of *System A* and non-shared single-node jobs of *System B*. However, non-shared single jobs for *System A* and non-shared multi jobs for *System B* are negatively correlated with tail I/O usage rate. As discussed earlier, negative correlation implies system issues such contention or bad network cause jobs to fail with low tail network usage. We observe overall positive correlation for shared single-node jobs for *System A* with a peak close to 30 MB/s.

5.5 Job Node-seconds

Null Hypothesis H_0^5: *Job failure rate is not correlated with the total job node-seconds.*

Here we analyze the effect of the total node-seconds of a job (refer Sec. 2.3) on the job failure rate. Prior studies have found that there is a positive correlation of failure rate with the total execution time of an application [25,26]. For example, [26] 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 (similar to our metric of job node seconds, with a different time unit). Here we analyze to see if a similar qualitative and quantitative 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. 5.5(a)), non-shared multi (Fig. 5.5(b)), and shared single (Negative correlation with distribution similar to Fig. 5.5(a). 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



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

(d) *System B* non-shared single (e) *System B* non-shared multi

Fig. 5.5.: Failure rate distribution for different ranges of node-seconds.

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. Further, they are not demanding enough in terms of system resources (network fabric, memory, etc.) to create resource exhaustion due to software ageing.

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. Such jobs are exposed to more faults in space or time or both. Interestingly, a significant fraction of failed jobs (including all failure categories i.e. *'user','system','user/system'*) fail with job execution time less than 1 minute—34% for *System A* and 45% for *System B*.

Implications for System Design: *A high level insight we gather is that with mature codes, the job failure rate goes up the longer the job executes. However, with naïve usage models, the failure rate is high for short jobs. These cause inefficient usage of cluster resources due to the constant overhead of scheduling, initiating, and terminating execution. If such errors can be caught through static and dynamic analysis, this would be beneficial for overall cluster utilization.*

6. PREDICTING JOB FAILURES

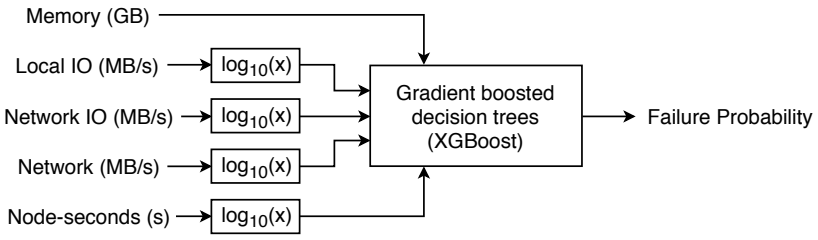


Fig. 6.1.: ML model for job failure prediction

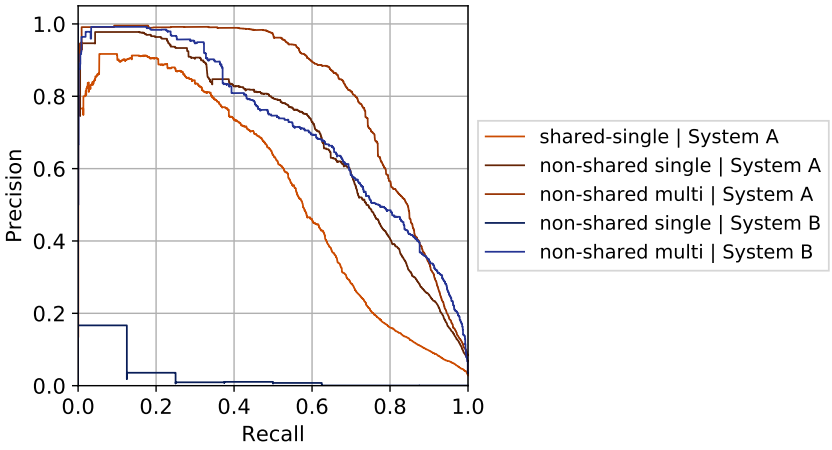


Fig. 6.2.: Precision vs recall curves

Table 6.1.: Maximum net savings of prediction models

	System A			*System B*	
	shared	non-shared		non-shared	
	single	single	multi	single	multi
Max Saving	48%	61%	66%	–	64%
Precision	0.38	0.39	0.45	–	0.41
Recall	0.66	0.81	0.85	–	0.85

The previous section shows that the job failure rate is correlated with different resource usages. The question we pose ourselves is can we predict impending failures so as to take mitigation actions, such as, taking a checkpoint and migrating the process. This motivates us to explore different ML models to predict job failures using the job usages for different resources as input features. After trying different ML models such as linear regression, logistic regression, and decision trees, we find that the gradient boosted decision trees performs the best based on their recall and precision scores. Fig. 6.2 shows the precision recall curves for different execution environments and job types for both systems. These models have been implemented using the 'XGBoost' package in Python (refer Figure 6.1). We observe that model performance corresponding to the non-shared single job types of *System B* is not good enough to be any value. This is expected as the number of jobs belonging to failure category due to system issues is insignificant (0.1%, refer Table 4.3) to identify a differentiating pattern between positive and negative classes. Additionally, precision scores of multi job types are better than corresponding single job types. This we suspect is because the diversity in the single node jobs is significantly higher than in the multi node jobs. The number of unique users submitting single node jobs are significantly more than that of multi node jobs. For instance, number of unique users with non-shared single and non-shared multi job types are 569 and 338 respectively for *System A*.

These job failure prediction models can help reduce resource wastage on these systems by triggering a checkpoint when failure is imminent. The next analysis shows how much saving can be achieved using these models. Let the number of total failures be x, total number of failures predicted by the model be y, precision of the model be P, recall of the model be R, average execution time of the job be t_j and checkpointing overhead of the job be C_o. Checkpointing overhead of C_o means that the time spent in checkpointing state is $C_o\%$ of the time of execution of the job that has generated that state. Then, True Positive $= Py = Rx$ or $y = \frac{Rx}{P}$. Total time in checkpointing, $T_c = C_o y t_j$. Saving by model, $T_s = Rxt_j$. Here saving is the

execution time that would be saved because we would predict the failure and initiate checkpointing. Now, for saving to be greater than 0, $T_s \geq T_c \Rightarrow Rxt_j \geq C_o y t_j \Rightarrow Rx \geq C_o \frac{Rx}{P} \Rightarrow P \geq C_o$ i.e., net saving is greater than 0 whenever precision (P) of the model is greater than checkpointing overhead (C_o) of the job. So, for a checkpointing overhead of 10%, there is a net saving whenever precision of the model is greater than 0.1. Net saving, $N_s = T_s - T_c = Rx\big(1 - \frac{C_o}{P}\big)t_j$. We now calculate the maximum net savings over all possible (precision, recall) pairs, for $C_o = 10\%$. The results are shown in Table 6.1, together with the precision and recall that achieved the maximum savings (as percentage of total time wastage due to system related issues). The chosen checkpointing overhead ($C_o = 10\%$) is a reasonably conservative assumption; consider for example real production jobs have overhead in the range 3-5% [44].

7. EFFECT OF LIBRARY UPDATES ON RELIABILITY

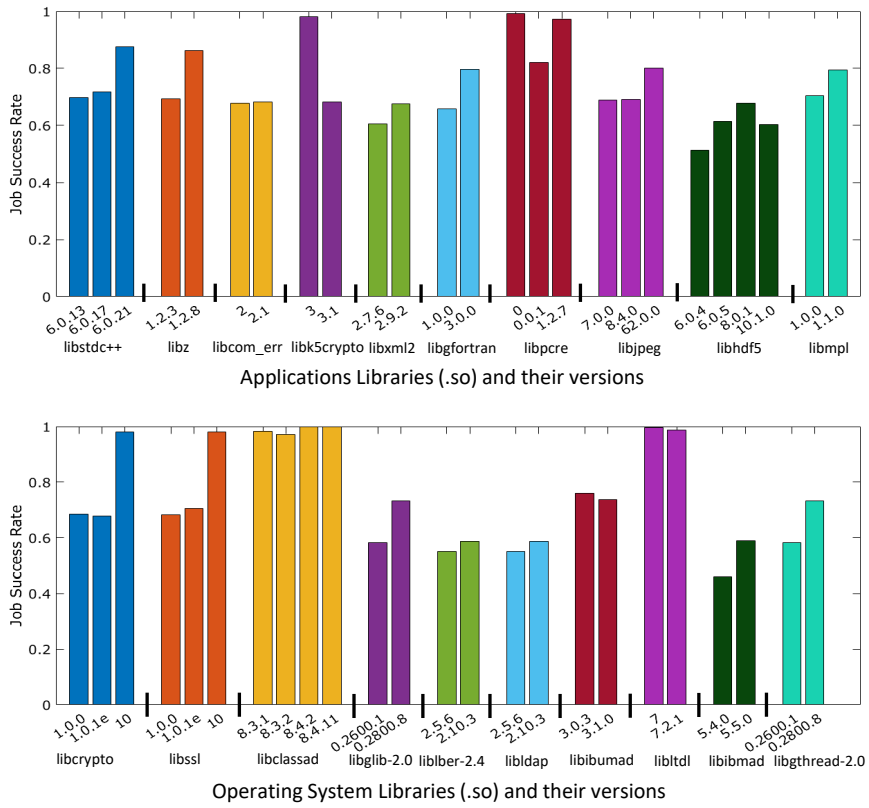


Fig. 7.1.: *System A*: Success rate of top ten application libraries and operating system libraries with different versions. These are sorted by their job usage counts with the most used library on the left of each plot.

Upgrades to dynamic libraries may positively or negatively impact the reliability of an HPC computing environment, as measured by the job failure rate for the jobs that use that dynamic library. A dynamic library is said to be used by the job if it is dynamically loaded during the execution of the job. We conducted an analysis to study the relationship between job success rate and dynamic software libraries for

System A[1]. In this study, we ignored: i) all libraries for which only one version was observed across all the studied jobs, and ii) library versions that were used less than 1,000 times for statistical reasons as well as to discard unstable or unpopular versions. We have defined application libraries to be libraries that are installed under */apps* directory, while operating system libraries are those that are installed under */usr/lib*, */lib*, */usr/lib64*, or */lib64*.

Fig. 7.1 demonstrates that job success rate, in general, increases with the increasing library version number. For thirteen out of the top twenty libraries (of both applications and operating system libraries), the job success rate monotonically increases with increasing version number. Only three libraries, namely *libibumad*, *libltdl*, *libk*5*crypto*, demonstrate decreasing job success rate with increasing version number. The resulting trend could be due to both user script errors and library implementation bugs. By Linux conventions, minor number changes (on the least significant digits) indicate bug fixes, while major number changes (on the most significant digits) indicate potential API changes [45]. Thus, it is reasonable for the likelihood of library failure to decrease from minor number changes due to bug fixes, as apparent in trends of *libstdc++* and *libcom_err* in Fig. 7.1. Furthermore, Fig. 7.1 shows more significant success rate gains from major number changes, as shown in *libcrypto*, *libssl*, and *libxml*2. A possible explanation is that major changes often alter the API of a library, leading to better developer interfaces and increased functionality, which would, in turn, reduce the likelihood of user script errors. In a software defects dataset compiled by Ullah *et al.* [46], it is shown that five out of seven major open source applications (i.e., GNOME, C++ Standard Library, HTTP Server, XML Beans, and ESME) have monotonically decreasing software defects with increasing version number. Our empirical data validates this prior result in the different context of dynamic libraries.

[1]This information was not available for jobs running on *System B*.

8. RELATED WORK

Failure analysis of large-scale computing systems [26, 34, 47–49] has been done in the past on many scopes of analysis, such as job-level, application-level, node-level, or system-level. Di Martino *et al.* [47] provided detailed characterization of system-wide outages. Similarly, Schroeder *et al.* [20] used statistical properties, such as the time between failures and the time to repair, to analyze node-level failures of clusters. Gupta *et al.* [49] studied temporal recurrence relationship between failure types on different systems. In contrast, Mitra *et al.* [48] 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 [26], 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 interference among jobs.

9. THREATS TO VALIDITY

Inferring failure from exit statuses. Most of 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 [50], there is no compulsion for a job script or application developer to follow the guideline. However, it is generally found by others [51,52] and us here that application developers of the popular applications (which also 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 code status.

I/O Caching and Utilization. We determine the I/O utilization using the `llite` counter. *Not* all 'read_bytes' and 'write_bytes' values recorded through `llite` 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 [33].

10. SUMMARY

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—job failures, node failures and recovery, and job resource usages, and dynamic libraries —we bring out important insights into how the clusters behave and implications for how they can be managed more effectively. Some key findings are: (i) the continuous node availability is less than in previous studies of dedicated commercial supercomputing clusters, while the differences in resiliency practices among *System A* and *System B* makes the latter's 95-th percentile availability 2.2X higher. (ii) A significant fraction of jobs hit against the walltime (33% and 43% for the two systems) and therefore event-driven, efficient application-level checkpointing is needed to avoid lost work. (iii) 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 for both systems). These have different implications for how resource contention should be handled. (iv) A significant fraction of jobs terminate early due to user-related errors. This leads to lost overhead in the clusters and should be avoided through static analysis of submitted job scripts to identify incorrect usage. 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.

REFERENCES

REFERENCES

[1] S. Bagchi, R. Kumar, R. Kalyanam, S. Harrell, C. A. Ellis, and C. Song, "Fresco: Open source data repository for computational usage and failures," Jan 2019. [Online]. Available: http://www.purdue.edu/fresco/

[2] S. Kaufmann and B. Homer, "Craypat-cray x1 performance analysis tool," *Cray User Group (May 2003)*, 2003.

[3] C. D. Martino, S. Jha, W. Kramer, Z. Kalbarczyk, and R. 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*. ACM, 2015, pp. 11–18.

[4] K. Park and V. S. Pai, "Comon: a mostly-scalable monitoring system for planetlab," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 1, pp. 65–74, 2006.

[5] A. K. Maji, S. Mitra, B. Zhou, S. Bagchi, and A. Verma, "Mitigating interference in cloud services by middleware reconfiguration," in *Proceedings of the 15th International Middleware Conference*. ACM, 2014, pp. 277–288.

[6] T. Xu, X. Jin, P. Huang, Y. Zhou, S. Lu, L. Jin, and S. Pasupathy, "Early detection of configuration errors to reduce failure damage." in *OSDI*, 2016, pp. 619–634.

[7] M. Attariyan and J. Flinn, "Automating configuration troubleshooting with dynamic information flow analysis." in *OSDI*, vol. 10, no. 2010, 2010, pp. 1–14.

[8] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, p. 2, 2014.

[9] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future Gener. Comput. Syst.*, vol. 22, no. 3, pp. 303–312, Feb. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.future.2004.11.016

[10] T. Z. Islam, K. Mohror, S. Bagchi, A. Moody, B. R. De Supinski, and R. Eigenmann, "Mcrengine: a scalable checkpointing system using data-aware aggregation and compression," *Scientific Programming*, vol. 21, no. 3-4, pp. 149–163, 2013.

[11] Y. 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*, April 2008, pp. 1–9.

[12] R. McKenna, S. N. Herbein, A. Moody, T. Gamblin, and M. Taufer, "Machine learning predictions of runtime and IO traffic on high-end clusters," in *IEEE Cluster*, 2016, pp. 255–258.

[13] T. G. 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/.

[14] "Detecting and avoiding stack overflow in embedded systems," https://www.iar.com/support/resources/articles/detecting-and-avoiding-stack-overflow-in-embedded-systems/.

[15] T. Taghavi, M. Lupetini, and Y. Kretchmer, "Compute job memory recommender system using machine learning," in *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 609–616. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939717

[16] J. Li, C. Pu, Y. Chen, V. Talwar, and D. Milojicic, "Improving preemptive scheduling with application-transparent checkpointing in shared clusters," 11 2015, pp. 222–234.

[17] H. Yang, A. Breslow, J. Mars, and L. Tang, "Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers," in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3. ACM, 2013, pp. 607–618.

[18] C. Delimitrou and C. Kozyrakis, "Paragon: Qos-aware scheduling for heterogeneous datacenters," in *ACM SIGPLAN Notices*, vol. 48, no. 4. ACM, 2013, pp. 77–88.

[19] "Docker," https://docs.docker.com/.

[20] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 4, pp. 337–350, 2010.

[21] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir, "Toward exascale resilience: 2014 update," *Supercomputing frontiers and innovations*, vol. 1, no. 1, pp. 5–28, 2014.

[22] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of the Third ACM Symposium on Cloud Computing*. ACM, 2012, p. 7.

[23] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 2015, p. 18.

[24] X. Chen, C.-D. Lu, and K. Pattabiraman, "Failure analysis of jobs in compute clouds: A google cluster case study," in *Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on*. IEEE, 2014, pp. 167–177.

[25] D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "A survey of software aging and rejuvenation studies," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 10, no. 1, p. 8, 2014.

[26] C. Di Martino, W. Kramer, Z. Kalbarczyk, and R. 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*. IEEE, 2015, pp. 25–36.

[27] R. Alverson, D. Roweth, and L. Kaplan, "The gemini system interconnect," in *2010 18th IEEE Symposium on High Performance Interconnects*. IEEE, 2010, pp. 83–87.

[28] P. J. Braam and P. Schwan, "Lustre: The intergalactic file system," in *Ottawa Linux Symposium*, 2002, p. 50.

[29] "https://www.redhat.com/en," Accessed: 2018-09-24.

[30] G. Staples, "Torque resource manager," in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, ser. SC '06. New York, NY, USA: ACM, 2006. [Online]. Available: http://doi.acm.org/10.1145/1188455.1188464

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

[32] J. Fullop *et al.*, "A diagnostic utility for analyzing periods of degraded job performance," in *Proc. Cray User Group*, 2014.

[33] A. Agelastos, B. Allan, J. Brandt, P. Cassella, J. Enos, J. Fullop, A. Gentile, S. Monk, N. Naksinehaboon, J. 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*. IEEE, 2014, pp. 154–165.

[34] T. Evans, W. L. Barth, J. C. Browne, R. L. DeLeon, T. R. Furlani, S. M. Gallo, M. D. Jones, and A. K. Patra, "Comprehensive resource use monitoring for hpc systems with tacc stats," in *Proceedings of the First International Workshop on HPC User Support Tools*. IEEE Press, 2014, pp. 13–21.

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

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

[37] C. D. Martino, S. Jha, W. Kramer, Z. Kalbarczyk, and R. 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*. ACM, 2015, pp. 11–18.

[38] K. M. Chandy and L. Lamport, "Distributed snapshots: Determining global states of distributed systems," *ACM Transactions on Computer Systems (TOCS)*, vol. 3, no. 1, pp. 63–75, 1985.

[39] A. Bouteiller, P. Lemarinier, G. Krawezik, and F. Cappello, "Coordinated checkpoint versus message log for fault tolerant mpi," in *null*. IEEE, 2003, p. 242.

[40] G.-A. Klutke, P. C. Kiessler, and M. A. Wortman, "A critical look at the bathtub curve," *IEEE Transactions on reliability*, vol. 52, no. 1, pp. 125–129, 2003.

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

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

[43] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[44] R. Gupta, H. Naik, and P. Beckman, "Understanding checkpointing overheads on massive-scale systems: Analysis on the ibm blue gene/p system," *IJHPCA*, vol. 25, pp. 180–192, 05 2011.

[45] P. Seebach, "Dissecting shared libraries," Jan 2005. [Online]. Available: https://www.ibm.com/developerworks/linux/library/l-shlibs/index.html

[46] N. Ullah, M. Morisio, and A. Vetrò, "Selecting the best reliability model to predict residual defects in open source software," *Computer*, vol. 48, no. 6, pp. 50–58, 2015.

[47] C. Di Martino, Z. Kalbarczyk, R. K. Iyer, F. Baccanico, J. Fullop, and W. 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*. IEEE, 2014, pp. 610–621.

[48] S. Mitra, S. Javagal, A. K. Maji, T. Gamblin, A. Moody, S. Harrell, and S. Bagchi, "A study of failures in community clusters: The case of conte," in *Software Reliability Engineering Workshops (ISSREW), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 189–196.

[49] S. Gupta, T. Patel, C. Engelmann, and D. 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*, ser. SC '17. New York, NY, USA: ACM, 2017, pp. 44:1–44:12. [Online]. Available: http://doi.acm.org/10.1145/3126908.3126937

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

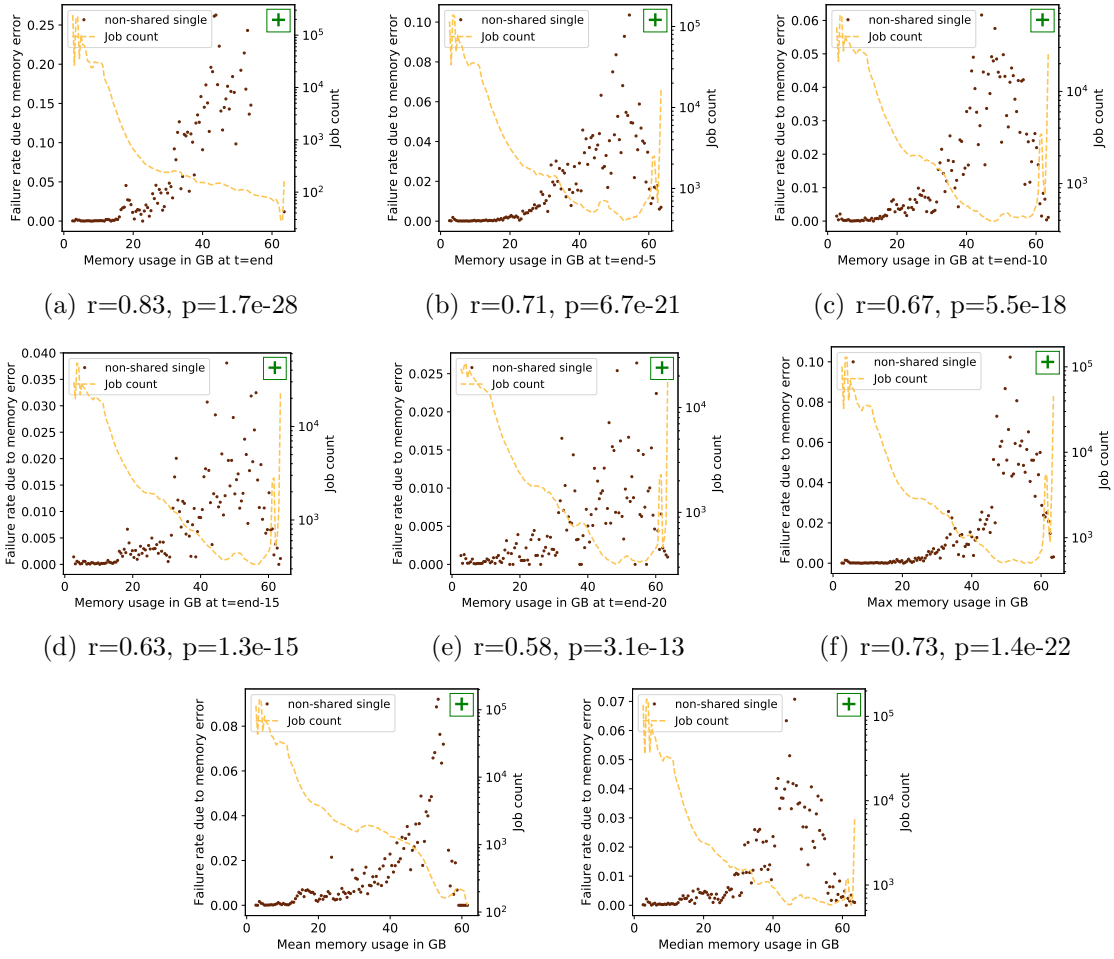
[51] M. Cooper, "Advanced Bash-Scripting Guide," http://tldp.org/LDP/abs/html/exitcodes.html, 2014.

[52] S. 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.

APPENDIX

A. APPENDIX

A.1 Resource Usage Correlation



(a) r=0.83, p=1.7e-28 (b) r=0.71, p=6.7e-21 (c) r=0.67, p=5.5e-18

(d) r=0.63, p=1.3e-15 (e) r=0.58, p=3.1e-13 (f) r=0.73, p=1.4e-22

(g) r=0.59, p=1.3e-12 (h) r=0.47, p=1.4e-8

Fig. A.1.: Failure rate (OOM error) distribution with different metrics

This section provides experimental validation regarding the metric most suitable for correlation study. The metrics are last 5 data points, max, mean and median of node memory usages of each job. Fig. A.1 shows the failure rate distribution

with respect to each of above mentioned metric along with the Pearson correlation coefficients and corresponding p-values for non-shared single node jobs for *System A*. The study reveals the failure rate is most strongly correlated with the memory usage at the end of jobs. The correlation coefficient decreases monotonically from 0.83 to 0.58 as the data point moves farther away from the end. Relatively weaker correlation is also observed in case max (0.73), mean (0.59) and median (0.47) memory usages as compared to memory usage at the tail (0.83).