Exploring the Trade-off Between Repair Time and Reliability in Large Scale Cluster Computers: A Simulation-Based Approach

Leslie A. Horace*
College of Computing
Georgia Institute of Technology
Atlanta, GA, USA
lhorace3@gatech.edu

Craig S. Walker*
Department of Computing Sciences
Coastal Carolina University
Conway, SC, USA
cswalke1@coastal.edu

William M. Jones†
Department of Computing Sciences
Coastal Carolina University
Conway, SC, USA
wjones@coastal.edu

Nathan A. DeBardeleben
High Performance Computing Design
Los Alamos National Laboratory
Los Alamos, NM, USA
ndebard@lanl.gov

Vivian E. Hafener
High Performance Computing Env.
Los Alamos National Laboratory
Los Alamos, NM, USA
vhafener@lanl.gov

Steven T. Senator†
High Performance Computing Env.
Los Alamos National Laboratory
Los Alamos, NM, USA
sts@lanl.gov

***Abstract*—As the size of high performance computing (HPC) computational clusters continues to increase in performance, scale and component count, the role that reliability and particularly the repair time plays a significant role in system specification, procurement, and ultimate operation of such systems. System administrators must find a balance among competing factors: initial capital investment, operational costs and observed system performance and utility from the end users' perspectives are chief among them. In this paper, we explore the trade-off between reliability, performance and node repair times in large-scale high performance computing (HPC) computational clusters using real historical workloads from Los Alamos National Laboratory (LANL). We enhance an existing cluster simulator to more quickly perform the large-scale parameter sweeps necessary to obtain meaningful results for these studies, in some cases by several orders of magnitude. Our results show that these simulations can be parameterized to identify trends that can be used to make decisions about system procurement and operation as a function of the operational parameters and constraints.**

***Index Terms*—parallel job scheduling, resilience, simulation, modeling.**

I. Introduction

Supercomputers have long been important fixtures in the scientific community. The size, scope and capability of these systems have grown considerably over the last two decades. It is becoming more common that these capabilities are provided by specialized components, including general-purpose graphics processing units (GPGPUs), tensor processing units (TPUs), high-bandwidth memory (HBM) and high-capacity interconnects. The typical scientific supercomputer cluster fielded today is changing from one composed of larger volumes of lower-cost, identical, homogeneous nodes to lower volumes, denser sets of higher-value high-capability specialized nodes. High capability nodes composed of a greater number of components have higher integration complexity. Higher integration complexity has a consequent increase in operational costs. Costs of operations are borne by the user community, the system owner and by the HPC administrative staff. These costs may be measured in time or dollars and include: repair, replacement, immediacy and validation.

Unavailability is quantifiably consequential to consumers of these scientific calculations, the end-user scientist, the system owner and the funding entity. End users experience the impact as reduced workflow, greater turn-around time and greater contention for a reduced set of resources. System owners see less efficient, reduced utilization and greater scheduling contention. A variety of metrics are used by the differing communities (i.e. mean time to job interruption, mean time between failures, mean time to repair) but all describe an effective reduction in in service. Funding agencies may quantify this as greater variance in predictable costs (i.e. power, cooling.)

In this paper, we contribute to the HPC community by

- extending BatSim [1], a state of the art high performance computer system simulator, to allow it to checkpoint itself as well as improving the runtime performance of the built in EASY Backfill algorithm resulting in a **5X** to **40X** overall simulation speedup.
- conducting large-scale parameter studies (over **900,000**

This manuscript has been approved for unlimited release and has been assigned LA-UR-24-25237. This work has been authored by an employee of Triad National Security, LLC which operates Los Alamos National Laboratory under Contract No. 89233218CNA000001 with the U.S. Department of Energy/National Nuclear Security Administration. The publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of the manuscript, or allow others to do so, for United States Government purposes.

*Graduate students & Coastal Carolina University staff members.

†Corresponding authors.

individual simulations) to characterize the trade-off among reliability, repair time, and performance. We use three data sets: two separate years of real production workload HPC accounting data from a 1500-node computational cluster ("Grizzly") and statistically similar synthetic HPC accounting data.
- providing the modifications [2] and these data sets for use by the general HPC community [3].

The rest of the paper is organized as follows: Section II provides relevant background for recent developments in the field of parallel job scheduling as well as supercomputer reliability studies. Section III provides a detailed description of our initial experimental setup and preliminary results that show significant simulation speedup factors that enable the intended trade-off analysis central to this paper. Section IV details the central trade-off experimentation and results that show how the simulation framework can be used to enable decision-makers to make *more* informed decisions about the procurement, deployment and operation of large-scale cluster computers. Section V provides a discussion and overarching summary of the contributions of this paper, and Section VI discusses the conclusions, limitations of this work, and future directions.

II. Related Works

The initial motivation for this work centered on exploring the trade-off space between reliability and performance in large-scale cluster computing systems [4]. Specifically, due to our prior work in particle-induced transient soft errors in memory [5], an interesting project was initially identified [6] that explored the DRAM failures and protection schemes on the total cost of ownership of a data center that appeared to also take into account the performance of such systems as a function of reliability.

Consequently, an extensive review of existing open-source cluster simulation frameworks was conducted that included installation, execution, and evaluation against the needed criteria, including two main criteria *beyond* the simulation of the cluster and job scheduling and resource management functionality, specifically, the ability to model and control component failures as well as the concept of job checkpoint-restart. The investigation included Alea [7], Desmo-j [8], MaGate [9], SlurmSim [10], [11], and finally, BatSim [1], [12], [13]. Of these, BatSim covered many of the capabilities needed and was the closest match for a number of reasons, including that BatSim is well-documented, modular, and extensible. It also appears to have ongoing support and code updates as reported in the Git repository. Furthermore, it is built on top of SimGrid [14], a well-known cluster platform simulation framework, thus adding an additional level of configurability and fidelity.

III. Initial Experimentation and Improvements

Based on our prior experience in HPC cluster simulation [4], [15] we decided to make use of BatSim [1], [13] as the basis for our parameter studies. Once we began our specific sets of experiments (Section IV), we noticed that we were not able to conduct individual experiments in BatSim fast enough to obtain results in a timely manner, especially given the stochastic nature of many of our simulations that depend on obtaining converged statistics across thousands of trials in order to have confidence in the generality of our results.

TABLE I
Specifications of the Chicoma Supercomputer

Attribute	Description
System Type	HPE Cray EX system
Networking	HPE Slingshot
Total Nodes	688
Memory	More than 300TB
Node Configuration	560 nodes with dual AMD Epyc 7H12 CPUs 128 nodes with single AMD Epyc 7713 CPUs and quadruple Nvidia A100 GPUs
Computing Power	More than eight petaflops

In some cases, experiments took on the order of days to complete on a single node of the Chicoma supercomputer at LANL, with specifications as shown in Table I. As such, before we were able to conduct our parameter studies on a large scale, our first objective was to profile the BatSim simulator, determine where the bulk of the time was being spent, and determine if we could speed up the simulation at all.

This lead to two significant improvements in the BatSim code base:

1) Dramatic timing improvement of the EASY backfilling [16] scheduler used across our simulations.
2) Integration of checkpointing [17] directly into BatSim, allowing our simulations to continue running after restarting from a previous saved state.[1]

Checkpointing is particularly important for LANL cluster resources, where jobs may only run continuously for a defined amount of time, such as 16 hours in the case of the standard batch partition on the Chicoma supercomputer. These improvements are discussed in Sections III-A and III-B, respectively.

A. EASY backfilling

BatSim's counterpart program, Batsched [18] is a scheduler that provides a set of scheduling algorithms, including different variants of backfilling. After profiling many of these algorithms, we discovered that the particular implementation of the provided EASY backfilling was the source of a relatively large fraction of the overall simulation runtime. This prompted a deeper investigation into the algorithm's implementation, with a goal of improving its performance.

The scheduling system of the original EASY backfilling algorithm is centered around the concept of *Time Slice Fitting*. Time slice fitting is a scheduling strategy that uses time segments to manage the schedule. The slices are represented

[1]To clarify, in our prior work, we implemented *simulated* checkpointing into BatSim, e.g. BatSim simulates parallel jobs, and we previously integrated simulated job checkpointing *inside* the BatSim simulator; however, in this work, we modified BatSim, the application itself, to checkpoint itself.

as nested objects, which hold the current state of system resources, list of scheduled jobs, and other metadata. This strategy was designed around the intricacies of managing resources with job reservations in conservative backfilling where jobs behind the head of the queue are reserved time slots and allocated resources for the future. In contrast, traditional EASY backfilling does not require reservations, its only constraint is that backfilled jobs cannot prevent the job waiting at the head of the queue from running at the next possible future time. Note, their specific implementation of EASY backfill appears to have been derived for their more complex and capable conservative backfill algorithm. This derivation introduced a feature in their algorithm that we do not need while incurring a very large overhead to maintain the schedule data structures.

Algorithm 1 represents a high-level overview of how the time slices were used to decide if a job can be backfilled. The general flow is to search for the first time slice where there is enough nodes for the job to run. In the case the job cannot finish before the time slices ends, future time slices that make up the remaining wall time are evaluated. The primary constraint is the job must be able to retain its node allocation across all time slices. This strategy provides little flexibility in the implementation, resulting in tightly coupled code and traversals over large, nested data structures.

An important observation in the algorithm is that jobs are appended to time slices as they are evaluated. The time slice fitting logic is completely agnostic to whether a job is a priority job or not. In the cases where a non-priority job cannot retain its node allocation, the job and its node allocation are later removed from each slice where it was previously determined to fit. The inefficiencies in memory management and decision processes are reflective of the slow simulation behavior observed during experimentation. These overhead costs increase with the problem size and queue complexity, resulting in nondeterministic delays in simulation progress.

The problems presented by the former implementation inspired a new backfilling strategy and other modifications made. The time slices data structures were traded for simple job objects holding the minimal necessary metadata. The updated approach consists of two cases for scheduling decisions, one for priority jobs and another for backfill jobs. The following algorithms demonstrate how priority jobs can be leveraged to make decisions for if a job should be backfilled or not. Algorithm 2 details how priority jobs are handled. The key logic is when a priority cannot run initially, we predict the next possible future time it can run.

The approach is to sort running jobs by *estimated end time* and search for the first such job where once completed, enough nodes are released for the priority job to run. After, the priority jobs *predicted start time* and *leftover nodes* are saved determine which jobs may be backfilled as shown in Algorithm 3. Non-priority jobs whose *estimated end time* is before the current priority jobs *predicted start time* are immediately scheduled given that there is enough available nodes in the system. Otherwise, we evaluate if the jobs requested nodes is less than the minimum of the systems *currently free nodes* and the priority jobs *leftover nodes*. This constraint ensures backfilled jobs do not prevent priority jobs from running, eliminating the risk of starvation for resource intensive jobs.

The final modifications entailed replacement of the original

Algorithm 1 Easy Backfill (Original) - Time slice Fitting

Require: Schedule $TS[0 \dots Z]$, New Job job, Simulator Date T_{date}, Free Nodes N_{free}, Max Nodes N_{max}
Ensure: $Q \neq \emptyset,\ TS[0] \leftarrow$ **new** $timeSliceObject,\ T_{now} \leftarrow T_{date}$
procedure ADDJOBTOSLICE(TS, j, job, N_{free})
  $N_{free} \leftarrow N_{free} - job.ReqNodes$
  $job.NodeIds \leftarrow$ AllocateNodeIds($job.ReqNodes$)
  $TS[j] \leftarrow TS[j] + job$
  if $j == (Z - 1)$ **then**
    $endDate \leftarrow TS[j].Begin + job.WallTime$
    Split Last Time Slice $TS[j]$ **by** $endDate$
  end if
end procedure
function CHECKTIMESLICES(job, N_{free}, T_{now})
  $canRun \leftarrow FALSE$
  for $i = 0$ **to** $Z - 1$ **do**
    if $TS[i].FreeNodes \geq job.ReqNodes$ **then**
      if $TS[i].Duration \geq job.WallTime$ **then**
        AddJobToSlice(TS, i, job, N_{free})
        $canRun \leftarrow (i == 0)$
        return $canRun$
      else
        for $k = i$ **to** $Z - 1$ **do**
          $timeSum \leftarrow (timeSum + TS[k].Duration)$
          if $job.ReqNodes \geq TS[k].FreeNodes$ **then**
            return $canRun$
          else if $timeSum \geq job.WallTime$ **then**
            AddJobToSlice(TS, k, job, N_{free})
            return $canRun$
          end if
        end for
      end if
    end if
  end for
  return $canRun$
end function

Algorithm 2 Easy Backfill (Modified) - Priority Jobs

Require: Schedule $S[0 \dots Z]$, Priority Job pj, Free Nodes N_{free}, Max Nodes N_{max}
Ensure: $Q \neq \emptyset,\ pj.ReqNodes \leq N_{Max}$
function CHECKPRIORITYJOB(pj, N_{free})
  if $pj.ReqNodes \leq N_{free}$ **then**
    $canRun \leftarrow TRUE$
  else
    $canRun \leftarrow FALSE$
    $N_{future} \leftarrow N_{free}$
    for $i = 0$ **to** $Z - 1$ **do**
      N_{future} += $S[i].ReqNodes$
      if $S[i].ReqNodes \geq N_{future}$ **then**
        $pj.PredictedStart \leftarrow S[i].EstimatedEnd$
        $pj.LeftoverNodes \leftarrow (N_{free} - pj.ReqNodes)$
        return $canRun$
      end if
    end for
  end if
end function

Algorithm 3 Easy Backfill (Modified) - Backfill Jobs

Require: Backfill Job bj, Priority Job pj, Simulator Date T_{date}, Free Nodes N_{free}, Max Nodes N_{max}
Ensure: $Q \neq \emptyset,\ bj.ReqNodes \leq N_{Max},\ T_{now} \leftarrow T_{date}$

```
function CHECKBACKFILLJOB(bj, N_free)
    bj.EstimatedFinish ← (T_now + bj.WallTime)
    if bj.EstimatedEnd ≤ pj.PredictedStart then
        N_future ← N_free
    else
        N_future ← MIN[N_free, pj.ExtraNodes]
    end if
    canRun ← (bj.ReqNodes ≤ N_Future)
    return canRun
end function
```

reservation queuing system, composed of a linked list of newly allocated job objects, each sequentially sorted by submission time. The improved queue now employs a maximum heap with flexibility to sort different attributes based on specific scenarios. For instance, when simulating node failures, jobs can be sorted by original submission time. This adjustment prioritizes resubmitted jobs, enhancing queue fairness and mitigating the risk of starvation for repeatedly killed jobs.

While the performance aspect of the new EASY backfilling strategy proved to beneficial, it was vitally important to verify the impacts these modifications would have on the simulated results. In other words, would the updated algorithm make the same decisions as the original and would it perform as well from a parallel job scheduling point of view? The ideal outcome would be that the modified version is no different from the original, but this was not entirely the case. After several tests, one deviation was found which impacted the decisions when a job could be backfilled in certain scenarios. This was found to be a by-product of the original algorithm's time slice fitting logic, in particular the *pre-allocation* of node IDs for waiting priority jobs.

Fig. 3 represents the normalized differences in simulated waiting times between algorithms for the Grizzly 2022 workload shown in Fig. 6. One initial observation reveals an outlier in bin $[512, 1024)$ for *Backfilled Jobs*, where the new version's mean wait times are approximately $\sim 58\%$ worse. This anomaly can be attributed to the fact that only $\sim 1.06\%$ of all backfilled jobs in the new version fell within that bin, obscuring the weighted mean. Observing *All Jobs*, we see that bins with the most backfilled jobs, $[1, 2)$ and $[32, 64)$, performed better with the old algorithm by $\sim 12{-}17\%$. The remaining bins show the new algorithm is better or very close to *Zero Difference*. The *Overall Difference* indicates that the modified algorithm's simulated waiting times are only $\sim 4\%$ worse on average.

Fig. 4 depicts the normalized differences in simulated waiting times for the Grizzly 2018 workload shown in Fig. 5. Examining *All Jobs*, we see that the older algorithm had better simulated waiting times on average by $\sim 22.5\%$ for one-node jobs. The remaining job bins alternate, revealing marginal differences of less than $\sim 10\%$, where the new algorithm is better or worse. This results in an *Overall Difference* of the

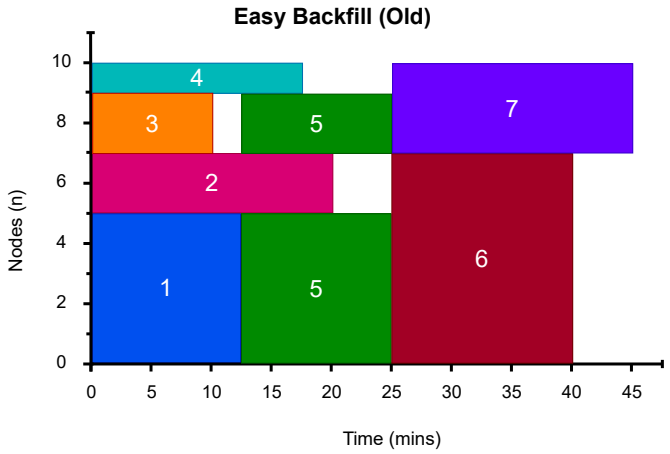

Fig. 1. Gantt chart demonstrating how the original EASY backfilling algorithm chooses when a job can be backfilled. It assumes that all seven jobs arrived sequentially at time zero, and the resulting schedule is generated using the time-slicing procedure outlined in Algorithm 1. Note: since the original algorithm pre-allocated node resources Job 6 prevents Job 7 from being backfilled alongside Job 5, where as in the new, *much faster*, implementation, Job 7 can be backfilled with no impact to Job 6's original start time as shown in Figure 2.

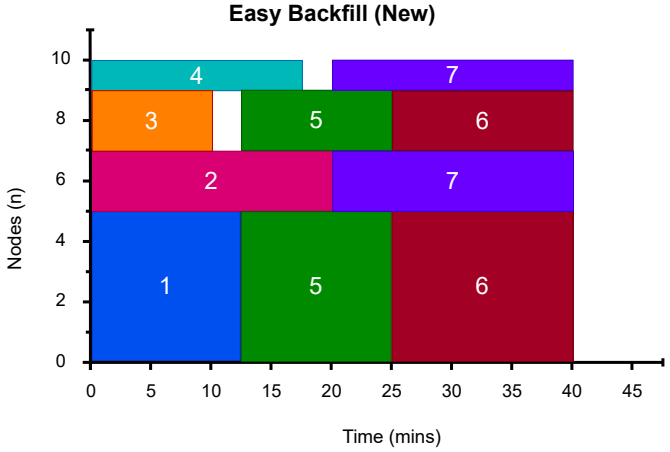

Fig. 2. Gantt chart illuminating how the modified EASY backfilling scheduler's decisions differ in contrast to originals from Fig. 1. The schedule is generated based the backfilling conditions shown in Algorithm 3, which depends on priority-based logic in Algorithm 2. Note: In contrast to the behavior shown in Figure 1, here, Job 7 backfills ahead of Job 6 with no detrimental impact to its original start time.

new algorithm being under $\sim 2.5\%$ worse than the original. Considering the observations made in both Figs. 3 and 4, the differences in the new algorithm show negligible impact on the simulated queue waiting time.

In short, the two algorithms do not make identical scheduling decisions; however, the original EASY backfill implementation in the BatSim source code was more constrained than it needed to be as far as pre-allocating *specific* nodes in the future schedule, versus simply ensuring that sufficient nodes would be available at that given time to satisfy the priority job request. By relaxing this constraint, we still allow jobs to be backfilled which also prevents starvation, but with

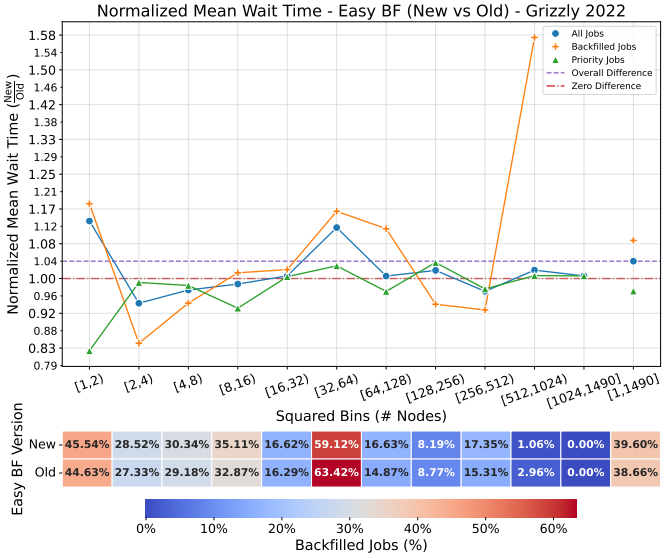

Fig. 3. Normalized mean wait times for 2022 Grizzly jobs, shown in Fig. 6, revealing the impacts of modification made to EASY backfilling to queued jobs with varying node allocations. Data points that fall below the *Zero Difference* line represent where the modified algorithm version is better. Below the plot is the percentage of backfilled jobs per bin, for insight on the density of affected jobs. Note, despite cases where the new algorithm performed worse, the *Overall Difference* is very close to *Zero Difference*.

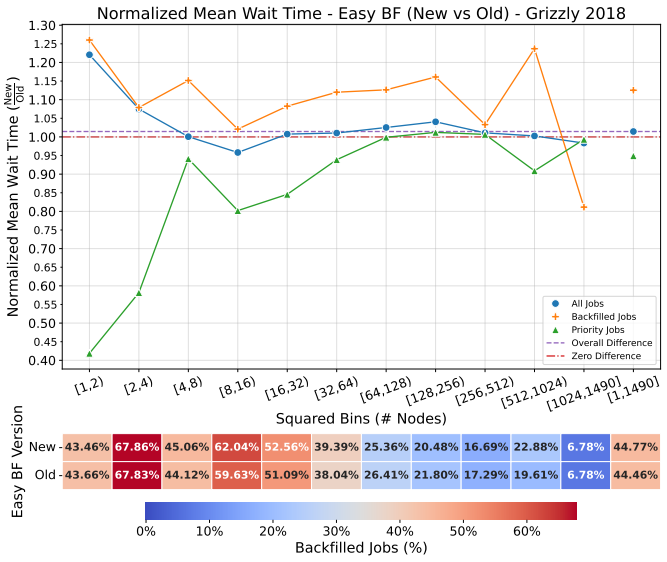

Fig. 4. Additional insight on the normalized mean wait times shown in Fig. 3, examining the impacts to another workload from Grizzly in 2018, shown in Fig. 5. Below the plot is the percentage of backfilled jobs per bin, showing on the density of affected jobs. Similar to Fig. 3 the *Overall Difference* is also extremely close to *Zero Difference*. The main takeaway here is that the overall difference between the mean job waiting time for original and new algorithms is negligible, while the speed of the new algorithm is far superior.

the new implementation, the execution times of our large-scale simulations are much shorter, allowing us to conduct simulations much faster and with increased accuracy through a larger number of Monte Carlo trials.

Table II highlights the performance results of each algorithm

TABLE II
PERFORMANCE ANALYSIS OF EASY BACKFILLING

Mean Elapsed Times *(seconds)* and Speedup			
Workload	**Algorithm**	**Overall Time *(s)***	**Decision Time *(s)***
Grizzly 2018	EBF Old	32,917.05	27,577.03
	EBF New	6,095.06	11.80
	Speedup	5.40	2,336.30
Grizzly 2022	EBF Old	476,276.35	467,990.22
	EBF New	12,047.28	41.51
	Speedup	39.53	11,273.23

* Mean results approximated from 100 serial node-exclusive trials

version for both Grizzly workloads. The mean elapsed time (μ_{Time}), was calculated for both the overall time spent and the time dedicated solely to decision-making as: $\mu_{Time} = \frac{\sum T_{Algorithm}}{N_{Runs}}$. The mean speedup ($\mu_{Speedup}$), was computed with the original captured times before approximation as: $\mu_{Speedup} = \frac{\mu_{TimeOld}}{\mu_{TimeNew}}$. Examining the mean elapsed times for old algorithm, it is evident the time spent making decisions accounted for a substantial portion of the overall time in both workloads. Conversely, the mean decision times in the new algorithm were less than one minute in both cases. This contributed to overall mean speedup factor of **5.4X** for the Grizzly 2018 workload and **39.53X** for the Grizzly 2022 workload. An interesting observation is the mean speedup factor for the more intensive Grizzly 2022 workload was **7X** higher than Grizzly 2018. As the queue depth increases, we observe larger improvements and thus higher associated speedups over the original algorithm. These findings suggest the new algorithm is effective in terms of scalability for increasing problem sizes, workloads, cluster scope and complexity.

B. Simulation checkpointing

In large-scale parallel systems, application checkpointing is the primary means of guarding long-running jobs against faults that interrupt the execution of these programs. In these cases, the entire state of an application is written to non-volatile storage so that the application can be restarted from the last checkpoint, rather than losing all the work it has completed since it began execution. While checkpointing is largely used to restart an application that has been interrupted by such a failure, it can also be used to allow long-running simulations to run on batch systems that have a limit to how long a job can run. Most organizations have such "wall limits" that prevent jobs from running "indefinitely". Many of the applications of interest to LANL run for weeks before they are complete, but are only allowed to run in time-limited intervals on the supercomputer systems. For example, this limit might be 24 hours, and to get a 3-week job completed, the job will need to be checkpointed and restarted many times over possibly many weeks (depending on system load) to complete the job.

In this work, we have added the capability for BatSim to checkpoint itself and to restart the simulation, thus allowing us to make use of LANL compute resources for some of our long-running simulations that would not otherwise complete

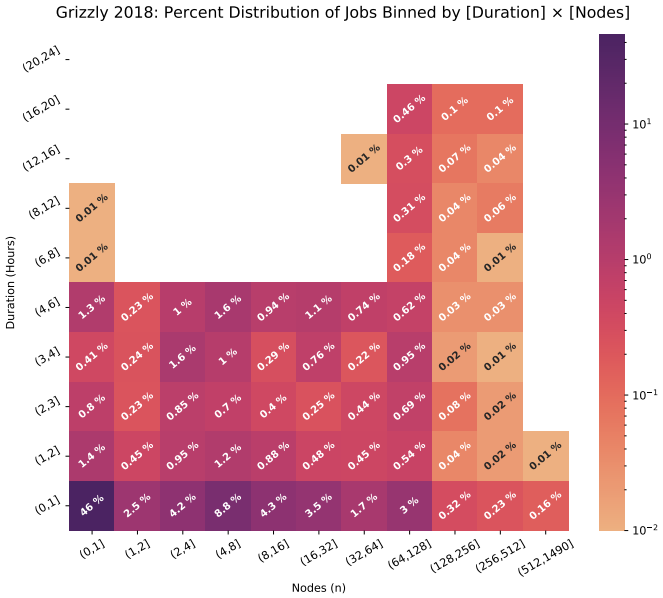

Fig. 5. Percentage distribution of jobs ran on Grizzly in 2018, binned by duration times and requested nodes. This set consists of approximately 130K jobs ran over the span of 11 months in 2018 [19].

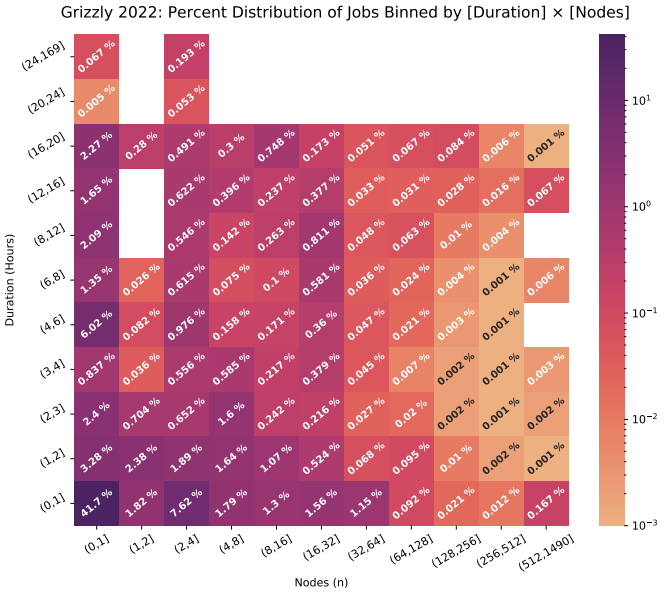

Fig. 6. Percentage distribution of jobs ran on Grizzly in 2022, binned by duration times and requested nodes. This set is composed of approximately 190K jobs from 7 months of job data in 2022.

during the time LANL allows on the systems to which we have access. Naturally, this has the added benefit of providing adding a degree of resilience to BatSim, in the event that BatSim fails while running. BatSim has been configured to allow it to checkpoint itself at regular intervals, or in response to a signal received from SLURM, a common resource manager and job scheduler used on modern cluster computing systems. Our experimentation has shown that these additions have had zero negative impact to BatSim's runtime performance when checkpointing is not used, and in the cases where it is, the impact is very small, at least for the simulations that we ran with an overhead of less than 2%.

IV. Trade-off Experimentation and Analysis

After incorporating the improvements discussed in Sections III-A and III-B into BatSim, we were then able to conduct a large-scale parametric study on the LANL Chicoma supercomputer to help us better understand the trade-offs between reliability, repair time and performance.

In these studies, we made use of 2 years of real workload data from the LANL 1500-node supercomputer, Grizzly. Figs. 5 and 6 show the distribution of the jobs in these workloads as a function of their runtimes and widths (i.e. number of nodes they require) for the 2018 and 2022 calendar years, respectively. We see that Grizzly is mostly uses as a capacity machine, with many of the submitted jobs (around 45%-50%) making use of only a single node in the cluster, with the remaining 50% being for big parallel jobs.

In these experiments the baseline Grizzly system was assumed to have a system mean time between failure (SMTBF) that was proportional to an equivalent of 24 hours on a 20,000 node system. This was chosen since LANL's Trinity supercomputer [20] was around 20,000 nodes and had this same average publicly failure rate. This baseline failure rate is referred to as "1X" in our results.

The failure rates were increased from 1X to 32X, in powers of two, (i.e. 6 different failure rates) representing decreasing levels of reliability with 32X indicating a system that that had a 32 times higher failure rate, for example. Additionally, in the event of failure, the time to repair a node was varied from 1 minute (simulating a transient error with a quick reboot) all the way to 20 days (i.e. 23 different repair times), thus simulating a much more significant problem to repair.

These repair times are derived from internal measurements of node failures. For example, the predominant node failure mode is detected by an end-of-job node health check. [2] These are repairable by an automated process to reboot and return the node to service, yielding an effective repair time measured in minutes. Repairs that require human inspection, analysis or physical repair or replacement constitute the long tail of repair modes.

For the 2022 data, Overall 276 distinct parameter combinations were completed. Due to the random nature of failures, in order to provide a more statistically relevant set of results, each of the 276 combinations were executed 1000 times resulting in **276K** simulations. For the 2018 data, around 624 distinct parameter combinations were performed 1000 times each for a grand total of around **900K** Monte Carlo simulated year-long supercomputer experiments across *both* datasets. These sim-

[2]These transient failures are the consequence of many faults. Such latent faults exist at the hardware, kernel software and system service layers, underlying the application-visible HPC substrate. Examples include fragmented kernel memory maps, transient network driver errors, unreleased buffered IO memory (parallel filesystem) space and complex dependencies among underlying system and Kubernetes services.

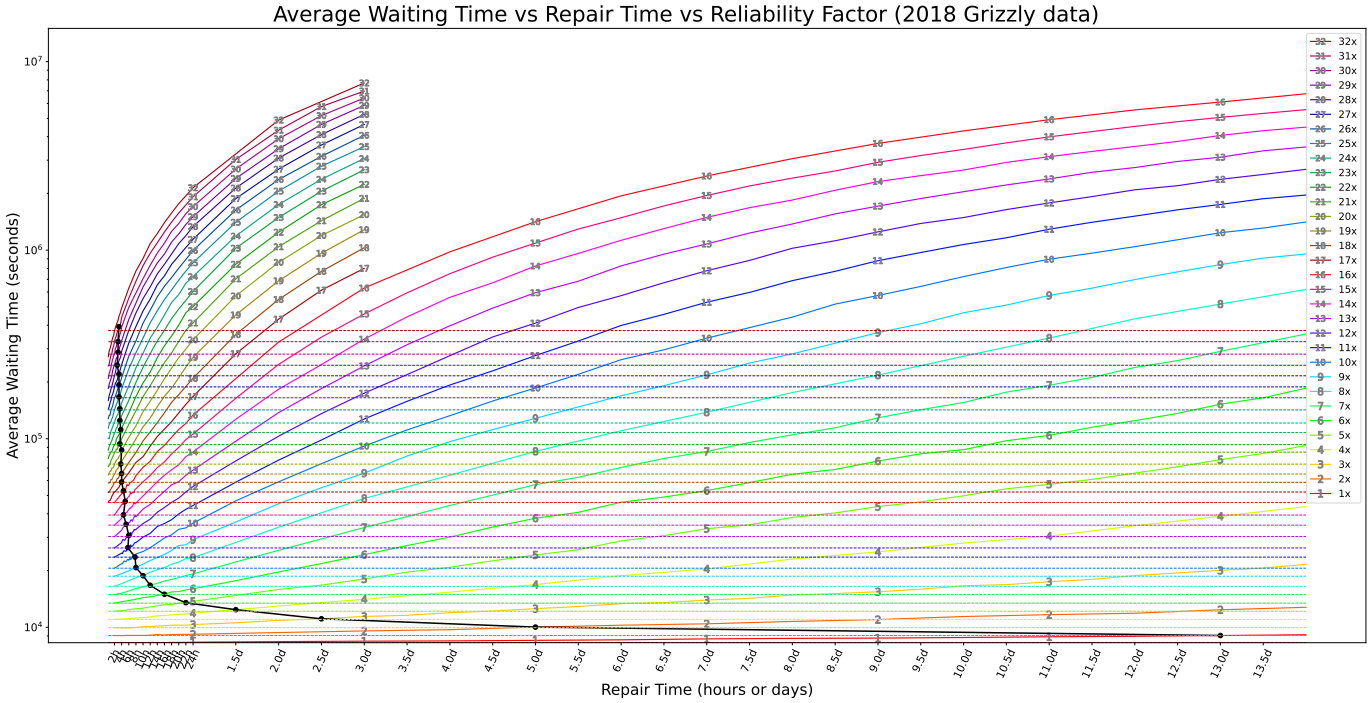

Fig. 7. This figure illustrates the average job waiting time on the Grizzly supercomputer across all jobs in 2018 as a function of both the normalized node reliability as well as the average node repair time when failures do occur. We can see that as node reliability decreases from 1X (the base-line system reliability that is proportional to a 24 hour MTBF on a 20K node system, like LANL's Trinity supercomputer) to 32X (i.e. 32 times less reliable than the base-line system) the allowable node repair time decreases that will allow the given system to maintain the same performance from an end-user perspective. For example, the highest reliability system (1X) with a 13-day node repair time produces the same performance as a system that is 2X less reliable with a 1-minute node repair time. Furthermore, for a given quality of service (queue waiting time target i.e. horizontal line), this simulation campaign can be used to determine the required repair times for a system as the reliability factor changes from more reliable to less reliable. As such, it enables a clear trade-off analysis between repair time and reliability.

ulations were executed on LANL's Chicoma supercomputer (Table I).

Fig. 8 shows the average waiting times seen across all submitted jobs as a function of system reliability and node repair time for the 2018 workload. The waiting times are normalized with respect to the 1-minute repair time waiting time values for each node reliability curve. We see the more reliable systems can afford to have much longer repair times compared to less reliable systems. This is not particularly a surprising result; however, the contribution here is that our simulation framework can be used by procurement teams, deployment teams as well as operations personnel to make decisions about how specify, produce and operate such systems, given hardware quality of service contractual agreements as well as for input parameters for models for the total cost of ownership under varying conditions.

Fig. 7 provides a quick way to identify performance-based break-even points. For example we can see that a system with a 4X relative reliability and an average node repair time of 5 days has the same performance (average job waiting time) as a much less reliable system with an 8X relative reliability, but with a much shorter required node repair time of 1 minute. This experimentally obtained break-even analysis can be used to make informed data-driven decisions about the procurement and operation of such large scale systems with targeted relative quality of service. In fact, in general, any arbitrary cluster configuration, input workload, fault model, checkpoint-restart model, and node reliability characteristic can be modeled against required end-user qualities of service to enable to determine where these break-even points are located.

V. Discussion

We have extended and improved the existing BatSim cluster computer system simulation and job scheduling framework to allow users to much more quickly evaluate the trade-off space that exists among such parameters as performance (queue waiting time), node repair time, and intrinsic node reliability. With the major improvement in the EASY backfilling scheduler implementation, we can now run tens of thousands of Monte Carlo simulations in a fraction of the time it previously took. In fact, based on the workloads we have studied in this paper, the overall simulation time has seen a speedup of between **5X** to **40X**. As the simulated systems become more loaded, the longer the queue depths are, the larger the speedup factor is over the prior EASY Backfill implementation. This will be particularly important for applying BatSim on larger parallel systems with longer queues, since the runtime of the old EASY backfill implementation is extremely sensitive to the depth of the waiting job queue.

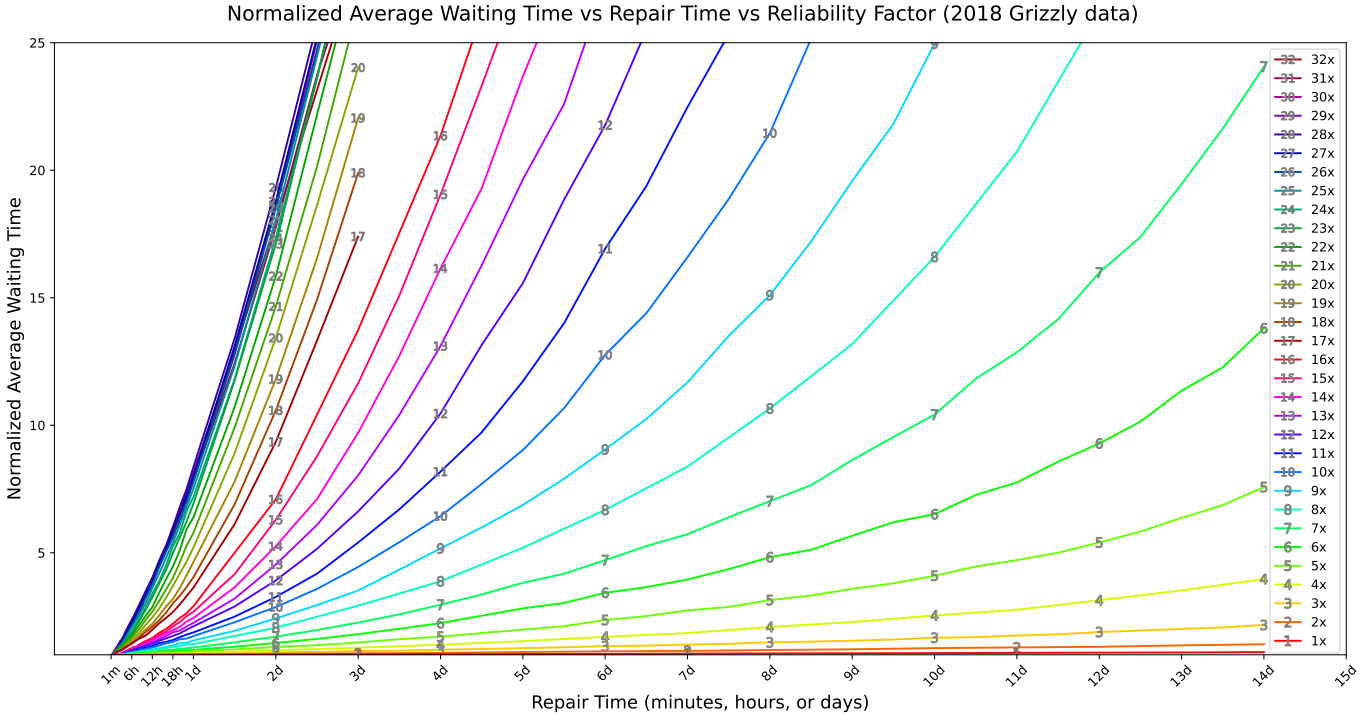

Fig. 8. This figure illustrates the *normalized* average job waiting time on the Grizzly supercomputer across all jobs in 2018 as a function of both the normalized node reliability and the average node repair time when failures do occur. (Note: *Non-normalized* waiting times are shown in Fig. 7). In these figures, 1X represents a base-line system reliability that is proportional to a 24 hour MTBF on a 20K node system, like LANL's Trinity supercomputer which is used a reference point in our reliability work at LANL. As such, 32X means as a system that is 32 times less reliable than the base-line system. Our simulation framework allows system procurement teams to weigh the relative importance of buying a more reliable *and expensive* system against the time associated with the number repairing nodes when they fail. This can then be used to drive a total cost of ownership analysis that factors in both initial capital investments along with operational costs that include the "allowable" time and money needed to repair the system, while concurrently providing a prescribed average quality of service (average waiting time in this case).

As such, this improvement has allowed us to explore a *much larger and more complete parameter space*, as evidenced in Section IV. In the case of the Grizzly supercomputer, there is a strong trend that as the node reliability is improved, that the amount of time that one is allowed to have to repair a given node dramatically increases. With our tool, different types of failures, from transient faults with relatively quick repair times such as a simple reboot, to more extensive faults that potentially require much more time to fix can be explored with our updates to the BatSim framework. This type of exploration allows users of to make more informed data-driven decisions about how to target bids, set quality of service expectations and to drive *further* financial total cost of ownership studies to more fully understand the performance trade-off for different types of equipment and recurring operational expenses.

The specific questions addressed in this work constitute a first pass at quantifiably exploring the data that production HPC sites have about their own problem space. For example, the specific plot in fig. 8 implies that job waiting time is the desired and optimizable value metric, at the expense of the quantity or unreliability of the computational nodes. Using such analysis one may quantifiably compare this to an alternate control value, such as an increase in the number of available, but unreliable, nodes. An alternate analysis may be driven by the time to repair, governed by the implied operating expense of having humans in the loop effecting the repair. Another example would be from the perspective of consumers of HPC services, their service level requirements could include the minimum acceptable waiting time for jobs to launch. This may be viewed as a horizontal slice across this diagram. An alternate perspective might be that of the provider of on-site HPC repair. This is a vertical slice through these data, constituting a limit on the localized operating expense. Repair exceeding this time limit could be defined by an external contractual service level requirement.

This work has been motivated by production-oriented questions arising from LANL's considerable investment in its own and the US DoE HPC ecosystem. Quantifiably analyzing problems such as these begins the process of engineering operating processes and procedures.

This and prior work has guided production HPC job scheduler configuration and optimized human system administration processes, increasing reliability. Questions that follow from this work include:

- If a cluster were to be expanded, doubling its node count, how should those nodes be included? Should an alternate cluster be built for A/B redundancy or should a single cluster be doubled in size? What are the impacts on the

workflow?

- The work shown was for a 1500 node cluster. Given a sweep of the sizes of affordable clusters, smaller then Grizzly, can we characterize the distinctly different populations and ecosystems of clusters? (ex. start-up company vs. academic clusters vs. national initiatives) How might these data be fed into external models? (acquisition, power and cooling operating budget)
- Given the availability of non-local data centers, how may we quantifiably model a multivariate (I/O transport and computational resources) constrained workflow?
- Does this analysis apply to a two-level computational resource, such as a node with CPU and GPUs?

VI. Conclusions and Future Work

The work presented in this paper is both experimental and applied. We have extended and improved the existing BatSim cluster computer system simulation and job scheduling framework to allow users to much more quickly evaluate the trade-off space that exists among such parameters as performance, node repair time, and node reliability. With the improvement in the EASY backfilling scheduler, we can now run tens of thousands of Monte Carlo simulations in a fraction of the time it previously took. This improvement has allowed us to explore a larger and more complete parameter space, as evidenced in Section IV. This type of exploration allows users of the tool to make more informed data-driven decisions about how to target bids, set quality of service expectations and to drive *further* financial total cost of ownership studies to more fully understand the performance trade-off for different types of equipment and recurring operational expenses. We have also published the production job accounting data sets. Our goal, in doing so, is to grow the referenceable body of HPC scientific and research job and workflow data. We are using this in our own follow-on studies and as training data. We hope that others will find it useful as well.

One of the current limitations of our study includes the need to provide an analysis across a more varied set of cluster computer workloads at LANL. LANL and other US DoE sites have a range of supercomputers. Our future work will include applying BatSim to workloads from several of these computers; however, most of these workloads are not publicly available and will be used internally by the US DoE.

References

[1] P.-F. Dutot, M. Mercier, M. Poquet, and O. Richard, "Batsim: a Realistic Language-Independent Resources and Jobs Management Systems Simulator," in *20th Workshop on Job Scheduling Strategies for Parallel Processing*, Chicago, United States, May 2016. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01333471

[2] Los Alamos National Laboratory. (2024) Simulator repo – batsim extensions. https://github.com/hpc/simulator.git.

[3] ——. (2024) Los Alamos National Laboratory Grizzly Log Data. https://github.com/hpc/LANL-grizzly-data.

[4] C. Walker, B. Slade, G. Bailey, N. Przybylski, N. DeBardeleben, and W. M. Jones, "Exploring the tradeoff between reliability and performance in hpc systems," in *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, 2021, pp. 1–7.

[5] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, "Memory errors in modern systems: The good, the bad, and the ugly," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '15, Istanbul, Turkey, March 14-18, 2015*, Ö. Özturk, K. Ebcioglu, and S. Dwarkadas, Eds. ACM, 2015, pp. 297–310. [Online]. Available: https://doi.org/10.1145/2694344.2694348

[6] P. Nikolaou, Y. Sazeides, L. Ndreu, and M. Kleanthous, "Modeling the implications of dram failures and protection techniques on datacenter tco," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015, pp. 572–584.

[7] D. Klusáček, v. Tóth, and G. Podolníková, "Complex job scheduling simulations with alea 4," in *Proceedings of the 9th EAI International Conference on Simulation Tools and Techniques*, ser. SIMUTOOLS'16. Brussels, BEL: ICST (Institute for Computer Sciences, Social-Informatics and Telecom. Engr.), 2016, p. 124–129.

[8] J. Göbel, P. Joschko, A. Koors, and B. Page, "The discrete event simulation framework DESMO-J: review, comparison to other frameworks and latest development," in *Proceedings of the 27th European Conference on Modelling and Simulation, ECMS 2013, Ålesund, Norway, May 27-30, 2013*, W. Rekdalsbakken, R. T. Bye, and H. Zhang, Eds. European Council for Modeling and Simulation, 2013, pp. 100–109. [Online]. Available: https://doi.org/10.7148/2013-0100

[9] Y. Huang, A. Brocco, M. Courant, B. Hirsbrunner, and P. Kuonen, "Magate simulator: A simulation environment for a decentralized grid scheduler," in *Advanced Parallel Processing Technologies*, Y. Dou, R. Gruber, and J. M. Joller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 273–287.

[10] N. A. Simakov, M. D. Innus, M. D. Jones, R. L. DeLeon, J. P. White, S. M. Gallo, A. K. Patra, and T. R. Furlani, "A slurm simulator: Implementation and parametric analysis," in *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, S. Jarvis, S. Wright, and S. Hammond, Eds. Cham: Springer International Publishing, 2018, pp. 197–217.

[11] N. A. Simakov, R. L. DeLeon, M. D. Innus, M. D. Jones, J. P. White, S. M. Gallo, A. K. Patra, and T. R. Furlani, "Slurm simulator: Improving slurm scheduler performance on large hpc systems by utilization of multiple controllers and node sharing," in *Proceedings of the Practice and Experience on Advanced Research Computing*, ser. PEARC '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3219104.3219111

[12] M. Poquet, "Simulation approach for resource management," Theses, Université Grenoble Alpes, Dec. 2017.

[13] Inria, "BatSim," Institut National de Recherche en Sciences et Technologies du Numérique - https://batsim.readthedocs.io/, accessed: 2024-05-14.

[14] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, Jun. 2014. [Online]. Available: http://hal.inria.fr/hal-01017319

[15] W. M. Jones, C. S. Walker, V. E. Hafener, W. D. Graham, N. A. DeBardeleben, and S. T. Senator, "Incorporating staggered planned maintenance reservations to improve performance in computational clusters," in *2023 IEEE International Conference on Cluster Computing Workshops (CLUSTER Workshops)*, 2023, pp. 32–36.

[16] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan, "Characterization of backfilling strategies for parallel job scheduling," in *Proceedings. International Conference on Parallel Processing Workshop*, 2002, pp. 514–519.

[17] 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, 2006. [Online]. Available: https://doi.org/10.1016/j.future.2004.11.016

[18] Inria, "Batsched," Institut National de Recherche en Sciences et Technologies du Numérique – https://gitlab.inria.fr/batsim/batsched, accessed: 2024-05-14.

[19] Los Alamos National Laboratory, "Grizzly 2018 log data – tinyurl links to lanl ftp server," https://tinyurl.com/LANL-grizzly-data, 2018, accessed: 2024-05-14.

[20] J. Lujan, M. Vigil, G. Kenyon, K. Sanbonmatsu, and B. Albright, "Trinity supercomputer now fully operational," *U.S. Department of Energy Office of Scientific and Technical Information*, 11 2017.