Exploring the impact of node failures on the resource allocation for parallel jobs

Ioannis Vardas$^{1[0000-0001-5461-556X]}$, Manolis Ploumidis$^{1[0000-0003-2173-062X]}$, and Manolis Marazakis$^{1[0000-0002-4768-3289]}$

Institute of Computer Science (ICS), Foundation for Research and Technology – Hellas (FORTH), Greece 100 N. Plastira Av., Vassilika Vouton, Heraklion, GR-70013, Greece {vardas,ploumid,maraz}@ics.forth.gr

Abstract. Increasing the size and complexity of modern HPC systems also increases the probability of various types of failures. Failures may disrupt application execution and waste valuable system resources due to failed executions. In this work, we explore the effect of node failures on the completion times of MPI parallel jobs. We introduce a simulation environment that generates synthetic traces of node failures, assuming that the times between failures for each node are independently distributed, following the same distribution but with different parameters. To highlight the importance of failure-awareness for resource allocation, we compare two failure-oblivious resource allocation approaches with one that considers node failure probabilities before assigning a partition to a job: a heuristic that randomly selects the partition for a job, and Slurm's linear resource allocation policy. We present results for a case study that assumes a 4D-torus topology and a Weibull distribution for each node's time between failures, and considers several different traces of node failures, capturing different failure patterns. For the synthetic traces explored, the benefit is more prominent for longer jobs, up to 82% depending on the trace, when compared with Slurm and a failure-oblivious heuristic. For shorter jobs, benefits are noticeable for systems with more frequent failures.

Keywords: Impact of node failures on MPI parallel jobs · Fault-aware resource allocation · Synthetic node failure trace generation.

1 Introduction

HPC systems grow in size to meet the increased demand for both capability and capacity. At the same time, heterogeneity and complexity also increase to keep pace with application demand for performance. Several studies have outlined that the higher scale and complexity of HPC systems comes at the cost of more frequent failures [16, 23, 21, 3]. Furthermore, larger scale and more complex systems will introduce more complex software stacks to exploit their resources, with more frequent software-related errors [23, 6]. To further motivate the importance of fault-tolerance, authors in [23, 6, 16] argue that reliability, along with resource management and energy efficiency, will be among the main obstacles towards robust exascale. Error resilience has been recognized to be one of the major technical research priorities for the next years, in the European Technology Platform for HPC (ETP4HPC) strategic research agenda [1].

By combining failure logs and traces with workload logs, several studies have outlined the impact of various system failures on system resource utilization. Authors in [7] report that in a large-scale HPC system, 20% or more of the computing resources are wasted due to failures and recovery. For one of Google's multipurpose clusters, it was found that a large fraction of time is spent for jobs that do not complete successfully [6]. The authors in [19] show that system related errors cause an application to fail once every 15 minutes. What is more, failed applications, although few in number, account for approximately 9% of total production hours. Authors in [21] examine node failure rate in the dataset collected during 1995–2005 at LANL. The number of failures per year per system can be as high as 1100, implying that an application requiring the entire cluster is expected to fail more than two times per day.

Therefore, node failures in a HPC system need to be considered both from the point of view of job completion times (a main concern for application owners) and from the point of view of potential resource wastage (one of the main concerns of system owners and operators). To mitigate the impact of failures, different approaches have been proposed including checkpointing [14, 27, 25], scheduling methods [5, 13] and methods for resource allocation and resource management [30, 10, 18, 11]. For evaluating the effectiveness of these approaches, failures traces and logs acquired from a real HPC system or cluster have been used. However, traces constitute merely a *snapshot* of a real system, corresponding to a specific size and period of operation. Moreover, several studies have shown that failures are affected by the workload [28, 22].

It is therefore important to be able to explore the efficiency of fault tolerance methods under different failure conditions in a controllable and configurable manner. In this work, we present a simulation environment based on a synthetic trace generator for node failures. Further to the evaluation of fault tolerance methods, this work is a tool for system operators to assess the cost of failures, expressed in node-hours lost, either due to failure-oblivious resource allocation, or the overhead of fault-tolerance methods. We assume that for each node time-to-failure (TTF) is independently distributed, with all nodes following the same distribution, but with different parameters for each node.

To highlight the importance of failure aware resource allocation, we compare three different approaches with two of them being failure oblivious. The first one is based on a simple heuristic that searches for different contiguous and rectangular partitions of a torus topology. Then, based on the findings of [12], we estimate for each such partition the probability of failing during the upcoming job. This approach is compared against Slurm's [29] linear resource allocation, and to a heuristic that selects a partition for each job in a random manner. Both of these comparison baselines are oblivious to the probability of node failures. Finally, we illustrate the potential benefits of the failure-aware resource allocation approach with a set of simulation results for a case study concerning a 4D-torus topology with 4096 nodes. The mean time between failures for each node is independently distributed, and follows the Weibull distribution [12, 11]. The scale and shape parameters of the corresponding distribution for each node are determined by two separate Gaussian distributions. Results derived with our simulation environment suggest that the benefit achieved by a failure-aware resource allocation approach depends on the system failure pattern. Specifically, the benefit is more prominent for larger jobs (job duration $\geq 24h$) in systems with less frequent failures. This benefit is up to 82%, depending on the simulated trace. For shorter jobs, the benefit becomes notable only on systems with more frequent failures.

2 Simulation environment

In this section we describe our simulation environment and its main components. *Fail-stop* errors cause the execution of an application to terminate due to a hardware or software fault, whereas *silent errors* can impact the result of an application without causing termination. In this paper, we focus on *Fail-stop* type of failures. Moreover, with the term *failure* we refer hardware- or software-related deviation from nominal operating behavior. We further assume that a node restart is enough to fix transient failures, and that nodes fail independently of each other. Our simulation environment is not meant to offer the same level of simulation accuracy like Simgrid [4] or xSim [9]. In the current version we do not rely on any networking or processor model for deriving an accurate job duration; instead, we assume that job durations are known and explore three resource allocation approaches under different failure patterns. Job durations can be specified either through a distribution a post-processed trace.

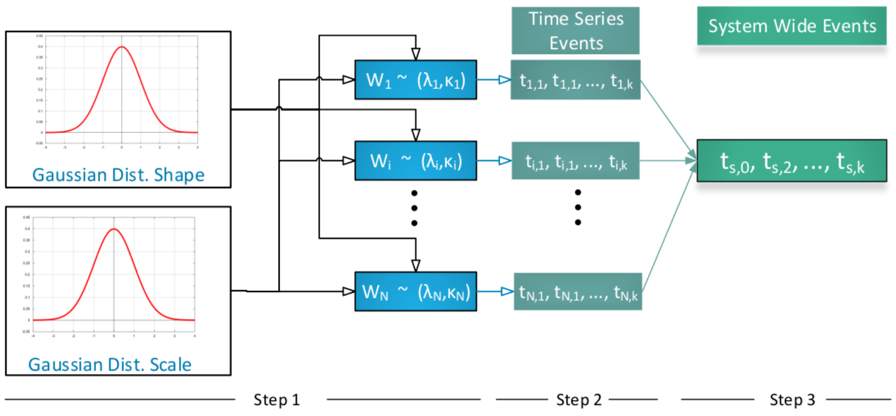

Gaussian Dist. Shape
Gaussian Dist. Scale
Time Series Events
System Wide Events
$W_1 \sim (\lambda_1, \kappa_1)$
$W_i \sim (\lambda_i, \kappa_i)$
$W_N \sim (\lambda_N, \kappa_N)$
$t_{1,1}, t_{1,1}, \ldots, t_{1,k}$
$t_{i,1}, t_{i,1}, \ldots, t_{i,k}$
$t_{N,1}, t_{N,1}, \ldots, t_{N,k}$
$t_{s,0}, t_{s,2}, \ldots, t_{s,k}$
Step 1
Step 2
Step 3

Fig. 1: Synthetic failure generation process.

2.1 Generator of node failure traces

The key component of our simulated environment is the node failure trace generator. It assumes that the time to failure (TTF) for each node are independently distributed and characterized by the same distribution, but with each node having different distribution parameters. The synthetic node failure generator assigns the parameters to each node's TTF distribution. Each such parameter value is a sample drawn from a normal distribution. The main idea behind this assignment is that appropriate choices for the mean and standard deviation of the normal distribution lead to a predictable range of generated parameters for the TTF distribution of each node, and provide a degree of control on the way in which node parameters are distributed within this range. To make the above failure trace generation method more clear, we use as an example the case study presented in Section 3. We assume a topology of N nodes where TTF for node i is modeled with a Weibull distribution $W_i \sim (\lambda_i, k_i)$, where λ_i denotes the scale and k_i the shape of the corresponding distribution, respectively. Figure 1 summarizes the trace generation flow for this case study. In step 1, scale and shape parameter values are assigned to each node's TTF distribution. $G_\lambda \sim (\mu_\lambda, \sigma_\lambda)$ denotes the normal distribution from which samples are drawn to provide the scale parameter. The corresponding distribution for the shape parameter is $G_k \sim (\mu_k, \sigma_k)$. In the second step, K different failure times are generated for each node. The j^{th} failure time for node i is estimated as $t_{i,j} = t_{i,j-1} + w_{i,j}$ where $w_{i,j}$ is the j^{th} sample of $W_i \sim (\lambda_i, k_i)$ and $t(i,0) = 0$. In the third step, all different $t_{i,j}$ are merged into a single time-series and then sorted to derive the time of system failures, denoted as $(t_{s,1}, t_{s,2}, \dots)$.

2.2 Resource allocation alternatives

The second major component of the simulation environment presented in this paper consists of the different resource allocation approaches. The first, denoted as *Slurm-linear*, is the resource allocation implemented by Slurm's linear selection plugin [2]. Nodes are arranged in an one-dimensional array, and for a request for k nodes with no overcommit requirement, the first k consecutive available nodes are allocated to the job. The other two approaches are specific to 4D-torus topologies, and are based on a heuristic that extracts a contiguous and rectangular partition from the 4D-torus topology. Its goal is to avoid contention from other partitions. If static routing is further assumed, this heuristic also ensures that failures of nodes that do not belong to the selected partition will not affect any job running on that. In the current version of the simulation environment, we rely on a simple heuristic for extracting such a partition. However, more elaborate approaches that also consider fragmentation may also be used [20, 15]. The second resource allocation approach implemented is *random partition selector (RPS).* When emulation of a new job's execution is needed, our heuristic populates a list of available contiguous and rectangular topology partitions, and then RPS selects one of them randomly, without consideration of any information or estimate about node failure probabilities. Both *Slurm-linear* and *RPS* are failure-oblivious approaches, and serve as comparison baselines.

The third approach implemented, will be denoted as *failure aware partition selection - FAPS* hereafter. It is based on the finding of [12] and consists of two steps. First, it utilizes the aforementioned heuristic to get a list of available contiguous and rectangular torus partitions, with P_i denoting the i^{th} partition. In the second step, the goal is to select the partition that is the least probable to fail during that the execution duration of the job being scheduled. Let us assume a topology of $n = 1...N$ nodes where resources needed to be allocated for the j^{th} with duration d_j. As per the case study presented in Section 3, we assume that each node's TTF follows a Weibull distribution. For simplicity, let t denote the uptime of the n^{th} node after its last failure. Following the findings of [12], the probability that a node n will fail in d_j given that it has survived until t is expressed through Equation 1, where k_n denotes the shape parameter of the n^{th} node's TTF distribution and λ_n the corresponding scale parameter.

$$p_n^f = P(T \leq d_j + t|t) = 1 - e^{\frac{t^{k_n} - (d_j + t)^{k_n}}{{\lambda_n}^{k_n}}} \tag{1}$$

Then, the probability of a partition P_i failing in d_j is derived via Equation 2, which enables the proposed resource allocation approach to identify the partition with the lowest failure probability $(\underset{i}{\text{argmin}} P_i^f)$.

$$P_i^f = 1 - \prod_{n=1}^{N} (1 - p_n^f) \tag{2}$$

2.3 Impact of node failures to a batch of MPI parallel jobs

The third component of the simulation environment is the logic that schedules jobs for execution. This component checks if the synthetic node failure generator has generated any fault on any node that belongs on the partition assigned to that job. The input to this component is a batch of jobs, where jobs durations are assumed to be known. More precisely, job durations can be specified through a post-processed trace of a distribution. In the current version of our simulation environment, no concurrent job execution is emulated. Instead, each job j_{k+1} is assigned resources and scheduled for execution only after job j_k completes. When a node failure occurs at some point in time, the uptime of that node is set to 0. If this node is assigned to the job whose execution is currently emulated, then this job is marked as aborted. When the job is aborted, both RPS and FAPS rerun their corresponding scheduling logic to determine a new partition to assign the job to.

3 Evaluation Case Study

In this section we present a case study of a system consisting of 4096 nodes, arranged in an 8x8x8x8 4D-torus topology. The TTF of every node follows a Weibull distribution, with different parameters for each node. We use our simulation environment to compare the time required to complete a batch of jobs in the presence of failures, when resources are allocated according to the approaches discussed in Section 2: the two failure-oblivious ones (*Slurm-linear*, *RPS*), and the failure-aware approach FAPS. We consider different simulation scenarios by using different parameters for the normal distributions depicted in Figure 1, thus controlling the combination of shape and scale parameters assigned to the Weibull distribution characterizing the TTF of each node.

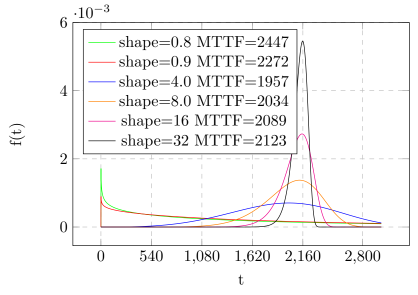


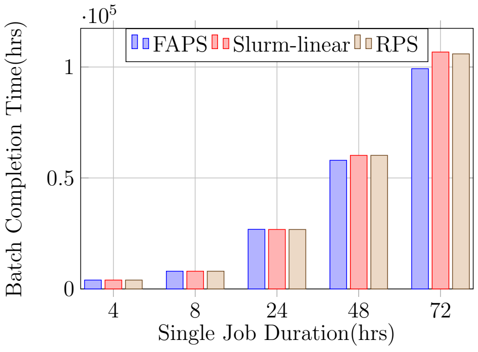
Fig. 2: Impact of shape on node TTF.

	Scale		Shape		Avg node	System
tid	μ_λ	σ_λ	μ_k	σ_k	MTTF	MTTF
1	5800	0.1	8	0.1	5462	1.37
2	5800	0.1	32	0.1	5700	1.42
3	8500	0.1	8	0.1	8004	2.04
4	8500	0.1	32	0.1	8345	2.13
5	16000	0.1	8	0.1	15068	4.04
6	16000	0.1	32	0.1	15726	4.27
7	22000	0.1	8	0.1	20718	5.36
8	22000	0.1	32	0.1	21623	5.72

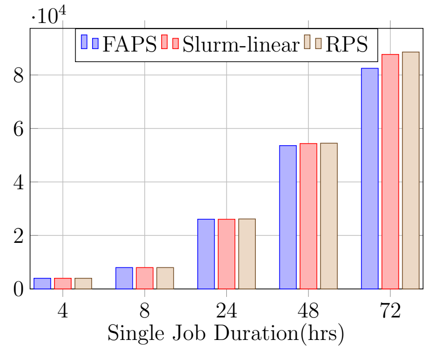
Table 1: Normal distribution parameters for scale & shape, and resulting MTTF (hours).

Figure 2 shows how the corresponding shape parameter affects a node's TTF, with the scale value set to a value corresponding to 2160 hours (i.e. uptime of approximately 3 months). Each curve corresponds to a different pdf. As shape values become larger, a single node's MTTF approaches the corresponding scale value. Each of the 8 main rows in Table 1 corresponds to a different node failure trace. For each trace, we derive scale and shape parameters for the TTF distribution of 4096 nodes following the process depicted in Figure 1. Columns 2 and 3 are, respectively, the mean and standard deviation of the normal distribution that generates the values for the scale parameter of each node. Columns 4 and 5 are the mean and standard deviation parameters of the normal distribution that generates the corresponding shape values for each node. Each pair of scale and shape values defines the parameters of each node's TTF following the Weibull distribution. Following the process described in Section 2.1, time series of failures for each node are generated. Merging and sorting the time series of all nodes, for a period of 10 years, allows us to extract the system-wide MTTF. For the experiments discussed in the rest of this section, for both normal distributions we set the standard deviation to a rather small number compared to the mean, resulting in an homogeneous cluster of nodes (in terms of their scale and shape parameters). For the normal distribution that generates scale values, we have used two alternative settings for the mean: 5800 and 22000. These settings correspond to average scale values over all nodes. From the first two traces in Table 1, we notice that, when the average scale value is 5800, the corresponding average MTTF over all nodes is 5700 for the higher average shape value, and 5462 for the setting of 8. So, the average scale value used in the first normal distribution, when combined with large shape values, directly affects each node's MTBF. Another observation is that, although average MTBF over all nodes is as high as 5800 in the first two traces, the corresponding system wide MTTF to failure is 1.37 hours, suggesting that, there is at least one node failure every 1.37 hours. Even when the average node MTBF is 21623 hours which corresponds to one failure per 2.46 years approximately, the system wide MTTF is 5.72 hours.

Next, we present results for each one of the simulated scenarios explored. The description of each simulated scenario consists of the following information: (a) number of jobs in the batch, (b) job duration for each job instance, (c) job size in terms of number of processes, and finally (d) a synthetic trace of failures. For the case study presented in this section, we use one of the eight traces listed in Table 2. For each simulated scenario, before emulating each job's execution, a resource allocation is carried out using the three methods described in Section 2.2. To determine whether a job execution fails, we extract all nodes that belong to the partition assigned to that job, for each allocation method. If the corresponding failure trace indicates a failure for one or more nodes, the job is considered aborted. Then, batch completion time is augmented by $t_{j,i} + rt$ where rt denotes the node repair time and $t_{j,i}$ job's j execution time until the failure. After a node is rebooted and considered fixed, its uptime is reset to 0. The aborted job is rescheduled for execution, i.e. the resource allocation step is repeated until its execution completes. For the simulation results presented here, the topology size is set to 4096 nodes, arranged in an 8x8x8x8 4D-torus. Job size is set to 256 processes, and we assume batches of homogeneous jobs, i.e. jobs of the same duration. However, different simulation scenarios are possible with different job sizes and durations drawn from traces. Batch size is set to 1000 jobs, and we consider 5 different job durations: 4, 8, 24, 48, and 72 hours. Repair time (rt) for a failed node is set to 9 minutes. The simulation environment discussed though, also allows to specify a distribution for deriving node repair times.



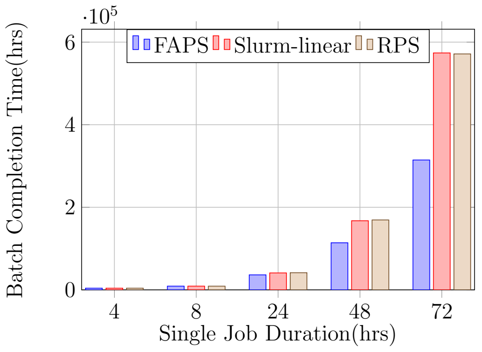
(a) SMTTF=5.36, shape=8



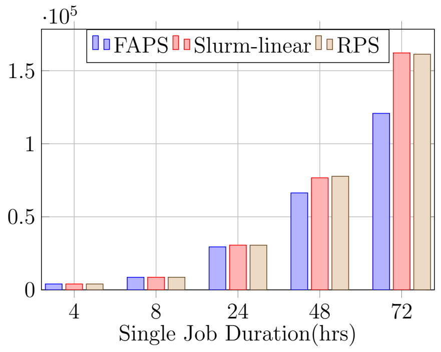
(b) SMTTF=5.72, shape=32

Fig. 3: Completion Time for 1000 Jobs, 256 processes, Scale=22000.

Due to space limitations, we focus only on the traces that resulted in the lower and higher system MTTF (denoted as *SMTTF*). These 4 traces are enough to describe the key patterns observed in all our simulated scenarios. A common observation for Figures 3a, 3b is that for the shorter duration jobs emulated (4, 8, 24 hours), the benefit of the failure aware resource allocation (*FAPS*), in



(a) SMTTF=1.37, shape=8



(b) SMTTF=1.42, shape=32

Fig. 4: Completion Time for 1000 Jobs, 256 processes, Scale=5800.

terms of batch completion time, is rather limited. More on that, for the scenario in Figure 3a that corresponds to an average shape value of 8, it is marginally worse than the two failure-oblivious approaches (*Slurm-linear*, *RPS*). There are two main reasons for that. First, for larger scale values of the node's TTF Weibull distribution, failures become more rare over time, and thus more shorter jobs may *fit* between two successive system failures. Moreover, a batch of 1000 72-hour long jobs, runs for longer and thus, more errors accumulate during the batch's lifetime on the cluster. In such scenarios, there is more room for a failure-aware approach to show benefit. Figures 3a, 3b also show that with longer jobs the failure aware resource allocation approach offers notable benefit. For the case of a batch of 48 hours long jobs and an average shape of 32, it achieves 1.4% and 1.6% lower batch completion when compared to *Slurm-linear* and *RPS*, respectively. Although this reduction might not seem much, the failure aware approach achieves to save 768 and 895 hours respectively. The corresponding improvements are even higher for the case of 72 hours-long jobs (6.3% and 7.3%). There is another interesting observation regarding the effect of the average shape parameter. Comparing the benefit of *FAPS* over *RPS*, for the scenarios in Figures 3a and 3b, we observe that it is higher for the case of the lower average shape value. The corresponding benefit is 5.6% for an average shape value of 8, and 5.1% for a value of 32. As Figure 2 shows, smaller shape values imply a more *flattened* curve for the TTF distribution which in turns implies that node failures are more *dispersed* over time. A larger shape value suggests that TTF samples are more *clustered* around the scale value; many nodes are expected to fail in close-by points in time. Under such conditions, all the partitions explored by *FAPS* are expected to have similar failure probabilities, leaving little room for a beneficial selection. An observation that further supports this assumption is that the difference in benefit would be larger in a trace with more frequent faults, where *FAPS* is more often incapable to find a *good* partition to assign to a job. Indeed, Figures 4a, 4b reveal that the benefit of *FAPS* over *RPS* is 23.5% for an average shape value of 8 and 79% for the case of 32. Our partition selection heuristic is a rather simple one, as it does not perform an exhaustive search for all possible partitions. Consequently, it yields a limited set of partitions in which *FAPS* will search for the less probable to fail. We are currently exploring further the validity of this intuition, and the improvement potential with an enhanced variant of the partition selection heuristic.

Figures 4a and 4b cover another pair of interesting scenarios. They correspond to traces 1 and 2 in Table 1 and describe a platform with more frequent failures. For the two different average shape values explored, the corresponding system MTTF are 1.37 and 1.42 hours, respectively. Following the pattern observed before, for jobs of shorter durations, such as, 4 and 8 hours, the failure aware resource allocation method achieves the same batch completion time with the failure oblivious ones. However, *FAPS* offers a notable benefit even for jobs with a 24 hour duration. Following the pattern regarding shape's effect before, this benefit is larger for the smaller average shape value explored. At the scenario with 48-hour long jobs and an average shape value of 8, *FAPS* achieves 46.8% and 48.4% lower batch completion time when compared to *Slurm-linear* and *RPS*. This benefit remains significant for the scenario depicted in Figure 4b where the average shape value is larger (32). These results indicate that the type of jobs (in terms of duration) that benefit from failure-aware resource allocation is affected by the system failure pattern. In the paragraphs above, we discussed results from simulation scenarios with 256 process. We have also derived results for scenarios with 512 processes; however, due to space limitations, the corresponding graphs are omitted. Our results correspond to traces 5 and 6 from Table 1. Again, we observe marginal or not benefit with the failure-aware resource allocation method for job durations up to 24 hours. For longer jobs and an average shape value of 8, we observe a more notable benefit. However, for the larger shape value, the benefit over the failure-oblivious approaches, even for 72-hour long jobs is limited. We are currently exploring which of two factors contributes the most to this effect: limited number of partitions enumerated by our heuristic, and implications of a large shape value on node failures.

4 Related Work

Simulations of parallel applications have been a valuable tool for exploring their performance and scalability in large scale setups. They also allow to evaluate applications in setups that are different than the real HPC platforms that are available, offering the advantages of a controlled and configurable environment. Towards this direction, several simulators have been proposed. Simgrid [4] allows the simulation of unmodified applications, while xSim [8] allows running an application at a scale of up to millions of concurrent threads. LogGOPSim [8] allows the simulation of parallel algorithms at large scale relying on an extended version of the *LogGPS* model. For systems of growing size and complexity, a large number of studies have outlined the importance and effect of various failures [16, 23, 21, 3] apart from performance. For mitigating the effect of failures several approaches have been proposed including checkpoint/restart [14, 27, 25], failure aware scheduling and resource allocation [5, 13, 30, 10, 18, 11]. The resource allocation approach explored in this study is heavily based on the findings of [12] and follows a similar path for the resource allocation with [11]. More precisely, authors in [11] present algorithms that allocate resources for MPI jobs, such that system reliability is maximized. They take into account the probability of nodes failing during the execution time of each job. A common approach for evaluating the effectiveness of fault tolerance mechanisms relies on failure logs and traces acquired from real HPC systems. It is important though, to be able to evaluate such approaches in larger scales and under different system configurations and failure patterns. Towards this direction several studies have proposed simulators aimed at evaluating fault tolerance mechanisms (apart from performance and scalability). The work in [9] extends xSim [8] adding support to inject MPI process failures and explore the efficiency of checkpoint/restart. Authors in [24] discuss a simulator that aims at exploring proactive and reactive fault tolerance mechanisms, as well as a combination of the two. For its evaluation, they replay traces from failure logs. With a focus on coordinated and uncoordinated checkpoint/restart protocols, authors in [17] suggest a simulation framework based on LogGOPSim [8]. The work in this study does not target a full system simulator. Our main focus is on different failure patterns and their effect on resource allocation for parallel jobs. Our key contribution is a configurable mechanism that generates different traces of node failures.

5 Conclusions and Future Work

This paper is an early exposition of our modeling and simulation approach towards quantifying the impact of node failures on the completion times of MPI parallel jobs. We generate synthetic traces of node failures in a case study that assumes a 4D-torus topology and a Weibull distribution for each node's mean time between failures. For the synthetic traces explored, benefit is more prominent for the longer jobs. It can be up to 82% depending on the failure trace, when compared with Slurm and a failure-oblivious heuristic. For shorter jobs, benefit is notable for systems with more frequent failures.

Our research plan going forward includes more comprehensive case studies for larger-scale supercomputers incorporating more nodes and more complex interconnection topologies (such as 6D-torus and Dragonfly). We also plan to extend the scheduling logic for concurrent job execution and explore different distributions for the time to failure. Furthermore, we plan to incorporate node failure prediction in Slurm, by taking advantage of its software plug-in architecture. This extension will be building upon the software infrastructure created in our prior work towards adding failure awareness to resource allocation in Slurm [26].

Acknowledgments

This research has received funding from the European Union's Horizon 2020/EuroHPC research and innovation programme under grant agreements 955606 (DEEP-SEA) and 754337 (EuroEXA). National contributions from the involved state members match the EuroHPC funding.

References

1. Etp4hpc-sra 4:strategic research agend for high performance computing in europe. `https://www.etp4hpc.eu/pujades/files/ETP4HPC_SRA4_2020_web.pdf`
2. Slurm Resource Selection Plugin. `https://slurm.schedmd.com/selectplugins.html`
3. Cappello, F., Al, G., Gropp, W., Kale, S., Kramer, B., Snir, M.: Toward exascale resilience: 2014 update. Supercomput. Front. Innov.: Int. J. **1**(1), 5–28 (Apr 2014)
4. Casanova, H., Giersch, A., Legrand, A., Quinson, M., Suter, F.: Versatile, scalable, and accurate simulation of distributed applications and platforms. Journal of Parallel and Distributed Computing **74**(10), 2899–2917 (Jun 2014)
5. Dogan, A., Ozguner, F.: Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems **13**(3), 308–323 (2002)
6. El-Sayed, N., Zhu, H., Schroeder, B.: Learning from failure across multiple clusters: A trace-driven approach to understanding, predicting, and mitigating job terminations. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS). pp. 1333–1344 (2017)
7. E.N.Elnozahy, Bianchini, R., El-Ghazawi, T., Fox, A., Godfrey, F., Hoisie, A., McKinley, K., Melhem, R., Plank, J., Ranganathan, P., Simons, J.: System resilience at extreme scale. Tech. rep., Defense Advanced Research Project Agency (2008)
8. Engelmann, C., Lauer, F.: Facilitating co-design for extreme-scale systems through lightweight simulation. In: 2010 IEEE International Conference On Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS). pp. 1–8 (2010)
9. Engelmann, C., Naughton, T.: Toward a performance/resilience tool for hardware/-software co-design of high-performance computing systems. In: 2013 42nd International Conference on Parallel Processing. pp. 960–969 (2013)
10. Fu, S.: Failure-aware resource management for high-availability computing clusters with distributed virtual machines. J. Parallel Distrib. Comput. **70**(4), 384–393 (Apr 2010)
11. Gottumukkala, N.R., Leangsuksun, C.B., Taerat, N., Nassar, R., Scott, S.L.: Reliability-aware resource allocation in hpc systems. In: 2007 IEEE International Conference on Cluster Computing. pp. 312–321 (2007)
12. Gottumukkala, N.R., Nassar, R., Paun, M., Leangsuksun, C.B., Scott, S.L.: Reliability of a system of k nodes for high performance computing applications. IEEE Transactions on Reliability **59**(1), 162–169 (2010)
13. Hakem, M., Butelle, F.: Reliability and scheduling on systems subject to failures. In: 2007 International Conference on Parallel Processing (ICPP 2007). pp. 38–38 (2007)
14. Heien, E., LaPine, D., Kondo, D., Kramer, B., Gainaru, A., Cappello, F.: Modeling and tolerating heterogeneous failures in large parallel systems. In: SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 1–11 (2011)
15. Hyunseung Choo, Seong-Moo Yoo, Hee Yong Youn: Processor scheduling and allocation for 3d torus multicomputer systems. IEEE Transactions on Parallel and Distributed Systems **11**(5), 475–484 (2000)
16. Jauk, D., Yang, D., Schulz, M.: Predicting faults in high performance computing systems: An in-depth survey of the state-of-the-practice. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. pp. 30:1–30:13. SC '19, ACM, New York, NY, USA (2019)

17. Levy, S., Topp, B., Ferreira, K.B., Arnold, D., Hoefler, T., Widener, P.: Using simulation to evaluate the performance of resilience strategies at scale. In: Jarvis, S.A., Wright, S.A., Hammond, S.D. (eds.) High Performance Computing Systems. Performance Modeling, Benchmarking and Simulation. pp. 91–114. Springer International Publishing, Cham (2014)
18. Machida, F., M. Kawato, Maeno, Y.: Redundant virtual machine placement for fault-tolerant consolidated server clusters. In: 2010 IEEE Network Operations and Management Symposium - NOMS 2010. pp. 32–39 (2010)
19. Martino, C.D., Kramer, W., Kalbarczyk, Z., Iyer, R.: Measuring and understanding extreme-scale application resilience: A field study of 5,000,000 hpc application runs. In: 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. pp. 25–36 (2015)
20. Oliner, A.J., Sahoo, R.K., Moreira, J.E., Gupta, M., Sivasubramaniam, A.: Fault-aware job scheduling for bluegene/l systems. In: 18th International Parallel and Distributed Processing Symposium, 2004. Proceedings. pp. 64– (2004)
21. Schroeder, B., Gibson, G.: Understanding failures in petascale computers. Journal of Physics: Conference Series **78** (09 2007)
22. Schroeder, B., Gibson, G.A.: A large-scale study of failures in high-performance computing systems. IEEE Transactions on Dependable and Secure Computing **7**(4), 337–350 (2010)
23. Snir, M., Wisniewski, R., Abraham, J., Adve, S., Bagchi, S., Balaji, P., Belak, J., Bose, P., Cappello, F., Carlson, B., Chien, A., Coteus, P., DeBardeleben, N., Diniz, P., Engelmann, C., Erez, M., Fazzari, S., Geist, A., Gupta, R., Van Hensbergen, E.: Addressing failures in exascale computing. ICiS Workshop **ANL/MCS-TM-332** (04 2013)
24. Tikotekar, A., Vallee, G., Naughton, T., Scott, S.L., Leangsuksun, C.: Evaluation of fault-tolerant policies using simulation. In: 2007 IEEE International Conference on Cluster Computing. pp. 303–311 (2007)
25. Tiwari, D., Gupta, S., Vazhkudai, S.S.: Lazy checkpointing: Exploiting temporal locality in failures to mitigate checkpointing overheads on extreme-scale systems. In: 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. pp. 25–36 (2014)
26. Vardas, I., Ploumidis, M., Marazakis, M.: Towards communication profile, topology and node failure aware process placement. In: 2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). pp. 241–248 (2020)
27. Y. Li, Z. Lan: Exploit failure prediction for adaptive fault-tolerance in cluster computing. In: Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06). vol. 1, pp. 8 pp.–538 (2006)
28. Yigitbasi, N., Gallet, M., Kondo, D., Iosup, A., Epema, D.: Analysis and modeling of time-correlated failures in large-scale distributed systems. In: 2010 11th IEEE/ACM International Conference on Grid Computing. pp. 65–72 (2010)
29. Yoo, A.B., Jette, M.A., Grondona, M.: Slurm: Simple linux utility for resource management. In: Feitelson, D., Rudolph, L., Schwiegelshohn, U. (eds.) Job Scheduling Strategies for Parallel Processing. pp. 44–60. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
30. Zhang, Y., Squillante, M.S., Sivasubramaniam, A., Sahoo, R.K.: Performance implications of failures in large-scale cluster scheduling. In: Proceedings of the 10th International Conference on Job Scheduling Strategies for Parallel Processing. p. 233–252. JSSPP'04, Springer-Verlag, Berlin, Heidelberg (2004)