



Early Termination of Failed HPC Jobs Through Machine and Deep Learning

Michał Zasadziński¹(✉), Victor Muntés-Mulero¹, Marc Solé¹, David Carrera²,
and Thomas Ludwig³

¹ CA Technologies, Barcelona, Spain

{michal.zasadzinski,victor.muntes,marc.solesimo}@ca.com

² Universitat Politècnica de Catalunya, Barcelona, Spain

dcarrera@ac.upc.edu

³ Deutsches Klimarechenzentrum GmbH, Hamburg, Germany

ludwig@dkrz.de

Abstract. Failed jobs in a supercomputer cause not only waste in CPU time or energy consumption but also decrease work efficiency of users. Mining data collected during the operation of data centers helps to find patterns explaining failures and can be used to predict them. Automating system reactions, e.g., early termination of jobs, when software failures are predicted does not only increase availability and reduce operating cost, but it also frees administrators' and users' time. In this paper, we explore a unique dataset containing the topology, operation metrics, and job scheduler history from the petascale Mistral supercomputer. We extract the most relevant system features deciding on the final state of a job through decision trees. Then, we successfully train a neural net to predict job evolution based on power time series of nodes. Finally, we evaluate the effect on CPU time saving for static and dynamic job termination policies.

Keywords: HPC · Slurm · Failure prediction · Failure prevention
Deep learning · Data center

1 Introduction

Data centers are a core element in most IT systems, hosting cloud applications, enabling HPC or performing intensive Big Data analytics. Although the optimal architecture of a data center may be different for each of these applications, general maintenance problems remain the same. Failures in hardware and infrastructure can both cause software failures or may be the result of such software failures. Software errors are the most common cause of failures [4]. Also, many jobs produce large network and storage system loads which degrade the system performance [3].

The original version of this chapter was revised: For detailed information please see correction chapter. The correction to this chapter is available at https://doi.org/10.1007/978-3-319-96983-1_58

Data presenting the state of a system is usually so complex that administrators might not take the best decision to recover a system efficiently. Moreover, in many cloud-oriented services, system monitoring information is limited to hardware metrics, and do not include user application logs. Thus, it is even more challenging to predict job failures and take proper action. Evaluating jobs in run-time augments administrative metrics and increases the confidence of taken decisions. Therefore, jobs which are likely to fail or decrease the performance of a system can be terminated in advance, saving resources, computing and human time, and lowering operational costs. According to the dataset used in this paper, completed jobs in the petascale Mistral¹ supercomputer consume about 45 million CPU hours per month and they are 91.3% of all submitted jobs. Predicting the final job state at the time of job submission and during run-time allows for forcing job termination before a failure occurs, enabling savings. However, deciding when it is necessary to terminate a job is a nontrivial task.

In this paper, we analyze the impact of both **static** and **dynamic** job termination **policies** using different data center metrics. We propose new job state prediction algorithms based on Decision Trees (DT) and Convolutional Neural Networks (CNN). We use data extracted from the Mistral supercomputer that includes system metrics, job scheduler history, and system topology information. We augment datasets during the exploration to show how knowledge coming from job scheduler, monitoring system, and topology and structure, can increase prediction capabilities and uncover new patterns. We discriminate among job submission features these that explain the termination status of jobs based on job traces. We use power series of nodes to build a model used for failure prediction at run-time. For this task, we use machine learning (ML) and a CNN. The trained CNN achieves 85% of precision in the classification of failed jobs by power series. The CNN predicts failures for more than 40% of failed jobs in the 20th percentile of their duration.

The remainder of this paper is divided into six sections. Work related to failure prediction and prevention in HPC is discussed in Sect. 2. In Sect. 3, we provide description of Mistral supercomputer (42nd most powerful computer in November 2017 (See footnote 1)) and data exported from this environment. Section 4 presents extraction of important features and their discovery by means of DTs that are created using these data. Then, in Sect. 5, we describe the training and use of a CNN for job state prediction. At the end of Sect. 5, we show savings applying different policies for early job termination. We discuss results, the usefulness of the proposed policies and include plans for future research in Sect. 6.

2 Related Work

Authors in [18] describe the role of software in failures [occurring in data centers. Software problems in an OS, middleware, application, or the wrong configuration, e.g., underestimated resources cause the majority of job failures in HPC

¹ <https://www.top500.org/system/178567>.

workload [1,22]. The authors of [5] discover the correlation between failures, and different characteristics of supercomputer operations, such as node usage, last state of a job, and hardware metrics. This research explores state sequences from the perspective of a node. The authors perform job-oriented analysis only to point users with a high failure rate. Analysis of logs and the rate of failed jobs allows detecting slow-downs and targeted failures [4]. Recently, the authors in [10] characterized workload in an HPC environment with the primary goals to find patterns across different applications and disciplines. Latest work presented in [9] analyzes failures of the Oak Ridge supercomputer. The authors describe hardware reliability, correlate failure types, and investigate failure trends in the time and spatial distribution. However, leveraging user history for prediction of failed jobs and learning application workload patterns is not a primary focus area in the prior publications. Also, there are not many publications addressing the separate analysis of jobs and job steps.

There is many research on ML in data center maintenance for either prediction or classification problems [6,7,17,20]. For instance, research in [8] uses dynamic association rules to predict failures in the Blue Gene. The authors of [15] focus on predicting failures in computing nodes, and as a reaction, redirecting a job to another set of nodes. Another possible action is checkpointing, and the authors of [2] investigate the optimal policy to reduce trade-off between checkpoints frequency and MTBF. The authors in [13] use power and temperature metrics to predict errors in GPU clusters via neural network (NN) model. Recently, decision trees are implemented for failure prediction in HPC domain [12]. The proposed algorithm identifies the causes of failures, performing better comparing to other SoA techniques.

Despite the popularity and progress in ML algorithms and software, the area of prediction of the final HPC job states through accurate modeling of power series seems to be unexplored. The focus of most of the work is put on predicting failures per hardware unit, rather than learning workload patterns of failed jobs. The complexity of IT systems and their dynamic structure are one of the main obstacles to create accurate models. The authors in [21] propose power modeling techniques via Petri networks, to estimate power consumption. Also, work presented in [19] reports research on power profiling in HPC environments. The authors discuss application network architecture, performance, and scalability in the dimension of power consumption, and they propose a system for accurate power monitoring. However, in our work, we aim to use as little information as possible – power metrics representing the load of nodes.

3 Mistral Supercomputer Dataset

DKRZ Mistral supercomputer contains 3336 computing nodes, about 90 special nodes dedicated to other activities, and a separate 54 PiB Lustre file system. Applications for climate science generate the production workload. Slurm² maintains node reservations, resource allocation, and accounting.

² <https://slurm.schedmd.com/>.

3.1 Job Scheduler History

Through analysis of historical data from the scheduler, we investigate which features are important, thus deciding on a final job state. This goal motivates our strategy, which is oriented to jobs rather than nodes. We use states from the scheduler to determine an output of a job. In the dataset, each job finishes with one of the following states, defined by Slurm documentation.

- **Cancelled** – A user or administrator cancelled a job. The job may or may not have been initiated. In the following analysis, we take into account only cancelled jobs longer than 0 s.
- **Completed** – Job has terminated all processes on all nodes with an exit code of zero.
- **Failed** – Job terminated with non-zero exit code or another failure condition. According to Mistral, another failure condition includes failures caused by any external factor to an allocated node, e.g., failures of Lustre FS, IB.
- **Node fail** – Job terminated due to a failure of one or more allocated nodes. This state includes only hardware related problems of a computational node.
- **Timeout** – Job terminated upon reaching its time limit.

Each job consists of one or more steps. A job submission script defines the execution order of steps; also the order can be read from Slurm history. The order can be sequential, parallel, or mixed, see example script in Listing 1.1.

Listing 1.1. Example Slurm batch script. Two steps run sequentially on 80 nodes.

```
#SBATCH --nodes 80
#SBATCH --tasks-per-node 10
# First step
srun --nodes 80 --tasks 10 mkdir /home/$USER/$SLURM_JOBID
# Second step
srun app.mpi in.csv out.csv
```

Most steps in Mistral dataset are executed sequentially. In the Slurm database, there are 76 columns. They contain information about jobs: (1) job configuration specified by a user, and (2) statistics known at the end of a job. We give more details about these data in Subsect. 3.3. In this paper, we consider all above **job** states. For **steps**, the dataset includes: Completed, Failed and Cancelled.

3.2 Time Series Data Analysis

Mistral metrics are acquired every 60s into an Open Time Series Database (OTSDB) instance that is installed on the top of HBase cluster. For this research, the data from the cluster are exported using the HBase ExportSnapshot tool. Then, we import a snapshot with the size of 0.5 TB from a regular continuous period of 10 months of system executions to our analysis environment containing 8 machines with 120 physical cores, 672 GB of RAM. We use Apache Spark for data processing. For training of a CNN, we need job scheduler data merged with power metrics. We merge Slurm steps with data from OTSDB representing power metrics of nodes used by a step during its run-time. That merged steps

should contain at least two power measurements. In the worst case, for steps shorter than 120 s, it is possible to merge only one timestamp with node power metrics. So, in the evaluation, we consider a subset of steps longer than 120 s. Discarding short jobs, we do not lose a lot of data: about 1.2M of all steps from the set run for more than 60 s and 1.1M more than 120 s.

3.3 Dataset Split

We show how different knowledge sources: software – job scheduler, hardware – monitoring system, and platform – topology and structure, impact prediction and classification accuracy. Also, we detect which part of the data increases the prediction capabilities of a model when the only used information is the one known at the time of a job submission; and which part of the data improves classification capabilities, when we use statistics of finished jobs. Datasets are divided into the following sets - named with a capital letter for later reference:

- **Slurm job configuration data:** information of either jobs or steps, which is known at the time of submission e.g., reserved time, allocated nodes, required CPU frequency, start time. [we call it *dataset C* in the experiments]
- **Slurm user data:** columns with information about prior user allocations. Also, this dataset contains aggregated user data. The set includes factors of jobs terminated with each of 5 possible states to a number of all submissions in different windows. We aggregate the data by user and windows with different sizes: last N submissions ($N = 1, 100, 1k, 10k$). [*dataset U*]
- **Slurm job summary data:** information is known at the **end of a job**, e.g., duration, disk read/write (R/W) – the sum of local storage and Lustre operations done by a job, virtual memory (VM) size, other hardware usage. [*dataset S*]
- **Power metrics of nodes (OTSDB data):** power metrics of computing nodes (blades). [*dataset P*]
- **Data center topology:** topology and localization of nodes. [*dataset T*]
- **Hardware profiles of nodes:** types of nodes, number and types of CPUs, amount of RAM. [*dataset H*]

4 Failed Job Analysis

According to the data from the job scheduler, more than 1.3M jobs, and more than 270k different job names are submitted in the 10-month period that is represented by the dataset extracted from the Mistral production environment. These submissions, which are mainly executed in batch mode (98.8%), result in over 4.8M steps. For detailed statistics, see Mistral technical report [23]. One of the observations from the statistics is coherent with usual state of the art reports - failed steps are usually more complex [22]. These statistics represent a convincing motivation for generating savings with the early termination of jobs that are predicted to fail. An average failed job consumes many more CPU hours than completed one and it also decreases resources availability.

4.1 Most Meaningful Features for Prediction of Job States

Extraction of Features. We generate DTs [14] to reveal job and step features explaining a job state. These ML models learn *if-then-else* rules, for either classification or regression task. An advantage of using a DT is the fact that it is a white-box model so that a human can easily understand a trained tree. We use all the features from each dataset for generation of a DT. To decide the optimal size of DTs, we consider (1) over-fitting and (2) readability of a model to a human. Firstly, we split our set into three sets using random stratified sampling. We create the training set containing 70% of jobs (samples), the validation set that has 10% of jobs (samples), and the test set with 20% of jobs. During the training, we measure accuracy on the validation set, while increasing depth of a tree. We set 100 as the minimum number of instances each node’s child must have after a split. Trees with depth 5 obtain satisfactory performance. For larger DTs, the accuracy increase is low (0.03%), and the increase of the number of nodes is high. For instance, a tree with depth 9 has 275 nodes, and it is 84 nodes more than a DT with depth 8. Thus, we choose the optimal depth of the DT to be 5, which has 63 nodes. To check if models are not over-fitting, we evaluate random forests (RF) for each dataset. RF create DTs and train them with different training sets that are subsets of the main training set. Then, results from each DT are combined. Created RFs improve neither classification nor prediction quality when compared to the above DTs.

The test evaluations shows the fitness of generated models of either classification (having all information about a finished job), or prediction (having only information at the time of submission). We present the results of evaluations in Tables 1 and 2, including only features with importance over 3%.

Table 1. Decision trees – evaluation of different combinations of data sets - **jobs**

Data set	Important features	Job state	Completed	Failed	Cancelled	Timeout	Node fail
Configuration (C)	Time limit (74%)	Precision	0.91	0.0	0.0	0.0	0.0
	Daytime (24%)	Recall	1.0	0.0	0.0	0.0	0.0
		F1-score	0.96	0.0	0.0	0.0	0.0
Configuration + user’s history (C + U)	Previous job state for a user (96%)	Precision	0.97	0.75	0.52	0.68	0.0
	Number of allocated nodes (3%)	Recall	0.98	0.70	0.44	0.30	0.0
		F1-score	0.98	0.72	0.48	0.42	0.0
Statistics + configuration + user’s history (S + C + U)	Previous job state (87%)	Precision	0.97	0.77	0.63	0.81	0.0
	Duration (9%)	Recall	0.99	0.74	0.36	0.35	0.0
	Number of allocated nodes (4%)	F1-score	0.98	0.75	0.46	0.49	0.0
Duration > 120 s statistics, configuration, user’s history (S + C + U)	Previous job state (85%)	Precision	0.97	0.81	0.62	0.80	0.0
	Duration (8%)	Recall	0.99	0.74	0.31	0.33	0.0
	Number of allocated nodes (6%)	F1-score	0.98	0.77	0.41	0.47	0.0

Jobs. The above results show that the size of resource reservation is a principal factor determining the final state of a job. Also, the results expose that final

Table 2. Decision trees – evaluation of different combinations of data sets - **steps**

Data set	Important features	Job state	Completed	Failed	Cancelled
Configuration (C)	Number of allocated nodes (98%)	Precision	0.95	0.50	0
		Recall	0.99	0.07	0
		F1-score	0.97	0.12	0
Configuration, statistics (C + S)	Number of allocated nodes (47%) Average disk W (40%) Duration (4%)	Precision	0.98	0.83	0.58
		Recall	0.99	0.76	0.04
		F1-score	0.99	0.79	0.79
Duration > 120 s configuration, statistics (C + S)	Average disk W (47%) Number of allocated nodes (36%) Average CPU frequency (9%); Duration (4%)	Precision	0.95	0.59	0.89
		Recall	0.98	0.23	0.83
		F1-score	0.97	0.33	0.86
Configuration, topology, hardware information (C + T + H)	Number of allocated nodes (79%) Number of nodes 36C 64 GB RAM (15%) Number of nodes 36C 128 GB RAM (3%)	Precision	0.97	0.75	0.49
		Recall	0.98	0.41	0.01
		F1-score	0.98	0.53	0.01
Configuration, statistics, topology, hardware information, power statistics (C + S + T + H + P)	Number of allocated nodes (46%) Average disk W (41%); Average disk R (5%) Average VM size (4%)	Precision	0.98	0.85	0.52
		Recall	0.99	0.75	0.10
		F1-score	0.98	0.80	0.17
Duration > 120 s, (C + S + T + H + P)	Average disk W (49%) Number of allocated nodes (35%) Average CPU frequency (10%)	Precision	0.94	0.93	0.81
		Recall	0.99	0.79	0.13
		F1-score	0.97	0.85	0.22

states are highly correlated with a user’s history. In general, this correlation is weaker for longer jobs.

Steps. Generated DTs reveal that the sum of *disk RW* is often higher for completed jobs than failed ones. Since the mean duration of failed steps is much higher than completed ones [23], higher storage usage can be explained by less active nodes in failed steps. We can state a hypothesis, that some nodes in failed steps stay in idle state, see Sect. 4.2. The evaluation shows the high importance of a number of allocated nodes with 36 cores. An investment done in DKRZ explains this phenomenon. The dataset includes the period when Broadwell nodes started their service in the production environment. That time, users were translating their software and scripts to the recently installed hardware. It is the primary cause of many job failures.

Conclusions. The evaluation of DT classification tasks shows that a DT model is unable to learn and recognize cancelled, node failed, or timeout jobs based only on configuration data. These data are the only information known to the scheduler after a job is submitted. The f1-score is 0 for all of the mentioned states. Augmenting this set with past user’s submissions improves recall of failed jobs to 72% and lifts precision of predicting cancellations to 52% and timeouts to 68%. This result shows a strong correlation inside a sequence of final job states. Adding to the training dataset metrics which are known after a job is finished increases the precision of a classifier. The recall does not change for any of the states. Regarding steps, precision and recall are lower than those for job

submissions. It is a reasonable result considering that steps have a lower number of features available for these evaluations. The number of allocated nodes is an important feature to predict the final state of a job even when used with hardware metrics features. Other important features are knowledge on past submissions and their states. According to the hardware statistics, average *disk W* is a highly important feature in the classification task of final job states, while general power statistics are features with low importance.

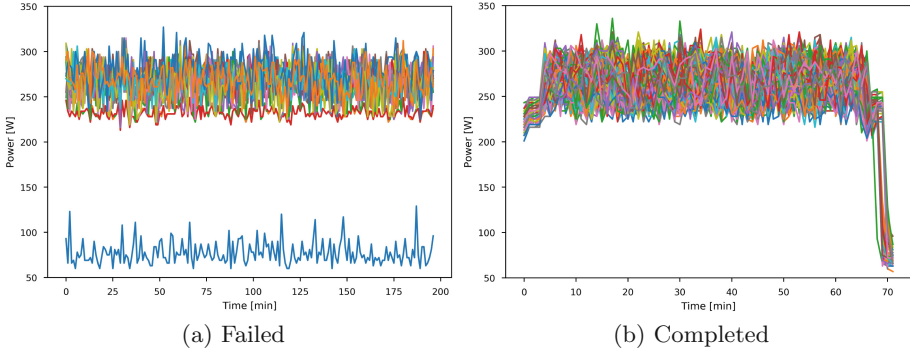


Fig. 1. Plots presenting power series of 198 nodes running in parallel a job from the same, user, project, and application. Two jobs were run in different points of time. First one is failed, the next one is completed.

4.2 Node-Power Analysis

We want to investigate the power statistics of failed jobs in comparison with completed ones, to detect idle states. Each computing blade is controlled and monitored by an isolated blade management controller which delivers power metrics. A controller is an external unit, and acquiring measurements does not infer with the workload of a blade. Power metrics of these blades perfectly depict their CPU load. Although in Subject. 4.1 we evaluate the usefulness of power statistics in prediction, we might also evaluate whether these series can improve job state prediction during the run-time. We correlate power series of nodes allocated for a step with this step’s final state and types of nodes. We analyze power statistics for steps longer than 1000 s, grouped by hardware profile to extract average values of power metrics in the last 300 s of the duration of a job. This value is at least 10% lower than for completed ones, when considering all hardware groups, and failed steps. The most probable explanation can be the fact that once a software failure occurs some of the nodes go to an idle state. For instance, Fig. 1 presents power series of 1-step jobs, both executed with the same configuration by the same user. This scenario represents a typical case where one node is in an idle state, and the rest are executing some workload. On

the contrary, power series of nodes executing a completed step do not show any node in an idle state. This phenomenon appears in other cases in the dataset and suggests that using power metrics would be relevant for classification of a job state. Moreover, this observation matches with the expert knowledge at DKRZ. In words of one of its system engineers: “We check the idle state of a node during a problematic job, looking at InfiniBand traffic of nodes. If it is low, a job is likely to fail”.

5 Prevention of Failures

Prior data exploration and evaluation of DTs show that power metrics and DTs can be used for prediction of final job state. Predictions contain probabilities for each step state. During prediction, we classify a step as failed, when the probability of failure is higher than a defined threshold and all other probabilities associated with other classes. Therefore, we propose two types of policies to be taken: a **static** and **dynamic** one. A static policy uses predictions based on a step configuration data, topology, and hardware information ($C + T + H$) through DTs. A dynamic policy uses predictions during run-time which are produced by a convolutional neural network (CNN), introduced in Sect. 5.1. The inputs to this model are power metrics, which are analyzed in Sect. 4.2. While using a dynamic policy, a job is killed when it is classified as failed for the first time – the earliest prediction over the given threshold.

The use of different types of models, one as a white-box and the other as a black-box has several advantages over, for instance, one complex NN model trained with both static and dynamic data. Firstly, the use of DTs enables to easily explain phenomena observed in a data center to system administrators. Since a model can evolve by repeating the training, changes in trends and user behavior occurred in a data center are observed as results of the comparison of models. Also, a failure prevention system gains performance during the run-time because of splitting evaluation to offline (time of submission only) and online (evaluation of a job during its runtime) one.

5.1 Convolutional Neural Networks

CNNs are a type of deep neural network following a design of biological vision systems [11]. They are widely used for image classification, natural language processing, and recommendation systems, and they have also been successfully used for time series classification and prediction. We propose to use a CNN for classification and prediction of multivariate time series, which are the power metrics of nodes (overall energy consumption of a computing blade) used in a step. Therefore, CNN learns “how a multivariate time series of nodes executing a step look like”. A major advantage of using CNNs over neural nets with fully connected (dense) layers only, is that they need much fewer neurons and parameters to solve a particular classification or prediction problem.

In Fig. 3, we present the best CNN model trained for this task. We create the final model after a few iterations, through dropping layers from more complex models which over-fit during the training and do not increase the accuracy. The model presented in Fig. 3 comprises a few types of layers. Each convolutional layer comprises filters with size 3×3 , and during the training, each filter learns weights. This layer is used to extract specific features, in this case from 2D matrices. Another important layer type used is a drop-out, which regularizes weights and through dropping neurons and connections, prevent overfitting [16]. A max pooling layer and dense layer are used to aggregate extracted features and classify them into defined classes and give probabilities. The input data are 2D matrices of size $M = 512$ (number of nodes) $\times T = 120$ (length of time series). For steps with matrices which shape is less than $M \times T$, we pad a sample with zeros - which are ignored by CNN during the training. For these matrices which are larger than that size, we downsample a matrix by averaging power metrics. The value for T is chosen so that it is large enough to represent the complete series of most of the steps (only 1.3% of steps are longer than 120 min) and at the same time it is small enough for the NN training to be practical. The dataset with steps is split randomly (the same split as in Sect. 4) into three sets: training (70% of the data), validation (10%), and test (20%) respectively.

The CNN is trained using tensorflow³ and keras⁴ libraries by means of 2x GPU GeForce 1080 Ti. Also, after a few trails and examining a shape of the loss curve, the learning rate is set to 0.001, and we choose a stochastic gradient descent optimizer. The final model, which contains 32261 parameters to train, is trained in 67 epochs with approximately 1 h per epoch. We stop training after lack of significant improvement in the loss curve, and when the model does not improve more than 1% in 5 epochs. We show results of the trained CNN in Table 3.

Table 3. CNN test results - classification. Data set: steps – power metrics, duration > 120 s

	Completed	Failed	Cancelled
Precision	0.93	0.85	0.79
Recall	0.98	0.66	0.15
F1-score	0.96	0.74	0.25
test set	168875	28605	4457

5.2 Evaluation – Static and Dynamic Job-Killing Policies

The primary goal of the evaluation is to explore possible savings and losses depending on the aggressiveness of job-killing policy. We measure the aggres-

³ <https://www.tensorflow.org/>.

⁴ <https://keras.io/>.

siveness of a policy as the *threshold of class prediction probability*. For instance, a threshold of 60% means that a job is classified as failed when the probability of predicting failed is higher than 60%. An aggressive policy is the one with a low threshold, and the less aggressive one is the one with a high threshold, e.g., greater than 90%. We evaluate the trained CNN model and DT to predict the final states of steps. We use a test set which contains jobs with total CPU time of 84.7M h. CNN predicts a final job state and outputs probabilities for each timestamp during the run of a job. We evaluate proposed policies by depicting lost and gained CPU time, expressed in hours. Lost CPU time stands for the resources consumed by a step that is labeled as completed, but it is killed (false positive). Saved CPU time represents resources that would be used until a step ends but are saved due to a decision of early step termination. Approximate performance of CNN evaluation is 5000 samples/s which is considered sufficient for these experiments.

Table 4. Summary of the dynamic policy evaluation over a test set containing 11M CPU hours of failed jobs

Dynamic policy metric	CPU h	Probability threshold
Maximum savings achievable	7.9M	From 0 to 0.42
Maximum loss (false positive)	4.1M	0.52
Global maximum (savings - loss)	4.0M	From 0 to 0.42
Local maximum of (savings - loss) with the highest value of threshold	0.7M	0.82

Considering the dynamic policy, the maximum value of true positives is 0.9, and for false positives, the maximum value is 0.45. Both metrics decrease smoothly when the threshold grows. Figure 2 shows true and false positive rates depending on the probability threshold for failure prediction with the CNN. On the other hand, the static policy is characterized by the maximum value of the true positive rate of 0.47 and a small value of 0.02 for the false positive rate. The static policy is more accurate in predictions comparing to the dynamic one, but the maximum number of predicted steps to fail are almost two times lower.

Wastes. When it comes to the CPU time, the static policy allows for maximum savings of 0.8M CPU h, and the dynamic one of 8M CPU h. In Table 4, we present a summary of the evaluation of the dynamic policy taking into account CPU hours of jobs. Note, that the earlier we kill a failed job, the bigger savings are. On the other hand, the confidence of prediction increases with time a job is running as we gather more data. Regarding this trade-offs, there is a global maximum of losses for threshold 0.52. For instance, applying a dynamic policy with a threshold of 0.82 (local maximum with the highest threshold value) to the test dataset saves 1.6M CPU h with 0.9M CPU h lost and the total profit of 0.7M CPU h. For instance, a less aggressive policy would be the application

of a threshold equal to 0.96. In this case, we save 210k CPU h, and we lose 24k CPU h, with the total profit of 190k CPU h. In contrast, executing static policy allows for maximum savings of 870k CPU h by killing 13k failed jobs with a side effect of killing 3.8k completed ones. Also, the application of the static policy, which is more conservative, does not cause a loss in CPU time, because it reacts after job submission.

Figure 4 presents the distribution of job time at which the dynamic policy will react and terminate a job. We can see that most of the jobs are killed during the first 30% of their total execution time (the time they take if they are not killed earlier). Then, for the remaining steps, prediction abilities increase after 60th percentile of their duration. Figure 4 shows that the dynamic policy can predict failures early.

Users and system administrators may use policies with different aggressiveness levels. For instance, a user might choose a very aggressive policy, both static and dynamic with a very low threshold, when the project budget is highly limited. On the other hand, a less aggressive policy, e.g., a dynamic policy with a high threshold, above 0.9, can be appropriate for long jobs, where user time is the most expensive factor to consider. Also, such a policy can maximize savings comparing to use of a static policy. A static policy used by system administrators can help eliminating problematic jobs, which may be causing the overload of a system. However, use of dynamic policy can cause dissatisfaction of users, since this policy can unexpectedly terminate their jobs without a known reason.

Also, supervised learning through interaction with a user can help improving the proposed policies. Firstly, users should receive a notification when their jobs are repeatedly killed after re-submissions. A user or a system administrator could label such a problematic job. This action provides a model with additional information for incremental improvement. Also, system administrators can decide to perform supervised learning, to set up the optimal aggressiveness of the policy (threshold).

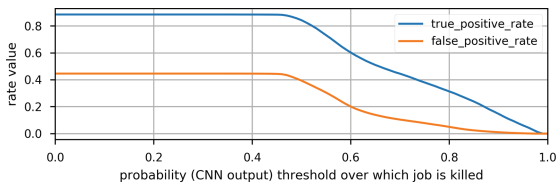


Fig. 2. Plot presents the evaluation of CNN model for different values of prediction probability threshold. The lower is the threshold, the more aggressive is the job terminating policy, greater savings, but we kill more good jobs as a consequence of inaccurate predictions. Total CPU Hours of failed jobs in a set: 11M

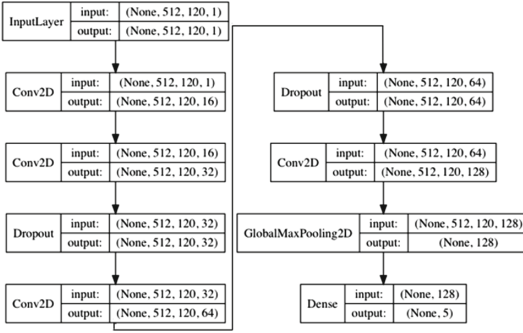


Fig. 3. Graph presenting a trained CNN with layers type and shape of the data

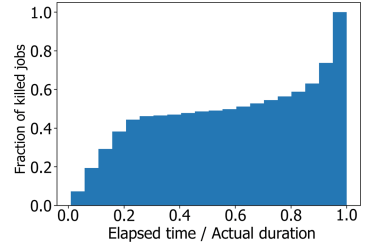


Fig. 4. Cumulative plot presenting the time when the probability of failure exceeds defined threshold 0.82. Number of samples $N = 7300$

6 Conclusions and Future Work

In this paper, we analyzed a dataset containing metrics, topology and job scheduler data for the Mistral supercomputer. We showed important features in a classification and prediction task of a job state. The number of allocated nodes, the state of a previous job submitted by a user, average storage writes are the most important ones. DTs detect specific node types as an important feature due to migration process from the old to the new computing nodes. DTs perform well as a classifier, with a recall nearly 80% and a precision of 93% for failed steps. As a predictor, DTs can point failed steps, using configuration and allocated hardware data exclusively, with a recall of 41% and a precision of 75%. In the case of CNNs, these scores increase to 66% and 85% respectively. This paper shows that one of the biggest influence on the next state of a job in a supercomputer like Mistral lies in the diversity and spatial distribution of allocated nodes, place of a job in a user sequence and number of disk operations.

We evaluated dynamic and static job-killing policies, pointing out possible savings related to the aggressiveness of both policies. For instance, using medium-aggressive approach, we can kill more than 28% of failed jobs. Through CNN predictions, the proposed dynamic policy kills 40% of jobs in the first 20% of their duration. These effects can be improved by utilizing feedback from users and system administrators and adjusting weights of CNN by supervised learning.

As future work, we would like to improve prediction capabilities of the created solution and focus on Lustre FS. Firstly, we can achieve more accurate analysis of final job states by adding OS logs to the analyzed dataset. Also, this would help to build prediction algorithm of final job states, which is not limited by Slurm job state but uses the utility of a job. For instance, the utility can be measured by analyzing users' actions after a job finishes, e.g., a user copied output data, re-run the same code with different parameters, changed the code. Therefore, this approach can differentiate jobs with a non-zero return code from

these which were run unnecessary and these which can provide any utility to a user, e.g., development progress, part of results. Then, we can consider a more complex model which takes into account step sequence for a job. Also, we would like to consider additional input information such as real-time metrics from the data center, e.g., Lustre I/O, overall system load and IB traffic. Finally, we would like to focus more on the deep learning algorithms for prediction of failures and Root Cause Analysis.

Acknowledgments. This research is supported by the BigStorage project (ref. 642963) funded by Marie Skłodowska-Curie ITN for Early Stage Researchers, and it is a part of a doctorate at UPC.

References

1. Barroso, L.A., Clidaras, J., Hölzle, U.: The datacenter as a computer: an introduction to the design of warehouse-scale machines. *Synth. Lect. Comput. Archit.* **8**(3), 1–154 (2013)
2. Bautista-Gomez, L., Gainaru, A., Perarnau, S., et al.: Reducing waste in extreme scale systems through introspective analysis. In: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 212–221 (2016)
3. Casas, M., Bronevetsky, G.: Prediction of the impact of network switch utilization on application performance via active measurement. *Parallel Comput.* **67**(Suppl. C), 38–56 (2017)
4. Clark, A.D., Tellez, L.M., Besse, S., et al.: Dynamic prediction and estimation of intentional failures in HPCs. In: International Conference on Advances in Social Networks Analysis and Mining, pp. 1244–1250 (2016)
5. El-Sayed, N., Schroeder, B.: Reading between the lines of failure logs: understanding how HPC systems fail. In: 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 1–12 (2013)
6. Fu, S., Xu, C.Z.: Exploring event correlation for failure prediction in coalitions of clusters. In: Proceedings of the Conference on Supercomputing, pp. 41:1–41:12 (2007)
7. Gao, J.: Machine learning applications for data center optimization. Google White Paper (2014). <https://research.google.com/pubs/archive/42542.pdf>
8. Gu, J., Zheng, Z., Lan, Z., et al.: Dynamic meta-learning for failure prediction in large-scale systems: a case study. In: 37th International Conference on Parallel Processing (2008)
9. Gupta, S., Patel, T., Engelmann, C., et al.: Failures in large scale systems: long-term measurement, analysis, and implications. In: SC 2017, pp. 1–12 (2017)
10. Jones, M.D., White, J.P., Innus, M., et al.: Workload analysis of blue waters. CoRR abs/1703.00924 (2017). <http://arxiv.org/abs/1703.00924>
11. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. *Nature* **521**(7553), 436–444 (2015)
12. Nakka, N., Agrawal, A., Choudhary, A.: Predicting node failure in high performance computing systems from failure and usage logs. In: 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum, pp. 1557–1566, May 2011

13. Nie, B., Xue, J., Gupta, S., et al.: Characterizing temperature, power, and soft-error behaviors in data center systems: insights, challenges, and opportunities. In: IEEE 25th MASCOTS, pp. 22–31 (2017)
14. Safavian, S.R., Landgrebe, D.: A survey of decision tree classifier methodology. *IEEE Trans. Syst. Man Cybern. Syst.* **21**(3), 660–674 (1991)
15. Sirbu, A., Babaoglu, O.: Towards operator-less data centers through data-driven, predictive, proactive autonomics. *J. Cluster Comput.* **19**(2), 865–878 (2016)
16. Srivastava, N., Hinton, G.E., Krizhevsky, A., et al.: Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **15**(1), 1929–1958 (2014)
17. Tuncer, O., et al.: Diagnosing performance variations in HPC applications using machine learning. In: Kunkel, J.M., Yokota, R., Balaji, P., Keyes, D. (eds.) ISC 2017. LNCS, vol. 10266, pp. 355–373. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58667-0_19
18. Vishwanath, K.V., Nagappan, N.: Characterizing cloud computing hardware reliability. In: Proceedings of the 1st ACM Symposium on Cloud Computing, pp. 193–204. ACM (2010)
19. Wallace, S., Zhou, Z., Vishwanath, V., et al.: Application power profiling on IBM Blue Gene/Q. *Parallel Comput.* **57**, 73–86 (2016)
20. Xie, B., Huang, Y., Chase, J.S., et al.: Predicting output performance of a petascale supercomputer. In: Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing, pp. 181–192 (2017)
21. Yu, L., Zhou, Z., Wallace, S., et al.: Quantitative modeling of power performance tradeoffs on extreme scale systems. *J. Parallel Distrib. Comput.* **84**(Suppl. C), 1–14 (2015)
22. Yuan, Y., Wu, Y., Wang, Q., et al.: Job failures in high performance computing systems: a large-scale empirical study. *Comput. Math. Appl.* **63**(2), 365–377 (2012)
23. Zasadziński, M., Muntés-Mulero, V., Sóle, M., Ludwig, T.: Mistral supercomputer job history analysis (2018). <https://arxiv.org/abs/1801.07624>