CPU overheating characterization in HPC systems: a case study

[1,2]Marc Platini, [1]Thomas Ropars, [2]Benoit Pelletier, [1]Noel De Palma
[1]*Univ. Grenoblet Alpes, {thomas.ropars, noel.depalma}@univ-grenoble-alpes.fr*
[2]*ATOS, Grenoble, {marc.platini, benoit.pelletier}@atos.net*

***Abstract*—With the increase in size of supercomputers, also increases the number of abnormal events. Some of these events might lead to an application failure. Others might simply impact the system efficiency. CPU overheating is one such event that decreases the system efficiency: when a CPU overheats, it reduces its frequency. This paper studies the problem of CPU overheating in supercomputers. In a first part, we analyze data collected over one year on a supercomputer of the top500 list to understand under which conditions CPU overheating occurs. Our analysis show that overheating events are due to some specific applications. In a second part, we evaluate the impact of such overheating events on the performance of MPI applications. Using 6 representative HPC benchmarks, we show that for a majority of the applications, a frequency drop on one CPU impacts the execution time of distributed runs proportionally to the duration and to the extent of the frequency drop.**

***Index Terms*—HPC, CPU overheating**

I. Introduction

Supercomputers grow in size every year. The Top 500 list [1] shows that the number of cores of the largest systems keeps increasing and unfortunately the more complex the systems are, the more abnormal events there will be [13].

Abnormal events in supercomputers that lead to crash failures [15], [17], [25], [29] or corrupted states [12], [28] are widely studied in the literature. On the other hand, a CPU overheating is an event that limits the performance of the system without directly compromising the results of the executions. A CPU overheating occurs when the temperature of a CPU is too high. To prevent hardware damages, an overheating CPU reduces dramatically its frequency, and thus its power consumption, for decreasing its temperature. This frequency drop may lead to a performance drop.

Quantifying the impact of CPU overheating events on the efficiency of supercomputers is not straightforward. High Performance Computing (HPC) applications are complex parallel applications. Their execution time is influenced by many system components in addition to the CPUs: the memory, the network, the Parallel File System, etc. As such, it is difficult to foresee the impact of a frequency drop due to a CPU overheating on the performance of an application.

Our work is a first step towards understanding the impact of CPU overheating on the execution time of applications running on HPC systems. To this end, we aim at answering the two following questions:

1) Under which circumstances, considering the characteristics of running jobs and the induced system load, does a CPU overheating occurs in a HPC system?
2) What is the impact of CPU overheating on HPC applications execution time?

To answer the first question, we analyze one year of data coming from a medium-range supercomputer[1]. Low-level metrics (CPU temperature, disk utilization, network traffic, etc.) as well as logs generated by different software components (job scheduler, operating system kernel, etc.) have been collected. The total amount of collected data is more than 1 TB. Using the job scheduler and the kernel logs, we identify the occurrence of overheating events, the impacted nodes and the impacted jobs. We analyze the characteristics of these jobs and the state of the system when overheating events occur to determine if some conditions may induce a higher probability of overheating.

To answer the second question, we use 6 representative benchmarks run in a distributed setting and we simulate an overheating by manually decreasing the frequency of one CPU. We study the impact on the execution of the applications when we vary the duration and the extend of the frequency drop, as well as the number of nodes the application uses.

The main results of our study are the following:

- We did not observe any correlation between the CPU overheatings and the global load of the system. It seems that CPU overheating occurs more frequently during working hours and working days, but this point cannot be explained by variations in the system load.
- We observed a correlation between the overheating events and specific applications. Indeed there are only a few applications impacted by the overheating and these applications are impacted several times during several runs.
- For a majority of our tested applications, a frequency drop on one CPU leads to a significant global performance drop, proportional to the frequency drop. Additional experiments show that these applications are CPU-bound.

Using the data included in the kernel logs, we can provide an estimation of the total duration during which one CPU has been overheating during one year of operation of the studied system. This duration is at most 431 minutes when ignoring events generated by one defective node (6239 minutes otherwise). Using the job scheduler database to identify the

[1]Ranked 55 in the Top 500 list of June 2018

impacted jobs, and considering that when one CPU operates at low frequency, it slows down the whole application, the 431 minutes of CPU overheating correspond to 519 CPU-hours of execution at a reduced efficiency. It implies that the global impact of CPU overheatings on the system efficiency is limited. However, data from other supercomputers should be analyzed to have a more global picture of the impact of this problem on existing and future large scale supercomputers.

The remainder of this paper is organized as follows. Section II presents the background on failures analysis and on the impact of the CPU frequency on the performance of HPC applications. Section III introduces CPU overheating and describes the way such events are reported in OS logs. Section IV studies correlations between overheating events and different parameters including the characteristics of applications. Section V studies the impact of overheating events on the execution time of HPC applications and Section VI derives from this study the amount wasted resources due to CPU overheating events on the studied system. Finally, we discuss the limits of our analysis in Section VII and draw conclusions in Section VIII.

II. Background

In this section, we first present related work analyzing failures in HPC systems. In a second step, we discuss work studying the impact of reducing the frequency of CPUs on the performance of HPC applications.

A. Analysis of failures in large scale systems

Several studies describe and analyze failures in large scale computing systems using data coming from production systems [14], [15], [17], [25], [27], [29]. Other work focus on the failures related to specific hardware components (e.g., hard drives [21], [24], GPUs [20], or memory [12], [28]).

Failures have been extensively analyzed in these systems because they directly impact the user: a failure (due to hardware or software) may prevent one or several users from getting the expected result from their submitted jobs as the jobs may crash or the results be corrupted. On the other hand, the consequence of a CPU overheating might not be visible to a user as it simply leads to a CPU frequency drop. Still, we think it is important to study this problem as it might reduce the efficiency of the system.

To the best of our knowledge, there is no study focusing on the problem of CPU overheating. But we can notice that some studies point out the link between the temperature of the system and the probability of experiencing a failure [29]. Also some of the analysis we present have already been applied to failures in the past. For instance, it has been shown that for some kinds of failures, the rate varies according to the time of the day or according to the period of the year [17], [29]. Some work also study the correlation between the load of the system and the failure rate [14], [29]. The results in [29] show a positive correlation for some components (e.g., the hard drives) between the load and the failures rate.

B. Impact of the CPU frequency on the performance of HPC applications

When a CPU overheats, its frequency is significantly decreased to help reducing the temperature. The impact of CPU frequency on the performance of HPC applications has already been studied, especially in the context of work that aim at reducing the power consumption [19], [22], [23]. However, only few studies consider large CPU frequency drops [11], [16], [18]. In these works, the impact of the frequency drop is evaluated with applications running on a single node. Their results show that not all applications are impacted in the same way. The performance of some applications might decrease proportionally to the decrease in frequency [11], [16] while applications including I/O operations might be less impacted [16]. The problem we study in this paper is different: we evaluate the impact of a frequency drop of one CPU on the performance of distributed MPI applications running on multiple nodes.

III. Studying CPU overheating based on logs

This section defines the main concepts associated with CPU overheating. It then presents the method we use to estimate CPU overheatings duration based on the Linux kernel logs.

A. Overheating on Intel processors

On Intel processors, a CPU is considered as overheating when its temperature reaches its maximum safe operating temperature. This value is specified by the constructor. When this limit is reached, the CPU reduces its frequency to reduce its temperature and avoid hardware damage. The temperature of the CPU is provided by the DTS [2] (Digital Temperature Sensor). The DTS value represents the distance to the overheating temperature. Thus, the DTS is most often lower than 0.

When the value of the DTS reaches 0, two mechanisms are used to decrease the temperature of the CPU: clock modulation and operating frequency adjustment [3]. Additionaly an interrupt is raised. Operating frequency adjustment implies that the processor transitions to its minimum frequency when CPU overheating is detected. The processor remains at the minimum frequency for at least a predefined minimum amount of time to avoid that the CPU would start overheating again immediatly when back to the normal frequency. Clock modulation can be used to additionally reduce the activity of the CPU.

B. Overheating events data

There are two sources of data that can give us information about CPU overheating events that occurred in a system. The first is the time series of DTS values for each CPU. The second is the operating system logs.

Ideally one would use the data provided by the DTS sensors to identify overheating events. However, to precisely capture the behavior of the processor one would have to store the DTS value of each CPU as frequently as the CPU checks its temperature. This would lead to store huge amount of data. To save storage space on the cluster we are studying, the choice

was made to store one value of the DTS of each CPU every minute. It implies that these data are not accurate enough to precisely evaluate the occurrence and the duration of the overheating events.

Hence the solution we choose to obtain information about overheating events is to analyze the logs generated by the Linux kernel on each compute node. According to the Intel documentation [3], an overheating CPU raises an interrupt that is processed by the Linux kernel. Logs entries are written with information about these events. We discuss this point in more details in the following.

C. Extracting information about CPU overheatings in the Linux kernel logs

When an interrupt due to overheating is raised by the CPU, the Linux kernel generates a log, such as the following one:

```
CPU46: Package temperature above threshold,
  cpu clock throttled (total events = 23193)
```

In addition to the message generated by the kernel, the system logs of the studied cluster provides us with other information including the timestamp associated with the log and the identifier of the node that generated the message.

According to the Linux kernel, there are two interrupts that are associated with overheating events: the package-level interrupt and the core-level interrupt. The package-level interrupt implies that all the cores of the CPU throttle whereas the core-level interrupt implies that only one core throttles. In practice, the two information are redundant: a core-level interrupt implies a package-level interrupt. As such, we only consider package-level interrupts in our study.

There are two information we are interested in regarding CPU overheating: the number of times the CPU overheats and the duration of the overheating events.

1) Counting the overheating interrupts: This information is included in the kernel logs. Each log includes a counter (`total events`) of the number of overheating interrupts observed since the boot of the node. According to the Linux Kernel code [4], the counter is shared between all the cores of the CPU and is increased by 1 for each interrupt on each core. Thus, to get the number of package-level interrupts, we should divide the value of the counter by the number of cores included in the package. We note the total number of package-level interrupts $N_{interrupts}$.

2) Duration of the overheating events: We cannot directly deduce the amount of time a CPU has been overheating from the number of overheating interrupts because the duration associated with one interrupt is not constant. Indeed if the temperature remains above the threshold, the CPU stays in overheating state but no new interruption is raised. Otherwise it comes back to the *normal* state. But we can estimate the minimal duration associated with one interrupt (we call it T_{min}). We consider that the minimal duration of one interrupt is reached when there is the maximum number of interrupts with respect to the delay between two logs. Using this method, we can get a lower bound of the duration of overheating events. More formally, we divide the maximum number of interrupts between two logs by the delay between them. Using this method, we get T_{min} = 33ms.

Moreover, after the kernel has generated a first overheating log, there is a minimum delay of 5 minutes before it generates a new log [4]. As such the occurrence of overheating messages in the logs does not give a good information on the duration of overheating periods but we can use it to get an upper bound of the overheating events.

Using these two information (the number of overheating interrupts and the occurrence of overheating logs), we can provide a lower bound (named T_{low}) and an upper bound (T_{up}) of the time during which a CPU has been overheating ($T_{overheat}$):

$$T_{low} < T_{overheat} < T_{up}$$

Computing T_{low} is straightforward as it boils down to multiplying the number of overheating interrupts $N_{interrupts}$ by the minimum duration of one interrupt, *i.e.*, T_{min}:

$$T_{low} = N_{interrupts} \times T_{min}$$

Computing the value of T_{up} is a bit more complex. Indeed, it can be the case that multiple overheating interrupts occur in the time window of 5 minutes between two consecutive overheating logs as illustrated by Figure 1. This figure presents an execution with log messages and interrupts generated by the overheating of a CPU. A first log is generated at 4:00pm and a second one at 4:05pm. According to the events counter, there has been 3 overheating interrupts between the two logs. The fact that there has been three overheating interrupts implies that during the time period, the processor came back into the *normal* state at least two times (recall that an interrupt is raised when the processor's state changes from *normal* to *overheating*). Hence, we can deduce that in this specific scenario, the value of T_{up} is 5 minutes minus at least two intervals of 33 ms in normal state: $T_{up} = 5\ min - 2 \times T_{min}$. We can generalize this computation with the following formula for one 5-minute time frame following one overheating log:

$$T_{up} = 5\ min - (N_{inter-frame} - 1) \times T_{min}$$

where $N_{inter-frame}$ is the number of interrupts in the time frame.

On the example of Figure 1, it implies that the bounds we can compute on the value of $T_{overheating}$ are:

$$3 \times 33\ ms < T_{overheating} < 5min - 66\ ms$$

Note that in this example, the total duration of the overheating event is 11 minimum time steps ($T_{overheating} = 11 \times T_{min} = 363\ ms$). Hence, the evaluation we obtain of the approximation of $T_{overheating}$ is not very precise.

IV. Analysis of overheating events

In this section, we analyze the data coming for a production system to try to understand under which conditions CPU overheating events occur. We start by presenting the system considered in this study. Then we analyze the logs to compute an estimation of the total CPU overheating time

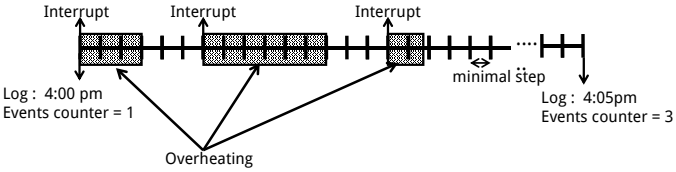


Fig. 1: Relation between overheating interrupts and overheating duration

Data	T_{low}	T_{up}
All data	973 minutes	6239 minutes
All data excepted the defective node	68 minutes	431 minutes

TABLE I: Overheating duration

on this system during one year. We continue by analyzing the distribution of the events according to the time and we study correlations between the occurrence of overheating events and the characteristics of the workload.

A. Presentation of the studied system

To study CPU overheating, we use data coming from the DKRZ (Deutsches Klimarechenzentrum) [5] cluster. According the Top500 ranking [1], this supercomputer is the 55th more powerful supercomputer in the world. It has more than 3000 nodes. The cluster includes two types of nodes: 1404 nodes using 2 12-cores Intel Xeon E5-2680 v3, 64 GB of RAM and 1420 nodes using 2 18-cores Intel Xeon E5-2695 v4, 64 GB of RAM. The peak performance is 3.14 PetaFLOPS. The nodes are cooled by a liquid cooling system.

All nodes are connected through FDR InfiniBand fabric with three Mellanox SX6536 director switches using a fat tree topology. The system features a Lustre parallel file system with a capacity of 54 PB. The nodes are manufactured by ATOS [6].

For our analysis, we exploit 2 sources of data: the system logs and the database of the job scheduler. Our study covers 11 months of data, from July 2017 to May 2018, which is the period during which we have data for the 2 sources. Note that we do not have access to the data about DTS of CPU during that period at the time of writting this paper.

Regarding the CPU overheating events during this time period, there is one particularity: There are a huge number of overheating events during July 2017 and August 2017 due to one defective node. We decided to include these events in the initial evaluation of the duration of overheating events. But these events are discarded for the rest of the study.

Without considering the events due to the defective node, the logs include 169 overheating messages that correspond to a total of 46997 package-level CPU interrupts. The 169 overheating messages generated over the studied period are distributed over 28 nodes; 144 of these overheating messages impact a running application. The 25 remaining messages were all generated on the same day and are probably due to an unplanned maintenance or an exceptional event.

B. Estimation of the overheating duration

Table I shows the results provided by the technique presented on Section III when applied to our data. The first row shows the overheating duration estimated for all the data. The second row shows the overheating duration estimated when the events generated by the defective node are not taken into account. The gap between the lower bound of $T_{overheating}$ (68 minutes) and its upper bound (431 minutes) is large due to the limited accuracy of the method we use to compute these numbers.

C. Analysis of the occurrences of overheating events

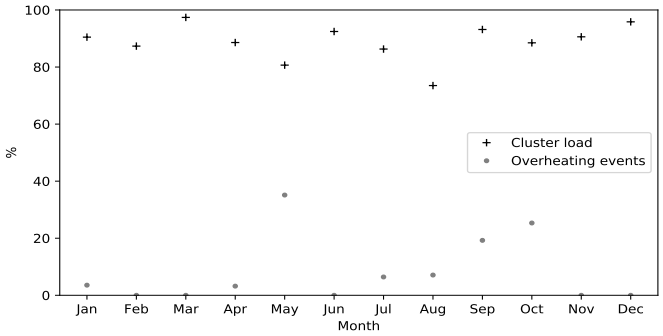
If we want to draw a picture of CPU overheating, we first have to understand if they are random events, if they are caused by exceptional events (failure of the cooling system for example), or if the probability of occurrence of such events depends on the characteristics of the workload (high load pick, type of applications, etc). In this section, we study the distribution of overheating events with respect to time, and we compare it to the load of the system. We also analyze the distribution with respect to the nodes and possible correlations with some characteristics of the running applications.

1) Time distribution of the overheating events: Figure 2 shows the load as a percentage of the available ressources and the percentage of overheating events across the months of the year (2a), the days of the week (2b), and the hours of the day (2c) for the data excluding the defective node. We recall that there is no data concerning the overheating events in June. We compute the load as the number of CPUs used divided by the number of CPUs available during one time step (1 minute for our case). These time steps are aggregated per hour, per day and per month.

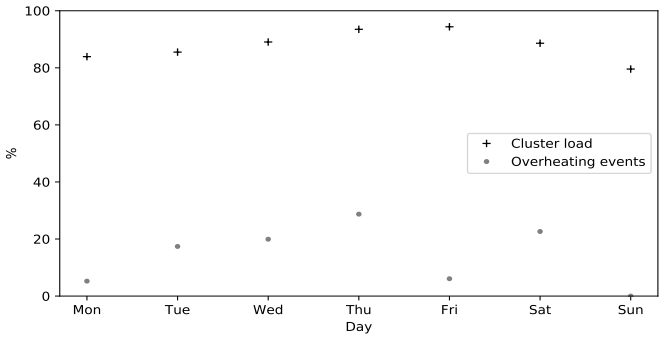
The occurrence of overheating events seems to be related to the activity on the cluster since Figure 2b shows a higher number of overheating events during working days compared to Sunday, and Figure 2b shows a higher number of overheating events during working hours (from 9am to 5pm). To try to better understand this point, we analyze below the load of the cluster.

2) Distribution with respect to the usage of the cluster: Figure 2 shows that the usage of the cluster is always greater than 80% except in August. As such, we cannot conclude about a strong correlation between the global load of the system and the occurrence of CPU overheating. Still we can notice that the global load of the system is lower on Sunday and during the night hours, and that the number of overheating events that occurred during that period is also low.

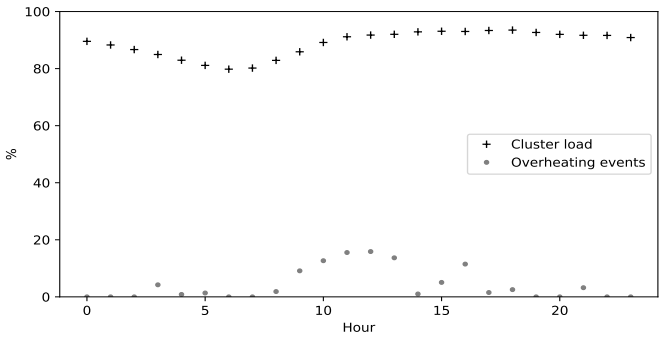
Figure 3 shows the distribution of the load of the cluster ("Global load") during the analyzed period aggregated over 10 minutes time windows, and the distribution of the load before the overheating events (resp. 10 mins, 1 hour and 1 day before the overheating events). On this figure, a load of 0.8 means that during a 10-minutes time window, 80% of the CPUs of the cluster were used by some jobs. We can see that the overheating events occurs when the load is lower than the



(a) Load and percentage of overheating events according to the month



(b) Load and percentage of overheating events according to the day



(c) Load and percentage of overheating events according to the hour of the day

Fig. 2: Load and number of overheating events distribution

median global load (0.95). Thus, we can conclude that the overheating events are not correlated with high load picks.

3) Distribution of the overheating events on the nodes of the cluster: The 169 overheating logs observed during the studied period are spread over 28 nodes, and none of these nodes have generated a significantly higher number of events compared to the rest of them. We compared the activity of these nodes to the one of the other nodes of the system, and we did not notice anything specific: the load of these nodes is similar to the one of the other nodes, and the applications run have similar characteristics in terms of size and duration.

4) Distribution with respect to the running applications: First, we have to make the difference between a job and an application. An application is a unique source code that

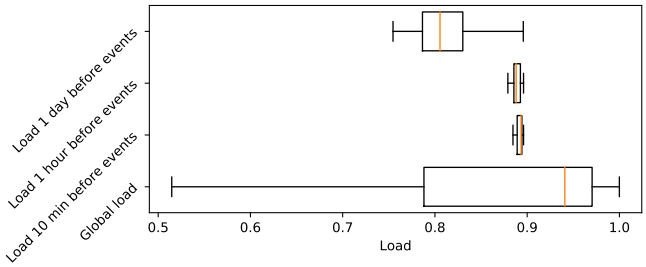


Fig. 3: Distribution of the global load and of the load before an overheating event

computes a solution to a problem. This application can be executed several times. A job is a single execution of an application. To identify unique applications, we use the logs coming from the jobs scheduler. The logs contain the name for each job. We keep only the alphabetical characters of this name. After this step, we consider two jobs are running the same application if they have the same *cleaned* name. This means that if one job is launched with the name "ocean_run_1" and another one with the name "ocean_run_2", we consider that it is two runs of the same application.

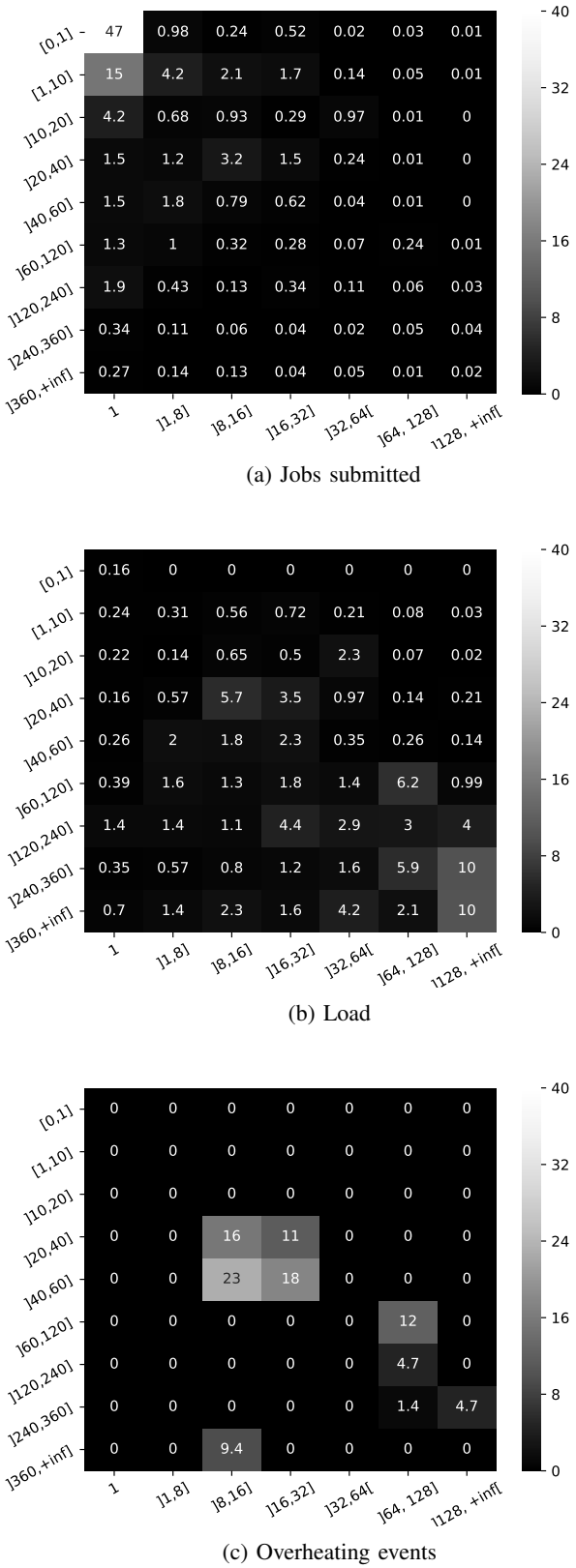
In the following, we study the correlation between the overheating events and the characteristics of the applications, taking into account the number of runs for each application. There are more than 5 billions of analyzed jobs. For analyzing the correlation between the overheating events and the jobs, we build 3 tables represented as heat-maps in Figure 4. On each of the figures, the X axis represents the size of the jobs in number of nodes, the Y axis represents the duration of the jobs in minutes. Figure 4a shows the distribution of the submitted jobs (as a percentage of the total number of submitted jobs). Figure 4b shows the percentage of the load on the system generated by each class of jobs. Figure 4c shows the percentage of the overheating events that affected each class of jobs. The first characteristic to be mentioned is the most submitted job uses one node and is shorter than 1 minute (47% of the submitted jobs) but it only corresponds to 0.16% of the load of the system. On the other hand, there are only few *large* jobs (less than 1.41% of submitted jobs have a duration greater than 60 minutes and use more than 16 nodes) but they correspond 61.29% of the load.

Overheating events mainly occur (69%) for jobs using between 8 and 32 nodes and running during 20 to 60 minutes. We observe that there are no jobs using less than these resources that have been impacted by an overheating. It means that, in our case, bellow this threshold of used resources, an application has a very low probability to be impacted by an overheating event. To understand why over this threshold, the number of overheating events is not proportional to the induced load, a more detailed analysis is required. Figure 5 presents the percentage of the distinct applications and the percentage of distinct applications impacted by an overheating event depending on the size of the jobs. Figure 6 presents the

same graph but according to the duration of the jobs. These figures only consider jobs using more than 8 nodes or jobs running during more than 20 minutes. The percentage for each class of distinct applications is computed using the number of distinct applications for one class divided by the total number of distinct applications. For example, 52% of the distinct applications between 8 and 16 nodes means that over a total of 4000 distinct applications, there are 2080 applications using between 8 and 16 nodes. There are 18 applications impacted by the overheating events. Table II provides details about the applications: the number of the jobs of the application impacted by the overheating, the total number of jobs of the application, the number of overheating events occurring during the jobs of the application, and the percentage of the jobs of the application impacted by the overheating events. We also analyze the duration of the execution time of each application. We observe too large variations in the execution time of different jobs of the same application to be able to quantify the performance decrease generated by an overheating event. Table II shows that the applications running on a large number of nodes do not have a higher probability of being impacted by an overheating, which tends to show that a CPU overheating is not a random event. Figure 5 and Figure 6 show a weak correlation between the number of distinct applications started and the number of distinctd applications impacted by the overheating events. For example, according to Figure 5, jobs using between 8 and 16 nodes are executing 52% of the unique applications started with more than 8 nodes. Also, 43% of the distinct applications impacted by an overheating event were run on jobs of this size. This observation makes us think that overheating events are due to some specific applications, and thus, the more distinct applications are run, the higher is the probability to run an application that may generate an overheating event. Moreover, according to Table II, for the majority of the applications that suffered from a CPU overheating, only a few jobs are impacted (less than 1% of the jobs for 11 of the 18 applications). We can see 3 outliers in Figure 5 and Figure 6: there is no application using between 32 and 64 nodes impacted by the overheating, and there are very few applications running for 60 to 120 mins and 120 to 240 mins impacted by overheating events. As mentioned before, there are very few distinct applications (18) impacted by the overheating, thus the probability is very low to start this type of applications. This may explain these outliers. Even if we identified specific applications that generate CPU overheating events, only a few of their runs is impacted by an overheating event, thus it is complex to determine the root causes of these. A more detailed analysis of the applications, including an analysis of their source code, would be required to better identify the reasons that make some applications more likely to generate CPU overheating events.

V. Relation between overheating and execution time

In this section, we analyze the impact of an overheating CPU on the performance of MPI applications. As already



(a) Jobs submitted

(b) Load

(c) Overheating events

Fig. 4: Distribution of the submitted jobs, the load, and the overheating events with respect to the number of nodes and the execution time of the jobs

ID	Nb of jobs	Nb of jobs impacted	Number of over-heating events	Average duration of 1 job (minutes)	Average nb of nodes used
1	8	1 (12.5%)	1	34	16
2	114	1 (0.8%)	1	29	20
3	180	1 (0.5%)	7	29	16
4	200	1 (0.5%)	11	45	14
5	230	1 (0.4%)	21	69	12
6	198	1 (0.5%)	1	52	10
7	20	1 (5.0%)	14	453	12
8	954	1 (0.1%)	11	49	20
9	37	2 (5.4%)	18	62	97
10	158	1 (0.6%)	5	32	16
11	398	2 (0.5%)	10	40	20
12	14	3 (21.4%)	9	245	108
13	13	1 (7.6%)	7	47	16
14	508	2 (0.3%)	3	48	20
15	74	3 (4.0%)	7	316	241
16	36	1 (2.7%)	1	30	20
17	500	3 (0.6%)	12	48	20
18	664	1 (0.1%)	5	30	20

TABLE II: Applications impacted by the overheating events

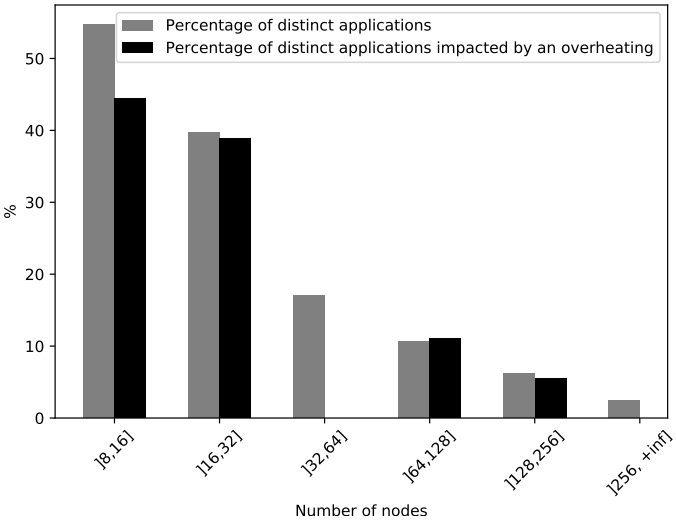


Fig. 5: Distribution of the distinct applications and distribution of the distinct applications impacted by overheating events with respect to the number of used nodes

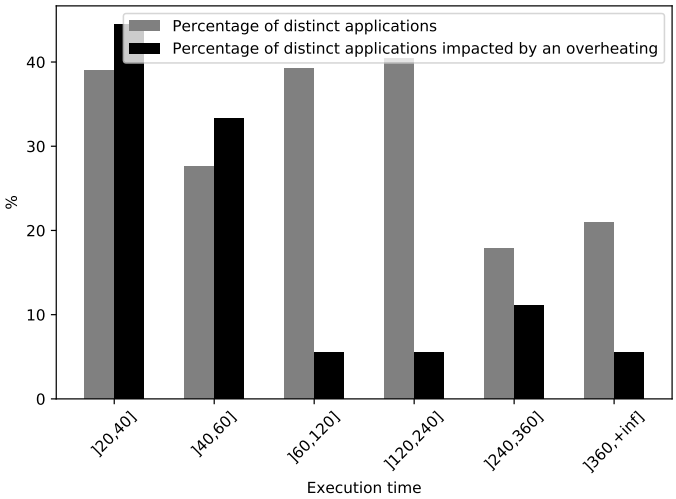


Fig. 6: Distribution of the distinct applications and distribution of the distinct applications impacted by overheating evetns with respect to the job execution time

mentioned, the consequence of a CPU overheating on Intel processors is a frequency drop. The percentage of the frequency drop depends on the CPU and on the situation. As discussed in Section II, most existing work on the impact of the CPU frequency on performance consider the case of applications executing on a single node, or cases where all nodes run at the same low frequency. We study the case of distributed MPI applications running on several nodes where the frequency of a single CPU is decreased.

In a first step, we evaluate the performance of a representative set of MPI HPC applications for different frequency drops on one node. In a second step, we try to explain the obtained results by analyzing the memory accesses of these applications. We start this section by describing the setup of our experiments.

A. Setup description

To test the impact of a frequency drop on one node on the performance of distributed MPI applications, a cluster featuring 120 nodes is used. Each node has an Intel Xeon X3440 CPU with 4 cores and 16 GB of RAM. They are connected through InfiniBand 20 Gbps. In a second step, to be able to monitor the memory accesses of the application, we use a single node equipped with 2 Intel Xeon E5-2630 v3 processors with 8 cores per CPU and 128 GB of RAM. Note that we use an in-house cluster for these experiments because we need to have sufficient rights on the nodes to be able to manually change the CPU frequency.

All experiments are run with Debian 9 as operating system and are run with the 4.9 version of the Linux kernel. To better control the frequency of the processors, turbo mode and hyper-threading are disabled. Governor mode is set to ”maximum performance”. MPICH v3.2.1 is used as MPI library.

To evaluate the impact of a frequency drop on the performance of HPC applications, we consider a representative set of 6 applications included in the NERSC8 Trinity Benchmark Suite [7]: AMG, coMD, GTC, MILC, miniFE, and miniGhost. Each application is representative of a type of scientific problem. For example coMD focuses on typical molecular dynamics algorithms whereas MILC focuses on lattice computation. Table III presents the parameters we use for each application. Only the parameters that are different from the default parameters used in the Trinity benchmark suite are specified in the table. These parameters have been selected so that the execution time would be in the order of hundreds of seconds. We run tests with different numbers of ranks (see Table III) according to the configurations accepted by each application. For each configuration, the parameters are adapted to keep the local problem size on each rank constant

(weak scaling). Note that Table III also includes the profile (memory footprint and the time spent in MPI communication) of each application as measured by the IPM [8] tool for the configuration with 128 MPI ranks. We can note that the IPM profile of each application is different and this leads to a different type of behavior for each application. Thus, we expect the impact of overheating to be different depending on the application.

In our experiments, we reduce the frequency of one processor using the `cpufreq`[2] package. We test different frequency drops and different durations for theses drops to better understand the impact on applications. The Intel Xeon X3440 processor can be set to different frequencies in the range [2.53GHz, 1.2GHz] using `cpufreq`. For the following tests we select 3 frequencies to compare with the performance when using the maximum frequency (which is 2.53 GHz): 2.27 GHz (90% of the maximum frequency), 1.87 GHz (74% of the maximum frequency) and 1.2 GHz (minimum frequency – 47% of the maximum frequency). We also test several durations for the frequency drop to ensure that our observations can be generalized to different drop durations. Namely, we test with a drop duration of 30s, 90s and 150s.

In our tests, we randomly select one of the node where the application is running and reduce the frequency of the processor for the predefined duration. The frequency drop is always applied after 100s of execution to avoid applying it to the initialization phase of the application. Note also that the policy that is selected for placing the MPI ranks is to first used all the cores on one node before stating using another node. This implies that on our cluster, each frequency drop impacts 4 MPI ranks. All results presented in following are average over 5 runs for each configuration. A different node is randomly selected to apply the frequency drop in each run.

B. Results

The results of our experiments are given in Table IV. We choose to aggregate the results for different number of ranks in each application as we could not observe significant differences in the results for different number of ranks. For each application, we present the increase of the application execution time as a percentage of the duration of the frequency drop. For instance, the overhead of 27% for GTC at 1.87GHz implies that the execution time of the application increased by 27s due to the frequency drop lasting 100s.

We can observe 3 different behaviors among our 6 applications: i) miniFE is insensitive to the frequency drops; ii) miniGhost and MILC are sensitive only to large frequency drops; iii) coMD, GTC, and AMG are very sensitive to frequency drops. In the case of these 3 applications the performance decrease is directly proportional to the range of the frequency drops, that occurs on one node.

These results show that for some applications, an overheating event affecting one CPU can have a significant impact on the overall performance of the application even if the application is distributed over a large number of nodes. To better understand why not all applications are impacted in the same way by a frequency drop, we complement our study in the next section by measuring the memory accesses of the applications.

[2]https://mirrors.edge.kernel.org/pub/linux/utils/kernel/cpufreq/cpufreq-set.html

C. Impact of the CPU frequency on memory accesses

We conduct an experiment to trace the memory accesses of the different applications when we decrease the frequency of the processor. We use Intel PCM monitoring tool [9] to extract from hardware counters information about memory accesses. PCM allows us to obtain the memory accesses in GB/s at 1Hz.

For this experiment, we use the node equipped with Intel Xeon E5-2630 processors and run tests with 3 CPU frequencies: 2.4 GHz (the maximum frequency of the CPU), 1.9 GHz, and 1.3 GHz. For each application, we start the application with the processor running at its maximum frequency and we reduce the frequency 4 minutes after the beginning of the run. During the whole run, we monitor the memory accesses using the PCM tool. Note that according to Intel Documentation [10], the maximum bandwidth is 42.6 GB/s per CPU (thus 85.2 GB/s for our two sockets nodes). The applications run using 16 ranks (one per CPU) and we update the parameters compared to the previous experiment to have a run for a duration greater than 10 mins.

Figure 7 presents the memory accesses profile of the applications for the different runs and Table V presents the increase of the application execution time as a percentage of the duration of the frequency drop. For MiniFE, at the maximum frequency and at 1.9 GHz, the limit of the memory bandwidth is reached, hence the performance is limited by the performance of the memory. When the CPU frequency is set to 1.3 GHz, the intensity of the memory accesses starts decreasing and the performance too.

In the case of AMG, coMD, GTC, MILC and miniGhost the maximum memory bandwidth is not reached when the CPU runs at its maximum performance. It implies that contrary to MiniFE, these applications are not memory-bound. As a consequence, decreasing the CPU frequency as a direct impact on the number of memory accesses per-second and on the performance of the applications.

These results show that, as one can expect, a CPU frequency drop has impact on the performance of application only if it is CPU-bound.

VI. CPU TIME LOST DUE TO OVERHEATING

Now that we have an overview of how HPC applications can be impacted by a frequency drop due to a CPU overheating, we can compute an estimation of the time lost due to overheating in the case of the DKRZ supercomputer for the time period studied in Section IV. As shown in the previous experiment, a frequency drop of one CPU can impact the overall performances of applications running on multiple nodes. For this computation, we consider a worst-case scenario where all applications experiencing an overheating are CPU-bound.

TABLE III: Configuration and profile of the tested applications

Application	Number of MPI ranks	MPI communication (128 ranks)	Memory footprint (128 ranks)	Execution time (128 ranks)	Parameters (128 ranks)
AMG	128-256-448	8.6%	0.52 GB/rank	347s	solver: 1 / size: (160,160,160)
coMD	128-256-480	4.28%	0.8 GB/rank	294s	size: (250,250,200)
GTC	128-256-480	8.91%	2.8 GB/rank	605s	steps: 30 / npartdom: 2 / micell: 200 / mecell: 200
MILC	128-256	11.23%	0.09 GB/rank	280s	size: (64,32,32,48)
miniFE	128-256-480	6.56%	0.49 GB/rank	362s	steps: 1600 / size: (600,600,600)
miniGhost	128-256-448	9.02%	2.65 GB/rank	453s	steps: 65 / size: (200,200,200)

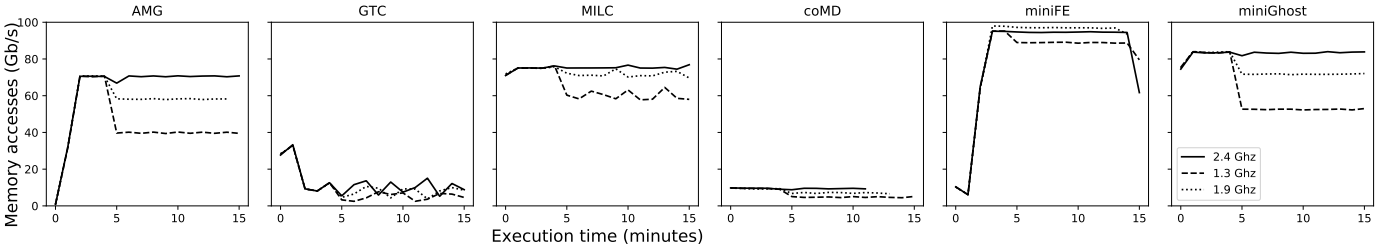


Fig. 7: Memory accesses profiles

TABLE IV: Impact of a CPU frequency decrease on the execution time of distributed application

Application	2.2 GHz	1.87GHz	1.3GHz
AMG	12%	24%	49%
coMD	11%	26%	53%
GTC	12%	27%	55%
MILC	0%	8%	19%
miniFE	0%	0%	0%
miniGhost	0%	0%	25%

TABLE V: Impact of a CPU frequency decrease on the execution time of single node application

Application	1.9GHz	1.3GHz
AMG	21%	43%
coMD	20%	50%
GTC	24%	50%
MILC	6%	25%
miniFE	0%	6%
miniGhost	15%	36%

To compute the total number of CPU-hours during which jobs have been running at a reduced efficiency, we take into account two things: the duration of overheating events and the number of nodes used by the jobs that have experienced a CPU overheating. For the duration, we use the value T_{up} as defined in Section III. We get the size of the impacted jobs in the job scheduler database. The final result is 519 CPU-hours of runs at a reduced efficiency.

Using this number, we can then compute the total amount of wasted CPU hours. As described in Section III, two mechanisms are used by Intel processors to deal with overheating: clock modulation and operating frequency adjustment. For this computation, we assume that only frequency adjustment is activated and that it sets the frequency to the minimum frequency of the processor. Here we reuse the value we observed in Section V, that is, the minimum frequency corresponds to 47% of the maximum. In this case, the upper bound on the CPU-hours lost due to overheating would be 275 hours.

VII. Discussion

The work presented in this paper is a first effort to understand and quantify the CPU overheating problem in HPC systems. Several points still need to be studied to better understand the root causes of CPU overheating. First, we should study the applications that suffer from overheating in more details. Our study shows that overheating events are mostly due to some specific applications. It would be interesting to study in more details the characteristics of these applications (programming model, libraries used, type of computation executed, etc.). Second, we should try to understand why only some runs of an application lead to a CPU overheating. From this point of view, it might be the case that the architecture of the supercomputer and of the cooling system play an important role. For instance, we should study the variations of the temperature in the system to see if some nodes are more prone to temperature increases (*e.g*, nodes at the top of cabinets).

The results obtained in Section VI regarding the total amount of CPU-Hours lost on the DKRZ cluster during one year, shows that on this system, CPU overheating is not a major problem for the efficiency of the system. However, it is difficult to generalize this conclusion. Indeed, we can notice that the size of most jobs on the DKRZ cluster is much smaller that the typical job size on other comparable systems. On the DKRZ cluster, a majority of the executed jobs uses less than 32 nodes. On the other hand, the data presented in [26] and [30] show that on other systems, the majority of jobs uses more than 128 nodes (HECToR and MareNostrum), more than 512 nodes (Jugene and Jubl), or even more than 1024 nodes (K

computer). This implies that the cost of CPU overheatings in these systems would probably be much larger on average.

VIII. Conclusion and future work

This paper is an intial step towards understanding the problem of CPU overheating in HPC systems. It analyses the overheating events that occurred during one year of operation on a mid-range supercomputer. The results show that CPU overheating events are not correlated to the global load of the system but are due to some specific applications.

The direct consequence of a CPU overheating is a large decrease of the frequency of the CPU. Using a representative set of 6 HPC benchmarks, we showed that the impact of such a decrease on the performance of the applications can be significant for many applications. For CPU-bound applications, the performance decrease is directly proportional to the value and the duration of the frequency drop even if only one CPU is impacted and the application runs on a large number of nodes. Thus, even if the impact of CPU overheating on the studied cluster is limited, we can expect it to be much larger on extreme scale supercomputers.

As future work, we plan to study solutions to predict the occurence of CPU overheating events by monitoring the CPU temperature. Being able to predict such events could allow us to take actions to prevent them from occuring.

IX. Acknowledgment

We would also like to thank DKRZ (Deutsches Klimarechenzentrum) for giving us access to their data and helping us analyzing them. The work presented in this paper has been partially founded by the ITEA PAPUD project and the Paris region SYSTEMATIC hub, as well as by the national project PIA FSN HYDDA. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

References

[1] https://www.top500.org/. Accessed: 2018-08-24.

[2] https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/cpu-monitoring-dts-peci-paper.pdf. Accessed: 2018-08-24.

[3] https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/xeon-e5-1600-2600-vol-1-datasheet.pdf. Accessed: 2018-08-24.

[4] https://elixir.bootlin.com/linux/latest/source/arch/x86/kernel/cpu/mcheck/therm_throt.c. Accessed: 2018-08-24.

[5] https://www.dkrz.de/. Accessed: 2018-08-24.

[6] https://atos.net/en/. Accessed: 2018-08-24.

[7] http://www.nersc.gov/users/computational-systems/cori/nersc-8-procurement/trinity-nersc-8-rfp/nersc-8-trinity-benchmarks/. Accessed: 2018-08-24.

[8] http://ipm-hpc.sourceforge.net/. Accessed: 2018-08-24.

[9] https://github.com/opcm/pcm. Accessed: 2018-08-24.

[10] https://ark.intel.com/fr/products/64593/Intel-Xeon-Processor-E5-2630-15M-Cache-2_30-GHz-7_20-GTs-Intel-QPI. Accessed: 2018-08-24.

[11] Austin, B., and Wright, N. J. Measurement and interpretation of micro-benchmark and application energy use on the cray xc30. In *Energy Efficient Supercomputing Workshop (E2SC)* (2014), pp. 51–59.

[12] Bautista-Gomez, L., Zyulkyarov, F., Unsal, O., and McIntosh-Smith, S. Unprotected computing: a large-scale study of dram raw error rate on a supercomputer. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC16)* (2016), IEEE, pp. 645–655.

[13] Cappello, F., Geist, A., Gropp, W., Kale, S., Kramer, B., and Snir, M. Toward Exascale Resilience: 2014 Update. *Supercomputing Frontiers and Innovations 1*, 1 (2014), 24.

[14] El-Sayed, N., and 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)* (2013), IEEE, pp. 1–12.

[15] Gainaru, A., Cappello, F., Snir, M., and Kramer, W. Fault prediction under the microscope: A closer look into HPC systems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (2012), p. 77.

[16] Ge, R., Vogt, R., Majumder, J., Alam, A., Burtscher, M., and Zong, Z. Effects of dynamic voltage and frequency scaling on a k20 gpu. In *42nd International Conference on Parallel Processing (ICPP)* (2013), IEEE, pp. 826–833.

[17] Gupta, S., Patel, T., Engelmann, C., and Tiwari, D. Failures in large scale systems: long-term measurement, analysis, and implications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2017), ACM, p. 44.

[18] Koutsikos, N. Investigating power efficiency and co-location effects on heterogeneous hpc architectures. Master's thesis, 2013.

[19] Marathe, A., Bailey, P. E., Lowenthal, D. K., Rountree, B., Schulz, M., and de Supinski, B. R. A run-time system for power-constrained hpc applications. In *International conference on high performance computing* (2015), Springer, pp. 394–408.

[20] Nie, B., Xue, J., Gupta, S., Engelmann, C., Smirni, E., and Tiwari, D. Characterizing temperature, power, and soft-error behaviors in data center systems: Insights, challenges, and opportunities. In *IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)* (2017), IEEE, pp. 22–31.

[21] Pinheiro, E., Weber, W.-D., and Barroso, L. A. Failure trends in a large disk drive population. In *FAST* (2007), vol. 7, pp. 17–23.

[22] Rountree, B., Ahn, D. H., De Supinski, B. R., Lowenthal, D. K., and Schulz, M. Beyond dvfs: A first look at performance under a hardware-enforced power bound. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)* (2012), IEEE, pp. 947–953.

[23] Rountree, B., Lownenthal, D. K., De Supinski, B. R., Schulz, M., Freeh, V. W., and Bletsch, T. Adagio: making dvs practical for complex hpc applications. In *Proceedings of the 23rd international conference on Supercomputing* (2009), ACM, pp. 460–469.

[24] Schroeder, B., and Gibson, G. A. Disk failures in the real world: What does an mttf of 1, 000, 000 hours mean to you? In *FAST* (2007), vol. 7, pp. 1–16.

[25] Schroeder, B., and Gibson, G. A. Understanding failures in petascale computers. In *Journal of Physics: Conference Series* (2007), vol. 78, IOP Publishing, p. 012022.

[26] Simpson, A. D., Bull, M., and Hill, J. Identification and categorisation of applications and initial benchmarks suite. *PRACE-PP Public Deliverables, http://www. prace-ri. eu/Public-Deliverables* (2008).

[27] Snir, M., Wisniewski, R. W., Abraham, J. A., Adve, S. V., Bagchi, S., Balaji, P., Belak, J., Bose, P., Cappello, F., Carlson, B., et al. Addressing failures in exascale computing. *The International Journal of High Performance Computing Applications 28*, 2 (2014), 129–173.

[28] Sridharan, V., DeBardeleben, N., Blanchard, S., Ferreira, K. B., Stearley, J., Shalf, J., and Gurumurthi, S. Memory errors in modern systems: The good, the bad, and the ugly. In *ACM SIGPLAN Notices* (2015), vol. 50, ACM, pp. 297–310.

[29] Wang, G., Zhang, L., and Xu, W. What can we learn from four years of data center hardware failures? In *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2017), IEEE, pp. 25–36.

[30] Yamamoto, K., Uno, A., Murai, H., Tsukamoto, T., Shoji, F., Matsui, S., Sekizawa, R., Sueyasu, F., Uchiyama, H., Okamoto, M., et al. The k computer operations: experiences and statistics. *Procedia Computer Science 29* (2014), 576–585.