Towards Communication Profile, Topology and Node Failure aware Process Placement

Ioannis Vardas, Manolis Ploumidis, and Manolis Marazakis

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

***Abstract*—HPC systems need to keep growing in size to meet the ever-increasing demand for high levels of capability and capacity, often in tight time windows for urgent computation. However, increasing the size, complexity and heterogeneity of HPC systems also increases the risk and impact of system failures, that result in resource waste and aborted jobs.**

A major contributor to job completion time is the cost of inter-process communication. To address performance and energy efficiency, several prior studies have targeted improvements of communication locality. To meet this goal, they derive a mapping of MPI processes to system nodes in a way that reduces communication cost. However, such approaches disregard the effect of system failures.

In this work, we propose a resource allocation approach for MPI jobs, considering both high performance and error resilience. Our approach, named Communication Profile, Topology and node Failure (*CPTF*), takes into account the application's communication profile, system topology and node failure probability for assigning job processes to nodes. We evaluate variants of *CPTF* through simulations of two MPI applications, one with a regular communication pattern (LAMMPS) and one with an irregular one (NPB-DT). In both cases, the variant of *CPTF* that strives to avoid failure-prone nodes and communication paths achieves lower time to complete job batches when compared to the default resource allocation policy of Slurm. It also exhibits the lowest ratio of aborted jobs. The average improvement in batch completion time is 67% for NPB-DT and 34% for LAMMPS.

***Index Terms*—Failure aware resource allocation, Resilience, MPI parallel jobs**

I. Introduction

There is a wide range of different approaches for improving the performance of MPI [1] parallel applications including topology or machine-aware collective primitives [2], hardware assistance for certain MPI primitives [3] and point-to-point primitives, tuned for RDMA-capable networks [4]. *Process placement* has received significant attention, focusing on the assignment of system resources to a job's processes with the goal to minimize communication cost while also achieving a fair load balance among system nodes [5]–[10]. The main motivation behind this approach is to assign processes with a *heavy* communication profile at *nearby* nodes so that communication cost is minimized.

Apart from managing resources and energy efficiency, reliability has been recognized as major challenge towards exascale systems. The emerging complex HPC systems are expected to exhibit frequent failures [11]–[13]. Moreover, increased system complexity is expected to lead to more complex and thus more error-prone software [12], [14]. Several studies have outlined the effect of various system failures on system resource utilization. Authors in [15] report that in a large scale HPC system, 20% or more of the computing resources are wasted due to failures and recoveries. For one of Google's multipurpose clusters, it was found that a large fraction of time is spent for jobs that do not complete successfully [14]. In [16] authors show that system related errors cause an application to fail once every 15 minutes. What is more, failed applications, although few in number, account for approximately 9% of total production hours. Authors in [13] examine node failure rate in the dataset collected during 1995–2005 at LANL. The number of failures per year per system can be as high as 1100 meaning that an application requiring the full cluster is expected to fail more than two times per day. For mitigating the impact of failures, failure awareness has been integrated in several different approaches including checkpointing [17]–[19], scheduling methods [20], [21] and methods for resource allocation and resource management [22]–[24].

In this work, we propose a resource allocation approach for MPI jobs, considering both high performance and error resilience. Our approach, named Communication Profile, Topology and node Failure (*CPTF*) aware process placement, takes into account the following input: (a) Communication Profile of the job; (b) Topology description; and (c) Node Failures. *CPTF* assumes that an initial training run of each MPI job has been completed, allowing the extraction of that job's communication profile. It also assumes that the topology graph of the parallel machine is available. We consider node failures in post-processing the topology graph and derive three variants of *CPTF*. The failure oblivious one ignores failure-prone nodes, while the pessimistic one tries to exclude failure prone nodes and communication paths from being used. Finally, the weighted variant assigns greater weights to links involving failure prone nodes. Process placement is derived by solving the corresponding topology mapping problem, using the Scotch graph mapping library [25], [26].

The process placement approaches discussed so far ([5]–[10]), take into account a description of the HPC system topology along with a characterization of the MPI jobs communication profile, before assigning processing elements to that job's processes. The difference of the proposed approach is

that it also takes into account the probability of node failures. As far as resource allocation techniques are concerned, we assume native execution of jobs, which means that they are not contained within VMs. Our approach aims at avoiding paths with low reliability instead of allocating redundant nodes for VM and job migration. In contrast to approaches that target a reliability aware schedule for task-based applications, we consider MPI jobs with processes coexisting for the duration of the execution, with no ordering dependencies.

For the evaluation of the proposed approach, we simulate batches of different MPI jobs in the SimGrid [27] environment, which is a well-known distributed computer system simulator. We use two different MPI applications, one exhibiting a regular communication pattern (*LAMMPS*) and one exhibiting an irregular one (*NPB-DT*). Simulations reveal that the pessimistic variant of our placement approach achieves a notable decrease in batch completion time, when compared to the default placement approach of Slurm. In scenarios where approximately 2% of the nodes exhibited an outage probability of 2%, the corresponding improvement over Slurm's policy was 67% and 34%, respectively, for the two MPI applications tried. Additionally, the proposed approach manages to reduce the job instances that are aborted due to node failures.

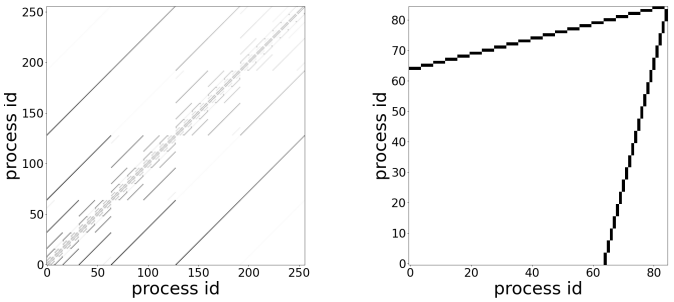
II. The CPTF process placement approach

In this section, we present our approach for assigning the processes of an MPI job onto nodes of a given platform, in manner that is aware of *Communication Profile, Topology and node Failures (*CPTF*)*. We focus on MPI applications whose processes coexist for the entire duration of the job execution. In accordance with relevant studies [7], [28], we formulate the problem of assigning processes to platform nodes as a topology mapping problem. The corresponding mapping is derived prior to program execution. We model the communication among different processes as an undirected graph $G = (V_G, E_G)$. Let $N = |V_G|$ denote the number of MPI processes involved in the corresponding application. For the rest of the study, we refer to this graph as the *communication graph*. Vertex $p_k \in V_G$ corresponds to process with rank k (MPI_COMM_WORLD), an edge $e_{k,l} \in E_G$, connecting vertices p_k and p_l, denotes communication between the corresponding processes. Edge weights may depict either number of messages or total traffic exchanged between the two processes. The evaluation results in Section III are derived by considering total traffic volume as edge weights.

A. MPI Application Analysis via Profiling

For extracting the total bytes exchanged between each pair of processes, we have implemented a custom profiling tool for MPI applications. This tool is a dynamically linked library that intercepts all calls to MPI primitives that initiate traffic, such as point-to-point, collective, and one-sided ones. The output produced is graph G which is represented as a matrix of size $N \times N$. Indexes in G are ranks in MPI_COMM_WORLD. To capture accurately traffic through other communicators, we take into account rank correspondence between each of them and MPI_COMM_WORLD. For the case of collective primitives, the profiling tool is tuned to emulate the appropriate algorithm for each collective. In this way, it is able to accurately capture the traffic exchanged between each pair of processes during each phase of that collective's schedule.

Another feature of our profiling tool is that it produces a traffic heatmap, which depicts the amount of bytes exchanged between each process pair. This traffic heatmap allows for visual inspection of the corresponding application's communication pattern and characterization of it as regular or irregular. Regularity is based on the total traffic exchange between each pair of processes for the whole execution of the application. More precisely, we consider a communication pattern as regular if traffic is arranged along the main diagonal on a similar manner for all processes. This implies that each MPI rank mostly communicates with *nearby* ranks. Discussion regarding the effect of traffic regularity and benchmark selection are deferred for Section III. Figures 1a and 1b depict the traffic heatmap produced by our profiling tool, for the two benchmarks used, that is, LAMMPS [29] and the DT NAS parallel benchmark (NPB-DT) [30], respectively. The darker the data point, the higher the amount of traffic exchanged for the corresponding process pair. Figure 1a shows that



(a) (b)

Fig. 1: (a) LAMMPS run with 256 processes. (b) NPB-DT class C run with 85 processes.

LAMMPS exhibits a more regular communication pattern with traffic points being arranged along the main diagonal on a similar manner for all processes.

The proposed process placement approach is profile-guided. To derive an assignment of MPI processes to nodes, a training run should be carried out first to obtain the corresponding communication graph. However, this cost is amortized by performing multiple runs of the same application using the same input or configuration.

B. System Model based on topology graph

The underlying platform is modeled through topology graph $H = (V_H, E_H)$. Each vertex $n_i \in V_H$, for $i = 1...|V_h|$, corresponds to one node. Vertices carry no weight. Let R denote the routing logic of the underlying platform. Assuming a 3D torus topology with fixed routing, the routing function $R(n_i, n_j)$ provides the list of links to be traversed by a message sent from node n_i to n_j and $|R(n_i, n_j)|$ the number

of them. The weight of the edge connecting nodes n_i to n_j is denoted as $e_{i,j}$ and depicts the number of hops traversed to reach n_i starting from n_j.

The main difference of the proposed process placement approach from similar studies is that assignment of processes to nodes also takes into account node failures. The fault model assumed is the following. Nodes may exhibit failures independently of each other. With the term *failure*, we refer to any hardware- or software-related event, or reboot, that may render the node being temporarily unavailable. We further assume that a node restart is enough to fix transient failures and that restart takes place instantaneously, i.e. we disregard recovery time. When a node enters the failed state, it is incapable of performing both computation and communication, i.e. cannot send, receive, or forward traffic on behalf of other nodes. Consequently, communication attempts initiated by the MPI library will result in error and, in turn, job abortion. Moreover, when a node is in the failed state, it is not able to respond to probes (*heartbeats*) aimed at inferring its availability. Finally, we assume that there is no checkpoint/restart mechanism; thus, after a node fails any affected application restarts from scratch.

We explore three variants of the proposed resource allocation approach (*CPTF*), based on how the expected node failures are handled. We assume that the failure probability of each node is a constant given value, that can be extracted by examining traces of failure events. A common approach is to use such data and find the distribution that provides the best fit for inter-failure time intervals [17], [19], [31]. A simpler approach is to count failure events over a time-series of heartbeats where a missing heartbeat is interpreted as node unavailability. We have integrated a heartbeat mechanism to Slurm for keeping a history of node failures [32].

The first variant, *CPTF-Obl*, takes a failure oblivious approach. Assuming a resource allocation request for *k* nodes, *CPTF-Obl* searches for the first *k* consecutive free nodes, disregarding failure probabilities, and forms a subgraph H_{ob} of the topology graph H.

The second variant, *CPTF-Pes*, adopts a pessimistic approach for handling failure prone nodes, allocating a partition that contains the minimum number of such nodes. An additional requirement is for this partition to be contiguous and rectangular. For the case of the 3D torus, these requirements ensure that there will be no contention among traffic from different partitions [33]. Several approaches have been proposed for extracting a partition from a 3D torus with the aforementioned restrictions, while also avoiding fragmentation [33], [34]. Fragmentation may affect both system utilization and the time that a job waits in the queue before being scheduled [33]. Such an approach would fit in a scheme that explores the interplay between resource allocation and scheduling, as the allocation for one job may impact jobs scheduled later. However, since the focus of this paper is on allocating resources for a single job instance, we rely on a simpler heuristic to extract a partition (subgraph) H_{pes} from H containing the least number of failure-prone nodes. Assume that H is arranged as a 16x8x8 3D torus, and there is a request for 128 nodes. Let $H_{x,y,z=i}$ denote a slice of the torus including every node with $x \in [0, 15]$, $y \in [0, 7]$ and $z = i$. Let also $NF(H_{x,y,z=i})$ denote the number of nodes with non-zero failure probability in slice $H_{x,y,z=i}$. The goal of this heuristic is to find the slice that satisfies $\underset{i}{\operatorname{argmin}} NF(H_{x,y,z=i})$. The same logic applies to other torus arrangements where different blocks of consecutive nodes are examined at various z positions.

The third variant, *CPTF-Weighted*, takes a more measured approach to handling of failure-prone nodes. The idea for *CPTF-Weighted* is to approximate the effect of node failures on the cost of traversing a path by assigning larger weights to paths that include nodes with a non-zero failure probability. This variant produces a weighted version H_w of the original topology graph H. From the routing function $R(n_i, n_j)$ we infer the list of links to be traversed by a message sent from $n_i \in V_H$ to $n_j \in V_H$. For each link $l \in R(n_i, n_j)$, l^s and l^d denote the origin and target of that link respectively. Using this information we maintain a registry, with input a node ID and output the list of paths with this node serving as an intermediate hop. Combining information provided by the routing function and node failure probabilities, we update edge weights in the topology graph using Equation 1.

$$w(e_{u,v}) = \sum_{l \in R(u,v)} 1 + c \times 100 \times \mathbb{1}[(p^f_{l^s} > 0) \vee (p^f_{l^d} > 0)], \quad (1)$$

In Equation 1, $p^f_{l^s}$ and $p^f_{l^d}$ are the failure probabilities for nodes l^s and l^d, respectively. In this equation, c is a constant that corresponds to the penalty assigned to links that include a node with a non-zero failure probability. The value of c is empirically set to 300.0, to lead Scotch to assign a prohibitive cost on failure-prone links, while avoiding an overflow in the computation of the total communication cost.

Moreover, $\mathbb{1}[p^f_{l^s} > 0]$ is an indicator function with a value of 1 if $p^f_{l^s} > 0$. Having described how the communication and topology graphs are populated, the final step is to describe the proposed process placement approach. All three *CPTF* variants use the Scotch graph mapping library [25], [26] to solve the corresponding graph mapping problem. The output is a mapping of vertices (processes) of communication graph G, onto vertices on the topology graphs H_{ob}, H_{pes} and H_w, respectively. For the case of the *CPTF-Obl* variant, this mapping is denoted as M_{ob} while $M_{ob}(p_k)$ returns the node $n_i \in H_{ob}$, where process p_k is assigned. As an approximation of the communication cost achieved by each mapping, we use the *Hops per Byte - HpB* metric which is defined in [35]. This metric expresses the average number of hops that each byte will pass through under a given mapping. Lower HpB values indicate that traffic will cross fewer links until delivery to the destination. For the *CPTF-Obl* variant, the HpB metric is expressed through Equation 2.

$$HpB_{ob} = \sum_{e_{k,l} \in G} \frac{|e_{k,l}| * |R(M_{ob}(p_k), M_{ob}(p_l))|}{|e_{k,l}|} \quad (2)$$

The proposed approach has been integrated into Slurm as a resource selection plugin (cf. [32] for details). This plugin is

invoked upon job arrival to allocate resources. If no resources are found, it is invoked again by the backfill scheduling plugin when the job is about to be dispatched.

III. Evaluation

A. Experimental Methodology

The proposed approach relies on Scotch for solving the corresponding graph mapping problem that derives a layout of MPI processes on platform nodes. In the first part of our evaluation (Section III-B), we verify that in the absence of node failures, Scotch is able to derive mappings that perform better than the default mapping performed by Slurm and the following two simple heuristics: a greedy one and a heuristic that randomly assigns processes to nodes. The evaluation of the proposed process placement approach that considers node failures is then presented in Section III-C.

For the evaluation of *CPTF*, we rely on the SimGrid [27] distributed system simulation framework. Within the SimGrid framework, the SMPI interface [36] is capable of simulating unmodified MPI applications. In SMPI, the communication calls of the application are intercepted and simulated, while the computations are carried out in full on the host machine. To enable the simulation of MPI application execution, we have created the description file expected by SimGrid, describing the links between nodes, nodes and routes. A node has a fixed computing capability, expressed in *floating point operations per second (FLOPS)*. In our case, it is fixed to 6 Gflops. Links are characterized by their bandwidth and latency, which we set to 10 Gbps and one usec, respectively. Bandwidth is intentionally set to a moderate value, to keep experiment duration relatively short. High bandwidth values would mask out the effect of communication cost on job completion time. We assume fixed routing, i.e. each pair of nodes is connected through a single static path. The simulated topology matches exactly the topology assumed for deriving the mapping of processes to platform nodes. The main advantage of the simulation-based evaluation is that it eases experimentation with different torus topologies and injection of artificial faults.

B. Assessing the quality of mappings produced by Scotch

All three variants of the proposed resource allocation approach utilize the Scotch library to derive a mapping of the communication graph to the post-processed topology graph. It is important thus to assess the quality of the mappings produced using Scotch. We compare its performance with the following three approaches: (a) *Default-Slurm*, (b) *Random*, and (c) *Greedy*.

Default-Slurm denotes the default policy employed by Slurm (referred to as Slurm hereafter). The default resource selection of Slurm is implemented through the *linear select* plugin [37] where nodes are arranged in an one-dimensional array. For a request for k nodes with no overcommit requirement, the first k consecutive available nodes are allocated to the job. The *Random* approach randomly picks the node for each process. Finally, the *Greedy* approach sorts all different process pairs with respect to the total traffic exchanged. Then, it iterates over all pairs, in sorted order as per traffic volume, and places processes as close to each other as possible.

Torus	Slurm	Scotch	Scotch-linear	Random	Greedy
16x8x8	4.30	3.80	3.51	6.14	5.58
8x8x16	2.52	3.42	2.71	5.92	4.04
8x16x8	3.47	3.70	3.12	6.78	4.60

TABLE I: HpB metric for LAMMPS with 256 processes.

Torus	Slurm	Scotch	Scotch-linear	Random	Greedy
16x8x8	6.32	2.89	2.35	6.87	3.91
8x8x16	4.89	1.99	2.19	6.52	3.78
8x16x8	6.29	2.34	2.35	7.1	3.7

TABLE II: HpB metric for NPB-DT with 85 processes.

We consider two different Scotch-based approaches. The first one, denoted simply as *Scotch*, uses all available nodes when selecting the subset where the processes will be placed. The second variant, denoted as *Scotch-linear*, considers only the nodes selected by Slurm.

For the evaluation process, we select benchmarks that capture three key parameters that affect the performance of similar resource allocation approaches: the communication to computation ratio, the mix of point-to-point and collective communication, and the communication pattern. The benchmarks selected are LAMMPS [29] and NPB-DT from the NAS parallel benchmark suite [30]. For NPB-DT we focus on the class C variant which involves 85 processes. LAMMPS is a state-of-the-art molecular dynamics code, while NPB-DT is representative of codes with unstructured computation, parallel I/O, and data movement. Our approach minimizes the communication cost; thus in applications where computation outweighs the communication, the expected speedup is insignificant. Both benchmarks spent a significant fraction of their execution time for communication. Moreover, benchmarks dominated by traffic through collective primitives leave limited room for communication cost minimization since there are no specific pairs with remarkably more traffic. Finally, Slurm's allocation policy iterates sequentially over the available nodes; thus it is highly probable for processes with *nearby* ranks to be placed on topologically nearby nodes. This is a reasonable approach for regular communication patterns and leaves limited room for optimizing the communication cost with an approach like the suggested one. However, the default placement policy used by Slurm does not match well with irregular communication patterns. NPB-DT exhibits an irregular communication pattern with a substantial amount of point-to-point traffic. LAMMPS on the other hand exhibits a regular communication pattern and has a significant amount of collective traffic. These two benchmarks allow us to probe the assumptions of Scotch and Slurm and cover all the aforementioned parameters.

We first compare the Hops-per-Byte (HpB) metric resulting from each of the aforementioned approaches, for three different arrangements of a 1024-node 3D torus. Table I presents the corresponding results for the case of a LAMMPS with 256 processes. Overall, Scotch-linear achieves a significant decrease in the HpB. In the 8x8x16 arrangement, the Slurm placement policy only achieves a marginal benefit over Scotch-

Torus	Slurm	Scotch	Scotch-linear	Random	Greedy
16x8x8	184	217	220	157	176
8x8x16	255	210	280	160	214
8x16x8	214	205	212	149	190

TABLE III: Timesteps/sec for LAMMPS with 256 processes.

Torus	Slurm	Scotch	Scotch-linear	Random	Greedy
16x8x8	43.3	29.7	27.0	48.2	29.6
8x8x16	34.6	29.8	29.3	43.1	29.4
8x16x8	39.0	28.9	28.9	51.4	27.6

TABLE IV: Completion time for NPB-DT with 85 processes.

linear. Random allocation, which is communication profile and topology oblivious, results in the highest value of the HpB metric implying higher communication cost. Greedy allocation is only better when compared to Random.

Table II presents the corresponding results for NPB-DT. Comparing Scotch-linear with Slurm, we observe that the improvement in the HpB metric, even with as few as 85 processes, is remarkable. This is due to the irregular communication pattern exhibited by NPB-DT that leaves enough room for Scotch to optimize the mapping of processes to nodes. Moreover, the Greedy heuristic is ranked third in all scenarios explored. Similarly to LAMMPS, the Random approach results in the worst performance.

We further assess the quality of the mappings produced by Scotch through simulations with SimGrid. We derive the assignment of processes to platform nodes, and provide this mapping to SimGrid as its machine file. Using SimGrid's smpirun tool, we run a simulation of the corresponding application. For NPB-DT, the performance metric is completion time. For LAMMPS, the metric is the number of timesteps per second (reported by the application). For the case of Scotch variants, each application's communication graph is extracted through a profiling run with the tool described in Section II. This communication graph becomes the input to Scotch, along with a representation of the platform. Scotch in turn produces the mapping of that job's processes onto platform nodes. Finally, this mapping is given as input to SimGrid.

Table III shows the timesteps per second achieved by each resource allocation approach on each different arrangement of the torus topology, for the case of LAMMPS. Table IV presents the corresponding results (completion time) for NPB-DT. For LAMMPS, in two out of the three torus arrangements, Scotch-linear achieves 20% and 10% more timesteps per second than the default allocation policy of Slurm. For the third arrangement, performance is almost equal. Therefore, even in the case of an application with a regular communication pattern, there is still room for improvement by taking into account the application communication profile and topology. For NPB-DT, an application with an irregular communication pattern, the benefits can be even more significant. Scotch-linear achieves 37.6%, 15.3%, and 26.5% lower completion time than Slurm for the three torus variants. Differing from the pattern observed for LAMMPS, the greedy heuristic achieves performance that is close to Scotch-linear.

Overall, in almost all scenarios explored, Scotch-linear achieves the best performance for both LAMMPS and NPB-DT. In one scenario for the NPB-DT benchmark Greedy achieves better performance but it is under-performing in the case of LAMMPS benchmark.

C. Performance and resilience with CPTF

In this section, we evaluate the performance of the proposed approach in the presence of node failures. Instead of simulating a single MPI job instance, we consider *job batches*, each consisting of 100 instances of the same MPI application. We use two criteria to evaluate each process placement approach: batch completion time, and abort ratio. The batch completion time is the total time required to complete the queue of 100 instances. The abort ratio is the fraction of instances that were aborted due to one or more node failures. As explained in Section II, when a job fails we assume that it is restarted from scratch, rather than being restored from a checkpoint. This assumption simplifies the way to update batch completion times in the presence of node failures. Each time a job is aborted, the batch completion time is augmented by a time interval equal to a successful run, and then the job is restarted.

We follow the fault model introduced in Section II-B, assuming that nodes fail independently of each other and that failures are due to temporary software or hardware errors. We further assume that reboots can fix the errors and that they incur zero overhead due to recovery time. For the first part of our work towards a failure-aware resource allocation approach, our goal is to explore the benefit of avoiding failures and the implications on performance. Replaying traces of node failures offers the advantage of realistic data with the drawback of being machine or load-dependent. Instead, for our evaluation process we relay on a more simple approach for simulating failures. For each application batch, we randomly select a set of nodes N_f that are assumed to exhibit failures. The number of nodes selected to form set N_f is fixed to 2% of the total number of nodes. For the 3D torus arrangement assumed (16x8x8), these nodes participate in approximately 23.2% of all the communication paths (average over all 5 different N_f sets). Instead of generating a synthetic trace of time intervals between successive node failures, we adopt the following approach. For each job instance, the state of each node i, is a random variable x_i with a value $x_i = 0$ indicating a failure. In [38], authors collect and analyze traces from 10 clusters at Los Alamos National Lab. Clusters are arranged in two groups and analysis reveals that the unconditional failure probability of a given node during a random day is 0.31% and 4.6%, respectively, for the two groups. Following these observations, we assume that the failure probability (p_f) of a given node during a job's lifetime is fixed and equal to 2%. For each job instance simulated, a Bernoulli trial on x_i with $p_f = p(x_i == 0) = 0.02$ determines if a failure will be simulated for that node or not.

The evaluation experiments in this section compare Slurm's default placement approach with variants of the proposed placement approach (Section II-B): *CPTF-Pes*, *CPTF-Obl*, *CPTF-Weighted*). *CPTF-Weighted* uses the Scotch method

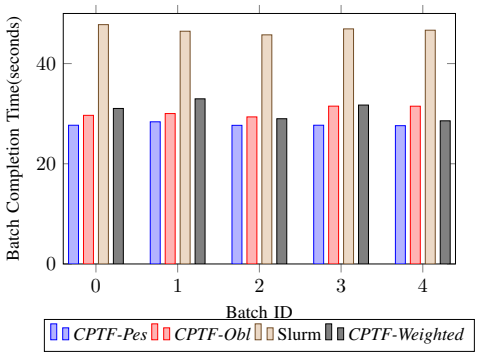


Fig. 2: Batch completion times for NPB-DT.

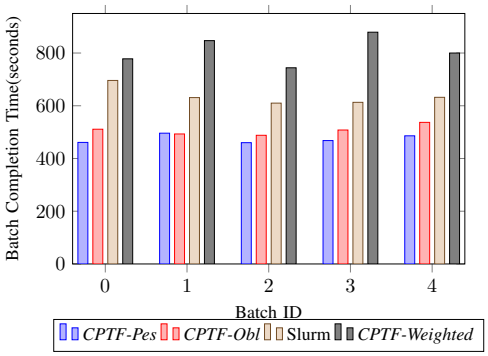


Fig. 3: Batch completion times for LAMMPS.

with the weighted version of the original topology graph, whereas both *CPTF-Pes* and *CPTF-Obl* use the Scotch-linear method with different partitions. Batches of jobs are run in the SimGrid simulator. Each simulated scenario corresponds to a single job instance. The MPI applications simulated are LAMMPS (256 processes) and NPB-DT (85 processes) of class C. The simulated platform consists of 1024 nodes arranged as a 16x8x8 3D Torus. For simulating a node failure, we rely on the fact that SimGrid allows to specify different values for a specific link's capacity, at different points in simulated time. Specifying a link capacity value of zero, results in all transmissions over this link failing, and consequently in aborting the MPI job. These variations in link capacity are defined in the platform description given as input to SimGrid. Thus, for every node that will be emulated as being in the failed state, the platform description is updated by assigning a zero bandwidth value to all links involving this node. Figure 2 presents batch completion times as derived by SimGrid simulations for the case of NPB-DT. The *CPTF-Pes* variant of the proposed approach achieves the lower completion time for all batches. *CPTF-Pes* is the variant that post processes the topology graph in an attempt to avoid failure-prone nodes and paths. The default resource allocation policy of Slurm achieves the higher batch completion time. Compared to Slurm, *CPTF-Pes* offers 67% lower batch completion times (on average over all five batches). This drop is due to two factors: (1) the reduction in the communication cost as the result of topology- and application profile-aware placement; and (2) the drop in the jobs being aborted due to node outages. The average job abort ratio (over 500 simulated NPB-DT instances) is 0.8% for the case of *CPTF-Pes*, and 5.6% for Slurm's default process placement policy. The failure-oblivious variant *CPTF-Obl* results in 9.3% higher average batch completion time when compared to the pessimistic variant *CPTF-Pes*, and an abort ratio of 5.6%. Overall, *CPTF-Pes* results in an abort ratio 85% lower than that of Slurm and *CPTF-Obl*.

Figure 3 presents the corresponding simulated results for the case of LAMMPS. On average over all 5 batches, *CPTF-Pes* achieves 34% lower batch completion time than Slurm. Comparing this result with the case of NPB-DT, the performance gain for an application with a regular communication pattern is reduced. However, the average job abort ratio over all batches is 5.4% for the case of *CPTF-Pes* and 12.4% for the case of Slurm. As in the case of NPB-DT, the drop in the batch completion time is attributed to the reduction of both the communication cost and the overhead of failed jobs that need to be restarted. Comparing the failure-oblivious and pessimistic variants reveals that ignoring node failures results in 7.0% higher average batch completion time. However, *CPTF-Pes* achieves a lower job abort ratio when compared to *CPTF-Obl*, with the corresponding ratios being 5.4% and 12.4%, respectively (i.e. a 57% reduction in job aborts).

For both LAMMPS and NPB-DT, the *CPTF* variant that disregards node failure probabilities does not achieve significantly worse performance than *CPTF-Pes*. For LAMMPS, requesting 25% of total nodes (i.e. 256 out of 1024), makes it hard to find a contiguous and rectangular partition with a minimum number of failure-prone nodes. For NPB-DT *CPTF-Pes* achieves an 85% reduction in job abort ratio, whereas for LAMMPS the reduction in job abort ratio is 57%. Moreover, the feasible improvement is also limited by the ratio of nodes that are emulated as failure-prone and their failure probability. Note though that the benefit achieved over the *CPTF-Obl* is not negligible, especially if one considers long running jobs. Avoidance of even a few job abortions saves a significant amount of cluster resources. Furthermore, failure avoidance is also important from the point of view of the user that will experience lower job completion delay. Another observation for both benchmarks is that the *CPTF* variant, which adjusts weights in the topology graph to capture the effect of node failures, exhibits rather high batch completion times. Despite the assignment of prohibitively high costs on failure-prone links, *CPTF-Weighted* fails in preventing Scotch from selecting paths with failure prone nodes. For LAMMPS, the average abort ratio is 49.6%, 12.4% and 5.4% for *CPTF-Weighted*, *CPTF-Obl* and *CPTF-Pes*, respectively.

IV. Related work

For improving the performance of MPI applications there are numerous studies targeting different parts of the MPI implementation including collectives [2], [39], point-to-point primitives [4] or utilizing hardware support [3]. Studies though that are most related to our work employ as input two graphs, capturing the application communication patterns along with the platform architecture, respectively. Most of

them address the mapping problem described in [40], i.e. assign processes of an application onto computational resources of the available platform [28]. Numerous such studies focus on deriving improved methods for solving the corresponding mapping problem [6]–[10]. The work in [5] presents an algorithm for reordering ranks of an MPI job so that communication cost is minimized. A similar approach [41] formulates the assignment of processes to nodes as a *Quadratic Assignment Problem (QAP)* [40], and applies a heuristic for solving it. The work in [10] discusses an approach that is complementary to that of assigning processes of a parallel job onto platform resources. Specifically, this approach rearranges the logical communication of broadcast and allgather operations taking into account process distances, instead of process ranks, along with topology information. In all aforementioned studies, resource allocation considers only the application's communication pattern and the target machine's topology, ignoring the impact of failures on performance.

Acknowledging the impact of failures on both application and system performance, several studies adopt failure-aware resource allocation approaches. The work in [23] suggests a resource management scheme for workloads that run within virtual machines (VMs). For allocating nodes to VMs both their capacity and reliability is taken into account. The key idea is that VMs in failed nodes can be migrated. However, VMs incur some overhead; therefore, our approach assumes jobs running directly on the available resources. The work in [42] presents algorithms that allocate resources for MPI jobs, such that system reliability is maximized, by taking into account the probability of nodes failing during the execution time of each job. Our work differs by considering both reliability and communication cost for assigning resources to job processes. Work in [33] suggests a scheduler for the BlueGene/L system that allocates nodes to a job, taking into account node reliability and the maximal free partition left available after allocation. Our work differs in that we also take into account application communication patterns before performing resource allocation.

Exploiting the predictability of failures at various levels [11], [22], [42], [43], several studies aim to mitigate the effect of faults through checkpoint and restart. The key idea is to mine the temporal pattern of faults and, based on inferred patterns, optimize the checkpointing interval [17]–[19], [22], thus reducing the checkpointing overhead which can be significant at large scales [13]. Our current work assumes no checkpoint mechanism; however, part of our future work is to explore the benefit of checkpointing.

From the point of view of the parameters taken into account for resource allocation, our approach is mostly related to studies aiming at reliability-aware scheduling of parallel applications. In such studies applications consist of multiple tasks that are modelled by a directed acyclic graph. Scheduling can then target both a minimized scheduled length and reduced failure probability [21]. The work in [20] extends the dynamic level scheduling (DLS) algorithm to consider the reliability of resources. The Dynamic Reliability-Cost-Driven (DRCD) Scheduling Algorithm has been developed in the context of real-time scheduling for parallel jobs [44], with the primary goal to minimize reliability cost. Our *CPTF* approach focuses on MPI jobs, i.e. jobs whose processes co-exist for the duration of the application run. There are no process deadlines to be satisfied nor any time-based dependencies among processes.

V. Conclusions

In this work we contribute *CPTF*, a resource allocation approach for improving the performance of parallel MPI jobs running on a cluster. It tackles two contributors to MPI job completion time: communication cost and job restarts due to node failures. We consider both the cluster topology and job communication profiles for minimizing communication cost. Moreover, we also take into account node failure probabilities before assigning job processes to cluster nodes. The motivation is that avoiding failure-prone nodes and communication paths limits the number of jobs that will be restarted due to node failures, thus reducing resource waste. We derive three variants of *CPTF* based on how node failure probabilities are used in processing the cluster topology graph.

We evaluate our approach through SimGrid simulations of two applications, one with a regular communication pattern (LAMMPS) and one with an irregular one (NPB-DT). In both cases, the variant of *CPTF* that strives to avoid failure-prone nodes and communication paths achieves lower time to complete job batches when compared to the default resource allocation policy of Slurm. It also exhibits the lowest ratio of aborted jobs. The average improvement in batch completion time is 67% for NPB-DT and 34% for LAMMPS.

As future work, we are comparing the *CPTF* approach with one that schedules checkpoints based on an estimation of likely failures, to determine which of the two results in improved batch completion time and resource utilization.

Acknowledgments

We thankfully acknowledge the support of the European Commission under the Horizon 2020 Framework Programme for Research and Innovation through the projects ExaNeSt (GA 671553) and EuroEXA (GA 754337). Moreover, we acknowledge GRNET S.A. for awarding us access to the Greek national high performance systems, ARIS.

References

[1] "MPI:A Message-Passing Interface Standard." [Online]. Available: https://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf

[2] W. Yu, T. S. Woodall, R. L. Graham, and D. K. Panda, "Design and implementation of open mpi over quadrics/elan4," in *19th IEEE International Parallel and Distributed Processing Symposium*, 2005, pp. 10 pp.–.

[3] Y. Peng, M. Saldana, and P. Chow, "Hardware support for broadcast and reduce in mpsoc," in *2011 21st International Conference on Field Programmable Logic and Applications*, 2011, pp. 144–150.

[4] M. J. Rashti and A. Afsahi, "Improving communication progress and overlap in mpi rendezvous protocol over rdma-enabled interconnects," in *2008 22nd International Symposium on High Performance Computing Systems and Applications*, June 2008, pp. 95–101.

[5] C. Sudhakar, P. Adhikari, and T. Ramesh, "Process assignment in multi-core clusters using job assignment algorithm," in *2016 Second International Conference on Computational Intelligence Communication Technology (CICT)*, Feb 2016, pp. 259–264.

[6] E. Rodrigues, F. Madruga, P. Navaux, and J. Panetta, "Multi-core aware process mapping and its impact on communication overhead of parallel applications," 07 2009, pp. 811–817.

[7] J. L. Traff, "Implementing the mpi process topology mechanism," in *SC '02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, Nov 2002, pp. 28–28.

[8] E. Jeannot, G. Mercier, and F. Tessier, "Process placement in multicore clusters:algorithmic issues and practical techniques," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 4, pp. 993–1002, April 2014.

[9] E. Jeannot and G. Mercier, "Near-optimal placement of mpi processes on hierarchical numa architectures," in *Euro-Par 2010 - Parallel Processing*, P. D'Ambra, M. Guarracino, and D. Talia, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 199–210.

[10] F. Ercal, J. Ramanujam, and P. Sadayappan, "Task allocation onto a hypercube by recursive mincut bipartitioning," *Journal of Parallel and Distributed Computing*, vol. 10, no. 1, pp. 35 – 44, 1990.

[11] D. Jauk, D. Yang, and M. Schulz, "Predicting faults in high performance computing systems: An in-depth survey of the state-of-the-practice," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: ACM, 2019, pp. 30:1–30:13.

[12] M. Snir, R. Wisniewski, J. Abraham, S. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, A. Chien, P. Coteus, N. DeBardeleben, P. Diniz, C. Engelmann, M. Erez, S. Fazzari, A. Geist, R. Gupta, and E. Van Hensbergen, "Addressing failures in exascale computing," *ICiS Workshop*, vol. ANL/MCS-TM-332, 04 2013.

[13] B. Schroeder and G. Gibson, "Understanding failures in petascale computers," *Journal of Physics: Conference Series*, vol. 78, 09 2007.

[14] N. El-Sayed, H. Zhu, and B. Schroeder, "Learning from failure across multiple clusters: A trace-driven approach to understanding, predicting, and mitigating job terminations," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 1333–1344.

[15] E.N.Elnozahy, R. Bianchini, T. El-Ghazawi, A. Fox, F. Godfrey, A. Hoisie, K. McKinley, R. Melhem, J. Plank, P. Ranganathan, and J. Simons, "System resilience at extreme scale," Defense Advanced Research Project Agency, Tech. Rep., 2008.

[16] C. D. Martino, W. Kramer, Z. Kalbarczyk, and R. Iyer, "Measuring and understanding extreme-scale application resilience: A field study of 5,000,000 hpc application runs," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2015, pp. 25–36.

[17] E. Heien, D. LaPine, D. Kondo, B. Kramer, A. Gainaru, and F. Cappello, "Modeling and tolerating heterogeneous failures in large parallel systems," in *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2011, pp. 1–11.

[18] Y. Li and Z. Lan, "Exploit failure prediction for adaptive fault-tolerance in cluster computing," in *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, vol. 1, 2006, pp. 8 pp.–538.

[19] D. Tiwari, S. Gupta, and S. S. Vazhkudai, "Lazy checkpointing: Exploiting temporal locality in failures to mitigate checkpointing overheads on extreme-scale systems," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2014, pp. 25–36.

[20] A. Dogan and F. Ozguner, "Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 308–323, 2002.

[21] M. Hakem and F. Butelle, "Reliability and scheduling on systems subject to failures," in *2007 International Conference on Parallel Processing (ICPP 2007)*, 2007, pp. 38–38.

[22] Y. Zhang, M. S. Squillante, A. Sivasubramaniam, and R. K. Sahoo, "Performance implications of failures in large-scale cluster scheduling," in *Job Scheduling Strategies for Parallel Processing*, D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 233–252.

[23] S. Fu, "Failure-aware resource management for high-availability computing clusters with distributed virtual machines," *J. Parallel Distrib. Comput.*, vol. 70, no. 4, p. 384–393, Apr. 2010.

[24] F. Machida, M. Kawato, and Y. Maeno, "Redundant virtual machine placement for fault-tolerant consolidated server clusters," in *2010 IEEE Network Operations and Management Symposium - NOMS 2010*, 2010, pp. 32–39.

[25] F. Pellegrini and J. Roman, "Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs," in *High-Performance Computing and Networking*, H. Liddell, A. Colbrook, B. Hertzberger, and P. Sloot, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 493–498.

[26] "Scotch: Software package and libraries for sequential and parallel graph partitioning, static mapping and parallel sparse matrix block ordering." [Online]. Available: http://www.labri.fr/perso/pelegrin/scotch/

[27] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899 – 2917, 2014.

[28] T. Hoefler and M. Snir, "Generic topology mapping strategies for large-scale parallel architectures," in *Proceedings of the International Conference on Supercomputing*, ser. ICS '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 75–84.

[29] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *J. Comput. Phys.*, vol. 117, no. 1, pp. 1–19, Mar. 1995. [Online]. Available: http://dx.doi.org/10.1006/jcph.1995.1039

[30] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, "The nas parallel benchmarks," The International Journal of Supercomputer Applications, Tech. Rep., 1991.

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

[32] I. Vardas, "Process Placement Optimizations and Heterogeneity Extensions to the Slurm Resource Manager," Institute of Computer Science, Foundation for Research and Technology - Hellas (FORTH), Tech. Rep. 477, 2020. [Online]. Available: https://tinyurl.com/y7rggcvj

[33] A. J. Oliner, R. K. Sahoo, J. E. Moreira, M. Gupta, and A. Sivasubramaniam, "Fault-aware job scheduling for bluegene/l systems," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, 2004, pp. 64–.

[34] Hyunseung Choo, Seong-Moo Yoo, and Hee Yong Youn, "Processor scheduling and allocation for 3d torus multicomputer systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 5, pp. 475–484, 2000.

[35] T. Agarwal, A. Sharma, A. Laxmikant, and L. V. Kale, "Topology-aware task mapping for reducing communication contention on large parallel machines," in *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, 2006, pp. 10 pp.–.

[36] A. Degomme, A. Legrand, G. Markomanolis, M. Quinson, M. L. Stillwell, and F. Suter, "Simulating MPI applications: the SMPI approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 8, p. 14, Aug. 2017. [Online]. Available: https://hal.inria.fr/hal-01415484

[37] "Slurm Resource Selection Plugin." [Online]. Available: https://slurm.schedmd.com/selectplugins.html

[38] N. El-Sayed and B. Schroeder, "Reading between the lines of failure logs: Understanding how hpc systems fail," in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2013, pp. 1–12.

[39] M. Ruefenacht, M. Bull, and S. Booth, "Generalisation of recursive doubling for allreduce," *Parallel Comput.*, vol. 69, no. C, pp. 24–44, Nov. 2017.

[40] Bokhari, "On the mapping problem," *IEEE Transactions on Computers*, vol. C-30, no. 3, pp. 207–214, March 1981.

[41] C. D. Sudheer and A. Srinivasan, "Optimization of the hop-byte metric for effective topology aware mapping," in *2012 19th International Conference on High Performance Computing*, Dec 2012, pp. 1–9.

[42] N. R. Gottumukkala, C. B. Leangsuksun, N. Taerat, R. Nassar, and S. L. Scott, "Reliability-aware resource allocation in hpc systems," in *2007 IEEE International Conference on Cluster Computing*, 2007, pp. 312–321.

[43] S. Di, H. Guo, R. Gupta, E. R. Pershey, M. Snir, and F. Cappello, "Exploring properties and correlations of fatal events in a large-scale hpc system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 2, pp. 361–374, 2019.

[44] X. Qin and H. Jiang, "A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters," *J. Parallel Distrib. Comput.*, vol. 65, pp. 885–900, 08 2005.