Research Project Proposal: Fault mitigation and tolerance for MPI applications

Roberto Rocco, roberto2.rocco@mail.polimi.it

1. Introduction to the problem

High-Performance Computing (HPC) is the field of computer architectures aimed at reaching the highest computation capabilities. With the evolution of technology, HPC changed too: from a single high-performance machine to a network of communicating and collaborating computers (also called a cluster). In such a distributed environment communication between machines plays a major role. Fault tolerance is the field of distributed computing aimed at the design of algorithms and systems able to deal with faults. These two areas used to be very distant but, due to the evolution of the architectures, nowadays more and more efforts are exploring the intersection of the two.

In the past fault tolerance in HPC used to be underestimated: when a single machine was used, the Mean Time Between Failures (MTBF) was big enough to allow execution without relevant problems. The more machines are used, however, the more the MTBF shrinks. This makes fault-tolerance a relevant topic in modern HPC because programs are likely to encounter one fault (at least) and must be able to handle them. The topic importance is crucial given the lack of proper fault management in the Message Passing Interface (MPI), the de-facto standard for HPC communication, where a single fault can stop the entire execution.

The importance of the field is supported by some efforts: Schroeder and Gibson [11] have collected data at two large high-performance computing sites, showing failure rates from 20 to more than 1000 failures per year. Future systems will be hit by error/faults much more frequently due to their scale and complexity [3].

The main effort towards providing fault tolerance mechanisms in MPI is the User Level Failure Mitigation (ULFM) library [2]. ULFM is a powerful tool for a programmer since it contains implementations of various useful algorithms for fault tolerance. It's based on the fact that the user (the programmer) knows how the application works, so the integration is done most efficiently. ULFM is the main effort in the field and will probably be integrated into the MPI standard in the future. Its automatic use, however, may lead to inefficient solutions.

The problem this research effort is committed to solve is the analysis of the feasibility and the performance costs of a transparent fault-tolerance integration in data-parallel MPI applications. Transparency is core: without the need to change the code of the MPI application, even the ones not directly dealing with fault-tolerance can obtain benefits. This aspect is very important since many HPC applications are well-tested and any change would need a lot of time to obtain the same assurances. Moreover, the restriction toward data-parallel applications should allow a more specific analysis that can ignore many difficult parts of a generic fault-tolerance analysis. The analysis can use ULFM as a starting point, exploiting all the facilities it provides.

The importance of such an analysis comes also from the diffusion of data-parallel applications in HPC due to their intrinsic scalability and compatibility with the upcoming exascale infrastructures. These applications are usually also called *embarassingly parallel* due to the simplicity of the parallel architecture they exploit.

2. Main related works

Few efforts focused on the transparent introduction of fault tolerance in MPI applications: most of the efforts are towards extensions of the MPI standard rather than changing it. This is mainly caused by the fact that the most used technique for implementing fault-tolerance (checkpoint and restart, C/R) prefers application knowledge: without it, the approach is a lot less efficient in terms of memory usage and consequently time needed. By coupling this with the fact that extending the standard is easier and more retro-compatible than changing it, it's

easy to see the reasons why the vast majority of the efforts tend to adopt a more intrusive approach ([8] [12] [9] [6]).

Some efforts tried using different approaches than C/R: Algorithm-Based Fault Tolerance (ABFT) [4] is one of the oldest approaches, but it's strictly application-specific and cannot be applied systematically; log-based and intercommunicator-based solutions were considered too but received few attentions due to their intrinsic scalability issues.

Some other efforts followed a more transparent approach, analyzing the possibility to change the behavior of the MPI calls to introduce fault tolerance ([1] [5]). These analyses had to tackle problems arising from the lack of knowledge of the running application and usually show a loss of performance when compared to a more intrusive approach. Managing to find the correct spot in the trade-off between intrusiveness and transparency may show possible gains both in terms of performance and reusability. A deeper analysis and classification of the main related works can be seen in [10].

3. Research plan

The research process will be split into three macro-phases: prelude to research, actual analyses and result evaluation and refinement.

The first one will provide all the knowledge that will be needed in the next phases: it includes the analysis of the state of the art, the definition of the problem and some experiments to practice with the tools that will be used (like ULFM). Most of it has already been completed.

The second macro-phase is the main part of the analysis, where various approaches are considered. It can be further divided into four phases:

- **Algorithm design**: in this phase, many problems that may arise during the future analyses are arisen and solved via the definition of various algorithms. Typical problems are the retrieval of information in an unstable or partial communicator, the routing of the communication in a multi-communicator approach, the approach to checkpoint and restart. The output of this step will be a series of algorithms that can be implemented in future phases.

- **Direct mitigation**: in this phase, the first approach is considered. The aim is to allow an application to continue its execution despite the presence of faults, realizing what is generally called *graceful degradation*. All the processes of the application will have to participate in the repair MPI communication network, but the operation must be completed transparently. The output of the phase will be a tool having the functionalities explored in the analysis that can be tested to deeply consider the impact in terms of performance. The phase includes functional testing and performance testing to evaluate the quality of the tool produced and to properly optimize it.

- **Hierarchical mitigation**: in [7], it was pointed out that a problem of ULFM is that every process has to participate in the recovery process and it didn't scale as well as the rest of the library. The effort explored different solutions to the problem but decided that they weren't successful and moved away from ULFM. In this phase, the same problem is tackled but from a different perspective: assuming that the recovery process doesn't scale well, we split the underlying communication in sub-networks so that a failure is further isolated. This approach exposes various challenges, like the creation of an efficient routing system for inter-sub-networks communication and the masking of the underlying infrastructure to the application, since it expects a compact network like in the previous approach. As in the previous phase, the output of the phase will be a tool having such functionalities (graceful degradation and non-global repair) and will contain functional and performance testing.

- **C/R fault tolerance**: in this phase, the focus is towards recovery rather than graceful degradation. Rather than continuing execution upon fault presence, the faulty process is restored from a previously saved checkpoint. The main challenges of this phase are checkpoint saving, checkpoint restore and dynamic

process management, all done transparently. Also in this phase, a tool will be produced and testing will be performed.

The last macro-phase will cover all of the comparison tests, evaluating the goodness of the previous analyses. It will be split into two phases:

- **Performance comparison**: in this phase, results from the tests are compared. Solutions produced by other efforts are analyzed too, to find optimal configurations with various applications and environments.
- **Paper writing**: in this phase, all the results obtained are synthesized in the final document written in the form of a scientific paper. This phase includes all the graph creation and data visualization steps.

The nature of the research is hybrid since it involves both the creation of tools and the evaluation of their performance: the latter is core since performance is the primary goal of any HPC application. The Gantt diagram provided in Figure 1 shows an approximate timeline of the project. It shall be noted that the timing overlaps between the various phases are mainly due to the reuse of some methodologies between the approaches. The project will be evaluated based on the reliability improvements it's able to provide, on the results of the performance tests and the quality of the produced paper.

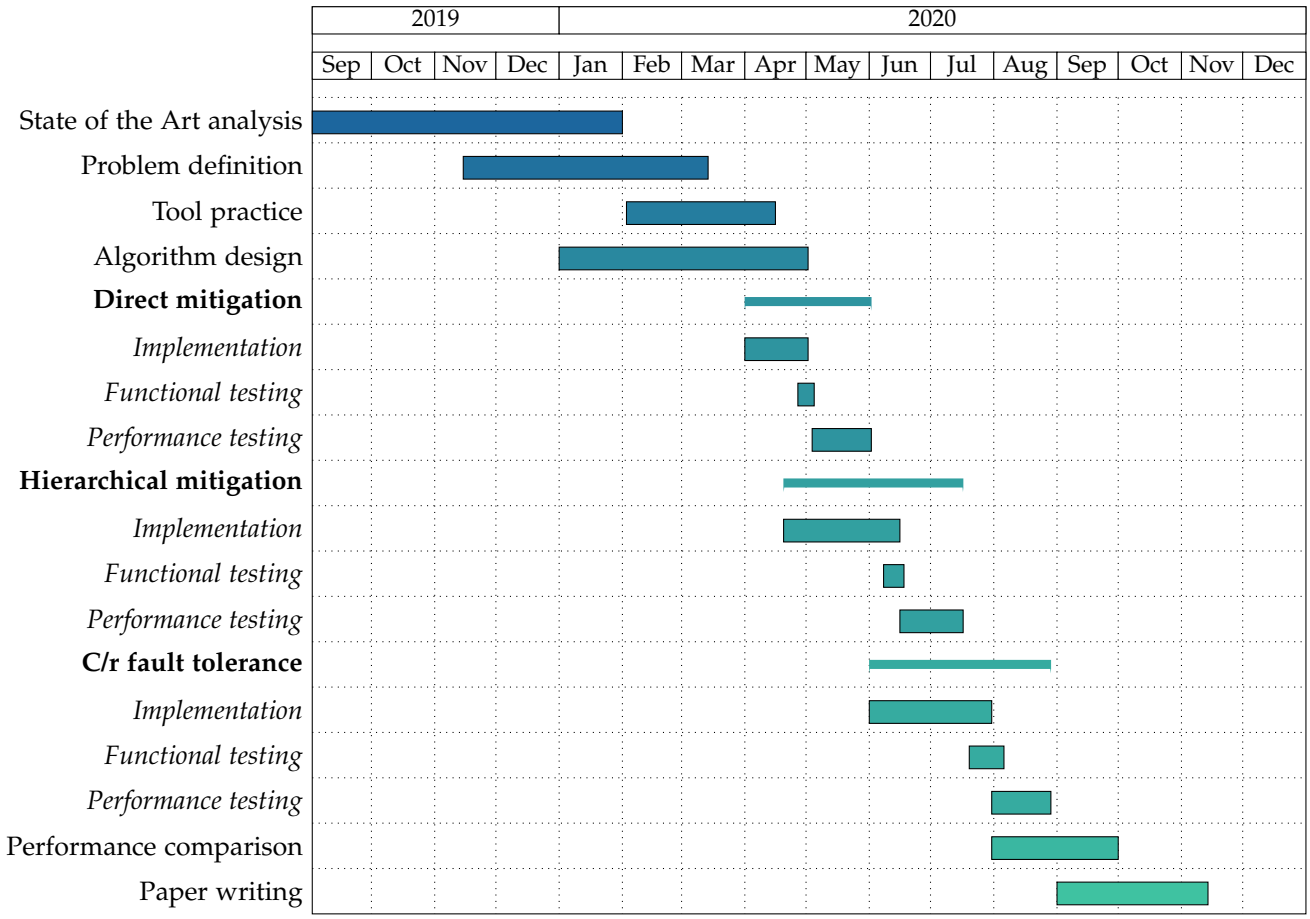

Figure 1: Gantt diagram of the research project

References

[1] Adam, J., Besnard, J.-B., Malony, A. D., Shende, S., Pérache, M., Carribault, P., and Jaeger, J. Transparent high-speed network checkpoint/restart in mpi. In *Proceedings of the 25th European MPI Users' Group Meeting* (2018), pp. 1–11.

[2] Bland, W., Bouteiller, A., Herault, T., Bosilca, G., and Dongarra, J. Post-failure recovery of mpi communication capability: Design and rationale. *The International Journal of High Performance Computing Applications 27*, 3 (2013), 244–254.

[3] Cappello, F. Fault tolerance in petascale/exascale systems: Current knowledge, challenges and research opportunities. *The International Journal of High Performance Computing Applications 23*, 3 (2009), 212–226.

[4] Du, P., Bouteiller, A., Bosilca, G., Herault, T., and Dongarra, J. Algorithm-based fault tolerance for dense matrix factorizations. *Acm sigplan notices 47*, 8 (2012), 225–234.

[5] Fagg, G. E., Bukovsky, A., and Dongarra, J. J. Harness and fault tolerant mpi. *Parallel Computing 27*, 11 (2001), 1479–1495.

[6] Gamell, M., Katz, D. S., Kolla, H., Chen, J., Klasky, S., and Parashar, M. Exploring automatic, online failure recovery for scientific applications at extreme scales. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2014), IEEE, pp. 895–906.

[7] Gamell, M., Teranishi, K., Heroux, M. A., Mayo, J., Kolla, H., Chen, J., and Parashar, M. Local recovery and failure masking for stencil-based applications at extreme scales. In *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2015), IEEE, pp. 1–12.

[8] Losada, N., Bautista-Gomez, L., Keller, K., and Unsal, O. Towards ad hoc recovery for soft errors. In *2018 IEEE/ACM 8th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS)* (2018), IEEE, pp. 1–10.

[9] Losada, N., Cores, I., Martín, M. J., and González, P. Resilient mpi applications using an application-level checkpointing framework and ulfm. *The Journal of Supercomputing 73*, 1 (2017), 100–113.

[10] Rocco, R. State of the art on: Fault-tolerant mpi.

[11] Schroeder, B., and Gibson, G. A large-scale study of failures in high-performance computing systems. *IEEE transactions on Dependable and Secure Computing 7*, 4 (2009), 337–350.

[12] Teranishi, K., and Heroux, M. A. Toward local failure local recovery resilience model using mpi-ulfm. In *Proceedings of the 21st european mpi users' group meeting* (2014), pp. 51–56.