State of the Art on: Fault-tolerance in MPI

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

1. Introduction to the research topic

High-Performance Computing (HPC) is the field of computer architectures aimed at reaching the highest computation capabilities. Its studies range from hardware to all the protocols and algorithms that make the entire structure performing. Performance is the main focus of the field, power and cost efficiency take a less important role. With the evolution of technology, HPC changed too: from a single high-performance machine to a network of communicating and collaborating computers (also called cluster). In such a distributed environment communication between machines plays a major role.

Fault tolerance is a distributed computing field aiming at the design of applications able to withstand some sort of failures. For a program to be fault-tolerant problems must be handled within the application, without the help of a user. Fault tolerance is very important where errors may create meaningful damage, such as database management (may cause data loss) or critical applications. Fault tolerance in HPC used to be underestimated: in the past, the Mean Time Between Failures (MTBF) was big enough to allow execution without relevant problems. During the years, however, the dimension of clusters has grown and the MTBF has shrunk: it used to be measured in days [20], but it is now measured in minutes and it's estimated that soon it will be measured in seconds [5]. This makes faults a relevant problem in HPC because programs are likely to encounter one (at least) and must be able to handle them.

In the last years, fault tolerance in HPC is gaining more and more attention: popular conferences like SC (International Conference for High-Performance Computing, Networking, Storage and Analysis), ICS (International Conference on Supercomputing), HPDC (International ACM Symposium on High-Performance Parallel and Distributed Computing) and DSN (IEEE/IFIP International Conference on Dependable Systems and Networks) often feature some efforts in the field. Discussion is further expanded by FTXS (Fault Tolerance for HPC at eXtreme Scale), a workshop part of many important conferences that every year gathers the most experts in the field. Moreover, the increase in the frequency of correlated publications in the last years shows that the interest in this field is growing. Some publications can be found also in journals published by important groups like IEEE Computer Society, Elsevier and Sage Publications.

1.1. Preliminaries

To better understand the topic some basic rudiments of the HPC environment and fault tolerance are needed. This subsection is structured as follows: the first part will cover the basics of the Message Passing Interface (MPI) and a quick overview of the main techniques and algorithm used in fault tolerance, while the second will focus on the utilities that are (or can be) needed or for the research process.

1.1.1 Research topic overview

Message Passing Interface (MPI) [19] is an interface used for message passing between processes. From its definition, it increased in popularity and it has been the standard de facto for HPC for a long time. MPI main focus is the reduction of the communication overhead to a minimum: this led to the creation of a very fast interface but lacking many useful features, like fault tolerance techniques.

Communication in MPI is based on **communicators**: these are MPI structures that represent a network. All the processes are part of a global communicator by default, but others can be created by calling proper functions. Each communicator assigns a **rank** to each process within it: it is a unique number from 0 to size-1, where **size** is the number of processes in the communicator. Ranks play a core role in MPI, especially the global communicator ones: they are used to identify processes and are treated like addresses during communication.

There are three types of communication in MPI: **Point-to-point** communication allows two processes to cooperate to achieve communication; **Collective** communication involves all the processes within a communicator; **One-sided** communication allows a process to read/write memory of another. Communication can happen only between processes part of the same communicator.

Fault tolerance is a wide field of study: errors are a very heterogeneous domain, thus leading to different solutions. Among the macro-subjects of Fault tolerance, redundancy, failure masking and recovery play a major role. **Redundancy** is the duplication of critical components or functions of a system to increase the reliability of the system. **Failure masking** is the ability to hide the erroneous behavior before it affects the program. **Recovery** is the ability of a program to go beyond failure. Each of these macro-areas contains various algorithms and concepts that must be considered. Other areas of distributed programming like **error detection**, **agreement**, **gossiping**, etc. contain notions and algorithms that may be useful in some solutions.

1.1.2 Utilities

HPC applications tend to be written in a language that can allow precise control of the underlying machine. The most used languages are C/C++ and Fortran since they have these prerequisites and are portable and fast. Knowledge and use of at least one of the two is needed to properly test the results of the research process.

There are many working implementations of MPI available, all differing in some underlying concepts. OpenMPI is one of the most used since it combines technologies and resources from several other efforts. The research process will use it since many recent efforts in the field are going in the same direction.

In the MPI standard, there are ways to implement profiling directly in the library by using the profiling calls (PMPI). These allow the user to intercept all the calls to the MPI library and to gather data. PMPI allows the almost complete customization of MPI calls, making the transparent introduction of code easier.

Recently many fault-tolerant MPI applications are adopting a set of functions grouped under the User Level Failure Mitigation (ULFM) library [3]. The library provides functions for error detection, propagation, agreement, and recovery. Its adoption is growing and, despite not being part of the MPI standard, it's the main effort towards providing fault tolerance mechanisms in MPI and will probably be integrated into future versions.

1.2. Research topic

In the past, MPI applications didn't deal with fault tolerance: a single fault could stop the execution, but properly handling it would have a higher impact than just restarting the application. With the increase of the computational needs, faults got more and more common, making the overheads smaller than the time lost to restarts. And with the rush toward more computational power (going to exaflops) [2], the problem will only get worse.

ULFM, while being a powerful and useful tool, is hard to use. To properly implement functionalities, the programmer must know the basics of fault tolerance and deeply analyze its code. Many efforts tried to overcome the problem by going towards automatic integration [1,8,9,13,15,16,18]. These solutions are, however, subject to a performance/flexibility tradeoff: some program patterns can simplify the fault-tolerance process, making a general approach excessive and slower than an ad-hoc one.

Any generic application, upon failure, must lead back to a coherent execution state. To check coherency, both *orphan* and *missing* messages must be considered: the first is a message received before a checkpoint but sent after, the second is the other way around. Orphan messages must be avoided while missing ones must be saved to be re-sent in case of rollback [11]. If the absence of orphan messages cannot be avoided, all the processes receiving those must rollback as the failed process does. This may cause a domino effect that would make execution restart from a very old point, wasting a lot of computations.

Not all applications would need such complexity though. Data-parallel applications (in particular *embarrassingly parallel applications*) have a simple structure: one master, many slaves [12]. Slaves receive pieces of information from the master, perform some computations and send the result back: no message between slaves is needed. This simplifies the process of reconstruction of a global view: upon failure, only the master and the failed process may have to rollback. Any check for orphan or missing messages as above would only worsen performance.

Data-parallel applications are compatible with the upcoming exascale infrastructures. Their scalability resides in the low communication cost in terms of performance, since the program will spend most of its time in local computations: this is a trait that many high-throughput application share. Focusing the analysis on these applications can be very useful for the upcoming future of exascale computing.

2. Main related works

In this section, we will analyze all the main efforts in the research field. The first subsection will present two classification metrics, the second will analyze more in detail the techniques and the approaches adopted by the various efforts.

2.1. Classification of the main related works

Fault-tolerant MPI can be placed inside two groups of more general research fields: the HPC-related one and the fault tolerance related one. Classification of these two fields is very different so it's better to analyze only the intersection between the two. Efforts can be classified based on the approach for the integration with the current MPI standard and the locality of the recovery protocol adopted.

Integration with the MPI standard can be done in different ways. The most common and easy approach is by the creation of an **extension**: the MPI standard is left unchanged and all the new features are in separate files [8,9,13–18]. This ensures compatibility with code not running the fault-tolerant functions but benefits are exploitable only by directly using those functions. On the other side, **changing the MPI standard** will allow fault-tolerance in any program, but full compatibility is not ensured [1,7]. The choice of the integration way implies a different focus on the use of the tool: an extension-based approach is more oriented towards future applications since the programmer has to know the tool features to properly exploit it; on the other hand, a change-based approach is focused more on making already-designed applications work with the wanted features without changes. Some efforts don't emphasize on the integration with the MPI standard: they just implement fault-tolerance on a single application by changing its code [4,10]. They focus more on the approach rather than the production of a reusable artifact.

The other classification criterion is based on the impact a failure has over the network: if the recovery process affects in a major way (it has to restart) all the nodes of the network, it is called **global recovery** [8,16,17]; if the affected nodes are only a subset of the network it is called **local recovery** [9,14,15,18]. Sometimes recovery is not addressed (the application continues only with the working processes or the failure is solved without rollback) [7,10,13]. Global recovery is easier to implement but doesn't scale well with the increase of the network size and the reduction of the MTBF. Local recovery speeds up the process concerning the global counterpart but needs dependency checking, more complex computations, more complex code, and proper application algorithm features to be effective. Many efforts tend to start from global recovery and then integrate local recovery as an improvement, due to the higher difficulty of the second approach ([9] is the evolution of [8], [15] is the evolution of [16]).

Following the above classification criteria, we can represent the most important efforts in the field in Table 1.

2.2. Brief description of the main related works

Fault tolerance can be achieved following many different approaches. In the following paragraphs, the most used approaches are analyzed. A more in-depth and complete analysis is also available in [11].

Among all the efforts produced, the most used technique used for implementing fault-tolerance in an application is **Checkpoint and Restart** (C/R) [1,8,9,14–18]. It consists of the periodic creation of saving points from which the execution can safely restart in case of failure. C/R is widely used in many other fields of distributed computing, like database management. It's further classified depending on the level in which the saving operation is performed: in system-level checkpointing the save operation is performed without knowing details of the running applications [1], in application-level checkpointing the programmer can specify which

		Recovery policy		
		Local recovery	Global recovery	None
Integration approach	Extension	[14] *[15]* [9] *[18]*	*[16] [8]* [17]	[13]
	Change		[1]	[7]
	None	*[4]*		[10]

Table 1: Classification of the main related works. In italics efforts using ULFM.

variables need to be saved [8,9,14–18]. The first is more general, but the latter uses less memory to achieve the same result. Almost all the solutions using C/R tend to use application-level checkpointing due to its efficiency. In a distributed environment C/R can also vary on the coordination of the saving routine and on the medium that will store the checkpoint data.

Another technique is a **log-based approach**. Its use is very low compared to C/R because the cost of adoption is too high in terms of performance [4].

An approach closer to MPI is the use of **intercommunicators**: MPI communicators are split into smaller groups and new structures are introduced to allow inter-group communication. Upon failure, only a smaller group detects the problem and has to handle with it, the others can continue their execution. This approach is adopted rarely because it doesn't solve the entire problem: coherence is not addressed and must be considered separately [10].

Algorithm-Based Fault Tolerance (ABFT) is a very application-specific approach that consists of the exploit of some characteristics of the application algorithm to achieve fault-tolerance [6]. Its use has been explored but its strict dependence on the application makes this approach very situational.

Other approaches tackle a specific type of fault and they adopt a completely different approach with regards to the ones above [13]. These solutions, however, exploit the characteristics of the fault they consider and, as a consequence, tend to not apply to other types of faults

It shall be noted that these approaches are correlated with the classification: an application-level C/R approach cannot be realized with a change of the MPI standard due to the necessity of additional information unavailable in a transparent approach. All the efforts try to evaluate all the different tradeoffs and choose the solution based on the target application.

2.3. Discussion

Fault tolerance in HPC received a lot of attention in recent conferences, leading to many efforts that widely described the problem and possible approaches. ULFM is a valuable tool that provides implementations of useful algorithms but needs proper knowledge to achieve anything. Its actual use depends on the application and its characteristics. Many programs don't use ULFM: many don't deal with fault tolerance at all, basing themselves on the "nothing will fail" axiom. The need for the use of these applications in a fault capable environment led to the creation of tools that transparently implemented fault tolerance (changing the MPI standard).

As was mentioned before, exploiting application characteristics can lead to more efficient results than a general approach. This direction is suggested by many efforts that aimed at the adaptation of a single application. This proves that, despite the presence of general-purpose fault-tolerance tools like Fenix or CPPC, a more specific approach is relevant.

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] Amarasinghe, S., Campbell, D., Carlson, W., Chien, A., Dally, W., Elnohazy, E., Hall, M., Harrison, R., Harrod, W., Hill, K., et al. Exascale software study: Software challenges in extreme scale systems. *DARPA IPTO, Air Force Research Labs, Tech. Rep* (2009), 1–153.

[3] 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.

[4] Dichev, K., Cameron, K., and Nikolopoulos, D. S. Energy-efficient localised rollback via data flow analysis and frequency scaling. In *Proceedings of the 25th European MPI Users' Group Meeting* (2018), pp. 1–11.

[5] Dongarra, J., Beckman, P., Moore, T., Aerts, P., Aloisio, G., Andre, J.-C., Barkai, D., Berthou, J.-Y., Boku, T., Braunschweig, B., et al. The international exascale software project roadmap. *The international journal of high performance computing applications 25*, 1 (2011), 3–60.

[6] 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.

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

[8] 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.

[9] 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.

[10] Gropp, W., and Lusk, E. Fault tolerance in message passing interface programs. *The International Journal of High Performance Computing Applications 18*, 3 (2004), 363–372.

[11] Herault, T., and Robert, Y. *Fault-tolerance techniques for high-performance computing*. Springer, 2015.

[12] Hillis, W. D., and Steele, G. L. Data parallel algorithms. *Commun. ACM 29*, 12 (Dec. 1986), 1170–1183.

[13] Kalim, U., Gardner, M. K., and Feng, W. A non-invasive approach for realizing resilience in mpi. In *Proceedings of the 2017 Workshop on Fault-Tolerance for HPC at Extreme Scale* (2017), pp. 1–8.

[14] 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.

[15] Losada, N., Bosilca, G., Bouteiller, A., González, P., and Martín, M. J. Local rollback for resilient mpi applications with application-level checkpointing and message logging. *Future Generation Computer Systems 91* (2019), 450–464.

[16] 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.

[17] Sultana, N., Skjellum, A., Laguna, I., Farmer, M. S., Mohror, K., and Emani, M. Mpi stages: Checkpointing mpi state for bulk synchronous applications. In *Proceedings of the 25th European MPI Users' Group Meeting* (2018), pp. 1–11.

[18] 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.

[19] The MPI Forum, C. Mpi: A message passing interface. In *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing* (New York, NY, USA, 1993), Supercomputing '93, Association for Computing Machinery, p. 878–883.

[20] Zheng, G., Ni, X., and Kalé, L. V. A scalable double in-memory checkpoint and restart scheme towards exascale. In *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)* (2012), IEEE, pp. 1–6.