Epidemic Fault Tolerance for Extreme-Scale Parallel Computing

Amogh Katti(✉) and Giuseppe Di Fatta

School of Systems Engineering, University of Reading,
Whiteknights, Reading, Berkshire RG6 6AY, UK
{a.p.katti,g.difatta}@reading.ac.uk

Abstract. Process failure rate in the next generation of high performance computing systems is expected to be very high. MPI Forum is working on providing semantics and support for fault tolerance. Run-Through Stabilization, User-Level Failure Mitigation and Process Recovery proposals are the resulting endeavors. Run-Through Stabilization/User Level Failure Mitigation proposals require a fault tolerant failure detection and consensus algorithm to inform the application of failures so that it can employ Algorithm Based Fault Tolerance for quicker recovery and continued execution. This paper discusses the proposals in short, the failure detectors available in the literature and their unsuitability for realizing fault tolerance in MPI. It then outlines an inherently fault-tolerant and scalable Epidemic (or Gossip-based) approach for failure detection and consensus. Some simulations and an initial experimental analysis are presented, which indicate that this is a promising research direction.

Keywords: Fault tolerance · Message Passing Interface (MPI) · Failure detection · Epidemic protocols · Gossip-based protocols

1 Introduction

Future High Performance Computing (HPC) systems will be prone to frequent failures. The System Mean Time Between Failures (SMTBF) for these systems is estimated to be approximately equal to an hour or even less [19] in contrast to the SMTBF of five to six hours observed for current HPC systems [21].

Checkpoint/Restart is a generic fault tolerance technique, wherein the application state is restored from the last saved checkpoint during recovery, that can be used with all kinds of High End Computing (HEC) applications and hence it is the prominent fault tolerance technique in use; it is the only technique available in most of the commercial HEC deployments. However, the technique is deemed to be ineffective for extreme-scale systems due to the high recovery time associated with it [6, 17].

Application specific techniques like Algorithm Based Fault Tolerance (ABFT) [11] are recommended for extreme-scale systems [7] for their efficiency in terms of resource and energy utilization and high performance. ABFT is a technique wherein the fault tolerance logic is embedded in the algorithm by the application developer to deal with the loss of application state at failure. This reduces recovery time thereby increasing

G. Di Fatta et al. (Eds.): IDCS 2015, LNCS 9258, pp. 201–208, 2015.
DOI: 10.1007/978-3-319-23237-9_18

efficiency. Applications typically use data encoding, algorithm redesign, diskless checkpointing, etc. ABFT techniques for recovery when failures occur.

Failure detection and notification support from the underlying programming library is required for applications to employ ABFT. Therefore the Message Passing Interface's (MPI) [13], the dominant parallel programming interface, Fault Tolerance Working Group (FTWG) is working on providing failure detection and notification and recovery services to applications to enable ABFT. Run-Through Stabilization (RTS) / User-Level Failure Mitigation (ULFM) proposal in combination with Process Recovery proposal provide the fault tolerance semantics and interfaces to serve these purposes.

In this paper a promising research direction for this problem is presented. The proposed approach is based on Epidemic (or Gossip-based) protocols to implement a failure detector for extreme-scale parallel computing.

Uniform Gossip is an inherently fault tolerant and highly scalable communication scheme. It is aptly suitable for information dissemination and data aggregation in large scale, distributed and fault prone networked systems [3, 8]. Recently, they have also been adopted in high performance computing tasks [18, 20].

The paper is organized as follows. FTWG's endeavors to make MPI fault tolerant are discussed in Sect. 2. Failure detectors available in the HPC literature are discussed in Sect. 3. Section 4 proposes a completely distributed Gossip-based and hence inherently fault tolerant failure detection and consensus approach. Simulations and an initial analysis are presented in Sect. 5. The paper concludes in Sect. 6 with a discussion of the future work to comprehensively realize scalable fault tolerance in extreme-scale parallel computing.

2 Fault Tolerance in MPI

MPI's FTWG proposed RTS proposal to define semantics and interfaces to allow an application execute uninterrupted despite the occurrence of faults. ULFM proposal replaces the RTS proposal. Process Recovery proposal allows failed processes to re-join. Only fail-stop (crash) process failures are considered by these proposals. When a process crashes it stops communicating with rest of the processes. The three proposals are briefly discussed in this section.

According to the RTS proposal [9], an implementation is expected to inform an application of all process failures and let it run using the fault-free processes. RTS expects an eventually perfect failure detector [5] that is both strongly accurate and strongly complete. Strong accuracy means that a process must not be reported failed before it actually fails and strong completeness means that every failed process must be known to every fault-free process. The proposal weakens the completeness requirement to allow the processes to return different failed processes by the end of failure detection.

The RTS proposal has been suspended because of the implementation complexity of the failure detection and notification mechanisms involved [2]. User-Level Failure Mitigation (ULFM) proposal [1] supersedes the RTS proposal. Under the ULFM proposal, no operation hangs in the presence of failures but completes by returning an error. Asynchronous failure notification is not necessary. The proposal demands a

weakly complete failure detector to achieve global consistency on the set of failed processes whenever necessary.

Process Recovery proposal [15] complements the RTS/ULFM proposal. It provides semantics and interfaces to facilitate recovery of a process that failed previously. Draft specification for the proposal is under development.

3 Failure Detectors

MPI requires failure detection and notification services to enable ABFT. Both centralized and completely distributed failure detectors are available in the HPC literature. Coordinator based and completely distributed Gossip-based failure detectors for fail-stop failures are discussed in this section.

3.1 Coordinator Based Failure Detectors

A two-phase fault-aware consensus algorithm over a static tree communication topology to construct a weekly complete failure detector was provided in [12]. A fault tolerant algorithm, in [4], provided an improvement to support strict completeness using an iterative formulation of the three-phase commit over a dynamic tree communication topology. Both the approaches are discussed in this section.

Over a Static Tree Topology. This approach assumes that processes locally know failed processes and participate in the consensus algorithm to consistently construct the global list of failed processes. A two-phase algorithm over a fault-aware tree topology constructs the global list of failed processes using reliable gather at the coordinator during the first phase and reliable broadcast to the participant processes during the broadcast phase. Participant failures are handles by routing around the failed processes to find the nearest parent and child process during the gather and broadcast operations respectively. Termination detection algorithm is used when the coordinator fails during the broadcast phase. Processes query the immediate children of the coordinator to get the global list of failed processes. If the coordinator fails during the gather phase or just before the broadcast phase, the algorithm aborts without constructing the global list of failed processes. Processes that fail during the algorithm will be detected during the next invocation of the algorithm.

Over a Dynamic Tree Topology. This approach also assumes that processes locally know failed processes and then participate in the consensus algorithm. A three-phase algorithm over a fault-tolerant dynamic tree topology constructs the global list of failed processes making sure that every process returns the same list of failed processes and thus implements a strongly complete failure detector. First phase constructs the list of failed processes and sends it to every participant and makes sure that every process has the same list of failed processes by the end of the phase, second phase informs to the participants that all the processes have the same failed process list by now and third phase commands the participants to terminate the algorithm. Every phase starts with a message from the coordinator and finishes when the coordinator receives acknowledgement from

all the participants for the current phase. If any participant fails during a phase, a new instance of the broadcast starts by reconstructing the tree with the current alive processes. Coordinator failure is handled by electing a new coordinator.

3.2 Completely Distributed Failure Detectors

Coordinator based failure detection and consensus algorithms do not scale to large number of processes. Completely distributed failure detection can be accomplished as a side effect of Gossiping. Gossip-based failure detectors in the distributed computing systems literature considered for HPC are discussed in this section.

Gossip-based failure detectors can be either passive "heartbeat" failure detector or active "ping" failure detector. A process in "heartbeat" failure detection passively waits for Gossip messages whereas in "ping" failure detection a process actively pings other processes.

"Heartbeat" Failure Detector. In [16] a Gossip-based failure detection algorithm using liveness analysis is given. A process in the system periodically announces that it is alive by sending a Gossip message to another random process in the system. This liveness information disseminates throughout the network and ultimately every process will have information about every other process in the system. A process is suspected to have failed if its liveness information is old. When a majority of processes suspect a process it is detected to have failed. When all fault free processes have detected a faulty process consensus on its failure is reached.

"Ping" Failure Detector. A failure detection algorithm using distributed diagnosis considering network partitioning is given in [10]. A process randomly selects another process and pings it to find its status. If it does not receive a response from the process, it asks a random sample of the processes in the system to ping the process as well. The process is detected to have failed if none of the selected processes receives a response.

4 Failure Detector Maintaining Global Knowledge

Completely distributed Gossip-based heartbeat failure detection and consensus algorithms are based on passive and slow liveness analysis and consume very high memory and network bandwidth. There is need for fault tolerant yet scalable communication schemes. In this section a novel scalable Gossip-based and inherently fault tolerant ping type failure detector for fail-stop failures using a matrix to store global view of all the processes in the system is proposed.

The algorithm detects fail-stop failures and the failures are assumed to be permanent. A synchronous model of the system is assumed with bounded message delay. Failures during the algorithm are assumed to stop at some point to allow the algorithm to complete with successful consensus detection. Figure 1 shows pseudocode for the algorithm.

```
At each process p
  Require: Fault Matrix Fp[r,c] where 0≤r,c<n
     Fp[r,c] - system view of processes
  Initialisation:
    1. for (r=0, r<n, r++)           //start with all alive processes
    2.    for (c=0, c<n, c++)
    3.       Fp[r,c]=0
    4.    end for
    5. end for
  At each cycle:
    6. q = getRandomProcess()
    7. send a PING message to a random process, say q, with Fp
    8. for (c=0, c<n, c++)                    //consensus check on c
    9.   cnt = 0
   10.   for (r=0, r<n, r++)
   11.      if (Fp[r,c] || Fp[p,r]) then
   12.        cnt++
   13.      end if
   14.   end for
   15.   if (cnt==n) then
   16.     consensus_reached(c)
   17.   end if
   18. end for
At event: received a message from j with Fj
   19. if(message == PING) then            //reply to the PING
   20.    send a REPLY message to j with Fp
   21. end if
   22. for (r=0, r<n, r++)           //merge the detections
   23.    for (c=0, c<n, c++)
   24.      Fp[r,c]= Fp[r,c]|| Fj[r,c]        //propagate failure detections
   25.      Fp[p,c]= Fp[p,c]|| Fj[j,c]        //indirect failure detections
   26.    end for
   27. end for
At event: timeout without receiving REPLY from q
   28. Fp[p,q]=1                             //direct failure detection
```

Fig. 1. Pseudocode of the Gossip-based failure detection and consensus

A process p maintains a fault matrix F_p to store the system view of all the processes in the system. $F_p[r, c]$ is the view at process p of the status of process c as detected by process r. A value of *1* indicates failure and a *0* indicates alive.

Every process in the system is assumed to be alive by every process at the beginning and hence the fault matrix is initialized with all *0*'s (lines 1-5).

During a cycle of Gossip, of length T_{gossip} time units, process p pings a random process to check its status. It also handles reception of Gossip message and ping timeout events. A random process q is selected and a ping message is sent to it with the local fault matrix F_p (lines 6-7). When a ping message is received, an asynchronous reply is sent with the local fault matrix (lines 19-21). When the ping message times out without receiving a reply message from q, it is detected to have failed and *1* is stored at $F_p[p, q]$ (line 28). On receiving a Gossip message from j, the local and the remote fault matrices, F_p and F_j, are merged. Thus process p performs indirect failure detection through j and propagates the failures known to j (lines 22-27).

Consensus on the failure of each process is checked during every Gossip cycle. Consensus is reached when all the fault-free processes have recognized the failed process (lines 8-18).

5 Simulations and Results

The algorithm was implemented in Java and the simulations were carried out on PeerSim [14], a scalable network simulator based on discrete events. The latency and bandwidth were set to nominal values as only the number of Gossip cycles required to reach consensus were measured. Failures were simulated by restraining a process from participating in communications.

The algorithm's scalability and fault tolerance properties were tested. Failures were injected into randomly chosen processes. In the first experiment a single failure was injected at the beginning of the simulation. In the second experiment failures were injected during the simulation to test the fault tolerance property of the algorithm. Because processes reach consensus on the injected failure(s) at different cycles, the cycle number of the last process reaching consensus is considered and recorded.

Figure 2 shows the relationship between the number of Gossip cycles (average over multiple simulations) and system size to reach consensus when a single failure is injected at the beginning of the simulation. Consensus is reached in logarithmic number of Gossip cycles.

Figure 3 shows the transition towards consensus in terms of the relative number of processes which have detected the failure at each cycle. A typical epidemic information spreading can be observed.

The consensus algorithm is completely fault tolerant and it can also detect failures that happen during its execution. Figure 4 shows the results of simulations where failures were injected in randomly chosen processes and at random time within the first 10 cycles. The number of Gossip cycles needed to achieve consensus is still logarithm in terms of the system size from the Gossip cycle at which the last failure was injected.

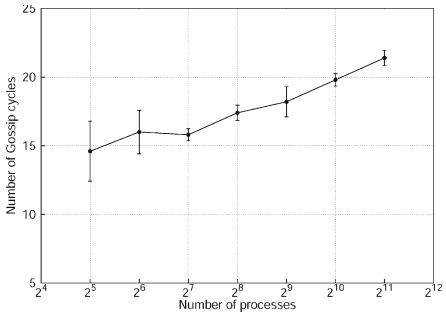

Fig. 2. Number of cycles to achieve consensus with a single failure

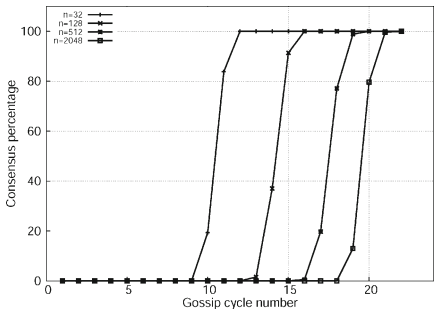

Fig. 3. Transition towards consensus with a single failure

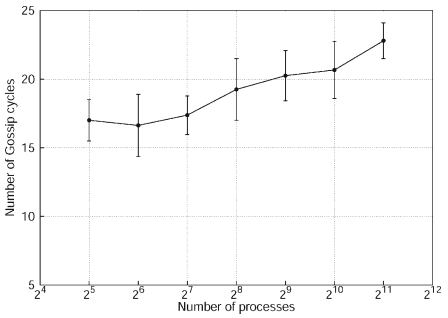

Fig. 4. Number of cycles to consensus with 4 failures injected during the simulations

6 Conclusion and Future Work

MPI's Fault Tolerance Working Group is working on including fault tolerance support into the standard to enable high performance computing systems to continue execution despite faults. Algorithm Based Fault Tolerance is the fault tolerance technique sought of and it requires failure detection and notification services.

Failure detection and consensus methods that use a coordinator do not scale to large number of processes. To overcome these limitations, this work has introduced a Gossip-based approach to provide scalable and fault tolerant failure detection and consensus. Each process builds and propagates a global view of the system. Failures are locally detected with direct timeout events based on Gossip messages and with indirect propagation of failures known to other processes. The experimental analysis based on simulations have shown that consensus on failures is reached in a logarithmic number of Gossip cycles w.r.t. the system size.

However, the proposed approach does not scale well in terms of memory requirements because each process has to maintain not only its own view of the system but also the views of all other processes. It also consumes a lot of network bandwidth due to transfer of this global view with each Gossip message.

Future work includes the design of memory and network bandwidth efficient methods for fault tolerant failure detection and consensus. In particular, fully decentralised algorithms for consensus detection and synchronization are being investigated. Supporting process re-spawning in the algorithm thereby bridging failure detection and process recovery is also an interesting future research direction.

References

1. Bland, W., Bosilca, G., Bouteiller, A., Herault, T., Dongarra, J.: A proposal for User-Level Failure Mitigation in the MPI-3 standard. University of Tennessee, Department of Electrical Engineering and Computer Science (2012)
2. Bland, W., Bouteiller, A., Herault, T., Bosilca, G., Dongarra, J.J.: Post-failure recovery of MPI communication capability: Design and rationale. Int. J. High Perform. Comput. Appl. (2013)

3. Blasa, F., Cafiero, S., Fortino, G., Di Fatta, G.: Symmetric push-sum protocol for decentralised aggregation (2011)
4. Buntinas, D.: Scalable distributed consensus to support MPI fault tolerance. In: 26th IEEE International Conference on Parallel & Distributed Processing Symposium (IPDPS), May 2012, pp. 1240–1249 (2012)
5. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM (JACM) **43**(2), 225–267 (1996)
6. Daly, J.T., Lead, R.: Application resilience for truculent systems. In: Workshop on Fault Tolerance for Extreme-Scale Computing, Albuquerque, NM – 19–20 March 2009, ANL/MCS-TM-312 (2009)
7. Daly, J., Harrod, B., Hoang, T., Nowell, L., Adolf, B., Borkar, S., Wu, J.: Inter-Agency Workshop on HPC resilience at extreme scale. In: National Security Agency Advanced Computing Systems, February 2012 (2012)
8. Di Fatta, G., Blasa, F., Cafiero, S., Fortino, G.: Fault tolerant decentralised K-Means clustering for asynchronous large-scale networks. J. Parallel Distrib. Comput. **73**(3), 317–329 (2013)
9. Fault Tolerance Working Group. Run-though stabilization interfaces and semantics. In: svn. mpi-forum. org/trac/mpi-forum-web/wiki/ft/run through stabilization (2012)
10. Gupta, I., Chandra, T.D., Goldszmidt, G.S.: On scalable and efficient distributed failure detectors. In: Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing, August 2001, pp. 170–179. ACM (2001)
11. Huang, K.H., Abraham, J.A.: Algorithm-based fault tolerance for matrix operations. IEEE Trans. Comput. **100**(6), 518–528 (1984)
12. Hursey, J., Naughton, T., Vallee, G., Graham, R.L.: A log-scaling fault tolerant agreement algorithm for a fault tolerant MPI. In: Cotronis, Y., Danalis, A., Nikolopoulos, D.S., Dongarra, J. (eds.) EuroMPI 2011. LNCS, vol. 6960, pp. 255–263. Springer, Heidelberg (2011)
13. Message Passing Interface Forum: MPI: A Message Passing Interface. In: Proceedings of Supercomputing 1993, pp. 878–883. IEEE Computer Society Press (1993)
14. Montresor, A., Jelasity, M.: PeerSim: A scalable P2P simulator. In: IEEE Ninth International Conference on Peer-to-Peer Computing, P2P 2009, pp. 99–100. IEEE (2009)
15. Process Recovery Proposal. https://svn.mpi-forum.org/trac/mpi-forum-web/wiki/ft/process_recovery_2. Accessed: 14 May 2015
16. Ranganathan, S., George, A.D., Todd, R.W., Chidester, M.C.: Gossip-style failure detection and distributed consensus for scalable heterogeneous clusters. Cluster Comput. **4**(3), 197–209 (2001)
17. Schroeder, B., Gibson, G.A.: Understanding failures in petascale computers. In: Journal of Physics: Conference Series, vol. 78(1), p. 012022. IOP Publishing, July 2007
18. Soltero, P., Bridges, P., Arnold, D., Lang, M.: A Gossip-based approach to exascale system services. In: Proceedings of the 3rd International Workshop on Runtime and Operating Systems for Supercomputers, p. 3. ACM, June 2013
19. Song, H., Leangsuksun, C., Nassar, R., Gottumukkala, N.R., Scott, S.: Availability modeling and analysis on high performance cluster computing systems. In: The First International Conference on Availability, Reliability and Security, ARES 2006, April 2006, p.8. IEEE (2006)
20. Straková, H., Niederbrucker, G., Gansterer, W.N.: Fault tolerance properties of gossip-based distributed orthogonal iteration methods. Procedia Comput. Sci. **18**, 189–198 (2013)
21. Taerat, N., Nakisinehaboon, N., Chandler, C., Elliot, J., Leangsuksun, C., Ostrouchov, G., Scott, S.L.: Using log information to perform statistical analysis on failures encountered by large-scale HPC deployments. In: Proceedings of the 2008 High Availability and Performance Computing Workshop, vol. 4, pp. 29–43 (2008)