Job Level Communication-Avoiding Detection and Correction of Silent Data Corruption in HPC Applications

Laslo Hunhold
hunhold@uni-koeln.de
University of Cologne
Parallel and Distributed Systems Group
Cologne, Germany

Stefan Wesner*
wesner@uni-koeln.de
University of Cologne
Parallel and Distributed Systems Group
Cologne, Germany

ABSTRACT

Detecting and correcting Silent Data Corruption (SDC) is of high interest for many HPC applications due to the dramatic consequences such undetected computation errors can have. Additionally, going into the exascale era of computing, SDC error rates are only increasing with growing system sizes. State of the art methods based on instruction duplication suffer from only partial error coverage, significant synchronization overhead and strong coupling of computation and validation.

This work proposes a novel communication-avoiding approach of detecting and mitigating SDCs at the job level within the workload manager, assuming a directed acyclic graph (DAG) job model. Each job only communicates a locally generated output data hash. Computation and validation are decoupled as separately schedulable jobs and dependency stalling is avoided with a special error recovery method. The implementation of this project as a SLURM plugin is in progress and key design aspects are outlined.

CCS CONCEPTS

• **Computer systems organization** → *Redundancy*; • **Software and its engineering** → Ultra-large-scale systems; **Error handling and recovery**; • **Computing methodologies** → *Parallel computing methodologies*.

1 INTRODUCTION

Silent data corruption (SDC) is a computing error that is undetected during execution. The causes of SDCs are manifold and can range from undetected hardware defects to single-event upsets (SEUs), where an ionizing particle affects a transistor. SDCs can have dramatic consequences for many HPC applications and there is a big interest in detecting and correcting them, especially with rising SDC error rates due to ever-increasing system sizes.

In this work we will only focus on SDC detection and correction methods which are based on duplicate execution of a given program/instruction. The alternatives are usually domain-specific, as they depend on a-priori knowledge of solution properties[1]. Error prediction[2][5] and selective duplication[4] will also be out of scope given they only provide partial error coverage. The most common approach is to observe MPI communication between processes and compare it between two identical runs[3]. This implies a significant synchronization overhead due to stalling, which can be mitigated by centrally storing the MPI messages and postponing validation at the cost of communication overhead[7]. Another approach is to abstract task duplication via MPI ranks[6], which however only provides partial error coverage.

Common to all approaches from the perspective of the workload manager is that they are job-local. Particularly, computation and validation are strongly coupled at the job level. This implies that one has to reserve twice the resources for a job in the workload manager and subsequent jobs that depend on the job have to wait for both computation and validation to finish.

2 GOALS

This work aims for a method to detect and correct SDCs at the job level within the workload manager with full error coverage, decoupling computation and validation. The proposed method should be easy to implement for end users.

3 METHODOLOGY

We assume a directed acyclic graph (DAG) job model and each job j to be reschedulable and idempotent. The latter assumption is not as restrictive as it sounds, as you can even include Monte-Carlo-type simulations with identically seeded PRNGs. Under these assumptions for j, job validation can be achieved by comparing hashes of the output data (majority vote), which minimizes communication overhead.

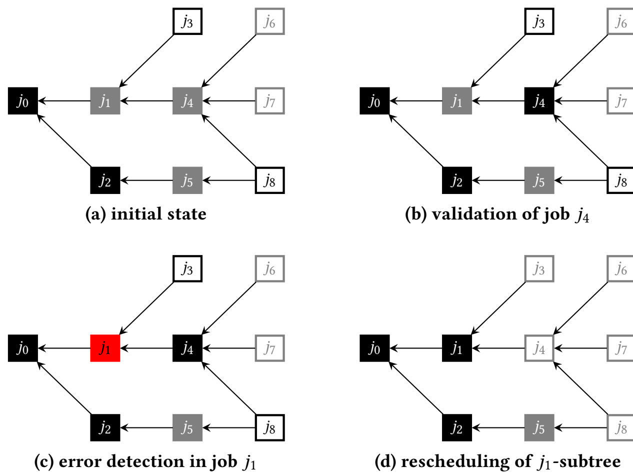
The jobs are validated as follows: For each job j, a separate identical validation job j' no other job depends on is scheduled on nodes not used by j. No dependency of j waits for j' to finish. When both j and j' have finished, the output data hashes of j and j' are compared. If they match, the job j is validated, otherwise new identical validation jobs are launched until two hashes agree. If j's output data hash is one of the two matching hashes, it is marked as valid. Otherwise the jobs in the subtree of j are rescheduled (see Figure 1).

4 IMPLEMENTATION

The proposed scheme is implemented as a SLURM workload manager plugin. SLURM is chosen for the implementation given it is the most popular workload manager in the TOP500. The validation jobs are given a lower `Priority` (tunable based on the system SDC error rate and job runtime for minimal rescheduling losses) to encourage

*Adviser

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SC '23, November 12–17, Denver, CO, USA
© 2023 Association for Computing Machinery.



(a) initial state (b) validation of job j_4 (c) error detection in job j_1 (d) rescheduling of j_1-subtree

Figure 1: The process of error detection and rescheduling shown for a simple job DAG. Unfilled grey nodes represent pending jobs, unfilled black nodes represent running jobs, filled grey nodes represent finished jobs that are waiting for validation, filled black nodes represent validated jobs and filled red nodes represent invalidated jobs. Following the initial state (a), (b) shows a possible scenario where validation may happen independently from the implicit DAG hierarchy. If a job run is found to be invalid by majority vote, as shown in (c) for job j_1, it is replaced with its valid run and all jobs depending on j_1 are rescheduled to depend on the valid run.

the computation job to finish first, given subsequent jobs depend on the latter. Node overlap is avoided by setting `ExcNodeList` for the validation jobs. Validation jobs are launched by the SLURM controller daemon in a new `WorkDir` respectively. Output hash calculation is the responsibility of the job itself, minimizing communication.

The most complex aspect is rescheduling jobs following an invalidation of a job j: After inspecting the job queue (for running and pending jobs) and job accounting (for completed jobs), all jobs within the subtree of j are rescheduled with an updated dependency list reflecting new job IDs. Completed jobs are invalidated by changing the `DerivedExitCode` with `sjobexitmod(1)` and setting the appropriate `SystemComment`. Likewise, validated jobs are marked accordingly in the job accounting system.

5 CONCLUSION

To the best of the author's knowledge the proposed approaches to consider SDC detection and correction both at the job level and within a DAG job model are completely novel. The minimal amount of communication to transmit the locally generated output data hashes promises good scalability irrespective of the system's heterogenicity and the minimal necessary modifications of the SLURM batch scripts ensure good usability.

Decoupling computation and validation as separate jobs not only for each job but in terms of the entire DAG hierarchy allows more efficient scheduling and more freedom compared to a linear task model as seen with previous approaches. This is also underlined by the fact that there is no stalling overhead by jobs waiting for their dependencies to finish validating. Despite the possible rescheduling overhead in the rare failure case, validating job runs can be adaptively prioritized depending on the system's SDC error rate and rescheduling cost. This is especially relevant for complex job relationships in exascale systems.

REFERENCES

[1] Tommaso Benacchio et al. 2021. Resilience and fault tolerance in high-performance computing for numerical weather and climate prediction. *The International Journal of High Performance Computing Applications*, 35, 4, (Feb. 2021), 285–311. DOI: 10.1177/1094342021990433.

[2] Sheng Di, Eduardo Berrocal and Franck Cappello. 2015. An Efficient Silent Data Corruption Detection Method with Error-Feedback Control and Even Sampling for HPC Applications. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (2015 IEEE/ACM 15th International Symposium on Cluster, Cloud, and Grid Computing (CCGrid 2015)). IEEE, Shenzhen, China, (July 2015), 271–280. DOI: 10.1109/CCGrid.2015.17.

[3] David Fiala, Frank Mueller, Christian Engelmann, Rolf Riesen, Kurt Ferreira and Ron Brightwell. 2013. Detection and Correction of Silent Data Corruption for Large-Scale High-Performance Computing. In *SC '12* (Salt Lake City, UT, USA). IEEE, (Feb. 2013), 1–12. DOI: 10.1109/SC.2012.49.

[4] Yafan Huang, Shengjian Guo, Sheng Di, Guanpeng Li and Franck Cappello. 2023. Mitigating Silent Data Corruptions in HPC Applications across Multiple Program Inputs. In *SC '22* (Dallas, TX, USA). IEEE, (Feb. 2023), 1–14. DOI: 10.1109/SC41404.2022.00022.

[5] Sihuan Li, Sheng Di, Kai Zhao, Xin Liang, Zizhong Chen and Franck Cappello. 2020. Towards End-to-end SDC Detection for HPC Applications Equipped with Lossy Compression. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. 2020 IEEE International Conference on Cluster Computing (CLUSTER) (Kobe, Japan). IEEE, (Sept. 2020), 326–336. DOI: 10.1109/CLUSTER49012.2020.00043.

[6] P. Samfass, T. Weinzierl, A. Reinarz and M. Bader. 2021. Doubt and Redundancy Kill Soft Errors—Towards Detection and Correction of Silent Data Corruption in Task-based Numerical Software. In *2021 IEEE/ACM 11th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS)*. IEEE Computer Society, Los Alamitos, CA, USA, (Nov. 2021), 1–10. DOI: 10.1109/FTXS54580.2021.00005.

[7] Guozhen Zhang, Yi Liu, Hailong Yang and Depei Qian. 2021. Efficient detection of silent data corruption in HPC applications with synchronization-free message verification. *The Journal of Supercomputing*, 78, (June 2021), 1381–1408. DOI: 10.1007/s11227-021-03892-4.