Designing an Adaptive Application-Level Checkpoint Management System for Malleable MPI Applications

Jophin John∗, Michael Gerndt†

Chair of Computer Architecture and Parallel Systems, Technische Universität München
Garching (near Munich), Germany

Email: john@in.tum.de, gerndt@in.tum.de

Abstract—Dynamic resource management opens up numerous opportunities in High Performance Computing. It improves the system-level services as well as application performance. Checkpointing can also be deemed as a system-level service and can reap the benefits offered by dynamism. A checkpointing system can have better resource availability by integrating with a malleable resource management system. In addition to fault tolerance, the checkpointing system can cater to the data redistribution demand of malleable applications during resource change. Therefore, we propose iCheck, an adaptive application-level checkpoint management system that can efficiently utilize the system and application level dynamism to provide better checkpointing and data redistribution services to applications.

Index Terms—Fault Tolerance, Adaptive Checkpointing, RDMA, Malleability, MPI

I. INTRODUCTION

Static resource management, still predominant in High-Performance Computing, takes away many optimization opportunities otherwise catered by dynamic resource management. This static behavior is visible from system services to applications [1], [2]. The potential offered by dynamic applications using dynamic resources spans beyond better system utilization [3]–[6], and checkpointing is no exception [7]–[9]. Trends in dynamism research bring malleability [10]–[15] into MPI [16] and resource management infrastructure.

To yield the benefits of such a malleable infrastructure, the checkpointing system must be able to react to the dynamic needs of resource managers and applications by adapting itself to the changing system requirements. This can result in faster checkpoint transfers and efficient utilization of checkpointable resources (available memory, parallel file system bandwidth). Furthermore, a checkpointing system can conduct data redistribution during resource change, thereby addressing one of the key challenges in writing malleable applications.

Hence, we posit checkpointing systems as a resource and data management system that can simultaneously handle the nuances in checkpointing requirements of multiple applications and interact with resource managers to provide seamless checkpointing capabilities. Towards this, we propose iCheck, a multi-level adaptive checkpoint management system that can reconfigure its checkpointing resources on the fly and offers application-level checkpointing and data redistribution services to malleable MPI applications. This approach of application-level checkpointing as an adaptable resource management and data distribution service brings novel contributions to the checkpointing domain.

II. iCHECK DESIGN

iCheck has a core system and a library. iCheck core runs on a dynamically configurable number of dedicated nodes (iCheck nodes) of the HPC system and communicates with connected applications using the iCheck library (See Figure 1). Application checkpoints are transferred to the memory of iCheck nodes using Remote Direct Memory Access (RDMA) and later written into the Parallel File System (PFS). The core system consists of an Agent, Manager, and Controller. The agent performs the functionality of checkpoint read/write (using libfabric [17]) and data redistribution (for malleable implementations). Multiple agents can be assigned to a single application, and iCheck can dynamically change the agent count to obtain an optimum checkpoint transfer rate. The manager manages the node-level activities of the software, such as launching the agents and monitoring and predicting the node usage parameters (e.g., memory usage, bandwidth usage). The controller has a global view and performs the agent and node selection for connected applications based on the iCheck agent scheduling policies. These policies consider various system metrics (available memory, checkpoint frequency and size, and bandwidth usage) and can impact the overall checkpointing performance. The controller may also request the resource manager for additional resources based on resource availability. In addition, the controller will also orchestrate the writing of the checkpoint data into PFS by minimizing the effect on running applications.

The iCheck core and iCheck library components in an application continually interact during the application life cycle. A generic workflow of these interactions is as follows:

1) An application registers itself with the controller.
2) The controller decides the number of agents and the iCheck nodes in which they will be launched.
3) The controller contacts the corresponding managers with the agent information.
4) The managers launch the agents in iCheck nodes and notify the controller.
5) The agents wait for the application processes to connect.
6) The controller provides the agent information to the application.
7) The application then connects with the agent.
8) The application and the agents register memory for RDMA.
9) The application and the agents perform checkpoint transfer operations.
10) Application finishes either with or without error.

During the restart:
1) The application contacts the controller for checkpoint information.
2) The controller passes the agent information.
3) The application can either start by using the checkpoint from the agents or start new.

During the data redistribution:
1) The application contacts the agent for data transfer.
2) The agents redistribute the data based on the mapping provided by the application.

iCheck, with its agent-based checkpoint retrieval model, easily integrates the asynchronous checkpoint retrieval capability in its library. Since the agents use RDMA, the application does not need to block for data transfer rather it can continue the execution immediately after notifying the agents about the checkpoints.

III. iCHECK AND MALLEABILITY

iCheck is malleable both at the system level and application level. At the system level, iCheck can horizontally scale its resources (available memory, number of agents) by attaching/removing iCheck nodes with the help of a resource manager. At the application level, iCheck can dynamically reconfigure the number of agents associated with connected applications.

A. iCheck and Malleable Infrastructure

We have used the malleable infrastructure from the InvisiC project [5], [11], [18] for our investigation into designing a malleable checkpointing system. It consists of an MPI implementation (an extension of MPICH with four additional methods) that can change the number of processes during the application execution and a malleable resource manager (extension of the Slurm [19]) that can orchestrate the resource changes in MPI.

To support malleable MPI applications, the application-level checkpointing systems should be able to handle the changes in resources, i.e., whenever new processes are added, they should be able to reinitialize the system (most of the application-level checkpointing systems [20]–[24] needs to be modified). iCheck provides this capability out-of-the-box. Whenever a new set of processes are added to the application, iCheck can add new agents, thereby maintaining the checkpointing performance or reinitializing with the same agents. Additionally, using special APIs (icheck_addadapt(), icheck_redistribute()) as in Listing 1, users can provide their data distribution mapping that can be used by iCheck when a resource change happens. iCheck also provides a special API call icheck_probe_agents() (line 29 in Listing 1) that can be used to query for agent change to improve the checkpointing time.

To facilitate system-level malleability in iCheck, we have created an iCheck-aware job scheduling plugin in the resource manager (RM). The following interactions with iCheck are supported by the new plugin.\(^1\) RM can give resources (iCheck nodes) to the iCheck based on the request and availability. For example, when iCheck runs out of memory in a node, the controller can request more memory and get additional nodes from RM.\(^2\) RM can retake nodes from iCheck. For example, to support resource requirements of a priority job or to satisfy power requirements.\(^3\) RM can ask the controller to migrate resources to a different iCheck node.\(^4\) RM can pass application-specific information to the controller. For example, it can inform the controller about an impending resource change of an application so that agents can prepare for data redistribution ahead of time.

B. iCheck library and a simple malleable MPI application

The pseudocode of a naive iCheck enabled malleable MPI application is shown in Listing 1. The four malleable MPI routines behave as follows. MPI_Init_adapt() (line 3) will register the application with the resource manager and returns the type of the MPI process. A process can be initial (created as part of an initial application launch) or joining (created as part of an application expansion). Initial processes continue the application execution (skipping lines 10-15) and regularly call MPI_Probe_adapt() (line 17) to check for any resource change. On the other hand, joining processes immediately calls the MPI_Comm_adapt_begin() (line 11) collective function and waits for initial processes to join. Meanwhile, the MPI_Probe_adapt() (in line 17) call informs initial processes about the resource change triggered by the malleable resource manager. Initial processes, then calls the collective MPI_Comm_adapt_begin() function (line 19). Once both the initial and joining pro-
I n i t
adapt
adapt
adapt)
Comm
restart();
commit();
available &&
agents();
commit();
add
commit();
Comm
adapt
adapt(
probe
begin(. . .)
Comm
size, BLOCK)
redistribute(
begin(. . .)
adapt
Probe
adapt(. . ., type)
size, BLOCK)
Comm
redistribute(
Finalize();
change, . . .)

There are a plethora of checkpointing works based on
different categories like application-level or system-level,
process recovery or data recovery, multilevel or single-level,
coordinated or uncoordinated, and blocking or non-blocking
[20–38]. Here we focus on works closest to iCheck.

iCheck is an adaptive asynchronous multilevel application-
level in-memory checkpointing system using RDMA that
provides data distribution service and fault tolerance. To the
best of our knowledge, no single system exists with all the above
characteristics for application-level checkpointing. Existing
application-level checkpointing libraries (for example, [21–
23]) cannot reconfigure their resources across multiple applica-
tions to optimize their checkpointing activity. In contrast,
iCheck can centrally manage checkpoints and improve the
overall checkpointing performance by dynamically scaling its
resources.

Regarding multilevel RDMA capability in checkpointing,
the closest work to iCheck is by Sato et al. [24]. iCheck differs
in the following aspects. iCheck can simultaneously cater to
the needs of multiple applications and reconfigure its resources
for RDMA capability on the fly. Further, iCheck does not use
extra processes inside the compute nodes of applications to
transfer the checkpoint.

VeloC [20] is a related work to iCheck with regards to the
adaptivity in checkpointing. It is an adaptive asynchronous
checkpointing system where adaptivity refers to efficiently
selecting the underlying checkpoint storage based on the
available heterogeneous solutions. In iCheck, adaptivity refers
to the flexibility in scaling the resources of our system.

V. CONCLUSION AND FUTURE WORK

Our current work on iCheck focuses on analyzing the
malleable characteristics of complex applications and adapting
iCheck to support the intricacies of data distribution in such
applications. The current naive data redistribution schemes
in iCheck are insufficient to utilize malleability in complex
applications (for example, data redistribution inside molecular
dynamics applications during a resource change).

Furthermore, we intend to extend the current iCheck-aware
job scheduling plugin inside RM. The current experimental
plugin prioritizes iCheck and will allocate nodes to iCheck
based on the controller’s request and node availability. We will
extend it by incorporating a malleable resource management
policy that removes nodes from running applications based on
their performance [18], ensuring fairness to running jobs and
jobs waiting in the queue.

Our prototype implementation is currently deployed on a
virtual cluster and analyzed with synthetic malleable applica-
tions. We plan to evaluate iCheck on SuperMUC-NG [39] with
real scientific applications developed using malleable MPI.

VI. ACKNOWLEDGMENT

The research leading to these results was funded by
the Deutsche Forschungsgemeinschaft (DFG, German Re-
search Foundation)-Projektnummer 146371743-TRR 89: Inva-
sive Computing.
REFERENCES

[1] “Jsc.” [Online]. Available: https://hbp-hpc-platform fz-juelich.de/?page=1836

[2] D. H. Ahn, J. Garlick, M. Grondona, D. Lipari, B. Springmeyer, and M. Schulz, “Flux: a next-generation resource management framework for large hpc centers,” in 2014 43rd International Conference on Parallel Processing Workshops. IEEE, 2014, pp. 9–17.

[3] T. Desell, K. Maghraoui, and C. Varela, “Malleable applications for scalable high performance computing,” Cluster Computing, vol. 10, pp. 323–337, 09 2007.

[4] M. Chadha, “Adaptive resource-aware batch scheduling for hpc systems,”

[5] J. John, S. Narváez, and M. Gerndt, “Invasive computing for power corridor management,” Parallel Computing: Technology Trends, vol. 36, p. 386, 2020.

[6] M. Abhishek, “Dynamic allocation of high performance computing resources,” International Journal of Advanced Trends in Computer Science and Engineering, vol. 9, pp. 3538–3543, 06 2020.

[7] C. George and S. S. Vadhiyar, “Adft: An adaptive framework for fault tolerance on large scale systems using application malleability,” Procedia Computer Science, vol. 9, pp. 166–175, 2012, proceedings of the International Conference on Computational Science, ICCS 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050912001391

[8] K. El Maghraoui, B. K. Szymanski, and C. Varela, “An architecture for reconfigurable iterative mpi applications in dynamic environments,” in 1st Parallel Processing and Applied Mathematics, R. Wyrzykowski, J. Dongarra, N. Meyer, and J. Waśniewski,Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 258–271.

[9] J. Posner, “Load balancing, fault tolerance, and resource elasticity for asynchronous many-task systems,” Ph.D. dissertation, Kassel, Universität Kassel, Fachbereich Elektrotechnik / Informatik, 2021.

[10] I. A. Comprés Ureña and M. Gerndt, “Towards elastic resource management,” in Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC, ser. SC ’10. USA: IEEE Computer Society, 2010, p. 1–11. [Online]. Available: https://doi.org/10.1109/SC.2010.18

[11] K. Sato, K. Mohror, A. Moody, T. Gamblin, B. Supinski, N. Maruyama, and S. Matsuoka, “Design and modeling of a non-blocking checkpointing system,” 11 2012, pp. 1–10.

[12] W. Bland, A. Bouteiller, T. Herault, G. Bosilca, and J. Dongarra, “Post-failure recovery of mpi communication capability: Design and rationale,” The International Journal of High Performance Computing Applications, vol. 27, no. 3, pp. 244–254, 2013. [Online]. Available: https://doi.org/10.1177/1094342013488238

[13] J. Stearley, J. Laros, K. Ferreira, K. Pedretti, R. Oldfield, R. Riesen, and R. Brightwell, “RMPI: increasing fault resiliency in a message-passing environment,” 04 2011.

[14] G. Zheng, L. Shi, and L. V. Kale, “Fic-charm++: An in-memory checkpoint-based fault tolerant runtime for charm++ and mpi,” in Proceedings of the 2004 IEEE International Conference on Cluster Computing, ser. CLUSTER ’04. USA: IEEE Computer Society, 2004, p. 93–103.

[15] G. Fagg and J. Dongarra, “Ft-mpi: Fault tolerant mpi, supporting dynamic applications in a dynamic world,” in Proceedings of the 7th European PVM/ MPI Users’ Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface. Berlin, Heidelberg: Springer-Verlag, 2000, p. 346–353.

[16] G. Rodriguez, M. Martin, P. Gonzalez, J. Tourinho, and R. Doallo, “Cppc: A compiler-assisted tool for portable checkpointing of message-passing applications,” Concurrency and Computation: Practice and Experience, vol. 22, pp. 749–766, 04 2010.

[17] S. Chakraborty, I. Laguna, M. Emani, K. Mohror, D. K. Panda, M. Schulz, and H. Subramoni, “Ereinit: Scalable and efficient fault-tolerance for bulk-synchronous mpi applications,” Concurrency and Computation: Practice and Experience, vol. 32, no. 3, p. e4863, 2020, e4863 1–11. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4863

[18] N. Sultana, A. Skjellum, I. Laguna, M. Farmer, K. Mohror, and M. Emani, “Mpi states: Checkpointing mpi state for bulk synchronous applications,” 09 2018, pp. 1–11.

[19] J. Ramkumar and V. Strumpen, “Portable checkpointing for heterogeneous architectures,” 07 1997, pp. 58 – 67.

[20] R. Arora, P. Bangalore, and M. Mernik, “A technique for non-invasive application-level checkpointing,” The Journal of Supercomputing, vol. 57, pp. 227–255, 09 2010.

[21] R. Arora and T. N. Ba, “Italc: Interactive tool for application-level checkpointing,” in Proceedings of the Fourth International Workshop on HPC User Support Tools, ser. HUST’17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3152493.3152558

[22] J. Plank, M. Beck, G. Kingsley, and K. Li, “Libckpt: Transparent checkpointing under unix,” TCON, 12 1995.

[23] T. Islam, K. Mohror, S. Bagchi, A. Moody, B. Supinski, and R. Eigenmann, “Mzrgenie: A scalable checkpointing system using data-aware aggregation and compression,” Scientific Programming, vol. 21, 01 2013.

[24] K. Sato, K. Mohror, A. Moody, T. Gamblin, B. Supinski, N. Maruyama, and S. Matsuoka, “A user-level infiniband-based file system and checkpoint strategy for burst buffers,” 05 2014, pp. 21–30.

[25] D. Kimpe, K. Mohror, A. T. Moody, B. C. V. Essen, M. B. Gokhale, R. B. Ross, and B. R. de Supinski, “Integrated in-system storage architecture for high performance computing,” in ROSS ’12, 2012.

[26] “Supermuc-ng.” [Online]. Available: https://doku.lrz.de/display/Public/SuperMUC-NG