Generation of Checkpoints for Hardware Architecture Simulators

Sebastian Weber
sebastian.weber@fzi.de
FZI Research Center for Information Technology

Lars Weber
lars.weber@student.kit.edu
Karlsruhe Institute of Technology

Thomas Weber
thomas.weber@kit.edu
Karlsruhe Institute of Technology

Jörg Henß
henss@fzi.de
FZI Research Center for Information Technology

Robert Heinrich
robert.heinrich@uni-ulm.de
Ulm University

Abstract

Simulations help to understand the behavior of systems before they are built or deployed. Combining different simulation tools can be useful to focus on specific aspects, such as overall functionality or detailed timing. Switching between tools during a simulation requires knowledge of the current simulated system state. If this state is missing, results may be unreliable. This is especially important in multi-level simulations, which often consist of many switches and short simulation phases. We present an approach that captures the internal state of a system, which we call checkpoint, from an emulation in QEMU (Quick Emulator), including memory, processor contents, and virtual disk data. The checkpoint can be reused across simulation tools, enabling realistic starting conditions, more accurate results, and support for modular simulation workflows.

1 Introduction

Simulation is a widely used technique to evaluate the behavior and performance of hardware/software systems in early development stages or during system evolution. The simulation of a system requires a model of its structure and behavior and a representation of how it is used. An additional influencing factor becomes important depending on the system and the simulated duration: the *initial state* of the system. Grassmann [1] describes this phenomenon as *initialization bias*, where simulation results may be significantly distorted if the system begins from an unrealistic or unrepresentative state. This issue can be addressed through warm-up periods, allowing the system to reach a statistical equilibrium before data is collected. However, such techniques require long simulation durations and become impractical in scenarios where only short simulation phases are executed. One such scenario is multi-level simulation, such as the approach proposed by Weber et al. [8], where the level of abstraction can change dynamically and frequently during a simulation run. These changes can result in many short simulation segments that lack the runtime needed for traditional bias-reduction methods to be effective. In such workflows, each transition between simulators demands a new, consistent, and realistic initial state to ensure continuity and validity of results.

To address this challenge, we present a method for extracting a detailed system state from a running QEMU[1] instance, which was developed during a bachelor's thesis [7]. QEMU is an open-source machine simulator and virtualizer that supports a wide range of processor architectures, including x86, ARM, and RISC-V. It can simulate complete hardware architectures, including CPUs, memory, and peripheral devices, allowing unmodified operating systems and software to run in a controlled environment. One goal of QEMU is the analysis of detailed hardware architectures, which software simulations typically do not support. Depending on the configuration, QEMU can simulate purely in software or use hardware-assisted virtualization when supported by the host. Its modular structure and external control interfaces, such as the QEMU Machine Protocol (QMP), make it suitable for automation, analysis, and integration into simulation workflows. Our approach produces structured system snapshots, which we call *checkpoints*, that capture the internal state of a simulated machine, including CPU registers, main memory, and the layout and contents of block devices. These checkpoints are serialized in an hardware-architecture-agnostic format and support deduplication of identical memory and storage regions across checkpoints. The extraction is performed via QEMU's Machine Protocol (QMP) and Human Monitor Protocol (QHM), requiring no modifications to QEMU itself. The ability to generate checkpoints directly from functional simulations enables more accurate timing analysis based on these checkpoints, reduces the need for heuristic or synthetic initialization, and allows reproducibility in

[1] `https://www.qemu.org/`

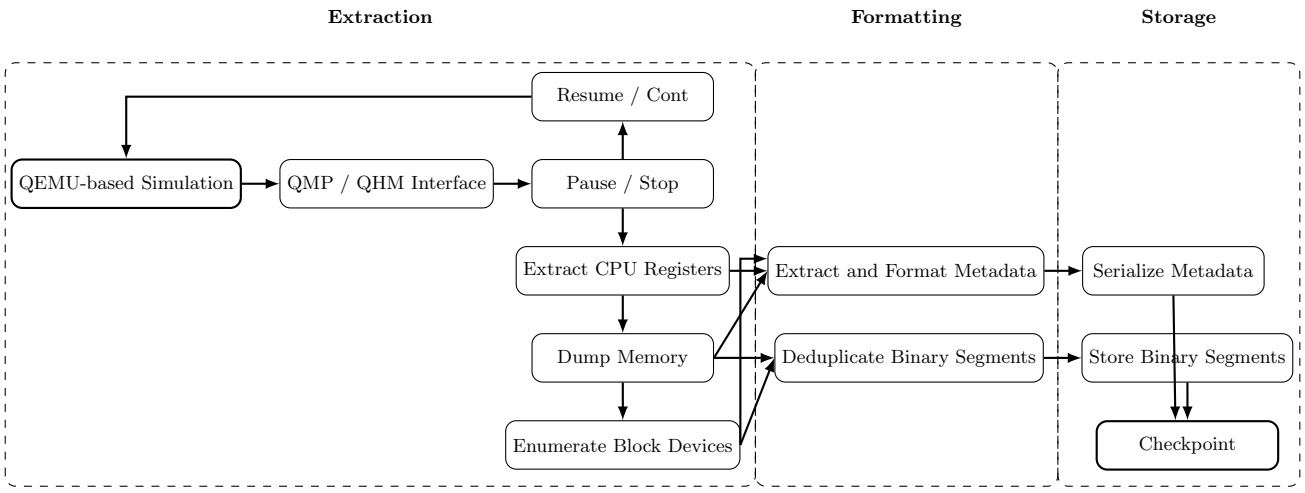

Figure 1: Checkpoint generation from a QEMU-based simulation: extraction via QMP/QHM, metadata and binary formatting, deduplication, and packaging into a portable checkpoint.

modular simulation workflows. In line with the vision of Weber et al. [8], our checkpoints act as a bridge between fast high-level and slow low-level architectural timing simulation. QEMU emulates the behavior of the system correctly, but not necessarily the timing. The generated checkpoints can then be used as input for both high- and low-level simulations, eliminating the need to transform between the states of these simulations. By providing realistic and consistent simulation entry points, they mitigate the risk of initialization bias and help to ensure that despite short simulation segments the overall results remain accurate and reliable.

2 Checkpoint Generation

Figure 1 illustrates our approach for generating checkpoints from QEMU-based simulations. The process can be divided into three stages: extraction of the system state, formatting of the extracted data, and storage of reusable checkpoints, which will be presented in the following paragraphs.

Extraction The extraction stage interfaces with a running QEMU instance through QMP and QHM. This allows the tool to operate without modifying QEMU itself, ensuring compatibility with QEMU across different versions. To guarantee consistency, the QEMU instance is paused before extraction begins, and resumed afterwards. During this pause, the tool collects CPU register states using QHM commands wrapped via QMP, dumps system memory in ELF format, and enumerates block devices using the query-block command. CPU registers are queried in a way that ensures they are captured in their entirety, while memory is dumped in a binary format that maintains its structure. Each device is listed with its associated image file, allowing for the recreation of the complete system state.

Formatting and Storage Once the system state has been extracted, it is formatted into a portable and modular structure to facilitate its reuse across different simulators. The extracted data is split into two categories: metadata and binary data. Metadata, including CPU registers and device configurations, is serialized as JSON, which provides a flexible and human-readable structure. The memory and block device contents are stored as binary segments. These binary segments are identified using cryptographic hashes, specifically SHA-256, allowing for efficient deduplication. Identical memory regions or device images across multiple checkpoints are stored only once, significantly reducing storage overhead. The binary segments of each checkpoint are compared using their hashes to avoid redundancy. This process is especially useful when frequent checkpoints are generated, as it minimizes the required storage space.

Evaluation The approach was evaluated based on three criteria: correctness and performance overhead. For correctness, we compared the extracted checkpoints with native QEMU snapshots, ensuring that CPU registers, memory dumps, and block device states were identical, We did this for both x86 and ARM. QEMU snapshots resemble internal memory dumps that are primarily intended for restoring virtual machines within the same QEMU instance. They are difficult to interpret externally without extensive parsing and can be subject to structural changes between QEMU versions. The results confirmed that our tool accurately captures the internal system state, preserving the integrity of the simulation. In terms of performance, we measured the time required to pause QEMU, extract data, and resume the simulation. While the extraction of CPU registers takes less than one second, extracting and saving the main memory and block devices depends on their size. Creating a checkpoint for the Windows 11 Setup with a size

of 6.6 GiB and 1.8 GiB RAM took about 15 seconds, which corresponds to the access speed of the NVMe SSD used, because the data had to be accessed three times. Working with smaller operating systems and embedded hardware therefore allows to take checkpoints in less than one second.

3 Related Work

Godala et al. [6] present QPoints[2], a tool for creating checkpoints in QEMU to enable deterministic replay and reproducibility of experiments. While QPoints demonstrates the feasibility of checkpointing in QEMU, its implementation requires invasive modifications to the emulator, which complicates maintenance across different versions. Baudis [3] proposed an approach to extract checkpoints from QEMU within a SimuBoost [4] setup. He modified the QEMU code and relied on the GNU debugger for data extraction, using primarily the QEMU Human Monitor. Due to the code being outdated and not publicly available the thesis could not be used as starting point for our work. In contrast to the these approaches, our approach avoids modifications to the QEMU code base by relying exclusively on its external interfaces (QMP and QHM). This enables us to capture the internal state of a simulated system, including CPU registers, memory, and block devices, in an architecture-agnostic and reusable format, thereby reducing the effort of integrating QEMU-based checkpoints into other simulation workflows.

Weisse et al. [5] developed Lapidary[3], a Python-based tool for extracting checkpoints from the timing simulation gem5[4] in the context of their work on speculative execution attacks. The simulation of systems in gem5 is done in a similar way than QEMU, while focusing on accurate performance as well. Therefore it served as a possible target for state injection in other simulators, which could not be implemented due to time constraints.

A related line of research focuses on deterministic replay of system executions in QEMU. Such approaches, exemplified by Dovgalyuk et al. [2], modify the QEMU internals to record all non-deterministic events during execution, enabling bit-exact reproduction of program behavior for debugging and dynamic analysis. Rather than reproducing execution histories, we aim to extract a reusable, architecture-independent representation of the system state that can serve as a realistic initial condition for further simulation or analysis.

4 Conclusion

We explored an approach for generating checkpoints from QEMU-based simulations in this paper. By leveraging QEMU's external interfaces (QMP and QHM), we were able to extract the internal state of a system, including CPU registers, memory, and block device configurations. The extracted data is stored in a modular, architecture-agnostic format that allows for reuse across different simulation tools and systems. Our approach was evaluated in terms of correctness, performance overhead, and storage efficiency. The results indicated that the tool generates accurate checkpoints with low processing overhead, and deduplication of binary segments reduces storage requirements by up to 80%.

We will carry out future work on extending support for larger memory sizes and adding compatibility for external devices such as GPUs or TPMs. Additionally, the reintegration of checkpoints into running QEMU instances, including the manipulation of system states for distributed simulations or state persistence across simulation phases is planned. Finally, integrating our checkpointing tool into broader multi-level simulation workflows will demonstrate its utility in real-world software architecture analysis and design processes.

[2] https://github.com/PrincetonUniversity/QPoints
[3] https://github.com/efeslab/lapidary
[4] `https://www.gem5.org/`

Acknowledgements

This work has received funding from the European Chips Joint Undertaking under Framework Partnership Agreement No 101139789 (HAL4SDV) including the national funding from the German Federal Ministry of Research, Technology and Space (BMFTR) under grant number 16MEE0468. The responsibility for the content of this publication lies with the authors. This work was funded by the DFG (German Research Foundation) – project number 499241390 (FeCoMASS), by the Topic Engineering Secure Systems of the Helmholtz Association (HGF) and supported by KASTEL Security Research Labs, Karlsruhe and supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - SFB 1608 - 501798263.

References

[1] W. Grassmann. "Rethinking the initialization bias problem in steady-state discrete event simulation". In: *Proceedings of the 2011 Winter Simulation Conference (WSC)*. 2011, pp. 593–599.

[2] P. Dovgalyuk. "Deterministic Replay of System's Execution with Multi-target QEMU Simulator for Dynamic Analysis and Reverse Debugging." In: *CSMR*. 2012, pp. 553–556.

[3] N. Baudis. *Deduplicating Virtual Machine Checkpoints for Distributed System Simulation*. Bachelor's Thesis. Karlsruhe Institute of Technology (KIT). 2013.

[4] M. Rittinghaus. "Simuboost: Scalable parallelization of functional system simulation". PhD thesis. Karlsruhe Institute of Technology (KIT), 2019.

[5] O. Weisse et al. "NDA: Preventing Speculative Execution Attacks at Their Source". In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO-52. Columbus, OH, USA: ACM, 2019, pp. 572–586.

[6] B. R. Godala et al. "QPoints: QEMU to gem5 ARM Full System Checkpointing". In: *gem5 Workshop at ISCA 2023*. 2023.

[7] L. Weber. *Generation of Checkpoints for Hardware Architecture Simulators*. Bachelor's Thesis. Unpublished, Karlsruher Institut für Technologie (KIT). 2024.

[8] S. Weber et al. "Combining a Functional Simulation with Multi-Level Timing Simulation for Software Architecture Models to Improve Extensibility". In: *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*. IEEE. 2024, pp. 74–78.