SUMMARY

Computers and their software play an ever increasing role in our daily life—software runs on our computers, phones, TVs and around our arms in the form of smart watches. While one can argue that software has improved our quality of life in many ways, it is plagued by a problem, which is as old as software itself: Reliable software is hard to build. "Have you tried turning it off and on again", has become our pop-culture's iconic manifestation of this problem. So it does not come as a surprise that the research of methodologies and technologies to make software *reliable* originated at the same time in which digital computers first hit the industry and universities.

Checkpointing is one important technique that originating from this research and has many applications inside the reliability domain, such as automated error recovery and debugging. An integral part of checkpointing is taking a snapshot of a process' memory, also known as *memory checkpointing*, which is the main subject of this thesis.

In particular this thesis concentrates on scenarios that require *high checkpointing frequencies*. Examples of such use cases are automatic error recovery techniques that require checkpoints on every client request or on carefully selected rescue points. Further, in debugging scenarios the ability take checkpoints with very high frequency allows the inspection of arbitrary memory states throughout the execution.

For these high-frequency checkpointing scenarios memory checkpointing is an important cost factor, which makes it an interesting and worthwhile target for optimization. Applications that require high checkpointing frequencies and a long checkpoint history also face the challenge of storing checkpoints efficiently lest they lose precious application memory.

In Chapter 2, we investigate pure user-level high frequency checkpointing. Limiting our investigation to a pure userland implementation addresses the issue that, when deploying checkpointing solutions in the real world, changes to the kernel are often not possible. We examine three page granular check-

pointing mechanisms, as well as one instrumentation-based checkpointing technique on a set of server applications. To achieve a high checkpoint frequency, we take a checkpoint for each request, a common use-case, for example, in request oriented recovery. Our experiments show that with higher checkpointing frequency, the page granular user-level techniques suffer from a high performance overhead. The instrumentation-based undolog, while offering better performance for high frequency applications, suffers from unbounded memory usage. We then present *Lightweight Memory Checkpointing* (LMC), a system for efficient memory-bound high frequency user-level checkpointing. Instead of relying on inefficient kernel primitives that offer page granular copy-on-write semantics, LMC utilizes compiler instrumentation to implement copy-on-write on byte-granularity. In contrast to the undolog approach, LMC stores checkpoint data in a shadow-state organization, thereby limiting memory usage.

In Chapter 3, we revisit page granular checkpointing, however, this time relaxing our requirement of not allowing kernel modifications. We start our investigation with an analysis of the costs of page granular checkpointing, showing that saving a page using Linux CoW mechanism—by forking—is circa eight times as expensive as just copying the page, ignoring the costs for the fork system call itself, which are substantial. The results of this investigation motivate the design of the second memory checkpointing technique that we present in this thesis, *Speculative Memory Checkointing* (SMC). While SMC allows applications to efficiently access the kernel's CoW mechanism, it also features a speculation component, which by speculatively copying hot pages aims to avoid the eight times higher cost of kernel CoW over plain copying. As the speculation essentially consists of Writable Working Set Estimation (WWSE), we evaluated two established WWSE techniques—Active-RND and Active-CKS—and our novel GSpec WWSE algorithm. In contrast to the traditional WWSE techniques, GSpec's approach can automatically tune to different workloads.

Our experiments show that a hypothetically perfect and overhead-free speculation can reduce the overhead of CoW checkpointing by 29.3 percentage points from 44.9% to 15.6% in the tested scenario. This confirms the usefulness of speculation to reduce COW induced checkpointing overhead. Further, we show that all tested speculation strategies improve the performance overhead while leading to a modest increase of memory consumption and recovery time. GSpec exhibits the highest speculation accuracy across the tested applications and reduced checkpointing overhead by 14.1 percentage points from 44.9% to 30.8%, compared to plain COW.

In Chapter 4, we presented *DeLorean*, a debugging system leveraging checkpoint-assisted time-traveling introspection, as a real-world use case of high frequency checkpointing. DeLorean emphasizes the importance of efficient checkpoint storage and exploration, two important problems that arise when many checkpoints are taken with high frequency. DeLorean uses the checkpointing component presented in Chapter 3 to achieve efficient high frequency checkpointing to reduce the run-time overhead during the recording phase. By compressing and deduplicating checkpointed pages, DeLorean is capable of reducing the memory footprint of the checkpointing storage by 95%, making it possible to store millions of checkpoints. Further, we explore different ways to inspect the checkpoint history. We show that when knowing memory locations of interest, partial on-demand rollback can save up to 88% of search time in our experiments. This can be further improved if certain properties of the search criteria are known, which in some cases allows us to use use a bisect search strategy, effectively enabling us to search one million checkpoints in around 8.5 seconds.

In summary, this thesis shows the need for specialized high frequency memory checkpointing techniques and proposes enhancements to current checkpointing techniques to make them fit for use cases that require checkpoints to be taken with a high frequency. We explored the deployability trade-off of different checkpointing techniques and showed that if the target application can be recompiled, pure userland techniques relying on compiler instrumentation can offer a significantly better run-time performance than their page-granular counterparts. LMC's shadow state inspired checkpoint organization shows that it is not necessary to give up memory guarantees to achieve this. Further, we showed that speculation and exporting copy-on-write functionality as a first level kernel primitive to the userland lowers the overhead of page granular checkpointing significantly. Finally, we explore techniques that allow for efficiently storing and searching a the large number of checkpointed data resulting from the high checkpoint frequency.

We hope that these techniques combined open up a whole new range of possible scenarios for high frequency memory checkpointing, such as the time traveling debugging system DeLorean presented in this thesis.