The Journal of Systems and Software 139 (2018) 1–13

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Efficient modeling and optimizing of checkpointing in concurrent component-based software systems

Noor Bajunaid, Daniel A. Menascé*

Department of Computer Science, George Mason University, Fairfax, VA 22030, USA

ARTICLE INFO

Article history:
Received 27 September 2017
Revised 17 December 2017
Accepted 21 January 2018

Keywords:
Checkpointing
Performance modeling
Markov Chains
Mean value analysis
Concurrent and heterogeneous component-based software systems

ABSTRACT

A common mechanism to improve availability and performance is checkpointing and rollback. When it is time to checkpoint, a system stores a job's state to nonvolatile memory, and, when a failure occurs, it rolls back to the latest stored state instead of restarting the job from the beginning, thus improving performance in the presence of failures. Too frequent checkpointing reduces the amount of work to be redone in case of failures but generates excessive overhead, degrading performance. This paper presents a novel and very efficient queuing network model that addresses software component contention for hardware resources and shows how it can be used to model checkpointing in heterogeneous component-based software systems. We validated this model against a previous model, developed by the authors, that used Markov Chains. Our new model is orders of magnitude faster than the previous one and can be used to plan for checkpointing at run-time. As an additional contribution of this paper, we present an optimizer to find, for each software component, the optimal checkpointing interval that minimizes execution time, maximizes availability, or minimizes checkpointing overhead.

1. Introduction

Long-running and critical computations are expected to be highly reliable. A well known mechanism to improve system availability and performance is *checkpointing and rollback*. When it is time to checkpoint, a system stores a job's state to nonvolatile memory, and, when a failure occurs, it rolls back to the latest stored state instead of restarting the job from the beginning. However, the time interval between checkpoints needs to be well planned: if this interval is too large, a big part of the work can be lost when failures occur, whereas too frequent checkpoints can slow down a system due to the checkpointing's performance overhead. In adaptive systems, planning for checkpointing can be a part of a control loop that continually monitors and analyzes a system state, and plans accordingly. Runtime models can be very helpful for planning in adaptive systems (Weyns et al., 2012) and as part of self-healing systems (Kaitovic and Malek, 2016). However, these models need to be efficient to avoid exhausting a system's performance and allow for near real-time decision making.

A vast amount of work has been done over the last several decades on the problem of finding the optimal checkpointing interval. An early example is Young's formula (Young, 1974) that relates the optimal checkpointing rate with the time needed to checkpoint and with the failure rate. However, none of the prior work takes into account contention for shared resources in component-based software systems. This contention arises from the fact that the components and their checkpointing processes compete for shared processing and I/O resources. It is shown in a later section of this paper, that Young's formula does not provide optimal results when components compete for hardware resources. Checkpointing at a component level is important, because it is cheaper than checkpointing an entire system and can improve the system's availability.

In a recent previous work (Bajunaid and Menasce, 2017), we proposed an analytical model, that handles contention among components, for checkpointing in component-based software systems. The model allowed us to compute system availability, execution time and the overhead of checkpointing on system performance in the presence of contention among components. That model can be used to statically plan the checkpointing rate given component attributes such as average job length, failure rate, and checkpointing resource demands. However, in modern systems that adapt at run-time, components and their attributes change over time, which mandates frequent planning for configurations, including the checkpointing interval. The models used for run-time planning need to be efficient enough so that they can be used to make near real-time decisions. Moreover, these models need to be scal-

* Corresponding author.
E-mail addresses: nbajunai@masonlive.gmu.edu (N. Bajunaid), menasce@gmu.edu (D.A. Menascé).

https://doi.org/10.1016/j.jss.2018.01.032
0164-1212/

able to handle modern systems that consist of hundreds of thousands of components.

In this paper, we propose a new and more *efficient* model, called Component Phase Transition (CPT), that addresses component contention for hardware resources. We show how this model can be used to model checkpointing in component-based software systems. This new model solves the same problem we addressed in Bajunaid and Menasce (2017) and calculates the same metrics using a significantly more efficient method. We compare the two models to highlight how the new model significantly outperforms previous models while producing the same accurate results (as shown by validation through simulation and experimentation). The second contribution of this paper is an *optimizer* that uses local search to find the optimal checkpointing rate for each component using the proposed model. This optimizer is efficient enough to be used as part of a run-time adaptation loop.

The rest of this paper is organized as follows. Section 2 discusses some of the relevant prior work in modeling checkpointing. Section 3 provides background on checkpointing in a concurrent component-based system and presents the core results we expect from our models. Section 4 discusses our previous modeling approach, which uses Markov Chains and consists of two models: (1) the homogeneous model for systems of components that have the same resource demands and failure rates, and (2) the heterogeneous model for systems of components that have different demands and/or different failure rates. Section 5 introduces the Component Phase Transition (CPT) approach and Section 6 shows how it can be used to model checkpointing in concurrent heterogeneous component-based systems. Section 7 presents the optimizer that uses a hill-climbing local search method and the CPT model to find the optimal checkpointing rate. The following section shows some results using the optimizer. Finally, the paper discusses some concluding remarks and ideas for future work.

2. Related work

There is a vast body of literature since the work of Young (1974) on analytic models for obtaining the checkpointing interval that optimizes a variety of metrics. Some examples include: minimize total execution time, maximize availability, maximize a job's progress, and minimize the overhead generated by checkpointing and wasted work due to rollback. A comprehensive and relatively recent book by Wolter (2010) contains a thorough description of many existing stochastic models for checkpointing, restart, and rejuvenation, including many discussed in this section, and other novel models introduced by Wolter. However, the aforementioned models do not consider contention among components while executing or checkpointing nor do they consider that components may be heterogeneous. We highlight here a few of the previous related works (see Wolter, 2010 for an extensive bibliography).

Gelenbe and colleagues developed comprehensive models for rollback and checkpointing under various assumptions regarding failure time distribution and static versus dynamic checkpointing (Gelenbe, 1976; Gelenbe and Derochette, 1978; Gelenbe, 1979). Other analytic models can be found in Chandy et al. (1975) and Tantawi and Ruschitzka (1984).

Previous checkpointing models used Markov Chains. For example, Geist et al. (1988) studied the selection of checkpointing that maximizes the probability of task completion in systems with limited repairs in which failures are allowed to occur during the checkpoint operation. The paper shows that the optimal checkpointing interval depends on the distribution of both the time between failures and the number of repairs the system can handle. Wong and Franklin (1993) proposed a model for synchronous checkpointing in scientific computation of multiple nodes, under the assumption of Markovian state occupancy and Poisson failures. They included models with and without load redistribution. Plank and Thomason (1999) modeled checkpointing in a parallel application that runs on a subset of processors. The goal of the model is to select the checkpointing interval and the number of parallel processors to maximize availability.

Nicola and Van Spanje (1990) study and compare different checkpointing strategies and models in order to select one that adequately represents a realistic system and is yet tractable for analysis. Dimitrov et al. (1991) developed analytic models to find a checkpointing schedule that optimizes a job's total processing time under implicit breakdowns, i.e., failures are not detected immediately but a special test has to be performed to detect the failure.

Kishor Trivedi has done substantial work in using performance modeling to assess software reliability and the impacts of software rejuvenation (Garg et al., 1996). The work by Ling et al. (2001) uses variational calculus to derive a closed form expression for the optimal checkpointing frequency as a function of the failure rate with the goal of minimizing the total expected cost of checkpointing and recovery.

Daly (2006) provides a high order estimate of the optimum checkpoint interval to minimize total application runtime under Poisson failures. Chen and Ren (2009) analyze the relationships between checkpointing interval and system availability, task execution time, and task deadline miss probability, for soft real-time applications. Bougeret et al. (2011) develop solutions for optimal checkpointing that minimize execution time for sequential and parallel jobs with Poisson failures and use a dynamic programming heuristic for the case of Weibull failures.

Lu et al. (2013) derive optimal checkpointing intervals for systems with latent errors, i.e., errors that may go undetected for some time. This assumption is more realistic than that of immediate failure detection assumed by the vast majority of the checkpointing modeling work, including ours. The authors discuss the importance of multiversion checkpoints to achieve acceptable failure coverage. Di et al. (2014) present a sophisticated deterministic multilevel checkpoint optimization model in the context of exascale systems with a large number of multicore nodes. The authors consider a parallel application with many processes running on many cores. Jones et al. (2012) use simulation with real workload data to demonstrate the impact of sub-optimal checkpoint intervals on application efficiency in HPC clusters. No analytic model is presented. A comprehensive survey of roll-back recovery protocols in message-passing systems was presented in Elnozahy et al. (2002). Leach (2008) conducted a study on the insertion of checkpoints within a legacy software system in the aerospace domain. Recent studies have leveraged the use of NVRAM as a replacement to disk to store checkpoints. Gao et al. (2015) discuss the design and implementation of a checkpointing system called *Mona* that combines NVRAM with DRAM; partial checkpoints are written from DRAM to NVRAM.

Besides its use for fault-tolerance, checkpointing has been used to store a process's state in speculative synchronization (Martínez and Torrellas, 2002), to ease the rollback to the synchronization point in case of conflicts. Waliullah and Stenstrom (2008) proposed a scheme that predicts the occurrence of conflicting accesses due to speculation in transactional memory systems. Their schema inserts checkpoints before predicted conflicts to minimize the time to roll-back and improve the system's speedup.

3. Checkpointing in concurrent component-based systems

A set of n software components (referred to as components hereafter), $C_1, \ldots, C_k, \ldots, C_n$, execute typically long jobs (e.g., scientific computations) using shared resources (e.g., processors, net-