ScienceDirect

Journal of Systems and Software
Volume 122, December 2016, Pages 364-377

An efficient validation approach for quasi-synchronous checkpointing oriented to distributed diagnosability

Houda Khlif [a], Hatem Hadj Kacem [a], Saúl E. Pomares Hernandez [b c d], Ahmed Hadj Kacem [a], Cédric Eichler [c d], Alberto Calixto Simón [e]

https://doi.org/10.1016/j.jss.2016.04.070
Get rights and content

Highlights

- Quasi-synchronous checkpointing algorithms are classified into: SZPF, ZPF and ZCF.
- We propose a validation approach to detect the previously mentioned properties.
- We model an algorithm execution into the HBR graph and IDR graph.
- We designed two sets of validation rules to work over the HBR graph and IDR graph.
- A lower cost is achieved using the IDR graph which is the minimal causal graph.

Abstract

The autonomic computing paradigm is oriented towards enabling complex distributed systems to manage themselves, even in faulty situations. The diagnosability analysis is *a priori* a study through which a system can be self-aware about its current state. It is from the determination of a consistent state that a system can take some action to repair or reconfigure itself. Nevertheless, in a distributed system it is hard to determine consistent states since we cannot observe simultaneously all the local variables of different processes. In this context, the challenge is to efficiently monitor the system execution over time to capture trace information in order to determine if the system accomplishes both functional and non-functional requirements. Quasi-synchronous checkpointing is a technique that collects information from which a system can establish consistent snapshots. Based on this technique, several checkpointing algorithms have been developed. According to the checkpoint properties detected and ensured, they are classified into: Strictly Z-Path Free (SZPF), Z-Path Free (ZPF) and Z-Cycle Free (ZCF). Generally, the method adopted for the performance evaluation of checkpointing algorithms involves simulation. However, few works have been designed to validate their correctness. In this paper, we propose an efficient validation approach based on a graph transformation oriented towards the automatic detection of the previously mentioned properties. To achieve this, we took the vector clocks resulting from an algorithm execution, and we modeled them into the happened-before graph and the immediate dependency graph (which is the minimal causal graph). Then, we designed a set of transformation rules to verify if in these graphs, the algorithm is exempt from non-desirable patterns, such as Z-paths or Z-cycles, according to the case.

Introduction

As computing systems have reached a level of complexity, their management has become increasingly difficult. As a result, the initiative of autonomic computing has been introduced to prevent human intervention and enable the system to manage itself, even in faulty situations. An Autonomic Distributed System is considered to be a set of geographically-distributed autonomic components that communicate and collaborate with each other. In general, autonomic computing has four elements: self-configuration, self-healing, self-optimization and self-protection. The diagnosability analysis is *a priori* a study through which a system can be self-aware about its current state (Cordieretal., 2008). It is from the determination of a consistent global state that a system can take some action to repair or reconfigure itself. Nevertheless, in a distributed system, it is hard to determine consistent states since we cannot observe simultaneously all the local variables of different processes (Pencole,2004). In this context, the challenge involves efficiently monitoring the system execution over time in order to capture trace information. With this information, consistent global states will be determined to evaluate if the system accomplishes both functional and non-functional requirements. Checkpointing is a well-known technique used to identify consistent global snapshots from local recorded states called checkpoints. Informally, a global snapshot is consistent if the set of checkpoints that compose it (one per process) accomplishes the following two constraints: first, all the local checkpoints in the snapshot are concurrent and no Z-path exists from one local checkpoint to another or itself (Netzerand Xu,1993). This last case is called a $\boldsymbol{Z-Cycle}$ (these patterns are formally defined in Section2.2).

Intuitively, to illustrate the concurrency constraint we present following example scenario (a Z-path example scenario is presented in Section2.2). In order to distributively diagnose the load balancing in two servers, the servers share a common variable *v*. At the beginning this variable is equal to zero, which means that there is no local load, and each time its load increases or decreases, such variable is locally incremented or decremented accordingly. If a server changes its variable by more than 10 units, it will send a message to the other server to inform it of the new value. At the reception of the message, the server updates the content of its local variable with the value of *v* piggybacked onto the message. The safety non-functional constraint is defined by $|v_1 - v_2| \leq \delta$. If the safety constraint is satisfied, this means that the system is balanced. The problem is distributively determining at what moments in time such constraint can be consistently evaluated. In the scenario depicted in Fig. 1, there are two snapshots: the first one is composed of the checkpoints C_1^2 and C_2^2, and the second one is composed of C_1^3 and C_2^3.

The first snapshot is inconsistent since it does not contain all the history of the causal events at the moment of the cut (the *delivery* event of m_2 is in the cut, but not its *send* event). If we evaluate the safety constraint in this snapshot at $\delta = 50$, the result will give a false alert of unbalance since $v_1 = 80$ in C_1^2 and $v_2 = 10$ in C_2^2. The second snapshot is consistent since the entire causal history of the events is included at the moment of the cut. For this reason, only in consistent snapshots does the safety constraint need to be evaluated in order to be certain of the result.

Checkpointing algorithms are organized into three classes: asynchronous, synchronous and quasi-synchronous (Manivannan and Singhal, 1999). In asynchronous checkpointing, also known as uncoordinated checkpointing, each process takes its checkpoints independently, which leads to the domino effect. In synchronous checkpointing, or coordinated checkpointing, processes coordinate their checkpoints by the addition of control messages so that a globally consistent set of checkpoints is always maintained in the system. Major disadvantages of coordinated checkpointing are: the process execution may have to be suspended during the checkpointing coordination, resulting in performance degradation, and it requires extra message overhead to synchronize the checkpointing activity. In quasi-synchronous checkpointing, or Communication Induced Checkpointing (CIC), coordination is achieved by