The Journal of Systems and Software 87 (2014) 74–86

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Surviving sensor node failures by MMU-less incremental checkpointing

Hsung-Pin Chang [a,*], Yen-Ting Liu [b], Shang-Sheng Yang [b]

[a] Department of Computer Science and Engineering, National Chung Hsing University, Taichung, Taiwan, ROC
[b] Institute of Networking and Multimedia, National Chung Hsing University, Taichung, Taiwan, ROC

ARTICLE INFO

Article history:
Received 16 October 2012
Received in revised form 3 September 2013
Accepted 3 September 2013
Available online 13 September 2013

Keywords:
Checkpointing
Incremental checkpointing
SOS

ABSTRACT

For some critical safety applications, sensor nodes embed valuable information, and they should be able to operate unattended and unfailing for several months or years. One promising solution is to adopt a checkpointing that periodically saves the state of a sensor node, thereby maintaining node reliability and network availability. Thus, this study first shows the design and implementation of a full checkpointing for WSNs. However, checkpointing is expensive. Therefore, incremental checkpointing was previously proposed to eliminate the checkpoint overhead by relying on the page protection hardware to identify dirty pages. Because sensor nodes are resource-constrained and do not equip with the page protection hardware, previous incremental checkpointings cannot be directly applied. To address this issue, this paper proposes three incremental checkpointings for WSNs. These three methods differ in the granularity of the checkpoint memory data unit and module execution overhead. In addition, we designed an incremental checkpoint file format that simultaneously supports proposed three different incremental checkpointings and accommodates them with sensor network characteristics. We implemented the full and three incremental checkpointings on SOS in the mica2 sensor motes. A performance evaluation of the three incremental checkpointings is presented. We also discuss and evaluate a method for selecting the appropriate incremental checkpointing. To the best of our knowledge, this study is the first to design and implement incremental checkpointing in MMU-less WSNs.

1. Introduction

In general, sensor nodes are expected to be autonomous and long lasting. However, hardware reliability poses a major challenge to this expectation (Borkar, 2005). Because of large-scale deployment, hardware fails in sensor nodes are highly possible. The problem is highlighted because sensor nodes are likely to interact closely with their physical environment that may be harsh or hostile (Pompili et al., 2006; Werner-Allen et al., 2006; Talzi et al., 2007), rendering the hardware components more failure-prone. Notably, hardware failures can be permanent or transient. In this paper, we address the fail-stop permanent hardware failures. In general, the ability to respond to a fail-stop hardware failure requires physical access to a node. Unfortunately, WSNs are often deployed in locations away from easy human access. Once sensor nodes are deployed, it is impractical and at times even impossible to physically access each individual node.

However, for some emergency or surveillance applications, e.g., medical care (Lorincz et al., 2004) and fire rescue (Tseng et al., 2006), node failure is unacceptable. Unexpected hardware failures could cause problems ranging from financial impact to loss of life. Even for normal applications, some sensor nodes, such as the grid heads that are responsible for maintaining routing tables and forwarding sensor data to the sink (Chi and Chang, 2012), have important roles in WSNs. As a result, the failure of these nodes could render the sensor network useless, and at worst, lead to the collapse of the entire network. Therefore, a failure-resilient mechanism that can recover from hardware failures is imperative for some sensor nodes and sensor network applications.

To realize a fault-tolerant operation, one possible solution is to use a space-redundant technique so that the system can continue to operate when a hardware component fails. Therefore, on the basis of mica2 motes (Mica motes), we proposed a dual-motes sensor node architecture that connects two motes, a primary and a backup node, using a RS-232 link (Yeh, 2009). In general, the primary node is active and the backup node lies dormant, drawing just a small current. Once the primary node fails, the backup node is activated to perform the task of primary node, thereby maintaining node reliability and network availability. However, the primary node may contain important information. Along with the failure of

* Corresponding author. Tel.: +886 4 22852106; fax: +886 4 22853869.
E-mail addresses: hpchang@cs.nchu.edu.tw (H.-P. Chang), hylswind@gmail.com (Y.-T. Liu), b3c4d5e6f7@hotmail.com (S.-S. Yang).

0164-1212/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2013.09.001

the primary node, this information, which is crucial in some applications, would also be lost. To further illustrate this problem, we study a representative scenario as follows.

1.1. Scenario: theft detection application

S. Guha et al. proposed the AutoWitness system to deter, detect, and track personal property theft (Guha et al., 2012). A property owner embeds a small Tag Node inside the asset to be protected, e.g., a television. If the Tag Node detects vehicular movement, it determines that the asset is stolen and begins to estimate the sequence of movements, stops, and turns. These estimates are stored locally until a cellular network connection is available, at which time the Tag Node is connected to a server to calculate the most probable route and resting destination.

In the above scenario, crucial information is embedded in the Tag Node. If the Tag Node fails, the crucial information is lost and the stolen asset cannot be found easily. To solve the problem, we can implement a checkpoint mechanism on the basis of the dual-motes sensor nodes. The primary node periodically makes a checkpoint of the current state to the backup node. Once the backup node detects that the primary node has failed, for example, using beacon messages or sophisticated fault detection schemes (Guo et al., 2009; Yeh, 2009), it substitutes the primary node and rolls back to the most recent checkpoint. However, checkpointing is an expensive operation. Therefore, many improvements have been proposed to reduce the checkpoint overhead (Plank, 1997). One of the well-known optimization schemes is incremental checkpointing, which utilizes the page protection hardware to identify the unchanged portion of a checkpoint and saves only the changed portion to reduce the amount of data written to the stable storage. Unfortunately, sensor nodes usually do not have the memory management unit (MMU). MMUs must contain page tables and the associated logic, which have significant impact on area, run-time, and power. Of these, power consumption is an area of concern because memory protection hardware must be activated for every memory reference instruction (Biswas et al., 2006). As a result, legacy incremental checkpointing cannot be applied in WSNs.

In contrast to the previous hardware-assisted solution, we propose software-based incremental checkpointing. However, the design and implementation of software-based incremental checkpointing faces various challenges. First, sensor nodes are usually highly resource constrained in terms of CPU, persistent storage, and most importantly, power. Second, WSNs are applied in numerous applications, and for a WSN application, different sensor nodes would play different roles. Thus, the proposed software-based incremental checkpointing must be able to accommodate WSNs with different applications and nodes playing different roles. Finally, sensor nodes usually employ solid-state flash memory for persistent storage. However, it is well-known that flash memory has the write endurance issue; repeated writes to the flash memory quickly exhaust the flash memory's lifetime.

Thus, the objective of this study is to design and implement an efficient, flexible, and flash-aware incremental checkpointing for micro sensor platforms. In particular, the main contributions of this study can be summarized as follows. First, we propose three software-based incremental checkpointings for WSNs: *scheduler-based*, *icall-based*, and *store-based checkpointing*. These three incremental checkpointings differ in the module execution overhead and checkpoint size, which in turn results in different CPU and I/O costs. Thus, on the basis of application characteristics, different application modules can choose different incremental checkpointings to minimize the total checkpoint overhead. Furthermore, we propose two optimization schemes to minimize the module execution overhead for *store-based incremental checkpointing*. In addition, we address the implementation issue for the three proposed solutions to minimize the checkpoint cost. Finally, we design a flash-aware incremental checkpoint file format to accommodate the flash memory characteristics. We implemented these three incremental checkpointings in the mica2 motes on the SOS kernel (Han et al., 2005) and presented a detailed performance evaluation. To the best of our knowledge, this study is the first that focuses on the design and implementation of incremental checkpointing in MMU-less WSNs.

The remainder of this paper is organized as follows. Section 2 describes the previous checkpointings and introduces the SOS operating system. Section 3 presents the design and implementation of our full checkpointing and three incremental checkpointings. In addition, the full and incremental checkpoint file formats are also presented. The experimental results are shown in Section 4. Finally, Section 5 provides conclusions and future work.

2. Background

In Section 2.1, we first present the related work that also deals with the failures of the sensor nodes. Then, we review legacy checkpoint schemes. In particular, we perform a review of incremental checkpointing in Section 2.2. Finally, because our work is based on the SOS operating system, we introduce SOS in Section 2.3.

2.1. Related work dealing with the failures of sensor nodes

To deal with the failure of the sensor nodes, one possible solution is to deploy a sensor network with a set of additional nodes to repair the network in the event that sensor nodes fail. For example, FTSHM (Fault Tolerance in Structural Health Monitoring) searches the strategic locations of a structure and places additional nodes at these locations, so as to repair the network to guarantee a specified degree of fault tolerance (Bhuiyan et al., 2012). Similarly, a variety of protocols proposed to achieve k-coverage or k-connectivity to tolerate sensor faults (Zhou et al., 2004). Notably, a k-coverage network means each location in a field is covered by at least k sensors (Ammari and Das, 2012), while a k-connected network requires k node failures to disconnect the network (Bredin et al., 2010). To tolerate node failures, they compute the locations where an approximately minimum number of backup nodes are deployed to maintain k-coverage or k-connectivity.

Besides, when some of the sensor nodes fail, some areas can become unavailable. As a result, the routing of sensor data could encounter errors. To address this problem, researches have proposed the multi-path routing and hole detection and detouring algorithms (Gao et al., 2011; Lee et al., 2010). Nevertheless, all of the above fault management schemes do not consider the state loss problem. As a result, when a node fails, the valuable state embedded in it is lost.

2.2. Checkpointing

Checkpointing aims to provide recovery capability for application programs. However, the cost of checkpointing is high. Thus, many techniques have been proposed to reduce the checkpoint overhead. In general, these techniques can be classified in two categories (Plank et al., 1999). The first is the *latency hiding* technique that attempts to hide the execution of checkpointing from application programs. Well-known examples are forked checkpointing and copy-on-write checkpointing (Plank et al., 1995). The other approach is the *size reduction* technique that aims to minimize the amount of checkpoint data. Examples include incremental checkpointing (Plank et al., 1995) and memory exclusion checkpointing (Plank et al., 1999).

Of these techniques, incremental checkpointing is especially attractive. Incremental checkpointing utilizes the page protection