Evaluation of Communication Induced Checkpointing Approaches for Reconfiguration-Based Fault-Tolerance in Embedded Systems

Belal H. Sababha and Osamah A. Rawashdeh
Electrical and Computer Engineering Department
Oakland University
Rochester, Michigan, USA
{bhsababh, rawashd2}@oakland.edu

***Abstract*—** Reconfiguration-Based Fault-Tolerance is an approach to developing dependable safety-critical embedded applications, where redundant active or standby resources are used to cope with faults through a system reconfiguration at run-time. Compared to traditional hardware and software redundancy, it is a promising technique that may achieve dependability with a significant reduction in cost, size, weight, and power requirements. Reconfiguration necessitates using proper checkpointing protocols to support state reservation to ensure correct task restarts after a system reconfiguration. Communication Induced Checkpointing (CIC) protocols are well developed and understood for large parallel and information systems, but not much has been done for resource limited embedded systems. This paper implements four common CIC protocols in a resource constrained distributed embedded system with a Controller Area Network (CAN) backbone. An example feedback control system implementation is used for a case study. The four implemented protocols are described and performances are contrasted. The paper compares the protocols in terms of network bandwidth consumptions, CPU usages, checkpointing times, and checkpoint sizes in additional to the traditional measures of forced to local checkpoint rations and total number of checkpoints.

Keywords- Distributed Embedded Systems; Fault-Tolerance; Reconfiguration; Communication Induced Checkpointing; CAN.

I. INTRODUCTION

Reconfiguration-based fault-tolerance is an approach for developing dependable applications, where a system is automatically reconfigured at run-time to handle the event of a failed component. This approach, compared to traditional hardware and software redundancy, can achieve dependability at a reduced cost [1]-[6]. Reconfiguration requires using checkpointing protocols to support state preservation to allow task restarting, replacement, and migration.

Communication Induced Checkpointing (CIC) is one of three main categories of checkpointing protocols [7]. The other two categories are uncoordinated checkpointing, and coordinated checkpointing [7]. Uncoordinated checkpointing maximizes the autonomy of processes in deciding when a checkpoint is taken locally. However, due to the lack of coordination, many of these checkpoints are likely useless because they would, not be part of any consistent global checkpoint. A rollback process to a global checkpoint containing useless checkpoints may cause a cascaded series of rollbacks that may lead to the well known Domino Effect Problem [8]. Because the last taken checkpoint is not guaranteed to be useful, a process has to maintain more than a single checkpoint; therefore, it is required to do garbage collection periodically to free up some storage space [8].

Coordinated checkpointing protocols do not suffer from the Domino Effect. A process does not maintain more than a single checkpoint, therefore less storage overhead and no garbage collection is required. However, protocols in this category, due to coordination, suffer from large latency in committing an output [7].

Processes applying CIC protocols overcome the Domino Effect through piggybacking control information over application messages. CIC protocols give processes the autonomy of taking local checkpoints. To prevent useless checkpoints, processes take extra forced checkpoints. The decision to take a forced checkpoint is based on the control information carried with application messages. Checkpointing in general and CIC in particular has been the focus of researchers for some time [7]-[13]. However, available literature mostly target information systems and parallel supercomputers. Very little has been published in the field of resource constrained distributed embedded systems, where processor time, communication bandwidth, program memory, and stable storage availability are limited and valuable.

In this paper, several CIC protocols are implemented and evaluated on a resource limited real-time distributed embedded system for reconfiguration-based fault-tolerance purposes. The system utilizes a Controller Area Network (CAN) for communication between processing elements (PEs). The PEs are16-bit microcontroller units (MCUs). Each MCU features up to 48 MHz clock frequency, 12 Kbyte of RAM, 4 Kbyte of EEPROM, and 256 Kbyte of flash memory. CIC protocols were evaluated for two applications. The first is a simulated application executing on the PE test-bed, where tasks periods, message destinations, and message frequencies are set randomly. Secondly, the CIC protocols were applied

DOI: 10.5176_2010-2283_1.4.102

to a feedback-control system for an unmanned aerial vehicle (UAV).

The findings of our work presented here agree with what has been presented in the literature as results of simulations in terms of the number of checkpoints forced by each CIC protocol and the ratio of forced to local checkpoint. The paper illustrates more in depth results found from actual implementations of these protocols on a resource constraint embedded system environment. To the authors' knowledge, this work of implementing and evaluating CIC protocols is a unique effort that addressed resource constraint distributed embedded systems. The authors were not able to find any related work that implemented CIC protocols in similar environments and applications.

This paper has the following main contributions. (1) It confirms results from the available checkpointing literature by implementing and evaluating the CIC protocols on real embedded systems. (2) The paper shows that some of the best CIC protocols in terms of conventional metrics are not appropriate for embedded systems in terms of network bandwidth and CPU usage overhead. (3) Finally, the paper reports real numbers concerning checkpoint sizes, actual processing time overheads, and bandwidth usage in addition to the traditional number of checkpoints and forced to local checkpoint ratios.

The paper is organized as follows. Section II highlights the main concerns of embedded systems in the context of communication induced checkpointing as opposed to conventional information systems. Section III overviews the computation model, and defines the concept of Z-Paths and Z-Cycles. Section IV introduces communication induced checkpointing and describes the four CIC protocols that were implemented and evaluated. In section V, the experimental setup is shown. Results are illustrated in section VI. Findings are discussed in section VIII. Finally, Section VII concludes the paper.

II. Embedded System Constraints

Due to stringent cost, size, and power requirements, distributed embedded systems are typically very limited in resources compared to conventional information systems and supercomputers, for which checkpointing techniques were originally developed. These limitations can be summarized as follows:

i. Embedded systems have limited processing power. Processing overhead required by a checkpointing protocol has to therefore be minimized. Application execution delays are especially unacceptable in real-time and safety-critical applications.

ii. Embedded systems usually have limited amounts of local memory for the temporary storage of local checkpoints and global checkpointing state information. This limitation necessitates optimization of checkpointing protocols to reduce local memory requirements.

iii. Embedded systems usually can only afford a limited amount of non-volatile memory for the purposes of checkpoint storage. Hence, it is important to minimize the number of checkpoints forced by a protocol.

iv. Distributed embedded systems using a broadcast network (e.g., CAN) have limited network bandwidth. As a result, the network may be overwhelmed by messages induced by some protocols, as some of the protocols piggy-back more information than others.

v. Embedded system applications are often periodic. This periodicity implies repeated network patterns, which can be advantageous in deciding when to take local checkpoints. For example, Preißinger *et. al.* introduced an approach that optimizes effective checkpoint-intervals to reduce the overhead of communication induced checkpointing depending on the application and communication patterns [14].

III. Preliminaries

A. The Computation Model

The computation model for the distributed environment consists of n processes $\{P_1,P_2, \ldots, P_n\}$. All processes synchronize by piggybacking control information over application messages. Processes are executed by processing elements that do not have any shared memory in common. Communication between every pair of processes occurs over a directed logical channel. Communication is asynchronous and reliable. It is also assumed that no messages will be lost during communication.

Three different kinds of events are considered by each individual process: *internal*, *send*, and *receive* events. An *internal* event does not involve any kind of access to the communication link. A *send* event models the action of placing a message over one of the output channels that connect the current process (P_i) with the destination process (P_j). Moreover, blocking until a message is received on one of the input channels is modeled by the *receive* event.

A process (P_i) produces a sequence of events during its execution. Every event of this sequence moves P_i from its current state to the next state. A recorded state of a process is called a *local checkpoint*. $C_{i,x}$ is the x^{th} *local checkpoint* of process P_i. The sequence of events occurring between $C_{i,x-1}$ and $C_{i,x}$ is called a *checkpoint interval* ($I_{i,x}$). By looking at all the events in the distributed system as a whole, states of processes may become dependent on each other due to inter-process communication. Events produced by all the processes of the distributed

computation can be ordered by the well-known Lamport's *happened-before* ($\overset{hb}{}$) relation [15].

A *global checkpoint* is a set of *local checkpoints*, one from each process of the distributed computation. A *global checkpoint* is said to be *consistent* if and only if there exists no *happened-before* relation between any pair of local checkpoints in the set. A checkpoint that is not part of any consistent global checkpoint is a *useless* checkpoint [16]. Useless checkpoints cost processor execution time overhead, in addition to wasting of communication bandwidth and storage space. Furthermore, the presence of useless checkpoints may lead to what is called the *Domino Effect*, where a process rollback may cause the rollback of another process and so on until a consistent global checkpoint is found (probably the initial state). A checkpoint is useless if and only if it appears within a Z-cycle [16]. Processes are forced by checkpointing algorithms to take *forced checkpoints* to break candidate Z-cycles before they occur. The next subsection explains Z-path and Z-cycle notions introduced by Netzer and Xu in [16] and how they relate to useless checkpoints.

B. Z-Paths and Z-Cycles

A sequence of messages $[m_1, m_2, \dots, m_k]$, $k \geq 1$, from a checkpoint $C_{i,m}$ to another checkpoint $C_{j,n}$ is called a *z-path* from $C_{i,m}$ to $C_{j,n}$ if all of the following conditions are true [16]:

(a) $C_{i,m} \overset{hb}{} send(m_i)$

(b) $\forall m_i, i < k: \ receive\ m_i \overset{hb}{} send\ m_{i+1} \cup receive\ m_i \in I_{j,q} \cap send\ m_{i+1} \in I_{j,r} \cap q = r$,

(c) $receive(m_i) \overset{hb}{} C_{j,n}$.

In other words, the first message in the sequence has to be sent after $C_{i,m}$ is taken, while the last message has to be received before $C_{j,n}$ is taken. In addition, for any message, except the last one, the reception of a message must occur in the same or the preceding checkpoint interval in which the following message is sent. In this case, we say that this message sequence is a z-path from $C_{i,m}$ to $C_{j,n}$. If $i = j$ and $m = n$, .then this z-path is called a *z-cycle*. In the case of a z-cycle, the starting and ending checkpoints are the same checkpoint ($C_{i,m}$). Hence, it is said that the z-cycle includes $C_{i,m}$. In this case $C_{i,m}$ is considered as a useless checkpoint because it cannot be part of any consistent global checkpoint.

Figure 1 is a distributed computation pattern that we will use to clarify the notions and concepts mentioned earlier in this section. The following sets of message sequences: $[m_3, m_4]$, $[m_6, m_5]$, $[m_6, m_7]$ and $[m_3, m_4, m_2]$ are all z-paths. However, some of these z-paths are *causal* and some of them are *noncausal* [16].

A z-path ($[m_1, m_2, \dots, m_k]$, $k \geq 1$) is causal if the following is true:

$\forall m_i, i < k: \ receive\ m_i \in I_{j,q} \cap send\ m_{i+1} \in I_{j,r} \cap q = r \cap receive\ m_i \overset{hb}{} send\ m_{i+1}$.

If the previous condition is not true and the event of sending a message (m_{i+1}) at a process P_i happens before receiving the previous message in the sequence (m_i), then the z-path is noncausal [16]. In Figure 1, z-paths [m3,m4] and [m6,m7] are causal paths. On the other hand, z-paths [m_6, m_5] and [m_4, m_2] are examples of noncausal paths. The chain of messages [m_3, m_4, m_2] is an example of a z-cycle. Checkpoint $C_{k,1}$ is included in the z-cycle, so it is considered as a useless checkpoint. Note that the z-cycle shown in the Figure and any z-cycle in general is noncausal.

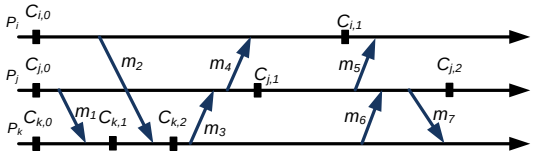

Figure 1. Distributed Computation Pattern

IV. CIC PROTOCOLS

CIC protocols are classified into Model-Based protocols and Index-Based protocols [7]. In both kinds, every process has two types of checkpoints: local checkpoints and forced checkpoints. A process takes local checkpoints independently from other processes depending on the application, while forced checkpoints are taken depending on the communication between processes to prevent useless local checkpoints.

In model-based protocols, processes maintain checkpointing and communication patterns to prevent the occurrence of useless checkpoints [9]. All model-based protocols focus on preventing communication patterns that may lead to z-cycles. In these protocols, processes are forced to take extra checkpoints to break candidate z-cycles. However, more than one process may take a forced checkpoint to break the same z-cycle, which results in more checkpoints than actually needed. Some of the most popular model-based protocols are the ones part of the FDAS (Fixed-Dependency-After-Send) family [9]: NRAS (No-Receive-After-Send), CAS (Checkpoint-After-Send), CBR (Checkpoint-Before-Receive), CASBR (Checkpoint-After-Send-Before-Receive), and FDI (Fixed-Dependency-Interval).

Index-based protocols, on the other hand, indices or timestamps are assigned to all checkpoints. These indices are piggybacked on application messages. Some index-based protocols go further by piggybacking more information than their own index, additional information may include information about other processes' checkpoints and communication pattern information that a process collected during its

communication with other processes in the network [10][11][17].

In this study, we evaluate two model-based protocols, and two index-based protocols. The first model-based protocol will be referred to as the CBS (Checkpoint-Before-Send) protocol [12], while the second model-based protocol is the MRS (Mark-Receive-Send) protocol [13]. The two implemented index-based protocols are: the BCS (Briatico, Ciuffoletti and Simoncini) [10], and the FI (Fully Informed) [17][18]. The protocols are popular in the literature, and that is why they were chosen to be evaluated here. The rest of the section will describe these four protocols in more detail.

The first protocol, CBS, is model-based and adapts at the extreme case by forcing a checkpoint before every message sending event [7]. This model was adapted by Bartlett in the implementation of the Tandem NonStop kernel as part of a fault-tolerant distributed computer system designed for online transaction processing [12].

In MRS, which is also a model-based protocol, M stands for Mark that means "to take a checkpoint," while R and S stand for "receive" and "send" respectively. A process in MRS forces a checkpoint before delivering a message that is not separated from its previous message-send event by a checkpoint. This will guarantee domino-effect free rollbacks [9][13].

The third protocol is an index-based approach known as the BCS (Briatico, Ciuffoletti and Simoncini) algorithm [10]. In this algorithm, each process maintains an index that is incremented every time a checkpoint is taken. The local index is piggybacked over application messages. A process, upon receiving a message, will compare its local index with the received one. A checkpoint will be forced if the received index is greater than the local index. Furthermore, the local index value is updated to equal the value received. A consistent global checkpoint is the set of local checkpoints stamped with the same index.

Finally, the forth protocol is an index-based protocol and the literature refers to it as the FI (Fully Informed) protocol [18]-[20]. The protocol which was presented by Helary *et. al.* in [17] piggybacks more than an integer index over application messages. In this protocol, every application message piggybacks one integer, one vector of integers, and two Boolean vectors (i.e., n+1 integers and 2n booleans ≡ 4n+2 Bytes). The integer is the index value (cl_i) of the sender process (P_i). The vector of integers ($ckpt_i[n]$) holds all the checkpoint sequence numbers of all other processes in the network up to the sender's (P_i) knowledge. Therefore, this vector is of size n, where n is the number of processes in the computation network. In addition, the two Boolean vectors are also of size n each. These two Boolean vectors are referred to as: $greater_i[n]$ and $taken_i[n]$. The previously mentioned piggybacked information, as well as a locally maintained vector ($sent\_to_i[n]$) are all used by the FI algorithm to detect possible z-cycles and to break them by forcing additional checkpoints [17]. A $greater_i[k]$ vector element is true if the value of the local sender's (P_i) index is greater than P_k's index upto P_i's knowledge. The $taken_i[k]$ vector element for all $k \neq i$ is true if a checkpoint is included within a z-path from the last checkpoint of P_k to the next checkpoint of P_i up to P_i's knowledge. The local and received vector elements: $ckpt_i[i]$ and $received.ckpt[i]$, respectively, will help a process to determine if there was a Z-path from the last checkpoint of P_k upto P_i's knowledge to the next checkpoint of P_i. The locally maintained vector $sent\_to_i[k]$ keeps track of whether P_i has sent a message to P_k since its last checkpoint or not [17]. All this information is used in the following condition by a process P_i to decide whether to force a checkpoint or not:

$$\exists k: sent_{to_i}\ k\ \cap received.greater\ k \cap received.index > index_i \cup (received.ckpt\ i\ = ckpt_i\ i\ \cap received.taken\ i\ )$$ [17].

BCS, FI and index-based protocols, in general, guarantee that all checkpoints with the same index will form a consistent global checkpoint. In all CIC protocols, the goal is to minimize the number of forced checkpoints while maintaining a consistent global checkpoint.

V. Experimental Setup

In this study, two experiments were conducted. In the first, checkpointing of a simulated application was conducted where tasks periods, message destinations, and message frequencies are set randomly. The test was performed on CAN networked PEs using four CIC protocols to compare and confirm results in the available literature. A test-bed was used to carry out this experiment. The test-bed is built from a number of microcontrollers communicating with each other through CAN (Controller Area Network) [21].

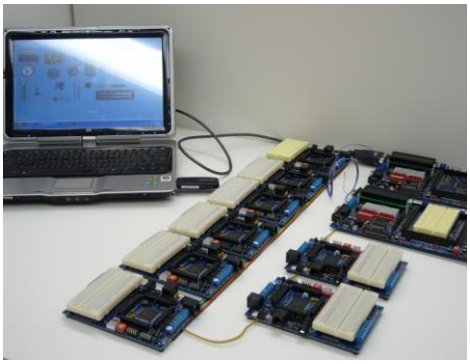

Figure 2. Test-Bed Hardware and User Interface.

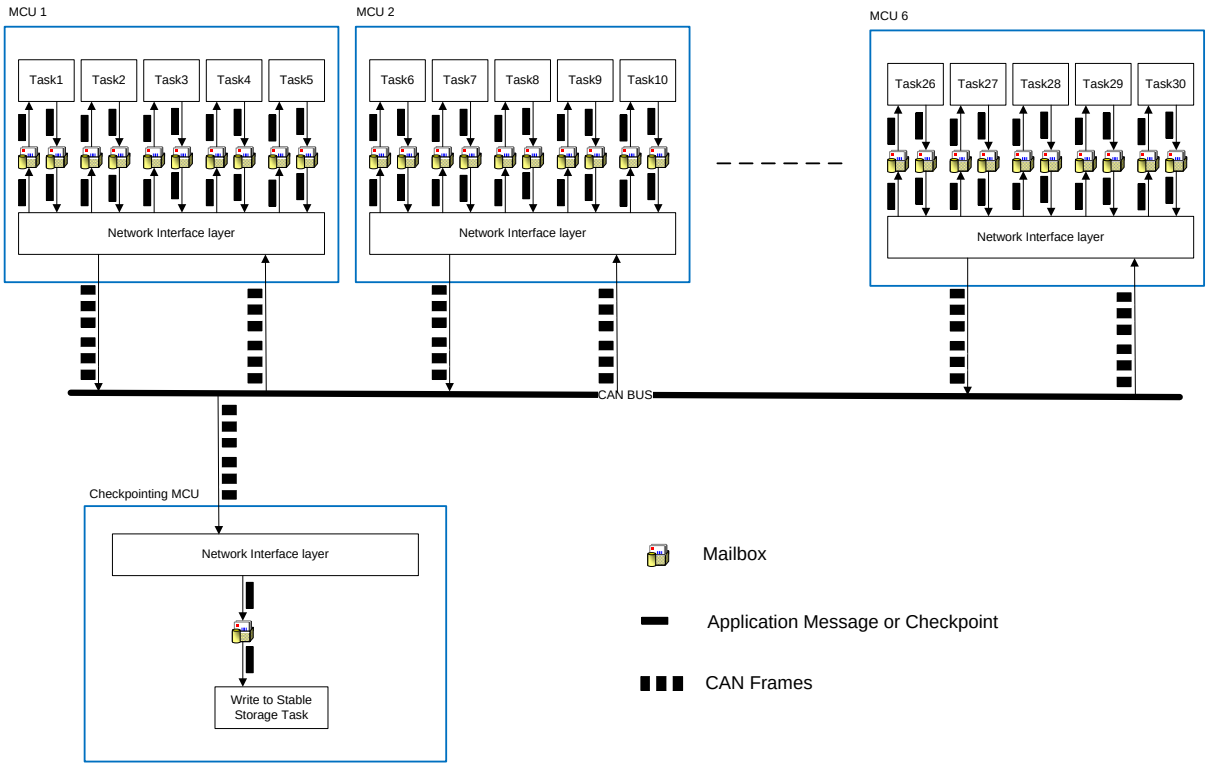

Figure 3. Software Communication Model – First Experiment

The number of microcontroller units used in the test-bed is variable. The user can easily add or remove microcontrollers to the test-bed. The microcontrollers, which serve as processing elements (PEs), carry out user tasks execution. A real-time operating system (OS), µC/OS-II^{TM}, running on each microcontroller manages hardware resources, provide inter-task communication, as well as providing other basic and necessary services. The test-bed provides electively the user with a system manager processor that may be used for managing the distributed application. The test-bed also provides the user with monitoring, data acquisition, and fault-injection tools.

The hardware, pictured in Figure 2, used to build the test-bed is composed of a group of Freescale HCS12 microcontroller units (MCUs) [22]. Specifically, the MC9S12DP256B model is used. An MC9S12DP256B microcontroller is a 16-bit device composed of a 16-bit central processing unit (HCS12 CPU) and many I/O options. The MCUs used in the test-bed are connected through a CAN network. Physically CAN is a twisted pair multidrop cable ranging from 1,000 meters to 40 meters in length operating at 40 Kbps to 1 Mbps data rates respectively. CAN is a message based protocol, where each message has an identifier, which can be treated differently depending on the application. Every node on the network receives all messages and is typically set up to process messages of interest as identified by their message IDs.

CAN has been selected to connect the microcontrollers due to its popularity (more than 2 billion nodes have been sold since the protocol's development in the early 1980s [23]), high data rates (1Mbps at 40m bus length), fault-tolerance capabilities (e.g., acknowledgment bits, differential signaling, and the ability to communicate through one of the two lines at lower data rates in case of damage [24]). The number of nodes could be any number up to 110 nodes.

For the CIC protocols evaluation, the number of networked processes was varied between 5 and 30 tasks, executed by 1 to 6 MCUs, and up to 5 processes (tasks) per MCU. All on-chip and off-chip task communication occurs over the CAN bus. A network interface layer was implemented on every MCU to make the inter-task communication transparent to the application. A process sends all checkpoints and application messages to this software layer through OS provided mailboxes. The software layer disassembles the incoming messages, forms CAN-compatible frames, and writes these frames to the CAN bus. On the other hand, the network interface layer on a certain MCU monitors the CAN bus and reads all CAN frames who's destination is one of the tasks executed by the same MCU. The software layer then assembles all related frames into a single

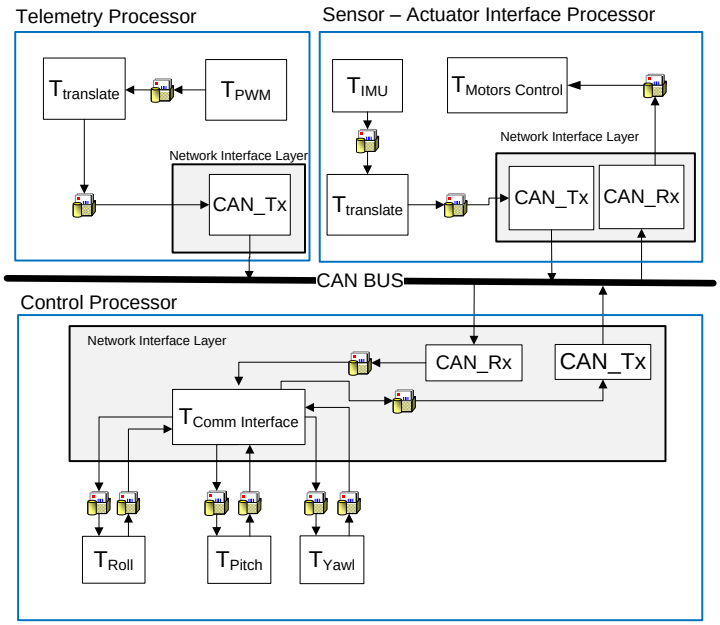

Figure 4. UAV Avionics system block diagram

message and forwards it to its destination through OS provided mailboxes. A dedicated MCU monitors the CAN bus, and reads all frames carrying process status checkpoints. In this paper, the dedicated MCU will be referred to as the checkpointing MCU. Figure 3 shows the software communication model. In all experiments, the MCUs were operating at 48MHz clock rate. The CAN network baud rate is set to 94.117 kbaud.

In the second test, the same four CIC protocols were applied on the tasks of an avionics feed-back control system of a Quadrotor unmanned aerial vehicle (UAV) being developed at Oakland University [25]. In this test, three HCS12 MCUs execute different UAV tasks. The main tasks running are: $T_{\_Comm,}$ $T_{\_Roll}$, $T_{\_Pitch}$, $T_{\_Yaw}$. The $T_{\_Comm}$ task forwards the current and desired attitude angles to the other three tasks through OS provided Mailboxes. $T_{\_Roll}$, $T_{\_Pitch}$, and $T_{\_Yaw}$ implement the roll, pitch, and yaw stability PID controllers respectively. The real-time OS executes the tasks on each of the three MCUs at the proper rates. All MCUs are connected to a CAN bus. A forth dedicated MCU collects checkpoints written by other MCUs from the CAN bus and stores them to stable storage. Figure 4 shows the block diagram of the system. MCUs in this test were also operating at 48MHz clock frequency, and communicating over a 94.117 Kbaud CAN bus speed.

As discussed above, local checkpoints in CIC protocols are taken depending on the application. Often, it may be beneficial to take a checkpoint when the process status size is minimal. Nevertheless, for other applications periodic checkpointing may be required. In this work, the focus is on evaluating CIC protocols in embedded systems, where executing the same software task in a periodic manner is usually the case. Hence, for both experiments, local checkpoints were taken periodically every fixed number of execution cycles.

VI. RESULTS

This section shows the results observed from the two experiments. Figure 5, shows the results from the simulated application experiment described in the previous section. It shows the total number of checkpoints as a function of process count for each of the four evaluated protocols. As observed from the figure, the total number of checkpoints for the two model-based protocols (CBS and MRS) are much higher than the two index-based ones (BCS and FI). Hence, the model-based protocols force more checkpoints than required. And the gap between the two kinds of protocols increases as more processes are added to the network.

In order to get a better picture of the number of checkpoints forced by each protocol, Figure 6 plots the forced to local checkpoint (F/L) ratio as a function of the number of processes in the network. The F/L ration is plotted for the four protocols evaluated in the first experiment with the simulated application.

Both Figure 5 and Figure 6 illustrate that the index-based protocols (BCS and FI) perform better than the other two model-based protocols in terms of total number of checkpoints and forced to local checkpoint ratio. The performance of BCS and FI is further studied below. FI show better performance in terms of the number of forced checkpoints. Figure 5 shows the sum of local and forced checkpoints, and

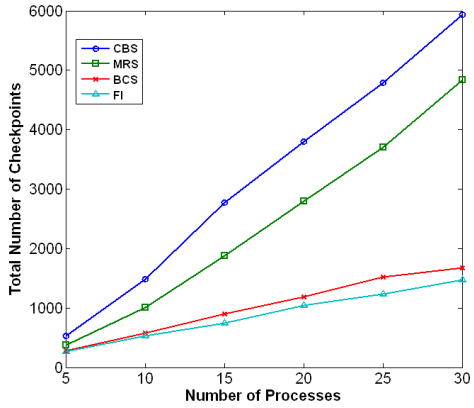

Figure 5. Simulated application experimental results (Total # of Checkpoints)

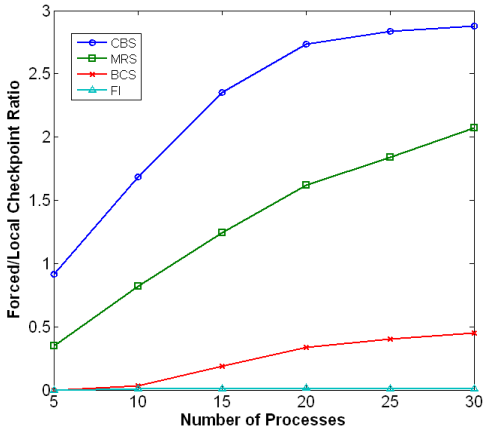

Figure 6. Simulated application experimental results (Forced/Local Checkpoint Ratio)

the two sums may look close enough because the number of local checkpoints taken periodically is relatively much higher than the forced checkpoints in both protocols. Therefore, this observation is highlighted in Figure 6 better than Figure 5.

For a better evaluation of the two index-based protocols on the embedded test-bed, the CAN bus traffic load is illustrated in Figure 7. The figure plots the average and peak bus load percentages for BCS and FI. FI out performed BCS in terms of forcing less number of checkpoints, but this comes with the expense of piggybacking more information. The relatively high amount of piggybacked information (compared to BCS) overwhelmed the network, and caused it to almost saturate as the number of processes got close to 30 nodes.

CAN frame messages do not carry more than 8 bytes of data. The network interface layer in the case of FI will need more clock cycles to convert application messages into CAN frames compared to BCS. This is because FI needs to piggyback much more information than BCS.

The increased amount of required clock cycles is considered as processing overhead, and may affect original software tasks by competing with it on processing resources.

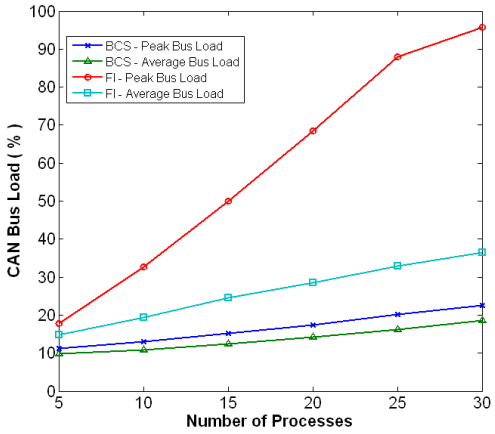

Figure 7. Simulated application experimental results (CAN Bus Load)

Figure 8 and Figure 9 compare the four CIC protocols in terms of total number of checkpoints and Forced/Local (F/L) checkpoint ratio for the four main tasks running on the attitude control processor in the UAV control system experiment. Figure 10 shows the CPU usage (µS) during a 60 second interval for the T_{Comm} task. The execution time is shown without applying checkpointing and with applying the four different protocols.

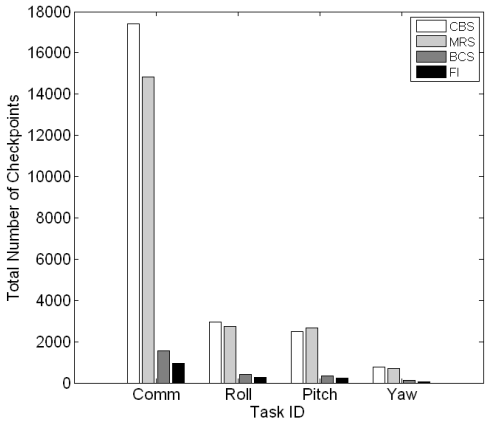

Figure 8. Attitude control application experimental results (Total # of Checkpoints)

Table 1 shows the worst case execution times for the tasks of the attitude control processor after applying the two index-based checkpointing protocols (BCS and FI). The second column is the number of times a task was executed by the OS over a period of 60 seconds. Third and fourth columns show the time (µS) needed for a single time execution of a task without checkpointing and the time for taking a

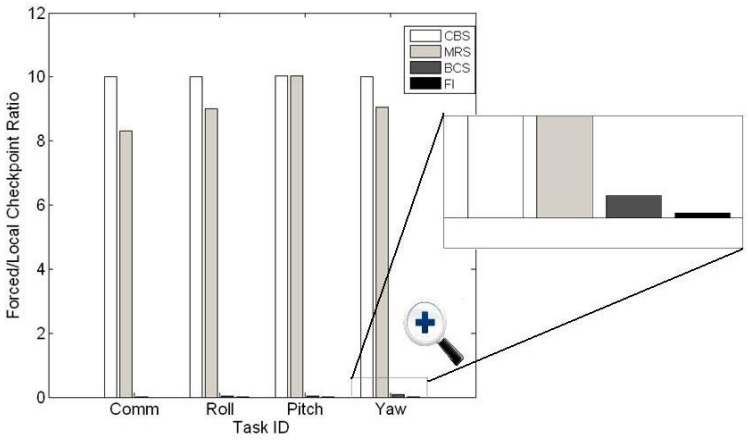

Figure 9. Attitude control application experimental results (Forced/Local Checkpoint Ratio)

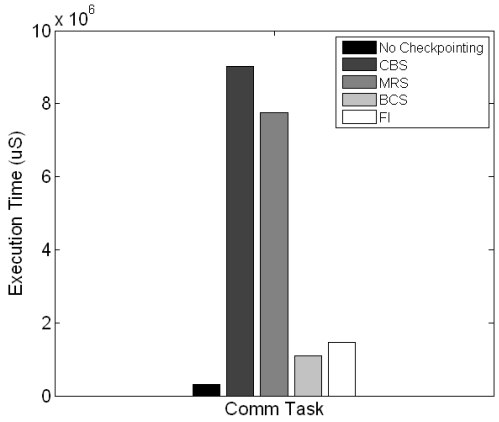

Figure 10. One Task CPU Usage with and without checkpointing

Table 1: Attitude control tasks execution times for BCS and FI

Task ID	# of Executions		No checkpointing (µS)	Checkpoint Time (µS)		CPU Usage With Chekpointing (µS)		Checkkpoint Size (B)
	BCS	FI		BCS	FI	BCS	FI	
T_Comm	15613	9522	312260	500	1345	1093760	1470880	40
T_Roll	4147	2510	352495	88	145	390071	250035	8
T_Pitch	3437	2315	292145	88	145	323385	230560	8
T_Yaw	1061	541	90185	88	145	100305	53960	8
Idle			58952915			58092479	57994565	

complete checkpoint respectively. Column five illustrate the processor time (μS) allocated to a task after applying checkpointing. The last column is the size (Bytes) of a single checkpoint. The next section will discuss all these results and explain the findings.

VII. DISCUSSION

The simulated application results agree with results from simulations reported for information and parallel processing systems found in the literature, in terms of the number of forced checkpoints and F/L checkpoint ration. The results show that the two index-based protocols outperformed the other two model-based protocols. However, the FI protocol had a much better F/L checkpoint ratio.

On the other hand, results from the first experiment show that the CAN bus load in the case of the FI protocol was the heaviest between all other protocols. This is a manifestation of the large amount of information the protocol had to piggyback over application messages.

Moreover, for the attitude control application, it is found that there was not much difference in terms of number of checkpoints between CBS and MRS protocols. It is believed that these close results found for the two model-based protocols are due to the fact that this attitude control is a periodic system, as is the case with many embedded systems. Furthermore, the BCS index-based protocol showed a significant amount of improvement in terms of the number of checkpoints and execution times as expected. However, the FI protocol had the fewest amount of checkpoints and it required more CPU resources than the BCS protocol (Figure 10). This is due to the additional time required to piggyback the information required by processes, as well as to process all this information. This means that the execution frequency of a task in the case of the FI protocol is less than the frequency in the case of any other protocol. The results in Figure 10 show that the two model-based protocols had a higher CPU usage than the FI protocol. This is because the two model-based protocols had a higher execution cycle rate and thus got the opportunity to use the CPU more frequently in the 60 seconds time period.

Table 1, column 1 show that the BCS protocol roughly executed two times more than the FI protocol. This relatively high amount of execution time overhead in the FI protocol caused it lower execution frequency. However, in real-time embedded system applications, lowering the execution frequency has a critical impact on the overall performance of the application. In the quadrotor UAV attitude control application, for example, the execution frequency of the PID control loops was not enough to keep the UAV stable and respond to commands correctly.

From the findings, the authors believe that a simple index-based CIC protocol such as the BCS protocol fits better in embedded system applications than other protocols that piggyback more information to reduce the amount of forced checkpoints. Meeting deadlines in this case is critical and is at least as important as saving checkpoints.

VIII. CONCLUSION

This paper overviews the work of implementing and evaluating CIC checkpointing protocols to support reconfiguration-based fault-tolerance in CAN-based distributed embedded systems. Four popular CIC protocols were evaluated on a resource constrained embedded system. Two experiments were conducted over CAN networked 16-bit microcontrollers serving as PEs. The first experiment evaluated the four protocols on a simulated application with random task periods, message destinations, and message frequencies. The second experiment evaluated the same protocols on the tasks of an avionics feed-back control system of a Quadrotor UAV. The findings in this research agree with available literature for large information systems and parallel computing in the context of number of checkpoints and forced to local checkpoint ratios. The findings also showed that reducing the number of forced checkpoints comes with the expense of overwhelming the network bandwidth as well as increased processing overhead, which is not suitable for embedded systems that have to meet certain deadlines. The main contributions of this paper are: confirmation of simulation results form available information system literature on distributed embedded systems, illustrating the inappropriateness of the FI protocol for resource constrained systems due to the required bandwidth and CPU overhead, and, finally, reporting real measures of checkpoint sizes, processing time overheads, and network bandwidth usage, in addition to the conventional metrics in realistic embedded system applications. It is believed that the reported information may be beneficial to embedded system designers considering checkpointing approaches.

REFERENCES

[1] Strunk, Elisabeth A., John C. Knight, and M. Anthony Aiello, "Distributed Reconfigurable Avionics Architectures," 23rd Digital Avionics Systems Conference, Salt Lake City, UT, October 2004.

[2] R. P. Dick, N. K. Jha. "CORDS: Hardware-Software Co-Synthesis of Reconfigurable Real-Time Distributed Embedded Systems." IEEWACM International Conference on Computer Aided Design, pages 62-68, San Jose, California, November, 1998.

[3] R. Feldmann, C. Haubelt, B. Monien, and J. Teich. "Fault Tolerance Analysis of Distributed Reconfigurable Systems Using SAT-Based Techniques," In P. Y. K. Cheung, G. A. Constantinides, and J. T. de Sousa, editors, Field-Programmable Logic and Applications, Lecture Notes in

Computer Science (LNCS), volume 2778, pages 478–487, Berlin, Heidelberg, Sept. 2003. Springer.

[4] Rawashdeh, O., and Lumpp, J., "Run-Time Behavior of Ardea: A Dynamically Reconfiguring Distributed Embedded Control Architecture," IEEE Aerospace Conference, IEEEAC Paper# 1516, March 2006.

[5] M.Eisenring, M.Platzner, "A framework for run-time reconfigurable systems", The Journal of Supercomputing, v.21, pp.145–159, 2002

[6] Thilo Streichert, Dirk Koch, Christian Haubelt, and Jrgen Teich, "Modeling and design of fault-tolerant and self-adaptive reconfigurable networked embedded systems," EURASIP Journal on Embedded Systems, Special Issue on Field-Programmable Gate Arrays in Embedded Systems., 2006.

[7] E. N. (Mootaz) Elnozahy, Lorenzo Alvisi, Yi-min Wang and David B. Johnson, "A Survey of Rollback-Recovery Protocols in Message-Passing Systems," ACM Computing Surveys, vol. 34, No. 3, pp. 375–408, September 2002.

[8] B. Randell, "System Structure for Software Fault Tolerance," IEEE Trans. Software Eng., vol. 1, no. 2, pp. 220-232, June 1975.

[9] Y.M Wang, "Consistent global checkpoints that contain a given set of local checkpoints," IEEE Trans. Computers, vol. 46, no. 4, pp. 456-468, Apr. 1997.

[10] D.Briatico, A. Ciuffoletti, and L. Simoncini, "A Distributed Domino-Effect Free Recovery Algorithm," In Proc. Of the IEEE International Symposium on Reliability in Distributed Software and Database Systems, pp.207-215, October 1984.

[11] J. M. Helary, A. Mostefaoui, R.H.B. Netzer, and M. Raynal, "Preventing Useless Checkpoints in Distributed Computations," In Proc. Of IEEE International Symposium on Reliable Distributed Systems, pp. 183-190, 1997.

[12] J. F. Barltlett, "A Non Stop Kernel," In Proc. of the Eighth ACM Symposium on Operating Systems Principles, pp. 22–29, 1981.

[13] D. L. Russell, "State restoration in systems of communicating processes," IEEE Transactions Software Engineering, Vol. 6, No. 2, pp. 183–194, 1980.

[14] J¨org Preißinger and Mark Pfl¨uger, "Compiler supported interval optimisation for communication induced checkpointing," Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA'07, volume II, pages 550–556, Las Vegas, NV, June 2007. CSREA Press.

[15] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," Commun. ACM, vol. 21, no. 7, pp. 558-565, 1978.

[16] R.H.B. Netzer and J. Xu, "Necessary and Sufficient Conditions for Consistent Global Snapshots," IEEE Trans. Parallel and Distributed Systems, vol. 6, no. 2, pp. 165–169, Feb. 1995.

[17] J.-M. Helary, A. Mostefaoui, R.H.B. Netzer, M. Raynal, "Communication-based prevention of useless checkpoints in distributed computations," Distributed Computing, 13:29-43, 2000.

[18] Jichiang Tsai, "An Efficient Index-Based Checkpointing Protocol with Constant-Size Control Information on Messages," IEEE Transactions on Dependable and Secure Computing, Vol. 2, No. 4, pp. 287-296, 2005.

[19] Y. Luo and D. Manivannan. "FINE: A Fully Informed aNd Efficient communication-induced checkpointing protocol," In IEEE Proceedings of the 3rd International Conference on Systems (ICONS'08), pages 16–22, Los Alamitos, CA, USA, 2008. IEEE Computer Society.

[20] Yi Luo and D.Manivannan. "Theoretical and Experimental Evaluation of Communication-Induced Checkpointing Protocols in F_E Family," In Proceedings of the 27th IEEE International Performance Computing and Communications Conference (IPCCC 2008)), December 7-9, 2008, Austin, Texas, USA pages:217 - 224.

[21] 20 Belal H. Sababha, Osamah A. Rawashdeh, and Guangzhi Qu, "A Test-Bed for Reconfiguration-Based Fault-Tolerance in Distributed Embedded Systems," The International Conference on Information and Communications Systems (ICICS2009), Paper # 500, Amman, Jordan, Dec 20, 2009.

[22] Freescale's website, Available: http://www.freescale.com/

[23] CAN in Automation Organisation website, Available: http://www.can-cia.org/

[24] The Official site of the Embedded Development Community, Available: http://www.embedded.com/columns/murphyslaw/13000304?_requestid=142578

[25] O. A. Rawashdeh, H.C. Yang, R. AbouSleiman, and B. H. Sababha, "Microraptor: A Low-Cost Autonomous Quadrotor System," Proc. of the 2009 ASME/IEEE International Conference on Mechatronic and Embedded Systems and Applications (MESA09), DETC2009-86490, San Diego, CA, Sep 1, 2009.

Belal H. Sababha, Ph.D. is a Powertrain Controls Senior Engineer at Chrysler Group LLC. He received his Ph.D. degree in Electrical and Computer Engineering – Embedded Systems from Oakland University in 2011. His B.Sc. and M.Sc in Computer Engineering were received from Yarmouk University and Jordan University of Science and Technology in the years 2000 and 2006 respectively. He has taught electrical and computer engineering undergrad and grad courses at Oakland University/Michigan/USA and at Yarmouk University/Jordan. His research concentration areas are UAV development, Biomedical instrumentation, sensor communication, routing in wireless ad hoc networks, embedded RTOS, CAN networks, distributed embedded systems, graceful degradation in embedded systems, rapid prototyping, machine vision, and automotive onboard diagnostics. Belal has several years of experience in IT related engineering and management careers. He is a member of AIAA, ASME and IEEE.

Osamah A. Rawashdeh, Ph.D., P.E. is an Assistant Professor in the Department of Electrical and Computer Engineering at Oakland University. He received his BS with honors, MS, and PhD in Electrical Engineering from the University of Kentucky in 2000, 2003, 2005 respectively. He absolved internships at Daimler Benz AG and at SIEMENS AG and is a member of ACM, AIAA, AUVSI, ARRL, and a senior member of IEEE. His research interests include embedded systems design, fault-tolerance, and reconfigurable computing. Before joining Oakland University in the fall of 2007, he served as a Lecturer the Department Electrical and Computer Engineering at the University of Kentucky. Dr. Rawashdeh is a licensed Professional Engineer in the State of Michigan.