An Architecture for Online-Diagnosis Systems supporting Compressed Communication

Seungbum Jo*, Markus Lohrey*, Damian Ludwig†, Simon Meckel†, Roman Obermaisser†, Simon Plasger*
*Theoretical Computer Science, University of Siegen, Germany
seungbum.jo@uni-siegen.de, lohrey@eti.uni-siegen.de, simon.plasger@student.uni-siegen.de
†Embedded Systems Group, University of Siegen, Germany
damian.ludwig@uni-siegen.de, simon.meckel@uni-siegen.de, roman.obermaisser@uni-siegen.de

***Abstract*—With its ability to detect, identify and, if applicable, recover from occurred faults, online-diagnosis can help achieving fault-tolerant systems. A sound decision on an occurred fault is the foundation for fault-specific recovery actions. For this, typically a large amount of data has to be analyzed and evaluated. A diagnostic process implemented on a distributed system needs to communicate all those data among the network which is an expensive affair in terms of communication resources and time. In this paper we present an architecture for a distributed online diagnosis system with real time constraints that supports data compression to reduce the communication time. We further present a lossy compression method with a guaranteed compression ratio that is suitable for real time purposes.**

I. Introduction

In safety-critical application domains such as avionics, health care or industrial automation, systems must provide their services with a high reliability. Even in the case of a fault in a hardware or software component or due to external influences, an acceptable quality of the services must be ensured. Typical faults are faults in the design of a component or a system, transient or permanent hardware faults or erroneous user operations amongst others. Applying fault-diagnosis methods to the system its reliability and stability can be significantly increased. In the widely-used *offline* fault-diagnosis process information, sensor data or fault indications exclusively get stored for later analysis. However, utilizing *online* fault-diagnosis with the ability to detect a fault, identify its location and potential effects at run time of the system, in combination with interactions with the system, e.g. reconfigurations, a fault-tolerant system can be formed.

Enabling the diagnostic algorithms to make correct and fast decisions on fault detection and identification, often a huge amount of raw data of a variety of sensors as well as system input and output parameters have to be processed, analyzed and stored during the diagnostic process. Considering that these tasks are computationally intensive in addition with the fact that the diagnostic process allows a concurrent processing of different tasks, parallelism can be exploited. By executing tasks on separate processing units within a distributed network, resources are planned meaningfully. According to the diagnostic process the calculated data must be consequently made available to the other units as a message via the network.

In [6], an inference-based system for active diagnosis in distributed embedded systems is presented, where diagnostic information is inferred using SPARQL[1] queries and stored in a distributed real-time database. The needed real-time guarantees are supported by a time-triggered schedule which controls the execution of diagnostic tasks and the replication of the database.

The quality of diagnostic decisions highly depends on the available amount of data. A limited storage capacity of the local real-time databases may therefore narrow the performance of diagnostic decisions, e.g. of long term trend analyses. Likewise, the required time that is needed to conclude a specific fault from a first symptom can be seen as quality characteristic of the diagnostic framework. It is the goal to minimize this time as much as possible. Since the diagnostic process requires both the storage and the fast exchange of a huge amount of data within the distributed network, the DAKODIS project deploys data compression for online fault-diagnosis. The data compression is a feasible instrument which can help to save bandwidth and hence provide stronger real-time guarantees or save communication resources. Additionally, it might save storage which can be used to store more diagnostic information in the real-time database, to improve the quality of the fault-diagnosis in terms of a more precise and extensive fault detection and identification.

To support compression without loss of real-time guarantees, one needs to obtain realistic bounds for the compression ratio and the computational time needed for compression and decompression. These bounds can then be used to consider compression for resource planning.

In this paper we present an architecture for a distributed online-diagnosis system with real-time constraints that supports compression for communication and is controlled by a static time-triggered schedule.

II. Model

Our system model consists of three parts: the logical model describing diagnostic tasks, the physical model describing a network and the compression model, describing a set of different compression algorithms and schemes. These models are needed to define a fourth model for scheduling diagnostic tasks to the network using data compression for communication.

[1]https://www.w3.org/TR/sparql11-query/

A. *Logical model*

The prerequisite for fault-diagnosis, i.e. the detection and identification of faults, is knowledge about the process or the system in order to distinguish a faulty process behavior from a healthy one. Often, knowledge about the system is available from different sources. If a mathematical model of the system is available, observer based fault classifiers can be utilized [4]. In signal-based fault analysis, the knowledge about the process is typically available in the form of a signal model (signal relations), process parameters (e.g. limits of a sensor signal) and corresponding digital signal processing techniques [4]. The system behavior can also be extracted from a large amount of historic data by utilizing digital signal and data processing techniques such as machine learning, amongst others [7]. A knowledge base combines the information about the healthy process behavior and information about fault indications.

The process of fault-diagnosis from a detection of a first fault-indication up to distinguished identification of a fault type and its location requires to extract, analyze, interpret and merge a multitude of diagnostic information. The diagnostic information is extracted in different manners. Analytical knowledge about the process, e.g. the signal model and corresponding processing techniques such as correlation or trend analyses, are utilized to extract characteristic information (features) about the signals which provides evidence of the current process state. Faults in the process are reflected in the measured signals and the diagnostic features, accordingly. In a complex system, several potential faults (e.g. of different components) may lead to the same faulty signal behavior of a monitored signal. In such a scenario, the fault detection is followed by a process of evaluating confirmative and unsupportive diagnostic information, respectively, in order to include or exclude particular faults, eventually specifying the actually occurred fault. The diagnosis of different faults requires the execution of different signal processing and feature extraction tasks in a defined order.

The dependencies between the diagnostic operations are modeled in a diagnostic directed acyclic graph (DDAG), denoted as $G = (T, E, (\ell_e)_{e \in E})$, with T being a set of tasks t and E being a set of ordered pairs (t, t') modeling a precedence relation between two tasks. Edges from E are also called *logical channels* or just *channels*. For a channel e, the number $\ell_e \in \mathbb{N}$ specifies the bit length of the data values sent via channel e. Thus, over a channel e a stream of data values is sent and every data value is encoded using ℓ_e bits.

Task t' depends directly on t, iff $(t, t') \in E$. A task may depend on multiple others and in turn may work as a prerequisite for other tasks. In a directed acyclic graph we can clearly identify all predecessors and successors of any task.

The DDAG combines the knowledge about the process and potential faults, i.e. how far faults effect a change of measured signals. Furthermore it describes the signal processing techniques (at the nodes) to extract the diagnostic features. With the edges showing the necessary information to be exchanged, the fault inference process is modeled. In order to allow the node operations (tasks) to be scheduled to the processing units of a distributed network, the DDAG design underlies certain rules which are abstracted in the following:

- A diagnostic task produces characteristic information (no intermediate results).
- A task may comprise multiple processing steps.
- A task uses as little as possible input information.
- For the order of fault reasoning (seeking confirmative or unsupportive information for a specific fault) several factors (e.g. probability of a fault's occurrence, computational cost of the diagnostic operation required) are taken into account.
- Fault reasoning and intermediate conclusions are allowed in every subgraph.

In the DAKODIS framework the processing steps are encapsulated into tasks which are executed at the processing units of a distributed network. According to its objective, a task utilizes modularized algorithms of the fields preprocessing of sensor or process data, diagnostic information extraction, further processing (e.g. combination) of diagnostic information and data compression.

All necessary data is made available to the dedicated nodes via messages through the network. As the diagnostic process is performed at run time on a distributed system the inference on faults is decomposed in the temporal and spatial domain. In case of a fault, the attained fault indications initially provide evidence for a certain fault but may yet hold inconclusive information at a certain node of the DDAG (e.g. confirmative information from other nodes is not yet available). The degree of confidence of a correct fault identification increases with more information merged.

B. *Physical model*

Our network model consists of a set C of computation nodes and a set R of routers, where $C \cap R = \emptyset$. The graph representing the network is defined as $\mathrm{Net} = (V, L)$, with $V = C \cup R$ and $L \subseteq (R \times R) \cup (C \times R) \cup (R \times C)$ being an undirected edge relation. The definition of L assures that a computation node can be directly connected to one or more routers but not to any other computation node. Only computation nodes can execute tasks and only routers are able to forward messages.

Figure 1 shows the architecture of the proposed system. Every computation node has access to a locally stored part of the distributed real time database that can be queried with SPARQL queries. This database holds all information needed to execute the diagnostic tasks located on this node and also stores the data produced by these tasks. The synchronization of the database among all nodes of the network is done by a dedicated synchronization unit available on each node. This unit also purges expired data from the storage as needed. As the communication between the nodes should use compression, the synchronization unit should compress outgoing messages, if the overhead is reasonable compared to the utility. For incoming compressed messages the unit needs to perform the decompression and write the newly arrived data to the

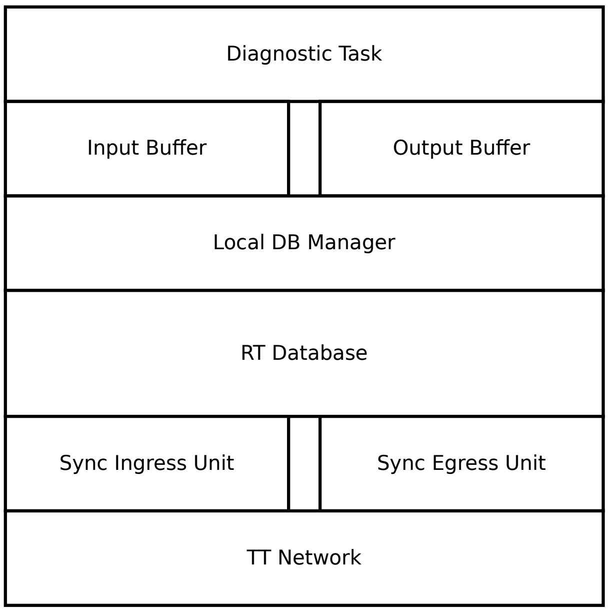

Fig. 1. Physical model architecture

database. Compute nodes may buffer outgoing messages. In contrast, routers are not able to do this. Instead, they must forward incoming messages directly.

The tasks running on a node do not have direct access to the database. Instead, they read from a task-exclusive input buffer and write their results to an also task-exclusive output buffer. These buffers are managed by a local database manager, which is aware of every tasks' needs. As the diagnostic tasks can operate on sliding time windows, the manager will only replace data which is not related to the current window, leaving all other data untouched. This reduces read-operations on the database and write-operations on the input buffers. The buffer size and the SPARQL queries needed to collect the data from the database are design-time parameters of the diagnostic task.

To put everything together, the local database manager, the synchronization unit and the execution of diagnostic tasks are controlled by a time triggered schedule. For that purpose the scheduler knows about each tasks' fireing rate and the worst-case execution times of all involved processes. The schedule is calculated at design-time and does not change during runtime. It considers resource allocation, path- and slot-selection and compression, while trying to keep the makespan minimal.

C. Compression model

The DAKODIS architecture is designed to work with different compression schemes. In order to provide real time guarantees the following parameters have to be known:

- the worst case compression time (WCCT) for a data value,
- the worst case decompression time (WCDT) for a compressed data value,
- and the worst case compression ratio (WCCR).

Formally, a compression scheme is a tuple $Z = (\ell, k, ct, dt)$, where $\ell \in \mathbb{N}$ is the bit length of an input data value, $k \in \mathbb{N}$ is the maximal bit length of a compressed data value, $ct \in \mathbb{N}$ is the WCCT, and $dt \in \mathbb{N}$ is the WCDT. Thus, a compression scheme abstracts from a concrete compression algorithm that receives a sequence of ℓ-bit strings and transforms it into a sequence of bit strings of length at most k. The time needed to convert a single ℓ-bit string into its compressed output (a bit string of length at most k) is at most ct, and the time needed to recover the original ℓ-bit string from the output is at most dt. A compression scheme is applicable to a channel as defined in II-A, if the bit length ℓ_e of the data values transmitted via channel e is ℓ. Note that the WCCR is k/ℓ. To profit from compression, we want to have $k < \ell$. This is clearly not possible using lossless compression. In section III we will propose a lossy compression algorithm with $k < \ell$. The algorithm fails to transmit data values with a small probability, but those data values that are transmitted can be perfectly recovered at the receiver.

D. Scheduling

Calculating a schedule is an important task when designing a real time embedded system. Basically, the schedule provides information about when and where a task can be executed without violating resource restrictions or dependency relations. For DAKODIS we choose a non-preemptive, static scheduling model similar to *partitioned scheduling*. As discussed in [3], a non-preemptive scheduling model has advantages on multi-core systems or distributed systems, as the overhead for migrating a task is more difficult to predict in case of preemptive scheduling. The disadvantage of non-preemptive tasks reducing the responsiveness of a system is not valid for multi-core systems, since the natural parallelism of such a system can hide this latency [3].

We need to consider task allocation, as the distance of two tasks in the network has an impact on the time needed for communication, thus on the makespan. An allocation function is a mapping $A : T \rightarrow C$ which maps tasks to computation nodes. We require that for every channel $e = (t, t') \in E$ and $A(t) \neq A(t')$ there exists a path of the form $A(t), r_1, \ldots, r_n, A(t')$ in the network Net, where $n \geq 1, r_1, \ldots, r_n \in R, (A(t), r_1) \in L, (r_i, r_{i+1}) \in L$ for all $1 \leq i \leq n-1, (r_n, A(t')) \in L$, and $r_i \neq r_j$ for $i \neq j$. For further consideration we fix such a path and denote it with $A(e)$.

Every path $A(e)$ has the following properties relevant for scheduling:

- it starts and ends with a computation node,
- only routers are allowed between those computation nodes,
- it is simple, i.e., a node does not appear twice on the path, and
- its length (number of nodes on the path) is between 3 and $|R| + 2$.

Obviously, a path $A(e)$ can intersect another path $A(e')$ in one or more computation nodes or routers. In order to avoid conflicts, one has to take a look at the ports being used at the shared resources.

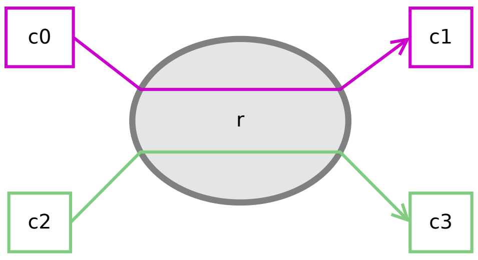

Fig. 2. Non-conflicting communication

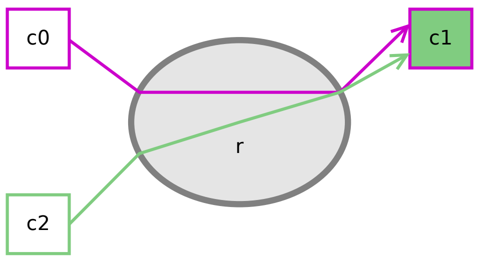

Fig. 3. Conflicting communication

In Figure 2 the communication between computation nodes c_0 and c_1 does not conflict with the communication between c_2 and c_3, because the paths use different sets of ports. Even if two data values arrive at the same time, no collision can occur. In contrast, Figure 3 shows conflicting communication, since both paths share the port that the router uses to forward data to c_1. To avoid conflicts and collisions of data values, the use of these ports has to be scheduled, too. This can either be managed by delaying values at the sender to make them arrive later at the shared resource (temporal separation), or by trying to choose a conflict free path (spatial separation). Data compression eases the scheduling of conflicting paths, as it reduces the unavailability of involved resources.

The DAKODIS architecture adds the utilization of data compression to the classical scheduling problems, like resource allocation and routing. As known from II-C a compression scheme provides real time guarantees in terms of WCCT and WCDT as well as a guarantee on the WCCR. If the compression scheme $Z = (\ell, k, ct, dt)$ is used for a channel e (with $\ell_e = \ell$) then the number of transmitted bits is reduced by $\ell - k$ for every single data value. On the other hand, the costs ct and dt have to be added to the execution times of the channel's end points. Thus, the usage of data compression is always a trade-off between less communication and longer execution times.

A simpler version of this scheduling model with only one compression algorithm and equal overheads for compression and decompression has been formulated for MILP solvers in [5]. This model also includes a stricter policy for using the routers, so that even communication as shown in Figure 2 will be treated as a potential conflict and needs temporal separation.

III. Compression

Our compression method has to cope with different types of data values: Initially, data values are obtained by physical sensors that produce data samples of real numbers (or tuples of real numbers). We map these real-valued data samples to a finite range of N values that we can identify with the numbers $0, \dots, N-1$. For this we use standard quantization techniques, see e.g. [8]. We can encode every number from the range $0, \dots, N-1$ with $\ell := \lceil \log_2 N \rceil$ bits. During the diagnostic process, the quantized values are transformed into symbolic data values.

Our specific requirements for the DAKODIS architecture imply the following features for the compression algorithm:

- Compression and decompression underlie hard real time constraints.
- Only online (one-pass) compression algorithms can be used that compress every arriving data value before the next data value arrives.
- No statistical data concerning the probability distribution of the data values is available.
- In order to utilize compression for the scheduling process, the compressor should guarantee a certain worst-case compression ratio below one.
- Compression should be lossless (after the initial quantization phase), with the exception that occasionally data values can be completely lost. This means that every data value is either transformed by the sender into a compressed representation that can be exactly recovered by the receiver, or it is transformed into a default value that tells the receiver that the original data value is lost. Loosing some data values is unavoidable if we want to guarantee a worst-case compression ratio below one. A small probability for loosing data values is tolerable in our context, since diagnosis is typically not dependent on single data values.

The above requirements rule out most of the classical compressors (see [8] for an overview on classical compression techniques):

- Lossy compressors based on transform coding (e.g. cosine transforms, wavelet transforms) are inherently lossy and do not allow to recover data values without error (which is a problem for symbolic data values that were produced during the diagnostic process). Moreover, these compressors typically exploit the limitations of human perception (e.g. for the compression of pictures, audio or video data), which is not relevant in our context.
- Lossless compressors (e.g. entropy-based coders like Huffman coding or arithmetic coding and the dictionary-based compressors from the Lempel-Ziv family) cannot guarantee a fixed compression ratio. A lossless compressor cannot properly compress every single input data value and compressing larger blocks of values would torpedo the real time constraints.

A. Cache-based code

In order to meet the above mentioned requirements for the compression algorithm, we developed a cache-based compression algorithm. In order to ensure a small rate of lost data values, we have to assume some locality in the data values: If the current data value is $i \in [0, N-1]$ then with high probability the next data value should belong to a small neighborhood of i. This is a reasonable assumption if the data values $0, 1, \ldots, N-1$ represent neighboring quantization levels of real valued sensor data.

For the compression algorithm, we assume that the number N of data values is of the form $N = 2^{s+t}$ for some $s, t \geq 1$ with $s \leq t$. Then every value $i \in [0, N-1]$ can be encoded by a bit string of length $\ell := s+t$. We call such a bit string a *code word* in the following. The first s bits (resp., last t bits) of a code word are its *head* (resp., *tail*). We assume that if the code word w encodes the data value i, then the lexicographically next code word encodes the next value $i+1$. In particular, for every fixed head $u \in \{0,1\}^s$, the set of code words $\{uv \mid v \in \{0,1\}^t\}$ corresponds to an interval of $[0, N-1]$. For example, if $s = 4$ and $t = 8$, then the head and tail of the code word 100101110101 are 1001 and 01110101, respectively. We also fix a number $r \leq s$ and construct a dictionary with at most $2^r - 1$ entries such that each entry stores a head. Every dictionary entry is addressed with a bit string $u' \in \{0,1\}^r$ with $u' \neq 0^r$ (there are $2^r - 1$ such bit strings). Initially, the dictionary is either empty or filled with some heads that are known to occur frequently in the data stream. Both, sender and receiver will store the same dictionary at every instant.

Now we compress the input data as follows. Consider a code word $w = uv$, where $|u| = s$ and $|v| = t$. If u is present in the dictionary (we can check this in constant time by implementing the dictionary using hash table) at position u' (where $u' \neq 0^r$ is a bit string of length r as described above), then we write the bit string $u'v$ of length $r+t$ on the communication channel. In this way, we save $s-r$ many bits. On the other hand, if u is not in the dictionary, then we write the bit string $0^r u$ of length $r+s \leq r+t$ on the communication channel. Moreover, the sender inserts the new head u into the dictionary. If the dictionary is not yet fully filled (i.e. contains less than $2^r - 1$ code words) then we just assign a free entry to the new head u. If the dictionary is already full, then we have to replace one of the old heads in the dictionary by u. For this we use a the least-recently-used (LRU) strategy. Note that the code word $w = uv$ is lost in this situation and we call this event a *miss*. If the receiver reads $0^r u$, then the prefix 0^r tells him that the next s many bits represent a new head u. The receiver inserts u into its dictionary using the same strategy as the sender. It is clear that the compression ratio of this algorithm is at most $(r+t)/(s+t)$.

B. Analysis of the probability of a miss

Let us now consider the probability of a miss. For this we model the sequence of code words $w_1 w_2 w_3 \cdots$ as a stochastic process. Recall that we have $N = 2^{s+t}$ different code words. Under the (not always realistic) assumption that successive code words are identically and independently distributed (iid), we have a so called iid process, which is described by a single probability distribution $(P[w] \in [0,1])_{w \in \{0,1\}^{s+t}}$ on the set of code words, where $P[w]$ is the probability that code word w appears. Whether a certain code word leads to a miss only depends on the head of the code word. From the probabilities $P[w]$ we can compute the probability p_u that a certain head $u \in \{0,1\}^s$ appears as

$$p_u = \sum_{v \in \{0,1\}^t} P[uv].$$

For a uniform distribution (i.e. $p_u = p_{u'}$ for all heads u, u') the probability of a miss is $1 - (2^r - 1)/2^s$. Flajolet et al. [1] showed that given the head probabilities p_u one can in principle compute the miss probability with the following formula, where $k = 2^r - 1$ is the size of the dictionary:

$$1 - \sum_{u \in \{0,1\}^s} p_u^2 \cdot \sum_{q=0}^{k-1} (-1)^{k-1-q} \binom{2^s - q - 2}{2^s - k - 1} \sum_{|J|=q, u \notin J} \frac{1}{1 - P_J},$$

where $P_J = \sum_{v \in J} p_v$. As also noted in [1] this formula is not suitable for practical calculations of the probability of a miss.

Franaszek and Wagner [2] consider the expected ratio $F_{\mathrm{lru}}/F_{\mathrm{opt}}$, where F_{lru} is the miss probability under the LRU strategy and F_{opt} is the miss probability of the optimal replacement strategy. The latter stores the $k-1$ heads with the highest probabilities in the dictionary. The remaining dictionary entry is used for replacement. Note that the optimal strategy assumes knowledge of the above probabilities p_u. The result from [2] (with our parameters) states that

$$F_{\mathrm{lru}}/F_{\mathrm{opt}} \leq 1 + \frac{(2^r - 1)(1 - \beta)}{1 + (2^r - 2)\beta},$$

where β is the sum of the smallest $2^s - 2^r + 1$ many head probabilities p_u. The result assumes again that the sequence of code words is produced by an iid process.

In practical situations, the next data value is highly dependent on the previous values. In particular locality, which is typically observed in physical processes, implies that the next data value is with a high probability in a small neighborhood of the previous data value. In such a setting, our cache based compressor will show a much smaller probability of a miss than in the above iid setting. This will be demonstrated by our experimental data in the next section.

IV. Examples and Evaluation

Recent developments in the automotive industry yielded a large number of driver assistance systems. Electronic stability control, assistance for parking and distance keeping, and especially autonomously driving vehicles (prototypes) require a multitude of different sensor information to be gathered, processed and evaluated. These systems demand a high reliability and with an increasing number of complex tasks taken over from the driver, safety-related aspects of the driver assistance systems become central concerns. In this context, online diagnosis helps to detect, identify and manage faults occurred in the system to prevent damage, stay operational or increase the maintainability, amongst others.

A. *Distributed architecture of modern cars*

Modern cars have many built-in electronic control units, most of which have to deal with specific tasks e.g. sensor data processing or providing control signals for other devices. However, these electronic control units may adopt diagnostic tasks as well which are related to their predestinated task. The overall electronic architecture of modern cars, can thus be seen as a distributed network where several processing units can execute diagnostic tasks leading to a distributed diagnostic process.

We introduce a Simulink model to exemplarily demonstrate the working principle of the diagnostic process on a distributed network by means of a hybrid-electric vehicle (HEV) model[2]. However, the aspects can be generalized for many other systems or processes.

The HEV-model offers an abstracted replica of a hybrid-electric car and allows to simulate the car's behavior according to a driving cycle input. The main components are the electrical part including an electrical motor, a generator, a voltage converter and a battery. The electrical motor is connected to the driveshaft. Besides, a power split device combines an internal combustion engine (ICE) with the generator and the driveshaft. Via the power split device the ICE fulfills two tasks: supporting the motor to drive the car and extending the car's operating range by charging the battery via the generator. A mode logic manages the interaction between these units (e.g. turning on or off the generator depending on the current battery state of charge and driving situation). The model is equipped with a variety of sensors, such as voltage sensors, current sensors, torque sensors or tachometers. The temperature of different components as well as the mode logic is also monitored. Since faults in the system (e.g. failure of a component) are reflected in the measured signals, a diagnostic decision can be conducted based on the analysis and processing of fault indications derived from the sensor signals.

B. *Working principles of a distributed fault-diagnosis process*

In the following, we demonstrate a fault detection and diagnostic inference process by means of a DDAG. The DDAG in Figure 4 only models the diagnostic dependencies for the faults specified in this example. It is not complete for the whole HEV-model yet suitable for demonstration purposes. Table I gives an overview of the diagnostic tasks and necessary input signals.

In the simplified scenario we assume that only one fault occurs at a time. Faults are limited to the following four events: wrong mode logic signal, failure of the generator, failure of the combustion engine or an electrical line defect. Since all of these components are involved into the battery charging process, a failure of one of the components leads to an abnormal signal behavior of the battery measurements. Applying limit, trend, and plausibility observations on the monitored state of charge (SOC) signal S_A, the first fault indication is determined at node A (Figure 4), specifically, battery not charged. At this stage a fault is detected, however, it is not identified as it may have arisen from different component failures. During the fault-inference process more sensor data is evaluated to isolate the fault. The diagnostic information obtained at node A is communicated via the message m_{AB}

[2]Hybrid-Electric Vehicle Model in Simulink, http://www.mathworks.com/matlabcentral/fileexchange/28441-hybrid-electric-vehicle-model-in-simulink

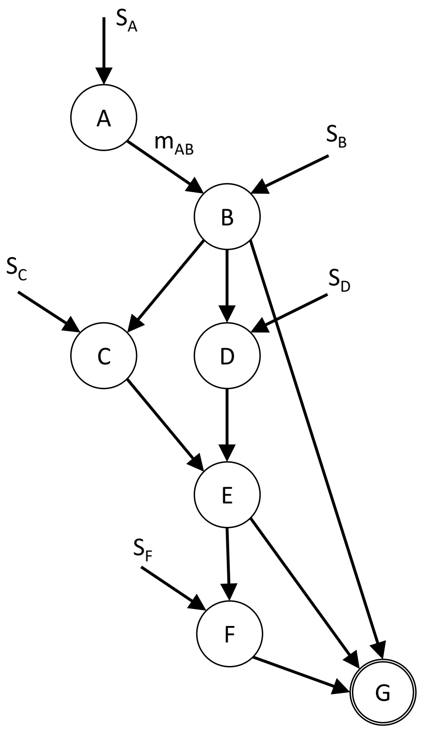

Fig. 4. Diagnostic directed acyclic graph for the example model

TABLE I
DIAGNOSTIC TASKS AND SIGNALS

Node	Task
A	Evaluation of battery state of charge
B	Verification of mode signaling
C	Evaluate generator functioning
D	Evaluate motor functioning
E	Intermediate fault decision
F	Electric circuit evaluation
G	Final fault decision
Signal	**Measurements**
S_A	Battery: state of charge
S_B	Mode logic: signaling
S_C	Generator: torque, generator speed, voltage, current
S_D	ICE: torque demand, torque, engine speed
S_F	Electrical system: voltages, currents

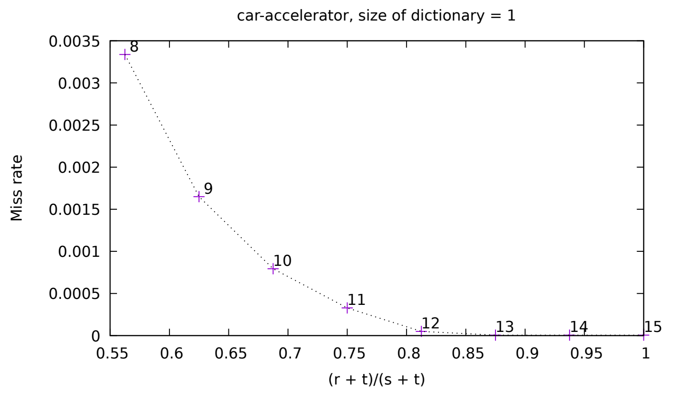

Fig. 5. Simulation with $r = 1$ (i.e. a dictionary of size $2^1 - 1 = 1$). The tail length t is written at each data point. The head length is $s = 16 - t$. Lines between data points are shown for illustration only.

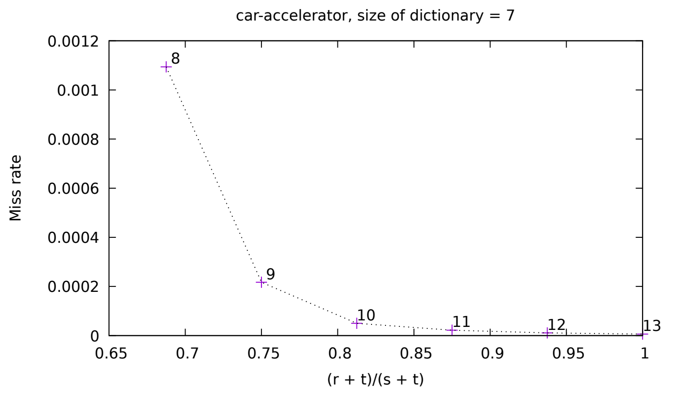

Fig. 6. Simulation with $r = 3$ (i.e. a dictionary of size $2^3 - 1 = 7$). The tail length t is written at each data point. The head length is $s = 16 - t$. Lines between data points are shown for illustration only.

to node B. Examining the signaling of the mode logic device (S_B), node B is able to reason about a fault in the mode logic. A faulty signaling immediately leads to a final fault decision bringing us to node G. In our simplified example, a deeper analysis for this type of fault is not performed, however, the DDAG could be extended to allow more accurate conclusions. A correct signaling excludes this potential fault and requires further analyses of the components generator and motor at nodes C and D, respectively. Evaluating torque, rotational speed and electric measurements of these components (S_C and S_D) and taking energy conservation into account, their status is determined and the diagnostic features are combined at node E, allowing a diagnostic intermediate decision based on plausibility relations. If a correct working of the generator and the ICE is stated, we proceed to node F. With S_F being voltage and current measurements in different locations of the electric circuit, the fault can be identified.

A stepwise fault-inference process matches real world systems. The successive generation, evaluation, and combination of the information for the fault-diagnosis process are often required. Especially when a system to be diagnosed consists of multiple independent processing units, not all relevant data may be available for the diagnosis at all times. Furthermore, the diagnostic feature extraction consumes computational resources. Likewise, the broadcasting of the information through the network impacts the overall transmission performance. In these cases, one may concentrate on monitoring fewer important signals continuously and ask for confirmative data after the fault detection step. The ability to adopt the diagnostic methods to a variety of applications may help to increase their reliability, availability and especially the safety.

C. Example for a cache-based code

To examine the performance of the cache-based compressor described in III-A, we implemented a simulation[3] that works on data sets from the HEV-Model. Figures 5 and 6 show the relationship between the miss rate and the compression ratio $(r+t)/(s+t)$ with different settings for the parameters r and t. For the experiments we used the samples of the car-accelerator signal from a WLTP[4]-Class 3 driving cycle simulation. The data set contains 180100 samples that were quantized with 16 bits and a sampling frequency of 100 Hz. We see that with a compression ratio of about 62.5% we achieve a miss rate of less than 0.2% (Figure 5 for $t = 9$). With a compression ratio of about 75% we even reach a miss rate of about 0.02% (Figure 6 for $t = 9$).

D. Minimizing latency

We scheduled the DDAG from Figure 4 to a network and analyzed the communication times between the computation nodes. Tasks observing similar objects are allocated on the same node. For simplicity the network is designed in such a way that each channel can use two completely disjoint routes. Because of the network design and the allocation there is no need to consider collisions.

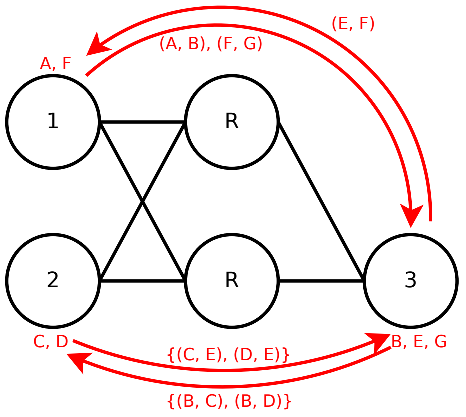

Fig. 7. Network with allocated tasks and communication

[3]Implementation of cache-based code, https://networked-embedded.de/es/index.php/dakodis.html

[4]Worldwide Harmonized Light-Duty Vehicles Test Procedure, https://www.unece.org/fileadmin/DAM/trans/doc/2014/wp29/ECE-TRANS-WP29-2014-027e.pdf

Figure 7 shows the network and the 7 tasks $A, B, \ldots, G$ allocated to the three computation nodes as well as the communication links. Note, that the stated allocation is not optimal as the concurrency of C and D is not exploited. The communication pattern of the DDAG is shown by red directed egdes between the involved computation nodes. These edges are labeled with channels (see Section II-A). A group of channels surrounded by curly braces means that these channels are mapped to disjoint communication paths (via the two routers). Hence, for the computation of the makespan, such a group can be considered as one channel. Note, that the two channels (B, G) and (E, G) are not present in Figure 7 because the three tasks B, E, G are mapped to the same computation node. In this example we assume a worst-case execution time (WCET) of 4 ticks per task and a per-hop transmission time of 8 ticks per message. Hence, we can calculate the makespan as $7 \cdot 4 + 5 \cdot (2 \cdot 8) = 108$ ticks (7 tasks, each requiring 4 ticks, and 5 (groups of) channels, each mapped to a communication path of length 2). Note, that the two topological orderings (which result from the two different orderings of C and D) have no influence on the makespan. Assuming the compression parameters $\mathrm{WCCR} = 0.75$ and $\mathrm{WCCT} = \mathrm{WCDT} = 1$, we are able to decrease the makespan. Task A must handle only one compression for channel (A, B) and G must handle only one decompression for channel (F, G). Therefore, both end up with an accumulated WCET of 5 ticks. Tasks C, D and F must each perform one compression *and* one decompression. They all end up with a WCET of 6 ticks. The remaining tasks B and E must handle two compressions and one decompression (B) or one compression and two decompressions (E). They both end up with a WCET of 7 ticks. The per-hop transmission time of the messages is reduced to $6 = 8 \cdot 0.75$ ticks. The makespan of the DDAG with compression is $2 \cdot 5 + 2 \cdot 7 + 3 \cdot 6 + 5 \cdot (2 \cdot 6) = 102$ ticks.

V. Future Work

In this section we provide an overview of research activities planned to improve the proposed architecture.

A. Diagnostic framework

The diagnostic framework contains different atomic components. In order to extract features from different streams, these atomics can be combined to more powerful blocks. Complex processes may require different procedures. For this, a library of diagnostic feature extraction blocks along with interfaces and instruction sets will be established to allow an application specific and user-friendly integration of the diagnostic methods for many fields. In the future, we also aspire an extension of the DDAG design-rules with a special focus on handling huge and complex DDAGs. A tool for timing analyses of the diagnostic feature extractions and all other processing blocks is also necessary, in order to provide worst case execution times and thus, time guarantees for the fault diagnosis.

B. Compression library

In order to complete the compression library, we will investigate in which way different compression methods fit the needs of the diagnostic framework while still providing real time guarantees. Additionally, we will try to make use of correlation between streams to achieve better compression.

C. Scheduler

The scheduling model introduced in section II-D has to be implemented. Our goal is to calculate a static schedule considering different compression methods using genetic algorithms. Although these types of algorithms will not necessarily solve the problem optimally, we hope to obtain near-optimal solutions in a feasible amount of computing time. First experiments have shown promising results regarding these topics. Furthermore, the model needs to be generalized, so that a computation node can handle multiple jobs.

VI. Conclusion

In this paper, we firstly motivated the usage of data compression in distributed online-diagnosis systems. We then presented an architecture for such a system by defining models for diagnostic graphs (DDAG), networks of computation nodes and routers, as well as a model for compression schemes that allows to provide real time guarantees. After discussing the scheduling model that now includes compression, a lossy cache-based compressor with a guaranteed compression ratio and low costs was proposed. With an example we showed that depending on the parameters more than one third of the bits ($\sim 37.5\%$) can be saved, while still not loosing more than 0.2% of the values. In turn, this means that 99.8% of the values are delivered correctly and without any losses, but with a significant reduction of bandwidth demands. The small loss-rate, the low costs and the fixed compression ratio make this compressor suitable for real time purposes.

Acknowledgment

This work was supported by the DFG research grants LO748/11-1 and OB384/5-1.

References

[1] Philippe Flajolet, Danièle Gardy, and Loÿs Thimonier. Birthday paradox, coupon collectors, caching algorithms and self-organizing search. *Discrete Applied Mathematics*, 39(3):207–229, 1992.

[2] Peter A. Franaszek and T. J. Wagner. Some distribution-free aspects of paging algorithm performance. *Journal of the ACM*, 21(1):31–39, 1974.

[3] Nan Guan. *Techniques for building timing-predictable embedded systems*. Springer, 2016.

[4] Rolf Isermann. *Fault-diagnosis systems: an introduction from fault detection to fault tolerance*. Springer Science & Business Media, 2006.

[5] Damian Ludwig and Roman Obermaisser. Scheduling of datacompression on distributed systems with time-and event-triggered messages. In *International Conference on Architecture of Computing Systems*, pages 193–204. Springer, 2017.

[6] Roman Obermaisser, Rubaiyat Islam Sadat, and Fabian Weber. Active diagnosis in distributed embedded systems based on the time-triggered execution of semantic web queries. In *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2014 IEEE 17th International Symposium on*, pages 222–229. IEEE, 2014.

[7] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[8] Khalid Sayood. *Introduction to Data Compression, fourth edition*. Morgan Kaufmann, 2012.