Combined Compression of Multiple Correlated Data Streams for Online-Diagnosis Systems

Seungbum Jo*, Markus Lohrey*, Simon Meckel†, Roman Obermaisser†, Simon Plasger*
*Theoretical Computer Science, University of Siegen, Germany
†Embedded Systems Group, University of Siegen, Germany

***Abstract*—Online fault-diagnosis is applied to various systems to enable an automatic monitoring and, if applicable, the recovery from faults to prevent the system from failing. For a sound decision on occurred faults, typically a large amount of sensor measurements and state variables has to be gathered, analyzed and evaluated in real-time. Due to the complexity and the nature of distributed systems all this data needs to be communicated among the network, which is an expensive affair in terms of communication resources and time. In this paper we present compression strategies that utilize the fact that many of these data streams are highly correlated and can be compressed simultaneously. Experimental results show that this can lead to better compression ratios compared to an individual compression of the data streams. Moreover, the algorithms support real-time constraints for time-triggered architectures and enable the data to be transmitted by means of shorter messages, leading to a reduced communication time and improved scheduling results.**

***Keywords*—Online-Diagnosis; Real-Time; Data Compression; Scheduling**

I. Introduction

It is the goal of online fault-diagnosis to detect, diagnose and, especially in safety-critical application domains, overcome system faults at run time, e.g., through reconfigurations, in order to provide the required services with a very high reliability. Typical faults are faults in the design of a component or a system, transient or permanent hardware faults or erroneous user operations amongst others. As faults cannot be prevented at all times it is of uttermost importance to equip the system with powerful diagnostic algorithms able to process, analyze and store the often huge amount of raw data of a variety of sensors as well as system input and output parameters necessary for correct and fast decisions on fault detection and identification [3], [5].

The quality of diagnostic decisions highly depends on the available amount of data. Especially in distributed systems where the diagnostic inference process is temporally and spatially decomposed to multiple processing units, a limited storage capacity of local real-time databases may narrow the performance of diagnostic decisions, e.g., of long term trend analyses. Likewise, limited communication resources or bottlenecks may require data to be discarded or may lead to communication delays, which are also disadvantageous with respect to the time needed to conclude a specific fault from a first symptom. This, however, is a quality characteristic of a diagnostic architecture.

Since a distributed diagnostic process requires both, the storage and the fast exchange of a huge amount of data within a network, the DAKODIS[1] project deploys data compression for online fault-diagnosis in a time-triggered architecture. In [4] it is shown that data compression is a feasible instrument to save bandwidth and consequently provide stronger real-time guarantees or save communication resources (see [9] for an introduction into the wide area of data compression). Especially systems with limited resources may require data compression to actually make fault-diagnosis possible or keep a diagnosis system alive if more and more traffic is scheduled to the existing communication channels.

The majority of compression algorithms such as entropy-based compressors or dictionary-based compressors (see e.g., [9]) are not suitable for time-triggered systems, as they do not guarantee a fixed compression ratio; see also Section II. This was overcome in [4], where a simple cache-based compression algorithm was presented that allows the loss of a small ratio of the input data values. It works especially well if the input data stream shows a temporal locality in the sense that consecutive data values are close with high probability. Physical sensor signals typically show this behavior.

In [4], only the compression of a single data stream is addressed. On the other hand, diagnostic data streams within distributed networks are often highly correlated. For instance, multiple redundant sensors measure the same physical quantities, or measurements of interdependent components or subsystems show other kinds of characteristic dependencies. These correlations yield the potential for further compression. In time-triggered architectures all routes for the messages are predetermined. In [7] it is demonstrated that if messages of multiple data streams that take the same routes can be compressed to one combined sample, scheduling results can be improved. However, no real-time compression algorithms for such a scenario have been introduced so far. In this paper we close this gap and extend the work from [4] to the simultaneous compression of multiple data streams. We evaluate the resulting compressor with voltage and current measurement signals from a DCDC converter and demonstrate that the simultaneous compression of multiple correlated data streams leads to a better compression ratio without increasing the overall number of lost data values. The proposed compressors support real-time constraints for time-triggered

[1]DAKODIS – Data compression for online-diagnosis systems. The primary objective of this research project is an increased efficiency and the reduction of overhead for online-diagnosis in open embedded systems using data compression. (see https://networked-embedded.de/es/index.php/dakodis.html)

architectures by maintaining a constant compression ratio. For the identification of potential candidates for a combined compression, a pre-analysis of the available data streams of the system is conducted. The results of the compression algorithm provide a scheduler with meaningful information to plan the most beneficial task allocations and message combinations.

II. Model

An architecture for time-triggered real-time systems is defined in [4]. The three core aspects of the model are (1) logical modeling of diagnostic processing tasks, (2) intertask communication procedure modeling supporting data compression and (3) task scheduling to map the processing tasks to computation nodes of a physical network infrastructure, and thus, establishing a time-triggered architecture (see [2] for an introduction to timing-predictable embedded systems). Within this environment, specific requirements for the compression algorithm arise:

- 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 information 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 an initial quantization phase for sensor data), 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. Losing 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.

These requirement rule out most of the classical compressors, e.g., lossy compressors based on transform coding, as they are inherently lossy and do not allow to recover data values without error and also lossless compressors (e.g., entropy-based coders, or arithmetic coding) as they cannot guarantee a fixed compression ratio. For a detailed overview on classical compression techniques see [9].

For our scenario we assume the data communication to be error free, i.e., a data value is received exactly as it was sent. Under challenging conditions, forward error correction is a feasible instrument to ensure this.

III. Compression

The compression schemes proposed in this paper are based on the work of [4], where the overall system architecture as well as the compression model is addressed in detail. Therefore, Section III-A briefly summarizes the working principles of the cache-based online data compression algorithm before the subsequent sections deal with the extension to the compression of multiple correlated data streams.

A. Compression of Individual Data Streams

The goal is to compress a sequence of n-bit data values (for a fixed n), which are produced by a sensor measuring a physical quantity. It is assumed that the data values exhibit some locality. For the compression every n-bit data value is split into a block of s high-order bits (called the *head*) and the remaining $t = n - s$ low-order bits (called the *tail*). Due to the locality of consecutive data values, the heads of consecutive data values are expected to only show a small variation over time. The $n = s + t$ bits in a data value are compressed to $r + t$ bits (for some $r < s$). For this, the algorithm transmits the t bits from the tail uncompressed and compresses the s bits from the head to r bits. To achieve the latter, a dictionary D stores the $2^r - 1$ most recently seen heads at dictionary entries $D[p]$, where the dictionary index p is an r-bit code different from the reserved sequence 0^r (r 0-bits). Let the next n-bit data value be $x = uv$, where $u \in \{0,1\}^s$ is the head and $v \in \{0,1\}^t$ is the tail. If the head u is in the dictionary and stored at entry $D[p]$ then the $(r + t)$-bit code pv is transmitted. Otherwise, a so called *miss* occurs which is indicated to the receiver by the bit sequence 0^r. The sender then transmits the bit sequence $0^r u$ (recall that $s \leq t$ so $0^r u$ fits into $r+t$ bits). Moreover, sender and receiver update their dictionaries by computing a dictionary index $p = \text{fresh}(D)$ and setting $D[p] := u$. Here $\text{fresh}(D)$ is the index of a free dictionary entry, or, if the dictionary is completely filled, an index that is computed by some replacement strategy (we use the least-recently-used strategy, LRU for short). The remaining $t - s$ bits following $0^r u$ can be used to transmit the $t - s$ most significant bits (MSBs) of the tail v to achieve a higher accuracy. Due to locality in the data values we expect a small number of misses over time. Note that sender and receiver can keep their dictionaries synchronized. The main features of the compression algorithm are:

- A fixed compression ratio of $(r + t)/(s + t) < 1$ is achieved for every data value. This is contrary to classical lossless compression, where only statements about the average compression ratio are possible. However, a fixed compression ratio is important for time-triggered architectures.
- To make a fixed compression ratio < 1 possible, we have to accept occasional losses of data values. A small number of lost data values is acceptable for many online-diagnosis applications. Moreover, even in the case of a miss, an approximation in form of the t MSBs of the data value is transmitted.
- Those data values that are not lost are transmitted without any loss in accuracy, which is in contrast to classical lossy compression.

Related work from data compression. The idea of using locality in the data values for compression can be found in

many works. One of the simplest ways of exploiting locality is delta-coding, where differences between consecutive data values are transmitted. These differences are typically small and can be further compressed with an entropy encoder. In the context of wireless sensor networks this idea is implemented in the LEC-compressor from [8], [10]. In contrast to our method, the LEC-compressor is a lossless entropy-based encoder, which does not show a fixed compression rate. In lossy compression, differential encoding [9, Chapter 11] exploits correlation between successive data values by transmitting the differences between a prediction of the next data value and the actual data value. In the area of information theory the problem of compressing correlated data streams is known as distributed source coding, see [1].

B. Simultaneous Compression of Multiple Data Streams

The cache-based compression algorithm introduced in Section III-A handles each data stream individually. For the compression it utilizes the fact that measurements of physical quantities can be often covered by just a part of the overall code word space for certain time intervals. With a view to the overall system architecture, many data streams for monitoring and diagnostic purposes (e.g., voltage, current or vibration measurements) of complex mechatronic systems are gathered and processed at different locations and need to be exchanged via a distributed network. Often, many of these data streams are highly correlated due to redundant measurements or physical relations of the measured signals. The enhanced compression scheme presented in the following takes advantage of both facts, the neighborhood assumption and the signal correlations. Since the overall data traffic needs to be scheduled in time-triggered architectures, the transmission routes are predetermined. Especially when the source and the destination nodes of two or more data streams are located close to each other or are the same, the overall message size for transmitting the data can be reduced, leading to advantageous scheduling results (e.g., a shorter makespan).

To exploit correlations of data streams we adapt the cache-based encoding scheme to encode multiple data streams at once. Assume that we have d data streams. Let x_i $(1 \leq i \leq d)$ be the current data value of the i-th stream. We further assume that x_i is an n_i-bit data value. For every $i \in [1, d]$ we fix a partition $n_i = s_i + t_i$ and split x_i into $x_i = u_i v_i$ with $|u_i| = s_i$ and $|v_i| = t_i$. The bit sequence u_i (resp., v_i) is the current head (resp., tail) of the i-th stream. We do not assume $s_i = s_j$ or $t_i = t_j$ for $i \neq j$. Let $s = \sum_{i=1}^{d} s_i$ and $t = \sum_{i=1}^{d} t_i$ for the rest of the section. One could apply the compression scheme from Section III-A to each of the d data streams separately by choosing numbers $r_i < s_i$ and maintaining a dictionary of size $2^{r_i} - 1$ for every $1 \leq i \leq d$. This leads to an overall compression ratio of $(\sum_{i=1}^{d} r_i + t)/(s+t)$. On the other hand, due to correlations between the data streams, the tuples of data values $(x_1, \ldots, x_d)$ will be scattered around a low dimensional subspace of the d-dimensional product space. If, for instance, $d = 2$ and $x_2 = f(x_1)$ for a function f then all tuples belong to a one-dimensional curve in the two-dimensional plane. In such a case we can obtain a better compression ratio of $(r + t)/(s+t)$ by using a single dictionary of size $2^r - 1$ for some $r < \sum_{i=1}^{d} r_i$ that stores tuples of heads $u = (u_1, \ldots, u_d)$, which need s bits. The compressed data value then consists of the dictionary index $p \in \{0,1\}^r$, where u is stored and the concatenation $v_1 v_2 \cdots v_d$ of the current tails. In case of a miss we transmit 0^r followed by u. This leaves $t - s$ unused bits. Likewise to the original approach we use them to transmit parts of the tails. In the multidimensional case there are several possibilities:

- Transmit as many tails as possible completely and fill the remaining bits with the MSBs of the next tail,
- transmit equally many MSBs for each tail,
- transmit MSBs for each tail; the number of MSBs is weighted by the complete tail length.

Note that in this approach all d data values from $(x_1, \ldots, x_d)$ are lost in case of a miss. In consequence, this method is a trade-off between a better (i.e., lower) compression ratio and a higher number of lost data values. Its implementation with a variable number d of streams can be found on our project website.

Algorithm 1 Cache-based algorithm for d streams

```
input : data values x_i ∈ {0,1}^{n_i} (1 ≤ i ≤ d)
output : bit string of length at most r + t
initialize dictionary D as empty hash table of size 2^r − 1
let x_i = u_i v_i with |u_i| = s_i and |v_i| = t_i for 1 ≤ i ≤ d
if there is p with D[p] = (u_1, ..., u_d) then
    send p v_1 v_2 ··· v_d to the receiver
else
    send 0^r u_1 u_2 ··· u_d to the receiver
    p := fresh(D)
    D[p] := (u_1, ..., u_d)
end if
```

C. Dynamic Simultaneous Compression of Multiple Data Streams

It turns out that one can reduce the number of misses by adding some flexibility using offsets in the dictionary entries. Consider a sequence of values $S = a_1, a_2, \ldots, a_k$ on a single stream such that for all $1 \leq i < k$, $|a_i - a_{i+1}| < 2^{t-1}$ but $\lfloor a_i / 2^t \rfloor \neq \lfloor a_{i+1} / 2^t \rfloor$ (s and t are the size of the head and tail, respectively). If one compresses S using the cache-based compression algorithm from Section III-A with a dictionary of size 1 (i.e., $r = 2$), then this results in k misses, since there are no consecutive values in S with the same head. To handle this case, Jo et al. proposed a *dynamic cache-based algorithm* [4]. In this algorithm, every dictionary entry $D[p]$ is a pair (u, δ) of a head $u \in \{0,1\}^s$ and an *offset* $\delta \in [-2^{t-1}, 2^{t-1} - 1]$. We define a corresponding interval $I[p] = [u2^t + \delta, u2^t + \delta + 2^t - 1]$. Here and in the following we identify heads (resp., tails) with numbers from the interval $[0, 2^s - 1]$ (resp., $[0, 2^t - 1]$) using their binary representation. We say that p *covers* the data values in the interval $I[p]$. The cache-based compression algorithm from Section III-A can be considered as the special case where δ is always 0. Let us write $(u[p], \delta[p])$ for $D[p]$.

When the sender transmits a code word $x = uv$ where u (resp. v) is the head (resp., tail) of x, the sender first checks whether there is a dictionary index p that covers x. Since $\delta[p] \in [-2^{t-1}, 2^{t-1}-1]$, we must have $u[p] \in \{u-1, u, u+1\}$. Now suppose that the dictionary contains an index p which covers x (otherwise, the sender transmits $0^r u$ and adds the pair $(u, 0)$ to the dictionary). The sender takes the smallest such p and transmits pv to the receiver. Then it updates the dictionary in such a way that x becomes the center of an interval $I[q]$ for some dictionary index q. For this, it first ensures that $u[q] = u$ holds for a unique index q. Then, it sets the offset $\delta[q]$ to $v - 2^{t-1}$. In this way, x becomes the center of the interval $I[q]$. The receiver reconstructs x from p and v and updates its dictionary analogously. One easily observes that in the above example only a_1 is lost with this dynamic cache-based compression algorithm while maintaining a dictionary of size 1.

To compress highly-correlated multiple streams efficiently when each stream has consecutive data values with small gaps, we make Algorithm 1 dynamic: Each dictionary entry $D[p]$ stores a d-tuple $(u_1, \ldots, u_d)$ of heads ($u_i \in \{0,1\}^{s_i}$) and an offset vector $(\delta_1, \ldots, \delta_d)$ with $\delta_i \in [-2^{t_i-1}, 2^{t_i-1} - 1]$. Let us define $u[p, i] = u_i$, $\delta[p, i] = \delta_i$ and the interval

$$I[p, i] = [u_i 2^{t_i} + \delta_i, u_i 2^{t_i} + \delta_i + 2^{t_i} - 1].$$

We say that p covers the data values in the d-dimensional hypercube

$$H[p] := \prod_{i=1}^{d} I[p, i].$$

We call this hypercube an *active hypercube* and the union of all active hypercubes is called the set of *active data tuples*. Now consider the case that the sender transmits a tuple $x = (x_1, \ldots, x_d)$ of data values from d data streams to the receiver. Let $x_i = u_i v_i$ as in Section III-B. Then the sender first checks whether there is a dictionary index p that covers x (otherwise, the sender transmits $0^r u_1 \cdots u_d$, and adds $(u_1, \ldots, u_d)$ with the offset vector $(0, \ldots, 0)$ to the dictionary). For every $1 \leq i \leq d$ we must have $u[p, i] \in \{u_i - 1, u_i, u_i + 1\}$. The sender again takes the first such p and transmits $pv_1 v_2 \cdots v_d$ to the receiver. Then it makes the same updates that were described above for the case $d = 1$ in every dimension, see Algorithm 2.

In the following two subsections we explain the improvements of Algorithm 2 that further reduces the number of lost data values considerably.

D. Partial Misses

When compressing d data streams simultaneously, a miss occurs if there is no p covering x. Consider an example with $d = 2$ streams where $s_1 = s_2 = 4$, $t_1 = t_2 = 8$, and $r = 2$, i.e., the dictionary D is of size $2^r - 1 = 3$. Table I shows an example for a dictionary; all offsets are assumed to be zero. The dictionary index 00 is reserved for the indication of the miss case. If the heads of the current data values x_1 and x_2 are $u_1 = 1001$ and $u_2 = 1101$, respectively, there is a miss as the required concatenation of heads is not in the dictionary. Hence, the tail bits are used to send the new head combination $u_1 u_2$ to the receiver. Figure 1 illustrates the active hypercubes (here squares) as blue squares with the missed data value marked as a red cross.

Algorithm 2 Dynamic cache-based algorithm

```
input : data values x_i ∈ {0,1}^{n_i} (1 ≤ i ≤ d)
output : bit string of length at most r + t
miss : variable for indicating the event of a miss
initialize dictionary D as empty hash table of size 2^r − 1
let x_i = u_i v_i with |u_i| = s_i and |v_i| = t_i for 1 ≤ i ≤ d.
if there is p ∈ {0,1}^r \ {0^r} covering (x_1, ..., x_d) then
    let p be the smallest index covering (x_1, ..., x_d)
    send p v_1 ··· v_d to receiver
    miss := 0
else
    send 0^r u_1 u_2 ··· u_d to receiver
    miss := 1
end if
if there is no q with u[q] = (u_1, ..., u_d) then
    q := fresh(D)
    u[q] := (u_1, ..., u_d)
    δ[q] := (0, ..., 0)
end if
if miss = 0 then
    let q be the unique index with u[q] = (u_1, ..., u_d)
    δ[q] := (v_1 − 2^{t_1−1}, ..., v_d − 2^{t_d−1})
end if
```

In the above situation, both data values x_1 and x_2 are lost. The strategy presented in the following overcomes this disadvantage of the algorithm in many cases and allows to communicate the current data values corresponding to some of the streams successfully. This is realized by reserving an additional dictionary index (the bit sequence $0^{r-1}1$) to indicate a so called *partial miss* (in contrast to a *full miss* which is indicated by 0^r). Hence, the dictionary size is reduced to $2^r - 2$. The number of bits transmitted per time step is $r + t$ (the length of the dictionary index plus the length of all tails) as before. In the case of a partial miss, the bit sequence following $0^{r-1}1$ is interpreted in a way different to our previous algorithm. Given the example of Table I, $D[01]$ is removed and $p = 01$ is used to indicate a partial miss.

If there is no dictionary index p that covers the whole tuple of data values $(x_1, \ldots, x_d)$, the algorithm seeks for a p that covers at least some of the data values. For this, we define

TABLE I
CODEBOOK FOR TWO-DIMENSIONAL STREAM COMPRESSION

dictionary index p	$u[p, 1]$	$u[p, 2]$
01	0110	1010
10	0111	1011
11	1001	1100

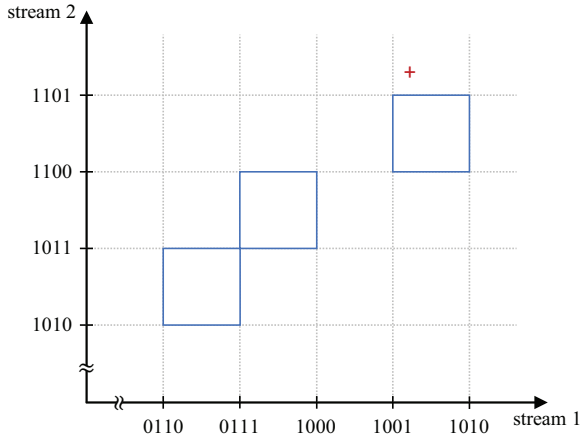

Fig. 1. Simultaneous compression of two data streams

the set $M[p] = \{i \in [1,d] \mid x_i \notin I[p,i]\}$ of those dimensions where p does not cover the corresponding data value. In our example ($u_1 = 1001$ and $u_2 = 1101$) we have $M[10] = \{1,2\}$ and $M[11] = \{2\}$.

Our algorithm tries to encode as many entries from the input tuple $(x_1, \ldots, x_d)$ as possible by looping over all dictionary indices $p \in \{0,1\}^r \setminus \{0^r, 0^{r-1}1\}$. Consider a specific index p. Assume that we tell the receiver p and the set $M[p]$. This leaves $t - r - d$ bits from the initial $r + t$ bits (we need r bits for the partial miss indicator $0^{r-1}1$, r bits for p and d bits for $M[p]$). We can use these $t - r - d$ bits in order to send the following data to the receiver:

1) All tails v_i for $i \in [1,d] \setminus M[p]$. Note that for $i \in [1,d] \setminus M[p]$, the receiver can obtain the head u_i of x_i (and consequently the data value x_i) from the dictionary entry $D[p]$ as in Section III-C. Note that $\sum_{i \in [1,d] \setminus M[p]} t_i$ bits are needed for these tails.
2) All differences $u_i - u[p,i]$ for every dimension $i \in M[p]$. Note that for each difference we need one bit to encode the sign of the difference. Also note that the differences $u_i - u[p,i]$ ($i \in M[p]$) together with the index p allow to reconstruct the missed heads u_i ($i \in M[p]$) at the receiver side, which is necessary in order to update the dictionary.

In our example, if $p = 11$ then $M[11] = \{2\}$ and the only needed difference is $u_2 - u[11,2] = 1$. For the differences in point 2 we have

$$h = \sum\nolimits_{i \in M[p]} t_i - r - d$$

bits available. If all the differences from point 2 fit into these h bits, then p is a valid choice for the algorithm and all data values x_i with $i \in [1,d] \setminus M[p]$ will be correctly transmitted. If it turns out that there is more than one valid choice for p, then one could choose that p that allows to transmit the maximal number of data values. There might also be scenarios where some of the data streams are more important; then one would give priority to these streams. If no valid index p is found then a full miss will be indicated via the bit sequence 0^r.

Especially for large dictionary sizes, where r is large, h becomes small. However, for a large dictionary (e.g., $r > 6$) one has the option to use more than one dictionary index to indicate a partial miss without increasing the number misses significantly. If we decide to reserve 2^k dictionary indices (for some $k < r$) to indicate partial misses (which results in the dictionary size $2^r - 1 - 2^k$), k bits can be additionally assigned to h as each of the 2^k reserved indices now refers to a certain part of the dictionary which can be encoded with less bits.

E. Grouping of Active Hypercubes

For the cache-based compression algorithm from Section III-B (where all offsets are zero) each active hypercube is maintained individually in terms of its corresponding head tuple in the dictionary, i.e., only in a miss case, the sender and the receiver synchronously update the dictionary by replacing the least-recently-used head tuple by the missed head tuple. The dynamic version of the algorithm from Section III-C uses an offset parameter to center an active hypercube around the transmitted sample to provide a better coverage of the sample's neighborhood.

The results of [4] show, that at the same compression ratio smaller dictionaries with larger hypercubes outperform larger dictionaries with smaller hypercubes, although the latter constellation manages more active values than the former, e.g., in the paper compare the constellation $d = 1$, $r = 3$, and $t = 8$ ($(2^3 - 1) \cdot 2^8 = 1792$ active values) against $r = 1$ and $t = 10$ (1024 active values). In many cases a newly inserted small active hypercube is not able to cover the next data value, especially if t is small compared to n (e.g., $t = n/2$). We overcome this disadvantage with new strategies for combining multiple adjacent active hypercubes to larger active regions that cover more values around the last transmitted sample. This can be applied to the individual stream compression and to the simultaneous compression of multiple data streams. In the latter case, various possibilities exist how to form a so called *region* of active hypercubes. Such a region is characterized by a subset of active hypercubes $H[p]$, one of which is defined as the *center hypercube*. An active region has to be connected in the following sense: Take the graph, whose vertices are the (d-dimensional) hypercubes from the region, and where two hypercubes are connected by an edge iff they intersect in a d'-dimensional sub-hypercube for some $0 \leq d' < d$. Then this graph has to be connected. We specify a group of active hypercubes by a set $P \subseteq \{0,1\}^r \setminus \{0^r\}$ of (usually consecutive) dictionary indices, all having the same offset vector (which we call the offset of the region): $\delta[p] = \delta[p']$ for all $p, p \in P$ with $p \neq p'$. This means that if one hypercube is moved (by changing the corresponding offset vector) then all other hypercubes from the region are moved in the same way. Moreover, there is a distinguished $p_c \in P$ such that $H[p_c]$ is the *center hypercube* of the region, and we call $u[p_c]$ the *center head* of the region. The active region corresponding to P is

$$R[P] = \bigcup_{p \in P} H[p].$$

We impose the same restriction on the offset vectors as in Section III-C, i.e., offset vectors have to belong to $\prod_{i=1}^{d}[-2^{t_i-1}, 2^{t_i-1}-1]$). After every successful transmission (successful in the sense that no miss occurs) we update the heads and the common offset for the region $R[P]$ that covers the current tuple of data values $x = (x_1, \ldots, x_d)$ in such a way that x becomes the center of the center hypercube. Let $x_i = u_i v_i$ where u_i is the head of x_i and v_i is the tail of x_i. Assume that x belongs to the region $R[P]$, i.e., no miss occurs. We then update every head $u[p]$ and offset $\lambda[p]$ for $p \in P$ by $u[p] := u[p] + (u_1, \ldots, u_d) - u[p_c]$ and $\lambda[p] := (v_1 - 2^{t_1-1}, \ldots, v_d - 2^{t_d-1})$. Some care has to be taken in case a head $u[p]$ is out the allowed range (e.g., contains a negative entry).

One may use only one active region (i.e., all dictionary indices contribute to the region) or several regions that can be moved independently from each other. If the data values are clustered around several areas than one ideally uses one active region per cluster. In a miss case, the least-recently-used region is moved to cover the current tuple of data values.

In the one-dimensional case a region is formed from a certain number (say k) of adjacent intervals which yields a single interval $I = [u2^t + \delta, u'2^t + \delta - 1]$ where $u, u' \in \{0,1\}^s$, $k = u' - u$ and δ is the common offset for the region. The size of this interval is $k2^t \geq 2^t$. This large interval provides a better coverage of the current data value and (under the locality assumption) leads to a higher probability that the next date value is also covered by I.

The grouping technique shows its full potential for the compression of several correlated data streams. Multiple correlated signals often show a characteristic behavior in the d-dimensional product space in the sense that they are concentrated around a low-dimensional subspace of the overall product space. This fact is utilized in multi-stream compression with grouping and enables us to significantly reduce the number of lost data values. For instance, data values of two physically related signals might rise and fall in a similar way. They will be predominantly covered by regions that cover imaginary slopes in the two-dimensional product space. In the current stage, our algorithm supports predefined fixed d-dimensional regions with a different number of hypercubes for different dictionary sizes (see Section V). We defined symmetric regions based on one center hypercube, e.g., a square or rectangle in the two-dimensional case, or a cuboid in the three-dimensional case.

Table II shows an example for a dictionary that defines a single active two-dimensional region consisting of 7 squares. This region corresponds to the 7 squares in Figure 2 with the black perimeter. Darker plotted squares indicate a higher hit rate with respect to the center of the region. We see that the marked region is a good choice to cover the majority of all values with a rather small dictionary.

Comparing the covered region of values with single-stream compression, one would need to concatenate 3 intervals in each stream to cover a square that encloses the defined region for the two-dimensional stream space. For this, the single-stream compression requires more bits (e.g., $r_1 = r_2 = 2$, $r = r_1 + r_2 = 4$ compared to $r = 3$ for the combined compression). This effect scales for larger regions, making multi-stream compression outperforming single-stream compression due to its ability to save bits by limiting the value coverage to the special region of interest.

TABLE II
HEAD COMBINATIONS FOR A TWO-DIMENSIONAL REGION OF 7 SQUARES

dictionary index p	$u[p,1]$	$u[p,2]$
001	$u_1 - 1$	$u_2 - 1$
010	$u_1 - 1$	u_2
011	u_1	$u_2 - 1$
100	u_1	u_2
101	u_1	$u_2 + 1$
110	$u_1 + 1$	u_2
111	$u_1 + 1$	$u_2 + 1$

Let us remark that the grouping technique from this section can be combined with the partial miss technique from Section III-D. In fact, this combination leads to the smallest number of lost data values in our experimental evaluation that will be discussed in the next section.

IV. EXAMPLES AND EVALUATION

The recent developments in the fields of automotive and industrial applications demand highly reliable systems, which are able to monitor and diagnose themselves from a large number of sensor data or state variables. In order to guarantee the real-time requirements of these systems, time-triggered architectures are utilized, where the complete communication of data streams is predetermined and scheduled. Fixed compression ratios are needed for this.

Up-to-date fault-diagnosis techniques are based on machine learning, neural networks and artificial intelligence and require the constant processing of large amounts of input data [6]. The complexity of these systems arises, amongst others, due to the spatial decomposition, where sensor data are gathered, merged, and processed at different locations, e.g., processing units or cloud services. It is our goal to show that a combined compression of multiple data streams is able to reduce the size of messages compared to an individual stream compression. This enables a scheduling algorithm to optimize the task allocation and consequently the makespan.

Typical sensor data used for fault-diagnosis are voltage and current measurements at electric components which need to be communicated from their origin at the sensors to the relevant processing units through a distributed network. For the analyses and evaluation of our compression algorithms we simulated data with the hybrid-electric vehicle (HEV)

Simulink model[2]. It allows to realistically simulate sensor measurements and state variables at the various included electrical and mechanical components according to a driving cycle input. A detailed introduction to the model can be found in [4].

The experimental results for the simultaneous two-dimensional data stream compression are based on the voltage and current measurements at the DCDC converter of the Simulink model output of a WLTP[3]-Class 3 driving cycle simulation. The signals are suitable for the evaluation as they offer a challenging signal behavior and can be seen as representatives of signals often to be analyzed in diagnosis systems. They include a broad coverage of quantization levels, slowly varying as well as rapidly varying signal sequences. Each data set contains 180100 samples, quantized with 16 bits with a sampling frequency of $100\,Hz$. Moreover, they show a correlation based on their physical relation, i.e., the two signals rise and drop in a similar fashion, often with different offsets. The implementations can be found on our project website[4].

The definition of a suitable shape of an active region in the sense of Section III-E significantly influences the compression ratio for the simultaneous compression of several data streams. The heat map in Figure 2 shows a region with $7 \times 7 = 49$ squares and the relative center $u_1 = u_2 = 0$. For the analysis we set $r = 6$ and used only 49 out of the available 63 dictionary entries. According to Section III-C, the region is always centered around the last transmitted sample. From darker to lighter colors we see that consecutive samples are covered by squares lying on a diagonal of the region. This is expected if two signals rise and fall in a similar fashion. Based on this analysis, a subset of the shown squares can be used for smaller dictionary sizes (see Table II), or more squares can form larger regions. A white square indicates that no input sample was covered by that square, meaning that this square is dispensable and can be removed from the region without increasing the number of lost data values.

Figure 3 shows the relationship between the loss rate (the total number of lost data values divided by the total number of data values in all streams) and the compression ratio for the combined compression of the two correlated signals with different settings for the parameters r and t. All results are calculated according to the static cache-based algorithm (Section III-B) extended by the grouping technique from Section III-E. Moreover, we highlight the partial miss strategy (Section III-D), which reduces the loss rate for all compression ratios, e.g., compare the blue line with the square markers with the red line with the diamond markers.

Figure 4 demonstrates the improvements of the simultaneous compression of two data streams when the hypercubes in the dictionary are maintained as one region according to

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

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

[4]https://networked-embedded.de/es/index.php/dakodis.html

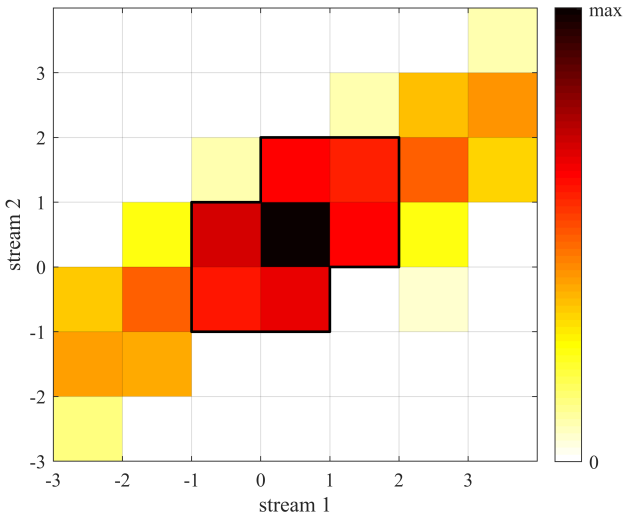

Fig. 2. Heat map (logarithmic scale) of the hitting squares around the center. The active region consists of 49 hypercubes (here squares).

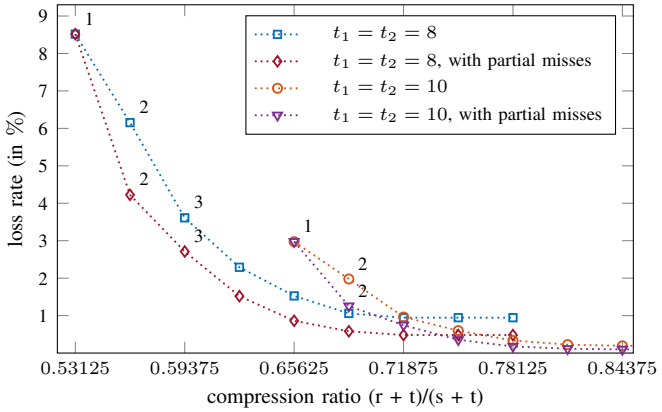

Fig. 3. Simultaneous compression of two data streams. Comparison of the cache-based algorithm with its improved version supporting partial misses. For all dictionary sizes one active region is maintained. The value for r is written to the data points. Lines between data points are shown for illustration only.

Section III-E. For the tail lengths $t_1 = t_2 = 8$ (compare the blue line with square markers with the red line with diamond markers) the loss rate drops from $2.2\,\%$ to $0.34\,\%$ at a compression ratio of 0.625. Moreover, we see that larger dictionaries with smaller hypercubes (here squares) enable to better adjust them to the signal, e.g., at a compression ratio of 0.625, the red line lies below the purple line.

In Figure 5 we compare the combined compression of the two data streams (according to the dynamic cache-based algorithm with the improvements from Sections III-D and III-E) with the two streams compressed individually (dynamic cache-based algorithm with $d = 1$ and grouping according to Section III-E), where the loss rate is calculated from the overall number of lost samples in both streams divided by the overall number of transmitted samples. To have comparisons for all compression ratios we performed two simulations for the individual stream compressions, one where the tail sizes

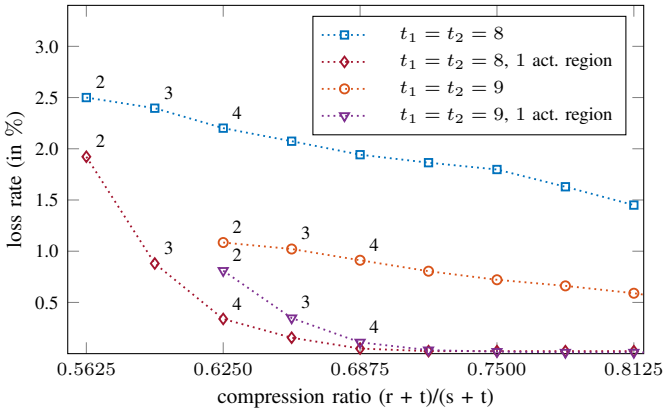

Fig. 4. Simultaneous compression of two data streams. Comparison of individually maintained hypercubes and a region of hypercubes. The value for r is written to the data points. Lines between data points are shown for illustration only.

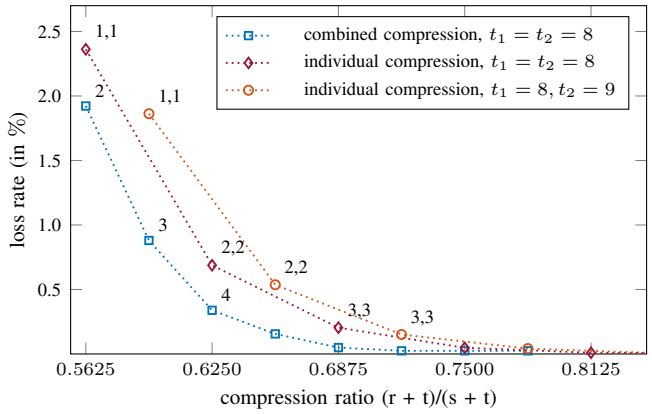

Fig. 5. Comparison of the dynamic simultaneous compression of two data streams with the compression of the individual data streams. The values for r are written to the data points. Lines between data points are shown for illustration only.

are $t_1 = t_2 = 8$ (red line with diamond markers) and another one with $t_1 = 8, t_2 = 9$ (orange line with circles). The results show, that the simultaneous compression of the two data streams outperforms the individual stream compression up to compression ratios of about 0.75, as the blue line with the square markers is the lowest. It is to be expected that this effect advantageously scales with the combination of more correlated streams. The fact that combined compression enables shorter message sizes without increasing the loss rate is especially important in the context of scheduling. This allows to reduce the makespan (and therefore leads to a better overall system performance) by computing a different job allocation as was demonstrated in [7]. Due to the selection of our example signals we expect similar results for many other signals and applications.

V. Future Work

The performance of the combined compression with grouping greatly relies on the defined active region to be able to beneficially capture the signal characteristics in the product space. As one cannot expect to always have this knowledge, or that a defined region works well at all times, further improvements to the algorithms include an automatic generation of suitable regions, which are also able to dynamically adapt to the signal characteristics over time, e.g., through a short time history analysis of the signal behaviors.

Moreover, new challenges arise when the achieved results are combined with the scheduling algorithms for the time-triggered DAKODIS architecture. Especially with respect to the makespan we want to focus on the possibilities to automatically determine when it is beneficial to combine some data streams and when they are better to be compressed individually.

VI. Conclusion

In this paper we firstly motivate the usage of data compression in distributed online-diagnosis systems. We highlight the fact that diagnostic data streams (e.g., sensor measurements of physical quantities) are often highly correlated and present compression algorithms that allow a simultaneous compression of such data streams utilizing the special signal characteristics. The introduced algorithms support real-time architectures by guaranteeing a fixed compression ratio. With an example we show that, depending on the parameters, more than one third of the bits (34.3 %) can be saved while still not losing more than 0.2 % of the values (see the fourth data point on the blue curve in Figure 5). In turn, this means that 99.8 % of the values are delivered correctly and without any losses but with a significant reduction of bandwith demands.

Acknowledgment

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

References

[1] Pier Luigi Dragotti and Michael Gastpar. *Distributed Source Coding - Theory, Algorithms and Applications*. Academic Press, 2009.
[2] Nan Guan. *Techniques for building timing-predictable embedded systems*. Springer, 2016.
[3] Rolf Isermann. *Fault-diagnosis systems: an introduction from fault detection to fault tolerance*. Springer Science & Business Media, 2006.
[4] Seungbum Jo, Markus Lohrey, Damian Ludwig, Simon Meckel, Roman Obermaisser, and Simon Plasger. An architecture for online-diagnosis systems supporting compressed communication. *Microprocessors and Microsystems*, 61:242 – 256, 2018.
[5] Hermann Kopetz. *Real-time systems: design principles for distributed embedded applications*. Springer Science & Business Media, 2011.
[6] Józef Korbicz, Jan M Koscielny, Zdzislaw Kowalczuk, and Wojciech Cholewa. *Fault diagnosis: models, artificial intelligence, applications*. Springer Science & Business Media, 2012.
[7] Setareh Majidi and Roman Obermaisser. Genetic algorithm for scheduling time-triggered communication networks with data compression. In *IEEE International Conference on Industrial Cyber-Physical Systems (ICPS)*, 2019.
[8] Francesco Marcelloni and Massimo Vecchio. An efficient lossless compression algorithm for tiny nodes of monitoring wireless sensor networks. *The Computer Journal*, 52(8):969–987, 2009.
[9] Khalid Sayood. *Introduction to Data Compression, fourth edition*. Morgan Kaufmann, 2012.
[10] Massimo Vecchio, Raffaele Giaffreda, and Francesco Marcelloni. Adaptive lossless entropy compressors for tiny IoT devices. *IEEE Trans. Wireless Communications*, 13(2):1088–1100, 2014.