An Energy-Efficient Fault-Tolerant Scheduling Algorithm Based on Variable Data Fragmentation

Chafik Arar, Mohamed Salah Khireddine, Abdelouahab Belazoui, Randa Megulati

An energy-efficient fault-tolerant scheduling algorithm based on variable data fragmentation

Chafik ARAR, Mohamed Salah KHIREDDINE, Abdelouahab BELAZOUI,
and Randa MEGULATI

Department of Computer Science, University of Banta
BATNA 05000, ALGERIA
{chafik.arar@gmail.com
mkhireddine,belazoui@yahoo.fr
randa_meguellati@hotmail.fr}
http://www.univ-batna.dz

Abstract. In this article, we propose an approach to build fault-tolerant distributed real-time embedded systems. From a given system description and a given fault hypothesis, we generate automatically a fault tolerant distributed schedule that achieves low energy consumption and high reliability efficiency. Our scheduling algorithm is dedicated to multi-bus heterogeneous architectures with multiple processors linked by several shared buses, which take as input a given system description and a given fault hypothesis. It is based on active redundancy to mask a fixed number L of processor failures supported in the system, and passive redundancy based on variable data fragmentation to tolerate N buses failures. In order to maximize the systems reliability, the replicas of each operation are scheduled on different reliable processors and the size of each fragmented data depends on GSFR and the bus failure rates. Finally, we show with an example that our approach can maximize reliability and reduce energy consumption when using active redundancy.

Keywords: Energy consumption, Scheduling, Embedded systems, Real time systems, Reliability, Active redundancy, Multi-bus architecture, variable data fragmentation.

1 Introduction

Nowadays, heterogeneous systems are being used in many sectors of human activity, such as transportation, robotics, and telecommunication. These systems are increasingly small and fast, but also more complex and critical, and thus more sensitive to faults. Due to catastrophic consequences (human, ecological, and/or financial disasters) that could result from a fault, these systems must be fault-tolerant. This is why fault tolerant techniques are necessary to make sure that the system continues to deliver a correct service in spite of faults Jalote [1], [2],

A fault can affect either the hardware or the software of the system; we chose to concentrate on hardware faults. More particularly, we consider processors and communication faults [3], [4]. In the literature, we can identify several fault-buses tolerance approaches for distributed embedded real-time systems, which we classify into two categories: proactive or reactive schemes.

In the proactive scheme [5], [6], multiple redundant copies of a message are sent along distinct buses. In contrast, in the reactive scheme only one copy of the message, called primary, is sent; if it fails, another copy of the message, called backup, will be transmitted. In [7], an original off-line fault tolerant scheduling algorithm which uses the active replication of tasks and communications to tolerate a set of failure patterns is proposed; each failure pattern is a set of processor and/or communications media that can fail simultaneously, and each failure pattern corresponds to a reduced architecture. The proposed algorithm starts by building a basic schedule for each reduced architecture plus the nominal architecture, and then merges these basic schedules to obtain a distributed fault tolerant schedule. It has been implemented in [8].

In [9], a method of identifying bus faults based on a support vector machine is proposed. In [2], faults of buses are tolerated using a TDMA (Time Division Multiple Access) communication protocol and an active redundancy approach. In [10] authors propose a fine grained transparent recovery, where the property of transparency can be selectively applied to processes and messages. In [11] authors survey the problem of how to schedule tasks in such a way that deadlines continue to be met despite processor and communication media (permanent or transient) or software failure.

In this paper, we are interested in approaches based on scheduling algorithms that maximize reliability and reduce energy consumption [12], [13], [14] when using active redundancy to tolerate processors faults and passive redundancy based on variable data fragmentation to tolerate buses faults.

The remaining of this paper is structured as follows: In section 2, we give detailed description of our system models. In section 3, we present our solution and we give detailed description of our scheduling algorithm. Section 4 shows with an example how our approach can maximize reliability and reduce energy consumption when using active redundancy. We finally conclude this work in section 5.

2 System Description

Distributed real-time embedded systems are composed of two principal parts, which are the algorithm (software part) and the distributed architecture (hardware part). The specification of these systems involve describing the algorithm (algorithm model), the architecture (architecture model), and the execution characteristics of the algorithm onto the architecture (execution model).

The algorithm is modeled as a data-flow graph noted ALG. Each vertex of ALG is an operation (task) and each edge is a data-dependence. A data-dependence, noted by →, corresponds to a data transfer between a producer operation and a consumer operation. $t_1 \rightarrow t_2$ means that t_1 is a predecessor of t_2

and t_2 is a successor of t_1. Operations with no predecessor (resp. no successor) are the input interfaces (resp. output).

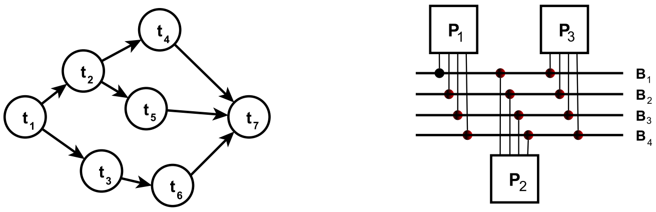

Fig. 1. ALG and ARC graphs.

The architecture is modeled by a non-directed graph, noted ARC, where each node is a processor, and each edge is a bus. Classically, a processor is made of one computation unit, one local memory, and one or more communication units, each connected to one communication link. Communication units execute data transfers. We assume that the architecture is heterogeneous and fully connected. Figure 1 presents an example of ALG with seven operations t_1, t_2, t_3, t_4, t_5, t_6 and t_7 and ARC, with three processors P_1, P_2, P_3 and four buses B_1, B_2, B_3 and B_4.

Our real-time system is based on cyclic executive; this means that a fixed schedule of the operations of ALG is executed cyclically on ARC at a fixed rate. This schedule must satisfy one real-time constraint which is the length of the schedule. As we target heterogeneous architecture, we associate to each operation t_i a worst case execution time (WCET) on each processor P_j of ARC, noted $Exe(t_i, P_j)$. Also, we associate to each data dependency $data_i$ a worst case transmission time (WCTT) on each bus B_j of the architecture, noted $Exe(data_i, B_j)$.

We assume only processors and buses failures. We consider only transient bus faults, which persist for a short duration. We assume that at most L processors faults and N bus faults can arise in the system, and that the architecture includes more than L processors and N buses.

3 The Proposed Approach

In this section, we first discuss the basic principles used in our solution, based on scheduling algorithms. Then, we describe in details our scheduling algorithm. The aims of this algorithm are twofold, first, maximize the reliability of the system and minimize the length of the whole generated schedule in both presence and absence of failures; Secondly, reduce energy consumption. In our approach. we achieve high reliability, reducting consuption and fault tolerance in tow ways:

3.1 Active redundancy with changing frequency

In order to tolerate up to L arbitrary processors faults, our solution is based on active redundancy approach. The advantage of the active redundancy of operations is that the obtained schedule is static; in particular, there is no need for complex on-line re-scheduling of the operations that were executed on a processor when the latter fails; also, it can be proved that the schedule meets a required real-time constraint, both in the absence and in the presence of faults. In many embedded systems, this is mandatory. To tolerate up to L processors faults, each operation t of Alg is actively replicated on *L+1* processors of Arc (see Figure 2). We assume that all values returned by the *L+1* replicas of any operation t of Alg are identical.

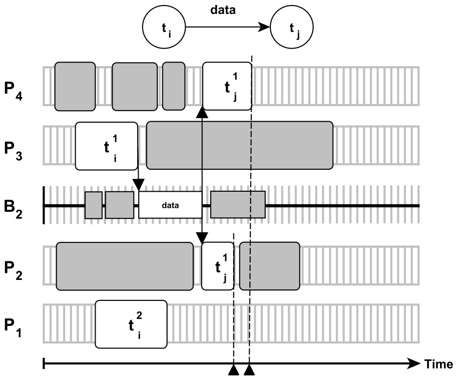

Fig. 2. Active redundancy

Voltage, frequency and energy consumption: the maximum supply voltage is noted V_{max} and the corresponding highest operating frequency is noted F_{max}. For each operation, its WCET assumes that the processor operates at F_{max} and V_{max} (and similarly for the WCCT of the data-dependencies). Because the circuit delay is almost linearly related to 1/V, there is a linear relationship between the supply voltage V and the operating frequency F. In the sequel, we will assume that the operating frequencies are normalized, that is, $F_{max} = 1$ and any other frequency F is in the interval $[0, 1]$. Accordingly, the execution time of the operation or data-dependency M placed onto the hardware component C (be it a processor or a communication link) running at frequency F (taken as a scaling factor) is :

$$Exe(M, C, F) = \frac{Exe(M, C)}{F} \tag{1}$$

To calculate the power consumption, we follow the model presented in [15]. For a single operation placed onto a single processor, the power consumption P is :

$$P = P_s + h(P_{ind} + P_d) \tag{2}$$

Where P_s is the static power (power to maintain basic circuits and to keep the clock running), h is equal to 1 when the circuit is active and 0 when it is inactive, P_{ind} is the frequency independent active power (the power portion that is independent of the voltage and the frequency; it becomes 0 when the system is put to sleep, but the cost of doing so is very expensive),

$$P_d = C_{ef} * V^2 * F \tag{3}$$

P_d is the frequency dependent active power (the processor dynamic power and any power that depends on the voltage or the frequency), C_{ef} is the switch capacitance, V is the supply voltage, and F is the operating frequency.

For processors, this model is widely accepted for average size applications, where C_{ef} can be assumed to be constant for the whole application. For a multiprocessor schedule S, we cannot apply directly the previous equation. Instead, we must compute the total energy *E(S)* consumed by S, and then divide by the schedule length *L(S)*:

$$P(S) = \frac{E(S)}{L(S)} \tag{4}$$

We compute *E(S)* by summing the contribution of each processor, depending on the voltage and frequency of each operation placed onto it. On the processor P_i, the energy consumed by each operation is the product of the active power $P^i_{ind} + P^i_d$ by its execution time.

In our approach, as *L+1* replicas of each operation are scheduled actively on *L+1* distinct processors, the energy consumed by the system is maximal. In order to reduce energy consumption, we propose to execute the *L+1* replicas of an operation with different frequencies *F*. As all the *L+1* replicas of an operation may have different end execution time (see Figure 2 for the replicas t^1_j and t^2_j), we choose to align the execution time of all the replica by changing the frequency *F* of each replica (As shown in Figure 3).

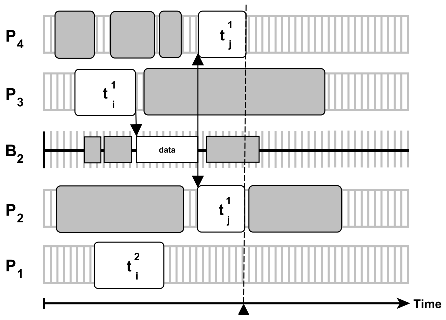

Fig. 3. Changing the frequency of t^1_j.

3.2 Passive redundancy with variable data fragmentation

In order to use efficiently the bus redundancy of the architecture, we propose to use a mechanism of communication, based on variable data fragmentation. Variable data fragmentation allows the fast recovering from buses errors, and it may also reduce the error detection latency. (the time it takes to detect the error). The communication of each data dependency $t_i \to t_j$ is fragmented into *N+1* fragments $data = data_1 \bullet \cdots \bullet data_{N+1}$, sent by t_i to t_j via *N+1* distinct buses (see Figure 4); The associative operation ($\bullet$) is used to concatenate two data packets. As our approach uses variable data fragmentation, the size of each fragmented data depends on *GSFR* and the bus failure rates λ_B.

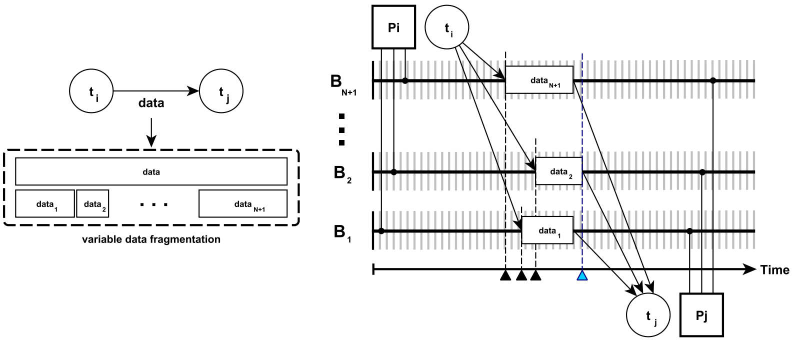

Fig. 4. Variable data fragmentation.

GSFR is the failure rate per time unit of the obtained multiprocessor schedule. Using the GSFR is very satisfactory in the area of periodically executed schedules. In such cases, applying brutally the exponential reliability model yields very low reliabilities due to very long execution times (the same remark applies also to very long schedules). Hence, one has to compute beforehand the desired reliability of a single iteration from the global reliability of the system during its full mission; but this computation depends on the total duration of the mission and on the duration of one single iteration.

Our fault tolerance heuristic is GSFR-based to control precisely the scheduling of each fragmented data from the beginning to the end of the schedule. In [16], The GSFR of scheduling an operation t_i, noted $\Lambda(S_n)$, by the following equation:

$$\Lambda(S_n) = \frac{-\log(\prod_i e^{-\lambda_k exe(t_i,P_j)+\sum_k \sum_j \lambda_c exe(dpd_j^k,b_c)})}{\sum_i^j exe(t_i,p_j) + \sum_k^m exe(dpd_k,b_m)} \tag{5}$$

Variable data fragmentation operates in three phases :

1. First, in order to tolerate at most *N* communication bus errors, each data dependency is fragmented into *N+1* fragments of equal size. The initial size

of each fragment is calculated by :

$$Size(data_i) = \frac{Size(data)}{N+1} \tag{6}$$

The main problem with the equal size data fragmentation comes from the difference between ending time of different fragments (Figure 5(a)) because the destination operation must wait to getting all the fragments of the data dependency to start execution.

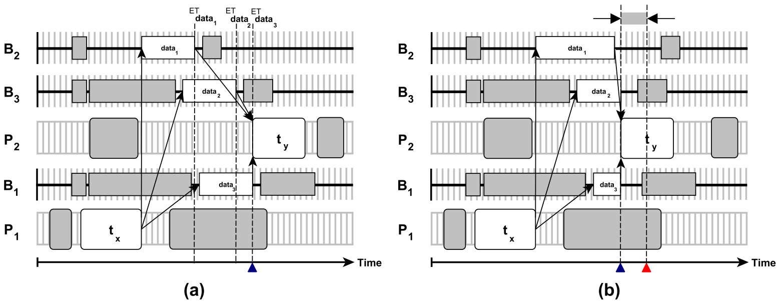

Fig. 5. Ending time : (a) ET in equal size data fragmentation, (b) Minimize difference between ending time

2. Second, the goal of passing from equal size data fragmentation to variable data fragmentation (Figure 5(a)) is to minimize the difference between ending time ET of different fragments (Figure 5(b)).

$$\begin{array}{l} ET_{data_1} \le ET_{data_2} \le \cdots \le ET_{data_{N+1}} \\ Minimize(ET_{data_{i+1}} - ET_{data_i})_{i \in \{1,\cdots,N+1\}} \end{array} \tag{7}$$

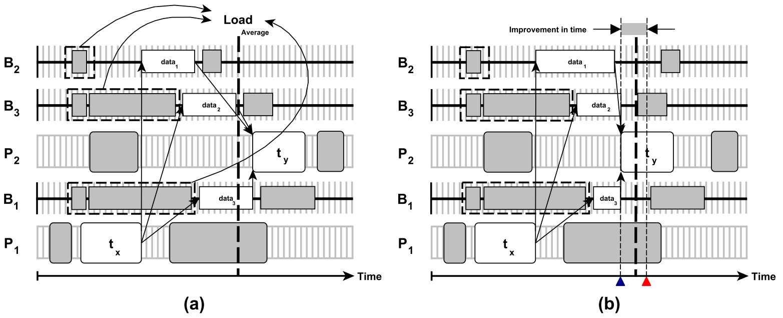

Fig. 6. (a) the Average Load $Load_{Average}$, (b) and the improvement in time of the scheduling.

With variable data fragmentation based on minimizing the difference between ending time, another problem can occur and grows extremely the execution time. The bus over which accumulates data may also fail, therefore the quantity of data to be retransmitted is more important.

3. Third, the definition of a compromise between the load of each communication bus and the maximum data to be transmitted on this bus, as illustrated in Figure 6(a). Variable data fragmentation must not exceed this value when defining the new fragments size. The improvement in time of the scheduling is shown in Figure 6(b).

The algorithm that enable variable data fragmentation is show in figure 7.

```
**Algorithm VDF**
**Input**: data-dependence ($data = t_i \rightarrow t_j$), $N$.
**Output**: the set of $N+1$ affectation ($data_i(B_x)$).

1. Each data dependency ($data = t_i \rightarrow t_j$) is fragmented into $N+1$ fragments of equal size:
       $$Size(data_1) = \cdots Size(data_{N+1}) = \frac{Size(data)}{N+1}$$
2. Compute the loading sill of buses.
       $$Load_{Average} = \frac{\sum \lambda_{B_i} * Load(B_i)}{N+1}$$
3. Schedule the $N+1$ fragments of data-dependence on $N+1$ bus.
4. Order the data fragments according to their ending Time.
       $$ET_1 \leq ET_2 \leq \cdots \leq ET_{N+1}$$
5. Compute the sum of the shift of Ending Time.
       $$Sum^{new}_{shift-time} := 0; Sum_{shift-time} = \sum ET_{i+1} - ET_i;$$
6. **While** ($Sum^{new}_{shift-time} \leq Sum_{shift-time}$) **do**
   (a) $Sum_{shift-time} := Sum^{new}_{shift-time}$.
   (b) Fragment the data Fragment with the last end time on tow
       fragments ($data(ET_{N+1}) = data_A \bullet data_B$), respecting the following three conditions:
         i. $Size(data_A) \geq Siz_{min}(data_{ET_1})$
        ii. $Siz(data_{ET_1} + Size(data_B) \leq Load_{Average}$
       iii. $ET_1 + Size(data_B)B_{data_1} \leq ET_{N+1}$
   (c) Order the data fragments according to their new ending time $ET_i$.
   (d) Compute the new value of $Sum^{new}_{shift-time}$
           $$Sum^{new}_{shift-time} = \sum ET_{i+1} - ET_i;$$
   **End While**.
**End**
```

Fig. 7. VDF : the variable data fragmentation algorithm.

3.3 Scheduling Algorithm

The principles of our approach are implemented by a scheduling algorithm, called Energy Fault Tolerant Heuristic *(EFTH-VDF)*. It is a greedy list scheduling heuristic, which schedules one operation at each step (n). It generates a distributed static schedule of a given algorithm Alg onto a given architecture Arc, which minimizes the system's run-time, and tolerates upto L processors and N buses faults, with respect to the real-time and the distribution constraints. At each step of the greedy list scheduling heuristic, the pressure schedule function (noted by $\sigma(n)(t_i, P_j))$ is used as a cost function to select the best operation to be scheduled.

$$\sigma^{(n)}(t_i, P_j) = S^{(n)}_{t_i,P_j} + \overline{S}^{(n)}_{t_i} - R^{(n-1)} \tag{8}$$

The *EFTH-VDF* algorithm (show in figure 8) is divided into seven steps.

```
Algorithm EFTH-VDF
Input: $ALG$, $ARC$, $N$;
Output: a reliable fault-tolerant schedule;

Initialize the lists of candidate and scheduled operations:
n := 0;
$T^{(0)}_{cand} := \{t \in T \mid pred(t) = \emptyset\}$;
$T^{(0)}_{sched} := \emptyset$;

While ($T^{(n)}_{cand} \neq \emptyset$) do
    1. For each candidate operation $t_{cand}$, compute $\sigma^{(n)}$ and GSFR on each processor $P_k$.
    2. For each candidate operation $t_{cand}$, select the best processor $p^{t_{cand}}_{best}$ which minimizes
       $\sigma^{(n)}$ and GSFR.
    3. Select the most urgent candidate operation $t_{urgent}$ between all $t^i_{cand}$ of $T^{(n)}_{cand}$.
    4. For each data dependencies whose $t_{urgent}$ is the producer operation: Fragment
       the data communication on $N$ fragments using the variable data fragmentation algorithm;
    5. Schedule $t_{urgent}$ and its fragmented data;
    6. Update the lists of candidate and scheduled operations:
       $T^{(n)}_{sched} := T^{(n-1)}_{sched} \cup \{t_{urgent}\}$;
       $T^{(n+1)}_{cand} := T^{(n)}_{cand} - \{t_{urgent}\} \cup \{t' \in succ(t_{urgent}) \mid pred(t') \subseteq T^{(n)}_{sched}\}$;
    7. n := n + 1;

End while
End
```

Fig. 8. The EFTH-VDF algorithm.

4 Simulations, results and discussion

We have applied the *EFTH-VDF* heuristic to an example of an algorithm graph and an architecture graph composed of four processors and four buses. The algorithm graph is show in Figure 9. The failure rates of the processors are respectively 10^{-5}, 10^{-5}, 10^{-6} and 10^{-6}, and the failure rate of the Buses SAM_{MP1},

SAM_{MP2}, SAM_{MP3} and SAM_{MP4} are respectively 10^{-6}, 10^{-6}, 10^{-5} and 10^{-4}.

Figure 10 shows the non-fault-tolerant schedule produced for our example with a basic scheduling heuristic. (for instance the one of SynDEx). SynDEx [17] is a tool for optimizing the implementation of real-time embedded applications on multi-component architecture.

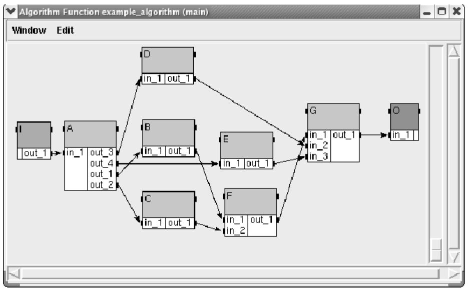

Fig. 9. Algorithm graph.

Figure 11 shows the fault-tolerant schedule produced for our example with a *EFTH-VDF* scheduling heuristic without changing frequencies. The schedule length generated by this heuristic is 21.6. The GSFR of the non-reliable schedule is equal to 0.0000287. The energy E is equal to 36.7.

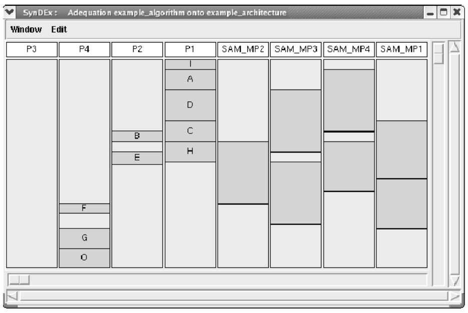
SynDEx : Adequation example_algorithm onto example_architecture
Window Edit
P3
P4
P2
P1
SAM_MP2
SAM_MP3
SAM_MP4
SAM_MP1
I
A
D
C
H
B
E
F
G
O

Fig. 10. Schedule generated by SynDEx.

Figure 12 shows the fault-tolerant schedule produced for our example with a *EFTH-VDF* scheduling heuristic. The schedule length generated by this heuristic is 27.3. The GSFR of the non-reliable schedule is equal to 0.0000276. The energy E is equal to 23.21.

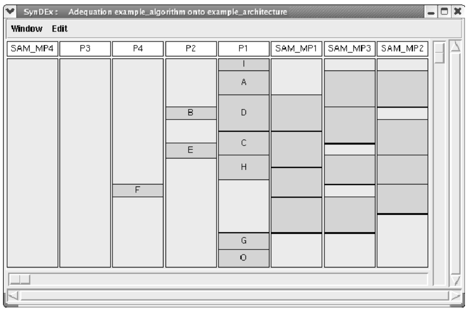
SynDEx : Adequation example_algorithm onto example_architecture
Window Edit
SAM_MP4
P3
P4
P2
P1
SAM_MP1
SAM_MP3
SAM_MP2
I
A
B
D
C
E
H
F
G
O

Fig. 11. $EFTH - VDF$ without changing frequencies

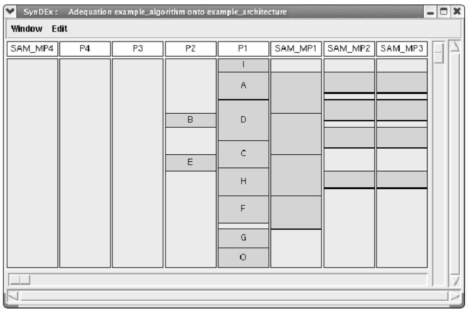
SynDEx : Adequation example_algorithm onto example_architecture
Window Edit
SAM_MP4
P4
P3
P2
P1
SAM_MP1
SAM_MP2
SAM_MP3
I
A
B
D
C
E
H
F
G
O

Fig. 12. A schedule generated by $EFTH - VDF$

5 Conclusion

We have proposed in this paper a solution to tolerate both processors and communication media faults in distributed heterogeneous architectures with multiple-bus topology. The proposed solution, based on active redundancy, is a list scheduling heuristic called *EFTH-VDF*. It generates automatically distributed static schedule of a given algorithm onto a given architecture, which minimizes the system's run-time, and tolerates upto L processors and N buses faults, with respect to real-time and distribution constraints. The scheduling strategy based on variable frequency and variable data fragmentation minimizes energy consumption and take communication failures into account.

References

1. Jalote, P.: Fault-Tolerance in Distributed Systems. Prentice Hall, Englewood Cliffs, New Jersey (1994)
2. Kopetz, H.: Real-time systems: design principles for distributed embedded applications. Springer Science & Business Media (2011)
3. Grünsteidl, G., Kantz, H., Kopetz, H.: Communication reliability in distributed real-time systems. Distributed Computer Control Systems 1991: Towards Distributed Real-Time Systems with Predictable Timing Properties p. 123 (2014)

4. Jun, Z., Sha, E.H., Zhuge, Q., Yi, J., Wu, K.: Efficient fault-tolerant scheduling on multiprocessor systems via replication and deallocation. International Journal of Embedded Systems 6(2), 216–224 (2014)
5. Kandasamy, N., Hayes, J.P., Murray, B.T.: Dependable communication synthesis for distributed embedded systems. Reliability Engineering & System Safety 89(1), 81–92 (2005)
6. Dulman, S., Nieberg, T., Wu, J., Havinga, P.: Trade-off between traffic overhead and reliability in multipath routing for wireless sensor networks. In: Wireless Communications and Networking Conference (2003)
7. Lavarenne;, C.D.A.G.C., Sorel, Y.: Off-line real-time fault-tolerant scheduling. In: 9th Euromicro Workshop on Parallel and Distributed Processing. pp. 410–417 (2001)
8. C. Pinello, L.C., Vincentelli, A.S.: Fault-tolerant deployment of embedded software for cost-sensitive real-time feedback-control applications design. In: Automation and Test in Europe , DATE'04, IEEE (2004)
9. Song, H., Wu, H.: The applied research of support vector machine in bus fault identification. In: Natural Computation (ICNC), 2010 Sixth International Conference on. vol. 3, pp. 1326–1329. IEEE (2010)
10. Izosimov, V., Pop, P., Eles, P., Peng, Z.: Scheduling and optimization of fault-tolerant embedded systems with transparency/performance trade-offs. ACM Transactions on Embedded Computing Systems (TECS) 11(3), 61 (2012)
11. Krishna, C.: Fault-tolerant scheduling in homogeneous real-time systems. ACM Computing Surveys (CSUR) 46(4), 48 (2014)
12. Huang, J., Buckl, C., Raabe, A., Knoll, A.: Energy-aware task allocation for network-on-chip based heterogeneous multiprocessor systems. In: Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on. pp. 447–454. IEEE (2011)
13. Agrawal, P., Rao, S.: Energy-aware scheduling of distributed systems using cellular automata. In: Systems Conference (SysCon), 2012 IEEE International. pp. 1–6. IEEE (2012)
14. Agrawal, P., Rao, S.: Energy-aware scheduling of distributed systems. IEEE (2014)
15. Zhu, D., Melhem, R., Mosse, D., Elnozahy, E.: Analysis of an energy efficient optimistic tmr scheme. In: Parallel and Distributed Systems, 2004. ICPADS 2004. Proceedings. Tenth International Conference on. pp. 559–568. IEEE (2004)
16. Girault, A., Kalla, H.: A novel bicriteria scheduling heuristics providing a guaranteed global system failure rate. IEEE Transactions on Dependable and Secure Computing 6(4), 241–254 (2009)
17. Forget, J., Gensoul, C., Guesdon, M., Lavarenne, C., Macabiau, C., Sorel, Y., Stentzel, C.: Syndex v7 user manual (2013)