Introducing Fault Tolerance into Process Networks

Jonas Ceponis, Egidijus Kazanavicius, Lina Ceponiene

Kaunas University of Technology, Studentu 50-202, LT 51368 Kaunas, Lithuania,
cepojona@ifko.ktu.lt, ekaza@ifko.ktu.lt, lina.ceponiene@ktu.lt

Abstract

Process network is a computation model in which many concurrent processes communicate through unbounded FIFO buffer and can be executed simultaneously. In real time digital signal processing applications execution time is infinite. However, failures of implementation hardware can occur. In our work, dynamic run-time reconfiguration is introduced into process network which ensures error handling, avoiding deadlocks, continuous and on-time result delivery. After dynamic reconfiguration, network execution results may become non deterministic, but this helps avoiding critical termination of network execution. In this paper, we present a description of possible failures of network execution and discuss the means for avoiding these failures.

Keywords: Process Networks, Data Flow, Fault Tolerant, Dynamic Reconfiguration, Concurrent Processes.

1. Introduction

Process Networks (PN) is often used in digital signal processing (DSP) and control applications. In these systems various methods for analysis, design, and implementation of the problem are applied. For effective implementation of the problem in PN, this problem must be decomposed into smaller parallel tasks. Each task is implemented in separate network node, which is also called actor. In process network concurrent processes representing the nodes communicate through unidirectional first-in, first-out (FIFO) channels [Lee et al. (1995)].

A well-designed system should always have a powerful mechanism for handling faults and changes of environment [Avizienis et al. (2004)]. In the design stage of system development it is difficult to predict all possible changes in system environment or in the system itself. These changes can even cause faulty execution or complete termination of execution of the system. The faults in system execution can be caused by software or hardware. In order to avoid such situations well-chosen design and implementation methods must be used which enable proper reactions to the changes and faults. Possible faults of elements of the system which can cause termination of system execution must be carefully considered in various critical systems. Such systems must also properly respond to changes of internal states or environment. In response to these changes, the system must reconfigure itself. Therefore, reconfiguration can be used in ensuring fault tolerance of the system.

2. Related Work

Today a large variety of process networks used in DSP applications exists. These process networks can be grouped into two groups: non-configurable (Kahn Process Network (KPN), Synchronous Dataflow (SDF), Cyclo-Static Dataflow (CSDF)) and configurable process networks (Parameterized Synchronous Dataflow (PSDF), Reactive Process Networks (RPN), Configurable Hierarchical Dataflow (CHDF)).
KPN is a subset of more general Process Network (PN) model and is used as bases for Fault Tolerant Process Network (FTPN) presented in this work. KPN consists of concurrent processes communicating over first-in first out unidirectional queues [Kahn (1974)]. KPN is useful for modelling and exploiting functional parallelism in streaming data applications. KPN uses completely dynamic execution of nodes and there is no need for describing schedule during system design, because scheduling does not affect the functional behaviour of the nodes. These features are very useful in dynamic network reconfiguration [Hofstee et al. (2002)]. In KPN, nodes are executed asynchronously, but the result of network execution is deterministic.
Synchronous Dataflow network is applicable to simple dataflow systems without complicated flow of control. In SDF, a node produces and consumes a fixed number of data tokens on each of its outgoing and incoming channels during each activation [Lee et al. (1987)]. In Cyclo-Static Dataflow network [Bilsen, et al. (1996)] the algorithms are implemented which ensure cyclic execution of the network nodes.
In order to capture the interaction between input events and execution units as well as reconfiguration in dynamic stream processing, reactive process networks are introduced [Geilen et al. (2004)]. The foundation for RPN was laid by efforts to integrate dataflow model and its reactive behaviour. Another means for specifying dynamic network reconfiguration during run-time is parameterizable SDF model [Bhattacharya et al. (2001)]. In PSDF, node execution is characterized by iterations that fire subprocesses in a particular order. Node execution can be reconfigured between iterations at run-time [Haim et al. (2006)]. Yet another approach for dynamic process network reconfiguration is Configurable Hierarchical Dataflow [Neuendorffer et al. (2004)]. It is concentrated on reconfiguration as a particular kind of event handling. The states of the network, when reconfigurations are allowed, are named quiescent states. CHDF focuses primarily on reconfiguration of SDF networks.
Our work differs from the above discussed approaches because it presents another point of view to the purpose of process network dynamic reconfiguration. This point of view is based on the idea that critical moments [Olson et al. (2005)] in network execution must be specified in order to avoid critical termination. These critical moments appear when communication between network nodes is broken or node fails. Fault tolerant process network proposed in this work can be reconfigured dynamically in order to capture and process critical moments of network execution [Čeponis et al. (2002)].The proposed FTPN enables further execution of the network

after faults of its elements using remaining operational resources. FTPN fits for both data processing and various types of control applications [Čeponis et al. (2006)].

3. A Model of Fault Tolerant Process Network

In our research the Fault Tolerant Process Network is developed. This process network is dynamically reconfigurable and the main purpose of dynamic network reconfiguration is reaction to failures in process network and handling of these failures. Moreover, dynamic reconfiguration can be used to ensure more effective consumption of available computational and memory resources.

First, some common definitions and notations for process network are described. For FIFO channel specification there is universal finite set of channels *CH* and for every channel $ch \in CH$ there is a corresponding finite channel alphabet Σ. Each channel $ch \in CH$ is described by its length *L* and pointer *ch(p)* which refers to the last data record in the channel. The actions for data transfer through the channel are $ch \rightarrow a, ch \leftarrow a | ch \in CH, a \in \Sigma$. The action $ch \leftarrow a$ denotes input of data into channel. The action $ch \rightarrow a$ denotes output of data from channel. These actions form the set of actions for data transfer $Ac=\{ch \rightarrow, ch \leftarrow a | ch \in CH, a \in \Sigma\}$.

For network nodes specification, a universal finite set of nodes *N* is used and for every node of this set $n \in N$ there is a corresponding set of atomic actions *Act*. All actions of all network nodes are defined by the set *A* and the actions of every node $Act \subseteq A$. Every node has a set of input and output channels $(ch_{in}, ch_{out}) \in CH$. The duration of execution of each action is also specified.

3.1. The Specification of Nodes in Fault Tolerant Process Network

In functional specification of FTPN, the operation of each node must be defined as a sequence of atomic actions. Such specification is necessary for implementation of dynamic reconfiguration and change of network parameters when operations must be allocated from one node to another. The set of operations for each node Act_n is specified as a sequence of atomic actions (Act_1, Act_2, ... ,Act_{k-1}, Act_k), where *k* is the number of atomic actions in the node.

The nodes in FTPN can perform operations $Cntrl_n$ which are used for network reconfiguration and changing network parameters. In order to define functional specification of the node in FTPN, the states of its execution must be determined.

During communication operations the node performs input or output of data. After successful data input the node performs computation operations. When computations are over, the node writes output data to output channels ch_{out}. Control operations are executed in case of dynamic network reconfiguration or changing network parameters. Execution of control operations can be planned (if reallocation of network resources is required) or unplanned (in case of failure of network element).

The node can be in one of the states $Ns = \{ns_{\perp}, ns_r, ns_{br}, ns_w, ns_{bw}, ns_{ex}, ns_c\}$. Initial node state $ns_{\perp}$ denote the starting point of node execution. In this state, initial working

parameters and values of variables are set for the node. Afterwards, the node moves to reading state ns_r and tries to read data from input channels $ch_{in} \in CH$. If at least one $ch(p)=null \mid ch \in ch_{in}$, the node transits to state ns_{br} and waits until data are available in the channel. When the node has successfully read data from input channels, it moves to state ns_{ex} and executes actions $Act \subseteq A$. After finishing execution, the node transits to writing state ns_w and writes results to its output channels $ch_{out} \in CH$. If at least one $ch(p) \neq null \mid ch \in ch_{out}, p=L,$ the node moves to blocked writing state ns_{bw} and waits for available free space in the output channel. When the node has finished writing data to channels, it moves to state ns_r.
The change of network execution parameters may be required in two cases: a node is in blocked reading or blocked writing states (ns_{br} or ns_{bw}) and timeout occurs; or external request is received. In any of these two cases the node moves to control state ns_c and changes required parameters. Afterwards, the node transits to reading or writing state (ns_r or ns_w) and continues execution.
In FTPN all nodes are divided into two groups: internal nodes (having both input and output channels) and interface nodes (having only input or only output channels).
The FSP specification of the node which has both input and output channels is presented below:

```
Node = Initial,
Initial = (set_parameters -> Reading),
Reading = (reading_ok -> Execution | channel_empty -> BlockedReading),
Execution = (execution_ok -> Writing),
Writing = (writing_ok -> Reading | channel_full -> BlockedWriting),
BlockedReading = (channel_not_empty -> Reading | reading_timeout -> Control),
BlockedWriting = (channel_not_full -> Writing | writing_timeout -> Control),
Control = (reading_continue -> Reading | writing_continue -> Writing).
```

The FSP specification can be mapped to LTS [Huth (2005)] graph using the tool LTSA 2.2 [Magee et al. (2006)]. The LTS graph for the specified internal node is presented in Fig. 1.
In process networks interface nodes are those who have only input channels ($ch_{out}=\varnothing$) or only output channels ($ch_{in}=\varnothing$). The states of interface nodes differ from the states of internal nodes and are also specified in this work. Interface nodes having only input channels do not perform writing to the channel; therefore such nodes do not have reading and blocked reading states (nb_r and nb_{br}). These nodes usually represent outputs of the system under implementation and are directly connected to external objects. Therefore, failure of the interface node having only input channels can terminate execution of the whole process network. The interface node with only input channels has five possible states.

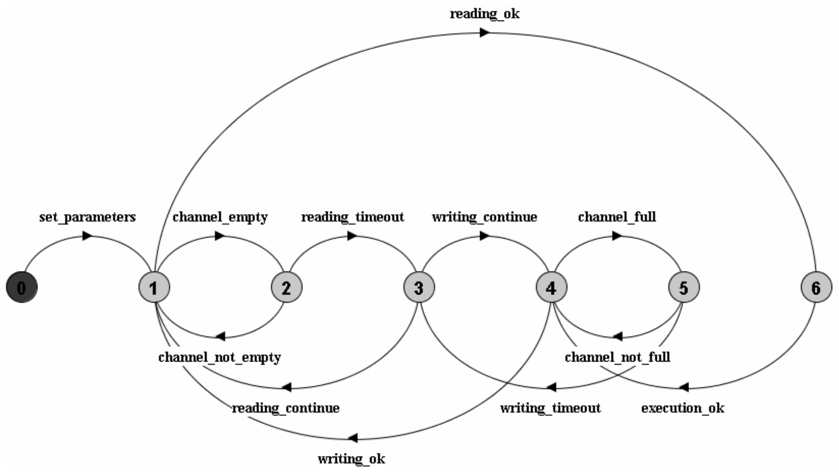

Figure 1. *The States of Internal Nodes in FTPN*

The other type of network interface nodes has only output channels $ch_{out} \in CH$. Such nodes do not perform operations of reading from communication channels and thus have less possible states than internal nodes. Interface nodes having only output channels cannot transit to writing and blocked writing states (nb_w and nb_{bw}).

3.2. The Specification of Channels in Fault Tolerant Process Network

The set $Sc=\{sc_{\perp},sc_w,sc_{cf},sc_{ce},sc_{pop},sc_{push},sc_{inc},sc_{dec}\}$ defines all possible states of the channel. After setting initial parameters in state $sc_{\perp}$, channel transits to waiting state sc_w. In this state channel waits for requests from the nodes. When request from writing node arrives, channel moves to state sc_{cf}, in which it checks the availability of free space in channel memory. If there is a free space in memory, channel moves to the state sc_{push}. During this state the incoming token is written to the channel. When request from reading node arrives, channel moves to state sc_{ce}, in which it checks the availability of data in channel. If there are at least one data token, channel moves to the state sc_{pop} and sends first data token to reading node. When the token from the writing node arrives and channel is full, channel transits to the state sc_{inc} in which the length of the channel is increased. When channel length increasing is successful it moves to state sc_{push}, otherwise it moves to state sc_w. The channel can transit to state sc_{dec} when a request from reading node arrives and other channels require to be increased. In state sc_{dec}, the length of the channel is decreased thus freeing memory for other channels. The FSP specification of channels in FTPN is presented below:

```
Channel = Initial,
Initial = (set_par -> Waiting),
Waiting = (read -> CheckFull | write -> CheckEmpty),
CheckFull = (full -> Waiting | not_full -> Push | inc -> Increase),
Push = (push_ok -> Waiting),
Increase = (inc_ok -> Push | inc_not_ok -> Waiting),
CheckEmpty = (empty -> Waiting | not_empty -> Pop | dec -> Decrease),
```

Decrease = (dec_ok -> Pop),
Pop = (pop_ok -> Waiting).

This FSP specification is mapped to LTS graph which is presented in Fig. 2.

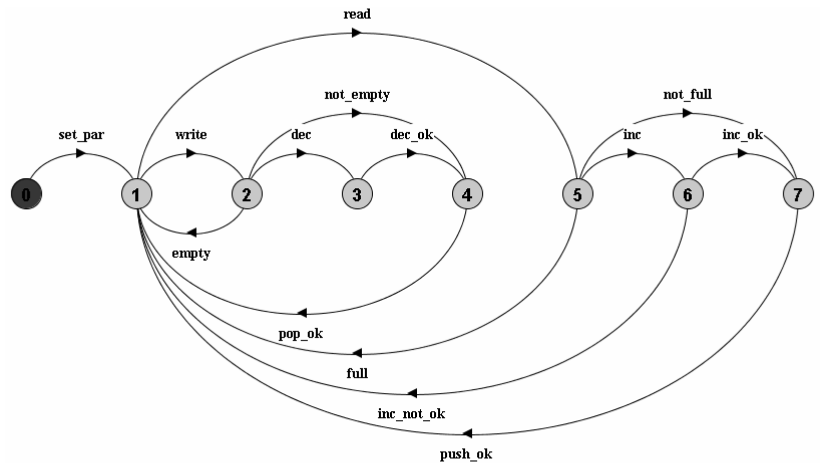

Figure 2. *The States of Channels in FTPN*

4. Algorithms for Handling Faults in Fault Tolerant Process Network

The algorithms for handling faults of elements in process network must be specified to ensure that network continues execution after failure. As the main elements of the process network are nodes and channels, the algorithms are specified for the cases of network node failure and for network channel failure. Network nodes are grouped into internal nodes (having input and output channels), interface nodes having only input channels, and interface nodes having only output channels; therefore the fault handling algorithms for each of these types are specified.

In case of internal network node failure we need to redistribute its actions to other node. We also need to redistribute the input and output channels of the faulty node. In order to minimize data loss, the reconfiguration must follow these rules:

- All nodes connected with faulty node *n* perform their actions until:
  - input channels of the faulty node become full;
  - output channels of the faulty node become empty.
- The nodes connected with faulty node *n* transit to blocked reading or blocked writing states.
- After timeout the nodes connected with faulty node *n* transit to control state. In this state:
  - actions of the faulty node *n* are transferred to node *n1* which first transits to control state and is connected to output channel of *n*;
  - the channel between *n* and *n1* is destroyed;

    - output channels of *n* are connected to *n1*;
    - input channels of *n* are connected to *n1*;
    - the data lost during failure is compensated.
- The nodes which are in control state move to writing or reading states and reconfigured network continues execution.

These rules are applicable to network nodes which have input and output channels $(ch_{in}, ch_{out}) \subseteq CH$. There are two exceptions when these rules cannot be applied: interface nodes having only output channels and interface nodes having only input channels.

If the faulty node does not have input channels $(ch_{in} = \varnothing)$, reconfiguration should follow these rules:

- All nodes connected with faulty node *n* perform their actions until output channels of the faulty node become empty.
- The nodes connected with faulty node *n* transit to blocked reading states.
- After timeout the nodes connected with faulty node *n* transit to control state. In this state:
    - actions of the faulty node *n* are transferred to node *n1* which first transits to control state and is connected to output channel of *n*;
    - the channel between *n* and *n1* is destroyed;
    - output channels of *n* are connected to *n1*;
    - the data lost during failure is compensated.
- The nodes which are in control state move to writing or reading states and reconfigured network continues execution.

If the faulty node does not have output channels $(ch_{out} = \varnothing)$, reconfiguration should follow these rules:

- All nodes connected with faulty node *n* perform their actions until input channels of the faulty node become full.
- The nodes connected with faulty node *n* transit to blocked writing states.
- After timeout the nodes connected with faulty node *n* transit to control state. In this state:
    - actions of the faulty node *n* are transferred to node *n1* which first transits to control state and is connected to input channel of *n*;
    - input channels of *n* are connected to *n1*;
    - the data lost during failure is compensated.
- The nodes which are in control state move to writing or reading states and reconfigured network continues execution.

In distributed process network, channel failure is also critical for network execution and can cause a global deadlock. In case of network channel failure, the change of network parameters must follow these rules:

- The nodes connected by faulty channel transit to blocked reading and writing states ns_{br} and ns_{bw}.
- After timeout the nodes connected by faulty channel transit to control state ns_c.
- Creation of a new channel is initiated by the node which first transits to control state.
- The new communication channel is connected to the reading node as its input channel.
- The new communication channel is connected to the writing node as its output channel.
- The data lost during failure are compensated.
- The nodes which are in control state move to writing or reading states and reconfigured network continues execution.

5. Implementing FTPN

In order to analyze the behaviour of process network in case of hardware failure, multi-threaded implementation of FTPN was used. The FIR filter process network was implemented in C# programming language using multiple threads. Separate thread was used for each element of the process network and main program was used for coordination. The failures of network elements were imitated by destroying a thread of node or channel. Multi-threaded implementation of the FTPN was used for FIR filter. The FTPN for the implemented FIR filter is presented in Fig. 3(a).

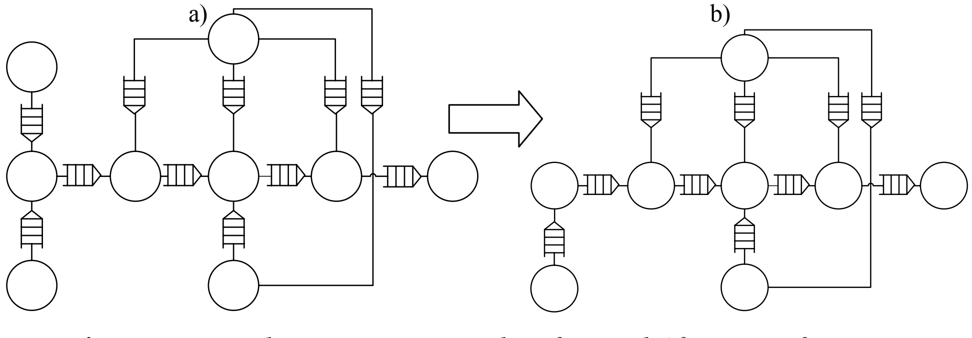

Figure 3. *FIR Filter Process Network Before and After Reconfiguration*

The implementation of FIR filter was used for testing fault handling in case of interface node failure. The interface node *in1* having one output channel was terminated and further execution of the network was observed. As the node *in1* stopped writing data to the channel connecting nodes *in1* and *digp*, the node *digp* consumed all data tokens from this channel and was blocked. After timeout the node moved to control state. In this state the node *digp* checked the state of the channel connecting *in1* and *digp* and state of the node *in1*. Since the node *in1* was not functioning, its operations were transferred to the node *digp*, and the channel between

in1 and *digp* was destroyed. Afterwards, the node *digp* moved to reading state and continued network execution. The FIR filter process network after modification is presented in Fig. 3(b).
Modelling failures and their handling in FTPN demonstrated, that in case of network node failure, the execution of the network can be continued by redistributing operations of the faulty node to the other nodes in the network and data loss can be minimized by reducing the length of channels.

6. Conclusions and Future Work

Process networks and their modifications used for development of digital signal processing systems do not ensure fault handling of the elements of the network. Therefore these process networks are not applicable for development of critical systems. For development of fault tolerant systems, dynamic reconfiguration of these systems is required. Such dynamic reconfiguration is ensured in fault tolerant process network proposed in this work. The main purpose of dynamic reconfiguration is reaction to system execution faults and handling of these faults.
The proposed FTPN uses roll-forward recovery from failures, which reduces requirements for computational and memory resources. Delay, load sharing and redundancy techniques are used for timely fault detection and handling. The possibility of dynamic reconfiguration can be ensured by introducing the control state for elements in process network. The proposed fault tolerant process network is formally specified using LTS, which enables specification of all states and transitions between these states for network nodes and channels.
In order to ensure the correct reallocation of operations from one node to another, these operations must be defined as sequences of atomic actions. Specification of atomic actions ensures that functionality of the process network does not change after reallocating operations of the faulty node to the other nodes in the network.
Currently the algorithms for processing several simultaneous failures are under development. In the future we are planning the implementation of the proposed FTPN in dedicated hardware and/or computer network.

References

Avizienis, A., Laprie, J.-C., Randell, B., Landwehr, C. (2004), *Basic Concepts and Taxonomy of Dependable and Secure Computing*, IEEE Transactions on Dependable and Secure Computing, vol. 1, no.1, pp. 11-33.

Bhattacharya, B., Bhattacharyya, S. (2001), *Parameterized dataflow modeling for DSP systems*, IEEE Transactions on Signal Processing, vol. 49(10), pp. 2408-2421.

Bilsen, G., Engels, M., Lauwereins, R., Peperstraete, J. (1996), *Cycle-static dataflow*, IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 44, pp. 397-408.

Čeponis, J., Kazanavičius, E., Mikuckas, A. (2002), *Design and analysis of DSP systems using Kahn process networks*, Ultragarsas, vol. 45, pp. 43-46.

Čeponis, J., Kazanavičius, E., Mikuckas, A. (2006), *Fault Tolerant Process Networks*, Information Technology And Control, vol. 35, no. 2, pp. 124-130.

Geilen, M., Basten, T. (2004), *Reactive Process Networks*, in Proc. EMSOFT'04, pp. 137-146.

Haim, F., Sen, M., Ko, D.-I., Bhattacharyya, S., Wolf, W. (2006), *Mapping Multimedia Applications onto Configurable Hardware with Parameterized Cyclo-Static Dataflow Graphs*, in Proc. International Conference on Acoustics, Speech, and Signal Processing, pp. 1052-1055.

Hofstee, D., Juurlink, B.H.H. (2002), *Determining the criticality of processes in Kahn process networks for design space exploration*, in Proc. ProRISC 2002, pp. 292-297.

Huth, M. (2005), *Labelled transition systems as a Stone space*, Logical Methods in Computer Science, vol. 1, pp. 1–28.

Kahn, G. (1974), *The semantics of a simple language for parallel programming*, in Proc. IFIP Congress 74, pp. 471-475.

Lee, E., Messerschmitt, D. (1987), *Synchronous data flow*, IEEE Proceedings, vol. 75(9), pp. 1235-1245.

Lee, E., Parks, T. M. (1995), *Dataflow process networks*, IEEE Proceedings, vol. 83(5), pp. 773-798.

Magee, J., Kramer, J. (2006), *Concurrency: State Models and Java Programs*, 2nd Editon, ISBN: 0-470-09355-2, p. 432.

Neuendorffer, S., Lee, E. A. (2004), *Hierarchical reconfiguration of dataflow models*, in Proc. Second ACM-IEEE International Conference on Formal Methods and Models for Codesign , pp. 179-188.

Olson, A.G., Evans, B.L. (2005), *Deadlock detection for distributed process networks*, in Proc. IEEE International Conference on Acoustics, Speech and Signal Processing, vol. 5, pp. 73-76.