Improving Fault-Tolerance in Intelligent Video Surveillance by Monitoring, Diagnosis and Dynamic Reconfiguration

Andreas Doblander[1], Arnold Maier[1], Bernhard Rinner[1] and Helmut Schwabach[2]

[1]Institute for Technical Informatics,
Graz University of Technology, Graz, Austria
{doblander,maier,rinner}@iti.tugraz.at

[2]Video & Safety Technology,
ARC Seibersdorf research, Seibersdorf, Austria
helmut.schwabach@arcs.ac.at

Abstract — *In this paper, we present an approach for improving fault-tolerance and service availability in intelligent video surveillance (IVS) systems. A typical IVS system consists of various intelligent video sensors that combine image sensing with video analysis and network streaming. System monitoring and fault diagnosis followed by appropriate dynamic system reconfiguration mitigate effects of faults and therefore enhance the system's fault-tolerance. The applied monitoring and diagnosis unit (MDU) allows the detection of both node- and system-level faults. Lacking redundant hardware such reconfigurations are established by graceful degradation of the overall application. An optimizer module that performs multi-criterion optimization is used to compute a new degraded system configuration by trading off quality of service (QoS), energy consumption, and service availability. We demonstrate the functionality of our approach by an illustrative example.*

1 Introduction

A typical intelligent video surveillance (IVS) system consists of various intelligent video sensors that combine image sensing with video analysis and network streaming. The design of these processing units allows to yield various parameters of a captured scene and to compress a live video-stream simultaneously. It is, therefore, a distributed system of collaborating intelligent cameras. Typically, the system nodes, i.e. intelligent cameras, are implemented as embedded multi-processor systems operating autonomously.

Typical video analysis algorithms are our target application for traffic surveillance including motion detection, MPEG-4 encoding, tracking of objects, detection of stationary vehicles as well as the calculation of traffic statistics (such as average speed, number of cars etc.). Obviously, quality of service (QoS) is a major concern in video surveillance.

Thus, IVS systems typically contain dedicated QoS-management mechanisms. Furthermore, QoS is closely related to power consumption making power-awareness an important design aspect as well. In recent work [1] we therefore take advantage of QoS-triggered dynamic power management.

Additionally, maintaining a high degree of service availability is also an important design goal for autonomous operation of an embedded distributed system [2]. This work focuses on improving service availability in IVS systems by enhancing its fault-tolerance. System monitoring and fault diagnosis followed by appropriate system reconfiguration mitigate effects of faults. Lacking redundant hardware such reconfigurations are established by graceful degradation of the overall application. Multi-criterion optimization is used to compute a new (degraded) system configuration by trading off QoS, energy consumption, and service availability.

2 Related Work

Typical QoS-parameters in video surveillance are video data quality and its distortions in network transmission (jitter). Further parameters include quality metrics such as image size, data rate or blockiness or the number of frames per second (fps). In case of low energy in parts of the system, service availability and QoS may get seriously affected and degraded. Thus, there is a close link of these three parameters in IVS systems.

In literature, there is an emphasis on investigating the trade-off between energy and QoS. In [3] for instance, the authors investigate the trade-off between image quality and power consumption in wireless video surveillance networks. The work mainly focuses on the evaluation of sophisticated image compression techniques. However, existing implementations lack of comprehensive handling of these three correlating parameters.

Fault-tolerance in embedded real-time systems is often achieved by checkpointing mechanisms. In [4], an adaptive checkpointing algorithm is proposed that also minimizes energy consumption. That is, a schedule is derived that includes the checkpointing task in a way that dynamic voltage scaling is most effective. In our system, however, we do not have the (non-volatile) memory capacity and the computational resources for a checkpointing approach. In general purpose distributed computing it is common to use redundant hardware and employ load sharing techniques to increase fault-tolerance (see, e.g., [5]).

Given the tight cost constraints in the embedded market, however, we cannot afford hardware redundancy but depend on graceful degradation. Similarly to [6] we also distinguish between system-level and application-level fault-tolerance techniques. In this work the component faults, e.g., processor failures, are handled by system-level mechanisms. Whereas, inconsistent observations of multiple cameras are addressed by the application logic. There are also several approaches for integrating fault-tolerance techniques into the middleware layer of distributed real-time and embedded systems. The goal is to shift the trade-off between real-time execution and fault-tolerance from design time to runtime to support the application developer [7]. In our ongoing work we currently have not spent much attention on that issue but we will focus on it in future work.

The field of multi-criterion optimization (MCO) [8] often deals with conflicting objectives. In the given work, MCO is applied for three different objectives (i.e. energy, QoS and fault-tolerance).

3 System Description

The considered video surveillance system is organized as a network of distributed smart cameras [9]. Each node is an embedded multi-processor platform equipped with a CMOS image sensor for video acquisition. A network processor (Intel XScale IXP425) and several DSPs (Texas Instruments TMS320C64x) provide the necessary communication capabilities and computing power for video analysis algorithms. Currently, the prototype node [10] is realized comprising two DSPs and one network processor that serves as the managing unit and connects the camera to an Ethernet network. All three processors are connected by a PCI bus. A VGA image sensor is directly connected to one DSP. Alternative network media such as GSM/GPRS or WLAN are also possible.

Due to the heterogeneous processor hardware the software architecture comprises two major parts. The network processor hosts the so called *SmartCam-Framework* (SC-FW) which is run on top of Linux. As a counterpart on the DSPs the *DSP-Framework* (DSP-FW) is run on top of the DSP/BIOS operating system from Texas Instruments. Communication between framework parts is based on messages and organized in a publish-subscribe architecture. A block diagram of the fundamental software architecture is depicted in Figure 1.

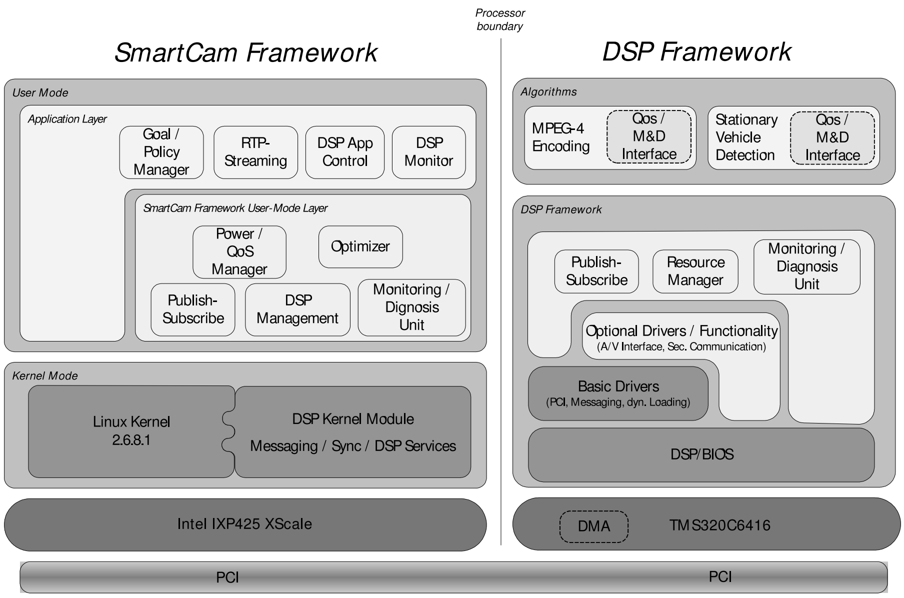

Figure 1: Overview of the software architecture of an intelligent camera node in our distributed surveillance system.

A key functionality in the software framework is the support for task migration. The dynamic loader (DL) provides dynamic linking for the DSPs and, therefore, allows for software reconfiguration at runtime. Tasks can be migrated to another DSP on the same node or to a DSP on a remote node. All algorithms, that is all parts depicted in the top most layer of the *DSP Framework* part on the right side of Figure 1, are dynamically added and removed as needed.

To assure eligible migrations there is a resource manager (RM) that keeps track of important system resources—CPU load, memory usage and DMA channel allocations for each processor, as well as PCI bus and network utilization for the overall node. Using this information it can be determined if there are sufficient resources on the target host for a planned migration.

The actual applications on the smart camera are video analysis and compression algorithms. Currently, four algorithms are considered:

- *Motion detection* to detect motion, camera blackouts and whiteouts.
- *MPEG-4 encoder* for video compression.
- *Stationary vehicle detection*, which can also be used for detecting lost cargo.
- *Traffic statistics* for computing average traffic speed, driving lane utilization and other dynamic traffic parameters.
- *Vehicle Tracking (VT)*, e.g., to track hazardous-cargo vehicles along tunnels.

Unfortunately, the above algorithms are very demanding with respect to computing resources and so it is hardly possible that all run simultaneously. However, not all algorithms are equally important at all times. So it is possible to reduce the QoS of less important components to permit others to be run, too. Different quality levels are distinguished in terms of frame rate and image size. In our system, each algorithm has to support three different QoS-levels Q_i:

- Full quality (Q_1)
- Reduced quality (Q_2)
- Minimum quality (Q_3)

Algorithms provide a dedicated interface for adjusting these three QoS-level settings. A QoS-level switch can be initiated by an algorithm itself or by the system software in reaction to special events. This interface also provides means for exchanging *alive messages* and status information with the *monitoring and diagnosis unit* (MDU).

The functionality of the MDU and the *optimizer* module is the main topic of this paper and is described in Sec. 4 and Sec. 5, respectively.

4 Monitoring and Diagnosis

A key requirement for dynamically reacting to faults and failures is to detect abnormal behavior and isolate affected system components. It is important for the reconfiguration process to have information about which resources are not available after a fault has occurred. In the system described in the previous section it is the responsibility of a special monitoring and diagnosis unit (MDU) to indicate faulty system behavior. A resulting diagnosis is then presented to the configuration manager as seen in Figure 2.

A major concern of this ongoing work is establishing eligible system configurations by multi-criterion optimization in case of faults. Monitoring and diagnosis techniques applied to yield necessary diagnostic information are, therefore, only briefly described.

The IVS system as described in Section 3 can be viewed from two perspectives. First, the system view considers the overall distributed application. That is, all nodes (hardware and system software) including all algorithms running on these nodes. Second, the node view considers only a single node.

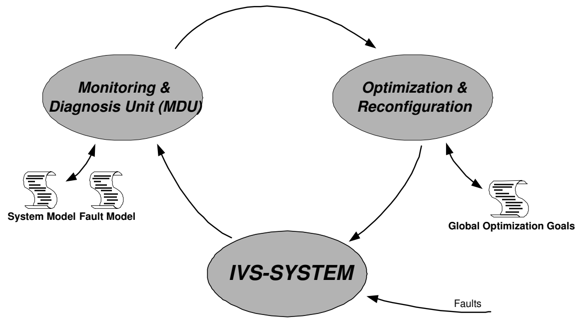

Figure 2: Basic concept of monitoring, diagnosis and reconfiguration in IVS systems.

According to these two different views the monitoring and diagnosis process comprises also a system-level part and a node-level part. For local faults only the MDU of the corresponding node is involved in the monitoring and diagnosis procedure. Node-level faults that are currently diagnosable are:

- *Crash-faults of DSPs*. These faults are detected by a software watchdog. All communication by any algorithm on the DSP resets the watchdog timer. If there is no other communication then polling is employed.
- *Crash-faults of algorithms*. Similar to the procedure above also communication of each algorithm is monitored. A fault is detected if an algorithm is not communicating actively and does not reply to polls on its *M&D interface* as indicated in Figure 1.
- *Memory leaks*. Substantial increase in dynamic memory consumption (heap size) in an unchanged system state indicates a memory leak.

More interesting, however, are the system-level faults. Detection of system-level problems involves multiple nodes sharing monitoring information and exploiting application specific knowledge. Faults of this category that are currently diagnosable are:

- Blackout of a node.
- Value-faults of instances of the stationary vehicle detection algorithm.
- Value-faults of instances of the traffic statistics algorithm.

A node blackout is detected if a node does not respond to periodic alive messages from its neighbors. To detect and diagnose erroneous algorithm behavior, i.e. value-vaults of algorithms, a majority decision is employed. Each node compares its own results to the results of its two nearest neighboring nodes. Because in typical traffic surveillance applications it can be expected that cameras within a dedicated geographical area observe very similar events.

Assuming that all camera nodes are arranged in regular intervals alongside a freeway (or tunnel) observations of neighboring cameras are equal but appear at different times. Of course, this kind of majority decision is only possible if comparable results are available on all three concerned nodes. The distributed diagnosis is performed by a simple communication protocol based on [11] and [12]. This kind of majority decision is only possible if comparable results are available on all three concerned nodes.

5 Optimization and Dynamic Reconfiguration

Generally, determining a new configuration is a mapping of intended functionality onto remaining system resources. The optimizer determines a set of feasible configurations by comparing the algorithms resource requirements and the available resources on the node. By adding "*maximum availability*" to "*minimum energy consumption*" and "*maximum QoS*" we get a total of three different optimization criteria. Thus, the goal of optimization may vary in between these three objectives. As these are conflicting criteria the resulting system configuration will always be subject to a trade-off.

It is the optimizer's task to compute a node configuration $C_{Node} = \{C_{a_i}, \ldots, C_{a_N}\}$, where $C_{a_i} = [q_j, d_k]$ with $q_j \in \{Q_1, \ldots, Q_3\}$ is the QoS, $D = \{d_1, \ldots, d_M\}$ are the DSPs within a node, and $A = \{a_1, \ldots, a_N\}$ are the available algorithms in the system. The set of algorithms currently assigned to a DSP d_k is written as A_{d_k} where $A_{d_k} \subseteq A$.

The objective functions for the described system are

$$E_{Node} = \sum_{a_i \in A_{d_k}} E_{a_i} \tag{1}$$

$$Q_{Node} = \sum_{a_i \in A_{d_k}} V_{a_i}, where\ V_{a_i} = \begin{cases} 3, & q_j = Q_1 \\ 2, & q_j = Q_2 \\ 1, & q_j = Q_3 \end{cases} \tag{2}$$

$$R_{Node} = \sum_{a_i \in A_{d_k}} R_{a_i}. \tag{3}$$

The energy consumed by an algorithm (E_{a_i}) is taken from a look-up table that is usually determined by experiment. Every algorithm has to provide a profile with information on its required resources (e.g., memory usage and DMA channels) and the resulting load on a specific CPU. Therefor, the resource utilization of each algorithm (R_{a_i}) is taken from this profile table. It is assumed that low resource utilization results in more redundancy and therefor increased fault-tolerance.

Given the conflicting objective functions in Equations 1 to 3 the optimizer has to solve the problem

$$\begin{aligned} \min(E_{Node}) \\ \max(Q_{Node}) \\ \min(R_{Node}) \\ subject\ to\ C_{a_i} \in S \end{aligned} \tag{4}$$

where S is the set of all possible configurations whose resource requirements can be satisfied on the node.

If the node's optimizer does not find a feasible solution to the problem in Equation 4 then the two nearest neighbors of the node are also considered in the solution process.

For setting a specific system configuration the optimizer uses the framework's capabilities for dynamic software reconfiguration. Possible mechanisms are changing quality levels of algorithms, migrating algorithms to other nodes or removing an algorithm from the system.

	d_1			d_2		
	Q_1	Q_2	Q_3	Q_1	Q_2	Q_3
a_1	3	2	1	6	4	2
a_2	6	4	2	9	6	3

Table 1: Determined energy consumption.

	d_1			d_2		
	Q_1	Q_2	Q_3	Q_1	Q_2	Q_3
a_1	30	20	10	60	40	20
a_2	60	40	20	90	60	30

Table 2: Determined resource requirements.

6 Illustrative Example

We demonstrate the functionality of the multi-criterion optimizer by the following setup. We consider two DSPs (d_1, d_2) with a total of four algorithms ($a_1, \ldots, a_4$). Only a_1 and a_2 are running on the observed node. The system policy forces that algorithm a_1 is run at quality level Q_1. Algorithm a_2 may run either at level Q_1 or Q_2.

Energy consumption and resource requirements are determined by profiling experiments and are summarized in Table 1 and Table 2, respectively. Maximum available resource for both DSPs is limited to 100. Equation 5 and Equation 6 define all possible configurations for algorithm a_1 and algorithm a_2, respectively.

$$C_{a_1} = \{c_{a_1,1}, c_{a_1,2}\} = \{[Q_1, d_1], [Q_1, d_2]\} \tag{5}$$
$$C_{a_2} = \{c_{a_2,1}, c_{a_2,2}, c_{a_2,3}, c_{a_2,4}\} = \{[Q_1, d_1], [Q_1, d_2], [Q_2, d_1], [Q_2, d_2]\} \tag{6}$$

Given the above resource limitation of the two DSPs, the set of feasible configurations is defined as

$$C = \{C_1, \ldots, C_8\} = \{\{c_{a_1,1}, c_{a_2,1}\}, \{c_{a_1,1}, c_{a_2,2}\}, \{c_{a_1,1}, c_{a_2,3}\}, \{c_{a_1,1}, c_{a_2,4}\}, \tag{7}$$
$$\{c_{a_1,2}, c_{a_2,1}\}, \{c_{a_1,2}, c_{a_2,2}\}, \{c_{a_1,2}, c_{a_2,3}\}, \{c_{a_1,2}, c_{a_2,4}\}\}. \tag{8}$$

For each of the above configurations C_i the values of the objective functions in Equations 1– 3 can now be calculated.

$$E_C = \{E_{C_1}, \ldots, E_{C_8}\} = \{9, 12, 7, 9, 12, 15, 10, 12\} \tag{9}$$
$$R_C = \{R_{C_1}, \ldots, R_{C_8}\} = \{90, 120, 70, 90, 120, 150, 100, 120\} \tag{10}$$
$$Q_C = \{Q_{C_1}, \ldots, Q_{C_8}\} = \{6, 6, 5, 5, 6, 6, 5, 5\} \tag{11}$$

From Equation 4 together with Equations 9– 11 it can easily be seen that C_3 is optimal with respect to energy consumption and redundancy (i.e. fault-tolerance), but C_3 is not optimal concerning overall quality. As there is no configuration that is optimal in all respects different global policies are employed to choose a final solution. If quality was most important C_1 would be the best configuration selected by the optimizer.

7 Conclusion

In this work we investigate a method for improving fault-tolerance and service availability in intelligent video surveillance (IVS) systems. Therefore, capabilities for monitoring, diagnosis and dynamic reconfiguration are added to the system. In case of faults a multi-criterion optimizer determines a new degraded system configuration maintaining as much functionality as possible. In an illustrative example we demonstrate the functionality of our approach.

Future work includes the evaluation of different sizes of the involved neighbored nodes for the case if a single node does not deliver a feasible solution for the optimal configuration. Especially for time-sensitive and safety-critical applications like traffic surveillance it is important to guarantee well defined time bounds also for a distributed implementation. Thus, we also intend to extend our approach to include means for achieving bounded detection latency and bounded recovery time. Furthermore, we aim in a better integration of the fault-tolerance mechanism into our software framework (i.e. middleware) to make them less dependent on a specific application and support application development.

References

[1] A. Maier, B. Rinner, H. Schwabach, and T.Trathnigg. Combined Management of Power- and Quality of Service in Distributed Embedded Video Surveillance Systems. In *Proceedings of the First Workshop on Power-Aware Real-Time Computing, Pisa, Italy*, 2004.

[2] J.O. Kephart and D.M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, pages 41–50, 2003.

[3] C.F. Chiasserini and E. Magli. Energy Consumption and Image Quality in Wireless Video-Surveillance Networks. In *Proceedings of the 13th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, 2002.

[4] Y. Zang and K. Chakrabarty. Energy-Aware Adaptive Checkpointing in Embedded Real-Time Systems. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'03)*, 2003.

[5] L. Lundberg, D. Hggander, K. Klonowska, and C. Svahnberg. Recovery Schemes for High Availability and High Performance Distributed Real-Time Computing. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)*, 2003.

[6] J. Haines, V. Lakamraju, I. Koren, and C.M. Krishna. Application-Level Fault Tolerance as a Complement to System-Level Fault Tolerance. *Journal of Supercomputing (Kluwer)*, pages 53–68, 2000.

[7] T. Bracewell and P. Narasimhan. A Middleware for Dependable Distributed Real-Time Systems. In *Proceedings of the Joint Systems and Software Engineering Symposium*, 2003.

[8] C. Coello Coello. A Short Tutorial on Evolutionary Multiobjective Optimization. In *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*, 2001.

[9] W. Wolf, B. Ozer, and T. Lv. Smart Cameras as Embedded Systems. *IEEE Computer*, pages 48–53, 2002.

[10] M. Bramberger, J. Brunner, B. Rinner, and H. Schwabach. Real-Time Video Analysis on an Embedded Smart Camera for Traffic Surveillance. In *Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium, Toronto, Canada*, 2004.

[11] J. G. Kuhl and S. M. Reddy. Fault-Diagnosis in Fully Distributed Systems. In *Proceedings of the 11th IEEE International Symposium on Fault-Tolerant Computing (FTCS-11)*, 1981.

[12] A. Subbiah and D. M. Blough. Distributed Diagnosis in Dynamic Fault Environments. *IEEE Trans. on Parallel and Distributed Systems*, pages 453–467, 2004.