Memory Concepts for Enabling Adaptivity in Distributed Embedded Systems

Philipp Schleiss, Marc Zeller, Gereon Weiss
Fraunhofer ESK, Munich, Germany
{name.surname}@esk.fraunhofer.de

***Abstract*—Establishing cost and resource efficient dependability through means of adaptivity in safety-critical distributed embedded systems is a strenuous endeavour, as the varying requirements on resilience, control and efficiency across domains prohibits a single solution to suit all needs. To assist the process of determining a safe and efficient system architecture with satisfactory precision, this work exemplifies the importance of differentiation by only addressing distributed embedded systems that perform multiple functions with alternating levels of criticality. Further, they do not require full fail-operational behaviour, thus allowing to sacrifice less important functions in the pursuit of preserving safety. Herein, a dynamic instantiation and graceful degradation strategy is developed to subsequently study its effect on cost when implemented in conjunction with execute-in-place (NOR-flash) or block-addressable (NAND-flash) memory concepts. Even though NOR-flash is generally considered to be a better candidate for such systems, this qualitative research produces evidence that NAND-flash memory concepts are likely to financially outperform traditional architectures when considering adaptivity.**

I. Introduction

In light of emerging technological advances, such as, multi-core CPUs, higher network bandwidth, and smart sensors, an opportunity arises to ease the path towards adaptivity in safety-critical distributed embedded systems. A promising step on this route is the replacement of single purpose processing units in exchange for platforms capable of controlling multiple arbitrary functions. Hereby, research on mixed-criticality systems forms a cornerstone in loosening this rigidity while paying special respect to the particularly stringent temporal and spatial isolation constraints found in the safety domain [1]. In turn, this gain in flexibility lays the foundation for more sophisticated forms of adaptivity.

Looking beyond the myriad of challenges occurring in the pursuit of attaining a reliable level of temporal and spatial isolation [2], one of the first questions that comes to mind is how the newly gained flexibility can contribute to system robustness in a more resource efficient manner. Despite this, and even though unit cost majorly influences mass production processes, as for instance seen in the automotive industry, guidance on economically designing such resilient distributed embedded systems is sparse. One notable facet of this economic efficiency perspective is the arrangement and utilisation of memory, which substantially affects overall cost. As such, this work analyses memory concepts based on NAND- and NOR-flash, the latter often being favoured for embedded systems due to is *execute-in-place (XIP)* capabilities [3]. Aside hereof, the contradicting emphasis on resilience, efficiency, and control across domains questions the informative value of a generalised blanket statement on financial efficiency. Consequently, the scope of systems is restricted to a set of not severely contravening constraints, which are expected to benefit most from this approach. More precisely, systems that necessitate full fail-operational behaviour, often make use of triple redundancy concepts, only facilitate one function, or associate cost as an ancillary influence, are disregarded in favour of the multitude of other use cases lying in between these extremes. These may only require a small set of functionalities to maintain a safe state or only have to remain operational for a short period of time after failure until a halt can occur safely. In this confined band of systems the potential to increase efficiency through introducing adaptivity and in particular exploiting the diversity of criticality characteristics is most auspicious and thus deserves further investigation.

Therefore, this work first of all illustrates the envisioned form of adaptivity by outlining challenges and methods belonging to its central principle of dynamic application reallocation (see Sec. II), and based thereon derives a degradation strategy to safeguard control of the most critical functions even after severe impairment (see Sec. III). Subsequently, the effects of this scheme on memory are determined (see Sec. IV), compared qualitatively (see Sec. V), and discussed (see Sec. VI). Thereafter, the results are concluded, finally leading to a brief outlook (see Sec. VII).

II. Principles of Resilient Failure Handling

To be able to gracefully handle device failures, a method for transferring function control, which was previously in the custody of the lost computational units, onto another physical platform is indispensable to safeguard correct operations [4]. Based on this software replacement technique, more elaborate transactional schemes can then transform the overall software deployment through isolated changes on an application per application basis, thus allowing the safe transition to a new robust configuration, which would have otherwise been inconceivable.

A. General Challenges

To ensure a safe migration of applications a multitude of aspects must be considered meticulously. First of all, allowing software relocation undermines the stringent design-time verification and validation process, because schedulability analysis, symbol resolution, CPU architecture compatibility, memory allocation, and temporal and spatial isolation questions must be addresses at run-time. In addition, finding a valid deployment in a restricted time span while considering multiple partially

The paper was presented at APRES 2014.

opposing mandatory and desirable constraints is non-trivial [4]. Then again, to address applications independently of their physical location a flexible and deterministic communication system is inevitable. Furthermore, due to current technical bounds on CPU clock speed, a device can only host multiple applications by using numerous CPUs or CPU cores, which in turn raises further isolation concerns [5]. The use of identical hardware platforms may ease the attainability of some of these goals but in turn would not embrace the diversity of distributed embedded systems. Regardless, for the remainder of this work these challenges are considered to be sufficiently solvable even in heterogeneous settings. As such, the focus is laid on factors affecting memory utilisation when implementing application mobility.

B. Memory Alignment

In view of varying application lengths and flat address spaces, an algorithm is needed to realign the entire memory layout or to fragment new applications to fit them into the freed areas. With respect to temporal and spatial isolation the simultaneous realignment of all applications in memory at run-time is questionable, as the time-frame in which an adaption can take place without breaching any deadlines is narrow and thus necessitates a fast and highly predicable method. Alternatively, an offline based reconfiguration approach can mitigate these impediments and allows the use of established tests, such as inspecting functions before productive use to verify correct linking and symbol resolution [6]. Then again, the interval in which a safe state for offline reconfiguration can be reached may depend on a human operator's decisions and is therefore not always reliably quantifiable, thus leading to a more conservative risk assessment. On the other hand, a fragmented replacement of applications at run-time is viable but depends on deterministic relocation and pointer adjustment techniques.

C. Criticality of Applications

Moreover, mixed-criticality systems consist of applications that differ in their contribution to safety, and as such span the spectrum from purely comfort and multimedia features to time- and life-critical functions [7]. After failure, it is crucial to restore critical software before any deadlines are breached, whereas the delayed reinstatement of less demanding applications is sufficient.

A straightforward approach to incorporate this time sensitivity into an application deployment strategy is to set up at least one additional application instance that independently calculates its state on a different hardware platform to be able to seamlessly take over operation after failure of the primary instance. Starting from this *hot-standby* approach, there are many techniques aiming at improving efficiency, as for instance, sharing states periodically instead of performing all calculations twice. Finding a sufficiently consistent and efficient method for sharing a synchronous state between multiple instances is however an application specific problem, which cannot be addressed from an overall architectural standpoint. Moreover, applying a hot-standby technique for less urgent fail-over demands is wasteful, as resources are unnecessarily occupied by standby instances instead of only initialising these instances after failure (*cold-standby*).

III. Redundancy & Degradation Strategy

In case a hardware platform hosting a safety-critical application fails, control is transferred to the backup instance. From this point on, the initial hot-standby redundancy is void, thus inducing a frail state incapable of withstanding another failure. Subsequently, the operator or system would typically initiate a safe halt at earliest convenience to prevent a catastrophic failure. Supposing that the system-wide remaining resources, such as, CPU, network bandwidth, and memory, allow to host a newly created hot-standby instance, this approach is unreasonably harsh. Furthermore, even if the overall capacity is non-sufficient to host all applications, the envisioned system shall unschedule less critical software in order to recreate redundancy for the most essential control functions, thus allowing operations to continue directly or after a short reconfiguration phase in exchange for taking a minor inconvenience into account. For this, only executable code and static data are regarded, whereas larger data sources and state persistence are discarded from the scope of this work, as they are respectively considered to be fetched directly from globally addressable external data sources or managed by intermediate use case specific software. In sum, these principles of degradation and recreation of redundancy define the basic behaviour of distributed embedded systems eligible for implementing the subsequent memory concepts.

IV. Robust Storage Concepts

Safety-critical embedded systems have specific demands on reliability, performance, power consumption, robustness to environmental influences, and cost. Regarding storage hardware more closely, non-volatile execute-in-place (XIP) storage circumvents the need for additional RAM by supporting byte-wise fetching of instructions. Such XIP memory is commonly based on NOR-flash with high read and low write speeds [8], or on erasable programmable read-only memory (EPROM), which can only be reprogrammed as an entire unit. On the contrary, block-addressable memory, such as NAND-flash, depends on shadowing or paging techniques that map executable code into byte-addressable memory. In return, this yields a higher bit-density and an up to 5-times lower cost-per-byte [8]. To evaluate the cost-efficiency, substantially different storage concepts aimed at stressing the particular advantages of each storage type are presented to further derive their memory utilisation characteristics.

A. Redundant Non-Volatile XIP Memory (Concept A)

Increasing resilience of a system holding code in XIP storage and software state in RAM is easily achieved by replicating every application onto a second device and scheduling all hot-standby instances on it. However, the inability to differentiate between cold- and hot-standby software with respect to storage utilisation leads to a twofold storage demand that further multiplies when taking multiple types of hardware platforms into account, as all versions of an application for every CPU-architecture must be stored at least twice.

B. Non-Volatile RAID5 XIP Memory (Concept B)

In search of memory conserving resilience, the redundancy, performance, and cost benefits attained through the use of

RAID5 in information systems motivate to obtain similar gains in the embedded domain [9]. In contrast to these RAID5 configurations, which focus on data redundancy, this approach aims at redundantly storing directly executable code. Therefore, instead of splitting an application according to a fixed bit-length amongst multiple devices, the entire application is placed into memory in an executable format. Furthermore, the parity block on one device is based on the bit-values found in the same absolute memory addresses of the other device, thus leading to a memory overhead of only $1/n$ (n: number of devices) and as such proportionally decreases with rising unit numbers. Despite this, the probability of information loss increases with the amount of platforms, limiting the scalability or requiring parity blocks on multiple devices. Additionally, memory is not utilised to the same extent on all devices and therefore an additional blank memory block is wasted, as displayed in a simplified example only including one type of CPU architecture in Fig. 1. Beyond this, placing a hardware platform with a second type of CPU architecture into this overall system setup doubles memory demands. Then again, as the memory block of any application can be recalculated after the loss of a device by contacting all remaining platforms, there is no eminent necessity to additionally hold cold-standby instances in an executable format.

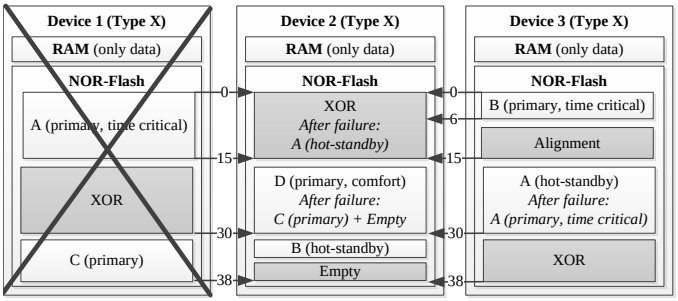


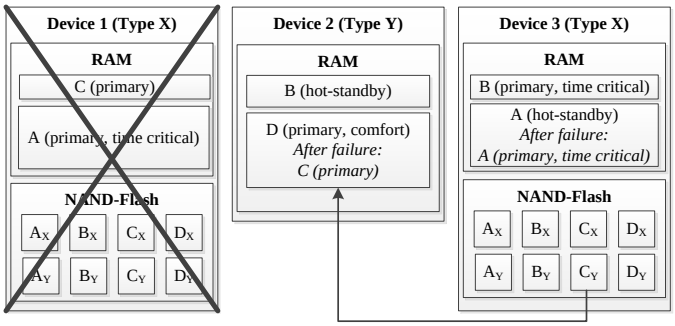
Fig. 1. RAID5 XIP Memory Concept

C. Page-Addressable Memory & Shadowing (Concept C)

The complexity of reaching non-wasteful resilience with non-volatile XIP-memory concepts sways to explore more simplistic ways of achieving cost efficiency without solely focusing on memory utilisation. For this, all applications compiled for every CPU architecture are placed into cheaper NAND-flash on all computation devices. In addition, the software that shall be executed on a certain platform must be shadowed into RAM to allow the efficient fetching of instructions, whereas cold-standby instances remain uninitialised. In sum, the non-volatile storage waste ratio is negatively affected through every additional application, whereas the absolute block-addressable consumption also rises with an increase in devices. Moreover, platform diversity simply multiplies memory demands.

D. Central Repositories & Partial Paging (Concept D)

With the advance of Ethernet in safety-critical embedded systems [10], the restraint to persistently store applications locally fades, as the bandwidth of the network may surpass local storage transfer rates. Therefore, the number of devices equipped with NAND-flash in concept C can be scaled down as long as compliance with an essential level of resilience is ensured, hence only necessitating a few central devices acting as application repositories (see Fig. 2). In addition, primary application instances that are only active in certain situations and have less strict deadlines can be removed from RAM and loaded on demand. Put more generally, using a deterministic paging algorithm [11] for software with less stringent deadlines can immensely lower minimal RAM limits while not endangering safety.



Note: Memory size of NAND-flash is not to scale
Fig. 2. RAM Shadowing with Page-Addressable Memory

V. Comparison of Concepts

To reach a sound statement on cost-efficiency, a decision process grounded on the comparison of concepts according to their characteristics most influential to cost is mandatory. Therefore, the dimensions of cost-per-byte, storage overhead, number of initialised applications, and simplicity of attaining adaptivity are presumed to sufficiently cover this prerequisite, and are in consequence further examined.

Next to the obvious **cost-per-byte** benefits of NAND-flash, unsurprisingly, the full replication of all applications (C) in the antagonistic dimension of **storage overhead** performs worst, followed by the duplication of software components (A), whereas the RAID5 concepts (B) even lessens relative wastefulness with an increasing number of devices. Likewise, the establishment of repositories that deterministically transfer applications over a network connection from a few dedicated devices with large NAND-flash to the target devices (D) also decreases storage overhead with similar magnitude.

Taking a closer look at memory used for holding **initialised applications**, the duplication of all applications into XIP-memory as a measure to prevent loss of programme code (A) exhibits poor efficiency. In contrast, RAID5 approaches (B) can profit from not maintaining initialised cold-standby instances. However, accounting for the slow write speeds of NOR-flash and the compulsion to contact all remaining nodes for reconstruction prolongs the reconstruction of operational instances, thus reducing their viability for cold-standby. Opposed hereto, the differentiation between cold- and hot-standby systems can be leveraged more fruitfully when applications are placed into volatile memory (C & D), as the manifold higher write speed of RAM and therewith intertwined software start-up speed allow more programmes to be capable of cold-standby. Then again, fetching applications from a remote repository (D) causes a delay, which is however marginalisable with higher network bandwidth. Moreover, the introduction of paging as an optimisation technique for less time critical application (D) once again diminishes the amount of initialised applications that must permanently remain executable, thus outweighing the previous deficit.

Finally, **simplicity** is elementary in the creation of safe systems. When looking beyond the base challenges of deterministic dynamic application loading, the additional complexity of managing code redundancy and restoring a redundant state after failure in RAID5 architectures (B) is most troublesome and can therefore severely complicate large transactional reconfiguration schemes. Surprisingly, also the pure duplication of applications (A) may distort the ease of realisation when multiple platform types are involved. For instance, the question arises if code redundancy of an application is required for a device type that only exists in the system once. Compared hereto, having a local copy of all applications for every platform variant (C) simplifies this process considerably and further fosters the reuse of methods already used for loading cold backup instances. On the contrary, the optimisation techniques of application repositories and deterministic paging (D) once again complicate system design.

In a nutshell, the relative performance of each memory concept in the four discussed dimensions is summarised under consideration of systems containing one and multiple ([]) CPU architectures in table I.

	Memory Type	Redundancy	Instruction Access	Cost per Byte	Storage Overhead	# Initialised Applications	Simplicity
A	NOR	Double	XIP	–	– [– –]	–	+ [–]
B	NOR	RAID5	XIP	–	++ [–]	0	– –
C	NAND	Full	Shadowing	+	– – [– –]	+	++
D	NAND	Repository	part. Paging	+	++ [–]	++	0

TABLE I. MEMORY CONCEPT EVALUATION MATRIX

VI. DISCUSSION

Regarded from a memory utilisation perspective, providing full application redundancy with block-addressable storage is wasteful, as storage is manifold occupied with identical applications, and RAM is needed to compensate for non-existent XIP characteristics. However, in aspiration of determining cost-efficiency with respect to application mobility, the inferior cost-per-byte ratio of NOR-flash places the less memory-efficient NAND-flash-based concept back within reach. Beyond this, application repositories can reduce this wastefulness similarly to more intricate RAID5 approaches. When factoring in paging of uncritical applications and uninitialised cold-standby optimisations, even the overall RAM demand can be lowered considerably, thus accentuating their virtue with respect to cost.

Moreover, simplicity is a key to acceptance of adaptivity in the safety domain, which in turn allows to break away from single purpose units, federated architectures and dedicated standby device, thus positively affecting overall cost in a superior manner by substantially reducing a system's bill of materials. Besides this, the existence of large and cheap storage may enable to implement data-intensive use cases that were prior deemed uneconomical. Alternatively, even if dynamic loading is suspected to be too laborious or unsafe for certain use cases, the availability of large memory capacities may instead encourage to store multiple pre-verified statically linked images and therethrough replace the entire executable memory content after reaching a safe halt.

VII. CONCLUSION & OUTLOOK

On the journey towards leveraging adaptivity as an instrument to improve safety, the free relocation of applications is an encouraging approach. To address this concept of application mobility from ground up, a critical reconsideration of current memory architectures is one of the first steps. As such, the work demonstrates that block-addressable NAND-flash in combination with RAM shadowing is favourable when implementing software mobility in safety-critical distributed embedded systems, because complexity is kept at bay while cost is not adversely affected by the more wasteful storage utilisation. Furthermore, application repositories, paging for uncritical applications, and uninitialised cold-standby instances pose as viable alternatives for unit cost driven domains through their fair exchange of simplicity for cost-effectiveness.

To evaluate the practicability of approaches based on block-addressable memory, a proof of concept implementation for dynamically loading applications is the next imminent challenge. Beyond this, and in anticipation of rigorous safety requirements, a thorough analysis and dutiful integration of run-time validation and verification methods is crucial to create acceptance and justify the intrusion on well-established safety design practices caused by this type of adaptivity.

ACKNOWLEDGMENT

This work was funded by the European Commission within the 7th Framework Programme as part of the SafeAdapt project under grant number 608945.

REFERENCES

[1] M. Mollison, J. Erickson, J. Anderson, S. Baruah, and J. Scoredos, "Mixed-criticality real-time scheduling for multicore systems," in *2010 IEEE 10th Int. Conference on Computer and Information Technology (CIT)*, 2010, pp. 1864–1871.

[2] J. Rushby, "Partitioning in avionics architectures: Requirements, mechanisms, and assurance," NASA, Langley, Tech. Rep., 1999.

[3] C. Park, J. Seo, D. Seo, S. Kim, and B. Kim, "Cost-efficient memory architecture design of NAND flash memory embedded systems," in *Proc. of 21st Int. Conference on Computer Design*, 2003, pp. 474–480.

[4] M. Zeller and C. Prehofer, "Timing constraints for runtime adaptation in real-time, networked embedded systems," in *Proc. of ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2012, pp. 73–82.

[5] B. Brandenburg and J. Anderson, "Integrating hard/soft real-time tasks and best-effort jobs on multiprocessors," in *Proc. of 19th Euromicro Conference on Real-Time Systems (ECRTS)*, 2007, pp. 61–70.

[6] N. Kajtazovic, C. Preschern, and C. Kreiner, "A component-based dynamic link support for safety-critical embedded systems," in *Proc. of 20th IEEE Int. Conference and Workshops on the Engineering of Computer Based Systems (ECBS)*, 2013, pp. 92–99.

[7] D. de Niz, K. Lakshmanan, and R. Rajkumar, "On the scheduling of mixed-criticality real-time task sets," in *Proc. of 30th IEEE Real-Time Systems Symposium (RTSS)*, 2009, pp. 291–300.

[8] Y.-H. Chang, J.-H. Lin, J.-W. Hsieh, and T.-W. Kuo, "A strategy to emulate NOR flash with NAND flash," *Trans. Storage*, vol. 6, no. 2, pp. 5:1–5:23, 2010.

[9] M. Zeller, S. Grosse, D. Eilers, and R. Knorr, "Fail-safe data management in self-healing automotive systems," in *Proc. of 6th Int. Conference on Autonomic and Autonomous Systems (ICAS)*, 2010, pp. 24–29.

[10] M. Jakovljevic and A. Ademaj, "Ethernet protocol services for critical embedded systems applications," in *Proc. of 29th IEEE/AIAA Digital Avionics Systems Conference (DASC)*, 2010, pp. 5.B.3–1 – 5.B.3–10.

[11] K. Cho, K.-S. We, C.-G. Lee, and K. Kim, "Using NAND flash memory for executing large volume real-time programs in automotive embedded systems," in *Proc. of 10th ACM Int. Conference on Embedded Software (EMSOFT)*, 2010, pp. 159–168.