Engineering Cross-Layer Fault Tolerance in Many-Core Systems

Rem Gensh, Alexander Romanovsky, Alex Yakovlev

Centre for Software Reliability
Newcastle University
Newcastle upon Tyne, UK
{r.gensh, alexander.romanovsky, alex.yakovlev}@newcastle.ac.uk

Abstract. Engineering modern many-core systems is a challenging task because of their scale and complexity. We cannot focus on ensuring their dependability without understanding its interplay with performance and energy consumption. This calls for developing new structuring mechanisms that step away from the traditional ways systems are developed (such as strict layering, strong encapsulation, abstractions, hiding). The paper reports on the initial steps of a PhD work focusing on development methods and tools for architecting cross-layer fault tolerance in many-core systems in which error detection and error recovery are applied at several system layers in a concerted coordinated fashion to ensure the overall system efficiency.

Keywords: Error detection, error recovery, performance, power consumption, abstractions, encapsulation

1 Introduction

Fault tolerance [1] is the means of dependability, allowing us to prevent system failures in the presence of faults. To achieve this after an error is detected in a component of a computer system (e.g. hardware, operating system or software), an error recovery mechanism returns the system to the full or reduced system functionality.

Term cross-layer interaction [2] was introduced to refer to the idea that it is better suited to ensure system efficiency with respect to various non-functional characteristics by reasoning about system layers together, rather than by completely abstracting the functionality of individual layers and trying to improve the efficiency of each of them in isolation. The examples of such characteristics are resource usage, reliability, performance and power consumption.

Many-core systems are likely to become the predominant type of the architectures used in the future. According to [3] the number and variety of cores will be continually increasing. One of the challenges in the area is that there is a need to understand the trade-off between reliability and energy-consumption, as more energy is necessary to support the operation of redundant cores. Another challenge is that fault tolerance is typically developed on one system layer even though these systems are always built as multilayer architectures.

Developing a useful and general support to help in engineering cross-layer fault tolerance is a challenge. The main problem is that in this work one needs to develop techniques that assist in breaking the conventional way the abstractions are created. This paper reports on the initial work in the PhD study conducted by the first author in the area of developing methods and tools to support engineering of cross-layer fault tolerance for the many-core systems.

2 TCP/IP as a motivating example

The TCP/IP stack (see Fig. 1) provides an excellent example of cross-layer fault tolerance applied to ensure fault tolerance and improved performance. All layers of the TCP/IP stack participate in error detection and recovery in a concerted fashion.

The Link layer, the lowest layer of the TCP/IP stack, is used to transmit the packets between the Internet layer interfaces of two nodes inside a local network segment. The Transport layer provides host-to-host communication for the Application layer. The Application layer supports data exchange between processes on different hosts over the network connection supported by the lower layers.

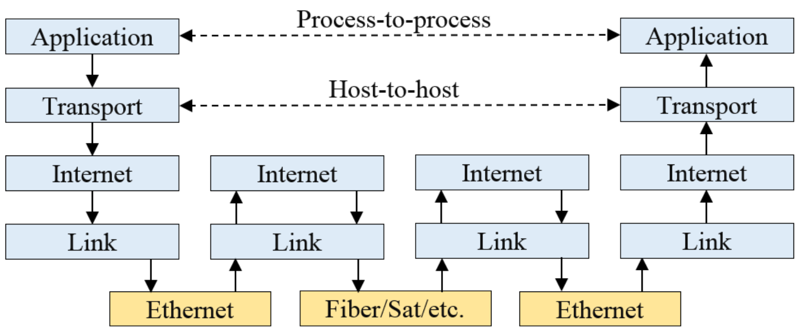

Fig. 1. TCP/IP Stack

TCP provides reliable packets transmission, even though the packets may be lost, corrupted or delivered out-of-order. At the Link layer, the Ethernet frame contains a CRC-32 checksum: a received frame with an incorrect checksum is discarded. The main protocol at the Internet layer is IP (Internet Protocol), which has two implementations, IPv4 and IPv6. The header of the IPv4 packet is protected by CRC-16 checksum. The IP packets with wrong checksums are dropped by the receiver. The IPv6 header does not contain a checksum, assuming that Link layer provides an adequate error detection. The UDP and TCP packets of the Transport Layer have CRC-16 checksums, which protect the payload and addressing information.

TCP sends Acknowledgement to the sender to confirm the correct receipt or Negative Acknowledgement if the packet checksum is incorrect. In the latter case, the Automatic Repeat reQuest (ARQ) method is used to retransmit the corrupted packet. If the sender receives neither Acknowledgement nor Negative acknowledgement by timeout, it resends the packet. Such situation can happen when the packet is lost or rejected by the lower layers due to incorrect checksum. In addition, a TCP packet contains a sequence number, which allows the receiver to discard duplicate packets and sequence reordered packets. This, in particular, shows that the errors of the lower layers are detected and recovered by concerted efforts at several layers.

At the Application layer, the developer can choose an appropriate Transport layer: either the connection oriented and reliable TCP or the connectionless UDP. The developer's choice between reliable data delivery and data delivery in time depends on the application requirements. If UDP was chosen, than it might be necessary to implement error detection and error recovery at the application layer by adding redundant data e.g. status code or encryption.

To conclude, TCP/IP is a useful example of how fault tolerance can be applied in coordination at several system layers. This was done to ensure its efficiency and flexibility. In our opinion the main factor contributing to the success of this protocol is the way the fault tolerance was designed and engineered.

3 Many-core systems

Computer systems with tens, hundreds or thousands processor cores are called many-core systems [4], whereas multi-core systems have typically only 2-8 cores. It is expected that these systems will replace multi-core systems in the near future and that they will become widely used in the safety-critical applications. Many-core architecture uses low performance small cores each of which alone is less productive than a large core, however hundred or thousand of small cores deliver better performance than ten large cores. Even though we can expect that the throughput will increase with the increasing number of cores, the performance growth is restricted by the percentage of serial code in the application (Amdahl's Law). Engineering of the efficient many-core systems is now an area of active research focusing on developing scalable methods for structuring complex many-core applications and for efficient parallelization at the OS and hardware layers.

Another challenge in developing these systems is ensuring their fault tolerance. The first problem is that when voltage and frequency scaling is applied to reduce power consumption the reliability is affected when near-threshold values are used. So we need to understand the interplay between energy and reliability. Moreover the modern semiconductors are more vulnerable to faults or negative effects like ageing and variation due to their extra small sizes. Many-core systems can provide redundancy to deal with these problems (e.g. some cores can be used to provide error detection and error recovery for other cores). Our analysis shows that in many-core systems fault tolerance is typically applied at the individual layers such as OS, application, communication middleware, memory, etc.

Ensuring high performance, low energy consumption, efficient resource utilisation and high reliability, as well as understanding their interplays are the main challenges for all types of many-core systems ranging from the large-scale systems, like data centres to the small-scale systems, like mobile devices.

4 Layered fault tolerance

Computer-based systems are prone to faults at different layers of the system stack, starting from circuitry degradation at the hardware layer to the bugs in the application source code. In designing large systems substantial efforts are being made to mitigate the effects of errors caused by faults at all layers of the system stack. Traditionally, the errors are handled at the layers where they are detected. Such an approach, reducing the complexity of system engineering, is very convenient for the developers and for the teams of developers as it simplifies system composition, reuse, maintenance and modification. This situation illustrates the predominance of convenience over the system efficiency in run time.

Let us look first into several examples of how fault tolerance of system components is typically ensured. Triple modular redundancy is a form of N-modular redundancy when three components perform the same operation and a single output is produced by a majority-voting system. The recovery block [5] works with several implementations of the same algorithm: after executing the primary variant, an acceptance test verifies the results. If the acceptance test fails, the system is rolled back and the secondary variant is tried. Eventually, either a variant passes the test or an exception handler is invoked. The N-version programming [6] is an approach aiming to reduce the probability of software faults by developing two or more functionally equivalent program versions independently in accordance with the same initial specification. These versions are executed concurrently and a special voting algorithm choses the correct output.

The two typical approaches used to ensuring fault tolerance at several layers are action nesting and extending component interfaces with exceptions. The best examples of the former are exception handling and nested ACID (Atomicity, Consistency, Isolation, Durability) transactions. The latter are best represented by F. Cristian's approach to providing recovery for modular software [7] and the idealised fault tolerance component pattern [8]. Even though these techniques support layered system structuring for fault tolerance they do not support concerted cross-layer fault tolerance at multiple layers when the decision to apply error detection and recovery is made for all layers together.

The substantial disadvantage of the layered approach is that the system layers are considered separately. Under such circumstances, it is impossible to adjust the layers in order to achieve optimal system operation in terms of performance, energy efficiency or resource utilization. Unnecessary error corrections are possible when the upper layer cannot specify the required quality of service of the bottom layer.

For example, let us consider a many-core system where the fault rate of one core is significantly larger than the rate of another core. When an error is detected, the error recovery could be achieved by re-executing the calculations on the same core making it slower. If there were a special cross-layer mechanism, which can make a decision that under some fault rate value, hardware layer error recovery should be applied, but after exceeding this value, it is necessary to inform OS about the faulty core, than the system fault tolerance as a whole would be more efficient. In the latter case, OS would be capable to hide the faulty core from the applications for some time.

The optimality and the effectiveness of the system fault tolerance could be achieved only when all the layers of the system are considered together. This is unfeasible when the strict layered approach is used.

5 Cross-layer fault tolerance

The cross-layer fault tolerance assumes that fault tolerance mechanisms are distributed among all layers of the system stack and designed together (see Fig. 2). The final decision is made according to the whole system state rather than to the states of the individual layers separately.

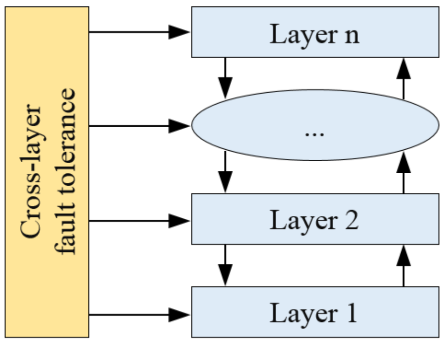

Fig. 2. Cross-layer fault tolerance

The Cross-Layer Reliability Visioning Study [2] proposes that it is necessary to use a cross-layer, full-system-design approach to reliability. The authors argue that in a cross-layer reliable system the entire system stack needs to collaborate in order to recover the errors and tolerate variations. This will be achieved because the relevant information about the system state is shared across the layers. In addition, the application domain of the system should always be taken into account, since different domains have various reliability requirements.

Study [2] introduces the cross-layer approach to the reliable system design, forecasting that the electronics industry is about to approach two inflection points that require drastic changes in integrated circuits design. The first point is reliability and predictability. In the fabrication technologies less than 65nm gate leakage became a serious problem that led to reliability deterioration. This will push the designers to alter the assumptions that semiconductors and other microelectronic elements will operate without fails during the whole system lifetime. The second point is energy consumption, which is a crucial issue for contemporary computer systems. Paper [9] states that nowadays the entire Information and Communication Technologies sector consumes about 10% of the energy generated in the whole world.

As mentioned in section 2, the TCP/IP stack illustrates the practical usage of cross-layer approach to ensure system fault tolerance. The cross-layer design is now widely used in the area of the wireless sensor networks (WSNs). Since reliability, performance and energy consumption are crucial factors for these systems, the optimal operation of the whole system can only be guaranteed when the layers are considered together. Single layer approach cannot share important information among different layers. Consequently, each layer does not have complete information and it is impossible to achieve the optimal system operation. In addition, the single layer approach is incapable of adapting to the environmental change. Paper [10] discusses cross-layer adaptivity techniques, which leverage functionalities at different layers of the protocol stack. The application layer is frequently involved in these activities, supporting current system operation in accordance with measurements and forecasts of the monitored system. Study [11] proposes a new routing protocol based on the cross-layer principle in order to manage faults in wireless sensor networks, decrease signalling overhead and power consumption. A cross-layer data delivery protocol for delay/fault-tolerant mobile sensor networks was developed in [12]. The protocol aims to optimize energy consumption in the light of throughput requirement, stable connectivity of the sensor nodes and sufficient channel bandwidth.

The on-going work on cross-layer fault tolerance is patchy and is mainly focusing on the area of WSNs. Cross-layer fault tolerance is not applied in many-core systems, which will be the predominant architecture in the future. Unfortunately, development of cross-layer fault tolerance complicates the system design and it breaks the abstractions and needs a holistic approach. To make it practical the developers need to be assisted by novel system and software engineering techniques. The aim of this PhD study is to develop such techniques (architectures, models, patterns, libraries, tools) and to demonstrate that applying the cross-layer fault tolerance for many-core systems can improve performance and energy-efficiency.

6 Ongoing and future work

Several topics are being investigated during the first year of the study to understand better the domain and to develop an initial understanding of the requirements for cross-layer fault tolerance engineering in many-core systems.

6.1 Experiments with Odroid-XU3 board

The Odroid-XU3 board is a small Octa-Core computing device implemented on energy-efficient hardware, which is based on the ARM big.LITTLE heterogeneous architecture and consists of a high performance Cortext-A15 quad core processor block (big), a low power Cortex-A7 quad core block (little), GPU and DRAM. The following experiments were carried out to understand the correlation between power consumption and performance. To clarify the voltage-frequency dependencies for the A7 and A15 power domains, the first experiment measured voltage, current and power at different frequencies without any additional workload. The second experiment involved measuring the same parameters under 100% load created by a stress test program executing 50 million square root operations, and brought unexpected results, that at the identical frequency, A7 was a bit faster than A15 and consumed four times less power. The same trend was observed with sine and cosine functions. Experiments with other operations gave the anticipated results when A15 was more than twice faster than A7 at the same frequency and almost three times faster at a maximum frequency. In the third experiment, we investigated the influence of thread sleep state (between active state periods) on energy consumption. Our results show that in terms of energy, it is more efficient to execute the task as fast as possible. Fourth experiment was held to investigate possible power and energy savings after disabling CPU cores for the cases when the workload is not very high. It was found that power consumption reduces more than 8 times after disabling all four cores of A15 processor. This technique can be used to reduce power and energy consumption of many-core systems during their idle time.

6.2 Threads scheduling

In order to understand the behaviour of the scheduler at the ultimate load in Windows and Ubuntu OSs the threads scheduling experiments were carried out. Two or more threads with 10 milliseconds tasks, requiring as much CPU time as possible were bound to one CPU core using thread affinity. It was observed that Windows and Ubuntu schedulers have different logic. Windows scheduler tries to run one thread to completion and after that switches to another thread, whereas Ubuntu scheduler tries to switch between different threads during execution. These findings should be taken into account while designing cross-layer fault tolerance for high performance and reliable many-core systems.

6.3 Global exception catch block

Breaking the abstractions will lead to the situation when the encapsulation principle is violated. This, in turn, can be the reason for the inconsistent state. Let us consider the hypothetical global catch block, which will specify that all exceptions down the call stack should be propagated to this global catch block, even though there are catch blocks below that can handle these exceptions. For example, we have module 1, that calls function `Do` of module 2 inside the try-global-catch statement. An implementation of the `Do` function already has a standard try-catch statement inside. If an exception is thrown in function `Do` and the standard catch block in the same function has only error loggers and does not attempt any recovery or rollback, than module 2 remains consistent after the exception is propagated to the global catch block of module 1. However, if function `Do` has a transaction that should be rolled back in the catch block, than the state of module 2 could become inconsistent after exception propagation to the global catch block in module 1, since the catch block in the `Do` function will not be applied. This simple example illustrates that it is necessary to study the effects of error propagation through the system layers in order to understand the problems to be faced during development of cross-layer fault tolerance. We are now developing an advanced exception handling scheme to support global exception handling as a programming mechanism for cross-layer fault tolerance.

6.4 Relax framework

The authors of [13] propose a co-called language-level Relax mechanism for providing energy-efficient reliability and cross-layer fault tolerance for supercomputers. It allows the developer to specify code regions where low-reliability computations should be tolerated. In case of error, recovery is performed by re-execution of the "relaxed" code block. We plan to apply the similar approach on Odroid-XU3 board, by using little cores for detection of the calculation errors of big cores. If the error is detected, the recovery will be done by simple re-execution of the calculation. This technique will be useful for developing more sophisticated cross-layer fault tolerance mechanisms for many-core systems.

6.5 Future work

Our short-term plans include work in the two areas. Firstly, we will apply the Order Graphs – the scalable approach developed in our group [14] - to model fault tolerance, power consumption and performance of many-core systems and to represent cross-layer fault tolerance. In particular, we would like to apply the idea of model fidelity to the area of fault/failure significance and of developing the corresponding cross-layer fault tolerance. Secondly, a medium-scale case study will be implemented to gain the experience in developing cross-layer fault tolerance for many-core systems.

Acknowledgments. This work is supported by the EPSRC/UK PRiME project and by the School of Computing Science, Newcastle University (UK).

7 References

1. A. Avizienis, J.C. Laprie, B. Randell, C. E. Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," IEEE Trans. Dependable and Sec. Comput., vol. 1, no. 1, pp. 11-33, 2004.
2. A. DeHon, N. Carter and H. Quinn, "Final Report for CCC Cross-Layer Reliability Visioning Study," http://relxlayer.org/, 2011.
3. S. Borkar, "Thousand Core Chips—A Technology Perspective," in Proc. of the 44th Annual Design Automation Conference (DAC), 2007.
4. A. Vajda, Programming Many-Core Chips, Springer US, 2011.
5. B. Randell and J. Xu, "The evolution of the recovery block concept," In Software Fault Tolerance, John Wiley & Sons Ltd, 1994, pp. 1-22.
6. L. Chen and A. Avizienis, "N-version programming: A fault tolerance approach to reliability of software operation," In Proc. of the Twenty-Fifth International Symposium on Fault-Tolerant Computing, pp. 113-119, June 1995.

7. F. Cristian, "A recovery mechanism for modular software," in Proc. of the 4th international conference on Software engineering, ICSE '79, 1979.
8. T. Anderson, P. A. Lee, Fault tolerance, principles and practice, Prentice/Hall International. 1981.
9. M. P. Mills, "The Cloud Begins With Coal," CEO, Digital Power Group, 2013.
10. L. Carnevali, L. Ridi, E. Vicario, "Stochastic Fault Trees for cross-layer power management of WSN monitoring systems," in Proc. of IEEE Conference on Emerging Technologies & Factory Automation, pp. 1-8, 2009.
11. P. Rachelin Sujae, M. Vigneshpandi, "A Cross Layer Fault Tolerant Communication Architecture for Wireless Sensor Networks," Middle-East Journal of Scientific Research, pp. 1292-1296, 2014.
12. Y. Wang, H. Wu, F. Lin, N.F. Tzeng, "Cross-Layer Protocol Design and Optimization for Delay/Fault-Tolerant Mobile Sensor Networks (DFT-MSN's)," IEEE Journal on selected areas in communications, vol. 26, no. 5, pp. 809-819, 2008.
13. C.H. Ho, M. de Kruijf, K. Sankaralingam, B. Rountree, M. Schulz, B. R. de Supinski, "Mechanisms and Evaluation of Cross-Layer Fault-Tolerance for Supercomputing," In Proc. of the 41st International Conference on Parallel Processing (ICPP), pp. 510-519, 2012.
14. A. Rafiev, F. Xia, A. Iliasov, R. Gensh, A. Aalsaud, A. Romanovsky, A. Yakovlev, "Order Graphs and Cross-layer Parametric Significance-driven Modelling," in Proc. of ACSD 2015. Brussels, Belgium. IEEE CS, 2015.