This article is part of the IEEE Reliability Society 2008 Annual Technology Report.

Influence of Architecture on Software Reliability

Norman Schneidewind
Email: ieeelife@yahoo.com

Introduction

Today, it is important to integrate reliability with the performance of software architecture because architectural characteristics, such as the complexity of software, can have a profound influence on reliability. The need to deal simultaneously with both performance and dependability (reliability) was initially recognized in the mid-1970s. More specifically, it surfaced as the consequence of then-experimental architectures such as PRIME (developed at the University of California, Berkeley [1]), which exhibited "degradable" performance in the presence of operational faults. Performance refers to how effectively (e.g., throughput, delay) or efficiently (e.g., resource utilization) a system delivers a specified service, presuming it is delivered correctly. On the other hand, dependability or what is also referred to as reliability is the trustworthiness of a computer system [2] with respect to delivery of a specified service. With this view, a system failure occurs if the delivered service no longer complies with its specification. Moreover, since faults are the source of failures, their existence and their effects are principal concerns [3].

Measures to Integrate Reliability and Performance

In order to integrate reliability and performance, the concept of performability was developed that refers to measures that quantify a system's ability to perform in the presence of faults. The definition of the computational capacity of a system state is the amount of useful computation per unit time available on the system in that state [4]. For example, the amount of computation that could be performed when not in the failed state. In addition, the expected efficiency of the computation should be evaluated, as we do in the following sections.

Definitions

T_a: time available to perform a task
T_{rj}: time required to perform a task r on architecture j
P_j: performance of architecture j
  P_j could be the instruction processing rate of a computer, equal to the clock rate, assuming instructions execute at clock rate
R_{rj} : software reliability of architecture j during time T_r
N: number of instructions in a program
E_{rj}: computational efficiency of task r on architecture j

Software Architectural Model

The amount of computation that can be performed by architecture j on task r is given by equation (1.1).

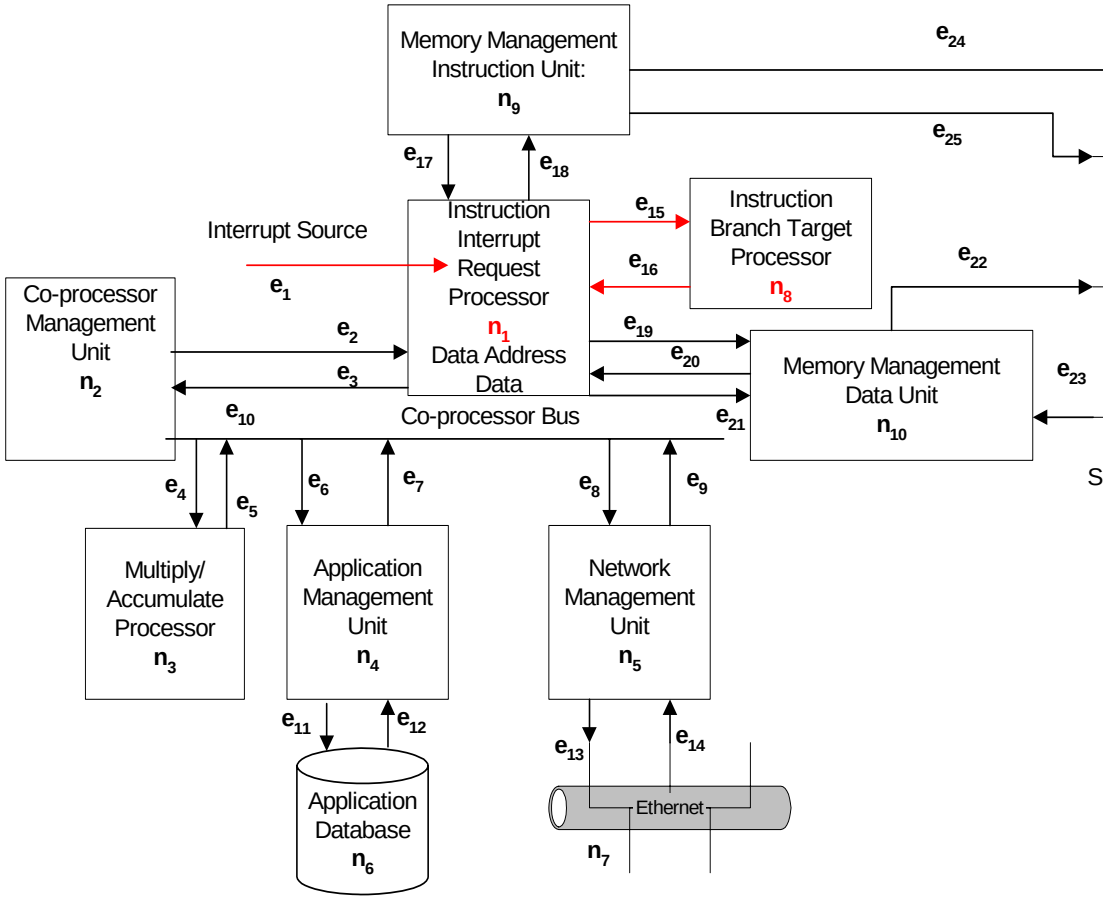
$$T_{rj} = N / P_j \tag{1.1}$$

Using equation (1.1), the computational efficiency of task r on architecture j is computed in equation (1.2), where lower values correspond to higher efficiencies.

$$E_{rj} = (T_{rj} / T_a) = (N / P_j) / T_a \tag{1.2}$$

Architectural Characteristics

Few techniques consider failure propagation in software architectures for software reliability assessment. One study that considered failure propagation analyzed failure propagation based on architectural service routes (ASRs). An ASR is a sequence of components that are connected through interfaces [5]. We consider failure propagation as generated by instruction and data flow paths, similar to components connected by interfaces. Architectural characteristics, such as data flow paths, influence the reliability that can be achieved in a system. A data flow path that is a sequence of nodes and edges is highlighted in red in Figure 1. This is an example of where failures could occur. In addition to the many avenues for failure that are exposed by an architecture with many data flow paths, this condition makes it difficult to recover from failures and to repair the faults responsible for failures.



Number of Nodes: Nn = 11
Number of Edges: Ne = 25
Cyclomatic Complexity: CC = Ne – Nn + 1 = 15

Figure 1. Operating System Architecture (Based on Intex XScale Technology)

Some characteristics of software architecture are related to software system configuration. An example is shown in Figure 1 that shows an operating system architecture based on the Intel XScale Technology [6]. The complexity of the configuration will influence software reliability.

We account for this complexity by enumerating the nodes and edges in the configuration, and then computing the cyclomatic complexity (CC) [7] from these elements, as shown in Figure 1. Cyclomatic complexity is used as a weighting factor and applied to reliability to account for the complexity of the architecture.

The cyclomatic complexity of software architecture j is computed in equation (1.3), where N_{je} is number of edges and N_{jn} is number of nodes.

$$CC_j = (N_{je} - N_{jn}) + 1 \quad (1.3)$$

To compute a weighting factor, a maximum value CC_m of CC_j is needed to normalize the complexity in equation (1.3). CC_m is obtained by recognizing that the most complex architecture is one that has a single master node that controls all slave edges through which control and application data must pass. In other words, $N_{jn} = 1$ in equation (1.3), so that maximum cyclomatic complexity is expressed in equation (1.4).

$$CCm = Nje \quad (1.4)$$

Using equations (1.3) and (1.4), the weighting factor for architecture j is expressed in equation (1.5)

$$w_j = CC_j / CC_m = (N_{jn} - 1) / N_{je} \quad (1.5)$$

Integrating Software Reliability

Now, we include software reliability in the architectural model. To do this, we use the binomial distribution to represent the probability P (x), of x number of instructions in a program that have faults and fail, each with a probability of failure p, in a program of size N, in equation (1.6). We define a reliable program as one that has x = 0 instructions that fail. Therefore, equation (1.6) is compressed to equation (1.7), which is the software reliability of a program with N instructions.

$$P(x) = \left(\frac{N!}{(x!)(N-x)!}\right)(p^x)(1-p)^{(N-x)} \quad (1.6)$$

$$R_{rj}(N) = P(0) = (1 - p)^N \quad (1.7)$$

Now, the weighting factor in equation (1.5) can be integrated with the equation for software reliability to produce equation (1.8).

$$E_{rj} = ((1 - p)^N (N / P_j) / T_a) ((N_{jn} - 1) / N_{je}) \quad (1.8)$$

The software architecture (operating system) is implemented on the Intel XScale processor, with the clock and instruction execution rates shown below. In order to do the evaluation of computational efficiency, using equation (1.8), a range of performances are used, as shown in Table 1, where the bolded quantities are the clock rates listed below. Note that in addition to computing computational efficiency, software reliability could be predicted for given values of computational efficiency.

Intel XScale Processor
Clock rates:
• 266 MHz, 266 million instructions per second
• 400 MHz, 400 million instructions per second
• 533 MHz, 533 million instructions per second

Result of Integrating Reliability with Computational Efficiency
The result of the integration of reliability with computational efficiency is shown in Figure 2. Fortuitously, both efficiency and reliability improve, as performance increases. The practical application of this analysis is that several architectures of operating systems, or application software, could be evaluated, using this process, to identify the one with the best efficiency and reliability. Using this approach, the engineer can do a tradeoff analysis to see how much processor computational power is warranted to achieve efficiency and reliability goals. For example, looking a Figure 2 you can see that increasing the processor clock rate beyond 800 Mhz (800 million instructions per second) would not significantly improve efficiency and reliability. Therefore, the additional cost would not be justified.

Table 1 Software Architecture Efficiency and Reliability

performance clock speed Mhz P_j	probability of 0 instructions failing p	computational efficiency E_{rj}	software reliability R_{ij} (N)
100	0.000100	1.47	0.3679
200	0.000090	0.81	0.4066
266	0.000080	0.68	0.4493
300	0.000070	0.66	0.4966
400	0.000060	0.55	0.5488
500	0.000050	0.49	0.6065
533	0.000040	0.50	0.6703
600	0.000030	0.49	0.7408
700	0.000020	0.47	0.8187
800	0.000010	0.45	0.9048
900	0.000009	0.41	0.9139
1000	0.000008	0.37	0.9231

N_{je}	25	number of edges
N_{jn}	11	number of nodes
N	10000	number of instructions
T_a	10	time available (hours)

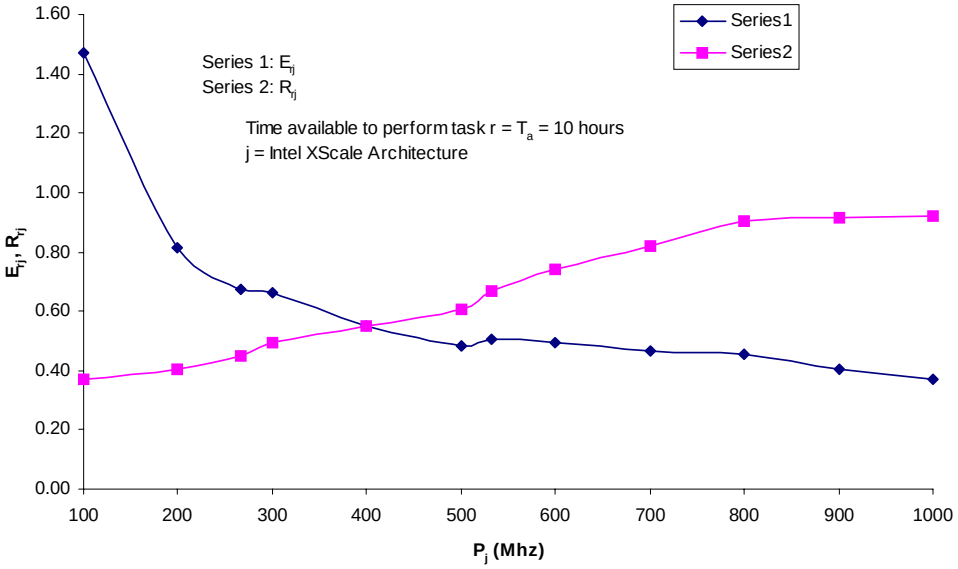


Figure 2. Computational Efficiency E_{rj} and Software Reliability R_{rj} vs. Performance of Architecture j, P_j

Summary

Recognizing the important need to consider the interaction between reliability and performance a model for integrating software reliability with software architecture has been developed. In addition to the reliability of the architecture, the computational efficiency of the architecture was evaluated. In order to illustrate the evaluation process, a microprocessor and its operating system from industry were used. This example showed that both efficiency and reliability improved as performance increased. Due to a space limitation, only one architecture was evaluated, but other architectures could be evaluated using the same process. A major determinant of efficiency and reliability is the complexity of the architecture. The reliability engineer can use this model to predict whether, for a given architecture, reliability goals are likely to be achieved. Conversely, given reliability goals, the engineer can assess whether computational efficiency could be achieved for an architecture.

References

[1] B. R. Borgerson and R. F. Freitas, "A Reliability Model for Gracefully Degrading and Standby Sparing Systems," IEEE Transactions on Computers, vol. C-24, no. 5, May 1975, pp. 517-525.

[2] J. C. Laprie, editor, Dependability: Basic Concepts and Terminology, volume 5 of Dependable Computing and Fault-Tolerant Systems, Springer-Verlag, 1992.

[3] Meyer, J.F., "Performability evaluation: where it is and what lies ahead", Proceedings of the International Computer Performance and Dependability Symposium, 24-26 April 1995, pp. 334 – 343.

[4] M. D. Beaudry, "Performance-Related Reliability Measures for Computing Systems," IEEE Transactions on Computers, vol. C-27, no. 6, June 1978, pp. 540-547.

[5] Atef Mohamed and Mohammad Zulkernine, "On Failure Propagation in Component-Based Software Systems," The Eighth International Conference on Quality Software, 2008, pp.402-411.

[6] http://download.intel.com/design/network/datashts/25247907.pdf

[7] Norman F. Fenton and Shari Lawrence Pfleeger, Software Metrics: A Rigorous & Practical Approach, Second Edition, PWS Publishing Company, 1997.