Applying non-constant volatility analysis methods to software timeliness

Shane Brennan Vinny Cahill Siobhán Clarke
Lero @ TCD, Distributed Systems Group,
Department of Computer Science,
Trinity College Dublin, Ireland
{*brennash, vinny.cahill, siobhan.clarke*}*@cs.tcd.ie*

***Abstract*—Timing analysis is the application of one or more well-established predictive methods to derive the likely timing behaviour of a specific software task executing on a particular hardware platform. Current approaches towards timing analysis are predicated on the presumption that the software under test is always fixed, i.e., it remains unchanged once deployed to the target hardware. A dynamically adaptable system modifies its behaviour in unanticipated ways, and at unpredictable intervals, to exploit the prevailing operational environment. However, when the software is capable of runtime adaptation, statically derived timing estimates are incapable of accurately capturing the changes in the software timeliness caused by functional adaptations. Traditional timing analysis methods cannot be applied to a dynamically adaptive system, due to the inconstant nature of the software, the unpredictable scheduling of functional adaptations, and the need to produce timing estimates at runtime. This paper describes a work in progress with the aim of statistically forecasting software timeliness using non-constant volatility methods. We outline how timing bounds may be derived for an adaptable software system, with changeable underlying functionality, and show how timing predictions can be modified, using novel statistical models, to mirror runtime functional changes to the software.**

I. Introduction

Timing analysis is defined by Wilhelm et al.[1] as "*the process of deriving execution-time bounds or estimates*" for software tasks, and traditionally infers the offline application of timing-analysis tools to evaluate source code. Various approaches to timing analysis exists, such as such as code simulation[2], tool-based analysis[3] and execution trace analysis[4], and these share a common emphasis on producing dependable worst-case execution time (WCET) bounds for the software, typically operating in real-time embedded or safety critical environments. The developer is compelled, in this context, to streamline software functionality so as to enable the later application of timing analysis methods, e.g., prohibiting unbounded loop structures, recursion and other execution flows determined dynamically. Once deployed to the target hardware, the software must remain unchanged in order to maintain the validity of the statically determined timing bounds. Since traditional timing analysis approaches proscribe further changes to the software after deployment, the software typically deals with any environmental variability using a pre-existing set of contingency behaviours. Each contingency behaviour is tested statically and verified before the entire code-base is deployed to the target hardware platform. This approach, while emphasising the safety of the timing bounds produced for the software, places extra requirements on the developer, and makes inefficient use of the available hardware resources.

Software developers are turning to dynamically adaptive software frameworks to provide a more reactive, flexible and resource-efficient approach towards developing software for highly variable operating environments. Rather than create software with a large set of behaviours for each possible operating condition, a dynamically adaptive system modifies its functionality, at runtime, to exploit the variability in its operating environment. This operating environment is in a state of constant flux, but can be sampled by continually examining sensor inputs, user requests, internal state changes or other software performance metrics that change in unpredictable ways, over time. The functional scope of the software is optimized, at runtime, to exploit changes in the operating environment, allowing only software behaviours relevant to the current operational context to be enabled, rather than the system being deployed encumbered with multiple, often obscure, redundant behaviours.

However, current timing analysis techniques are inapplicable to dynamically adaptable software, since the timing analysis is either performed offline, or cannot adapt timing estimates quickly enough to reflect functional changes to the software occurring at runtime. We believe that although dynamically adaptable software cannot currently offer the safety guarantees required for real-time embedded and safety-critical systems, soft real-time requirements can be met without sacrificing the flexibility and optimization of a dynamically adaptable system. This requires a fresh approach to timing analysis that does not require the system to be halted, and can automatically refresh the timing estimate to match the latest configuration of the system without impinging of normal operation.

Instead of statically producing a timing estimate, we propose to develop a statistical model of software timeliness that can be adjusted at runtime, to match functional adaptations to the software. A statistical model provides a convenient, and well-established means of studying the behaviour of a changeable system. The complexities inherent in hardware

and software systems can be abstracted into a simplified performance model, allowing forecasts of software timeliness to be derived quickly, and without radically impinging on normal operations.

The remainder of this paper is structured as follows, section II introduces volatility forecasting and the ARCH statistical model, section III outlines our current approach integrating statistical models with runtime adaptable software, section IV describes some of the related work, and section V presents our initial conclusions and future work.

II. Volatility Forecasting

Statistically modelling complex software timeliness shifts the focus away from the cause-and-effect investigations used to derive timing predictions with traditional methods, to a more measurement-based process applied to the software it situ. The complexity of current processors, and the subtle interplay of execution-time effects, e.g., caching, pipelining, pre-fetching etc. all produce timing effects that are difficult to determine statically, even when the software itself remains unchanged after deployment. As the complexity of the underlying systems increases, establishing the timeliness of the software becomes more difficult, and it has been cited as beginning to overwhelm the capability of many formal methods-based approaches[5].

In addition should the software be subject to functional adaptations at runtime, the consequent changes in the software timeliness will confound all current approaches to timing analysis. Static timing analysis methods, i.e., those used in an offline context on a static code-base, have an open-ended analysis period requiring human intervention, and are thus unsuitable for a continuously executing system. Dynamic timing analysis methods, such as measurement-based analysis, will view the effects of any software adaptation as a series of unlikely timing variations within the original software configuration, rather than a more fundamental change in the underlying software timing behaviour. A reactive statistical model, aware of the adaptable nature of the software, can provide more flexible forecasting tool to differentiate the effects of random variation within a single configuration from systematic variation caused by software adaptation.

Using simplistic statistical models to forecast future software timeliness will not produce a good measure of the central tendency of an adaptive system. By averaging measurements over the entire execution lifetime of an adaptive system, it may be difficult to distinguish functional adaptations within the software from random variations in timing measurements of the system. A time series can be used to capture a sequence of software timing measurements, taken at successive time intervals, and provide a forecast of future timeliness. However, time series analysis traditionally assumes that the variance present in a set of measurements is constant, i.e., the statistical dispersion of measurements from the expected mean is fixed. For a dynamically adaptive system, the functionality of the software will change at runtime, leading to alterations in its timing behaviour, occurring at unpredictable points during execution. A dynamically adaptive system will have heteroskedastic timing behaviour, requiring a statistical modelling and forecasting technique, applicable at runtime, that can cope with changeable variance in timing measurements caused by functional adaptations to the software.

Performing regression analysis using heteroskedastic time series data is difficult, however the AutoRegressive Conditional Heteroskedastic (ARCH) model, introduced by Engle[6] was created to model time series with non-constant variances. The ARCH model calculates the likely degree of difference between the observation and the sample mean (called the error term), as a function of the variances of previous periods' error terms, i.e.,

$$\epsilon_t = \sigma_t z_t,$$

where ϵ_t is the error term for time t, σ_t is the running variance at time t, and z_t is a independent and identically distributed normal distribution with a mean of 0 and a standard deviation of 1. In simple terms, future trends are predicted as an exponentially decreasing sum of the errors from previous forecasts, i.e.,

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{q} \alpha_i \epsilon_{t-1}^2,$$

where σ_t^2 is the standard deviation, α is a reducing coefficient for previous error terms, ϵ_{t-1} are previous error terms and q is the lag length. The lag length is a defineable set of relevant terms within the time series used to estimate the standard deviation. Applying the ARCH model to forecast software timeliness requires a careful selection of this lag length, to consider only timing measurements relevant to the current software configuration, i.e., it should not include timing measurements relating to previous configurations of the system. The lag period can be adjusted to suit the overall level of variation, e.g., where the timing measurements have been observed to rapidly and unexpected diverge from the sample mean, a smaller lag length provides a faster adjustment to the statistical model.

Bollerslev[7] extended the ARCH model with the Generalized Autoregressive Conditional Heteroskedastic (GARCH(p,q)) model[7] to allow greater flexibility in calculating the current error term. Instead of simply estimating the current standard deviation as a sum of the previous error terms in the series, he includes the previous standard deviations in the calculation also, i.e.,

$$\sigma_t^2 = \alpha_0 + \sum_{i=1}^{p} \alpha_i \epsilon_{t-1}^2 + \sum_{i=1}^{q} \beta_i \sigma_{t-1}^2,$$

where α and β are reducing coefficients, for the error and

the standard deviation terms respectively.

The ARCH/GARCH statistical model can be easily fitted with a regression line to provide a forecast of likely future timing measurements for the system. Changes in the number of terms considered in the lag length will affect the fit of the regression line, i.e., a smaller number of terms can cause errant values to bias the forecast more than in a larger data set.

This statistical approach was designed to analyse volatile environments affected by obscure, often hidden, parameters, such as the currency exchange markets, national rates of inflation and the stock market. Software timeliness provides a different challenge, in that there will be a finite best-case and worst-case time (aside from halt) that bound the fluctuations of the timing measurements. However, certain analogies from the markets remain, such as the effect of large positive jumps, e.g., timing delays, on subsequent timing measurements. Both software timeliness and volatile markets demonstrate an asymmetry associated with the effects of equal positive and negative jumps on subsequent measurements within a time series.

III. Current Approach

Currently, we are looking at the application of a specialised statistical model to forecast the mean timeliness of a dynamically adaptive system, in near real-time. We plan to extend existing volatility forecasting models to estimate the average-case timing behaviour of an adaptable system, using a concurrently executing timing analysis process. Adaptations to the software will trigger a refresh of the timing estimate for the software, using our extension to the basic GARCH model which we call Real-Time Adaptive GARCH (RTA-GARCH). The key difference between RTA-GARCH and other forecasting models is the requirement to complete the analysis quickly, and return an estimate of software timeliness while the software remains executing. The limited time in which to generate useable timing measurements will be made more difficult by the timing errors introduced by the measurement process itself, executing on a single system. We plan to fast track the measurement process, and validate the timing measurements, by pre-calculating as much of the data as possible before the adaptation is triggered in the software.

Figure 1 shows a series of timing measurements from a function calculating a DES encryption key. There are periodic spikes signifying timing delays, possibly caused by I/O interrupts in the system, as well as time-varying volatility clustering within the data.

We plan to use several C/C++ functions to benchmark our statistical model, and we are currently generating timing estimates for a series of software functions available from the MRTC at Mälardalen University[1]. So far, we have

[1]http://www.mrtc.mdh.se/projects/wcet/benchmarks.html

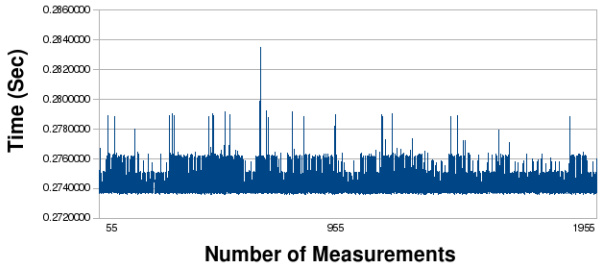

Figure 1. An graph showing the variation in timing measurements within a single function.

applied our initial statistical model to pre-generated timing measurements of a factorial function, a DES cipher function and a function simulating a Petri net. The bounded timing estimates we have produced so far have successfully captured the timing variability within individual functions, as well as simulated adaptations between functions. We simulate adaptations by interleaving blocks of timing measurements from different functions to construct a combined data set with multiple apparent timing behaviours, and have seen our model successfully adapting to the varying execution times of the different functions.

IV. Related Work

Our work is related to a number of distinct areas of research, ranging from dynamically adaptable software frameworks, to timing analysis methods and statistical forecasting models for heteroskedastic time series data.

Currently, there are a number of dynamically adaptable component-based software frameworks that provide varying levels of support for runtime adaptation. Fractal[8], OpenCom[9], DACIA[10], Q-Components[11] and the K-Component model[12] support the adaptation of functional elements within the software during runtime, either through exchanging functional elements in the case of Fractal, or by enabling redundant behaviours in the software as in the Q-Component model. Unfortunately, none of these adaptive frameworks include any built-in functionality to derive estimates of software timeliness after an adaptation.

A few software frameworks do exist that provide integrated timing analysis within the component framework, such as PECT/PACC[13] and the Palladio Component Model[14], however they do not support any adaptation within the software once deployed and executing. A similar static analysis of component-based software is described by Lüders et al.[15], who showed how timing analysis may be performed on a Microsoft COM component model before execution. Similarly, Cheung et al.[2], have proposed a framework to predicting the reliability of software components during architectural design, and Eskenazi et al.[16] present an incremental timing analysis process, to be applied at each stage of the software design process. Again, these

analytic approaches cannot re-validate timing estimates at runtime when an adaptation occurs within the software.

Measurement-based timing analysis methods, such as those proposed by Wenzel[17] and Schaefer[18], are used as an adjunct to static timing analysis methods, to target specific code sections for rigorous offline analysis. However, these approaches are insufficient for an adaptable system, since the estimate requires extensive inputs from domain experts, and must be generated offline.

V. Conclusions and Future Work

Our initial investigations into using GARCH models to predict the timeliness of dynamically adaptable systems show that our approach is feasible, and could be applied to software running in a soft real-time environment. What remains, is to refine our RTA-GARCH model and implement it within a dynamically adaptable system with strong timing requirements. We need to discover the effects of executing our model on the timeliness of the underlying system, and remove as much of this effect from our timing predictions. Lastly, we wish to study any trade-offs that may exist between the acceptable accuracy and precision of a prediction, weighed against the time taken to derive it.

A. Acknowledgements

This work was supported, in part, by Science Foundation Ireland grant 03/CE2/I303_1 to Lero - the Irish Software Engineering Research Centre (www.lero.ie).

References

[1] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The determination of worst-case execution times—overview of the methods and survey of tools," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, pp. 1–53, 2008.

[2] L. Cheung, R. Roshandel, N. Medvidovic, and L. Golubchik, "Early Prediction of Software Component Reliability," in *30th International Conference on Software Engineering (ICSE'08)*, 2008, pp. 111–120.

[3] J. Engblom, A. Ermedahl, M. Sjodin, J. Gustafsson, and H. Hansson, "Applying Static WCET Analysis to Automotive Communication Software," *International Journal of Software Tools for Technology Transfer*, vol. 4, pp. 437–455, 2001.

[4] C. Burguiére and C. Rochange, "History-based Schemes and Implicit Path Enumeration," in *6th International Workshop on Worst-Case Execution Time (WCET) Analysis*, 2006.

[5] H. W. Schmidt, "Architecture-Based Reasoning About Performability in Component-Based Systems," in *33rd Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2007)*, 2007, pp. 130–137.

[6] R. F. Engle, "Autoregressive Conditional Heteroskedasticity with Estimates of the Variance of U.K. Inflation," *Econometrica*, vol. 50, pp. 987–1008, 1982.

[7] T. Bollerslev, "Generalized Autoregressive Conditional Heteroskedasticity," *Journal of Econometrics*, vol. 31, pp. 307–327, 1986.

[8] P.-C. David and T. Ledoux, "Towards a Framework for Self-Adaptive Component-Based Applications," in *Proceedings of the International Working Conference on Distributed Applications and Interoperable Systems (DAIS 2003)*. Paris, France: Springer-Verlag, 2003.

[9] G. S. Blair, G. Coulson, and P. Grace, "Research directions in reflective middleware: the Lancaster experience," in *Proceedings of the 3rd Workshop on Adaptive and Reflective Middleware (ARM'04)*, 2004, pp. 262–267.

[10] R. Litiu and A. Parakash, "Developing adaptive groupware applications using a mobile component framework," in *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'00)*, 2000, pp. 107–116.

[11] D. A. Menascé, H. Ruan, and H. Gomaa, "A framework for QoS-aware software components," in *Proceedings of the 4th International Workshop on Software and Performance*, 2004, pp. 186–196.

[12] J. Dowling and V. Cahill, "Dynamic Software Evolution and the K-Component Model," in *Workshop on Software Evolution, OOPSLA*, 2001.

[13] S. Hissam and J. Ivers, "Prediction-Enabled Component Technology (PECT) Infrastructure: A Rough Sketch," Software Engineering Institute, Carnegie-Mellon University, Tech. Rep. CMU/SEI-2002-TN-033, 2002.

[14] M. Kuperberg, K. Krogmann, and R. Reussner, "Performance Prediction for Black-Box Components Using Reengineered Parametric Behaviour Models," in *11th International Symposium on Component-Based Software Engineering (CBSE 2008)*, 2008, pp. 48–63.

[15] F. Lüders, S. Ahmad, F. Khizer, and G. Singh-Dhillon, "Using Software Component Models and Services in Embedded Real-Time Systems," in *Proceedings of the 40th Hawaii International Conference on System Sciences (HICSS'07)*, 2007.

[16] E. Eskenazi, A. Fioukov, and D. Hammer, "Performance Prediction for Component Compositions," in *7th International Symposium on Component-based Software Engineering (CBSE'04)*, 2004, pp. 280–293.

[17] I. Wenzel, R. Kirner, B. Rieder, and P. Puschner, "Measurement-based worst-case execution time analysis," in *In Proceedings of 3rd IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS'05)*, 2005, pp. 7–10.

[18] S. Schaefer, B. Scholz, S. M. Petters, and G. Heiser, "Static analysis support for measurement-based WCET analysis," in *12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2006.