CORE

Metadata, citation and similar papers at core.ac.uk

Provided by Kent Academic Repository

Using Probability to Reason about Soft Deadlines

Andy King and Jeremy Bryans

Computing Laboratory,
University of Kent at Canterbury, CT2 7NF, UK.
{amk, jwb}@ukc.ac.uk

Abstract. Soft deadlines are significant in systems in which a bound on the response time is important, but the failure to meet the response time is not a disaster. Soft deadlines occur, for example, in telephony and switching networks. We investigate how to put probabilistic bounds on the time-complexity of a concurrent logic program by combining (on-line) profiling with an (off-line) probabilistic complexity analysis. The profiling collects information on the likelihood of case selection and the analysis uses this information to infer the probability of an agent terminating within k steps. Although the approach does not reason about synchronization, we believe that its simplicity and good (essentially quadratic) complexity mean that it is a promising first step in reasoning about soft deadlines.

1 Introduction

Time-critical constraint applications, such as those that arise in robotics, distributed multimedia and embedded systems, typically have to answer a request for a service within a specified time. In a real-time command and control system, failure to meet a deadline may lead to the loss of human life. In telephony or switching, however, failure to meet a deadline, may simply compromise the quality of a video stream. This would be undesirable rather than catastrophic. These deadlines are said to be soft [6] and it is sufficient to ensure that there is a high probability that the deadline is met.

In this short paper we propose a simple analysis that infers the likelihood of an agent, expressed in a concurrent logic language, of meeting a soft deadline. The basic problem is to infer a distribution for the probability of an agent terminating (and hence servicing a request) in, say, up to k steps of computation. By examining this distribution the programmer can isolate a performance bug/bottleneck and thus patch/redesign the code. This probabilistic information is not available from existing techniques, such as profiling [5], that would typically associate each agent with an average execution time. Although profiles can be enriched with variance information [8], variance is just a crude (and often hard to interpret) measure of spread. For example, suppose that we can model the likelihood of an agent answering a request for a service with an exponential distribution (this sort of assumption is routinely made in probabilistic model checking [6]). Information like "the probability of servicing the request within k steps is $1 - e^{-\lambda k}$" is far more useful than "the agent has a mean of $\frac{1}{\lambda}$ steps with a variance of $\frac{1}{\lambda^2}$". Put simply, different distributions have the same variance and mean and variance alone are not enough to infer the probability of terminating with k steps.

Rather than just inferring that the application (the top-level or main agent) terminates, our analysis infers the probability of each agent (and hence each service) in the system terminating. This is important because time-critical applications are frequently persistent. Although the approach is applicable to concurrent constraint programming, we simply measure time in terms of the number of resolution (goal-head matching) steps. It is not clear whether this measure would be a sensible unit of time with a more elabourate constraint solver where the time to check for entailment and satisfiability is likely to vary dramatically from agent to agent and from store to store. Deciding the time unit that is most appropriate for a constraint application, we feel, is an independent problem and it is not addressed in this paper.

This paper is structured as follows: In section 2 we discuss the need for an (on-line) probabilistic profiling component in the analysis. In section 3 we present two (off-line) analyses that take the profiling information, as input, and produce, as output, the distribution of each agent terminating within k steps. Sections 4 and 5 respectively present the related and future work; whereas section 6 presents our conclusions.

2 On-line profiling component

Like other probabilistic analyses [1], our analysis is based on being able to associate execution frequencies (or, when non-zero, normalized, execution probabilities), with the cases defining an agent. Light-weight profiling techniques have been developed for logic programs [5] and adapted to concurrent logic programs such as Strand [4] so acquiring the probabilities is not a problem. What is a potential problem is the reliability of this execution frequency information. The essential problem is that the termination behaviour of a concurrent logic programs relates to (1) synchronization and (2) non-deterministic case selection and these properties are store sensitive. This is illustrated in the program P:

$$P = \begin{cases} \mathsf{p(x)} :\!- \mathsf{x = a} \rightarrow \mathsf{p(x)} & \mathsf{q} :\!- \mathsf{p(x)} \\ \qquad + \mathsf{x = a} \rightarrow \mathsf{stop} & \mathsf{r} :\!- (\mathsf{tell(x = b)} \parallel \mathsf{p(x)}) \\ \qquad + \mathsf{x = b} \rightarrow \mathsf{stop} & \mathsf{s} :\!- (\mathsf{tell(x = a)} \parallel \mathsf{p(x)}) \end{cases}$$

The agents q, r and s essentially invoke the agent $\mathsf{p(x)}$ with a different store (different test data). q will never terminate since $\mathsf{p(x)}$ suspends immediately. r will always terminate within $k = 2$ steps. s, however, may or may not terminate depending on the non-deterministic case selection. In fact, assuming that each case has an equal chance of selection, then s has a probability of $1 - \frac{1}{2^k}$ of terminating within k steps. Thus, profiling the program with different test data can lead to very different termination behaviour. This is the weakness of profiling. It is also strength of profiling. Case selection probabilities abstract away from the synchronization and non-deterministic behaviour of the program. Probabilities give a high level view of the way computation paths can flow through a program. In fact probabilistic information is almost certainly necessary as well as sufficient for soft deadline analysis: without probabilistic information the only useful termination property would be eventual reachability. Thus, henceforth, we assume that the different rules that define an agent are annotated with selection probabilities. We borrow a notation from [3] and express a program and a profile together as:

$$Q = \left\{ \begin{array}{ll} \mathsf{p(x)} \text{ :- } \mathsf{x = a} \mid \frac{1}{2} \rightarrow \mathsf{p(x)} & \mathsf{q} \text{ :- } \mathsf{true} \mid 0 \rightarrow \mathsf{p(x)} \\ \qquad + \mathsf{x = a} \mid \frac{1}{2} \rightarrow \mathsf{stop} & \mathsf{r} \text{ :- } \mathsf{true} \mid 0 \rightarrow (\mathsf{tell(x = b)} \parallel \mathsf{p(x)}) \\ \qquad + \mathsf{x = b} \mid 0 \rightarrow \mathsf{stop} & \mathsf{s} \text{ :- } \mathsf{true} \mid 1 \rightarrow (\mathsf{tell(x = a)} \parallel \mathsf{p(x)}) \end{array} \right.$$

In this case the profile is for when the top-level agent agent is s. Note that we use probabilities descriptively rather than prescriptively as in [3]. Note also that only non-zero frequencies are normalised and that q and r are annotated with probabilities of zero.

3 Off-line termination component

The program Q takes a special form in that the left-hand-side of each rule contains at most one user-defined agent. We can exploit this and apply an algorithm that appears in the model checking literature [6]. Algorithm 1 of [6] can be reinterpreted as a way calculating the probability of an agent reducing to stop within, say $k = 4$, steps. We let p_n (respectively q_n, r_n, s_n and $stop_n$) denote the probability of reaching the agent p (respectively q, r, s and stop) at step n. Thus, assuming our initial agent is s, we have $p_0 = 0$ $q_0 = 0$, $r_0 = 0$, $s_0 = 1$ and $stop_0 = 0$. More generally, for $n > 0$, we have the system:

$$p_{n+1} = \frac{1}{2}p_n + 0q_n + 0r_n + s_n, \quad stop_{n+1} = \frac{1}{2}p_n, \quad \begin{array}{l} q_{n+1} = 0 \\ r_{n+1} = 0 \\ s_{n+1} = 0 \end{array}$$

The equation $p_{n+1} = \frac{1}{2}p_n + 0q_n + 0r_n + s_n$, for example, expresses the probability of reducing to p (at step $n + 1$) given the likelihood of reducing to either p, q, r or s (at step n). Computing p_n, q_n, r_n, s_n and $stop_n$ for $1 \leq n \leq k$ and then $\sum_{n=1}^{n=k} stop_n$ gives the probability of terminating in at most n steps. For program Q, with $k = 4$, $\sum_{n=1}^{n=4} stop_n = \frac{7}{8}$ as is shown below.

n	p_n	q_n	r_n	s_n	$stop_n$
0	0	0	0	1	0
1	1	0	0	0	0
2	$\frac{1}{2}$	0	0	0	$\frac{1}{2}$
3	$\frac{1}{4}$	0	0	0	$\frac{1}{4}$
4	$\frac{1}{8}$	0	0	0	$\frac{1}{8}$

Also, not surprisingly, $\sum_{n=1}^{\infty} \mathit{stop}_n = 1$. As [6] point out, their algorithm is attractive as the total number of multiplications required in the (off-line) analysis is just $O(k.r)$ where r is the total number of cases that are annotated with a non-zero probability. In Q, for example, $r = 3$.

The limitation of the algorithm is that it is only applicable when the program assumes a particularly simple syntactic form. The analysis cannot be applied, for instance, to the program R that is listed below. We thus extend the method of [6].

$$R = \left\{ \begin{array}{l} \mathsf{p(x)} \;\text{:-}\; \mathsf{true} \mid \frac{2}{3} \rightarrow (\mathsf{p(x)} \parallel \mathsf{p(x)} \parallel \mathsf{p(x)}) \\ \qquad\quad + \;\mathsf{true} \mid \frac{1}{3} \rightarrow \mathsf{stop} \end{array} \right.$$

Consider the likelihood of the agent p terminating at exactly step n. p can only terminate in $n = 1$ steps through case 2 and hence $p_1 = \frac{1}{3}$. Otherwise, if $n > 1$, then the first, second and third agents of the composition in case 1 must terminate in $i, j, k > 0$ steps where $i + j + k = n$. Case 1 has a probability of $\frac{2}{3}$ of being selected which leads the system:

$$p_1 = \frac{1}{3}, \quad p_{n+1} = \frac{2}{3} \underset{\wedge\, i+j+k=n}{\sum_{i=1}^{i<n} \sum_{j=1}^{j<n} \sum_{k=1}^{k<n}} p_i \, p_j \, p_k$$

The nested summand means that the cost of computing p_n for $1 \leq n \leq k$ is $O(\sum_{n=1}^{k} n^2) = O(k^3)$. (The index k of the innermost summand is fixed by the i and j in the outermost summands.) More generally, if each rule contains a composition of at most m agents, then the complexity would be $O(\sum_{n=1}^{k} n^m)$ $= O(k^{m+1})$. Note, however, that p_n can be expressed as a double summand by introducing t_n that can itself be expressed as a double summand.

$$p_1 = \frac{1}{3}, \quad t_n = \underset{\wedge\, i+j=n}{\sum_{i=1}^{i<n} \sum_{j=1}^{j<n}} p_i \, p_j, \quad p_{n+1} = \frac{2}{3} \underset{\wedge\, i+j=n}{\sum_{i=1}^{i<n} \sum_{j=1}^{j<n}} t_i \, p_j$$

This reduces the complexity to $O(\sum_{n=1}^{k} n) = O(k^2)$. More generally, a nested summand $\sum_{i_1=1}^{i_1<n} \sum_{i_2=1}^{i_2<n} \cdots \sum_{i_m=1}^{i_m<n} \prod_{j=1}^{j=m} p^j_{i_j}$ where $p^1, \ldots, p^m$ are the m agents, can be decomposed into $m-1$ double summands by introducing $m-2$ temporary variables. Using this tactic we obtain a complexity of $O(mk^2)$. We are really introducing a form of memoisation since t avoids recomputation. The usefulness of memoisation is illustrated below. c_{2^k} denotes the number of multiplications required to compute $\sum_{n=1}^{n=2^k} p_n$. The $\log_2 c_{2^k}$ column indicates that memoisation is required for the analysis to be quadratic.

	naive		*memoisation*	
k	c_{2^k}	$\log_2 c_{2^k}$	c_{2^k}	$\log_2 c_{2^k}$
2	5	2.3	15	3.9
3	77	6.3	63	6.0
4	925	9.9	255	8.0
5	9021	13.1	1023	10.0
6	79485	16.3	4095	12.0
7	666877	19.3	16383	14.0
8	5462525	22.4	65535	16.0
9	44217341	25.4	262143	18.0
10	355821565	28.4	1048575	20.0

n	p_n	t_n
1	0.3333	0.0000
2	0.0000	0.1111
3	0.0000	0.0000
4	0.0247	0.0000
5	0.0000	0.0165
6	0.0000	0.0000
7	0.0055	0.0000
8	0.0000	0.0043
9	0.0000	0.0000
10	0.0016	0.0000

In fact $\sum_{n=1}^{\infty} p_n \approx 0.366025$ and indeed, we would expect $\sum_{n=1}^{\infty} p_n$ to be slightly larger than $\frac{1}{3}$.

4 Related work

A Markov model for probabilistic concurrent constraint programming in presented in [3] which, we believe, might form another basis for a probabilistic soft deadline analysis. By using the short-circuit protocol [9], an observation $\langle x = y, p \rangle$ for an agent like $q(x, y)$ could be reinterpreted as expressing the probability p of q (eventually) terminating. Unfortunately, the constraint systems of [3] are required to be finite and it is not clear that a termination analysis based on an (infinite dimensional operator algebra) extension would be practical.

Generating functions are used in [2] to quantify the relative efficiency of backtracking search and parallel search. Generating functions have been used in the average case analysis of imperative programs and so it might be possible to apply this approach to soft deadline analysis. Parellisation is also the focus of the time-complexity work in [7, 10]. These papers describe criteria for recognising short-lived agents so that fine-grained processed can be coalesced into more course-grained units. Neither papers describe probabilistic techniques.

The most closely related work is that described in [6]. This paper extends the temporal logic CTL with time and probability to reason about properties such as the probability of a service being carried out within a certain time. Like our work, the objective is to verify the likelihood of satisfying soft deadlines. Our work shows how to generalise algorithm 1 of [6].

5 Future work

An interesting direction for future work is to try and infer probabilistic estimates on the size of the store. Another challenge will be to put probabilistic bounds on the time-complexity of, say, branch-and-bound for a particular class of input data. We think that it is unlikely that estimated profiles will be able to deliver an accurate soft deadline analysis but the experiment should be attempted for completeness. We also intend to investigate how our work can be used in granularity control [7, 8, 10].

6 Discussion and conclusions

The problem with using profiling based analyses, as [11] points out, is that a profile for one run may not adequately predict the behaviour of another run. Put simply, the distributions inferred for the analysis may not accurately characterize the behaviour of the program for another data set. One way forward is to decouple the on-line and off-line components of the analysis so that the frequency information can be acquired from several program runs with different sample data. By enriching the analysis with intervals it would it possible to characterize the probability of case selection as a range, $[0.3, 0.33]$, say. Furthermore, with an appropriate GUI, the user would be able to alter/extend/contract the intervals. The off-line component could then be recomputed to construct a what-if style probabilistic profiling tool.

To summarize, although our approach relies on profiling information, we believe that its simplicity and good complexity mean that it is a sensible and useful first step in inferring the likelihood of an agent meeting a deadline.

Acknowledgements

We thank Kish Shen for motivating discussions; the EPSRC grants GR/MO8769 and GR/L95878 that funded the work; and "COTIC" European working group 23677 that funded the travel.

References

1. S. Debray, S. Kannan, and M. Paithane. Weighted Decision Trees. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 654–668. MIT Press, 1992.
2. N. Dershowitz and N. Lindenstrauss. Average Time Analyses Related to Logic Programming. In *Proceedings of the Sixth International Conference on Logic Programming*, pages 369–381. MIT Press, 1989. Also available from http://www.cs.huji.ac.il/ naomil/.
3. A. Di Pierro and H. Wiklicky. A Markov Model for Probabilistic Concurrent Constraint Programming. In *To appear in proceedings of APPIA-GULP-PRODE'98, Joint Conference on Declarative Programming*, 1998. Also available from http://www.soi.city.ac.uk/ adp/.
4. I. Foster and S. Taylor. *Strand: New Concepts in Parallel Programming.* Prentice-Hall, 1989.
5. M. M. Gorlick and C. F. Kesselman. Timing Prolog Programs Without Clocks. In *Proceedings of the Symposium on Logic Programming*, pages 426–432. IEEE Computer Society, 1987.

6. H. Hanson and B. Jonsson. A Logic for Reasoning about Time and Reliability. *Formal Aspects of Computing*, 6:512–535, 1994.
7. A. King, K. Shen, and F. Benoy. Lower-bound Time-complexity Analysis of Logic programs. In *International Symposium on Logic Programming*, pages 261–276. MIT Press, 1997. http://www.cs.ukc.ac.uk/pubs/1997/506/.
8. V. Sarkar. Determining Average Program Execution Times and their Variance. In *SIGPLAN Conference on Programming Language Design and Implementation*, pages 298–312, 1989.
9. E. Shapiro. The Family of Concurrent Logic Programming Languages. *ACM Computing Surveys*, 21(3), 1989.
10. K. Shen, V. Santos Costa, and A. King. Distance: a new metric for controlling granularity for parallel execution. *Journal of Functional and Logic Programming*, To appear, 1998. http://www.cs.ukc.ac.uk/pubs/1998/588.
11. D. W. Wall. Predicting Program Behaviour Using Real or Estimates Profiles. In *SIGPLAN Conference on Programming Language Design and Implementation*, pages 59–70, 1991.

This article was processed using the LaTeX macro package with LLNCS style