Applications of wait/lock-free protocols to real-time systems

Research Plan for Yi Zhang

Introduction

The research goal of the project titled ``Applications of wait/lock-free protocols to real-time systems'' is to explore research advances in the area of wait/lock-free synchronisation and non-blocking concurrent data structures implementation and apply them for gaining in efficiency in the OS kernel level; the target is to test our results on a commercial real-time OS (ENEA OSE).

Wait/lock-free inter-process communication/coordination permits access to concurrent objects without the use of locking; thus, it offers guarantees not only regarding priority inversion (it eliminates this problem altogether), but also regarding *efficiency* (by allowing maximum concurrency and thus, low completion times) and *fault-tolerance* (no task is blocked due to a task that crashed while holding a lock). The **wait-free** condition guarantees progress and completion for every job (of every task) regardless of the execution speeds (and priorities) of the other tasks in the system. **Lock-free** implementations have a more relaxed requirement: locking is ruled out, as in wait-free implementations, but repeated interferences (e.g. under extreme load conditions of the system) may cause a given operation to take longer time to complete. Comparing the two, lock-free methods imply less-overhead (i.e. in memory requirements), while wait-free methods guarantee fully predictable behaviour for every task.

Our so-far experience with non-locking (lock/wait-free) synchronisation has shown that it can offer significant benefits in practice and ---more significantly--- it is efficient alternative to lock-based synchronisation methods for managing concurrency and access to shared resources. However, our experience has also shown that, although, **lock-free** communication protocols, as opposed to **wait-free** protocols, do not guarantee bounded response time (e.g. under extreme load conditions of the system), in practice their performance is stable and experimentally "predictable" [2,3]. Moreover, they are faster and they require less space than the respective wait-free ones. Last, but not least, from the software engineering standpoint, lock-free communication protocols are easier to implement than the wait-free ones.

Goals and Plan

The research plan for Yi's next year and a half is to try to show that the lack of predictability of lock-free objects can be solved in an efficient way.

The approaches that we are going to use in order to reach that goal are the following:

- We will try to develop a general framework for real-time synchronization in the form of a class of protocols that can be used on-top of (together with) any lock-free communication protocol in

order to guarantee fully predictable behaviour for any task that is using the lock-free communication protocol. An efficient implementation of such a framework will augment the real-time characteristics of the lock-free synchronisation protocols without changing their simplicity and their good performance characteristics. More significantly, this will provide a general framework for real-time communication, like the one that the PIP and similar schemes provide, but, without the drawbacks that these schemes suffer from, drawbacks that come from the blocking that they invoke.

- As we mentioned above the empirical evidence that we have suggests that lock-free communication protocols perform quite stable and "predictable" in practical settings [2,3]. This empirical evidence motivates us to try to develop probabilistic analysis techniques for lock-free synchronisation protocols that will allow us to argue about the behaviour of the lock-free synchronisation protocols when the input of the algorithm is chosen from a probability distribution. Such techniques will offer a framework that will allow:

  i) engineers working with hard-real time systems to check the predictability of a lock-free protocols for their task set.

  ii) engineers working with soft-real time systems to check whether the frequency that the lock-free protocol becomes unpredictable (deadline miss) is acceptable for their applications.

Recently, when designing the Real Time Specification for JAVA (RTSJ) [1], the Real Time for Java Experts Group (RTJEG) noticed that wait-free synchronisation was the only way out from a priority inversion problem that was taking place during thread synchronisation in their RTSJ proposal. We were pleased to see that, of course. We looked at the Real-time Specification of JAVA and the wait-free classes required by the RTSJ and we noticed that the same synchronisation classes can be used also on the operating system level. We would like to look closer to the RTSJ and address the implementation issues that are introduced there. We think this line of research is very promising and is also of interest to our industrial partner ENEA OSE, who is member of the consortium for real time JAVA.

References

[1] G. Bollella, J. Gosling, B. Brosgol, P. Dibble, S. Furr, and M. Turnbull (2000). *The Real-Time Specfication for Java*. Java Series. Addison-Wesley, June 2000. URL: www.javaseries.com/rtj.pdf.

[2] Tsigas, P. and Y. Zhang (2001). *Evaluating The Performance of Non-Blocking Synchronisation on Modern Shared-Memory Multiprocessors.* ACM Sigmetrics 2001 / Performance 2001, ACM.

[3] Tsigas, P. and Y. Zhang (2001). *A Simple, Fast and Scalable Non-Blocking Concurrent FIFO queue for Shared Memory Multiprocessor Systems.* 13th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '01), ACM.