Cooperative Scheduling Behaviour from Modeling to Runtime

Vlad Serbanescu, Frank de Boer, and Mahdi Jaghouri

Leiden Institute of Advanced Computer Science
Centrum Wiskunde and Informatica
{vlad.serbanescu,frank.s.de.boer,jaghouri}@cwi.nl
http://www.cwi.nl

Abstract. Modeling applications at the design phase of any project is highly important in order to build a reliable, fast and robust system. Understanding the control flow of execution from just the model is crucial for the applications next software engineering phases such as implementation, testing, release and maintenance. If we add that a large portion software systems are running several jobs in parallel, with a good model we can observe data consistency and detect deadlocks efficiently. One of the toughest models to execute, especially in a parallel or distributed environment is the Actor-based model which introduces the notion of cooperative scheduling, a high-level synchronization mechanism which allows an actor to continue to execute messages from its queue when the current message execution is suspended . In this paper we will investigate some of the challenges of translating the cooperative scheduling behavior into the Java Runtime Environment. We will analyze the Abstract Behavioral Specification (ABS) Language concepts which are very well suited for Agent-based applications modeling and provide a benchmark for testing several solutions of efficiently running cooperative scheduling behavior.

Keywords: Cooperative Scheduling, Mult-Agent Systems, High-Performace Computing, Language Design

1 Introduction

-high-level description of cooperative scheduling
-position the paper with scala and Akka, state of the art
- problem statement: how stacks need to be saved in OO, how expensive a thread is
- strongly typed actors messages. - the scope of the paper is implementation in the Java Runtime Environment/Java Virtual MAchine(JVM) - using ABS as a mute programming language, real software development context, support for object-oriented design and foreign lanuage interface.

2 ABS Language Concepts

- the main concepts: async calls, await on boolean and futures, each object has its own queue. - example for coroutines (Paul Klint paper at SEN)

2.1 Proof of Concept: Actor-based implementation of Agent-based modelling

-expressive generic agent model
-software engineering context
- support for functional data types, FLI, type-checking
-sequence of events that maximize agent throughput

3 Cooperative Scheduling Implementation Schemes

-sequence diagram of what happens in an actor during synchronous and asynchronous method calls.

3.1 Modeling Language Concepts in the JVM

An Actor has a lock and Every Asynchronous call is a Thread The trivial straightforward approach for implementing cooperative scheduling in Java is to model each actor as an object with a Thread Pool with a queue and a single execution thread. Each asynchronous call would then generate a new thread with it own stack and context and put in the pool's queue. Whenever a control switch statement occurs the thread would be suspended by the JVM's normal behaviour. When the release condition is enabled the thread would compete with the thread pool and other suspended threads for a lock in order to execute on the actor.
We have one lock per Actor for ensuring single thread execution.

Every Actor is a Thread Pool -free queue, less space -Each async. call is modeled as task.

Every System has a Thread Pool -each actor is modeled as a task with a queue of lambda expressions. -particular uses of an executor service.

Synchronous calls context -to save execution context we save the current call stack as a Thread and optimally suspend it

Continuations as tasks - fully asynchronous environment
- modeling continuations as labeled functions and converting them to lambda expressions.
- allocating more memory in the heap and saving memory on the stack.

3.2 Optimizations for the JVM

Demand-driven Approach

Optimal Usage of System Threads

Eliminating Busy Waiting

Using JVM Garbage Collection

4 Continuation Generation at Compile Time

- Mahdi's blog post - formal explanation for creating continuations.

5 Runtime Behaviour

- the new Java backend

6 Benchmarking the Implementation Schemes

-multiple stack frame problem. - Old Java, vs Erlang vs New Java benchmark, vs possibly Optimized suspended threads.

7 Conclusions

References

1. Smith, T.F., Waterman, M.S.: Identification of Common Molecular Subsequences. J. Mol. Biol. 147, 195–197 (1981)
2. May, P., Ehrlich, H.C., Steinke, T.: ZIB Structure Prediction Pipeline: Composing a Complex Biological Workflow through Web Services. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 1148–1158. Springer, Heidelberg (2006)
3. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1999)
4. Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C.: Grid Information Services for Distributed Resource Sharing. In: 10th IEEE International Symposium on High Performance Distributed Computing, pp. 181–184. IEEE Press, New York (2001)
5. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: an Open Grid Services Architecture for Distributed Systems Integration. Technical report, Global Grid Forum (2002)
6. National Center for Biotechnology Information, `http://www.ncbi.nlm.nih.gov`

Appendix: Springer-Author Discount

LNCS authors are entitled to a 33.3% discount off all Springer publications. Before placing an order, the author should send an email, giving full details of his or her Springer publication, to `orders-HD-individuals@springer.com` to obtain a so-called token. This token is a number, which must be entered when placing an order via the Internet, in order to obtain the discount.

8 Checklist of Items to be Sent to Volume Editors

Here is a checklist of everything the volume editor requires from you:

- ☐ The final LaTeX source files
- ☐ A final PDF file
- ☐ A copyright form, signed by one author on behalf of all of the authors of the paper.
- ☐ A readme giving the name and email address of the corresponding author.