Programming and Optimizing Distributed Algorithms An Overview

Yanhong A. Liu Bo Lin Scott D. Stoller
Computer Science Department, State University of New York at Stony Brook
liu,bolin,stoller@cs.stonybrook.edu

***Abstract*— This paper gives an overview of a high-level language for programming distributed algorithms, compilation of the language to generate executable code, and powerful optimizations that incrementalize expensive synchronization conditions with respect to messages sent and received. The language, DistAlgo, allows distributed algorithms to be expressed easily and clearly, making it easier to understand them precisely and to generate executable implementations. Incrementalization transforms complex quantifications over sets of processes and sequences of messages sent and received into message handlers that perform low-level assignments and incremental updates, avoiding the need to write them manually. This drastically reduces local processing time complexities and reduces unbounded space used for message history sequences to auxiliary space needed for incremental computation. We have implemented a prototype of the compiler and the optimizations and experimented with a variety of well-known distributed algorithms. Our results strongly confirm the benefits of the language and the optimizations.**

Keywords- distributed algorithms; incrementalization; optimization; very high-level languages

I. MOTIVATION

Distributed algorithms are at the core of distributed systems, which are increasingly indispensable in our daily lives. Yet, developing practical implementations of distributed algorithms with correctness and efficiency assurance remains a challenging, recurring task.

- Study of distributed algorithms has relied on either pseudo-code with English, which is high-level but imprecise, or formal specification languages, which are precise but harder to understand, lacking mechanisms for building real distributed systems, or not executable at all.
- At the same time, programming of distributed systems has mainly been concerned with program efficiency and has relied mostly on the use of low-level or complex libraries and to a lesser extent on built-in mechanisms in restricted programming models.

What's needed is (1) a simple and powerful language that can express distributed algorithms at a high level and yet has a clear semantics for precise execution as well as verification, and is fully integrated into widely used programming languages for building real distributed systems, together with (2) powerful optimizations that can transform high-level algorithm descriptions into efficient implementations.

II. A VERY HIGH LEVEL LANGUAGE FOR CLEAR DESCRIPTION OF DISTRIBUTED ALGORITHMS

We have developed DistAlgo, a very high-level language for expressing distributed algorithms that combines advantages of pseudo-code, formal specification languages, and programming languages. We identified the following basic features needed for distributed algorithms, and designed DistAlgo to support them clearly and precisely:

(A) distributed processes that can send messages

(B) handling of received messages with support for atomicity

(C) waiting on conditions for synchronization, involving quantifications over sets of processes and history of messages, which are the deepest part of the algorithms

(D) configuration with library support

DistAlgo supports these features by building on an object-oriented language that supports high-level queries and quantifications over sets and sequences.

DistAlgo allows distributed algorithms to be expressed clearly at a high level, almost exactly like pseudo-code, but also precisely, like in formal specification languages, and be executed as part of real applications, as in programming languages. The main reason that algorithms written in DistAlgo are almost like pseudo-code is that complex synchronization conditions can be expressed using high-level quantifications over sets and sequences, including especially the history of messages sent and received.

Expressing synchronization conditions at such a high level allows the correctness of the algorithms to be proved much more easily. However, if executed straightforwardly, each quantifier will cause a linear factor in running time, and any use of the message history will cause space usage to be unbounded.

III. POWERFUL OPTIMIZATIONS FOR GENERATING EFFICIENT IMPLEMENTATIONS

The main challenges in generating efficient implementations are higher-level control structures and data types. The most expensive features, by far, are synchronization conditions expressed using quantifications over complex

978-1-4577-1591-4/11/$26.00 ©2011 IEEE

objects and high-level data types, which are most often the set of processes and the history of messages.

To address this problem, quantifications over the history of messages sent to and received from other processes must be performed incrementally as messages are sent and received. There has been much previous research on incrementalizing expensive computations, for set languages, recursive functions, logic rules, and objects, but not for general quantifications. Also, previously studied general methods are for centralized programs, not distributed programs.

Our method is to automatically transform each send and receive clause in the program into an update to the sequence for message history, incrementally maintain the truth values of synchronization conditions and necessary auxiliary values as the sequences of messages sent and received are updated, and finally remove those sequences as dead variables.

To incrementally maintain the truth values of general quantifications, our method first transforms them into set queries. However, translating nested quantifications simply into nested queries can incur asymptotically more space and time overhead than necessary. The key idea is to minimize the nesting of queries. This avoids maintaining unnecessary intermediate results, saving both time and space, and furthermore yielding simpler programs. Our incrementalization method also allows the time and space complexity of the generated programs to be easily analyzed. Overall, our method allows efficient implementations to be generated, without manually coding the algorithms and applying ad hoc optimizations. It can even lead to simplification of the original algorithms.

IV. IMPLEMENTATIONS AND EXPERIMENTS

We have implemented an initial prototype of DistAlgo by extending the Python programming language, because Python has rich support for very high-level constructs for ease of programming, and simple and consistent syntax for ease of reading; Python is also widely used in practice and increasingly used in teaching.

We have applied the method to a set of well-known distributed algorithms from main distributed algorithm textbooks and papers, including different variants of algorithms for distributed mutual exclusion, leader election, atomic commit, and Paxos and Byzantine Paxos for distributed consensus.

We have performed several sets of experiments, which helped show that (1) DistAlgo allows distributed algorithms to be expressed more easily and clearly, compared with other languages, (2) our optimizations improve the time and space asymptotically as analyzed, and (3) generated implementations, even though not yet optimized for constant factors, are not too far in performance from manually optimized programs.

Directions for future work include formal semantics for the language, verification, additional optimizations, and improved implementations.

978-1-4577-1591-4/11/$26.00 ©2011 IEEE