Distributed Watchpoints: Debugging Very Large Ensembles of Robots (Extended Abstract)

Michael De Rosa
Seth Goldstein
Peter Lee
School of Computer Science
Carnegie Mellon University
[mderosa,seth,petel]@cs.cmu.edu

Jason Campbell
Padmanabhan Pillai
Intel Research Pittsburgh
[jason.campbell,padmanabhan.s.pillai]@intel.com

***Abstract*— We describe a debugging tool for modular robotics that introduces the concept of distributed watchpoint triggers. This technique can initiate debugging actions (system halt, global snapshot, logging, etc.) in an ensemble of robots based on temporal, physical, and logical conditions distributed over multiple robots. Our technique is specifically designed to be effective in debugging modular robotic ensembles, where many important types of failure conditions can be detected within small, physically connected subsets of the total ensemble.**

I. Introduction

The task of debugging algorithms on modular robotic ensembles presents a number of challenges, because modular robots are parallel elements operating asynchronously. Unlike traditional parallel processing nodes, modular robots operate in a bandwidth-limited, highly reconfigurable network. As a result, many of the traditional tools for parallel debugging are impractical in the context of modular robotics. Debugging a modular robotic ensemble is, at least in the general case, much more difficult than debugging a super-computing cluster or multicore CPU of comparable scale.

II. Motivation & Approach

We need tools that mitigate the complexities inherent in programming robotic systems with thousands to millions of autonomous agents. One of the key elements to this is debugging support, both for simulated ensembles and real hardware. When examining the current state-of-the-art in distributed and parallel debuggers, one finds them inadequate for the task of debugging large ensembles of robots. Most debugging tools, such as gdb [1] and TotalView [2] (a distributed debugger), focus on analysis of one thread at a time. They provide little or no assistance in locating fault conditions that may span multiple threads. Even for a multithreaded simulator running on one machine, gdb can take several hours to load the contexts for the thousands of threads used in large simulations. In the absence of better tools, programmers are forced to fall on back on techniques such as `printf`, which can generate confusing output, lead quickly to information overload, and may even introduce inadvertent synchronization into the system through the presence of I/O flush and locking routines.

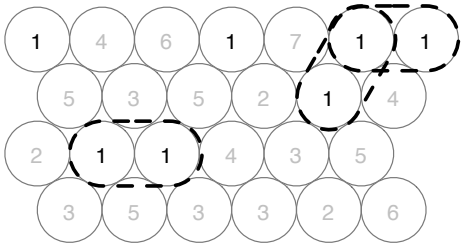

Fig. 1. An ensemble of modular robots with an active distributed watchpoint. State values are shown as the contents of each circle, and dotted regions represent subsets of modules that match the watchpoint:
neighbors(x,y) && (x.current.var == 1) && (y.current.var == 1)

Distributed debugging presents a number of challenges, including coherent snapshotting, specification of error conditions, combinatorial state explosion, and detection of timing-sensitive errors. A true distributed debugger must solve many problems to be useful. It must be able to operate as if it possessed a coherent picture of the ensemble's global state, when the modules are operating independently and asynchronously. To detect error conditions that may involve arbitrary nodes spread throughout the ensemble, it faces a combinatorial explosion of states that must be searched. Finally, a distributed debugging system must minimize the effect that its operation will have on timing-sensitive errors, or heisenbugs.

We introduce a useful construct for such debugging tasks which we call a distributed watchpoint: a triggering mechanism for some arbitrary action, based on a function of the states of several physically connected modular robots. The action that a distributed watchpoint can trigger is deliberately decoupled from the trigger mechanism, and can include such actions as halting the system, triggering a global snapshot, or enabling some expensive logging functionality on the triggering modules. The functions over module state and connectivity that serve as trigger conditions are expressed in terms of simple predicates over state variables and boolean connectivity relations. This provides an easy to use, powerful technique for identifying multi-robot errors in ensembles of modular robots.

III. Related Work

Existing literature on distributed debugging provides many useful techniques that have varying degrees of applicability to modular robotics. Message-tracing algorithms [3] allow one to trace the causal effect of messages on multiple programs, which is sufficient if such message passing is the only causal channel between processes. (However, note that because modular robots have both physical and network interactions, such is not the case.) Global snapshot algorithms [4], [5] can provide the ability to obtain a consistent global view of a distributed system's state, but require time proportional to the number of nodes [6], making them unsuitable for continuous use in debugging very large ensembles of robots. (Although, global snapshots *can* be useful as a trigger action for watchpoints.) Finally, global predicate evaluation algorithms [7], [8] allow one to determine if an entire ensemble matches a boolean predicate expression. While many of the techniques used to implement global predicate evaluation may be of use to us, the problem of distributed watchpoints translates to one of evaluating multiple overlapping local predicates, rather than one global predicate.

IV. Design

In designing our distributed watchpoints, we consider a simplified machine model for each modular robot: each robot is represented by a number of named state variables, and an array of neighbors. We assume that each robot is an asynchronous state machine that iterates through three atomic phases: computation, state variable assignment, and communication.

As a general-purpose distributed debugging system poses many challenges, we consider what classes of errors a limited debugger should detect in order to provide a useful service to developers of modular robotic applications. Errors that are confined to a single module can be easily detected with a debugging or monitoring process that runs concurrently with the main thread of execution on each robot. Detecting errors that span multiple modules requires the capability to capture and reason about the state of multiple robots over time. To make this problem more tractable, we take advantage of certain common assumptions about modular robotic systems. First, we assume that all robots are running identical programs, though each copy of the program may be in a different state. We also assume that robots can only communicate with their neighbors, which greatly limits the complexity of the network. As an additional limitation, we assume that most errors can be detected by physically connected subsets of the ensemble. This dramatically reduces the search space for the debugger, and removes the requirement to gather globally consistent data at some centralized location.

We use a simple grammar for the description of watchpoint trigger criteria, which is inspired by a subset of linear temporal logic. This description language allows programmers to specify, at run time, what distributed condition they wish to detect. The language consists of comparison predicates over named state variables, temporal modal operators to provide access to a module's previous and future state, module-module connectivity tests, and boolean connectives. With these simple constructs, we can describe target configurations along the axes of network connectivity, logical state, and temporal separation.

To implement distributed watchpoints, we first convert the textual representation of the target function to a simple pattern-matching automata. In our multi-robot simulator, we allow each robot to carry a set of such pattern matchers, representing the robot's potential membership in multiple instances of the distributed criteria. At each simulator timestep, the system determines if each pattern matcher should spread to a module's neighbors, match the condition, or be deleted.

Finally, to lower the memory and computational costs associated with the watchpoint system, we have developed a number of optimizations that reduce the number of active pattern matchers in the system, and limit how fast they spread.

V. Conclusion

We have described the motivation and initial design of a distributed debugging tool for modular robotic ensembles. With such distributed watchpoints, programmers can more easily debug error conditions which involve incorrect state configurations over multiple modules. We are currently implementing distributed watchpoints in our multi-robot simulator, and plan to create a fully distributed version of the algorithm for use in real ensembles of robots.

References

[1] GDB: The GNU Project Debugger. [Online]. Available: http://www.gnu.org/software/gdb/

[2] Etnus TotalView. [Online]. Available: http://www.etnus.com/

[3] R. Wismüller, "Debugging message passing programs using invisible message tags," in *Proceedings of the 4th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*. London, UK: Springer-Verlag, 1997, pp. 295–302.

[4] O. Babaoglu and K. Marzullo, "Consistent global states of distributed systems: Fundamental concepts and mechanisms," in *Distributed Systems*, S. Mullender, Ed. Addison-Wesley, 1993, pp. 55–96.

[5] Z. Yang and T. A. Marsland, "Global snapshots for distributed debugging," in *International Conference on Computing and Information*, 1992, pp. 436–440.

[6] C. M. Chase and V. K. Garg, "Detection of global predicates: Techniques and their limitations," *Distributed Computing*, vol. 11, no. 4, pp. 191–201, 1998.

[7] E. Fromentin, M. Raynal, V. K. Garg, and A. I. Tomlinson, "On the fly testing of regular patterns in distributed computations," in *International Conference on Parallel Processing*, 1994, pp. 73–76.

[8] V. K. Garg and B. Waldecker, "Detection of weak unstable predicates in distributed programs," in *Zhonghua Yang and T. Anthony Marsland (Eds.), Global States and Time in Distributed Systems, IEEE Computer Society Press*, 1994.