FULLY-DISTRIBUTED DEBUGGING AND VISUALIZATION OF DISTRIBUTED SYSTEMS IN ANONYMOUS NETWORKS

Cédric Aguerre, Thomas Morsellino and Mohamed Mosbah
LaBRI, CNRS, Université de Bordeaux 351, Cours de la Libération, 33405 Talence, France

Keywords: Distributed Algorithm, Visualization, Debugging, Anonymous Network, Snapshot, Global Predicate Evaluation.

Abstract: The debugging of distributed algorithms is a major challenge which greatly benefits from the help of an interactive and informative human-computer interface. In this paper we present ViSiDiA, a platform for the visualization, simulation and debugging of distributed algorithms. Our approach respects real-life constraints such as process anonymity and privacy, network synchronicity. We propose a new fully-distributed method for the debugging and monitoring of distributed systems, based on the computation of global states and global predicates from local information in anonymous and asynchronous networks. We show how the debug information can be visualized concurrently with the algorithm execution.

1 INTRODUCTION

The analysis and understanding of distributed algorithms involved in complex information systems are fundamental. These algorithms must be proved, implemented, debugged and tested in a context where several processes collaborate to the execution of a same task. Main issues concern concurrent access to resources, critical failure detection, or even process communication strategy.

In recent years, several tools help in assessing the question of simulation and visualization of distributed algorithms (Moses et al., 1998; Stasko and Kraemer, 1993; Koldehofe et al., 2003; Ben-Ari, 2001; Carr et al., 2003; Pongor, 1993; Chang, 1999). Most of them consider that processes have unique identifiers or have particular knowledge on the network. However in large and heterogeneous networks, processes may not have unique identifiers or may not wish to divulge them for privacy reasons (Guerraoui and Ruppert, 2005). We thus focus on fully-distributed debugging issues in anonymous networks.

In this paper, we expose a new design of ViSiDiA, a tool for simulating and visualizing distributed algorithms. We add debugging features in both simulation and visualization parts. We are interested in a way of visualizing debug information along with algorithm execution, and we present our theoretical approach for debugging.

2 THE VISIDIA PLATFORM

Concept. ViSiDiA[1] (Visualization and Simulation of Distributed Algorithms) is a tool aiming at simulating and visualizing the execution of distributed algorithms (Bauderon et al., 2001; Derbel and Mosbah, 2003; Bauderon and Mosbah, 2003), used as an educational and research utility.

Distributed networks can be defined using an editor in ViSiDiA. Algorithms are run along with a visualization which respects events sequencing. The user can interact with the network whilst the simulation executes. This all makes it possible to study algorithms, to detect errors or to compute complexity.

ViSiDiA encompasses a simulation core, a Graphical User Interface (GUI), and an Application Programming Interface (API) to implement distributed algorithms thanks to a set of simple primitives.

Illustrative Example. For the sake of clarity, we here illustrate visual components using a standard Broadcast algorithm applied to a small distributed network with a message passing model (Yamashita and Kameda, 1996). The network is modeled as a graph, whose nodes and edges correspond to autonomous processes and to communication links, respectively. The graph is such that one node is labeled A, all the others being labeled N. We call *X*-

[1] http://visidia.labri.fr

Aguerre C., Morsellino T. and Mosbah M..
FULLY-DISTRIBUTED DEBUGGING AND VISUALIZATION OF DISTRIBUTED SYSTEMS IN ANONYMOUS NETWORKS.
DOI: 10.5220/0003861807640767
In *Proceedings of the International Conference on Computer Graphics Theory and Applications* (IVAPP-2012), pages 764-767
ISBN: 978-989-8565-02-0

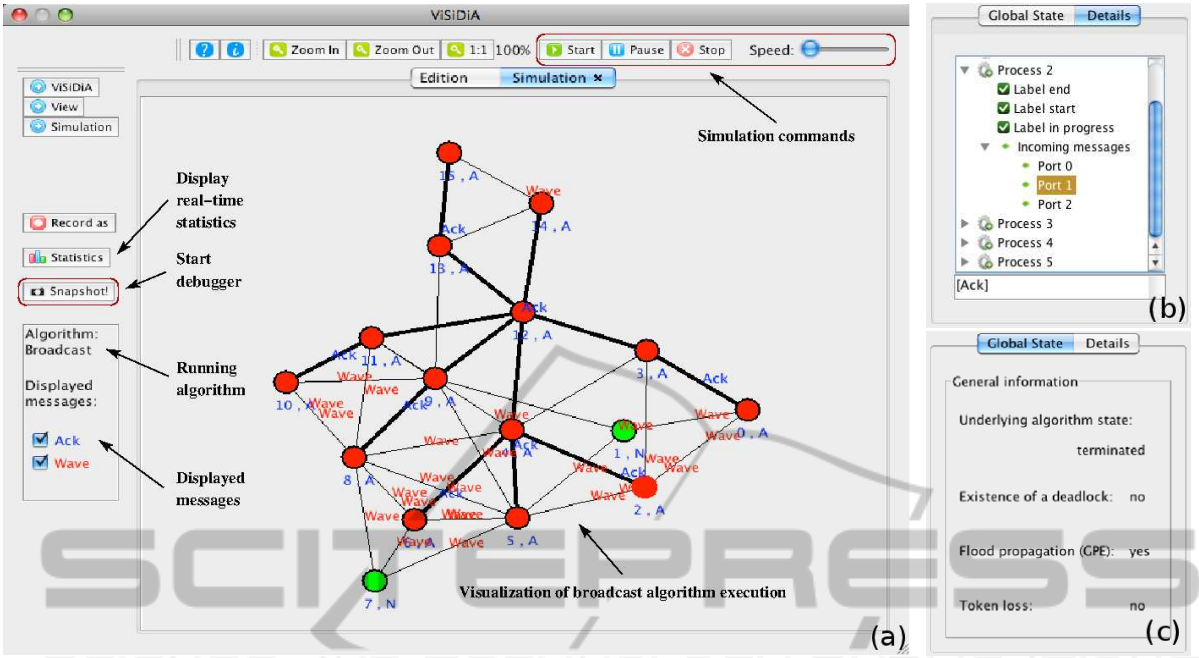

Figure 1: Overview of the visual interface. (a): Settings are accessible on top and left panels. Algorithm executes on the network. The user can observe changes on node and edge states, as well as transiting messages. A button on left panel launches the debugger. (b-c): Debugging results displayed during algorithm execution. (b): monitoring process 2 with the state of its incoming channels. (c): some evaluated predicates such as algorithm termination.

node a node with label *X*.

The algorithm is then the following. At any time, each A-node sends a message to each of its neighbors, and each N-node receiving a message changes its label to A and propagates the message to its neighbors. When a message is delivered, the edge between the sender and the receiver takes a special *marked* state.

The algorithm terminates when no N-nodes remain. All *marked* edges and connected nodes form a spanning tree of the initial graph.

Visual Semantics. Nodes are circles whose inner part represents a label according to a user-modifiable color palette. In Figure 1(a), inner parts of A-nodes and N-nodes are filled with red and green, respectively. A node outer part indicates if the node is selected (red) or not (black). Below a node appear its properties: an id, a label, or any user-defined value. The user controls which information is displayed.

Edges are line segments, or arrows if they are oriented. Default visualization uses thin black lines. Thick lines indicate *marked* edges. A selected edge is red, additional colors representing user-defined states. Other properties, if any, appear close to the edge.

Messages are textual information sliding from a sender node to a receiver node. Different colors are used according to the message types defined by the running algorithm. In the case of Broadcast algorithm, there are wave messages (in red) and acknowledgements (in blue). The user can switch on and off message display during the simulation.

Dynamic changes in color, position and thickness of graph elements give an instantaneous global perception of the running algorithm. To access local or detailed information, simulation speed can be adjusted or the simulation can be paused, edge/node properties can be modified, the message display can be adapted to fit user requirements.

The Simulation Core. A node owns an autonomous thread which only operates on its own properties and its connected edges. As the network is anonymous each process holds a copy of the simulated algorithm.

A unique simulation console manages requests from processes. For example a process can ask it for sending a message. The console is then responsible for delivering the message, pushing it into the receiver FIFO-based mailbox. An event/acknowledgement system ensures the order of requests.

2.1 Monitoring and Debugging Feature

Message Passing. Consider a distributed system in which a process crashes: the corresponding thread is deadlocked, whereas its neighbors are still waiting

for its termination. We want our debugger to propagate the information “a neighbor has crashed”. As the debugging operator cannot be the deadlocked thread itself, we associate to each process a second thread used for debugging.

A process owns two threads: one for algorithm execution, the other for debugging. In the console now transit both execution and debug messages. A process distributes each message according to its type to the appropriate thread. The debugging thread monitors both the algorithm thread and its incoming messages. If the algorithm fails, all messages are processed by the debugging thread (Figure 2).

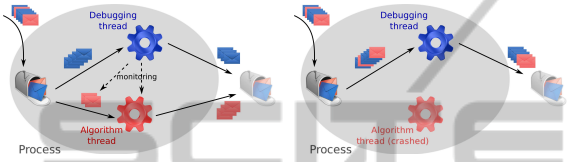

Figure 2: Message passing at process scale. A process contains a mailbox and two threads: one for algorithm execution, the other for debugging. Left: In normal case, messages are routed according to their type. Right: If the algorithm crashes, the debugging thread takes over.

Visual Components. The ViSiDiA graphical interface contains a button to launch the debugger during an algorithm execution. Debug information shows up as a tree view for processes and incoming channels (Figure 1(b)), and some evaluated global predicates are listed in another panel (Figure 1(c)). This information can also be visualized when the mouse hovers over a node. See Section 3 for details on debug information nature.

API Extension. Debugging features have been added to the ViSiDiA API. Algorithm developers can thus monitor the value of specified variables and test if a global predicate occurs, just adding a few lines of code. In the case of Broadcast (Algorithm 1), we follow changes in the value of processes label (*registerVariable* method). We have created a global predicate *sp* and we tell the debugger to use it (*addGlobalPredicate* method). Finally, we want a feedback on algorithm termination (*setTerminated* method).

3 DEBUGGING ALGORITHMS

A debugger needs network snapshots, composed of both processes and communication channels states. Such global snapshots are computed using only local information processes exchange. These snapshots are then exploited to evaluate global predicates (GP, for short), i.e., properties which remain true as soon as they are verified (Tel, 2000). The main motivation for GP evaluation is to react against particular situations which can occur in distributed systems.

Our solution (Chalopin et al., 2011) is a combination of the Chandy-Lamport algorithm (Chandy and Lamport, 1985) with an algorithm by Szymanski, Shy, and Prywes (Szymanski et al., 1985) to compute snapshots and to evaluate GP anonymously.

Computing Snapshots. The Chandy-Lamport algorithm determines global snapshots in which each process has computed its local snapshot within finite time. Once local snapshots are computed, this knowledge is fully distributed over the system then exploited by processes. From this knowledge, one can simulate a global clock (Raynal, 1988), nevertheless it does not enable iterated computation of snapshots. Another way to exploit this knowledge is based on wave algorithms: a message is passed to each process by a single initiator according to the network topology (Matocha and Camp, 1998). These solutions are not available in the context of anonymous networks with no distinguished processes and no particular knowledge on the topology.

```
begin
    static Message wave = new Message(”Wave”, true);
    static Message ack = new Message(”Ack”, true, Color.blue);
    int arity = getArity();
    String label = getProperty(”label”);
    registerVariable(”label start”, label);
    addGlobalPredicate(sp);
    if label.compareTo(”A”) == 0 then
        for neighbor = 0 to arity-1 do
            sendTo(neighbor, Broadcast.wave);
    else
        Door door = receiveMessage();
        int doorNum = door.getNum();
        sendTo(doorNum, Broadcast.ack);
        putProperty(”label”, new String(”A”));
        registerVariable(”label in progress”, label);
        setDoorState(new MarkedState(true), doorNum);
        for neighbor = 0 to arity-1 do
            if neighbor != doorNum then
                sendTo(neighbor, Broadcast.wave);
    registerVariable(”label end”, label);
    setTerminated(true);
```

Algorithm 1: Example of a Broadcast algorithm written in Java using the ViSiDiA API. In black, the algorithm as written without any debug procedure. In blue, the only 5 lines of code needed to enable debugging this algorithm on the visual interface.

Termination of the Chandy-Lamport Algorithm: Checkpoints. The algorithm by Szymanski, Shy, and Prywes (the SSP algorithm, for short) detects when each process has reached its termination condition. Running both the Chandy-Lamport and the SSP algorithms, each process can thus locally and anonymously detect when all processes have computed their local snapshot.

This algorithms combination gives us a fundamental debugging feature: it defines checkpoints for the distributed algorithm of interest. First in case of system failure, the computation can be restarted from the last valid checkpoint. Second our debugger can offer step-by-step forward and rewind functionalities.

Global Predicates. From execution checkpoints, we can run once again the SSP algorithm to evaluate GP (graph invariants). The most obvious predicate is the termination detection of the monitored algorithm. Hence, in ViSiDiA we obtain a semi automatic debugging; user control is required to react against GP evaluation (e.g., a detected predicate could be a system failure).

A more elegant approach is, still from the execution checkpoints, to apply an adaptation of the algorithm by Mazurkiewicz (Mazurkiewicz, 1997) which gives a distributed way to compute graph coverings. More precisely, each process can compute a graph from which the network graph is a covering. From this graph, predicates can be locally analyzed and verified; processes can then automatically react against any system state.

4 CONCLUSIONS

In this paper, we presented a new design of the ViSiDiA platform for the simulation and visualization of distributed algorithms. We added debugging features with a fully-distributed approach in the context of anonymous and asynchronous networks. These are made effortless accessible to users: the ViSiDiA API contains new primitives, and the GUI offers visualization of debugging information along with algorithm execution. We also introduced a new method to build our debugger.

Our proposal helps in monitoring a distributed system, determining its global state from local information and detecting failures. We set a checkpoint and rollback recovery system, and implemented a semi automatic debugger. User oversight can be released computing local graph coverings.

We plan to focus on this technique, and to visualize the graphs within each process in a multi-scale approach. Finally, our theoretical basis can be extended to rewriting rules and mobile agents.

REFERENCES

Bauderon, M., Gruner, S., Métivier, Y., Mosbah, M., and Sellami, A. (2001). Visualization of distributed algorithms based on labeled rewriting systems. In *GT-VMT'01*, volume 50 of *ENTCS*, pages 229–239.

Bauderon, M. and Mosbah, M. (2003). A unified framework for designing, implementing and visualizing distributed algorithms. *ENTCS*, 72(3):13 – 24.

Ben-Ari, M. (2001). Interactive execution of distributed algorithms. *J. Educ. Resour. Comput.*, 1.

Carr, S., Fang, C., Jozwowski, T., Mayo, J., and Shene, C.-K. (2003). Concurrent mentor: A visualization system for distributed programming education. In *PDPTA'03*.

Chalopin, J., Métivier, Y., and Morsellino, T. (2011). On snapshots and stable properties detection in anonymous fully distributed systems. *submitted*.

Chandy, K. M. and Lamport, L. (1985). Distributed snapshots: Determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1):63–75.

Chang, X. (1999). *Network simulations with OPNET*, pages 307–314. ACM.

Derbel, B. and Mosbah, M. (2003). Distributing the execution of a distributed algorithm over a network. In *INFOVIS'03*, pages 485 – 490.

Guerraoui, R. and Ruppert, E. (2005). What can be implemented anonymously? In *DISC*, pages 244–259.

Koldehofe, B., Papatriantafilou, M., and Tsigas, P. (2003). Integrating a simulation-visualisation environment in a basic distributed systems course: a case study using lydian. In *ITiCSE'03*, pages 35–39. ACM.

Matocha, J. and Camp, T. (1998). A taxonomy of distributed termination detection algorithms. *Journal of Systems and Software*, 43(3):207–221.

Mazurkiewicz, A. (1997). Distributed enumeration. *Inf. Processing Letters*, 61:233–239.

Moses, Y., Polunsky, Z., Tal, A., and Ulitsky, L. (1998). Algorithm visualization for distributed environments. In *INFOVIS'98*, pages 71–78.

Pongor, G. (1993). Omnet: Objective modular network testbed. In *MASCOTS '93*, pages 323–326.

Raynal, M. (1988). *Networks and distributed computation*. MIT Press.

Stasko, J. T. and Kraemer, E. (1993). A methodology for building application-specific visualizations of parallel programs. *J. Parallel Distrib. Comput.*, 18:258–264.

Szymanski, B., Shy, Y., and Prywes, N. (1985). Synchronized distributed termination. *IEEE Transactions on software engineering*, SE-11(10):1136–1140.

Tel, G. (2000). *Introduction to distributed algorithms*. Cambridge University Press.

Yamashita, M. and Kameda, T. (1996). Computing on anonymous networks: Part i - characterizing the solvable cases. *IEEE TPDS*, 7(1):69–89.