Model-checking distributed applications with GRAS

Cristian Rosa Martin Quinson Stephan Merz

27/06/2009

Table of Contents

Introduction

Main Issues to Solve

Prototype

Related Work

Work Plan

Introduction

Developing Distributed Systems (DS) is challenging because:

- Lack of a global view of the state
- Asynchrony

SimGrid is a framework for the simulation of DS
It's development is a common effort of different teams

It provides:

- Simulation by real execution of the program (C and Java)
- Controlled execution environment
- Several APIs for developing DS (ex. MPI)

SimGrid also provides testing capabilities, but it is not enough.
Simulation is deterministic and conditions the scenarios.

Introduction

Model-checking can provide:

- a more exhaustive exploration of the state space
- counter examples on errors

Model-checking and Simulation has a lot in common, both need:

- simulation of the environment (processes, network, messages)
- control over the scheduling of the processes
- interception of the communication

We decided to add an explicit state software model-checker to SimGrid

SimGrid Layout

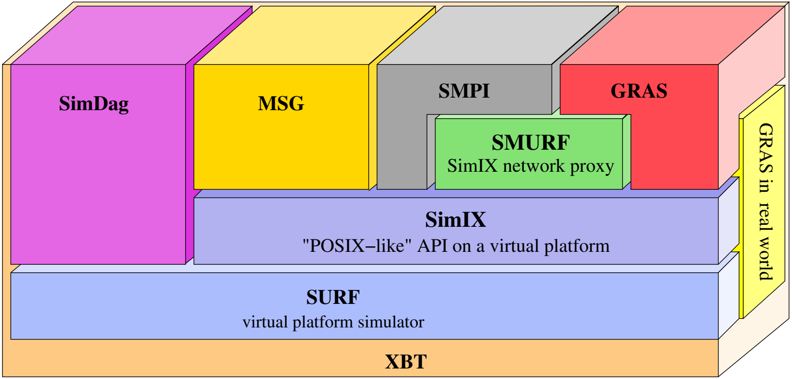


Figure: SimGrid Layout

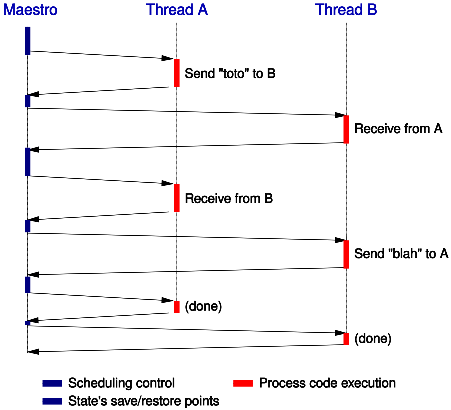
Identifying Transitions

- General C programs model-checking is difficult
- Mainly because their semantics are not given in terms of LTS
- In DS the address space of the processes are isolated
- Processes interact by message interchange
- We focus on protocol verification for DS
- We consider as transitions only the send/receive actions

Transitions and Scheduling

Internal operation of SimGrid:

- one thread per simulated process
- a *maestro* thread that decides the scheduling
- synchronous execution model
- each communicating action yields control back to maestro



Exploring the State Space

The state exploration is performed using a depth-first search

Back tracking is necessary to take different execution paths

We considered two approaches:

- re-execution from initial state
- threads' checkpoint mechanism

We decided to implement a checkpoint mechanism

Save and Restore capabilities are achieved by:

- intercepting all memory allocation functions
- using a special sanitized dynamic memory manager
- Saving/Restoring the heap, stack and data segments

Prototype

We developed a prototype that is able to check toy examples

- verifies simple unmodified C GRAS programs
- checks safety properties limited by C scoping rules
- properties are expressed as C boolean functions (assertions)
- simple state saving/restoring mechanism
- simple depth-first search exploration

Related Work

- MaceMC is a model-checker for the Mace language
  - DSL for developing Distributed Systems
  - Describes the system as a reactive state transition system
  - Source-to-Source compiler
- CMC general C/C++ program model-checker
  - Explicit-state model-checker
  - Safety properties verification
- ISP model-checker, University of Utah
  - Verification of unmodified MPI C programs
  - Safety properties verification

Work plan

Current Work

- isolate address spaces of simulated processes
- define a memory region for the network
- add support for the rest of the SimGrid APIs

Future Work

- exploit heap symmetries (heap canonicalization)
- implement partial order reductions
- add support for LTL properties verification

and at a very long term...

- experiment with out-of-core memory
- distribution of the model-checking process itself

Thank You!