Using SimGrid 101

Getting Started to Use SimGrid

Da SimGrid Team

January 8, 2018

About this Presentation

Goals and Contents

- Installing the framework
- Writing your first MSG simulator (in C or Java)
- Trace replay execution mode
- Other practical considerations

The SimGrid 101 serie

- This is part of a serie of presentations introducing various aspects of SimGrid
- SimGrid 101. Introduction to the SimGrid Scientific Project
- SimGrid User 101. Practical introduction to SimGrid and MSG
- SimGrid User::Platform 101. Defining platforms and experiments in SimGrid
- SimGrid User::SimDag 101. Practical introduction to the use of SimDag
- SimGrid User::Visualization 101. Visualization of SimGrid simulation results
- SimGrid User::SMPI 101. Simulation MPI applications in practice
- SimGrid User::Model-checking 101. Formal Verification of SimGrid programs
- SimGrid Internal::Models. The Platform Models underlying SimGrid
- SimGrid Internal::Kernel. Under the Hood of SimGrid
- Get them from `http://simgrid.gforge.inria.fr/documentation.html`

Outline

- Installing SimGrid
  - Stable release
  - Unstable Version
  - The Bindings
- Your First SimGrid Program
  - User Interface(s)
  - Master/Workers
  - Trace Replay
- Further topics
  - Configuring your simulators
  - Surviving in C
  - Bindings Performance
- Conclusion

Installing a stable version (most advised for users)

Full Instructions: `http://simgrid.org/simgrid/latest/doc/install.html`

On Debian, Ubuntu and similar

- `sudo apt-get install simgrid`
- Manual download: `http://packages.debian.org/simgrid`

For Java (regardless of your OS)

- Get the binary jarfile: `http://simgrid.gforge.inria.fr/download.php`
- Add it to your classpath. That's it: C library included for your convenience

From the sources

1. Get the archive: `http://simgrid.gforge.inria.fr/download.php`
2. Open, & build:

```
tar xfz simgrid-*.tar.gz && cd SimGrid-* && cmake .  && make
```

Installing an unstable version (developers only!)

So you want to keep on the bleeding edge, hu?

Unstable is not for anyone

- Only use it if you want to improve SimGrid
- Stable releases are frequent enough to use SimGrid
- Hint: it's called *unstable*. It may harm your kittens even if we do our best

Actually installing unstable

- `git clone git://scm.gforge.inria.fr/simgrid/simgrid.git`
- Configure and build source as usual

Additional Build Dependencies

- Please refer to the full documentation at `http://simgrid.org/simgrid/latest/doc/install.html`

The Bindings

So you don't want to code in C, hu?

Some people don't like coding in C

- C is the modern assembly language: potentially fast but tedious
- Using C is not enough for maximal performance: you need to really master it

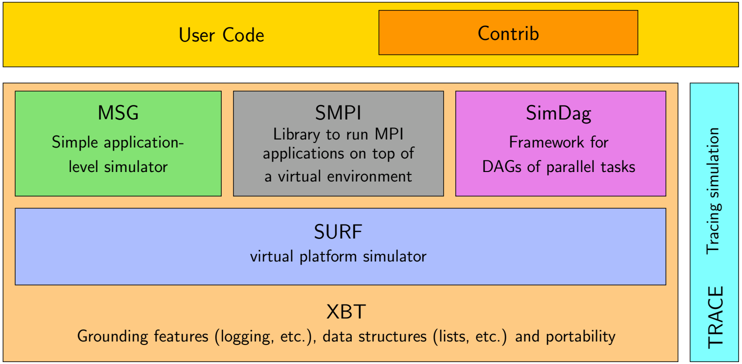
Bindings available

- Java bindings: Rock stable, very efficient, used by many people
- Patches to add news bindings are welcome (but uneasy: threading mess)

Outline

- Installing SimGrid
  - Stable release
  - Unstable Version
  - The Bindings
- Your First SimGrid Program
  - User Interface(s)
  - Master/Workers
  - Trace Replay
- Further topics
  - Configuring your simulators
  - Surviving in C
  - Bindings Performance
- Conclusion

SimGrid Overview


User Code
Contrib
MSG
Simple application-level simulator
SMPI
Library to run MPI applications on top of a virtual environment
SimDag
Framework for DAGs of parallel tasks
SURF
virtual platform simulator
XBT
Grounding features (logging, etc.), data structures (lists, etc.) and portability
TRACE
Tracing simulation

SimGrid user APIs

- If your application is a DAG of (parallel) tasks $\rightsquigarrow$ use SimDag
- To study an existing MPI code $\rightsquigarrow$ use SMPI
- In any other cases $\rightsquigarrow$ use MSG
  (easily study concurrent processes and prototype distributed applications)

The MSG User Interface

Main MSG abstractions

- **Agent:** some code, some private data, running on a given host
- **Task:** amount of work to do and of data to exchange

- Host: location on which agents execute
- Mailbox: Rendez-vous points between agents (think of MPI tags)
  - You send stuff to a mailbox; you receive stuff from a mailbox
  - Establish rendez-vous regardless of network location
  - Mailboxes identified as *strings* $\rightsquigarrow$ `host:port`, yellow pages or whatever

More information

- examples/msg in archive; Reference doc: `doc/group__MSG__API.html`
- Interface extended, never modified since 2002 (if using MSG_USE_DEPRECATED)

Executive Summary (detailed below)

1. Write the Code of your Agents

```
int master(int argc, char **argv) {

for (i = 0; i < number_of_tasks; i++) {
 t=MSG_task_create(name,comp_size,comm_size,data);
 sprintf(mailbox,"worker-%d",i % workers_count);
 MSG_task_send(t, mailbox);}
```

```
int worker(int ,char**){

sprintf(my_mailbox,"worker-%d",my_id);
while(1) {
  MSG_task_receive(&task, my_mailbox);
  MSG_task_execute(task);
  MSG_task_destroy(task);}
```

2. Describe your Experiment

XML Platform File

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM
"http://simgrid.gforge.inria.fr/simgrid.dtd">
<platform version="3">
<AS id="blah" routing="Full">
  <host id="host1" power="1E8"/>
  <host id="host2" power="1E8"/>
   ...
  <link id="link1" bandwidth="1E6"
                   latency="1E-2"  />
   ...
  <route src="host1" dst="host2">
    <link_ctn id="link1"/>
  </route>
</AS>
</platform>
```

XML Deployment File

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM
"http://simgrid.gforge.inria.fr/simgrid.dtd">
<platform version="3">
<!-- The master process -->
<process host="host1" function="master">
 <argument value="10"/><!--argv[1]:#tasks-->
 <argument value="1"/><!--argv[2]:#workers-->
</process>

<!-- The workers -->
<process host="host2" function="worker">
  <argument value="0"/></process>
</platform>
```

3. Write a main gluing things together, link and run

Master/Workers: Describing the Agents (1/2)

The master has a large number of tasks to dispatch to its workers for execution

```
int master(int argc, char *argv[ ]) {
 int number_of_tasks = atoi(argv[1]);       double task_comp_size = atof(argv[2]);
 double task_comm_size = atof(argv[3]);     int workers_count = atoi(argv[4]);
 char mailbox[80];                          char buff[64];
 int i;                                     msg_task_t task;

 /* Dispatching (dumb round-robin algorithm) */
 for (i = 0; i < number_of_tasks; i++) {
   sprintf(buff, "Task_%d", i);
   task = MSG_task_create(buff, task_comp_size, task_comm_size, NULL);
   sprintf(mailbox,"worker-%d",i % workers_count);
   XBT_INFO("Sending \"%s\" to mailbox \"%s\"", task->name, mailbox);
   MSG_task_send(task, mailbox);
 }
 /* Send finalization message to workers */
 XBT_INFO("All tasks dispatched. Let's stop workers");
 for (i = 0; i < workers_count; i++) {
   sprintf(mailbox,"worker-%ld",i % workers_count);
   MSG_task_send(MSG_task_create("finalize", 0, 0, 0), mailbox);
 }

 XBT_INFO("Goodbye now!"); return 0;
}
```

(the full code is in the archive under `examples/msg/app-masterworker`)

Master/Workers: Describing the Agents (2/2)

```
int worker(int argc, char *argv[ ]) {
  msg_task_t task;                 int errcode;
  int id = atoi(argv[1]);
  char mailbox[80];

  sprintf(mailbox,"worker-%d",id);

  while(1) {
    errcode = MSG_task_receive(&task, mailbox);
    xbt_assert(errcode == MSG_OK, "MSG_task_get failed");

    if (!strcmp(MSG_task_get_name(task),"finalize")) {
      MSG_task_destroy(task);
      break;
    }

    XBT_INFO("Processing '%s'", MSG_task_get_name(task));
    MSG_task_execute(task);
    XBT_INFO("'%s' done", MSG_task_get_name(task));
    MSG_task_destroy(task);
  }

  XBT_INFO("I'm done. See you!");
  return 0;
}
```

Master/Workers: gluing things together

```
int main(int argc, char *argv[ ]) {

  MSG_init(&argc,argv);

  /* Declare all existing agents, binding their name to their function */
  MSG_function_register("master", &master);
  MSG_function_register("worker", &worker);

  /* Load a platform instance */
  MSG_create_environment("my_platform.xml"); // we could take the names of XML files as argv
  /* Load a deployment file */
  MSG_launch_application("my_deployment.xml");

  /* Launch the simulation (until its end) */
  MSG_main();

  XBT_INFO("Simulation took %g seconds",MSG_get_clock());
}
```

Compiling and Executing the result

```
$ gcc *.c -lsimgrid -o my_simulator
$ ./my_simulator
[verbose output removed]
```

Master/Workers: deployment file

Specifying which agent must be run on which host, and with which arguments

XML deployment file

```
<?xml version="1.0"?>
<!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">
<platform version="3">

  <!-- The master process (with some arguments) -->
  <process host="host1" function="master">
     <argument value="6"/>         <!-- Number of tasks -->
     <argument value="50000000"/> <!-- Computation size of tasks -->
     <argument value="1000000"/>  <!-- Communication size of tasks -->
     <argument value="3"/>         <!-- Number of workers -->
  </process>

  <!-- The worker process (argument: mailbox number to use) -->
  <process host="host2" function="worker"><argument value="0"/></process>
  <process host="host2" function="worker"><argument value="1"/></process>
  <process host="host2" function="worker"><argument value="2"/></process>

</platform>
```

Thanks to mailboxes, the master doesn't have to know where the workers are (nor the contrary)

Master/Worker in Java (1/2)

```
import org.simgrid.msg.*;
public class BasicTask extends org.simgrid.msg.Task {
   public BasicTask(String name, double computeDuration, double messageSize) {
       super(name, computeDuration, messageSize);
}  }
public class FinalizeTask extends org.simgrid.msg.Task {
   public FinalizeTask() {
      super("finalize",0,0);
}  }
public class Worker extends org.simgrid.msg.Process {
   public Worker(Host host, String name, String[]args) { // Mandatory: this constructor is
      super(host,name,args);                              //            used internally
   }
   public void main(String[ ] args) throws TransferFailureException, HostFailureException,
                                            TimeoutException, TaskCancelledException {
      String id = args[0];

      while (true) {
         Task t = Task.receive("worker-" + id);
         if (t instanceof FinalizeTask)
            break;
         BasicTask task = (BasicTask)t;
         Msg.info("Processing '" + task.getName() + "'");
         task.execute();
         Msg.info("'" + task.getName() + "' done ");
       }
      Msg.info("Received Finalize. I'm done. See you!");
}   }
```

Master/Workers in Java (2/2)

```
import org.simgrid.msg.*;
public class Master extends org.simgrid.msg.Process {
   public Master(Host host, String name, String[]args) { // mandatory constructor
      super(host,name,args);
   }
   public void main(String[ ] args) throws MsgException {
      int numberOfTasks = Integer.valueOf(args[0]).intValue();
      double taskComputeSize = Double.valueOf(args[1]).doubleValue();
      double taskCommunicateSize = Double.valueOf(args[2]).doubleValue();
      int workerCount = Integer.valueOf(args[3]).intValue();

      Msg.info("Got "+  workerCount + " workers and " + numberOfTasks + " tasks.");

      for (int i = 0; i < numberOfTasks; i++) {
         BasicTask task = new BasicTask("Task_" + i ,taskComputeSize,taskCommunicateSize);
         task.send("worker-" + (i % workerCount));

         Msg.info("Send completed for the task " + task.getName() +
                  " on the mailbox 'worker-" + (i % workerCount) +  "'");
      }
      Msg.info("Goodbye now!");
}  }
```

The rest of the story

- No need to write the glue (thanks to Java introspection)
- Same XML files (in deployment, capitalization and package name matters)

Trace Replay: Separate your applicative workload

- If your application is event-oriented (as a P2P DHT or a scheduling heuristic), you need to get the applicative workload from somewhere

C code

```c
static void action_blah(xbt_dynar_t parameters) { ... }
static void action_blih(xbt_dynar_t parameters) { ... }
int main(int argc, char *argv[]) {
    MSG_init(&argc, argv);
    MSG_create_environment(argv[1]);
    MSG_launch_application(argv[2]);
    /* No need to register functions as usual: actions started anyway */
    xbt_replay_action_register("blah", blah);
    xbt_replay_action_register("blih", blih);
    MSG_action_trace_run(argv[3]); // The trace file to run
}
```

Deployment

```xml
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dt
<platform version="3">
  <process host="host1" function="toto"/>
  <process host="host2" function="tutu"/>
</platform>
```

Trace file

```
tutu blah toto 1e10
tutu blih 12
toto blih 12
```

- Alternatives for DAG-formated workload: DAX or dot files (see SimDag 101)

Trace Replay (2/2)

Separating the trace of each process

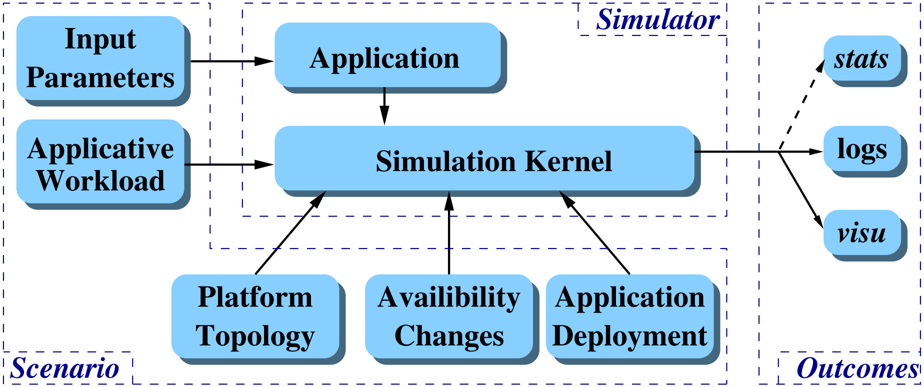
- Because it's sometimes more convenient (for MPI, you'd have to merge them)
- Simply pass NULL to MSG_action_trace_run()
- Pass the trace file to use as argument to each process in deployment

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">
<platform version="3">
  <process host="host1" function="toto">
    <argument value="actions_toto.txt"/>
  </process>
  <process host="host2" function="tutu">
    <argument value="actions_tutu.txt"/>
  </process>
</platform>
```

Action Semantic

- This mecanism is completely agnostic: attach the meaning you want to events
- In examples/actions/action.c, we have pre-written event functions for:
  - Basics: send, recv, sleep, compute
  - MPI-specific: isend, irecv, wait, barrier, reduce, bcast, allReduce

SimGrid is not a Simulator



That's a Generic Simulation Framework

Configuring your simulators

Every simulator using SimGrid accepts a set of options

- -help: get some help
- -help-models: long help on models
- -log: configure the verbosity
- -cfg: change some settings

Note: SMPI-specific settings, are only visible in SMPI simulators

The log argument

- It's similar to Log4J, but in C
- You can increase the amount of output for some specific parts of SimGrid
- Example: See everything by using –log=root.thres:debug
- List of all existing channels: doc/html/group__XBT__log__cats.html

XBT from 10,000 feets

C is a basic language: we reinvented the wheel for you

Logging support: Log4C

```
XBT_LOG_NEW_DEFAULT_CATEGORY(test,
    "my own little channel");
XBT_LOG_NEW_SUBCATEGORY(details, test,
    "Another channel");

INFO("Value: %d", variable);
CDEBUG(details,"blah %d %f %d", x,y,z);
```

Exception support

```
xbt_ex_t e;
TRY {
  block
} CATCH(e) {
  block /* DO NOT RETURN FROM THERE */
}
```

Debugging your code

- Ctrl-C once: see processes' status
- Press it twice (in 5s): kill simulator

xbt_backtrace_display_current()

```
Backtrace (displayed in thread 0x90961c0):
--->  In master() at masterslave_mailbox.c:35
--->  In ?? ([0x4a69ba5])
```

Advanced data structures

- Hash tables (Perl's ones)
- Dynamic arrays, FIFOs; Graphs

String functions

- bprintf: malloc()ing sprintf
- trim, split, subst, diff
- string buffers

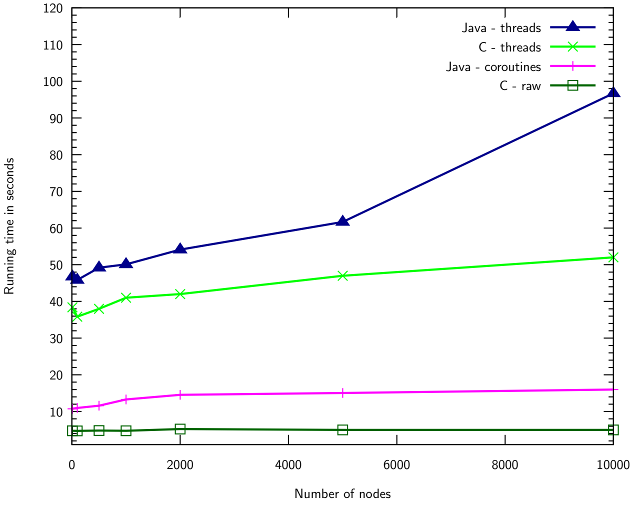
Threading support

- Portable wrappers (Lin, Win, Mac, Sim)
- Synchro (mutex, conds, semaphores)

Other

- Mallocators
- Configuration support
- Unit testing (check src/testall)
- Integration tests (tesh: testing shell)

Bindings Performance



- C: breath taking
- Java: not too bad (JVM patch $\leadsto$ good)
- Others: a bit behind

(version 3.7.1)

More information on using SimGrid

Read more

- Tutorials (http://simgrid.gforge.inria.fr/101)
  - SimGrid 101. Introduction to the SimGrid Scientific Project
  - SimGrid User 101. Practical introduction to SimGrid and MSG
  - SimGrid User::Platform 101. Defining platforms and experiments in SimGrid
  - SimGrid User::SimDag 101. Practical introduction to the use of SimDag
  - SimGrid User::Visualization 101. Visualization of SimGrid simulation results
  - SimGrid User::SMPI 101. Simulation MPI applications in practice
  - SimGrid User::Model-checking 101. Formal Verification of SimGrid programs
  - SimGrid Internal::Models. The Platform Models underlying SimGrid
  - SimGrid Internal::Kernel. Under the Hood of SimGrid
- Examples for almost all features included in archives
- The documentation itself should be ok now [SimGrid v3.9 and higher]

Get in touch

- Mailing list: `mailto:simgrid-user@lists.gforge.inria.fr`
- IRC: #simgrid on irc.debian.org
- Ask your questions on Stack Overflow, and participate to the community
- Report bugs: `https://gforge.inria.fr/tracker/?atid=165&group_id=12`

Please RTFM because we WTFM

- The documentation used to be *even worse*
- Our classical answers to users shouting “Write The Fine Manual” were:

User manuals are for wimps

- Real Men read some slides ’cause they are more concise
- They read the examples, pick one modify it to fit their needs
- They may read 2 or 5% of the reference guide to check the syntax
- In doubt, they just check the source code

lusers don’t read the manual either

- Proof: that’s why the RTFM expression were coined out
- Instead, they always ask same questions to lists, and get pointed to the FAQ

But things improved; We still try to help Real Men working the way they like :-)