Treats and Tricks, or use SimGrid

Martin Quinson

SimGrid User Days 2010, Cargese

Outline

- Wannabe User Manual
  - Configuring and Installing
  - Configuring your simulators
  - Trace Replay
  - Some "Do not ..." advices
- The bindings
  - Java
  - Lua
  - Ruby
- Surviving in C
- Conclusion

How to install the tool

In release 3.3.4 and before

- Grab it: `https://gforge.inria.fr/project/showfiles.php?group_id=12`
- ./configure && make && sudo make install

In release 3.4 and afterward

- Grab it; cmake . && make && sudo make install-simgrid

Get it from SVN

- svn checkout svn://scm.gforge.inria.fr/svn/simgrid/simgrid/trunk simgrid
- cd simgrid
- cmake . && make && sudo make install-simgrid

From autotools to cmake

- So, yeah, we moved from autoconf to cmake for release 3.4
- It offers a better user interface, with ccmake (demo)
- If allows for software quality dashboards (demo)

Configuring your simulators

Every simulator using SimGrid accepts a set of options

- -help: get some help (demo)
- -help-models: long help on models (3.4-only; demo)
- -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 (demo)
- List of all existing channels: doc/html/group__XBT__log__cats.html

Trace Replay: Separate your applicative workload

C code

```
static void action_blah(xbt_dynar_t parameters) { ... }
static void action_blih(xbt_dynar_t parameters) { ... }
static void action_bluh(xbt_dynar_t parameters) { ... }
int main(int argc, char *argv[]) {
    MSG_global_init(&argc, argv);
    MSG_create_environment(argv[1]);
    MSG_launch_application(argv[2]);
    /* No need to register functions as usual: actions started anyway */
    MSG_action_register("blah", blah);
    MSG_action_register("blih", blih);
    MSG_action_register("bluh", bluh);

    MSG_action_trace_run(argv[3]); // The trace file to run
}
```

Deployment

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "simgrid.dtd">
<platform version="2">
  <process host="Tremblay" function="toto"/>
  <process host="Jupiter" function="tutu"/>
  <process host="Fafard" function="tata"/>
</platform>
```

Trace file

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

Trace Replay (2/2)

Separating the trace of each process

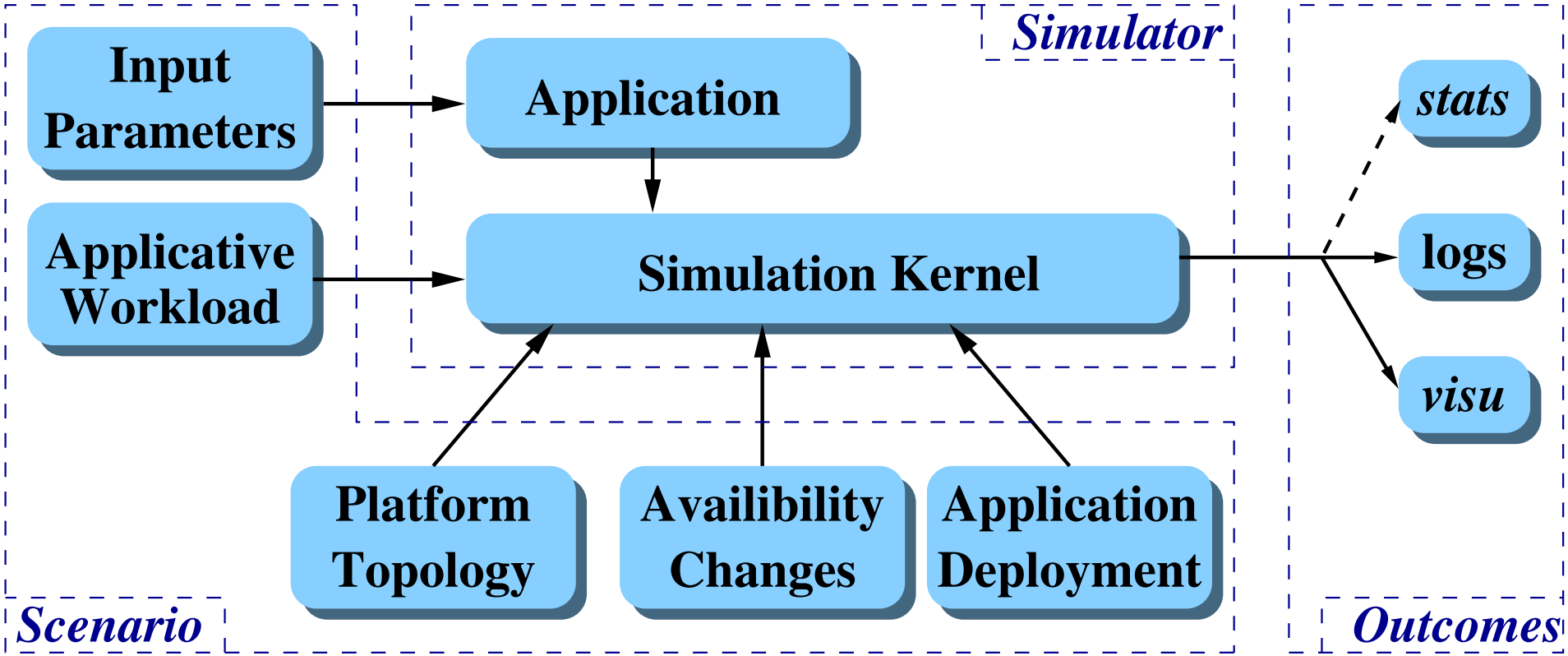
- Because it's sometimes more convinient (for MPI, you'd have to merge them)
- Simply pass NULL to MSG_action_trace_run()
- Pass the trace file to use as argument to each process in deployment

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "simgrid.dtd">
<platform version="2">
  <process host="Tremblay" function="toto">
    <argument value="actions_toto.txt"/>
  </process>
  <process host="Jupiter" 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

Outline

Do not mix results between releases!

Main issue: The events order does change

- Models don't change, only the order of events occuring at the same time
- It may/will change your application's behavior if
- In a master/slaves sending tasks in a round/robin
- In a scheduling algorithm considering the ready tasks in order
- In almost every application, actually

Why that?

- We improve the data structure used for future event set
- We sort the events to not traverse the ones which cannot be done yet
- We do lazy evaluation

Side note

- When this happens, there is a big fat warning in the ChangeLog
- (you should read it anyway, don't wait for the next SUD to get infos)

Optimistic note

- The MSG interface not changed since 2002 (backward compatibility)
- At least with MSG_USE_DEPRECATED

Do not use the <cluster> tag

But you got it already, right?

- We know the issue, will fix it at some point

Stop using MSG_task_put / MSG_task_get! Now!

Solving the rendez-vous issues in task exchanges

- MSG have a strange interaface *somehow* mimicking BSD sockets ports
  - You send to host:port, as in BSD
  - but 2 people could send to the same (which is somehow ok)
  - and 2 people could get from the same (which is annoying)
- Now, we have a much cleaner interface, based on mailboxes

Mailboxes

- Mailboxes are represented by a string (whatever you would like)
- You send stuff to a mailbox; you receive stuff from a mailbox
- Where in network the sender and receiver are do not matter for rendez-vous
- The communication timings of course take their locations into account
- This makes the user code *ways* easier

- I know that the examples were not all updated yet, sorry
- Read `examples/msg/masterslave/masterslave_mailbox.c`, ignore others

The cleaner Master/Workers in MSG with mailboxes

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;

 /* Dispatching (dumb round-robin algorithm) */
 for (i = 0; i < number_of_tasks; i++) {
   sprintf(buff, "Task_%d", i);
   task = MSG_task_create(sprintf_buffer, task_comp_size, task_comm_size, NULL);
   sprintf(mailbox,"worker-%d",i % workers_count);
   INFO2("Sending ¨%s¨ to mailbox ¨%s¨", task->name, mailbox);
   MSG_task_send(task, mailbox);
 }

 /* Send finalization message to workers */
 INFO0("All tasks dispatched. Let's stop workers");
 for (i = 0; i < workers_count; i++) {
   sprintf(mailbox,"slave-%ld",i % slaves_count);
   MSG_task_send(MSG_task_create("finalize", 0, 0, 0), mailbox);
 }

 INFO0("Goodbye now!"); return 0;
}
```

The MSG master/workers example: the worker

```
int worker(int argc, char *argv[ ]) {
  m_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_assert0(errcode == MSG_OK, "MSG_task_get failed");

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

    INFO1("Processing '%s'", MSG_task_get_name(task));
    MSG_task_execute(task);
    INFO1("'%s' done", MSG_task_get_name(task));
    MSG_task_destroy(task);
  }

  INFO0("I'm done. See you!");
  return 0;
}
```

The MSG master/workers example: 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 "surfxml.dtd">
<platform version="2">

  <!-- The master process (with some arguments) -->
  <process host="Tremblay" 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="Jupiter" function="worker"><argument value="0"/></process>
  <process host="Fafard"  function="worker"><argument value="1"/></process>
  <process host="Ginette" function="worker"><argument value="2"/></process>

</platform>
```

Thanks to mailboxes, the master don't have to know where the slaves live (nor the contrary)

The MSG master/workers example: the `main()`

Putting things together

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

  /* Declare all existing agent, 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");
  /* Load a deployment file */
  MSG_launch_application("my_deployment.xml");

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

  INFO1("Simulation took %g seconds",MSG_get_clock());
}
```

Outline

The bindings

Some people don't like coding in C

- We have some Java bindings since 2008 at least
- We have some Lua bindings
- We have some Ruby bindings

Why these languages?

- Every potential intern knows Java (I guess)
- Lucas (the office next to mine) is a very effective Ruby lobbyist
- Lua is said to allow very efficient bindings with C

"Will you add my favorite language?"

- We could, but it's rather time consuming (threading mess, at least)
- I'm not willing to start a collection here (Medhi's time is limited)

$\Rightarrow$ Patch welcome (and I wish you good luck; we see it in next SUD)

The bindings

Some people don't like coding in C

- We have some Java bindings since 2008 at least
- We have some Lua bindings since March 23. 2010
- We have some Ruby bindings since ... April 7. 2010

Why these languages?

- Every potential intern knows Java (I guess)
- Lucas (the office next to mine) is a very effective Ruby lobbyist
- Lua is said to allow very efficient bindings with C

"Will you add my favorite language?"

- We could, but it's rather time consuming (threading mess, at least)
- I'm not willing to start a collection here (Medhi's time is limited)

$\Rightarrow$ Patch welcome (and I wish you good luck; we see it in next SUD)

Master/slaves in Java (1/2)

```
import simgrid.msg.*;
public class BasicTask extends simgrid.msg.Task {
    public BasicTask(String name, double computeDuration, double messageSize) {
       super(name, computeDuration, messageSize);
   }
}
public class FinalizeTask extends simgrid.msg.Task {
   public FinalizeTask() {
      super("finalize",0,0);
  }
}
public class Worker extends simgrid.msg.Process {
   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/slaves in Java (2/2)

```
import simgrid.msg.*;
public class Master extends simgrid.msg.Process {
   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)
- The XML files are exactly the same (beware of capitalization for deployment)
- Output very similar too

Master/slaves in Lua (1/2)

```
function Master(...)
  nb_task = arg[1];
  comp_size = arg[2];
  comm_size = arg[3];
  slave_count = arg[4]

  -- Dispatch the tasks
  for i=1,nb_task do
    tk = simgrid.Task.new("Task "..i,comp_size,comm_size);
    alias = "slave "..(i%slave_count);
    simgrid.info("Master sending  '" .. simgrid.Task.name(tk) .."' To '" .. alias .."'");
    simgrid.Task.send(tk,alias);
    simgrid.info("Master done sending '".. simgrid.Task.name(tk) .."' To '" .. alias .."'");
  end
  -- Sending Finalize Message To Others
  for i=0,slave_count-1 do
    alias = "slave "..i;
    simgrid.info("Master: sending finalize to "..alias);
    finalize = simgrid.Task.new("finalize",comp_size,comm_size);
    simgrid.Task.send(finalize,alias)
  end
end
```

Some more polishing is needed

- We'd prefer `tk:send(alias)` instead of `simgrid.Task.send(tk,alias)`

Master/slaves in Lua (2/2)

The slave

```
function Slave(...)
  local my_mailbox="slave "..arg[1]

  while true do
    local tk = simgrid.Task.recv(my_mailbox);
    if (simgrid.Task.name(tk) == "finalize") then
      simgrid.info("Slave '" ..my_mailbox.."' got finalize msg");
      break
    end
    simgrid.Task.execute(tk)
  end

  simgrid.info("Slave '" ..my_mailbox.."': I'm Done . See You !!");
end
```

Setting up your experiment

```
require "simgrid"
simgrid.platform("my_platform.xml")
simgrid.application("my_deployment.xml")
simgrid.run()
simgrid.info("Simulation's over.See you.")
simgrid.clean()
```

Master/slaves in Ruby (1/2)

Some mandatory headers

```
require 'simgrid'
include MSG
```

The master

```
class Master < MSG::Process
  def main(args)
     numberOfTask = Integer(args[0])
     taskComputeSize = Float(args[1])
     taskCommunicationSize = Float(args[2])
     slaveCount = Integer(args[3])
     for i in 0..numberOfTask-1
        task = Task.new("Task_"+ i.to_s, taskComputeSize , taskCommunicationSize);
        mailbox = "slave " + (i%slaveCount).to_s
        MSG::info("Master Sending "+ task.name + " to " + mailbox)
        task.send(mailbox)
        MSG::info("Master Done Sending " + task.name + " to " + mailbox)
     end
     for i in 0..slaveCount-1
        mailbox = "slave " + i.to_s
        finalize_task = Task.new("finalize",0,0)
        finalize_task.send(mailbox)
     end
  end
end
```

Master/slaves in Ruby (2/2)

The slave

```
class Slave < MSG::Process
  def main(args)
    mailbox = "slave " + args[0]
    while true
       task = Task.receive(mailbox)
       if (task.name == "finalize")
          break
       end
       task.execute
       MSG::debug("Slave '" + mailbox + "' done executing task "+ task.name + ".")
    end
    MSG::info("I'm done, see you")
  end
end
```

Setting up your experiment

```
MSG.createEnvironment("platform.xml")
MSG.deployApplication("deploy.xml")
MSG.run
puts "Simulation time : " + MSG.getClock .to_s
MSG.exit
```

Some more polishing is needed

- Exceptions on timeout/host failure and so on?

MSG bindings: performance

What about performance loss for Java?

(Warning: these values are 2 years old)

tasks \ workers		100	500	1,000	5,000	10,000
1,000	native	.16	.19	.21	.42	0.74
	java	.41	.59	.94	7.6	27.
10,000	native	.48	.52	.54	.83	1.1
	java	1.6	1.9	2.38	13.	40.
100,000	native	3.7	3.8	4.0	4.4	4.5
	java	14.	13.	15.	29.	77.
1,000,000	native	36.	37.	38.	41.	40.
	java	121.	130.	134.	163.	200.

- Small platforms: ok
- Larger ones: not quite...

What about the others?

- It's too recent, I didn't had time to rerun the full benchs
- Very preliminary results for Master/slaves (10 slaves; 200,000 tasks):

C (native)	7s
Lua	10.5s
Ruby	45s
Java	47s

- That's improvable
- It's garbage-collected
- User stack is dynamic in lua&ruby(?)
  $\Rightarrow$ better scalability?

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");

INFO1("Value: %d", variable);
CDEBUG3(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
- SWAG (don't use); 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)

Conclusion: Finding the documentation

Conclusion: Finding the documentation

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

Conclusion: Finding the documentation

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

Conclusion: Finding the documentation

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

So, where is all SimGrid documentation?

- The SimGrid tutorial is a 200 slides presentation (motivation, models, example of use, internals)
- Almost all features of UAPI are demoed in an example (coverage testing)
- The reference guide contains a lot in introduction sections (about XBT)
- The FAQ contains a lot too (installing, visu, XML, exotic features)
- The code is LGPL anyway