Tracing and Visualization 101

Getting Started with Tracing/Visualization in SimGrid

Da SimGrid Team

May 13, 2016

About this Presentation

Presentation Goals and Contents

- Tracing SimGrid simulations: registering behavior
- Visualization of Results: understanding behavior

The SimGrid 101 Series

- 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::SMPI 101. Simulation MPI applications in practice
- SimGrid User::Visualization 101. Visualization of SimGrid simulation results
- 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

Introduction

Alright! SimGrid-based simulator is coded, now what?

- Result analysis!
- Does the simulator behaves as expected?
- Extract metrics from the simulation?
- Is there something unexpected, or anomalies, going on?
- Need illustrations of specific scenarios for your papers?

Introduction

Alright! SimGrid-based simulator is coded, now what?

- Result analysis!
- Does the simulator behaves as expected?
- Extract metrics from the simulation?
- Is there something unexpected, or anomalies, going on?
- Need illustrations of specific scenarios for your papers?

Implementing by yourself might be a solution, but ...

- Time-consuming, probably will only work for your simulator
- Hard to get simulated data from SURF (the kernel with CPU/network models)

Introduction

Alright! SimGrid-based simulator is coded, now what?

- Result analysis!
- Does the simulator behaves as expected?
- Extract metrics from the simulation?
- Is there something unexpected, or anomalies, going on?
- Need illustrations of specific scenarios for your papers?

Implementing by yourself might be a solution, but ...

- Time-consuming, probably will only work for your simulator
- Hard to get simulated data from SURF (the kernel with CPU/network models)

The **TRACE** Module: SimGrid built-in tracing mechanism

- Can be used to trace any SimGrid simulation
- Extensible, you can trace your own simulator-specific data
- You get Pajé trace files as output: generic format, easy to visualize

Outline

- Built-in Tracing Facilities
  - Tracing the MSG interface
  - Tracing the Simulated MPI (SMPI)
  - Uncategorized Resource utilization
  - Categorized Resource Utilization
  - Tracing User Variables & States
- Visualizing the Traces
  - Space-Time view
  - Treemap view
  - Graph view
  - Statistical Analysis and Beyond
- Tracing methods ⇒ visualization techniques
- Further Topics
- Conclusion

Tracing the MSG interface

Registering MSG processes behavior (For each process, timestamped data)

- Processes are grouped by <host>, following the platform file AS hierarchy
- `Sleep` ⇒ MSG_process_sleep
- `Suspend` ⇒ MSG_process_suspend, MSG_process_resume
- `Receive` ⇒ MSG_task_receive
- `Send` ⇒ MSG_task_send
- `Task_execute` ⇒ MSG_task_execute
- Match MSG_task_send with the corresponding MSG_task_receive
- Process migrations with MSG_process_migrate

What you can do with

- Space/Time, Treemap views; Correlate processes behavior (see Visualization)
- Derive statistics from traces; Analyze process migrations

Activate this type of tracing using these parameters

`--cfg=tracing:1` and `--cfg=tracing/msg/process:1`

Tracing the Simulated MPI (SMPI) interface

Registering MPI ranks behavior (For each rank, timestamped data)

- Like tracing tools you already know (scorep, TAU, ...)
- Start/End of each MPI operation, examples `MPI_Send` , `MPI_Reduce` , ...
- Point-to-point and collective communications
- Rank organization
  - Ungrouped, non-hierarchical: as usually done for most tracing mechanisms
  - Grouped, hierarchical: according to the AS hierarchy of the platform file
- MPE Interface (you can use your preferred tracing library)
  Attention: you need to timestamp events with the simulated clock

What you can do with

- Space/Time, Treemap views; Correlate processes behavior (see Visualization)
- Derive statistics from traces

Activate this type of tracing using these parameters

smpirun `-trace` ...

smpirun `-trace-grouped` ...

⇒ See smpirun `--help` for details

(Uncategorized) Resource Utilization Tracing

Trace <host> and <link> resource capacity and utilization

- Bounds: power for hosts, bandwidth (and latency) for links
- Capacity variations along time (if availability traces are used)
- Utilization: power_uncategorized (hosts) and bandwidth_uncategorized (links)

Advantages

- No modifications required (can be used to trace all SimGrid simulators)
- Changes on capacity/utilization are extracted from the SURF kernel

What you can do with

- Network topology correlation
- Treemap, Graph views, but also derive statistics from traces

Activate this type of tracing using these parameters

`--cfg=tracing:1` `--cfg=tracing/uncategorized:1` for MSG and SimDag

$ **smpirun** `-trace` `-trace-resource` for SMPI

Categorized Resource Utilization

Motivation

- Alright, with *uncategorized* tracing, we known how much of resource is used
- But it is hard to associate that utilization to the application code

Categorized Resource Utilization

Motivation

- Alright, with *uncategorized* tracing, we known how much of resource is used
- But it is hard to associate that utilization to the application code

Solution: Categorize the resource utilization

- Declare tracing categories, with `TRACE_category` or `TRACE_category_with_color`
- Classify (MSG, SimDAG) tasks by giving them one (and only one) category with `MSG_task_set_category` or `SD_task_set_category`
- Trace will contain for all <host> and <link> resource
  - Bounds: power for hosts, bandwidth (and latency) for links
  - Utilization: **p**category (for hosts) and **b**category (for links)
- Advantages
  - Detect the tasks that are the CPU/network bottleneck
  - Verify application phases (and their eventual overlappings)
  - Check competing applications or users
  - Correlate all that with the network topology
  - ______________________________ ⇐ your study case here
- To use: `--cfg=tracing:1` `--cfg=tracing/categorized:1` (MSG/SimDag)

Registering User Variables

How to trace application-specific data

- Simulator keeps track of its own variables
- User Variables can be associated to <host>s and <link>s
- All events are timestamped with current simulated time

- Associating variables to <host>s
  - Declare once: `TRACE_host_variable_declare (variable)`
    Note: Each variable should be declared only once
  - Set/Add/Sub as needed: `TRACE_host_variable_[set|add|sub]`
    Note: first parameter is the hostname (as present in the platform file)
- Associating to <link>s
  - Declare once: TRACE_**link**_variable_declare (variable)
  - Set/Add/Sub: TRACE_**link**_variable_[set|add|sub]
    Note: Link name has to be provided. Alternative way below.
  - If you need: TRACE_**link**_srcdst_variable_[set|add|sub]
    Note: You provide source and destination hosts, **Trace** uses get_route, and update the variable for all the links connecting the two hosts.

Activate this type of tracing using these parameters

`--cfg=tracing:1` `--cfg=tracing/platform:1` for MSG and SimDag

Registering User States

States? What for?

- Periods of time where the application is within a particular state. Examples:
  - Simulated process is checkpointing (`Checkpointing` state)
  - Server is dealing with client requests (`Processing` state)
- User states are always associated to <host>s
- Space/Time views show states for all processes along a time axis

API – How to use it

Node: all events are timestamped with current simulated time

- Declare: `TRACE_host_state_declare (state_name)`
- Declare values: `TRACE_host_state_declare_value (state_name, value, color)`
- Then, set the beginning of a state: `TRACE_host_set_state (...)`
- Or push/pop like a stack: `TRACE_host_[push/pop]_state (...)`

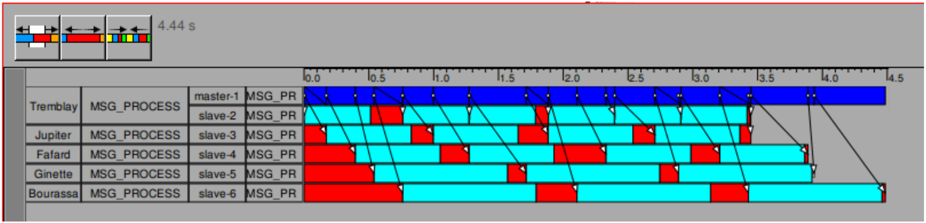
  Note: Make sure to pop all your pushes, or reset as below.
- You can also kill the stack/finish current state: `TRACE_host_reset_state (...)`
- To use: `--cfg=tracing:1 --cfg=tracing/platform:1` (MSG/SimDag)

Space-Time view #1

Gantt-like graphical view (you are looking for causalities)

- Horizontal axis represents time
- Vertical axis has the list of monitored entities (Processes, Hosts, ...)
  Note: The AS hierarchy of the platform file is represented on the left.
- Arrows represent communication (origin and destination)
- Colors represent the states:
  Blue: MSG_task_send, Red: MSG_task_receive, Cyan: MSG_task_execute

View of a trace obtained with `--cfg=tracing:1` `--cfg=tracing/msg/process:1`



Paje Trace Visualization Tools

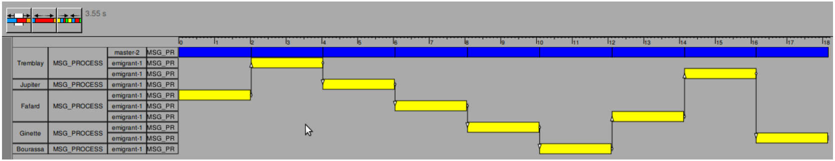
Vite	**FrameSOC**
http://vite.gforge.inria.fr	http://github.com/soctrace-inria/

Space-Time view #2

Process migrations

- Arrows might also represent process migrations
- Color keys
  - Blue: MSG_task_send
  - Yellow: MSG_process_sleep
- Several filtering/interaction capabilities, examples
  - Remove some states, links
  - Change the order of monitored entities
  - Adjust the vertical size occupied by each process

View of the trace obtained with `--cfg=tracing:1` `--cfg=tracing/msg/process:1`



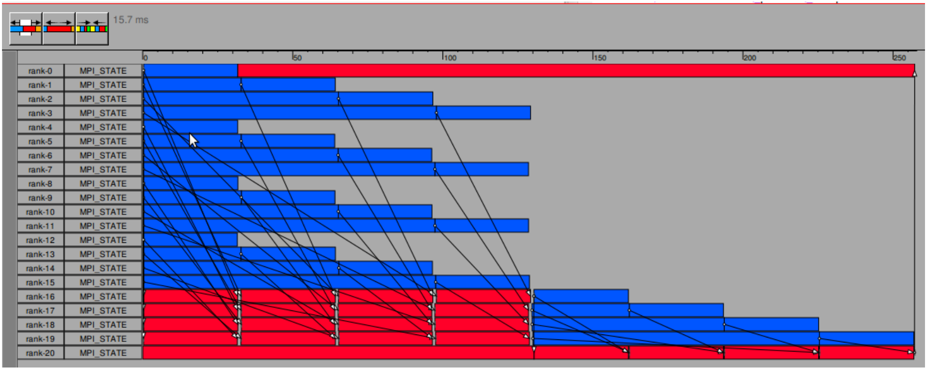
Space-Time view #3

Simulated MPI visualization

- Each MPI rank is listed vertically
- One color for each MPI operation, arrows are point-to-point communications
  - Blue: MPI_Send
  - Red: MPI_Recv

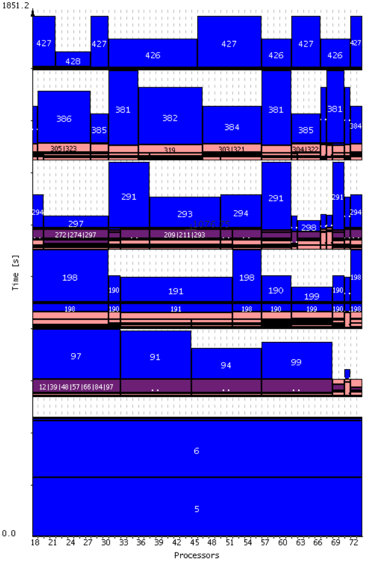
Example video for SC'10

View of the trace obtained with **smpirun** `-trace`



Space-Time view #4

Execution of Parallel Tasks



- Horizontal axis represents resources (hosts)
- Vertical axis represents time
- A task occupies multiple computing resources during some duration
- Contiguous usage of resources by a single task is represented by a single rectangle
- E.g., look for task 190 or 381

Jedule

http://jedule.sourceforge.net

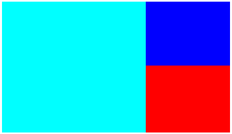
Treemap view #1

Scalable and hierarchical representation

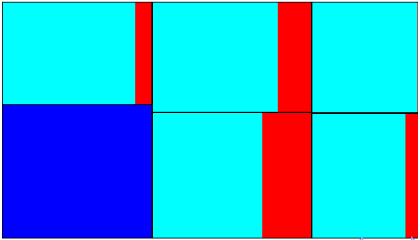
- Good for comparing monitored entities behavior
  - What are the processes that spent more time on MSG_task_send?
  - Which hosts are more used?
  - All MPI ranks behave equally?
  - Which cluster has more aggregated computing power?
- Temporal/Spatial data aggregation (user select a time slice)
- Can be used to compare all kinds of traces generated by SimGrid

How does it work?

- Trace data ⇒ screen space
- Colors are states (same color key)
- Spatial data aggregation



Trace obtained with `--cfg=tracing:1 --cfg=tracing/msg/process:1`

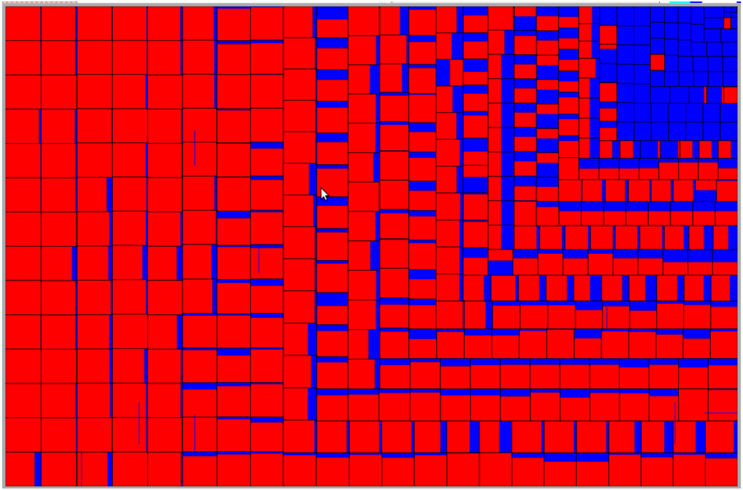


Treemap view #2

What about Simulated MPI (SMPI)?

- 448 Processes, MPI_Recv (red), MPI_Send (blue)

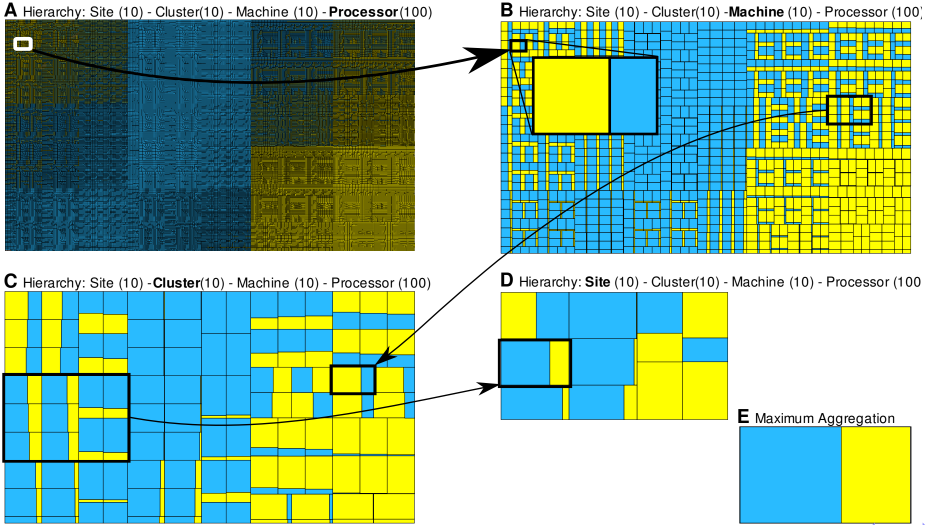
Example video for SC'10



Treemap view #3

- Synthetic trace, 100 thousands processes, 2 states
- Hierarchical representation (follows the hierarchy of the SimGrid platform file)

  Note: Better platform hierarchy, better the treemap analysis



A Hierarchy: Site (10) - Cluster(10) - Machine (10) - **Processor**(100)
B Hierarchy: Site (10) - Cluster(10) -**Machine** (10) - Processor (100)
C Hierarchy: Site (10) -**Cluster**(10) - Machine (10) - Processor (100)
D Hierarchy: **Site** (10) - Cluster(10) - Machine (10) - Processor (100
E Maximum Aggregation

Graph view (for a Topological Analysis) #1

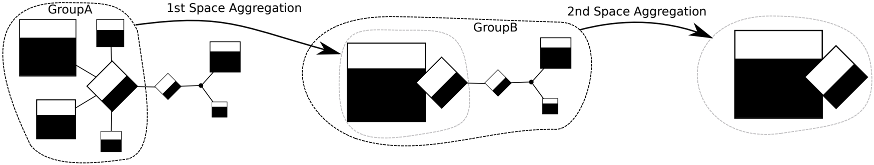
Scalable representation

- Good for correlating application behavior to network topology
  - Where is the bottleneck of my simulation?
  - What is limiting my application: the CPU power, or the network links?
  - Is the bottleneck permanent or temporary?
  - Which part of my application causes the bottleneck?

Graph view (for a Topological Analysis) #1

Scalable representation

- Good for correlating application behavior to network topology
  - Where is the bottleneck of my simulation?
  - What is limiting my application: the CPU power, or the network links?
  - Is the bottleneck permanent or temporary?
  - Which part of my application causes the bottleneck?
- Start with a hypergraph
  - Platform ASes, hosts, network links and routers are the **nodes**
  - Routes are represented by the **edges**
- Spatial data aggregation, but also temporal data aggregation



Graph view (for a Topological Analysis) #2

How does it work with SimGrid?

- Uncategorized or categorized tracing

- Configuration files generated by SimGrid
- (Uncategorized) resource utilization

  `--cfg=triva/uncategorized:uncat.plist`
- Categorized resource utilization

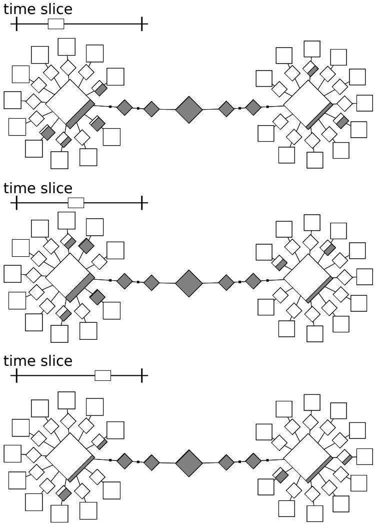
  `--cfg=triva/categorized:cat.plist`

Possible Graph Configurations

- Node size mapped to
  - CPU power, link bandwidth
  - User variables

Viva Visualization Tool

- http://triva.gforge.inria.fr
- **http://github.com/schnorr/viva**



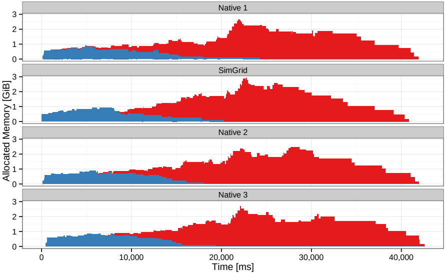
Example video for SC'10 **Categorized View**

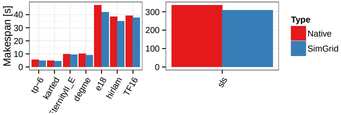
Topology Aggregation

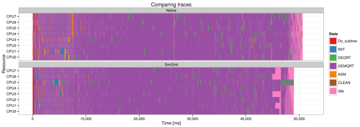
Statistical Analysis and Beyond

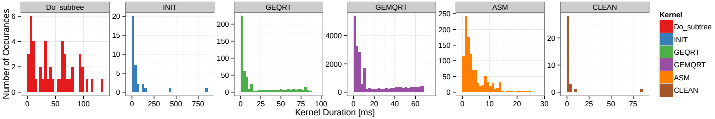
Need Some Real Numbers and Advanced Statistics

- Use pj_dump from Pajeng (http://github.com/schnorr/pajeng) to convert SimGrid/Pajé traces into CSV (comma separated values)
- Then load the CSV files in R and have fun!
- Use org-mode or knitR to automatically regenerate your articles/figures from your SimGrid traces!!!









Summary

Mapping tracing methods to visualization techniques

- Tracing the MSG, SMPI, User States
  ⇒ Space/Time view – Vite or Framesoc
  ⇒ Treemap view – Viva
- Tracing SIMDAG
  ⇒ Space/Time view – Vite, Framesoc, or Jedule
- Tracing Uncategorized/Categorized resource utilization, User variables
  ⇒ Treemap or Graph view – Viva

Visualization Tools

- Paje – http://paje.sourceforge.net
- Jedule – http://jedule.sourceforge.net
- Vite – http://vite.gforge.inria.fr
- Viva – http://github.com/schnorr/viva
- Framesoc – http://github.com/soctrace-inria
- **Pj_dump** – http://github.com/schnorr/pajeng
- Deprecated softwares to not use: Paje and Triva

Random Additional Topics

Tracing SMPI with an external library: **Akypuera**

- Low-memory footprint, binary format (http://github.com/schnorr/akypuera)
- Configure **aky** to use the simulated timestamps
  Note: Compile Aky with THREADED flag, launch SMPI with the thread context factory

Understanding the **Pajé** Trace Format

- Self-defined, textual and generic trace file format
- More information: http://paje.sourceforge.net/download/publication/lang-paje.pdf

Deadlock during simulation?

- You get a `Go fix your code!!` message from the SimGrid framework
- Run with `--cfg=tracing:1` `--cfg=tracing/msg/process:1`
  Space/Time view to see the last state of all blocked processes (MSG-only)

Turn your platform file into a graph with **graphicator**

- Transforms any XML platform file into a flat dot file (in the GraphViz format)

Conclusion

More information, check the documentation

- http://simgrid.gforge.inria.fr
- Tracing simulations section
- **Trace** API Module
- simgrid_dir/examples/msg/tracing/

We are also at the simgrid-user mailing list

Bug reports are welcome