Simulating DAG Scheduling Algorithms with SimDAG

Frédéric Suter (CNRS, IN2P3 Computing Center, France)
Martin Quinson (Nancy University, France)
Arnaud Legrand (CNRS, Grenoble University, France)
Henri Casanova (Hawai'i University at Manoa, USA)

simgrid-dev@gforge.inria.fr

About this Presentation

Presentation Goals and Contents

- Scientific Context of SimDAG: Studying DAG scheduling algorithms
- History of SimDag: Revival of the original API of SimGrid
- Learning by the example: Step by step introduction to the main features

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
- Retrieve them from `http://simgrid.gforge.inria.fr/101`

Agenda

- Introduction
  - History
  - Context
  - To know before starting
- DAGs in SimDag
- Experimental Environment
- Scheduling Tasks on Resources
- Running the Simulation
- A Complete Scheduling Simulator Example
- Conclusion

SimDAG's History

SimGrid v1

- Started in 1999 by H. Casanova and A. Legrand
- study of"centralized" scheduling
  - off-line DAG scheduling on heterogeneous compute nodes
  - API very close to that of SimDAG

SimGrid v2

- Arnaud studies "decentralized" scheduling heuristics
  - MSG becomes the new default API in 2003
- The original API remained alive up to v2.18.5
  - And totally disappeared with the raise of SURF in v2.90 (02/2005)

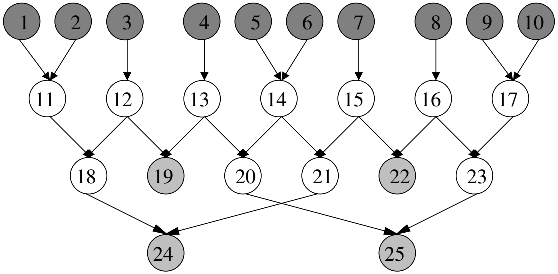
SimGrid v3

- Some people liked the lost SG API
- C. Thierry ports back the original API on top of SURF
  - SimDAG is born (again) in v3.1 (07/2006)

Definition of a DAG

Directed Acyclic Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

- $\mathcal{V} = \{v_i \mid i = 1, \ldots, V\}$
  - A set of vertices representing tasks
- $\mathcal{E} = \{e_{i,j} \mid (i,j) \in \{1, \ldots, V\} \times \{1, \ldots, V\}\}$
  - A set of edges representing precedence constraints between tasks



Definition of DAG Scheduling

Basic principle

- For each task
  - Assign a (set of) resource(s) for execution
  - Define an execution order
- Respect the precedence constraints
  - A task cannot start before all its predecessors have completed

Types of scheduling

- Offline
  - Take all decisions beforehand and then simulate
- Online
  - Take the decisions as the simulation goes

Fundamental Data Types

Link — SD_link_t

- A network resource with a bandwidth and a latency
- A route between two hosts is a list of links

Workstation — SD_workstation_t

- A place to execute some computation
- Has some computing capabilities
- Encompasses an host and one or more links

Task — SD_task_t

- A sequential or parallel computation
- Represented by some computing amount to execute

Dependency — SD_dependency_t

- Expresses an execution order between two tasks

SimDag Code Sample

Before starting to code anything

- `#include "simdag/simdag.h"` is mandatory

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

  /* Insert your code here */

  return 0;
}
```

SimDag Code Sample

Before starting to code anything

- `#include "simdag/simdag.h"` is mandatory
- Start by initializing the SimDag stuff

```
int main(int argc, char **argv) {
  SD_init(&argc, argv);

  /* Insert your code here */



  return 0;
}
```

SimDag Code Sample

Before starting to code anything

- `#include "simdag/simdag.h"` is mandatory
- Start by initializing the SimDag stuff
- End by cleaning this stuff neatly

```
int main(int argc, char **argv) {
  SD_init(&argc, argv);

  /* Insert your code here */

  SD_exit();

  return 0;
}
```

Agenda

- Introduction
- DAGs in SimDag
  - Vertices/Tasks
  - Edges/Dependencies
  - Describing a Complete DAG
- Experimental Environment
- Scheduling Tasks on Resources
- Running the Simulation
- A Complete Scheduling Simulator Example
- Conclusion

How to Represent Vertices/Tasks

Sequential computation

- Use a task of type `SD_TASK_COMP_SEQ`
- Constructor: `SD_task_create_comp_seq(name, data, amount)`
  - `name`: the name of the task, as given by the user
  - `data`: some user data attached to the task
    - Useful for scheduling attributes
  - `amount`: the number of flops computed by this task
- Destructor: `SD_task_destroy(task)`
- Can be used with any model compound handled by SURF
  - see `--help-models` for details

Parallel computation

- No type (`SD_TASK_NOT_TYPED`), default kind of task
- Constructor: `SD_task_create(name, data, amount)`
  - `amount`: represents the sequential cost of the task
- Restricted to the `ptask_L07` model

SimDag Code Sample

Sequential computation task creation and destruction

```
int main(int argc, char **argv) {
  SD_task_t task;

  SD_init(&argc, argv);





  SD_exit();
  return 0;
}
```

SimDag Code Sample

Sequential computation task creation and destruction

- Create a task that computes 1 billion of floating operations

```
int main(int argc, char **argv) {
  SD_task_t task;

  SD_init(&argc, argv);

  task = SD_task_create_comp_seq("my_sequential_task", NULL, 1E9);



  SD_exit();
  return 0;
}
```

SimDag Code Sample

Sequential computation task creation and destruction

- Create a task that computes 1 billion of floating operations
- And destroy it

```
int main(int argc, char **argv) {
  SD_task_t task;

  SD_init(&argc, argv);

  task = SD_task_create_comp_seq("my_sequential_task", NULL, 1E9);

  SD_task_destroy(task);

  SD_exit();
  return 0;
}
```

SimDag Code Sample

Parallel computation task creation and destruction

```
int main(int argc, char **argv) {
  SD_task_t task;

  SD_init(&argc, argv);





  SD_exit();
  return 0;
}
```

SimDag Code Sample

Parallel computation task creation and destruction

- Create a task that computes 1 billion of floating operations

```
int main(int argc, char **argv) {
  SD_task_t task;

  SD_init(&argc, argv);

  task = SD_task_create("my_parallel_task", NULL, 1E9);



  SD_exit();
  return 0;
}
```

SimDag Code Sample

Parallel computation task creation and destruction

- Create a task that computes 1 billion of floating operations
- And destroy it

```
int main(int argc, char **argv) {
  SD_task_t task;

  SD_init(&argc, argv);

  task = SD_task_create("my_parallel_task", NULL, 1E9);

  SD_task_destroy(task);

  SD_exit();
  return 0;
}
```

How to Represent Edges

Control flow dependency

- a.k.a precedence constraint
- Goal: Force SimDAG to wait for the completion of ● to start ●
- Create a `SD_task_dependency`

How to Represent Edges

Control flow dependency

- a.k.a precedence constraint
- Goal: Force SimDAG to wait for the completion of ● to start ●
- Create a `SD_task_dependency`
  - `SD_task_dependency_add (name, data,●, ●)`

SimDag Code Sample

Addition of a control flow dependency

```
int main(int argc, char **argv) {
  SD_task_t taskA, taskB;
  SD_init(&argc, argv);









  SD_exit();
  return 0;
}
```

SimDag Code Sample

Addition of a control flow dependency

- Create two typed sequential tasks
  - Similar for parallel tasks

```
int main(int argc, char **argv) {
  SD_task_t taskA, taskB;
  SD_init(&argc, argv);

  taskA = SD_task_create_comp_seq("taskA", NULL, 1E9);
  taskB = SD_task_create_comp_seq("taskB", NULL, 1E9);

  SD_exit();
  return 0;
}
```

SimDag Code Sample

Addition of a control flow dependency

- Create two typed sequential tasks
  - Similar for parallel tasks
- Add the dependency

```
int main(int argc, char **argv) {
  SD_task_t taskA, taskB;
  SD_init(&argc, argv);

  taskA = SD_task_create_comp_seq("taskA", NULL, 1E9);
  taskB = SD_task_create_comp_seq("taskB", NULL, 1E9);

  SD_task_dependency_add ("my_dependency", NULL, taskA, taskB);




  SD_exit();
  return 0;
}
```

SimDag Code Sample

Addition of a control flow dependency

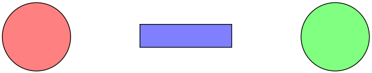
- Create two typed sequential tasks
  - Similar for parallel tasks
- Add the dependency
- Destroy the tasks
  - The dependency is automatically cleaned

```
int main(int argc, char **argv) {
  SD_task_t taskA, taskB;
  SD_init(&argc, argv);

  taskA = SD_task_create_comp_seq("taskA", NULL, 1E9);
  taskB = SD_task_create_comp_seq("taskB", NULL, 1E9);

  SD_task_dependency_add ("my_dependency", NULL, taskA, taskB);

  SD_task_destroy(taskA);
  SD_task_destroy(taskB);

  SD_exit();
  return 0;
}
```

How to Represent Edges

Data flow dependency

- a.k.a passing data from a task to another
- Need to create a transfer ■ task between ● and ●



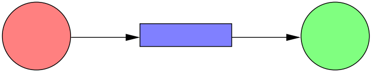
Question

- How to declare a transfer task?

How to Represent Edges

Data flow dependency

- a.k.a passing data from a task to another
- Need to create a transfer task between and
  - Add `SD_task_dependency` accordingly
  - `SD_task_dependency_add (name, data, , )`
  - `SD_task_dependency_add (name, data, , )`



Question

- How to declare a transfer task?

How to Represent Transfer Tasks

End-to-end communications

- If both source and destination are sequential tasks
- Use a task of type `SD_TASK_COMM_E2E`
- Constructor: `SD_task_create_comm_e2e(name, data, amount)`
  - `name`: the name of the task, as given by the user
  - `data`: some user data attached to the task
  - `amount`: the number of bytes transfered by this task

MxN data redistributions

- Same as for parallel computations
- No type (`SD_TASK_NOT_TYPED`)
- Constructor: `SD_task_create(name, data, amount)`
  - `amount`: represents the total number of bytes
  - Communication scheme defined at scheduling time
- Restricted to the `ptask_L07` model

SimDag Code Sample

End-to-End transfer task creation and destruction

```
int main(int argc, char **argv) {
  SD_task_t task;

  SD_init(&argc, argv);





  SD_exit();
  return 0;
}
```

SimDag Code Sample

End-to-End transfer task creation and destruction

- Create a task that moves 1 billion of bytes

```
int main(int argc, char **argv) {
  SD_task_t task;

  SD_init(&argc, argv);

  task = SD_task_create_comm_e2e("my_end-to-end_transfer", NULL, 1E9);



  SD_exit();
  return 0;
}
```

SimDag Code Sample

End-to-End transfer task creation and destruction

- Create a task that moves 1 billion of bytes
- And destroy it

```
int main(int argc, char **argv) {
  SD_task_t task;

  SD_init(&argc, argv);

  task = SD_task_create_comm_e2e("my_end-to-end_transfer", NULL, 1E9);

  SD_task_destroy(task);

  SD_exit();
  return 0;
}
```

SimDag Code Sample

$M \times N$ transfer task creation and destruction

```
int main(int argc, char **argv) {
  SD_task_t task;

  SD_init(&argc, argv);





  SD_exit();
  return 0;
}
```

SimDag Code Sample

$M \times N$ transfer task creation and destruction

- Create a task that moves 1 billion of bytes

```
int main(int argc, char **argv) {
  SD_task_t task;

  SD_init(&argc, argv);

  task = SD_task_create("my_m-x-n_transfer", NULL, 1E9);



  SD_exit();
  return 0;
}
```

SimDag Code Sample

$M \times N$ transfer task creation and destruction

- Create a task that moves 1 billion of bytes
- And destroy it

```
int main(int argc, char **argv) {
  SD_task_t task;

  SD_init(&argc, argv);

  task = SD_task_create("my_m-x-n_transfer", NULL, 1E9);

  SD_task_destroy(task);

  SD_exit();
  return 0;
}
```

SimDag Code Sample

Addition of a data flow dependency

```
int main(int argc, char **argv) {
  SD_task_t src, dest, comm;
  SD_init(&argc, argv);








  SD_exit();
  return 0;
}
```

SimDag Code Sample

Addition of a data flow dependency

- Create two computation tasks and one transfer task

```
int main(int argc, char **argv) {
  SD_task_t src, dest, comm;
  SD_init(&argc, argv);

  src = SD_task_create_comp_seq("src", NULL, 1E9);
  dest = SD_task_create_comp_seq("dest", NULL, 1E9);
  comm = SD_task_create_comm_e2e("comm", NULL, 1E9);









  SD_exit();
  return 0;
}
```

SimDag Code Sample

Addition of a data flow dependency

- Create two computation tasks and one transfer task
- Add the dependencies

```
int main(int argc, char **argv) {
  SD_task_t src, dest, comm;
  SD_init(&argc, argv);

  src = SD_task_create_comp_seq("src", NULL, 1E9);
  dest = SD_task_create_comp_seq("dest", NULL, 1E9);
  comm = SD_task_create_comm_e2e("comm", NULL, 1E9);

  SD_task_dependency_add ("src_comm", NULL, src, comm);
  SD_task_dependency_add ("comm_dest", NULL, comm, dest);





  SD_exit();
  return 0;
}
```

SimDag Code Sample

Addition of a data flow dependency

- Create two computation tasks and one transfer task
- Add the dependencies
- Destroy the tasks

```
int main(int argc, char **argv) {
  SD_task_t src, dest, comm;
  SD_init(&argc, argv);

  src = SD_task_create_comp_seq("src", NULL, 1E9);
  dest = SD_task_create_comp_seq("dest", NULL, 1E9);
  comm = SD_task_create_comm_e2e("comm", NULL, 1E9);

  SD_task_dependency_add ("src_comm", NULL, src, comm);
  SD_task_dependency_add ("comm_dest", NULL, comm, dest);

  SD_task_destroy(src);
  SD_task_destroy(dest);
  SD_task_destroy(comm);

  SD_exit();
  return 0;
}
```

Describing a Complete DAG

By hand

1. Create all computation tasks
2. Create all transfer tasks
3. Create all the control flow dependencies
4. Create all the data flow dependencies

Use a loader

- `SD_daxload(filename)`: loader for DAX files
  - Format of workflow used by Pegasus (`http://pegasus.isi.edu/`)
- `SD_dotload(filename)`: loader for DOT files
  - Well-known format of the `graphviz` tool suite
- Creates everything automatically
- Adds two special dummy tasks: `root` and `end`
- Returns a `xbt_dynar_t` of typed `SD_task_t`
  - `SD_TASK_COMP_SEQ` and `SD_TASK_COMM_E2E`

DAX File Sample (1/3)

Header

- Name space and schema declaration
  - From Pegasus
- Name of the DAX
- Number of jobs: `jobCount`
- Number of control dependencies: `childCount`

```
<?xml version="1.0" encoding="UTF-8"?>
<adag xmlns="http://pegasus.isi.edu/schema/DAX"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://pegasus.isi.edu/schema/DAX
                          http://pegasus.isi.edu/schema/dax-2.1.xsd"
      version="2.1" count="1" index="0" name="smalldax"
      jobCount="3" fileCount="0" childCount="1">

<!-- Job and control dependencies go here -->

</adag>
```

DAX File Sample (2/3)

Job description

- Described by: id, name, runtime, input and output files
  - Only computations are described ($\texttt{amount} = \texttt{runtime} \times 4.2e9$)
  - Output of task1 is a input of task2 $\Rightarrow$ Transfer task + data flow dependency

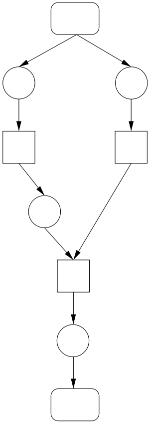
```
<adag ...">
  <job id="1" namespace="SG" name="task1" version="1.0" runtime="10">
    <uses file="i1" link="input" register="true" transfer="true"
          optional="false" type="data" size="1000000"/>
    <uses file="o1" link="output" register="true" transfer="true"
          optional="false" type="data" size="1000000"/>
  </job>
  <job id="2" namespace="SG" name="task2" version="1.0" runtime="10">
    <uses file="i2" link="input" register="true" transfer="true"
          optional="false" type="data" size="1000000"/>
  </job>
  <job id="3" namespace="SG" name="task3" version="1.0" runtime="10">
    <uses file="o1" link="input" register="true" transfer="true"
          optional="false" type="data" size="1000000"/>
    <uses file="o3" link="output" register="true" transfer="true"
          optional="false" type="data" size="1000000"/>
  </job>
<!-- Control-flow dependencies -->
</adag>
```

DAX File Sample (3/3)

Control flow dependencies

- task3 cannot start before the completion of task2
  - While there is no data flow dependency

```
<adag ...>

<!-- Job descriptions -->

  <child ref="3">
    <parent ref="2"/>
  </child>
</adag>
```

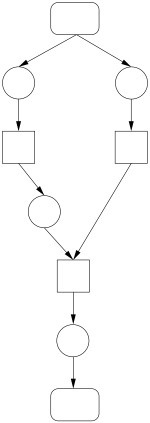


DOT File Sample

Task Description

- Described by an id and a size
  - The size correspond to the `amount` parameter of the task creator
  - Expressed in flops

```
digraph G {
# Tasks
1 [size="42000000000.00"];
2 [size="42000000000.00"];
3 [size="42000000000.00"];

}
```

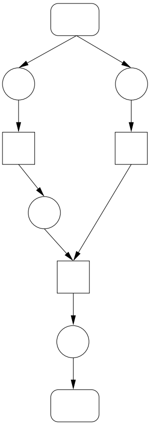


DOT File Sample

Task Description

- Described by an id and a size
  - The size correspond to the `amount` parameter of the task creator
  - Expressed in flops

```
digraph G {
# Tasks
1 [size="42000000000.00"];
2 [size="42000000000.00"];
3 [size="42000000000.00"];
# Dependencies
root->1 [size="1000000.00"];
root->2 [size="1000000.00"];
1->3    [size="1000000.00"];
2->3    [size="-1."]; # Control dependency
3->end  [size="1000000.00"];
}
```



Dependency Description

- Described by `src`→`dst` and a size
  - The size also corresponds to `amount`
  - Expressed in bytes

SimDag Code Sample

Using the DAX loader

```
int main(int argc, char **argv) {
  unsigned int cpt;
  SD_task_t task;
  xbt_dynar_t DAG;
  SD_init(&argc, argv);









  SD_exit();
  return 0;
}
```

SimDag Code Sample

Using the DAX loader

- Call the loader

```
int main(int argc, char **argv) {
  unsigned int cpt;
  SD_task_t task;
  xbt_dynar_t DAG;
  SD_init(&argc, argv);

  dag = SD_daxload(argv[1]);






  SD_exit();
  return 0;
}
```

SimDag Code Sample

Using the DAX loader

- Call the loader
- Destroy each task

```
int main(int argc, char **argv) {
  unsigned int cpt;
  SD_task_t task;
  xbt_dynar_t DAG;
  SD_init(&argc, argv);

  dag = SD_daxload(argv[1]);

  xbt_dynar_foreach(dag, cpt, task){
    SD_task_destroy(task);
  }

  SD_exit();
  return 0;
}
```

SimDag Code Sample

Using the DOT loader

```
int main(int argc, char **argv) {
  unsigned int cpt;
  SD_task_t task;
  xbt_dynar_t DAG;
  SD_init(&argc, argv);







  SD_exit();
  return 0;
}
```

SimDag Code Sample

Using the DOT loader

- Call the loader

```
int main(int argc, char **argv) {
  unsigned int cpt;
  SD_task_t task;
  xbt_dynar_t DAG;
  SD_init(&argc, argv);

  dag = SD_dotload(argv[1]);






  SD_exit();
  return 0;
}
```

SimDag Code Sample

Using the DOT loader

- Call the loader
- Destroy each task

```
int main(int argc, char **argv) {
  unsigned int cpt;
  SD_task_t task;
  xbt_dynar_t DAG;
  SD_init(&argc, argv);

  dag = SD_dotload(argv[1]);

  xbt_dynar_foreach(dag, cpt, task){
    SD_task_destroy(task);
  }

  SD_exit();
  return 0;
}
```

Retrieving Information About a Task

Parameters of the constructor

- `SD_task_get_name(task)`
  - Can also be modified with `SD_task_set_name(task, "new_name")`
- `SD_task_get_data(task)`
  - Returns a `(void*)`, has to be casted by the user
  - Data can be attached at any time: `SD_task_set_data(task, (void*) data)`
- `SD_task_get_amount(task)` (non modifiable)
- `SD_task_get_kind(task)` (non modifiable)

Dependencies of task `T`

- Tasks on which `T` depends: `SD_task_get_parents(T)`
- Tasks depending on `T`: `SD_task_get_children(T)`
- Both functions return a `xbt_dynar_t`

Retrieving everything

- `SD_task_dump(task)`

SimDag Code Sample

```
int main(int argc, char **argv) {
  SD_task_t task;
  SD_init(&argc, argv);

  task = SD_task_create_comp_seq("src", NULL, 1E9);

  SD_task_dump(task);

  SD_exit();
  return 0;
}
```

```
[0.000000] [sd_task/INFO] Displaying task src
[0.000000] [sd_task/INFO]   - state:    not runnable
[0.000000] [sd_task/INFO]   - kind: sequential computation
[0.000000] [sd_task/INFO]   - amount: 1000000000
[0.000000] [sd_task/INFO]   - Dependencies to satisfy: 0
[0.000000] [sd_task/INFO]   - pre-dependencies:
[0.000000] [sd_task/INFO]   - post-dependencies:
```

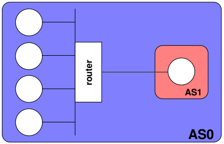
Agenda

- Introduction
- DAGs in SimDag
- Experimental Environment
  - Computing and Network Resources
  - Retrieving Information About Workstations
  - Retrieving Information About Network
- Scheduling Tasks on Resources
- Running the Simulation
- A Complete Scheduling Simulator Example
- Conclusion

How to Represent Resources

Types of resources

- Single Hosts: `id` and `power`
- Links: `id`, `latency` and `bandwidth`
- Clusters
  - `id` and name (`prefix radical suffix`)
  - `power`
  - private link latency (`lat`) and bandwidth (`bw`)
  - backbone latency (`bb_lat`) and bandwidth (`bb_bw`)
  - `router`



- routes: `src` and `dst`
- Resources grouped in Autonomous Systems (`AS`)
- Description in an XML platform file

Platform File Sample

cluster_and_one_host.xml

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">
<platform version="3">
  <AS id="AS0" routing="Full">
    <cluster id="my_cluster_1" prefix="c-" suffix=".me" radical="0-4"
             power="1000000000" bw="125000000" lat="5E-5"
             bb_bw="2250000000" bb_lat="5E-4"
             router_id="router1"/>

    <AS id="AS1" routing="none">
      <host id="host1" power="1000000000"/>
    </AS>

    <link id="link1" bandwidth="100000" latency="0.01"/>

    <ASroute src="my_cluster_1" dst="AS1" gw_src="router1" gw_dst="host1">
      <link_ctn id="link1"/>
    </ASroute>
  </AS>
</platform>
```

Adding Extra Information on Resources

Properties

- Attach arbitrary data to any `host` or `link`
- Key/Value dictionaries
- User responsible of management (not simulated by SimGrid)

```
<host id="host1" power="1000000000"/>
  <prop key="memory" value="1000000000"/>
  <prop key="OS" value="Linux 2.6.22-14"/>
</host>
```

Dynamic availability traces

- Connects to a trace detailing the changing behavior of a resource
  - Availability (ON/OFF) and power for `host`
  - Availability, latency, and bandwidth for `link` and `route`

```
<host id="host1" power="1000000000"/>
<trace id="mytrace" file="host1.trace"/>
<trace:connect element="host1" kind="POWER" trace="mytrace"/>
```

SimDag Code Sample

Loading the platform file

```
int main(int argc, char **argv) {
  SD_init(&argc, argv);



  SD_exit();

  return 0;
}
```

SimDag Code Sample

Loading the platform file

- Use the SD_create_environment function
  - Takes a filename as input
  - Creates an array of SD_workstation_t
    - workstation: a compute host and a set of network links
  - No need for deployment file in SimDag

```
int main(int argc, char **argv) {
  SD_init(&argc, argv);

  SD_create_environment(argv[1]);

  SD_exit();

  return 0;
}
```

Retrieving Information About Workstations

Getting all the workstations

- `SD_workstation_get_number()`
  - Returns the number of workstations
- `SD_workstation_get_list()`
  - Returns the workstation list

Workstation specific information

- `SD_workstation_get_name(workstation)` (non modifiable)
- `SD_workstation_get_power(workstation)` (non modifiable)
- `SD_workstation_get_data(workstation)`
  - Returns a `(void*)`, has to be casted by the user
  - Data can be attached at any time
    `SD_task_set_data(workstation, (void*) data)`
- `SD_workstation_get_properties(workstation)`
  - Returns a `xbt_dict_t` will all the properties
- `SD_workstation_get_property_value (workstation, name)`
  - Returns the value stored for property called `name`

Retrieving Information About Network

Getting all the links

- `SD_link_get_number()` returns the number of links
- `SD_link_get_list()` returns the list of links

Link specific information

- `SD_link_get_name(link)` (non modifiable)
- `SD_link_get_data(workstation)`
  - Returns a `(void*)`, has to be casted by the user
  - Data can be attached at any time: `SD_link_set_data(link, (void*) data)`
- `SD_link_get_sharing_policy (link)`
  - May this link cause contention or not

Route specific information

- `SD_route_get_size (src_workstation, dst_workstation)`
  - Returns the number of links on the route between two workstations
- `SD_route_get_list (src_workstation, dst_workstation)`
  - Returns the list of links on the route between two workstations

Agenda

Scheduling Default Parallel Tasks

Issue: different semantics

- Creation
  - `task = SD_task_create(name, data, amount)`
  - Where `amount` is the compute work to do or data size to transfer
- Scheduling
  - `SD_task_schedule(task, workstation_nb, workstation_list, computation_amount, communication_amount, rate)`

Things to be done

- Determine
  - How many workstations to use
  - Which workstations
- Distribute
  - the work to compute (in an array)
  - the data to communicate (in a communication matrix)
- Can't schedule a transfer task before having scheduled its parent and child

SimDag Code Sample

Scenario

- Schedule 2 compute tasks and 1 transfer on `cluster_and_one_host.xml`
  - Compute task `t1` ⇒ on the cluster (parallel)
  - Compute task `t2` ⇒ on the host (sequential)
  - Data transfer `c1` from the cluster to the host

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

  [...]
}
```

SimDag Code Sample

Step 1

- Declare the needed data structures

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

  [...]
}
```

SimDag Code Sample

Step 1

- Declare the needed data structures
  - The task themselves

```
int main(int argc, char **argv) {
  int i;
  SD_task_t c1, c2, t1;











  [...]
}
```

SimDag Code Sample

Step 1

- Declare the needed data structures
  - The task themselves
  - Scheduling and complete lists of workstations

```
int main(int argc, char **argv) {
  int i;
  SD_task_t c1, c2, t1;
  SD_workstation_t *ws_list, *workstations;

  [...]
}
```

SimDag Code Sample

Step 1

- Declare the needed data structures
  - The task themselves
  - Scheduling and complete lists of workstations
  - Computation array and communication matrix

```
int main(int argc, char **argv) {
  int i;
  SD_task_t c1, c2, t1;
  SD_workstation_t *ws_list, *workstations;
  double *computation_amount, *communication_amount;

  [...]
}
```

SimDag Code Sample

Step 2

- Initialize SimDag

```
int main(int argc, char **argv) {
  int i;
  SD_task_t c1, c2, t1;
  SD_workstation_t *ws_list, *workstations;
  double *computation_amount, *communication_amount;

  SD_init(&argc, argv);

  [...]
}
```

SimDag Code Sample

Step 2

- Initialize SimDag
- Load the platform file to create resources
  - and get the complete list of workstations

```
int main(int argc, char **argv) {
  int i;
  SD_task_t c1, c2, t1;
  SD_workstation_t *ws_list, *workstations;
  double *computation_amount, *communication_amount;

  SD_init(&argc, argv);
  SD_create_environment("cluster_and_one_host.xml");
  workstations = SD_workstation_get_list();




  [...]
}
```

SimDag Code Sample

Step 3

- Creating the tasks
  - amount represents
    - The total number of flops to compute for c1 and c2

```
int main(int argc, char **argv) {
  int i;
  SD_task_t c1, c2, t1;
  SD_workstation_t *ws_list, *workstations;
  double *computation_amount, *communication_amount;

  SD_init(&argc, argv);
  SD_create_environment("cluster_and_one_host.xml");
  workstations = SD_workstation_get_list();

  c1 = SD_task_create("c1", NULL, 1e9); /* 1 billion flops */
  c2 = SD_task_create("c2", NULL, 1e9); /* 1 billion flops */

  [...]
}
```

SimDag Code Sample

Step 3

- Creating the tasks
  - `amount` represents
    - The total number of flops to compute for `c1` and `c2`
    - The total number of bytes to transfer for `t1`

```
int main(int argc, char **argv) {
  int i;
  SD_task_t c1, c2, t1;
  SD_workstation_t *ws_list, *workstations;
  double *computation_amount, *communication_amount;

  SD_init(&argc, argv);
  SD_create_environment("cluster_and_one_host.xml");
  workstations = SD_workstation_get_list();

  c1 = SD_task_create("c1", NULL, 1e9); /* 1 billion flops */
  c2 = SD_task_create("c2", NULL, 1e9); /* 1 billion flops */
  t1 = SD_task_create("t1", NULL, 1e9); /* 1 GB */
  [...]
}
```

SimDag Code Sample

Step 4

- Scheduling parallel compute task c1 on four hosts

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

SimDag Code Sample

Step 4

- Scheduling parallel compute task `c1` on four hosts
  - Allocate `ws_list`, `computation_amount`, and `communication_amount`

```
int main(int argc, char **argv) {
  [...]
  ws_list = (SD_workstation_t*) calloc (4, sizeof(SD_workstation_t));
  computation_amount = (double*) calloc (4, sizeof(double));
  communication_amount = (double*) calloc (16, sizeof(double)); /* 4x4 matrix */
}
```

SimDag Code Sample

Step 4

- Scheduling parallel compute task c1 on four hosts
  - Allocate ws_list, computation_amount, and communication_amount
  - Fill these structures
    - Evenly distribute the amount of work among hosts
    - The communication matrix remains empty

```
int main(int argc, char **argv) {
  [...]
  ws_list = (SD_workstation_t*) calloc (4, sizeof(SD_workstation_t));
  computation_amount = (double*) calloc (4, sizeof(double));
  communication_amount = (double*) calloc (16, sizeof(double)); /* 4x4 matrix */

  for (i=0;i<4;i++){
    ws_list[i]=workstations[i];
    computation_amount[i] = SD_task_get_amount(c1)/4.;
  }

}
```

SimDag Code Sample

Step 4

- Scheduling parallel compute task c1 on four hosts
  - Allocate ws_list, computation_amount, and communication_amount
  - Fill these structures
    - Evenly distribute the amount of work among hosts
    - The communication matrix remains empty
  - Call SD_task_schedule

```
int main(int argc, char **argv) {
  [...]
  ws_list = (SD_workstation_t*) calloc (4, sizeof(SD_workstation_t));
  computation_amount = (double*) calloc (4, sizeof(double));
  communication_amount = (double*) calloc (16, sizeof(double)); /* 4x4 matrix */

  for (i=0;i<4;i++){
    ws_list[i]=workstations[i];
    computation_amount[i] = SD_task_get_amount(c1)/4.;
  }

  SD_task_schedule(c1,4,ws_list,computation_amount,communication_amount,-1);

  [...]
}
```

SimDag Code Sample

Step 5

- Scheduling parallel compute task c2 on one host

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

  [...]
}
```

SimDag Code Sample

Step 5

- Scheduling parallel compute task c2 on one host
  - Reallocate computation_amount
    - Don't care about ws_list and communication_amount

```
int main(int argc, char **argv) {
  [...]
  computation_amount = (double*) realloc (computation_amount, sizeof(double));

  [...]
}
```

SimDag Code Sample

Step 5

- Scheduling parallel compute task c2 on one host
  - Reallocate computation_amount
    - Don't care about ws_list and communication_amount
  - Set computation_amount[0] to amount

```
int main(int argc, char **argv) {
  [...]
  computation_amount = (double*) realloc (computation_amount, sizeof(double));
  computation_amount[0] = SD_task_get_amount(c2);

  [...]
}
```

SimDag Code Sample

Step 5

- Scheduling parallel compute task c2 on one host
  - Reallocate computation_amount
    - Don't care about ws_list and communication_amount
  - Set computation_amount[0] to amount
  - Call SD_task_schedule
    - Direct mention of the host and no communication matrix

int main(int argc, char **argv) {

```
  [...]
  computation_amount = (double*) realloc (computation_amount, sizeof(double));
  computation_amount[0] = SD_task_get_amount(c2);
  SD_task_schedule(c2,1,&(workstations[5]),computation_amount,NULL,-1);
  [...]
}
```

SimDag Code Sample

Step 6

- Scheduling data transfer task t1 from cluster to host

```
int main(int argc, char **argv) {
  [...]
  computation_amount = (double*) realloc (computation_amount, sizeof(double));
  computation_amount[0] = SD_task_get_amount(c2);
  SD_task_schedule(c2,1,&(workstations[5]),computation_amount,NULL,-1);








  [...]
}
```

SimDag Code Sample

Step 6

- Scheduling data transfer task `t1` from cluster to host
  - Reallocate `computation_amount` and `communication_amount`
    - `computation_amount` is actually useless

```
int main(int argc, char **argv) {
  [...]
  computation_amount = (double*) realloc (computation_amount, sizeof(double));
  computation_amount[0] = SD_task_get_amount(c2);
  SD_task_schedule(c2,1,&(workstations[5]),computation_amount,NULL,-1);

  computation_amount = (double*) realloc (computation_amount, 5*sizeof(double));
  communication_amount = (double*) realloc (communication_amount, 25*sizeof(double));






  [...]
}
```

SimDag Code Sample

Step 6

- Scheduling data transfer task t1 from cluster to host
  - Reallocate computation_amount and communication_amount
    - computation_amount is actually useless
  - Fill communication_amount

```
int main(int argc, char **argv) {
  [...]
  computation_amount = (double*) realloc (computation_amount, sizeof(double));
  computation_amount[0] = SD_task_get_amount(c2);
  SD_task_schedule(c2,1,&(workstations[5]),computation_amount,NULL,-1);

  computation_amount = (double*) realloc (computation_amount, 5*sizeof(double));
  communication_amount = (double*) realloc (communication_amount, 25*sizeof(double));

  for(i=0; i<4;i++)
    communication_amount[i*5+4]=SD_task_get_amount(t1)/4.;


  [...]
}
```

SimDag Code Sample

Step 6

- Scheduling data transfer task t1 from cluster to host
  - Reallocate computation_amount and communication_amount
    - computation_amount is actually useless
  - Fill communication_amount
  - Call SD_task_schedule
    - Use workstations directly as all hosts are involved

```
int main(int argc, char **argv) {
  [...]
  computation_amount = (double*) realloc (computation_amount, sizeof(double));
  computation_amount[0] = SD_task_get_amount(c2);
  SD_task_schedule(c2,1,&(workstations[5]),computation_amount,NULL,-1);

  computation_amount = (double*) realloc (computation_amount, 5*sizeof(double));
  communication_amount = (double*) realloc (communication_amount, 25*sizeof(double));

  for(i=0; i<4;i++)
    communication_amount[i*5+4]=SD_task_get_amount(t1)/4.;

  SD_task_schedule(t1,5,workstations,computation_amount,communication_amount,-1);
  [...]
}
```

SimDag Code Sample

Step 7

- Unschedule the tasks

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

  SD_task_unschedule(c1);
  SD_task_unschedule(c2);
  SD_task_unschedule(t1);

}
```

SimDag Code Sample

Step 7

- Unschedule the tasks
- Clean stuff

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

  SD_task_unschedule(c1);
  SD_task_unschedule(c2);
  SD_task_unschedule(t1);

  free(computation_amount);
  free(communication_amount);
  free(ws_list);

  SD_exit();

  return 0;
}
```

SimDag Code Sample

Step 7

- Unschedule the tasks
- Clean stuff
- Remark: The simulation itself is not implemented in this example

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

  SD_task_unschedule(c1);
  SD_task_unschedule(c2);
  SD_task_unschedule(t1);

  free(computation_amount);
  free(communication_amount);
  free(ws_list);

  SD_exit();

  return 0;
}
```

Auto-Scheduling Typed Tasks

- When scheduling DAGs
  - Compute tasks run on one host only
  - Data transfers are point-to-point communications

⇒ Get rid off all the useless complexity of parallel tasks
  - Thanks to typed tasks
- Creation
  - `compute_task = SD_task_create_comp_seq(name, data, amount)`
  - `transfer_task = SD_task_create_comm_e2e(name, data, amount)`
- Scheduling
  - `SD_task_schedulev(task, workstation_nb, workstation_list)`
  - `SD_task_schedulel(task, workstation_nb, ...)`
  - `amount` will be directly used
- Transfers can be auto-scheduled

SimDag Code Sample

Scenario

- Schedule 2 compute tasks and 1 transfer on `cluster_and_one_host.xml`
  - Compute task t1 ⇒ on one host of the cluster (sequential)
  - Compute task t2 ⇒ on the host (sequential)
  - Point-to-point data transfer c1 from the cluster to the host

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

  [...]
}
```

SimDag Code Sample

Step 1

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

  [...]
}
```

SimDag Code Sample

Step 1

- Declare the tasks

```
int main(int argc, char **argv) {
  SD_task_t c1, c2, t1;









  [...]
}
```

SimDag Code Sample

Step 1

- Declare the tasks
- Declare the list of workstations

```
int main(int argc, char **argv) {
  SD_task_t c1, c2, t1;
  SD_workstation_t *workstations;









  [...]
}
```

SimDag Code Sample

Step 1

- Declare the tasks
- Declare the list of workstations
- Initialize SimDag

```
int main(int argc, char **argv) {
  SD_task_t c1, c2, t1;
  SD_workstation_t *workstations;

  SD_init(&argc, argv);

  [...]
}
```

SimDag Code Sample

Step 1

- Declare the tasks
- Declare the list of workstations
- Initialize SimDag
- Load the platform file to create resources
  - and get the complete list of workstations

```
int main(int argc, char **argv) {
  SD_task_t c1, c2, t1;
  SD_workstation_t *workstations;

  SD_init(&argc, argv);

  SD_create_environment("cluster_and_one_host.xml");
  workstations = SD_workstation_get_list();





  [...]
}
```

SimDag Code Sample

Step 2

- Create the tasks
  - `amount` represents
    - The total number of flops to compute for `c1` and `c2`

```
int main(int argc, char **argv) {
  SD_task_t c1, c2, t1;
  SD_workstation_t *workstations;

  SD_init(&argc, argv);

  SD_create_environment("cluster_and_one_host.xml");
  workstations = SD_workstation_get_list();

  c1 = SD_task_create_comp_seq("c1", NULL, 1e9); /* 1 billion flops */
  c2 = SD_task_create_comp_seq("c2", NULL, 1e9); /* 1 billion flops */


  [...]
}
```

SimDag Code Sample

Step 2

- Create the tasks
  - `amount` represents
    - The total number of flops to compute for `c1` and `c2`
    - The total number of bytes to transfer for `t1`

```
int main(int argc, char **argv) {
  SD_task_t c1, c2, t1;
  SD_workstation_t *workstations;

  SD_init(&argc, argv);

  SD_create_environment("cluster_and_one_host.xml");
  workstations = SD_workstation_get_list();

  c1 = SD_task_create_comp_seq("c1", NULL, 1e9); /* 1 billion flops */
  c2 = SD_task_create_comp_seq("c2", NULL, 1e9); /* 1 billion flops */

  t1 = SD_task_create_comm_e2e("t1", NULL, 1e9); /* 1 GB */
  [...]
}
```

SimDag Code Sample

Step 3

- Scheduling the tasks
  - c1 goes on the first host in the cluster

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

  SD_task_schedulel(c1, 1, workstations[0]);

}
```

SimDag Code Sample

Step 3

- Scheduling the tasks
  - c1 goes on the first host in the cluster
  - c2 goes on the host

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

  SD_task_schedulel(c1, 1, workstations[0]);
  SD_task_schedulel(c2, 1, workstations[5]);

}
```

SimDag Code Sample

Step 3

- Scheduling the tasks
  - c1 goes on the first host in the cluster
  - c2 goes on the host
  - And ... that's all!
    - t1 is automatically scheduled between these two workstations

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

  SD_task_schedulel(c1, 1, workstations[0]);
  SD_task_schedulel(c2, 1, workstations[5]);

}
```

SimDag Code Sample

Step 4

- Unschedule the tasks
  - t1 has to be explicitly unscheduled

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

  SD_task_schedulel(c1, 1, workstations[0]);
  SD_task_schedulel(c2, 1, workstations[5]);

  SD_task_unschedule(c1);
  SD_task_unschedule(c2);

  SD_task_unschedule(t1);

}
```

SimDag Code Sample

Step 4

- Unschedule the tasks
  - t1 has to be explicitly unscheduled
- Clean stuff

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

  SD_task_schedulel(c1, 1, workstations[0]);
  SD_task_schedulel(c2, 1, workstations[5]);

  SD_task_unschedule(c1);
  SD_task_unschedule(c2);

  SD_task_unschedule(t1);

  SD_exit();

  return 0;
}
```

SimDag Code Sample

Step 4

- Unschedule the tasks
  - t1 has to be explicitly unscheduled
- Clean stuff
- Remark: The simulation itself is still not implemented in this example

int main(int argc, char **argv) {

```
  [...]

  SD_task_schedulel(c1, 1, workstations[0]);
  SD_task_schedulel(c2, 1, workstations[5]);

  SD_task_unschedule(c1);
  SD_task_unschedule(c2);

  SD_task_unschedule(t1);

  SD_exit();

  return 0;
}
```

Some Useful Prediction Functions

Default Parallel Tasks

- `SD_task_get_execution_time`
  - Workstation list
  - Array of computation amounts
  - Communication matrix
- `SD_task_get_remaining_amount`
  - The simulation is hold, how much computation remains for this task?

Sequential Computation and End-to-End Communications

- `SD_workstation_get_computation_time (workstation, amount)`
- `SD_route_get_communication_time (src, dst, amount)`
- These functions do not take concurrent executions into account

Routes and workstations

- `SD_route_get_current_bandwidth (src, dest)`
- `SD_route_get_current_latency (src, dest)`
- `SD_workstation_get_available_power(workstation)`

Running the Simulation

Static Schedules

- Build the complete schedule before running the simulation
  - Call a `SD_task_schedule*` function for each task
- Then call `SD_simulate(-1.)`
  - It will stop when all the work has been done
  - Or if no more tasks are reachable

Dynamic Schedules

- Build the schedule during the simulation
- Two options
  - Hold the simulation every X seconds to take more decisions: `SD_simulate(X)`
  - Add watchpoints on the state of tasks
    - `SD_task_watch (task, state)`
    - The simulation will be hold each time a watch point is reached
    - For in time when a task goes from `SD_TASK_RUNNING` to `SD_TASK_DONE`
- This requires to add an outer loop
- Dynamic rescheduling is possible with `SD_task_unschedule`

What You Can Get After the Simulation

- When the task did actually start
  - `SD_task_get_start_time (task)`
- When the task did actually finish
  - `SD_task_get_finish_time (task)`
- How many workstation were used to execute a task
  - `SD_task_get_workstation_count (task)`
- And which ones
  - `SD_task_get_workstation_list (task)`
- Now you can plot your Gantt chart and analyze your performance metrics
  - Using either Jedule or Pajé built-in instrumentation

Agenda

A Complete Scheduling Simulator Example

The Min-Min List Scheduling Algorithm

- For each ready task
  - get the workstation that minimizes the completion time
- select the task that has the minimum completion time on its best workstation
  - And schedule it there
- Full code available at `$SIMGRID_HOME/examples/simdag/scheduling/minmin_test.c`

What follows in the code sample

- Some management functions to attach attributes to workstations
  - Availability time
  - Last task scheduled on the workstation
- Functions to
  - Find the workstation that minimizes the completion time
  - Get the list of ready tasks
- The main scheduling function

SimDag Code Sample

```
typedef struct _WorkstationAttribute {
  double available_at;
  SD_task_t last_scheduled_task;
} *WorkstationAttribute;

static void SD_workstation_allocate_attribute(SD_workstation_t ws){
  void *data = calloc(1, sizeof(struct _WorkstationAttribute));
  SD_workstation_set_data(ws, data);
}
static void SD_workstation_free_attribute(SD_workstation_t ws) {
  free(SD_workstation_get_data(ws));  SD_workstation_set_data(ws, NULL);
}
static double SD_workstation_get_available_at(SD_workstation_t ws) {
  WorkstationAttribute attr = (WorkstationAttribute) SD_workstation_get_data(ws);
  return attr->available_at;
}
static void SD_workstation_set_available_at(SD_workstation_t ws, double time){
  WorkstationAttribute attr = (WorkstationAttribute) SD_workstation_get_data(ws);
  attr->available_at = time;  SD_workstation_set_data(ws, attr);
}
static SD_task_t SD_workstation_get_last_scheduled_task( SD_workstation_t ws){
  WorkstationAttribute attr = (WorkstationAttribute) SD_workstation_get_data(ws);
  return attr->last_scheduled_task;
}
static void SD_workstation_set_last_scheduled_task(SD_workstation_t ws, SD_task_t task){
  WorkstationAttribute attr = (WorkstationAttribute) SD_workstation_get_data(ws);
  attr->last_scheduled_task=task;  SD_workstation_set_data(ws, attr);
}
```

SimDag Code Sample

double finish_on_at(SD_task_t task, SD_workstation_t workstation){

```
double result, data_available = 0., last_data_available, redist_time = 0;
unsigned int i;
SD_task_t parent, grand_parent;
xbt_dynar_t parents, grand_parents;

SD_workstation_t *grand_parent_workstation_list;

parents = SD_task_get_parents(task);

if (!xbt_dynar_is_empty(parents)) {
  /* compute last_data_available */
  last_data_available = -1.0;
  xbt_dynar_foreach(parents, i, parent) {
    if (SD_task_get_kind(parent) == SD_TASK_COMM_E2E) {      /* normal case */
      grand_parents = SD_task_get_parents(parent);

      xbt_dynar_get_cpy(grand_parents, 0, &grand_parent);
      grand_parent_workstation_list = SD_task_get_workstation_list(grand_parent);

      /* Estimate the redistribution time from this parent */
      redist_time = SD_route_get_communication_time(grand_parent_workstation_list[0],
                                                     workstation, SD_task_get_amount(parent));
      data_available = SD_task_get_finish_time(grand_parent) + redist_time;

      xbt_dynar_free_container(&grand_parents);
    }
```

SimDag Code Sample

double finish_on_at(SD_task_t task, SD_workstation_t workstation){

```
    if (SD_task_get_kind(parent) == SD_TASK_COMP_SEQ) { /* no transfer, control dependency */
      data_available = SD_task_get_finish_time(parent);
    }

    if (last_data_available < data_available)
      last_data_available = data_available;
  }

  xbt_dynar_free_container(&parents);

  result = MAX(SD_workstation_get_available_at(workstation), last_data_available) +
      SD_workstation_get_computation_time(workstation, SD_task_get_amount(task));
 } else {
  xbt_dynar_free_container(&parents);

  result = SD_workstation_get_available_at(workstation) +
      SD_workstation_get_computation_time(workstation, SD_task_get_amount(task));
 }
 return result;
}
```

SimDag Code Sample

```
SD_workstation_t SD_task_get_best_workstation(SD_task_t task) {
  int i, nworkstations = SD_workstation_get_number();
  double EFT, min_EFT = -1.0;
  const SD_workstation_t *workstations = SD_workstation_get_list();
  SD_workstation_t best_workstation;

  best_workstation = workstations[0];
  min_EFT = finish_on_at(task, workstations[0]);

  for (i = 1; i < nworkstations; i++) {
    EFT = finish_on_at(task, workstations[i]);
    if (EFT < min_EFT) {
      min_EFT = EFT;  best_workstation = workstations[i];
    }
  }
  return best_workstation;
}
xbt_dynar_t get_ready_tasks(xbt_dynar_t dax) {
  unsigned int i;
  xbt_dynar_t ready_tasks = xbt_dynar_new(sizeof(SD_task_t), NULL);
  SD_task_t task;

  xbt_dynar_foreach(dax, i, task)
    if (SD_task_get_kind(task)==SD_TASK_COMP_SEQ && SD_task_get_state(task)==SD_SCHEDULABLE)
      xbt_dynar_push(ready_tasks, &task);

  return ready_tasks;
}
```

SimDag Code Sample

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

  unsigned int cursor;
  double finish_time, min_finish_time = -1.0;
  SD_task_t task, selected_task = NULL, last_scheduled_task;
  xbt_dynar_t ready_tasks;
  SD_workstation_t workstation, selected_workstation = NULL;
  int total_nworkstations = 0;
  const SD_workstation_t *workstations = NULL;
  xbt_dynar_t dax, changed;

  SD_init(&argc, argv); /* initialization of SD */

  SD_create_environment(argv[1]);  /* creation of the environment */

  /*  Allocating the workstation attribute */
  total_nworkstations = SD_workstation_get_number();
  workstations = SD_workstation_get_list();

  for (cursor = 0; cursor < total_nworkstations; cursor++)
    SD_workstation_allocate_attribute(workstations[cursor]);

  dax = SD_daxload(argv[2]);   /* load the DAX file */

  xbt_dynar_foreach(dax, cursor, task) /* add watchpoint on task completion */
    SD_task_watch(task, SD_DONE);
  [...]
}
```

SimDag Code Sample

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

  /* Schedule the DAX root first */
  xbt_dynar_get_cpy(dax, 0, &task);
  workstation = SD_task_get_best_workstation(task);
  SD_task_schedulel(task, 1, workstation);

  while (!xbt_dynar_is_empty((changed = SD_simulate(-1.0)))) {
    /* Get the set of ready tasks */
    ready_tasks = get_ready_tasks(dax);
    if (xbt_dynar_is_empty(ready_tasks)) {
      xbt_dynar_free_container(&ready_tasks);
      xbt_dynar_free_container(&changed);
      continue;  /* there is no ready task, let advance the simulation */
    }
    xbt_dynar_foreach(ready_tasks, cursor, task) {
      workstation = SD_task_get_best_workstation(task);
      finish_time = finish_on_at(task, workstation);
      if (min_finish_time == -1. || finish_time < min_finish_time) {
        min_finish_time = finish_time;
        selected_task = task;
        selected_workstation = workstation;
      }
    }

    SD_task_schedulel(selected_task, 1, selected_workstation);
  [...]
}
```

SimDag Code Sample

```
int main(int argc, char **argv) {
    /* Manage resource dependencies */
    last_scheduled_task =  SD_workstation_get_last_scheduled_task(selected_workstation);
    if (last_scheduled_task && (SD_task_get_state(last_scheduled_task) != SD_DONE) &&
        (SD_task_get_state(last_scheduled_task) != SD_FAILED) &&
        !SD_task_dependency_exists(SD_workstation_get_last_scheduled_task(
                                   selected_workstation), selected_task))
      SD_task_dependency_add("resource", NULL, last_scheduled_task, selected_task);

    SD_workstation_set_last_scheduled_task(selected_workstation, selected_task);
    SD_workstation_set_available_at(selected_workstation, min_finish_time);

    xbt_dynar_free_container(&ready_tasks);
    xbt_dynar_free_container(&changed);
    min_finish_time = -1.;     /* reset the min_finish_time for the next round */
  }
  xbt_dynar_foreach(dax, cursor, task)
    SD_task_destroy(task);
  xbt_dynar_free_container(&dax);
  xbt_dynar_free_container(&changed);

  for (cursor = 0; cursor < total_nworkstations; cursor++)
    SD_workstation_free_attribute(workstations[cursor]);

  SD_exit();
  return 0;

}
```

Conclusion

- This 101 tutorial gives examples of the basic usage of most SimDag function
  - You should be able to code your own simulator now!
- Where to find more information on SimDag
  - in `$SIMGRID_HOME/examples/simdag`
  - in the contrib section of SimGrid
    - A set of implementations of classical DAG scheduling algorithms
    - `svn co svn://scm.gforge.inria.fr/svn/simgrid/contrib/trunk/DAGSched`
- Feel free to contribute to SimDag and the contrib section
  - And to ask questions on `simgrid-user@lists.gforge.inria.fr`