Assessing the Performance of MPI Applications Through Time-Independent Trace Replay

F. Desprez[1] **G. Markomanolis**[1] M. Quinson[2] F. Suter[3]

[1]INRIA, LIP, ENS de Lyon,
Lyon, France

[2]Nancy University, LORIA, INRIA,
Nancy, France

[3]Computing Center, CNRS, IN2P3,
Lyon-Villeurbanne, France

Avalon working group 24/01/2011

Outline

Introduction

- Dimensioning of compute clusters
- Simulation Frameworks:
  - off-line simulation
    - replay an execution trace
    - timed traces
  - on-line simulation
    - a part of the application is simulated
- The current framework follows the off-line approach

Contribution

1. Time-independent trace format
2. Decouple the acquisition of the trace from its replay
3. A trace replay tool
4. Experimental results that show the simulation accuracy, the acquisition time, the simulation time, and the trace size

Time-Independent Trace Format

Format: **id**, **type**, **volume/parameters related to the action**

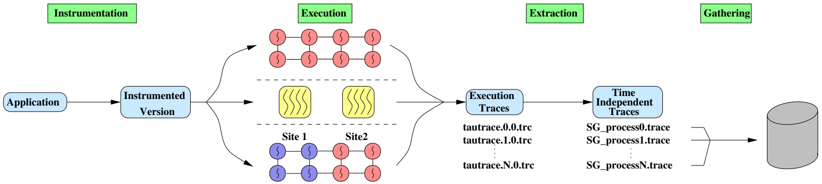
```
for (i=0; i<4; i++){
 if (myId == 0){
  /* Compute 1Mflop */
  MPI_Send(..., (myId+1));
  MPI_Recv(...);
 } else {
  MPI_Recv(...);
  /* Compute 1Mflop */
  MPI_Send(..., (myId+1)% nproc);
 }
}
```

```
p0 compute 1e6
p0 send p1 1e6
p0 recv p3

p1 recv p0
p1 compute 1e6
p1 send p2 1e6

p2 recv p1
p2 compute 1e6
p2 send p3 1e6

p3 recv p2
p3 compute 1e6
p3 send p0 1e6
```

Supported Functions

MPI actions	Trace entry
CPU burst	`<id> compute <volume>`
`MPI_Send`	`<id> send <dst_id> <volume>`
`MPI_Isend`	`<id> Isend <dst_id> <volume>`
`MPI_Recv`	`<id> recv <src_id> <volume>`
`MPI_Irecv`	`<id> Irecv <src_id> <volume>`
`MPI_Broadcast`	`<id> bcast <volume>`
`MPI_Reduce`	`<id> reduce <vcomm> <vcomp>`
`MPI_Allreduce`	`<id> allReduce <vcomm> <vcomp>`
`MPI_Barrier`	`<id> barrier`
`MPI_Comm_size`	`<id> comm_size #proc`
`MPI_Wait`	`<id> wait`

Trace Acquisition Process



Instrumentation

- Program Database Toolkit (PDT):
  - source level instrumentation
- TAU Performance System is a profiling and tracing tookit:
  - Using PDT
  - Record every MPI message
  - Measuring the hardware counters through the PAPI interface (PAPI_FP_OPS)
  - Selective instrumentation

Selective Instrumentation

- Declare a list with the functions that should be instrumented or not
- Use TAU instrumentation API:

```
        call TAU_ENABLE_INSTRUMENTATION()
        call ssor(itmax)
        call TAU_DISABLE_INSTRUMENTATION()
```

Acquisition modes

- **Regular mode**: one process per CPU
  - Need many CPUs for large instances
- **Folding mode:** more than one process per CPU
  - Limited scalability by the available memory
- **Scattering mode:** the CPUs do not necessarily belong to the same cluster
  - Many nodes available
  - Easier to acquire the traces when a lot of resources of some clusters are not available
  - The execution time depends on many factors
- **Scattering and Folding:** the combination of Folding and Scattering mode
- The trace remains the same for all the modes

Post-processing of the Execution Traces

After the execution of an instrumented application, there are:

- trace files
- event files

Example of event file:

```
49 MPI 0 "MPI_Send()   " EntryExit
1 TAUEVENT 1 "PAPI_FP_OPS" TriggerValue
```

Need to:

- *Extract* a time-independent trace from the trace and event files
- *Gather* the extracted traces on a single node

Tau2SimGrid I

A C/MPI parallel application, called `tau2simgrid`:

- Using TAU Trace Format Reader library
- No communication between different processes
- Handling all the execution modes
- Each process creates its own file independently from the other ones
- During the execution there is no need to read a trace file many times even for non blocking MPI commands

Tau2SimGrid II

Example:

```
1 0 1.42947e+06 EnterState     49
1 0 1.42947e+06 EventTrigger    1   164035532
1 0 1.4295e+06  EventTrigger   46   163840
1 0 1.4295e+06  SendMessage   0 0   163840      1 0
1 0 1.4299e+06  EventTrigger    1   164035624
1 0 1.4299e+06  LeaveState     49
```

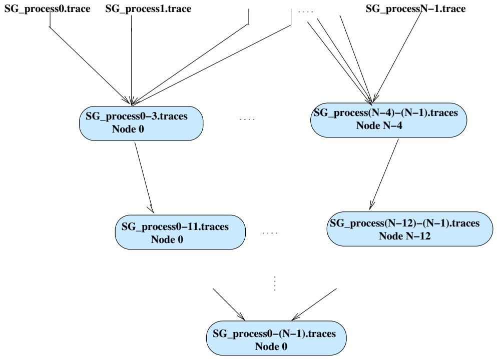
Time independent trace:

```
p1 send p0 163840
```

Gathering the traces

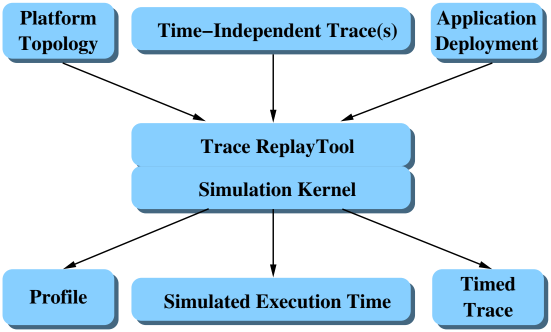
`GatherSimGrid`:

- K-nomial tree reduction
- $\log_{(K+1)} N$ steps, where N is the total number of files, and K is the arity of the tree



Trace Replay with SimGrid

Inputs and outputs of the SIMGrid trace replay framework



Platform file

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "simgrid.dtd">
<platform version="3">
 <AS id="AS_mysite" routing="Full">
  <cluster id="AS_mycluster"
           prefix="mycluster-" suffix=".mysite.fr"
           radical="0-3" power="1.17E9"
           bw="1.25E8" lat="16.67E-6"
           bb_bw="1.25E9" bb_lat="16.67E-6"/>
 </AS>
</platform>
```

Deployment file

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM "simgrid.dtd">
<platform version="3">
  <process host="mycluster-0.mysite.fr"
           function="p0"/>
  <process host="mycluster-1.mysite.fr"
           function="p1"/>
  <process host="mycluster-2.mysite.fr"
           function="p2"/>
  <process host="mycluster-3.mysite.fr"
           function="p3"/>
</platform>
```

Trace replay tool

Using MSG API:

1. A function for every action, for instance:

```
static void compute(xbt_dynar_t action){
  char *amount = xbt_dynar_get_as(action,
                     2, char *);
  m_task_t task = MSG_task_create(NULL,
                     parse_double(amount),
                     0, NULL);
  MSG_task_execute(task);
  MSG_task_destroy(task);
}
```

2. Register the function:

```
MSG_action_register("compute", compute);
```

3. Call the function `MSG_action_trace_run`

```
<process host="mycluster-0.mysite.fr" function="p0">
<argument value="SG_process0.trace"/>
</process>
```

Calibration I

- Computation:
  - Execute a small instrumented instance of the target application
  - Determine the number of flops of each event and the time spent to compute them
  - Compute the flop rate for every event
  - Compute a weighted average of the flop rates for each process
  - Compute the average flop rate for all the process set
  - Compute an average over these five runs
- Bandwidth: We use the nominal value of the links

Calibration II

- Latency:
  - Executing `Pingpong_Send_Recv` experiment of SkaMPI benchmark
  - Divide the value obtained for a 1-byte message by six
- Piece-wise linear model used by SIMGrid dedicated to MPI communications:
  - Latency and bandwidth correction factors

Benchmarks, Clusters

NAS Parallel Benchmarks (NPB):

- We use the LU factorization (LU) program
- We have 7 different *classes*, denoting different problem sizes: S (the smallest), W, A, B, C, D, and E (the largest)

Clusters:

- *Bordereau*: 93 2.6GHz Dual-Proc, Dual-Core AMD Opteron 2218 nodes. Single 10 Gigabit switch
- *Gdx*: 86 2.0 GHz Dual-Proc AMD Opteron 246 scattered across 18 cabinets
- These two clusters are interconnected through a dedicated 10 Gigabit network

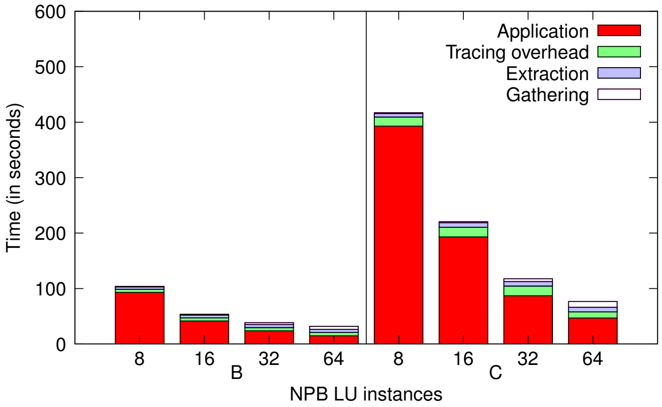
Software

One of the key concept of the Grid'5000 experimental platform is to offer its users the capacity to deploy their own system image at will

Debian Lenny image:

- Kernel (v2.6.25.9)
- Perfctr driver (v2.6.38)
- TAU (v2.18.3)
- PDT (v3.14.1)
- PAPI (v3.7.0)
- NAS Parallel Benchmarks (v3.3)
- OpenMPI (v1.3.3)
- SIMGrid (v3.6-r9069)

Acquisition time



- The wrost value for the percentage of extraction and gathering steps is 34.91% for class B, 64 processes
- Many “what-if” sceranrios can be applied with one acquisition

Evolution of the execution time

Evolution of the execution time of an instrumented LU benchmark executed by 64 processes with regard to the acquisition mode

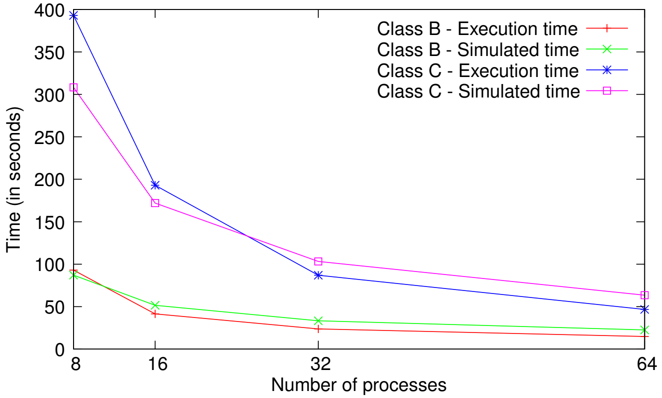
	Acquisition mode	R	F-2	F-4	F-8	F-16	F-32	S-2	SF-(2,2)	SF-(2,4)	SF-(2,8)	SF-(2,16)
	Number of nodes	64	32	16	8	4	2	(32,32)	(16,16)	(8,8)	(4,4)	(2,2)
B	Execution Time (in sec.)	20.73	52.96	88.66	179.07	347.27	689.18	37.54	79.19	134.05	277.25	505.64
	Ratio to regular mode	1	2.55	4.28	8.64	16.75	33.25	1.81	3.82	6.47	13.37	24.39
C	Execution Time (in sec.)	57.77	143.45	272.45	511.75	1,011.59	1,970.05	85.71	211.95	421.71	772.56	1,442.79
	Ratio to regular mode	1	2.22	4.13	7.79	15.14	31.79	1.48	3.67	7.3	13.37	24.97

- The time needed to execute the instrumented application increases roughly linearly with the folding factor
- During the Scattering the overhead comes from the wide area network and the progression of the execution depends on the slowest cluster
- Same simulated time for all the acquisition modes with variations less than 1%

Analysis of Trace Sizes

	#Processes	Trace size in MiB: TAU	Trace size in MiB: Time Independent	$\frac{\text{TAU}}{\text{SimGrid}}$	#Actions (in millions)
Class B	8	320.2	29.9	10.71	2.03
	16	716.5	72.6	9.87	4.87
	32	1,509	161.3	9.36	10.55
	64	3,166.1	344.9	9.18	22.73
Class C	8	508.2	48.4	10.5	3.23
	16	1,136.5	117	9.71	7.75
	32	2,393	256.8	9.32	16.79
	64	5,026.1	552.5	9.1	36.17

Accuracy of Time-Independent Trace Replay

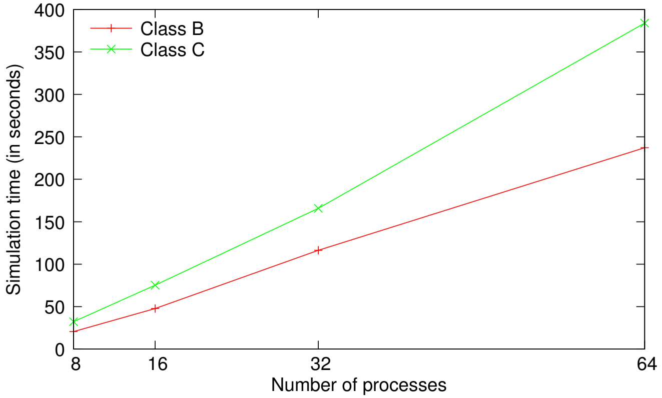


- The local relative error may be quite high (up to 51.5% for Class B on 64 processes) and not constant
- Trends are correctly predicted

Acquiring a Large Trace

- Executing the acquisition process on the *bordereau* cluster for a Class D instance executed on 1,024 processes
- We only use 32 nodes, 128 individual cores, and a folding factor of 8
- Less than 25 minutes to acquire (including extraction and gathering) the time-independent trace
- Its size is 32.5 GiB, which is 7.8 times smaller than the TAU trace (252.5 GiB)

Simulation Time



- The speed of simulating actions is around to 94205 actions/second
- Linear with the number of processes

Conclusion and Future Work

Conclusion

- Time-independent traces for the off-line simulation of MPI applications
- Tools for our Framework:
  - TAU is a well known profiling tool since 1997
  - SIMGrid, validaded simulator
- Some issues have to be solved

Future Work

- Solve the accuracy and simulation time issues
- We also aim at exploring techniques to reduce the size of the traces
- Compare off-line simulations results with those produced by on-line simulators

Thank you!
Questions?