The SimGrid Project
Simulation and Deployment of Distributed Applications

Arnaud Legrand
CNRS, MESCAL INRIA project
Laboratoire Informatique et Distribution

Martin Quinson
Université Henri Poincaré, Nancy 1
LORIA

Henri Casanova Kayo Fujiwara
Dept. of Information and Computer Sciences
University of Hawaii at Manoa

Developing efficient large-scale concurrent applications poses many challenges:

- Understanding the performance behavior of the code is non-trivial
- Conducting experiments in real-world large-scale platforms is non-trivial
  - Requires a fully functional implementation
  - Limited to a few particular platform configurations
  - In many cases, non-repeatable
- Accurate and validated simulations results are elusive
  - Potentially more accurate emulation is extremely time consuming
- Simulation code is often "throw-away" and may differ from the real code

The SimGrid project addresses all the above challenges via a multi-component software infrastructure for application prototyping, development, and deployment.

Available at **http://simgrid.gforge.inria.fr/**

Examples of target applications:

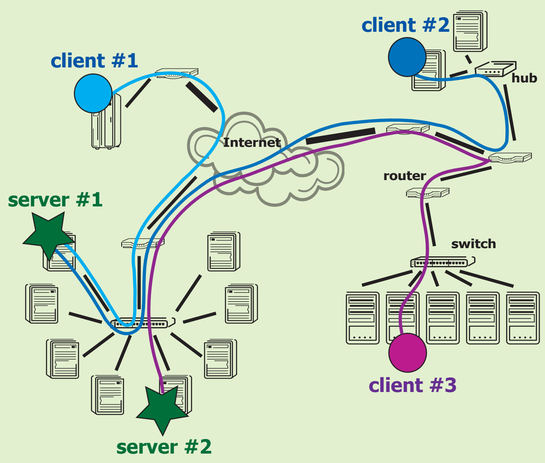
- A parallel linear system solver on a commodity cluster
- A parallel rendering application running on a network of workstations
- A scientific simulation running on a multi-site high-end grid platform
- A network monitoring application running on a wide-area network
- A peer-to-peer file-sharing application running on volatile Internet hosts

Features

- Fast and accurate simulation capabilities (SURF)
- Ability to run the same code in full or partial simulation mode or in real-world mode (GRAS, SMPI)
- An API for rapid application prototyping to test and evaluate distributed algorithms (MSG)
  - Only in simulation mode
- An API for application development to obtain fast, robust and portable application code (GRAS)
  - Either in simulation or in real-world mode
- An API for MPI application simulation to study the effect of platform heterogenity (SMPI)
  - In partial simulation mode

Application and algorithm prototyping

- Enables the the easy prototyping of distributed algorithms
  - No need to realize a complete implementation
  - Just focus on the fundamentals of distributed computing
- Uses a convenient and standard abstraction of a distributed applications
  - Applications consist of processes
  - Processes can be created, suspended, resumed and terminated dynamically
  - Processes can synchronize by exchanging tasks
  - Tasks have a communication payload and an execution payload
  - All processes are in the same address space
    - Enables convenient communication via global data structure




```c
int client(int argc, char **argv) {
  m_task_t local, remote, ack;
  m_host_t destination;

  destination = MSG_get_host_by_name(server_host_name);

  /* simulated data transfer */
  remote = MSG_task_create("Remote", 30.0, 3.2);
           /* 30.0 MFlop, 3.2 MB */
  MSG_task_put(remote, destination, PORT_22);

  /* simulated task execution */
  local = MSG_task_create("Loca", 10.50, 3.2);
          /* 10.50 MFlop, 3.2 MB */
  MSG_task_execute(local);

  /* simulate data reception */
  MSG_task_get(&ack, PORT_23);

  return 0;
}
```

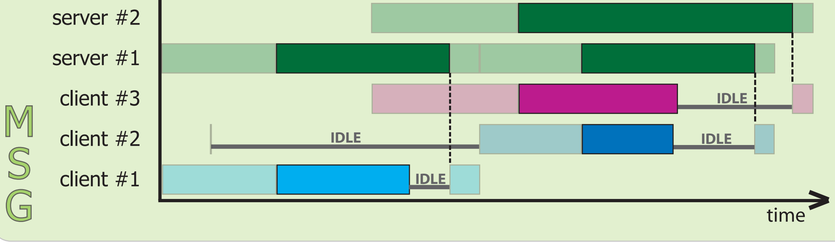
```c
int server(int argc, char **argv) {
  m_task_t task;
  m_host_t source;

  while(1) {
    /* simulated data reception */
    MSG_task_get(&task, PORT_22);

    /* simulated task execution */
    MSG_task_execute(task);

    source = MSG_get_host_by_name(client_host_name);

    /* simulated data transfer */
    ack = MSG_task_create("Ack", 0, 0.01);
          /* 0 MFlop, 10KB */
    MSG_task_put(ack, source, PORT_23);
  }
  return 0;
}
```



Gantt chart for an execution of the above code for **2 servers** and **3 clients**

Dark portions denote computations, light portions denote communications

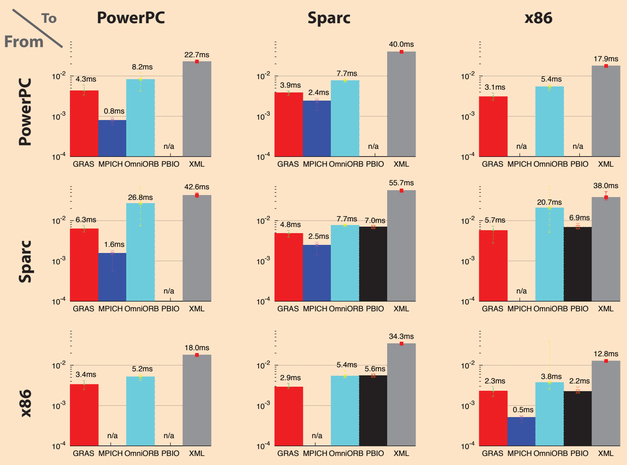
Concurrent communications interfere with each other as the TCP flows share network links

Application development

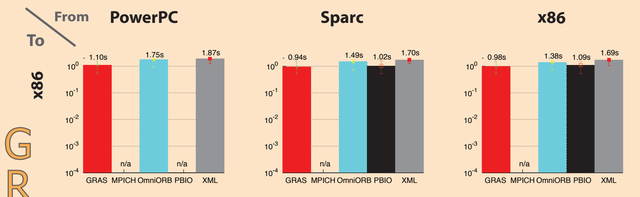
- Convenience
  - API for rapid development of real-world distributed applications
  - Simple and cross-architecture communication of complex data structures
- Portability (Linux, Mac OSX, Solaris, AIX, IRIX; 12 CPU architectures)
- Performance via efficient communication (computation unchanged)
- Resulting application is production, not prototype

Application testing and evaluation

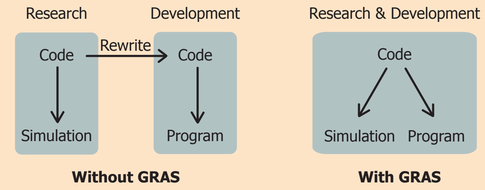
- Unmodified code run in simulation mode or in real-world mode
- Automatic benchmarking of application code for simulation (CPU)
- Automatic computation of communication volume for simulation (network)



Average time to exchange one Pastry message on a LAN (in seconds) for MPICH, OmniORB, PBIO, and XML-based communication, between PowerPC, Sparc, and x86 architectures



Average time to exchange one Pastry message on a WAN (in seconds) for MPICH, OmniORB, PBIO, and XML-based communication, between PowerPC, Sparc, and x86 architectures (WAN: California - France)



Without GRAS **With GRAS**

```c
int client(int argc, char **argv) {
  gras_socket_t peer, from;
  int ping=1234, pong;

  gras_init(&argc, argv);
  gras_os_sleep(1); /* Wait for the server startup */

  gras_msgtype_declare("ping", gras_datadesc_by_name("int"));
                       /* name, payload */
  gras_msgtype_declare("pong", gras_datadesc_by_name("int"));

  peer = gras_socket_client("127.0.0.1", 4000);
  gras_msg_send(peer, gras_msgtype_by_name("ping"), &ping);
                /* dest, msgtype, payload */
  gras_msg_wait(6000, gras_msgtype_by_name("pong"), &from, &pong);
                /* timeout, wanted msgtype, &source, &payload */

  gras_exit();
  return 0;
}
```

```c
int server(int argc, char **argv) {

  gras_init(&argc, argv);

  gras_msgtype_declare("ping", gras_datadesc_by_name("int"));
  gras_msgtype_declare("pong", gras_datadesc_by_name("int"));
  gras_cb_register(gras_msg_type_by_name("ping"), ping_callback);
  gras_socket_server(4000);
  /* wait for next message (up to 600s) and handle it */
  gras_msg_handle(600.0);
  gras_exit();
  return 0;
}

int ping_callback(gras_socket_t experditor, void *payload_data) {
  int msg = *(int *)payload_data;

  GRAS_BENCH_ALWAYS_BEGIN();
  /* Some computation whose duration should be simulated */
  GRAS_BENCH_ALWAYS_END();
  /* Send data back as payload of pong message to the ping's source */
  gras_msg_send(source, gras_msgtype_by_name("pong"), &msg);
}
```

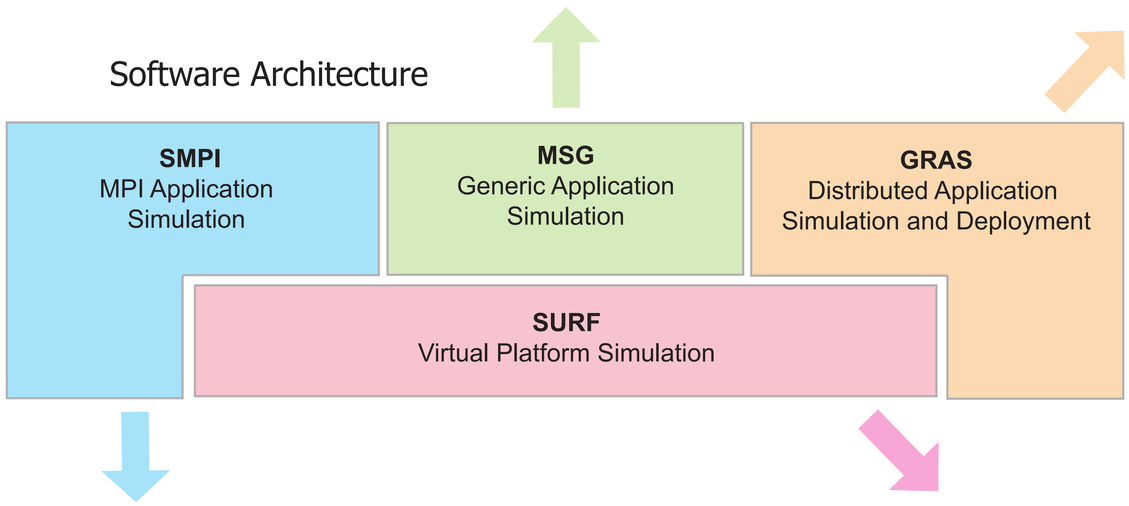
Work in Progress

- Parallel simulation for better scalability
- Port to Windows
- Native multi-threading support

Grid Application Toolbox

- Platform monitoring (CPU and network)
- Network topology discovery

Software Architecture



Why three interfaces?

- **MSG:** Rapid prototyping of distributed applications / algorithms
- **GRAS:** Development of production distributed applications
- **SMPI:** Study how an existing MPI application reacts to platform heterogenity

Simulation and Concurrency

- **MSG:** All simulated application processes run within a single process
- **GRAS:** Subsets of simulated application processes run within multiple processes on multiple hosts, for increased scalability
- **SMPI:** Each application process runs as a separate process

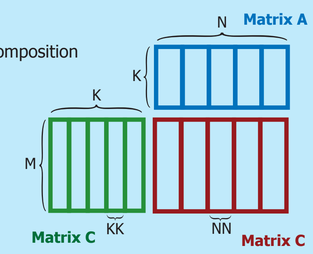
Work in Progress

Simulation of an existing MPI application

- Automatic (but directed) benchmarking of communication and computation costs during an application execution on an homogeneous platform
- Easy simulation of the application on a heterogeneous platform
- No code modification required beyond inserting benchmarking commands so that the simulation can be instanciated

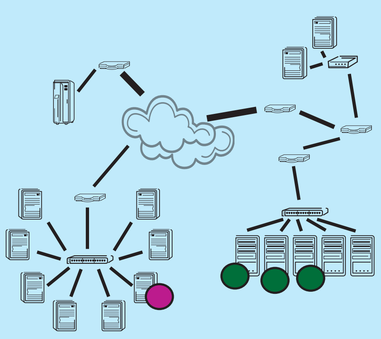
Example: 1-D Matrix Multiplication in MPI

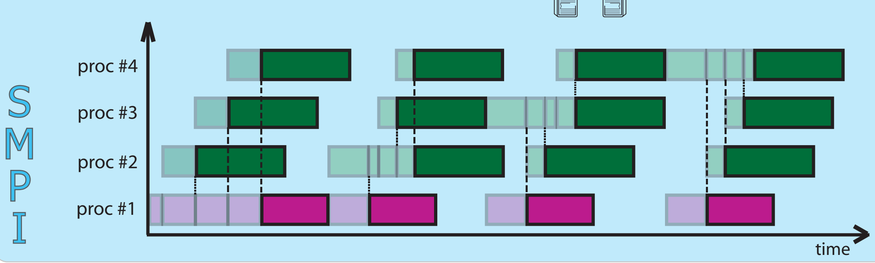
- Matrices are distributed among processors using a vertical strip decomposition
- Column blocs are broadcasted at every step




```c
void parallel_mat_mult(int M, int N, int K, double alpha,
                       double *A, double *B, double beta, double *C)
{
  int KK = K/num_proc;
  int NN = N/num_proc;
  int i,k;
  double *buf_col = calloc(M, sizeof(double));

  for(k=0; k< K ; k++) {
    if (k/KK == my_id)
      for(i=0; i<M; i++)
        buf_col[i]=A[i*KK+(k % KK)];
    MPI_Bcast(buf_col, M, MPI_DOUBLE, k / KK, MPI_COMM_WORLD);
    /* Start benchmarking */
    SMPI_BENCH_ONCE_RUN_ONCE_BEGIN();
    /* Call he CBLAS dgem() routine */
    cblas_dgemm(CblasRowMajor,CblasNoTrans,CblasNoTrans,
                M, NN, 1, alpha, buf_col, 1, &B[k*NN], NN,
                k?1.0:beta, C, NN);
    /* Stop benchmarking */
    SMPI_BENCH_ONCE_RUN_ONCE_END();
  }
}
```



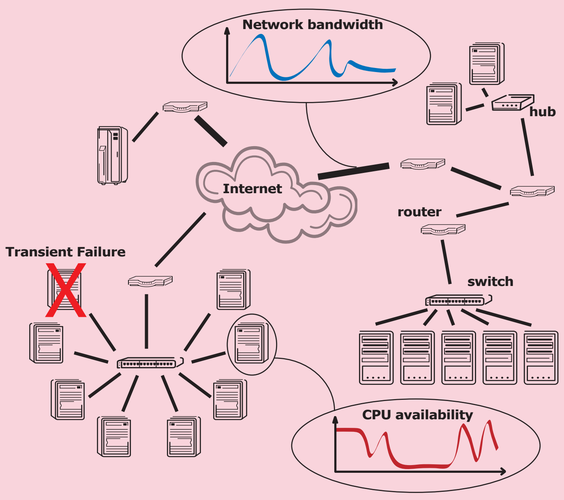


Virtual platform simulation

- Computation
- Point-to-point Communication

Features and Capabilities

- Simulation of complex communications (multi-hop routing)
- Simulation of resource sharing
- Simulation LAN and WAN links
- Topology can be imported from topology generators (such as BRITE)
- Trace-based simulation of performance variations due to external load
- Trace-based simulation of dynamic resource failures



Work in Progress

Interfacing to packet-level network simulators

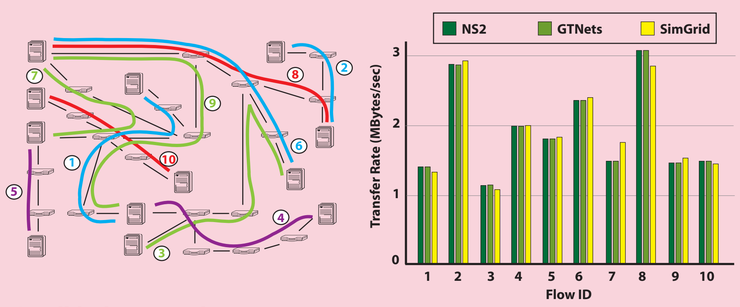
- MaxMin fairness less accurate for short-lived TCP flows
- For short-lived flows, one can use more accurate, but more expensive, packet-level simulation
- SURF will provide a seamless interface to packet-level simulators such as NS, GTNets, or SSFNet
- Users can choose between MaxMin and packet- level simulation

Simulation of resource sharing

- Consider a set of resources, R.
- Consider a set of "tasks", T
- Each task is defined as the subset of R it uses
- SURF uses the unifying **MaxMin Fairness** model: allocate as much capacity to all tasks in a way that maximizes the minimum capacity allocation over all tasks

C_r : Resource capacity	$\forall\ r \in R,\ \sum_{t \ni r} \rho_t \leq C_r$
ρ_t : Task work rate	$\forall\ r \in T, \rho_t \geq 0$
MaxMin Fairness:	$\text{maximize}\ \min_{t \in T} \rho_t$

- Used for computation and communication resources
  - Multiple TCP flows sharing links
  - Multiple CPU-bound processes sharing a CPU
  - Interference of communication and computation
  - Parallel tasks
- Eficient, accurate in many scenarios



A validation experiment

- Random topology generated with BRITE (random bandwidths and latencies)
- 10 random flows for 10 random source-destination pairs
- Each flow transfers 100 MBytes (operation in steady-state)
- Comparison between NS2, GTNets, and SimGrid

Results

- Flow transfer rates simulated by SimGrid are within +/- 15% of those obtained with packet-level simulators, with most within only a few percents
- Simulation time is orders of magnitude faster

SURF