New simulation techniques in Warped kernel

Index

Abstract

This technical report is divided in two parts. In the first one we present two new simulation protocols, called Local Rollback Frequency Model and Global Rollback Frequency Model; the explanation contains details about the protocols and the way we have modified the Warped kernel so that it be able to run them. In the second part we show how to install Warped, the added code for the new protocols and finally how to compile and run a simulation.

Introduction

We have modified the Warped kernel in order to run simulations under different protocols. These protocols are modifications of the optimist one that Warped implements, Time Warp. The idea is to "sleep" objects with a large number of rollbacks (although the objects continue receiving input events), so that they don't flood the net with antimessages, and just the objects that do correct calculus are able to advance. In some moment of the simulation, the delayed objects won't suffer any more rollbacks, so we "wake them up" to go on simulating.

These new protocols are based on the "Near Perfect State Information - NPSI" protocol, described in detail in: RINIVASAN S.; REYNOLDS, J, "Elastic time", ACM Transactions on Modeling and Computer Simulation, Vol. 8, No. 2. 103-139. April 1998.

Briefly, Elastic Time is composed of two parts:

- Identifying the Near Perfect State Information of the simulation, and
- Design a mechanism that translates the NPSI in optimism on the objects of the simulation.

There is various ways to implement each part. To connect them, we introduce the concept of potential error (PE) associated to each Logical Process *i*, to control the optimism of LP*i*. This protocol keeps an updated value of each *PE* during simulation run, at high frequencies, by evaluating a function of name *M1* that uses the state information that receives from the feedback system. Then, the function *M2* translates dynamically new values of PE_i in delays in the execution of events.

Protocols implemented above Warped

Local Rollbacks Frequency Model

Each object uses only local information. This means that each object decides to sleep or to go on with the simulation on the base of the number of rollbacks it had. Then we define the functions M1 and M2 as the following:

Function M1: The potential error of an object is the number of rollbacks that the object had from a time *T1* until the actual time *T2, with T2-T1 <= T*, where *T* is the interval after which the local number of rollbacks restarts (assigning a value of 0 to this variable).

Function M2: If the number of rollbacks for an object in a predefined interval *T* is greater than a specified umbral, the object suspends simulating, adopting a conservative behavior; however, the Logical Process where the object resides will continue accepting events from the neighborhood (although the events will not be processed until the object simulates again). If the number of rollbacks doesn't reach the specified umbral, the object simulates aggressively, as it does with the TimeWarp protocol.

To implement this protocol, we needed to include in Warped a way to inform to each Logical Process about two values: *u* y *T*. These values will be used the same way in every object of a Logical Process, and indicate that an object will simulate aggressively until it reach a number of rollbacks equal or greater than *u,* in a period of time *T*.

Here we show the pseudo code of the protocol for each Logical Process and each Warped simulation cycle. In the Algorithm 1 we use the variables *max_rollbacks* and *period* to indicate the values of umbral *u* and period *T* respectively.

Algorithm 1 - Protocol Local Rollback Frequency Model in Warped

```
1. In each Logical Process, at the simulation start:
   Read the maximum number of local rollbacks and the check period from the configuration file;
   store these values in the variables max_rollbacks and period
2. In each object, at the simulation start:
   previous_time  = 0
3. In each object, every time that the scheduler decides to invoke the Logical Process to run:
   actual_time = Warped.TotalSimulationlTime ()
   if (actual_time - previous_time >= period)
       simulateNextEvent()
       previous_time  = actual_time
       rollbacks = 0
   else
   if  (rollbacks < max_rollbacks)
        simulateNextEvent()
  /* else, we don't do anything, because the object suspends */
```

With the inclusion of this protocol, in every simulation cycle an object will simulate the lowest timestamp event (as Warped does originally) if the number of its rollbacks in the period T is smaller than the umbral *u*; if not, the object suspends executing until the new period of time *T*, after which Warped restarts the rollbacks number to zero.

In order for a Logical Process to be able to simulate objects that mustn't be delayed, we have modified the scheduler policy of Warped that choose the next object to simulate. It chooses the first object of the input event list (that is, the object with the lowest input event timestamp) only if its rollbacks count doesn't exceed the umbral *u*; else, the scheduler checks the next object of the input event list and so on, until it finds an object in condition to be simulated or until it reaches the end of the list.

IMPLEMENTATION OF THE PROTOCOL IN WARPED

The class Logical Process (file *LogicalProcess.hh*) has two new member variables: rfm_max and rfm_interval, for umbral *u* and period *T*. The function LogicalProcess::LogicalProcess (file *LogicalProcess.cc*) initializes these member variables, by calling the function readRFMConfig. This function reads the configuration file *local_rfm.config*, and exits if it doesn't have a correct format.

The class BasicTimeWarp (file *BasicTimeWarp.cc*) has a new member variable, called *rfmRollbacks*. It counts the number of rollbacks that a simulation object had in a run period *T*.

The class simulationObjectStats (file *TimeWarpObjectStats.hh*) has a new member variable, *suspendedCount*. It is used to store the total number of suspensions that every object suffered, and this number is showed at the end of the simulation for statistical purposes.

The functions *LTSFScheduler::runProcesses* (file *LTSFScheduler.cc)* and *LogicalProcess::simulate* (file *LogicalProcess.cc*) reflect the scheduler policy modifications.

In the file *config.hh* it's defined the name of the LRFM simulation configuration file:
#define LOCAL_RFM_CFG_FILE "local_rfm.config"

CONFIGURATION FILE FORMAT

To run a simulation with the Local Rollbacks Frequency Model, it must exist a file called local_rfm.config in the same directory of the simulation. The format of this file is:
max: x
rfm_interval: y

where *x* is the umbral of rollbacks, and *y* is the period of time, in seconds

Example:
max: 10
rfm_interval: 0.05

Local RFM Step

We noted in several cases of Local Frequency Model simulations that simulation times were high, because some objects spent more time suspended than the time it was taken to the simulation to run under TimeWarp protocol. The reason is that when an object reaches an umbral of rollbacks, and decides to suspend, it does it until the end of the actual check period finalizes, afterward the counters are reset.
Then, we decided to suspend an object just a defined number of N simulation cycles, after which the object resumes his simulation.
This way, we don't have the problem that if an object suspends at the beginning of the actual simulation period, it suspends almost all the period, but just a number of a defined N simulation cycles.

Global Rollbacks Frequency Model

In this protocol, each object uses global information. So each object decides to sleep if it have had the greater number of rollbacks during the actual check period. Then we define the functions M1 and M2 as the following:

Function M1: The potential error of an object is the number of rollbacks that the object had minus the maximum number of rollbacks of the other objects of the simulation, from a time *T1* until the actual time *T2, with T2-T1 <= T*, where *T* is the interval after which the local number of rollbacks restart (assigning a value of 0 to this variable).

Function M2: If the number of rollbacks for an object in a predefined interval *T* is greater than other number of rollbacks (that is, potential error > 0), the object suspends simulating, adopting a conservative behavior; however, the Logical Process where the object resides will continue accepting events from the neighborhood (although the events will not be processed until the object simulates again). If not, the object simulates aggressively, as it does with the TimeWarp protocol.

To implement this protocol, we needed to include in Warped a way to inform to each Logical Process about a period of time *T*. We also had to add to the Warped kernel a way so that each Logical Process may inform to the others LP about the number of rollbacks that its objects have had.

In Algorithm 2 it's showed the pseudo code of the protocol for each Logical Process and each Warped simulation cycle.

Algorithm 2 - Protocol Global Rollback Frequency Model in Warped

```
1. In each Logical Process, at the simulation start:
   Read the check period from the configuration file; store this value in the variable period
2. In each object, at the simulation start:
   previous_time  = 0
   max_rollbacks = 0
3. In each object, every time that the scheduler decides to invoke the Logical Process to run:
   actual_time = Warped.TotalSimulationlTime ()
   if (actual_time - previous_time >= period)
       simulateNextEvent()
       previous_time  = actual_time
       rollbacks = 0
   else
   if  (rollbacks < max_rollbacks)
        simulateNextEvent()
  /* else, we don't do anything, because the object suspends */
4. for i from 1 until the number of Logical Process of the simulation
       if (i is distinct from this PL id)
          send to LP i the number of rollbacks of the objects of the LP id
5. Subroutine that receives the number of rollbacks from other logical process:
   For j from 1 until the amount of numbers received
   If (rollbacks[j] > max_rollbacks)
   max_rollbacks = rollbacks[j]
```

IMPLEMENTATION OF THE PROTOCOL IN WARPED

The implementation of this protocol is similar to the implementation of the Local Rollback Frequency Model. However, the main difference is the inclusion of a new type of simulation message: RollbackCountMsg. Then, we implemented in the class CommMgrInterface (file *CommMgrInterface.cc*), the ability with which each Logical Process can inform to the others about its objects rollbacks.

In the file *config.hh* it's defined the name of the simulation run configuration:
#define GLOBAL_RFM_CFG_FILE "global_rfm.config"

CONFIGURATION FILE FORMAT

To run a simulation with the Global Rollbacks Frequency Model, it must exist a file called global_rfm.config in the same directory of the simulation. The format of this file is:
rfm_interval: y

where *y* is the period of time, in seconds

Example:
rfm_interval: 0.05

Compilation of warped simulations with the new protocols

Introduction

In this chapter we'll explain how to install the Warped simulation kernel in the Operating Systems Solaris (for Sun machines) and Linux (for any compatible PC 386).
First we detail the steps to follow in order to install the environment for each Operating System. Then, how to install Warped, with the same procedure for both platforms.
To understand the text from now on, it's required a minimum experience in Unix Operating System: file manipulation, programs compilation and user administration.

Solaris 2.7 Operating System

We explain here how to install the Warped simulation kernel in a Sun host, with Solaris 2.7 or above. First, it's necessary to install some software packets to prepare the environment to compile this kernel, which is written in C++ language.
Then, the mpich libraries must be installed. This libraries are an open source implementation of the Message Passing Interface (MPI) protocol. Finally, comes the Warped installation.

Important: Every instruction from now on must be executed as root user (system administrator), and in every host of the network that take part of the simulations.

ENVIRONMENT AND SOFTWARE PACKETS

The first tasks to do are the creation of a system user to run the simulations (we use the name **thesis** in this report), and installation of the software necessary to modify and compile the Warped code.

Get from http://www.sunfreeware.com the packets gzip-1.3, gcc-2.95.2-sol7-sparc-local.gz and make-3.78.1-sol7-sparc-local.gz for the Solaris (SPARC) version of the hosts to use.
If you want to use a more comfortable shell instead of sh or korn shell (that are installed in Solaris by default), get the file bash-2.03-sol7-sparc-local.gz.

Note: Don't use gcc version 2.8.1 (files gcc-2.8.1-sol7-sparc-local.gz and libstdc++-2.8.1.1-sol7-sparc-local.gz) because we've analyzed that this version doesn't compile correctly the Warped source code.

Do the following steps in the specified order:

- Creation of the user thesis and his home directory
  - useradd -g 10 -s /usr/local/bin/bash -c "User thesis" -d /home/thesis thesis
  - mkdir /home/thesis
  - chown thesis:staff /home/thesis

- Define a password for the user thesis
  passwd thesis (enter the password twice)

- Packet installation
  - pkgadd –d gzip, and install the packet from the interactive menu
  - gunzip gcc-2.95.2-sol7-sparc-local.gz, pkgadd –d gcc-2.95.2-sol7-sparc-local, and install the packet from the interactive menu
  - gunzip make-3.78.1-sol7-sparc-local.gz, pkgadd –d make-3.78.1-sol7-sparc-local, , and install the packet from the interactive menu
  - ln -s /usr/local/include/g++-3 /usr/local/include/g++
  - ln -s /usr/local/lib/libstdc++.so.2.10.0 /usr/lib

- Edit the file /etc/hosts, and add one name per line for every host of the network. Let's suppose that the hosts are hosts1 with IP 192.168.1.1, host2 with IP 192.168.1.2 and host3 with IP 192.168.1.3, the file will contain the following text:

```
127.0.0.1        localhost
192.168.1.1      host1
192.168.1.2      host2
192.168.1.3      host3
```

  Note: The spaces must be tabular

- Install the mpich software in the directory /usr/local. It's possible to download the soft from http://www.mcs.anl.gov/home/lusk/mpich/index.html.
  First, it's necessary to decompress the file mpich-1.2.1.tar.gz in a work directory (in our case we choose the directory home from the thesis user) and then compile and install the sources.

  cd ~tesis
  gunzip -c mpich-1.2.1.tar.gz | tar xvf -
  cd mpich-1.2.1
  ./configure --prefix=/usr/local/mpich -cc=gcc
  make
  make install
  cd ..; rm –rf mpich-1.2.1 (deletion of the temporal directory where the sources were compiled)

  Note: Don't use the version 1.2.0 of mpich software under Solaris Operating System, because it gives an error and stops during compilation.

- Create the file .rhosts in the home of the user thesis and include, one name per line, every host that will take part of the simulations (it's not necessary to include the name of the host where the file is created). For the example of the hosts host1, host2 and host3 the file would be:

```
host1
host2
host3
```

- Assign the correct permissions to the file .rhosts:

  chmod go-rwx ~tesis/.rhosts

- Enable remote shell in the host. For this check that the file /etc/inetd.conf contains the following line:

```
shell  stream  tcp     nowait  root    in.tcpd  in.rshd
```

  To probe that remote shell is working well, , execute the next command, after which it must appear in the screen the list of the files of the root dir of the remote host:

```
rsh remote_host ls /
```

- Installation of the utility rdist. Get it from http://www.magnicomp.com/rdist. Don't use the packet that comes by default with Solaris, because it doesn't work as we need. After compilation of the packet, it creates the files /usr/local/bin/rdist and /usr/local/bin/rdistd.

From now on, you have to choose a host as a "master". Every modification done to this host will be replicated on the other hosts of the network, through the rdist command. This way, you don't have to modify the code in every host of the simulation.

Linux Operating System

We detail how to install Warped simulation kernel in a Linux host. The distribution we've used is RedHat 6.2, but the procedure is the same for any other Linux distribution. The only requirement is that it's kernel must be 2.2.x or above, because 2.0.3x serie has problems in the TCP layer, so that it doesn't work well with the mpich libraries.

In the Linux installation it's required to select the following software packets: gcc C/C++ compiler and the utilities make, rdist and gzip. This way the host will have installed all the applications required in order to compile Warped simulations. If the Operating Systems is already installed, it's possible to install the mentioned software from the Linux CD-ROM, through the packet installer rpm.
Then it's required to install the mpich library – open source implementation of the MPICH protocol (Message Passing Interface) -, and finally Warped itself.

Important: Every instruction from now on must be executed as root user (system administrator), and in every host of the network that take part of the simulations.

ENVIRONMENT AND SOFTWARE PACKETS

The first tasks to do are the creation of a system user to run the simulations (we use the name **thesis** in this report), and installation of the software necessary to modify and compile the Warped code (if it wasn't done in the Linux installation).

Do the following steps in the specified order:

- Creation of the user thesis and his home directory
  - useradd -s /usr/local/bin/bash -c "User thesis" -d /home/thesis thesis
  - mkdir /home/thesis

- chown thesis:thesis /home/thesis

- Define a password for the user thesis
  passwd thesis (enter the password twice)

- Edit the file /etc/hosts, and add one name per line for every host of the network. Let's suppose that the hosts are hosts1 with IP 192.168.1.1, host2 with IP 192.168.1.2 and host3 with IP 192.168.1.3, the file will contain the following text:

```
127.0.0.1        localhost
192.168.1.1      host1
192.168.1.4      host2
192.168.1.5      host3
```

  Note: The spaces must be tabular

- Install the mpich software in the directory /usr/local. It's possible to download the soft from http://www.mcs.anl.gov/home/lusk/mpich/index.html.
  First, it's necessary to decompress the file mpich-1.2.1.tar.gz in a work directory (in our case we choose the directory home from the tesis user) and then compile and install the sources.

  cd ~tesis
  gunzip -c mpich-1.2.1.tar.gz | tar xvf -
  cd mpich-1.2.1
  ./configure --prefix=/usr/local/mpich
  make
  make install
  cd ..; rm –rf mpich-1.2.1 (deletion of the temporal directory where the sources were compiled).

- Create the file .rhosts in the home of the user thesis and include, one name per line, every host that will take part of the simulations (it's not necessary to include the name of the host where the file is created). For the example of the hosts host1, host2 and host3 the file would be:

```
host1
host2
host3
```

- Assign the correct permissions to the file .rhosts:

chmod go-rwx ~tesis/.rhosts

- Enable remote shell in the host. For this check that the file /etc/inetd.conf contains the following line:

```
shell  stream  tcp     nowait  root    in.tcpd  in.rshd
```

  To probe that remote shell is working well, , execute the next command, after which it must appear in the screen the list of the files of the root dir of the remote host:

```
rsh equipo_destino ls /
```

- In Linux the services defined in the file /etc/inetd.conf are invoked through the tcp_wrapper application. Thus, if we suppose that host1 is the “master” from where the simulations will be launched, it must be added in the file /etc/hosts.allow, of all other network hosts, the following line:
  ```
  in.rshd: host1
  ```

From now on, you have to choose a host as a “master”. Every modification done to this host will be replicated on the other hosts of the network, through the rdist command. This way, you don’t have to modify the code in every host of the simulation.

Warped

The procedure to install this simulation kernel is similar for the operating systems mentioned.. Assuming that all the environment is installed, it's not necessary to go on working as root user (system administrator). From here all the commands to execute must be entered as the user thesis.

Important: It's possible to get the Warped simulation kernel from http://www.ece.uc.edu/~paw/warped/ . However, the version used in this thesis is a modification, done to compile the source code with no problems. Even we used different modified Warped code for each Operating System, because the compilers used are distinct for both platforms, and the Makefile files (used to compile Warped with the *make* command) are different. Because of this it's recommended to use the Warped kernel provided with this thesis work.

We have used the latest Warped version today, 1.02. We open and compile this code in the home directory of the user thesis. The actions to do are:

- tar xvfz warped-v1.02.1.tar.gz
  cd warped
  ./configure
  make
  make install

INSTALLATION AND COMPILATION OF WARPED KERNEL MODIFICATIONS

For compilation of a simulation, it's enough to enter to the simulation directory and run the *make* command. For example, to compile the "ping-pong" simulation (located at *warped/TimeWarp/examples*) do the following:

cd warped/TimeWarp/ examples
make

Since Warped uses the directory tree *warped/TimeWarp/src* to access the kernel sources, it's enough to link the directory who contains the new simulation technique to the former, and then compile the simulation as usual.

For example, to compile the "ping-pong" simulation (located at *warped/TimeWarp/examples*) with the protocol Local Rollback Frequency Model, do the following:

cd warped/TimeWarp
mv src src_original
ln -s src_local_rfm src
cd examples
make

In order to compile any simulation with the protocols Local RFM Step or Global RFM, use the names

src_local_rfm_step and src_local_rfm respectively instead of src_local_rfm in the *ln* command.

Remember that in order to run simulations with these protocols, it's necessary to write a configuration file in the same directory of the simulation to run, as described above for each protocol.

Simulations execution

We explain here several aspects regarding compilation and execution of Warped simulations, the ones provided with the sources of Warped and the news developed in this thesis work.

PROCGROUP FILE

In order to execute a simulation simultaneously in several networked hosts, it's mandatory to indicate to the "master" host (that is, the host who launches the simulation) the name of the participating hosts. Thus, you must write a file called procgroup (this requirement is imposed by mpich software) in the same directory of the source code of the simulation, and it must contain the network hosts names, one name per line. If we consider the first example mentioned in the Warped documentation, the "ping-pong" simulation, and we suppose it must run on the hosts host1 (master), host2 and host3, the file will have the following text:

```
host1 0
host2 1 /home/thesis/warped/TimeWarp/examples/PING thesis
host3 1 /home/thesis/warped/TimeWarp/examples/PING thesis
```

The second field of this file indicates if the host is the launcher of the simulation, the third is the absolute path of the simulation in the remote hosts, and the fourth is the user name that require, via remote shell, the remote execution of the simulation.

RDIST

To test different simulation alternatives, it's usual to modify source code or configuration files of the simulation. This is the case, for example, when it's required to execute a simulation with different parameters, or with a different amount of hosts. The task of modifying the same file in several hosts is tedious, so we decided to use the *rdist* tool, which updates the contents of remote hosts files according to a central or master host. This way, each time that a file of the master host is modified, it's possible to update the same file in the other hosts of the network with the simple invocation of the rdist command.

It's possible to use the utility this way:

```
rdist -f distfile -l notify=nerror,ferror,warning -p /usr/sbin/rdistd
```

This command reads the configuration file *distfile*, notifies via e-mail if an error occurs, and knows that the rdist server of the other hosts is located at the absolute path /usr/sbin/rdistd.

The contents of the file distfile for our "ping-pong" example is:

```
FILES = (
     /home/thesis/warped/TimeWarp/examples/PING
     /home/thesis/warped/TimeWarp/examples/ping.config
     /home/thesis/warped/TimeWarp/examples/procgroup
)
```

```
HOST = (host2 host3)
USER = (tesis)

${FILES} -> ${USER}@${HOST}
     notify root@localhost;
```

This file says that the files `PING, ping.config` and `procgroup` of directory /home/thesis/warped/TimeWarp/examples/ from the hosts host2 and host3 must be updated (if the originals were modified), and in case of error notify it to *root@localhost*.

COMPILATION AND EXECUTION

In our "ping-pong" example, it's enough to compile and run the simulation to follow these steps:

- Create the file procgroup
- Create the file distfile
- rdist -f distfile -l notify=nerror,ferror,warning -p /usr/sbin/rdistd
- cd /home/thesis/warped//TimeWarp/ping-pong
- make
- ./PING

NOTIME SIMULATIONS

Here it's explained how to compile an existing simulation with the NoTime protocol (developed by the same team that developed the Warped tool).

In order to do it correctly, you must follow the following steps:

- Copy the directory that contains the simulation chosen (and that is located under directory /home/thesis/warped//TimeWarp) to the directory /home/thesis/warped//NoTime.

- Modify the file *Makefile* of the simulation, replacing each appearance of the string *libTW.a* with the string *libNoTime.a*, and erase all the references to the *RNDDIR* variable.

- In the Linux hosts, modify the file NoTime/src/Makefile, adding in the line 3:
  `CPPFLAGS += -I/usr/include/g++-2`

- Comment out all the appearances of the string *state->nameOfObject* in the files with extension *.c*.

- Modify the following two lines of the file main.cc:
  ```
  physicalCommInit(&argc , &argv);
  id = physicalCommGetId();
  ```
  with the lines:
  ```
  MPI_Init(&argc, &argv);
  ```

```
MPI_Comm_rank(MPI_COMM_WORLD, &id);
```

- To avoid the compilation error "ifstream undeclared", add at the beginning of the main.cc file the line:
  `#include <fstream.h>`

- Modify in the same file the path of the simulation configuration file.