Université de Lorraine
ESIAL - TELECOM Nancy, 2nd year internship report
Industrial supervisor : Martin Quinson
Academic supervisor : Do Van Phuc

Simulation of distributed application with usage of syscalls interception

Guillaume SERRIERE

Villers-lès-Nancy, from May 29th to August 17th 2012

Contents

Thanks

First of all, I want to thank Martin QUINSON for accepting me as a trainee. He allows me to discover the world of research and help me to improve my skills and my knowledge in computer science.

Then, I thank Marion GUTHMULLER for the time she gaves to me in order to explain his work in which my training is based on and for responses she gave to my questions.

I want also thank the AlGorille team for this welcome and this friendship that its member show to me during this 12 weeks I spent in the LORIA.

Finally, I thank the LORIA for accepting me in the laboratory and to help me whenever I needed.

1 Institutional Context

1.1 Laboratory

1.1.1 General presentation

LORIA is the French acronym for the "Lorraine Research Laboratory in Computer Science and its Applications" and is a research unit (UMR 7503), common to :

- CNRS (Centre Nationale de Recherche Scientifique),
- University of Lorraine
- INRIA (Institut National de Recherche en Informatique et Automatique).

This unit was officially created in 1997.

LORIA's missions mainly deal with fundamental and applied research in computer sciences. The lab is a member of the Charles Hermite Federation, which groups the four main research labs in Mathematics, in Information and Communication Sciences and in Control and Automation. Bolstered by the 500 people working in the lab, its scientific work is conducted in 27 teams including 16 common teams with INRIA. LORIA is today one of the biggest research labs in Lorraine.

LORIA was structured in five departments :

- algorithms: computation, image and geometry,
- formal methods,
- networks, systems and services
- knowledge and language management,
- complex systems and artificial intelligence.

The repartition per post of the 500 workers is :

- 168 full time researchers,
- 14 full time administrative support staff
- 130 PhD students,
- 105 post-doc,
- 50 engineers,
- 50 trainees.

1.1.2 Organization

The organization chart 1.1 shows the hierarchy structure.

Direction team

This instance gathers the director of the LORIA, the two Deputy Directors and scientists representing the different major themes of the lab. Its role is to assist the director in decisions and their implementations.

Laboratory council

The laboratory council is composed in four-year elected members and appointed members. It is the only advisory instance of the lab where the whole staff is represented. It has an advisory role to the director for all questions concerning the research unit: scientific policy, laboratory organization, budget, recruitment, trainings, internal rules, management of working hours, work conditions... The laboratory council meets at least once each quarter.

1.2 Research team

Presentation

During my training, I was hosted by the AlGorille [AlG] team under the supervision of Martin QUINSON.

It is composed of 4 permanent members, 3 PhD students, 2 associated members and 4 engineers. Jens GUSTEDT is the leader.

The team is involved in research on distributed system.Their approach emphasizes on algorithmic aspects of grid computing, in particular it addresses the problems of organizing the computation efficiently, be it on the side of a service provider, be it within the application program of a customer. Their methodology is based upon three points modeling, design and engineering of algorithms.

AlGorille aslo has the responsabilities of the Grid'5000 [Gri] Nancy's cluster. Grid'5000 is an experimental plateform, reparted in 10 locations whose objective is to experiment distributed system, and provides to researchers hardware to make large scale computation too.

Research activities

- Structuring of applications for scalability
- Transparent resource management
- Experimental validation

Simgrid

Simgrid[Sima] is a project leading by Martin QUINSON (LORIA, University of Lorraine), Arnaud LEGRAND (CNRS Grenoble), Henri CASANOVA (University of Hawaii at Monoa) and Frederic SUTER. The project was started in 2000.

The aim is to provide a tool box which can be use to make simulation with distributed system. Simgrid has different interface depending on the work we want

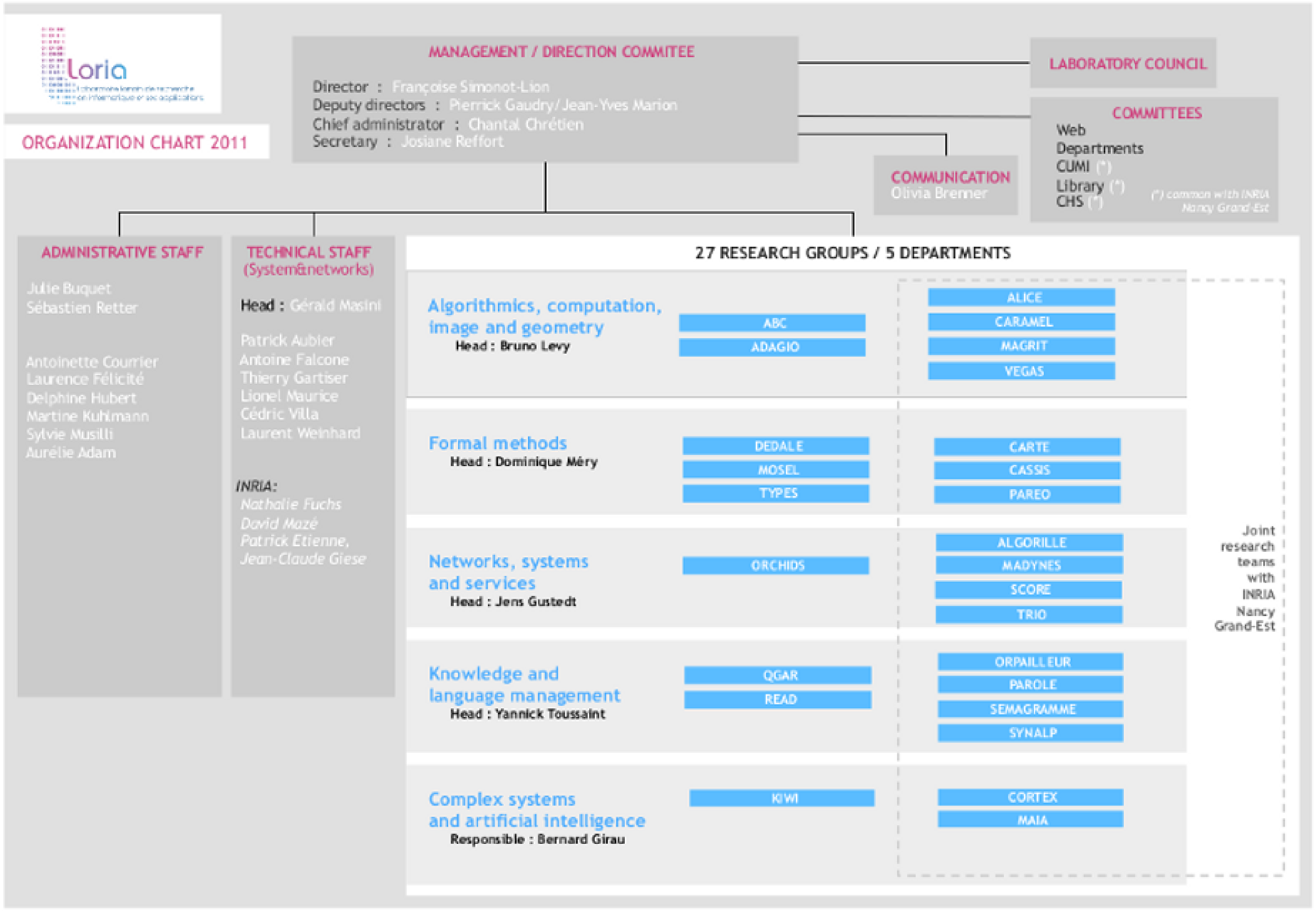


Figure 1.1: Organization chart

do. It is delivered under LGPL licence so its source code is available on the net. The framework allows reproductible simulation, with a lot of tool to easily describe the experimentation. Simgrid is a powerfull simulation framework and it is still improved by AlGorille team and the ones who are associated to.

2 Introduction

2.1 Context

Nowaday, distributed systems are more and more frequent. They can be found in various form from BitTorrent [Bit] to BOINC [BOI] project. They have a real advantage compared to classical systems.

First, they improve highly the amount of data available. All machines could have data that others need, and could need data from others. In system such BitTorrent or eMule [eMu], a client will contact a server, a machine which have already the data to take them. When they have it all, it become a server too.

Then we have the possibiliy to share computation capability of host. That case appeared in system like BOINC or render farm, which necessite huge and long computation. In such system, we exploit the computation power of others machines to improve our machine.

But these systems also raise a lot of problem because of their distributions. It is a really hard task to develop a good distributed application whose algorithms make a good usage of hardware capabilities. This is the result of the heterogeneous structure of a distributed system. In system, like BiTorrent, this problem is real. All machines which are on the net have different computations capabilities, different bandwith connections. There are the users who have to be take into account, because they may have different expectations from the one expected by developpers. For example, some of them will want to maximize their benefits despite consequences on the global system.

Those who develop such applications also encounter the problem of correctness and safety. In this software, race condition and deadlock where very hard to found and to fix, because of the lack of possibility to run them easily.

Actually there is three ways to run appplication to see their performance.

First of all, you can run them in real plateform. For this you can use dedicated plateform like Grid5000 to make a real deployment. But, this kind of experimentation can be hard to run because you need to set up the plateform. More, it's hard to simulate users interaction.

Then, you can simulate your application. This method consists in model your application to make it interact with an simulated environment and make him acts like real environment does. This way have been already employed quite often to study distributed application's behavior because it is possible to run thousands of node with many running configuration. More, it allows reproducibility of experiements. However, because you run model of your application, and not the real application, you cannot highlight implementation's bugs. More, it not helps you to face problem of real deployment.

Finally, you can emulate a part or all the environment of the application. This method tends to fix the problem of simulation as it allows to use real application and avoid the duplication of code need for simulation. Main difficulties of emulation is to catch interactions did by the application.

2.2 Problem

Our work consists in make run together simulation and emulation. And our main goal is to run real application, without access to this source code and give time estimation of duration. From the beginning, we fix as limitation for our work to use the real binary of each application, and make them run as they do in real condition without changing anything in the code. This work will be an extension of Simgrid's framework and will repose on it. We want an application which read in file the detail of deployment and of the plateform and perform the experience and give at the end the simulation time.

2.3 Methodology

As part of Simgrid project, we will obviously use this framework for the simulation part. More precisely, we will use SimDag API.

For the emulation part, we base our decision on the work made by Marion GUTHMULLER [simb], a Simgrid team member who developed an interception module. She looked for many option like valgrind or LD_PRELOAD, and utimately she found that ptrace (cf. C appendix) will be the best choice to reach our goal.

Ptrace is a linux syscall which allows a process to control another one. It is used in many way like retro engineering or on debuger like GDB because it provide a complete control of the process memory and processor's register.

Functionnement of ptrace is simple. When a condition is reach (breakpoints, next instruction, ...) the process is stop and then you can read process memory or write in. For Simterpose, we need to find out the communication beetween process, so we decide to use syscall as breakpoint for ptrace. Fortunately, this is one of the basic comportment of ptrace.

Syscalls (cf. B appendix) are composed of two parts in linux based OS. The first one, is when the application initiates the call. At this time, processor's registers are set with arguments of the function. And one of them contain the numero of the syscall whose application want execution. The second part is after the execution of the syscall. At thistime, the kernel return control to application with all register set with previous value, and result store in one register. Ptrace stop execution of process at each phase, so we can manipulate processor's registers to change arguments or even change the syscall which be executed by the kernel.

With ptrace, we can also read and write in process memory with PEEK_DATA and POKE_DATA argument. This method in necessary in some mediation, like when argument are only memory reference to larger data structures. It also be used to pick up data that application want to send and put in receiver memory.

Figure 2.1 shows the structure in which Simterpose operate. It is directly liked to the Simgrid module which is responsible of the simulation. Simterpose is interposed between the application and the kernel, and each communication between

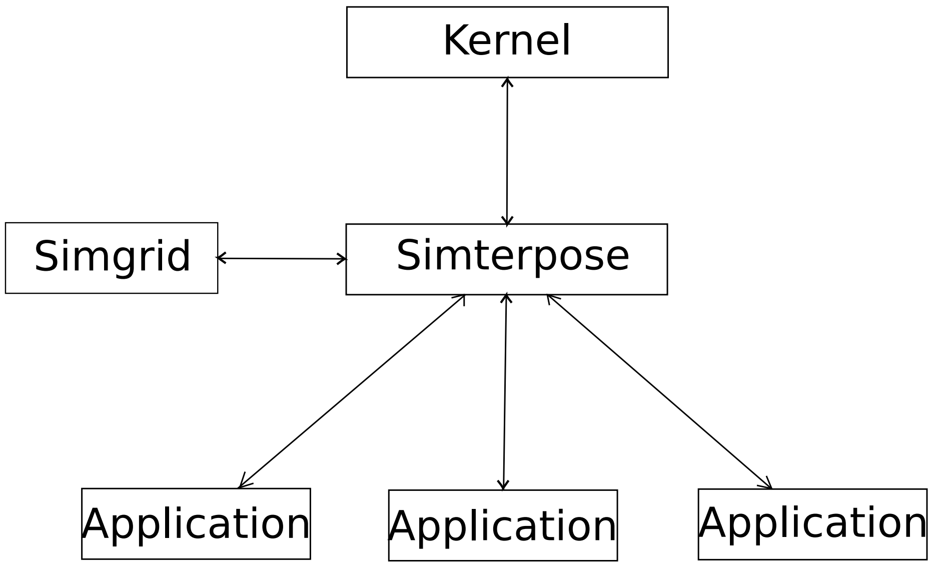


Figure 2.1: Structure of simterpose

them have to get throught Simterpose first. As each application use I/O syscall to deal with network and thefore to communicate, Simterpose will be naturally between each application, independantly of their behavior.

3 State of the art

This approach of emulation through interception is rather uncommon to study the application performance as it mandates both a working simulator and an interception frame- work. To the best of my knowledge, MicroGrid [Mic] is the only related project in the literature. Other interception frameworks serve other goals. Fuzzing frameworks such asSpike1 or Autodafé [Aut] try to discover bugs and security issues by intercepting the file and network operations and changing random bits in the program's input. Profiling and tracing tools (such as Tau [Tau]) allow to inspect the application's execution.

Our work repose on the interception module developped by Marion GUTHMULLER. It was designed in 2010 during a training. This module was a simpler implementation of software strace. This module can be used to pick out communication made by a process and them child. We start with this source code, and extend it to insert emulation and simuation part on his work.

4 Work

4.1 Interception and mediation of communication

Simterpose has for goal to simulate as well as possible interaction between process, so it's be really important to make a good interception and mediation of networking syscall (cf. B appendix), like connect or sendto. For achieve this goal, we have set up two model to make comparison between full mediation and only address and port translation. The difference is important because in one case, we use socket, and in other we only declare socket but do no syscall on them.

Full mediation In this mode, we don't let applications communicate with socket, or even to make connection. Concerning the network address, as we don't make any connection, we directly use simulated address.

Address translation Here, we let all communication go to the kernel. In this case, we don't have control of traffic data, we let the kernel rules it. We set up an address translation which relate simulated network address to local network address to allow process to communicate by local network while they use simulated network address.

4.1.1 Connection

To handle port usage on the different node, we keep a global table which retain the list of port in use on each node. These tables are used at every connection asking to find the owner of the listening port. They are also used when we have to reserved port for a socket on a bind call, or on an accept call too.

With address translation only

Our goal with simterpose is to simulate a real network on a local network. To do this, we have to make a translation between global network address used by simulated applications, and local network address that will be really used by applications. So we have to intercept all connect and accept syscall to produce an address and port translation.

First of all syscalls involved in connection is bind. So we have to translate binding port and associate the local and the simluated port. When a bind is request by an application, first we check if the port is free.

If not, we put on result register the error return for corresponding to a non free port, and we let the application run.

If binding could be do on the requested port on the node, we have to find a free port on local network. For this, we make successive bind, changing the port on each try until we found a free one. When we have this port, we put him on translation table.

Finally this we restore all previous argument to make believe to applications that it really bind on a giving port in a dedicated node.

The second part of a connection protocol is the couple connect/accept. The real problem of them is their blocking caracter. If we let the call go to the kernel when it happen, we could face a deadlock because we won't realise the connection. Indeed, the application which will realize the connection is block and will not run again until the application which waiting for the connection receive it. So we have to let an application which process an accept or a connect syscalls blocked, and make them restart only when the connection can really be done (when there is an application which make an accept on a socket and in the same time, there is an application that make a connect on this socket).

After this, the mediation is really easy to make beacause the only thing we do is to replace the address requested by another one. To achieve this, we look the translation table to find the local network port for address and socket asking on the syscall. Then we write on processus memory to make it use the real address when the syscall is going to the kernel, and when the connection is done we write the simulated state of communication (the port and address used in simulated work for connection).

With full mediation

In this case, we don't let connection be done. We neutralize bind, listen, connect and accept syscall to prevent usage of kernel in connection.

Mediation of bind syscalls consists only on watch in port table if the port is free for binding. If not, we put on result's register the error return for corresponding to a non free port else, we put 0. Finally, we let the application run again.

Like in non full mediation, we make the application wait for the same conditions connection (one application on accept and one on connect).

When that occured, we first attribute a port to the socket creating by the accept call. As we don't let the connect syscalls be run, we have to reserve a file descriptor for the application to ensure the consistency of the application's model. To achieve this goal, we make the application asking for a socket creation using manipulation of registers. When the socket was created, we used the result of this syscalls as result of connect syscalls. After this, we write in the application memory the result of the connection, like in the other mode.

4.1.2 Sending and receiving syscalls

With address translation only

In this case, we let sending and receiving syscalls be processed by the kernel.

All receiving calls could be blocking if there is no data to read on the socket, so we have to be sure that the application receives something on them before let the syscall goes to the kernel. To ensure this, we set up a mecanism which stores

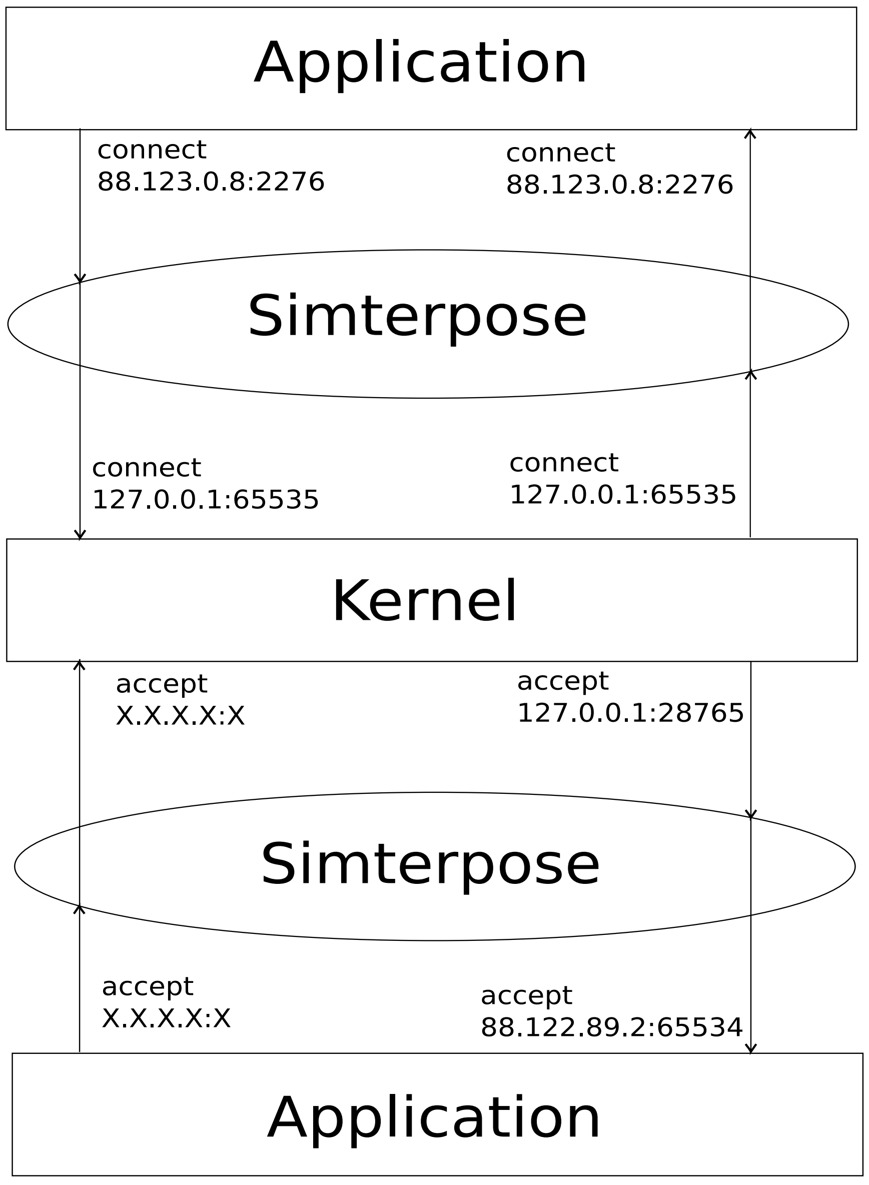


Figure 4.1: Example of a connect/accept session

all data receive on a socket. On the other hand, we have considered send as non blocking syscalls.

When receiving or sending become non blocking operation, we let the syscalls gone to the kernel and we use the value returned for the simulation.

Like we don't control the amount of data send or receive (it's the kernel which decide), we have to remember how many data, store by send, are waiting for reception. So, at each receiving, we check if we receive a new send to simulate the task. If we don't receive something new, we don't block the simulation because the sending of data to the node had already be take in account. Else, we block the application until the simulator notify the end of the task.

When this is a sending, we use the value returned by the kernel, and inject it into the simulator after store for receiving mecasime. We block the application until the end of the task.

With full mediation

In this case we don't let applications sending data through the network because, we don't let application make conection. So we have to use PEEK_DATA and POKE_DATA argument to manipulate memory and allow simterpose runs communications.

Like in the other mode, we first check for the presence of data which waiting on the socket.

In case of sending, we start with the neutralization of the syscall. When it's done, we read the data sending by the application directly in its memory and we store them for the receiver. Then, we use the amount of data requested for the sending and inject it on the simulator. Finally, when Simgrid notify the end of the task, we put the amount of size of send in result's register and let the application start again.

When we face a reception, we neutralize the syscalls. Then, write directly in the memory of the application the data send on this socket which are store in mecanism. For the amount of data, we use the minimum between the amount requested, and the amount stored for the result of the syscall. When the writing is done, we check if we received new data or not like on the other mode. If we receive new ones, we stop the application until the end of the task on simulation. When it's done, we let the application run again. Else, we let the application run.

4.2 Interception of computation time

To make emulation more complete, it is important to take in account computation time. Simgrid framework allows us to simulate pure computation task. So we juste have to found computation time and inject it into the simulator. To find this, we use TASKSTAT [tas] interface to ask kernel. This interface provides a lot of function and one ofthem are to ask to system, real, user and system time. For this usage, we only sum user and system time because real time doesn't be revelant at all.

4.3 Interception of time syscall

The syscall time is used to ask to the kernel the current time in second. In Simterpose, we want to make run application at rythm of simulted clock. So, we intercept this call andrespond to applications with the simulated clock. The origin of time are not zero but the time system at the beginning of the simulation. This allows application to have coherent time regardless of real time took by simulation.

5 Experimentation

5.1 Qualitative

The next list represents the amount of syscall which are mediate by simterpose : listen, close, shutdown, bind, accept, connect, send, recv, sendto, recvfrom, sendmsg, recvmsg, socket, open[1], creat[1], time, getpeername, dup[1], dup2[1], fcntl, getsockopt, setsockopt.
Syscalls which deal with file descriptor are mediate only if they imply PF_INET socket.

Simterpose is also able to support more than the C langage. He can also be used with Pyhton or C++.
In fact, it will work with all langages which finally used linux syscalls interface to communicate through the network, or to handle time function.

5.2 Quantitative

For all experiences, we use the shell command time to measure all datas.
The next array contain result of comparison between address only and full mediation on data exchange only.

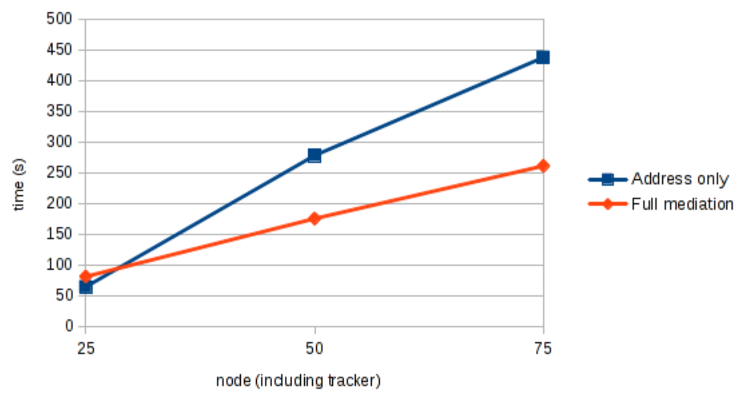
	Address Only	**Full Mediation**
40 bytes	1,71	1,86
10000 bytes	1,885	26,42
100000 bytes	2,09	247,24

Figure 5.1: Comparison between full mediation and address translation only for 10000 sends

This proves that using kernel when we do heavy data exchange between application is more efficient. Indeed, reading memory with ptrace necessites to use two syscalls for every byte you send.

However, we have to take in account that isn't the same case with real application. With real application, you don't make only exchange, you also make socket operation, you make more than one connection, and because you make little data exchange, full mediation mode can become more efficient.
To prove this, we will use a BitTorrent deployment. We will use bitornado for the tracker, and ctorrent as client. We run the experiment on a homogeneous

[1]Mediation is made to ensure the coherence of file descriptor table of each processus



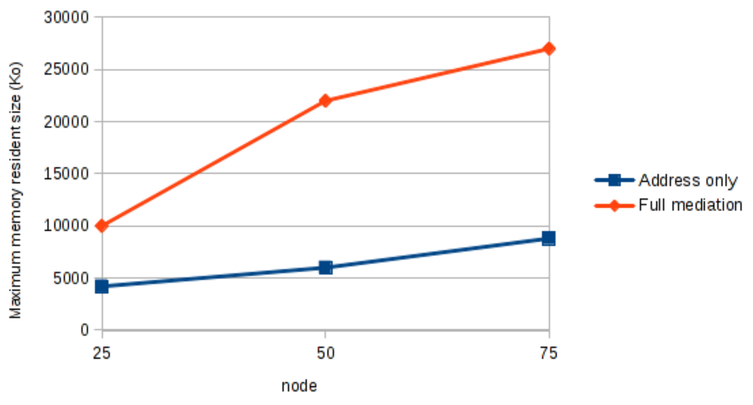


Figure 5.2: Experimentation results for BitTorrent

platforme in each node are linked with a 1Mo bandwith connection. All node have a computation capability of 10 MFlops.

For the scenario, we will start at 0 seconds with the launchment of the tracker. At 5 seconds, we will launch the only client which has the data. After this, every 3 seconds, we launch an other client which doesn't have data. We use a 16Mo file for experimentation. We run simterpose during 1000 seconds of simulation, and after we stop.

As we can see, there's no immediate choose between different mode. Is this experiment, the full mediation mode is more efficient for time issue, but in the same time the other mode has less print in memory.

As full mediation mode seems to be the more efficent for time issues, we will make more tests to see evolution of memory consumption and of time. The chart 5.3 show the time and the memory used by the simulation for the full mediation mode.

Experimentations show the linearity of the simulator. Obviously, if you simulate more application, it will take a longer time. However, we can see that simulated time runs faster than real time.

Towards memory, the memory print of Simterpose obviously grown with the amount of process.

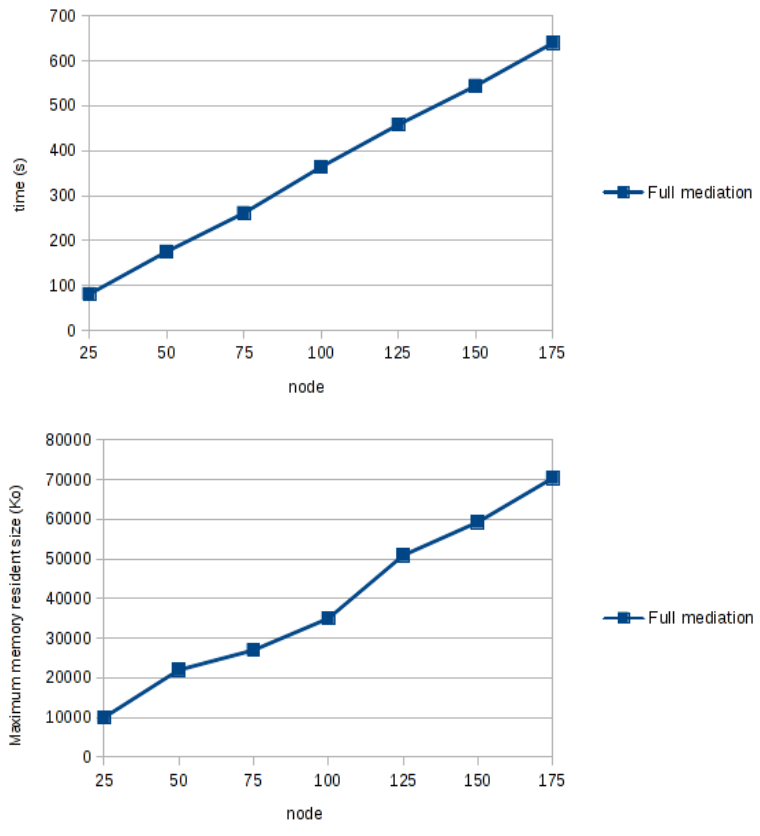


Figure 5.3: Experimentation results for full mediation

6 Conclusion

As we can see, Simterpose provides tools in order to use in the same time emulation and simulation. With simterpose, we can run the real application on any host and simulate a deployment on a real platform.

The network mediation, with full or only address translation represents to method which can be use. The experiment reveal that there is no real method better than the other in all situation. We have to choose the one which best fill the task you need, regarding you're limitation.

Even if full mediation mode is more efficient for time, it has the disadvantage of being less realistic than the address translation mode. This is due to the necessity to not let a syscall operate directly on a socket. But in this call, you also have option on buffer size, or several way to manipulate control message. All of them can be take on account on simulation for now.

This mode also lead to a worse memoory print.

However, the address translation only mode is limited by the amount of port on a system, limit that doesn't bother the other mode. Fortunately, there are method to increase the amount of port in a syste, like with the usage of cgroups [cgr].

Experimentation also show that this mode can be less faster than full mediation.

Simterpose is already able to run several application with real mediation and is able to give some results too. However, there are many way to improve it. Firstly, it would be easy to insert thread support because we use tid and not pid on Simterpose. This imply only a few modifications in the structure. It is the same for program which make fork.

We can also upgrade the amount of syscalls handle by Simterpose. Simgrid will integrate soon I/O simulation on hard disk. This will improve the quality of simulation by take in account the duration of harddisk I/O operation.

In other way, it would be interesting to combine kernel level interception and library level interception to improve the quality of the simulation. Some functions like time's functions or thread posix interface are using mecanisms which not use syscalls or that use non obvious methods. This will be possible with the use of LD_PRELOAD in the launcher.

We also can make Simterpose be itself distributed to improve the amount of applications which can be run in the same time. This improvement would reduce the problem of memory lack and could open the way of parallelization of the simulation. Our best speed limitation is the fact that we don't run more than one application in the same time. Parallelized the interception would lead to a significant increase of speed of the simulation.

7 Personnal conclusion

This training was a great experience for me.

First of all, it allows me to discovered the world of research and conforts me in my personal project to go on a laboratory. I really like the mixing of people with different outlooks, and different background. I also like the global ambience in offices.

Then, I really was interested in the project that Mr. QUINSON gave to me. At the very first look, it seemed to be a very hard task, but little by little, I arrive to make Simterpose run. Although I didn't complete all the points set in the first time of my stage, I was very sastified to be able to make the reach reach his actual state.

A How to use

Simterpose use simgrid interface so it use deployment and plateform description file of simgrid framework.

For plateform file, we strictly use the same file as when you run any simulations with this. This file must contains the computation power of host, description of each link on the connection, and the routage of the plateform. Each node could have an attribute which contains an address ipV4 in dot notation, this ip will be attribuate to the node on the simulation. If ip is not given, we take a random ip and attribuate this to the node. However, this ip doesn't fit a network logic.

Deployment file describes the repartition of application on the plateform. For each process, the name will be the name of the node on which the application will run, the field function will contain the path to the binary of the application. For Simterpose, all process must contain the start time attribute that will be used to launch the application at the good time on simulation.

Option

Simterpose is provide with the possibility to provide the computation capability in flops of the host which will make run the application. This allows Simterpose to skeep the benchmark that it will make before the beginning of the simultion.

B Description of a syscall on a 64 bits system

A syscall, or system call, is a function which are part of the linux kernel interface. These functions provide an abstraction to deal with the hardware. The most used syscalls are I/O syscalls which allow to deal with disk or with network.

Syscalls are on reality a software interruption, called a "trap", the interruption 0x80. All syscalls follow the same outline, and this only the numero contained on a register which indicates the kind of syscalls to execute. Here, we will take the example of the x86_64 architecture. The gestion of syscalls greatly differs from 32 bits to 64 bits mainly for networking syscalls, so we don't go to speak of this.

Register's content

When the application requires the 0x80 interruption, all argument have to been set on the different register. We can get the register by using ptrace syscall with PTRACE_GETREGS in first argument. The following list explains the content of each processor's registers (we take the name of field of the structure return by ptrace) :

Register	Content
orig_rax	numero of syscall
rax	value returned by the kernel
rdi	first argument
rsi	second argument
rdx	third argument
r10	fourth argument
r8	fifth argument
r9	sixth argument

Figure B.1: Content of processor's register when a syscall is done

If we modifiy the content of the orig_rax register, we modifiy the kind of syscall executed by the kernel. This could be usefull if you want to neutralized a syscall to avoid any kernel action.

Other register would contain data or pointer depending on the sycalls executed.

Progess of a syscall

There is three part in a syscall.

First, the application call 0x80 interruption. At this time, processor's register are set with the argument and the numero of the syscall. When the interruption is done, kernel take control of the execution thread, and the application wait for the end of the execution.

The second part is the execution of the syscall by the kernel. This section could take a long time depending of the system call. Syscalls like time (return the current time to the application) which don't necessite calculation of special conditions will return immediatly and other like accept, can take more time and could block application for seconds.

The third and last part is the moment when the application regains control of the execution, at the end of the syscall. At this time, rax register contains the result of the syscalls and other register contains same values as values set before the trap or other results of syscall when the returned value isn't enough.

C Ptrace

Ptrace is a linux syscall used to control other processes and manipulate their memories. The most popular usages are debuger (like gdb) and tracing tool (like strace). For more details see the Wikipedia article `http://en.wikipedia.org/wiki/Ptrace`.

Usage with syscalls as break points

For the rest of the explanation, we will use the term debuger to name process that control and debugee for the application controled.

Tracing of an application with usage of the syscall as breakpoints follow this schema :

- Attach the debuger to the debugee
- Wait for breakpoints
- Manipulate debugee
- Resume debugee
- Wait for breakpoints
- ...
- Detach process

Attach the debuger to the debugee

Attach the debuger means that we ask the kernel to have the right to control the other process. To make this, we run ptrace syscalls in twice process :

- Debuger : in this case we give PTRACE_SETOPTIONS in first argument, the pid of child in second, NULL on the third, and for the fourth, we can use various option To ask for tracing every child of the debugee for example.
- Debugee : in this case we give PTRACE_TRACEME in first argument, 0 for the second, and NULL for the others two.

There is no need to worry about the order of execution. After this sequence, the debugee will be under control of the debuger and will stop at each signal, or syscalls.

Wait for breakpoints

When a a debugee is traced by another program using ptrace, it will stop at every signal or syscall it will encounter. Each stop provokes the send of a signal to the debuger, a SIGCHLD signal. So, we can deal with this signal to found out breakpoints. But it is possible to use wait or waitpid syscalls too. In this case, the status returned by the call may contains a return value and a cause in case of ending, or could contains informations about forking.

Manipulate debugee

All of the following ptrace's calls must be performed when the debugee is stopped in order to avoid error.

- Register's manipualtion : with ptrace, you can obtain the processor's register and also give to them other values. For collect them, you have to perform a ptrace call with PTRACE_GETREGS as first argument, the tid of debugee in second argument, NULL, and to finish, a pointer to an user_regs_struct's data in which the data will be fit. For set the register, replace PTRACE_GETREGS by PTRACE_SETREGS, and values contained in the data points by the fourth argument will be set in registers.
- Memory writing : you can only write word by word on memory. The size of a word depend of your architecture (64 bits for example). To perform this, you must use PTRACE_POKEDATA as first argument, tid of the debugee for the second, the memory address in debugee memory which is the destination of the data and a pointer to the word you resquest for writingfor fourth argument. Even with ptrace, you must be aware of segmentation failure. Fortunately, instead of cause the crash of the application, doing such error only provoke negative return of ptrace call with errno set with EIO error value.
- Memory Reading : like with writing data , you can only read in debugee word by word. If you want read memory, you have to set the first with PTRACE_PEEKDATA, the second with the debugee tid, the address of the world you want read, and NULL. When you make this kind of call, the data read is the value returned by ptrace, this imply to make a double check using errno value to differentiate the error and the data when -1 is returned. Like with writing you should be aware of memory address.

There no need to worry about the reading data which are less longer than a word because linux system used to split memory into world size and store data in them box, even they don't fill the places. So you will never make segmentation fault until you give a vald address.
Here, we can notice the existence of PEEK_TEXT and POKE_TEXT. We don't speak of them because they are exactly the same behavior on a linux system.

Resume debugee

To resume a system and let it continues is execution, you have to perform a ptrace call with PTRACE_SYSCALL on first argument, the tid of the debugee as second, and NULL for the two others.

Detach process

When you want to stop tracing a process, you have to detach it. After this operation, it will run normally without interruption from ptrace. To achive this, you have to run a ptrace call with PTRACE_DETACH as first argument, tid of debugee next, and NULL for the others.

Bibliography

[AlG] AlGorille: Algorithmes pour la grille. http://www.loria.fr/equipes/algorille.

[Aut] Autodafé. Martin Vuagnoux. Autodafé : an act of software torture. In 22nd Chaos Communications Congress, Berlin, 2005.

[Bit] BitTorrent, a peer-to-peer file sharing protocol. http://en.wikipedia.org/wiki/BitTorrent.

[BOI] BOINC, computation for science. http://boinc.berkeley.edu/.

[cgr] cgroups. http://www.kernel.org/doc/Documentation/cgroups/cgroups.txt.

[eMu] eMule, free peer-to-peer file sharing application. http://en.wikipedia.org/wiki/EMule.

[Gri] Grid5000, large scale infrastructure. https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home.

[Mic] Microgrid. H. Xia, H. Dail, H. Casanova, and A. Chien. The MicroGrid: Using Emulation to Predict Application Performance in Diverse Grid Network Environments. In Workshop on Challenges of Large Applications in Distributed Environments, Honolulu, June 2004.

[Sima] Simgrid. http://simgrid.gforge.inria.fr/.

[simb] Simterpose. Marion Guthmuller, Lucas Nussbaum, and Martin Quinson. emulation d'applications distribuées sur des plates-formes virtuelles simulées. In Rencontres francophones du Parallélisme (RenPar'20), Saint Malo, France, May 2011.

[tas] Taskstat. http://www.kernel.org/doc/Documentation/accounting/taskstats.txt.

[Tau] Tau. S. Shende and A. D. Malony. The tau parallel performance system. International Journal of High Performance Computing Applications, 20(2):287–331, 2006.