Last modif: 09/05/13

HOWTO: RBFT

NOTE: it is assumed that the environment where the code is compiled, linked and executed is a 64 bit environment, typically Ubuntu 10.10

Requirements

-Linux kernel >= 2.6.32 and its souce (to compile a kernel module)
-Ubuntu/Debian
-known to work with g++ 4.4.5
-the code cannot compile natively on 64bits systems. Install the ia32-libs (on Debian) for cross-compile
-1 machine for each RBFT node + c machines for the clients + 1 manager machine from which you launch your experiments (this last machine can be one of the clients machines)
-ssh access to all the machines without password and with the same username
-sar (sysstat package) to monitor network and cpu usage
-python (v2.x) to be able to run the scripts
-perl 5 for a few scripts
-rsync to update changes from the manager node to all the other ones

Setup the environment on each node (clients and replicas)

1. Download the code and put it in a folder called `RBFT`
2. In the `RBFT` root directory, you can find in particular the following four folders:
    - `lybbiz.pir` -> the Protocol Instance Replica (PIR) code base
    - `lybbiz.verifier` -> the Verifier code base + code for the client
    - `lybbiz.common` -> files that are common to the PIR and the Verifier
    - `bft-simple` -> the executables for manager, client, PIRs and verifier
    - `sfslite` -> code of sfs-lite, used for cryptographic operations
3. The first time, you need to compile `sfslite` by doing:
    - `./install.sh` (Note: this script exports flags for a compilation in 32bits on a 64bits machine)
    - If the code does not compile (`error: template with C linkage` errors) then you need to modify the file *`sfslite/crypt/bigint.h`* as follows:

Previously	**Current**

#if defined (HAVE_GMP_CXX_OPS) || !defined (__cplusplus)
#include <gmp.h>
#else /* !HAVE_GMP_CXX_OPS */
/* Some older C++ header files fail to include some declarations
 * inside an extern "C". */
extern "C" {
#include <gmp.h>
}
#endif /* !HAVE_GMP_CXX_OPS */
	#include <gmp.h>

4. Go to `bft-simple` directory and compile RBFT launching: `./compile.sh.` It will produce the following binaries:
    - `bft_client`
    - `bft_pir` (Protocol Instance Replicas)
    - `bft_verifier`
    - `bft_manager` (Protocol Instance Replicas)
    - `bft_latency_extractor` (Not used anymore, or only by a few scripts
5. Edit the script bft-simple/util.sh with your environment settings:
    - BASE_DIR: the directory where you have placed your code
    - NB_FAULTS: value of **f**, i.e. max number of simultaneous tolerated faults. Recall that the number of nodes **n** must be equal to 3***f**+1.

Local development with Eclipse CDT

- Checkout the code as an Eclipse C/C++ project
- The project has various build configurations, you can switch between the build configurations on the horizonatal menu, click on the *hammer* icon, and switch build configuration

Launching

Connecting to the manager node (i.e., the node from which you will launch the experiments)

ssh **manager**

From the manager node

1. Go to `bft-simple` directory.
2. (If changing machines) Edit the file `machines.sh` with the list of machines you want to use. The format of the lines is the following one:

CLIENTS:	`hostname:ip_addr:nb_clients`
VERIFIERS:	`hostname:ip_addr`
PIR0:	`hostname:ip_addr`
PIR1:	`hostname:ip_addr`

- PIR2:   `hostname:ip_addr`
- MANAGER (only one):   `hostname:ip_addr`

NOTE:

a. the clients connect to the the verifiers (“`VERIFIERS`” entry)
b. the verifiers talk with the PIRs on the same machine via the `kzimp` kernel module (nothing to specify about this on the file)
c. on each individual machine, the PIRx and verifier IP addresses should placed on 4 different NICs for robustness (but not mandatory to test the code; 4 is for the case **f=1**), e.g on sci74 it could be:
   i. 199.194.**21**.74   -> verifier
   ii. 199.194.**22**.74   -> PIR0
   iii. 199.194.**23**.74   -> PIR1
   iv. 199.194.**24**.74   -> PIR2
d. the order of the machines listed for each PIR0, PIR1, PIR2 should differ in order to specify different primary nodes for each PIR instance

3. (If modified machines.sh) Create the configuration file. Its name will be *config*:
```
./create_config.sh
```
4. (If modified machines.sh) Deploy the configuration file to all the nodes:
```
./deploy_config_on_all_nodes.sh
```
   - *NOTE: the manager node must be able to ssh on all nodes without being asked for the ssh password*

NOTE: Each node (PIRs + Verifier + Verifier_thread) is associated to a specific core. The association is defined in launch_verifier_pirs.sh and is for now terminators-specific.

NOTE: The Forwarder thread uses a dedicated NIC for its communication. The IP address is defined in libbyz.verifier/parameters.h: NETWORK_INTERFACE_IP_FOR_FORWARDER

The manager is a special program which synchronizes the clients during an experiment. In particular, it sets the maximal throughput at which clients send their requests and the size of the requests and replies.

There is a lot of .sh scripts in bft-simple. Each of them is used to launch a specific experiment. You can execute such a script without any argument to see how it is used.

Here are some examples:

- **Fault-free execution:**
  - `./launch_xp_fault_free.sh <CLIENTS> <request_size>`
    - `CLIENTS` is a file containing, one per line, host name of a client machine and the number of clients on the machine
- **Client flooding:**
  - Important parameters in libbyz.verifier/parameters.h
    - FAULTY_FOR_MASTER or FAULTY_FOR_ALL: you need to define one of them. They define to which node the faulty clients send their (faulty) requests.

        - CLIENT_BLACKLISTER_SLIDING_WINDOW_SIZE: size of the sliding window of every client in the blacklister. It defines how many requests are considered to blacklist or not a client that sends bad requests (mac invalid)
        - READING_FROM_BLACKLISTED_CLIENTS_PERIOD: period at which the select (in Verifier_thread) considers the blacklisted clients.
    - `./launch_xp_fault_free.sh <CLIENTS> <request_size> <percentage_of_faulty_clients>`
        - `CLIENTS` is a file containing, one per line, host name of a client machine and the number of clients on the machine
    - Note that the clients are either correct or faulty, i.e. they do not send each a proportion g of bad requests. The proportion is global, considering all the clients

A word about kZIMP

kZIMP (present in the kzimp directory) is a kernel module which provides an efficient and robust communication mechanism for multicore machines. Technical report: http://membres-liglab.imag.fr/aublin/zimp/reicom_TR.pdf
This module is used to multicast messages from the Verifier to its PIRs.
It creats files in /dev/kzimp that can be used to send and receive messages using the traditionnal read(2) and write(2) syscalls.
It exports the file /proc/kzimp which can be read and written to get/set its settings.

About the authors

RBFT has been written by Pierre-Louis Aublin and Sonia Ben Mokhtar, based on the code of PBFT by Miguel Castro.
kZIMP has been written by Pierre-Louis Aublin.
This work has been done during P.L. Aublin Ph.D. thesis and has lead to a publication:

- **RBFT: Redundant Byzantine Fault Tolerance**, Proceedings of the 33rd International Conference on Distributed Computing Systems, July 2013.