A guide to TakTuk use

Guillaume Huard

Version 5, December, 2008

Contents

Chapter 1

Introduction

TakTuk is an inuit word meaning "fog". I don't know what the original creator of this tool, Cyrille Martin, had in mind when choosing this name. Maybe it was just poetic inspiration... Nevertheless, we can find afterwards some similarities between fog and TakTuk: both spread everywhere fast and seamlessly.

The two first TakTuk versions have been written in C++ by Cyrille Martin during his PhD thesis. The third one has been completely rewritten in Perl by Guillaume Huard. TakTuk 1 and 2 are now obsolete as they are not maintained anymore. Although Cyrille Martin did not take part in the development of the third version, he proposed in his PhD thesis the concept of remote execution deployment and the overall architecture of a deployment engine which are still used in TakTuk. Among other contributor to TakTuk, the most important ones are: Jacques Briat, Thierry Gautier, Benoit Claudel, Johann Bourcier and Olivier Richard. TakTuk is available at `http://taktuk.gforge.inria.fr`.

1.1 What is TakTuk ?

TakTuk is a tool for deploying parallel remote executions of commands to a potentially large set of remote nodes. It spreads itself using an adaptive algorithm and sets up an interconnection network to transport commands and perform I/Os multiplexing/demultiplexing. The TakTuk mechanics dynamically adapt to environment (machine performance and current load, network contention) by using a reactive work-stealing algorithm that mix local parallelization and work distribution.

- Characteristics
  - adaptivity: efficient work distribution is achieved even on heterogeneous platforms thanks to an adaptive work-stealing algorithm
  - scalability: TakTuk has been tested to perform large size deployments (hundreds of nodes), either on SMPs, regular clusters or clusters of SMPs
  - portability: TakTuk is architecture independent (tested on x86, PPC, IA-64) and distinct instances can communicate whatever the machine they're running on
  - configurability: mechanics are configurable (deployment window size, timeouts, ...) and TakTuk outputs can be suppressed/formatted using I/O templates
- Outstanding features
  - autopropagation: the engine can spread its own code to remote nodes in order to deploy itself
  - communication layer: nodes successfully deployed are numbered and programs executed by TakTuk can send multicast communication to other nodes using this logical number
  - informations redirection: I/O and commands status are multiplexed from/to the root node

1.2 Do I need it ?

When executed, the TakTuk engine establishes a logical interconnection network between remote hosts. Then, it executes commands given on the command line (batch mode) or read by a simple command interpreter (interactive mode). TakTuk is able to:

- execute the same command on all/some of the deployed node
- send inputs to all/some of the commands in execution
- gather the output of all executed commands to the root host
- allow commands to communicate using the logical interconnection network

1.2.1 Parallel machines administration

TakTuk is a tool especially suited to the administration of parallel machines because it eases the handling of groups of hosts. It might be used in batch mode for simple machine state tests (e.g. test hosts responsiveness simply by letting the engine setting the network up using its default connector - ssh) or in interactive mode for deep investigation on several hosts, using the TakTuk command interpreter to execute multiple commands on multiple hosts (standard test sequence on a group of hosts, ping pong test between several machines, ...).

1.2.2 Parallel applications development

During the development and the exploitation of a parallel code TakTuk eases the work of the programmer. It constitutes a fast and scalable deployment solution (which enables quick tests of an application in development: no need to wait ten minutes to experiment the result of a code modification). It also provides a portable control network (which eases the initial setup of a parallel application by providing to processes a way to communicate - it can even cross firewalls as long as a gateway can be accessed).

Chapter 2

Deploying remote executions

The main task of TakTuk is to deploy efficiently a potentially large set of remote commands execution on a target machine (usually a parallel one). This chapter covers the use of TakTuk to perform this deployment process, it also presents the main characteristics of the tool that should be usefull to all its users.

2.1 First steps with TakTuk

TakTuk is a very versatile tool and its advanced use can be tricky and require complex command lines. At the time of this writing, TakTuk understands 26 different options and 23 commands (with some combination possibilities). Nevertheless, basic TakTuk use and simple execution of common parallel tasks is straightforward.

2.1.1 My first connection, autopropagation

In essence, TakTuk works a little like the `ssh` command: it connects to some remote peer and allows the user to execute commands on this remote peer. The main differences are that TakTuk can connect to several peers at once, does not execute by default a user shell on them and provides mechanisms to redirect commands I/O as well as to allow interprocess communication. The strength of TakTuk is its efficient management of large sets of remote peers. It can seemlessly manage thousands of connections from one single initiating machine.

To connect to a peer with TakTuk there are several requirements:

- the peer must be configured to allow a single connection without asking for any password. By default TakTuk uses `ssh` for individual connections: in this case, you must have generated and installed an empty password rsa/dsa keypair to allow the connection without password prompt.
- TakTuk has to execute its own engine on the remote peer, thus: the remote peer must have Perl installed. It is not necessary to install TakTuk on the remote peer, TakTuk knows how to propagate itself using the connection, just tell it to do so with the `-s` option.
- you have to give to TakTuk both the remote peer name (or address) and commands that tell it what to do (typically execute something on the connected peer).

As a first example, the following command line executes the command `hostname` on the machine named `toto.nowhere.org`:

```
taktuk -s -m toto.nowhere.org broadcast exec { hostname }
```

As a result of this command, TakTuk will connect to `toto.nowhere.org`, broadcast the execution of `hostname` on all connected peers (that is `toto.nowhere.org` only), gather outputs of this command and redirect them to the local machine and, finally, close the connection. Thus, the final output will look like:

```
toto.nowhere.org-1: hostname (6742): output > toto.nowhere.org
toto.nowhere.org-1: hostname (6742): status > Exited with status 0
```

2.1.2 TakTuk outputs, streams

TakTuk displays to the user informations about what's happening with connections and commands as streams of data. The different streams that TakTuk outputs distinguish semantically different informations. TakTuk manages the following streams:

- `output`: data outputted on `stdout` by remotely executed commands
- `error`: data outputted on `stderr` by remotely executed commands
- `status`: cause of remote command termination and exit value
- `connector`: data outputted on `stderr` by connector commands (typically `ssh`)
- `taktuk`: messages from the TakTuk engine itself, usually warnings, errors or debug informations
- `info`: miscellaneous informations that do not fall into one of the other categories

By default, the most common streams (`output`, `error` and `status`), display their information using the following format:

```
hostname-number: command (pid): stream > data
```

2.1.3 My first parallel command

Executing the same command on a given set of peers is straightforward with TakTuk. Actually this is the original intent of this tool. Thus, you just have to give to TakTuk the list of peers and to tell it that you want to broadcast some execution. For instance, to execute the command `uptime` on the three hosts `host1`, `host2` and `host3`:

```
taktuk -s -m host1 -m host2 -m host3 broadcast exec { uptime }
```

Of course, every attentive reader will notice at this point that typing the `-m` switch before each peer name can either require patience or scripting intermediate. Actually, this is not necessary as you can store your list of peer names into a file and let TakTuk help itself using the `-f` switch:

```
taktuk -s -f hosts broadcast exec { uptime }
```

where the file `hosts` contains:

```
host1
host2
host3
```

This feature makes TakTuk usable right out of the box with most batch schedulers. Furthermore, if you dig into TakTuk documentation, you will soon realize that you can even store a part or whole of your command line into a file transmitted to TakTuk using the `-F` switch.

2.1.4 TakTuk commands

As you might have noticed, TakTuk understands simple commands (some of them taking an argument) and combination of simple commands. We have already met the `exec` command, this is a simple command that takes one argument. Its result is to spawn the execution of the command given as argument in the TakTuk instance that receives the `exec`.

Arguments to simple command are given enclosed within a separated pair of matching characters. This might be braces as in previous examples, brackets, parenthesis or any pair of identical usual non alphanumeric characters. The only thing you have to take care of, is the shell interpretation: if you use unprotected quotes or parenthesis, they will be interpreted by the shell and not passed to TakTuk. As an example, one of the previous commands could also have been written:

```
taktuk -s -f hosts broadcast exec = uptime =
```

Simple commands can be modified by special combining commands such as `broadcast`. This combining command sends the following TakTuk command to all the remote peers deployed by TakTuk except the local one. In the previous examples, the local peer was the peer on which the TakTuk command line was executed, and the others ones were deployed using the `-m` switch.

TakTuk can accept any number of successive command separated by semicolons or commas (by default). Additionnally, if you don't give TakTuk any command (or if you use the `-i` switch), TakTuk will enter interactive mode in which it reads successive commands from `stdin`. Finally, notice that TakTuk understands any non ambiguous prefix of commands and supports `readline` when in interactive mode.

2.2 The deployment process: topology and performance

By default, TakTuk uses deployment settings that work well on most machines: it initiates a few connections in parallel (the deployment window) and distribute the remaining deployment work to already deployed peers using an adaptive work-stealing algorithm. Nevertheless, it is possible to finely tune this deployment process if your target architecture has some topological particularities.

2.2.1 Connector performance

If you are interested in speeding up the time TakTuk takes to deploy, the most direct way is to speed up the time taken by individual connections. Of course, the `-s` switch that we employed up to now is very convenient, but it badly affect connection time: it require one more exchange between the local and the remote peer in addition to the raw transmission time of the TakTuk code itself.

Thus, the first way to improve TakTuk performance is to install it on all the remote peers before initiating the deployment. You only have to be sure that TakTuk is findable on the remote peer: either in $PATH or in the same path as on the local node. In some very difficult cases, you can specify the exact path in which taktuk is installed using the `-T` switch. By the following we will assume that TakTuk is installed everywhere.

The second way to improve TakTuk performance is to use a faster command for individual connection. For instance `rsh` is much faster than `ssh`. By default, TakTuk uses `ssh` because it is usually installed and in service everywhere, but you can provide any other connector (a command for remote login) using the `-c` switch.

Finally, you can improve connectors responsiveness by overloading their timeout value using the `-t` switch. This way connection failures (if any) will occur faster than with usual `ssh`. Of course, this will increase the risk of considering slow peers as faulty.

2.2.2 Topology

Because of the work-stealing algorithm used by TakTuk, the topology of the logical network set up during deployment can vary depending on target peers characteristics and load. But it is possible to change partly or even completely the way TakTuk deploys itself.

You can easily get a completely flat deployment topology by disabling the work-stealing using the `-d` switch:

```
taktuk -d-1 -m host1 -m host2 -m host3 -m host4 -m host5 broadcast exec { uptime }
```

The result will be something similar to the deployment tree presented in figure 2.1. A textual representation of this tree can also be displayed by TakTuk using the `network state` command.

In general, the `-d` switch controls the maximal arity resulting from dynamic deployment. When set to -1, the dynamic deployment is not used (no work-stealing). When set to 0 (default), the dynamic deployment has no maximal arity. When set to any positive value, the maximal arity of the tree of dynamically deployed peers is limited to this value. Thus, one can deploy in chain (figure 2.2) using the following command:

```
taktuk -d1 -m host1 -m host2 -m host3 -m host4 -m host5 broadcast exec { uptime }
```

or deploy using a binary tree (one of the possible final tree in figure 2.3) with the command:

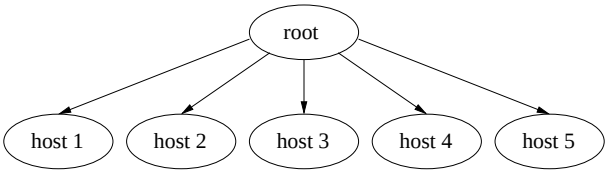

Figure 2.1: Flat deployment on five hosts

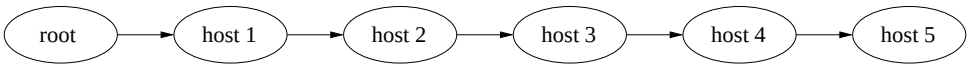

Figure 2.2: Chain deployment on five hosts

```
taktuk -d2 -m host1 -m host2 -m host3 -m host4 -m host5 broadcast exec { uptime }
```

Nevertheless, the `-d` switch has its limitations. For instance, if you want the `host1` to deploy dynamically `host3`, `host4` and `host5` while `host1` itself and `host2` remain deployed statically by the root node, you have to use peer arguments:

```
taktuk -d-1 -m host1 -[ -d0 -m host3 -m host4 -m host5 -] -m host2
    broadcast exec { uptime }
```

you will probably end up with the tree represented in figure 2.4 (keep in mind that the actual topology of the subtree rooted at `host1` will be elaborated dynamically). Using peer arguments, you can even specify a completely static deployment topology. For instance, the chain in one of the previous examples could as well be enforced using peer arguments:

```
taktuk -m host1 -[ -m host2 -[ -m host3 -[ -m host4 -[ -m host5 -] -] -] -]
    broadcast exec { uptime }
```

Peer arguments can also be used to execute a different command on each host (notice that, on this last exemple, we ask the root node to execute the quit command to prevent it to enter interactive mode):

```
taktuk -m host1 -[ exec { uptime } -] -m host2 -[ exec { hostname } -] quit
```

2.2.3 Optimal window

Based on its deployment window, TakTuk decides which connections to initiate locally and which to distribute when it receives work-stealing requests. The size of this window is the maximal number of parallel ongoing connection initiations that can exists within the local TakTuk instance. By default TakTuk sets this window size to 10 which is a value that works well in most cases. Nevertheless, the optimal value for this window – at least when peers are homogeneous – is not necessarily 10 (see publications about TakTuk for more details about optimal deployment).

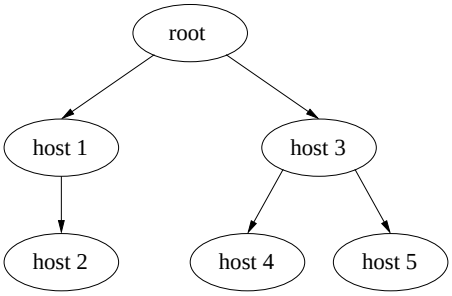

Figure 2.3: Binary tree deployment on five hosts

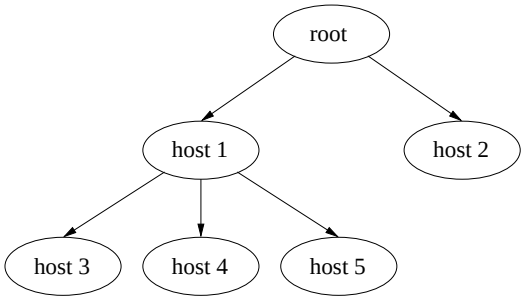

Figure 2.4: Custom tree deployment on five hosts

Thus, if you want to improve further your total deployment time, you have to set the window size to the value which is optimal for your cluster (and do the same thing for each cluster and each individual machine of your grid). A good approximate of the optimal value is the smallest integer x such that a dynamic deployment of p peers using a flat tree topology and a window size s such that $x < s \leq p$ does not take less time than using a window size of x. Using more or less than this value might degrade deployment performance (and for sure will not improve it).

Automatic dynamic window size adaptation is being developped in TakTuk. This is something really difficult as the dynamic evaluator should be both reactive and acurate at time scales of a few tenth of milliseconds. You can try it using the `-W` option, but if you really want the best possible performance, you would better find the optimal window size by direct experiments on the target machine.

2.2.4 Centralized services

Centralised services such as `nfs` (if TakTuk is installed on an `nfs` share for instance) or `ldap` (for connections authentication) will obviously degrade connection time and TakTuk performance when scaling to a large number of nodes. For now, it is recommended to avoid overloading these external services if you are looking for good deployment time and cluster stability.

Future TakTuk improvements will include mechanisms to detect external contention. This way, TakTuk will be able to slow down parallel connection initiations to avoid external servers overloading.

2.3 Configuring TakTuk

Did I tell you that TakTuk was a very versatile tool ? Indeed, all taktuk options can also be set by environment variables. Furthermore, it is possible to change what and where TakTuk displays things, it is even possible to execute some decision program whenever something has to be displayed.

2.3.1 Environment settings

All taktuk options default values can be overridden using environment variable. The name of these environment variables is derived from the option long name: it begins by `TAKTUK_` to which the option long name is concatened after capitalization and replacement of dashes with underscores. If the option accepts named parameters (with the syntax name=value as in `-o` or `-D` options) then an underscore and the concerned name have to be concatened to the preceding string. For examples of environment variables names use TakTuk with the `-P` option.

As with values given as options on the command line, values given as environment variables are propagated to children TakTuk instances. Nevertheless, environment variables settings are overridden by other propagated values and by command line options.

2.3.2 Streams redirection and formating

TakTuk output streams can be redirected to any numeric file descriptor opened on the root instance using the `-R` switch. This gives a way to store each kind of information into a separate file: just write a

wrapper that opens files and forks TakTuk execution with the appropriate redirections. It is even possible to redirect informations outputed by one single peer to a specific descriptor by using peer arguments. By default, TakTuk output everything to descriptor 1 (stdout) except for `taktuk` stream which is redirected to 2 (stderr).

If you are not satisfied with the way TakTuk outputs informations, you can change it using the `-o` option. With this option you can suppress the outputing of one stream if you are not interested with it. For instance, to suppress the status stream for all peers executing commands, use:

```
taktuk -o status -m localhost broadcast exec { uptime }
```

and the output will look like:

```
localhost-1: uptime (8537): output > 18:43  up 8 days, 23:51, 4 users, load averages:
0.19 0.31 0.23
```

But you can also choose which information is displayed and how if you associate an output template to a stream. These templates are Perl expressions in which each information available is stored into a specific variable. This template is applied for each line of output of a command. For instance, if you want to output, for each line of the `output` stream, the peer name followed by a dash and the content of the line, just use the following command:

```
taktuk -o output='"$host - $line\n"' -m localhost broadcast exec { uptime }
```

and the output will look like:

```
localhost - 18:44  up 8 days, 23:52, 4 users, load averages: 0.31 0.31 0.24
localhost-1: uptime (8546): status > Exited with status 0
```

Notice here that the double quotes must be protected from the shell interpretation otherwise the resulting argument would not be a correct Perl expression.

As I mentioned, these templates are Perl expressions. This means that you can perform almost any computation to determine what to output. TakTuk will output the result of the evaluation of this expression. For instance to print the capitalized output of the `uptime` command:

```
taktuk -o output='$line =~ tr /a-z/A-Z/,$line."\n"' -m host1 broadcast exec { uptime }
```

and the output will look like:

```
18:44  UP 8 DAYS, 23:52, 4 USERS, LOAD AVERAGES: 0.43 0.34 0.25
host1-1: uptime (8555): status > Exited with status 0
```

The only thing you have to care about is that your Perl expression must be correct otherwise it will produce some error message whenever some information is outputed on the concerned stream.

These templates might also contain a state information. The variable named `$user_scalar` is defined in TakTuk and destined to store user information. It is initially set to undef. Thus, one can display only odd lines (strating from 0) of the output of the `ls` command by using the following template:

```
taktuk -o output='$user_scalar = defined($user_scalar)?$user_scalar+1:0,
    ($user_scalar%2)?$line."\n":undef' -m localhost broadcast exec { ls }
```

which will output:

```
Ancien Compte
Desktop
Enseignement
Local
Music
Pictures
Research
Telechargements
public_html
localhost-1: ls (27341): status > Exited with status 0
```

where my home directory contains:

```
Admin
Ancien Compte
Archives
Desktop
Documents
Enseignement
Library
Local
Movies
Music
Packages
Pictures
Public
Research
Sites
Telechargements
bin
public_html
```

Finally, it seem important to mention that, due to concurrency, lines outputted by commands might be splitted into several parts. To distinguish between an incomplete line (without final eol character) and a final part of some line, the `$eol` variable contains respectively an empty string or a carriage return.

2.3.3 Options and commands separators

By default, TakTuk uses reasonable characters to separate successive options and successive commands. The overall rationale is: try to behave as much as possible as users expect. The difficult point in TakTuk is that options can be read from the command line as well as from a file (`-F` option). In the first case this is the shell which separates arguments while in the second this is TakTuk itself. But TakTuk does not implement a quotation mechanism as complex as the shell does.

This means, for instance, that if you want to give to an option a value that contains a space character you might have to play with TakTuk parsing options. While this is straightforward using the command line (just protect the space character from shell interpolation), if the option is read from a file you will have either to change the character that separates options (so that the space character is included in the option value) or to protect each space character with an escape character (see option `-E`).

The same consideration applies to command parameters: for instance, if you want to include the character used as a closing brace. In this case, you have either to change your delimiter characters or to protect the closing brace with an escape character.

Chapter 3

Managing the deployed application

This chapters covers more advanced topics that should be of interest to all parallel applications developper. It presents TakTuk communication facilities that have been designed to help parallel applications setup as well as TakTuk deployment network management that enable application processes addition or removal.

3.1 TakTuk communication

TakTuk is not limited to the remote parallel execution of individual independent commands. It can actually be used to start parallel applications. To this intent it provides services such as logical numbering and interprocess communication that help applications setup.

3.1.1 Informations made available to programs

TakTuk makes some environment variables available to all processes that it executes. It includes:

- TAKTUK_COUNT: the total number of successfully deployed TakTuk instances
- TAKTUK_RANK: the logical number of the local instance (numbers range from 1 to $TAKTUK_COUNT)
- TAKTUK_PIDS: the pids of other commands in execution in the local instance (to enable local commands interaction using signals)
- TAKTUK_HOSTNAME: the local hostname (that has been used by TakTuk to contact it)

Notice that if you do not need TakTuk mogical numbering (if you do not plan to use communication or $TAKTUK_COUNT and $TAKTUK_RANK variables) you can use the `-n` option. This option slightly speeds up deployment as it eliminates the last numbering synchronization. In this case the concerned variables are not defined.

3.1.2 Peers set specification

TakTuk implements communication between peers based on their logical number. It is able to send messages in a multicast fashion to any subset of the successfully deployed peers. TakTuk describes a subset of the successfully deployed peers as an union of subintervals of 1..$TAKTUK_COUNT. Each interval is either a single number or the two extremum numbers separated by a dash. The union is given by a list of slash separated intervals.

Peer set specification can also be used to send a command to only a subset of the peers. For instance:

```
taktuk -m host1 -m host2 -m host3 -m host4 1/3-4 exec { echo \$\$ }
```

will execute `echo $$` on three of the four peers (which ones depend on the final numbering which depend itself on the actual deployment tree).

3.1.3 TakTuk communication interface

TakTuk provides an interface that enables programs it launches to communicate. Basically, this communication layer is based on the send/receive model. It is able to send multicast messages and to perform timeouted receives. Notice that this communication layer is not a high performance communication library but rather a control network useful for application setup. It is insensible to machines byte order and is only able to transmit a buffer of bytes from one peer to the other in the same order.

There are two flavors of the communication interface: a perl module built in TakTuk (but that can also be installed on the system) and a C library shipped with the TakTuk package.

Perl version

The Perl interface is built in TakTuk and can be directly used by scripts launched by TakTuk using its `taktuk_perl` command. This command executes a script (just as would do a perl interpreter) but prefetch the interpreter with the `TakTuk` Perl package. Thus the script can use `TakTuk::send` and `TakTuk::recv` functions. Errors and diagnostic messages are also handled by this package (`TakTuk::error` and `TakTuk::error_msg`). The following script gives an idea of how it works, it implements a simple token ring communication using TakTuk:

```
use strict;

my $rank = TakTuk::get('rank');
my $count = TakTuk::get('count');

print "I'm process $rank among $count (new version for 3.6)\n";

if ($rank > 1)
  {
    my ($to, $from, $message) = TakTuk::recv();
    if (not defined($message))
      {
        print "Trying to recv: ",TakTuk::error_msg($TakTuk::error), "\n";
      }
    else
      {
        print "$to received $message from $from\n";
      }
  }

sleep 1;
my $next = $rank+1;
$next = 1 if ($next > $count);
if (not TakTuk::send(to=>$next,body=>"[Salut numero $rank]"))
  {
    print "Trying to send to $next: ",TakTuk::error_msg($TakTuk::error), "\n";
  }

if ($rank == 1)
  {
    my ($to, $from, $message) = TakTuk::recv(timeout=>5);
    if (not defined($message))
      {
        print "Trying to recv :", TakTuk::error_msg($TakTuk::error), "\n";
      }
    else
      {
        print "$to received $message from $from\n";
```

```
    }
  }
```

To execute it, one just has to save it (let us say as `$HOME/comm.pl`) and to type the proper taktuk command:

```
taktuk -m host1 -m host2 broadcast taktuk_perl { comm.pl }
```

then the output will look like:

```
host2-2: taktuk_perl comm.pl (40494): output > I'm process 2 among 2
host2-2: taktuk_perl comm.pl (40494): output > 2 received [Salut numero 1] from 1
host2-2: taktuk_perl comm.pl (40494): status > Exited with status 0
host1-1: taktuk_perl comm.pl (40495): output > I'm process 1 among 2
host1-1: taktuk_perl comm.pl (40495): output > 1 received [Salut numero 2] from 2
host1-1: taktuk_perl comm.pl (40495): status > Exited with status 0
```

C version

TakTuk also provides a C interface to its communication facilities. The use of this interface requires to include `taktuk.h` to program, to link it with taktuk and pthread (because the C interface is thread-safe) and to install the program on all the remote peers. Then it can be executed using the `exec` TakTuk command. For instance, the following C program performs a simple point-to-point communication:

```
#include <taktuk.h>
#include <stdio.h>

int main()
  {
    char buffer[1024];
    size_t length;
    char sample_string[128] = "Salut  toi";
    long rank, from;
    int result;

    result = taktuk_get("rank", &rank);
    if (result)
      {
        fprintf(stderr,"Invalid rank: %s, do you use TakTuk ?\n",
                                                    taktuk_error_msg(result));
        return 1;
      }
    printf("I'm process %ld\n", rank);

    if (rank == 1)
      {
        result = taktuk_send(2, sample_string, 128);
        if (result)
          {
            printf("Error in %ld : %s\n", rank, taktuk_error_msg(result));
          }
      }
    else if (rank == 2)
      {
        fflush(stdout);
        result = taktuk_recv(&from, buffer, &length, 0);
        if (result)
          {
```

```
          printf("Error in %ld : %s\n", rank, taktuk_error_msg(result));
        }
      else
        {
          printf("Received : %s of length %ld from %ld\n", buffer, length,
                                                                    from);
        }
    }
  else
    {
      printf("Got nothing to do\n");
    }
  return 0;
}
```

3.2 Adding new nodes to an already completed deployment

TakTuk is able to connect to new nodes after an aready completed initial deployment phase. The only real issue when adding new nodes is that they did not take part in the numbering step, thus TakTuk should give them a logical rank that is compatible with the overall numbering. This issue is solved by TakTuk by separating these nodes additions in two steps: the addition of new connection and the numbering modification.

3.2.1 New nodes

New nodes are added to an existing TakTuk deployment network by using its `option` command to give to TakTuk new machine deployment options (`-m` or `-f`). These new nodes are deployed from the TakTuk instance that executes de `option` command. For instance, the following TakTuk command:

```
taktuk -d-1 -m host1
```

deploys one node (named `host1`) and enters interactive mode. Then, typing the following commands:

```
option [ -m host3 -[ -m host4 -m host5 -] ]
1 option m [ host2 ]
```

you will end up with a deployment tree similar to the one presented in figure 2.3 (in section 2.2.2).

Notice that the order of the two options commands does not matter: during this step, TakTuk does not assign logical number to the newly deployed nodes, the host `host1` is the only one to have a logical number in this example and the second option command is not ambiguous. Notice also that communication to these new nodes is not possible as long as they do not have a logical number.

3.2.2 Numbering modification

TakTuk is able to modify dynamically the numbering of its nodes. This is the second step of the dynamic extension of the deployment network, wich is used to assign logical numbers to newly added nodes. This features comes in two flavors: a conservative numbering update or a complete renumbering.

One can require TakTuk to perform a numbering update by sending the `network update` command to its root node. The numbering update might be only partially possible: some nodes remain unnumbered after its completion. This is due to TakTuk internal messages routing which rely on a depth-first numbering scheme to eliminate the need for rooting tables and routing informations propagation. Thus, assigning a logical number to a newly added node will only be possible if all the already numbered ancestors of this node are located on the rightmost branch of the subtree formed by already numbered nodes.

This means that in our previous example (in section 3.2.1) all the newly added nodes can be updated using the `network update` command. In contrary, in the following example, the numbering update will fail for `host2`:

```
taktuk -d-1 -m host1 -m host3 -[ -m host4 -]
network state
astaroth.local (0, 1)
    host1 (1, 1)
    host3 (2, 1)
        host4 (3, 1)
1 option m [ host2 ]
2 option m [ host5 ]
network state
astaroth.local (0, 1)
    host1 (1, 1)
        host2 (-1, 1)
    host3 (2, 1)
        host4 (3, 1)
        host5 (-1, 1)
network update
astaroth.local (0, 1)
    host1 (1, 1)
        host2 (-1, 1)
    host3 (2, 1)
        host4 (3, 1)
        host5 (4, 1)
```

In this example, the subtree comprising `host1`, `host3` and `host4` of tree from figure 2.3 is first deployed. Then `host2` and `host5` are added to the tree and the numbering update is performed. This update fail for `host2` would require a number between 1 (`host1`) and 2 (`host3`). Notice that these events (numbering success or failure) are reported in the TakTuk stream `state`.

The other possibility is to require a complete renumbering from TakTuk using its `network renumber` command. In this case all nodes are assigned a logical number, possibly different than the number they might had before renumbering. Using a complete renumbering on the previous example end up in the assignment of a new logical number for `host3` and `host4`:

```
taktuk -d-1 -m host1 -m host3 -[ -m host4 -]
network state
astaroth.local (0, 1)
    host1 (1, 1)
    host3 (2, 1)
        host4 (3, 1)
1 option m [ host2 ]
2 option m [ host5 ]
network renumber
network state
astaroth.local (0, 1)
    host1 (1, 1)
        host2 (2, 1)
    host3 (3, 1)
        host4 (4, 1)
        host5 (5, 1)
```

Chapter 4

TakTuk and the outside world

This last chapter covers existing interaction mechanisms between TakTuk and the executino environment. It explains how to use TakTuk to manage files on remote hosts, to perform usual shell idioms in a parallel way and to use TakTuk from within another application.

4.1 TakTuk files I/O

TakTuk provides some basic file transfer capabilities. The transfer is performed using the deployment network, that is a tree made by default of `ssh` connections. This is not the most efficient way to transfer large file. Thus, keep in mind that TakTuk files I/O capabilities have been implemented for convenience, not high performance.

4.1.1 On connection transfers

Using the `-S` option, one can ask TakTuk to tranfer some files or directories to remote hosts just after connection initialisation. This feature has been developped only to help user in writing their own connection programs. This means that if the program you use to connect to individual remote hosts requires some files that are not installed on all of these remote hosts, TakTuk can propagate them for you during the deployment itself. As TakTuk tries to keep files permissions unchanged, this is perfectly suited to the propagation of connection scripts.

4.1.2 Regular transfers

For more usual files transfers, TakTuk accepts the commands `put` and `get`. The `put` command copies a source file or directory to some destination path on all the target hosts. For instance, the following command copies the file `/tmp/configuration.txt` to the `/tmp` directory of `host1` and `host2`:

```
taktuk -m host1 -m host2 broadcast put { /tmp/configuration.txt } { /tmp }
```

Overall, the put command works like `scp`. It can handle either single files or directories. Furthermore, contrary to `scp`, it also tries to keep files permissions unchanged in the copy.

The natural complement of `put` is the `get` command. This command copies some source files or directories (the first argument, which is a single path that must exists on all the target hosts) to the node executing the command. In addition to shell interpolation of environment variables, `put` and `get` commands understand the `$host`, `$rank` and `$position` variables which are respectively expanded to the hostname, the logical rank and the position on the command line of the source host. This means that the following command copies the files named `/tmp/configuration.txt` on `host1` and `host2` to the files `/tmp/configuration-host1.txt` and `/tmp/configuration-host2.txt` on the localhost.

```
taktuk -m host1 -m host2 broadcast get { /tmp/configuration.txt }
    { /tmp/configuration-'$host'.txt }
```

4.2 Howto

This section contains some "trick" commands to perform simple tasks that are usually useful in a parallel environment but are not part of TakTuk functionalities (and never will be).

4.2.1 Execute on remote peers a script stored on root node

The principle is very simple: execute a command able to execute code piped on its standard input on all the peers and send the content of the script as input to all these commands. This translates into:

```
taktuk -m host1 -m host2 broadcast exec { perl -- - }\;
    broadcast input file { myscript.pl }
```

4.2.2 Pipe the result of a local execution to the input of remote commands

This could be named the 'parallel pipe', the principle si the same as in the previous example, but in addition, it rely on the fact that TakTuk uses Perl's `open` function to open its files:

```
taktuk -m host1 -m host2 broadcast exec { cat '>' /tmp/result.txt }\;
    broadcast input file { 'echo result |' }
```

Notice here that it is necessary to protect the > and the semicolons from the shell (the first one has to be interpreted on the remote peer only, and the second by TakTuk).

4.3 Using TakTuk from within an application

Using TakTuk from within an application can be tedious: one has to manage a forked TakTuk process, to redirect I/Os while handling I/Os from the application itself, to demultiplex TakTuk output, and so on. Thus, since its 3.6.2 release, TakTuk is shipped with a pilot library which alleviate these issues. At the time of this writting, this library is only available for the Perl scripting language under the form of a module.

4.3.1 Running TakTuk and sending it commands

The TakTuk pilot library is I/Os "events" driven. This means that the function which actually executes TakTuk is blocking, it returns only when TakTuk terminates its execution. As we will see in the next section, it is nevertheless possible to interact with a running TakTuk using callback functions.

In its simplest form, the pilot library can be used by performing the following actions: creating a new TakTuk object, possibly sending some commands (that will be buffered) to this object, sending a termination request (that will also be buffered) and running a TakTuk command line. The last step will actually create a TakTuk process to execute the given command line, execute buffered TakTuk commands (if any) and wait for TakTuk completion to return.

Nevertheless, this simplest form is not really useful as it does not print any command output. Actually, I/Os are managed in a special way in TakTuk pilot. By default, they are just thrown away. Thus, we will wait the following section, which covers the I/Os management aspects in TakTuk pilot, to present a detailed code example.

4.3.2 Managing I/O and TakTuk data

I/Os management in TakTuk pilot can be handled by callback functions. These functions have to be written by the user and registered in the TakTuk pilot object. Upon registration, a callback function is attached to some TakTuk output stream: whenever data is outputted on this stream, it is decoded by TakTuk pilot and passed as arguments to a call to the attached function.

When registering a callback function in some TakTuk object for some TakTuk stream, the user has to provide the list of output fields he is interested in. These fields can be any variable name usable in a template formatting specification for the related stream (see TakTuk manual for a list of all variables usable in templates). Afterward, whenever some output occur in any part of the deployed TakTuk

processes, all these requested fields are packed by TakTuk, propagated to the root node, decoded by TakTuk pilot and passed to the callback function as a hash.

These callback function are a way to interact with a running TakTuk object. Indeed, within a call to such function, the user can send new commands to the TakTuk object or ask it for termination. Fortunately this interaction is not limited to TakTuk outputs: user can also require TakTuk pilot to monitor I/O events on their own file streams. In this last case, TakTuk does not perform any data reading or writing on the monitored streams: this is just a way to give control back to the application temporarily without returning from the `run` function.

The following example is a quite comprehensive illustration of TakTuk pilot capabilities. It is a minimal parallel shell. It registers two callback functions and uses TakTuk pilot to connect to remote host which names are passed as arguments. Then, whenever the running TakTuk produces output, the first callback function prints it along with a per-host line numbering. Furthermore, whenever the user types a line, the second callback function reads it and uses it as a command to execute on all remote hosts.

```
use strict;

use TakTuk::Pilot;

our @line_counter;

sub output_callback(%) {
    my %parameters = @_;
    my $field = $parameters{fields};
    my $rank = $field->{rank};
    my $argument = $parameters{argument};

    $argument->[$rank] = 1 unless defined($argument->[$rank]);
    print "$field->{host}-$rank : $argument->[$rank] > $field->{line}\n";
    $argument->[$rank]++;
}

sub user_input_callback(%) {
    my %parameters = @_;
    my $taktuk = $parameters{taktuk};
    my $descriptor = $parameters{filehandle};
    my $buffer;

    my $result = sysread($descriptor, $buffer, 1024);
    warn "Read error $!" if not defined($result);
    chomp($buffer);

    if (length($buffer)) {
        print "Executing $buffer\n";
        $taktuk->send_command("broadcast exec [ $buffer ]");
    }
    if (not $result) {
        print "Terminating\n";
        $taktuk->remove_descriptor(type=>'read', filehandle=>$descriptor);
        $taktuk->send_termination();
    }
}

die "This script requieres as arguments hostnames to contact\n"
    unless scalar(@ARGV);

my $taktuk = TakTuk::Pilot->new();
```

```
$taktuk->add_callback(callback=>\&output_callback, stream=>'output',
                      argument=>\@line_counter,
                      fields=>['host', 'rank', 'line']);
$taktuk->add_descriptor(type=>'read', filehandle=>\*STDIN,
                        callback=>\&user_input_callback);
$taktuk->run(command=>"./../../taktuk -s -m ".join(" -m ", @ARGV));
```

As the reader must have guessed, one should not use TakTuk template or reading from `stdin` (using the `-` special name for files) in the command given to the `run` function. Doing this would break TakTuk pilot behavior.

Chapter 5

Conclusion

Although this guide gives a fairly good overview of what it is possible to do using TakTuk, some options and commands are not described here. For a complete and detailed description of TakTuk, please refer to its manpage. Nevertheless, the intend of this guide is mainly to teach you TakTuk usage, if possible in a didactic way. I hope it will be usefull for most users, as advanced TakTuk usage is not always obvious. Beside user documentation, more informations can be found on the TakTuk website:
`http://taktuk.gforge.inria.fr`
This includes references to scientific publications regarding TakTuk as well as information about its use in other project and real world experiments.