Chapter 3

Using Schedplot

Schedplot is normally used in three sequential steps: profile, analyze, view.

3.1 Profile

In this step, Schedplot profiles the desired program using `erlang:trace/3`. The simplest way to start the profiling is by using `schedplot:start/[1-3]`

> `start(Fun) → 'ok'.`
> Same as `start(Fun, "schedplot_trace", [])`
>
> `start(Fun, Dir) → 'ok'`
> Same as `start(Fun, Dir, [])`
>
> `start(Fun, Dir, Flags) → 'ok'`

Types:

> `Fun = fun()`
>
> `Dir = file:filename() | atom() | integer()`
>
> `Flags = [start_flag()]`
>
> `start_flag = 'gc' | 'no_auto_stop' | 'trace_all' | 'trace_mfa'.`

`Dir` should be an (existing or not) directory name. If the directory does not exist it will be (attempted to be) created. Missing parent directories will not be created. In this directory all the trace files will be stored – if there are any files with the names used by schedplot to store the trace files, they will be overwritten.

After the initalizations, `schedplot:start/3` will use `erlang:apply/2` to run `Fun` in a separate process. The process that applied `Fun` as well as its children will be the only ones traced (unless the `trace_all` flag is used). Once `Fun` returns, the profiling will stop (unless the `no_auto_stop` flag is given) and the trace will be saved. It is therefore of utmost importance to make certain that `Fun` will not return before we want to end the tracing. Consider for example the following code:

```
fib(N, M) ->
    \_ = [spawn(fun() -> fib(N) end) || \_ <- L],
    ok.
```

Listing 3.1: Wrong way of profiling a program

which spawns M processes that will calculate the nth fibonacci number. One might expect to trace it by calling `schedplot(module,fib,[N,M])` but that would only trace the spawning part (a few micro-seconds).

To trace the whole program something like the following should be done:

```
fib(N, M) ->
    L = lists:seq(1, M),
    \_ = [spawn(fun() -> fib\_w(self(), N) end) || \_ <- L],
    \_ = [receive ok -> ok end || \_ <- L],
    ok.
```

Listing 3.2: Correct way of profiling a program

where `fib_w/2 is` a function wrapper that would send ok after calculating fib(N).

Of course, there could be different approaches but the gist is that the first Fun termination will signal the end of the tracing.

If the `no_auto_stop` flag is used however, the tracing will continue until `stop/0` is called. It is up to the user to either call it in the program under whatever conditions he wishes or by hand during execution.

The profiling will produce N files, one per scheduler trace plus one if the gc flag was used, named `‘‘raw_trace’’++N` where N is an integer denoting the scheduler ID. It will also produce a header file named raw_trace_header

Note that `schedplot:start/[1-3]` may be called from within a program multiple times; however, if the directory name is the same only the last trace will remain It is also impossible at the moment to combine multiple traces in one.

Flags

gc: enables the tracing of the garbage collector.

trace_all: will trace all processes in the node (including the tracer)

trace_mfa: the tracer will store the `PID` and `MFArgs` when a traced process enters or leaves a scheduler (of course the `PID` will be the same both times). Currently there is no use for those data from schedplot's viewer but they could be used later or by the user

no_auto_stop: the tracing will not stop when `Fun` returns

3.1.1 Using `schedplot:print/1`

```
print(Label) → 'ok'
Label = string() | atom() | integer()
```

There is no need to use any flags in `start/[1-3]` or start other tracing processes; just call `schedplot:print/1` from the desired points of the profiled program. Note however that it is not recommended to make too many calls (over 100 thousand)

3.2 Analyze

Analyzing is a straight-forward procedure; simply use the following functions to produce the files that will be used by the viewer.

`analyze() → 'ok'`
same with `analyze(“schedplot_trace”)`

`analyze(Dir) → 'ok'`
same with `analyze(Dir, [])`

`analyze(Dir, Flags)`

`Dir` is the name of the directory where the raw trace files are stored and where the analyzed trace files will be stored (therefore it is required to have both read and write privileges in that folder).

Analyzing the traces usually is almost instantaneous and can be done in a different machine than the one used to do the profiling; generally it will be faster when the number of schedulers increases up to the point that we reach the number of schedulers used in the profiled program. It is also interesting to note that the speedup will be similar to the speedup of the profiled program meaning that analyzing the trace of a sequential program that ran in N schedulers where N>1 will yield little to no speedup when run in M>1 schedulers. Typically the analyzed files would be larger (yet not a lot larger) than the raw trace files, so, considering the speed of the analyzer, it might be better to store them for future use instead of the analyzed files if memory is an issue.

The analyzer will produce N files, one per scheduler trace plus one if the gc flag was used, named `“analyzed_trace”++N` where N is an integer denoting the scheduler ID. Afterwards the raw trace files are no longer required so it is safe to remove them.

3.3 View

To start the viewer use one of the following functions:

`view() → 'ok'` same with `view(“schedplot_trace”)`

`view(Dir) → 'ok'` same with `view(Dir, 1000,700)`

`view(Dir, Max_Width, Max_Height)` starts the viewer displaying the trace stored in `Dir` directory; the window's max width will be `Max_Width` and max height will be `Max_Height`.

Note that the final width and height will depend on the size of the graph: the viewer will start with a size that fits the graph at the maximum zoom possible without exceeding the max size given. It is not possible at the moment to detect the maximum screen size in a trivial way due to wxErlang limitations; however, it is possible to change the macro values to fit the screen resolution used in the schedplot.hrl file (macros WIDTH and HEIGHT).

It is suggested to run the viewer in a local machine to avoid any network latency

Once the viewer started, the following keys are used for navigation:

- Move right/left:
  use the arrow keys (right/left) or 4/6 numpad keys to move (50px)
  if the alt key is pressed the movement will be smaller (10px)
  if the control key is pressed the movement will be larger (200px)
- Move up/down:
  use the arrow keys (up/down) or 8/2 numpad keys to move 1 scheduler up or down
  if the alt key is pressed then it will move 10 schedulers up or down
- Zoom in/out:
  use the numpad +/- or the regular +/- or =/_ to zoom in or out
  zooming in will result in 2x while zooming out in 0.5x
  the zoom will preserve the same starting position; that is, if the graph started at 12.4 sec and ended at 22.4 sec the zoomed-in graph will start in 12.4sec and end at 17.4sec and the zoomed-out graph will start in 12.4sec and end at 32.4sec
- Select an area:
  Click the right mouse button and drag the mouse. Release the right button at the desired point. A cyan box will appear around the selected area (it does not matter if the dragging is from left to right or otherwise or if it constantly to the same direction; all that matters is the first and last point). It does not matter if an area was selected previously
- Cancel a selection:
  Press escape
- Zoom in to selection:
  While having selected an area press a zoom in key. The viewer will display an area starting at the start of the selection and using the maximum zoom (from the valid zoom levels) that would display the whole selection. It is therefore possible to have some extra data displayed at the end of the graph.

- Reset:
  Press the Home key to return to the first state of the viewer.

Note it is not possible to zoom in or out more than a certain number while displaying times before zero is possible although it has little practical meaning.