SSWAT: System Wide Analysis Tool

User's Manual v1.0

Table of Contents

1. Introduction
2. Requirements
3. Installation
4. Functional overview
5. Configuration
6. Visualization
7. Step-by-step guide
8. Utilities

1. Introduction

The purpose of this tool is to provide the user with an infrastructure for system wide analysis. The analysis is divided into three different parts which we will discuss in detail later: gathering, analyzing and visualizing the data.

Oprofile (a system wide profiler) is used in order to profile the kernel and along with other UNIX features, data from various sources are gathered. By analyzing these data a visual representation of live system state is possible and a wealth of accompanying information is available which aid in decoding complex system behavior.

2. Requirements

Oprofile installation and proper configuration of the kernel. Follow the instructions found here [http://oprofile.sourceforge.net/news/].

In order to profile the kernel you have to provide the `vmlinux` file to oprofile [http://en.wikipedia.org/wiki/Vmlinux] .

In order to use the graphing features of SSWAT your system must have either gnuplot (version 4.6 and up) or jgraph.

In order to use the parser provided as a utility the C++ boost library is needed.

Standard UNIX utilities have to be installed in order to use SSWAT. To name a few: `grep/egrep, cat/tac, sort, uniq,sed, awk, bs`.

3. Installation

Copy the application directory [`/sswat`] at your selected location. Only the parser [`sswat/utils/name_extractor.cpp`] provided as a utility requires compilation (c++/boost library needed). It is not an integral part of the system and can be substituted by any other application that provides the same functionality [see section 6. Visualization].

4. Functional overview

This section gives a general overview of SSWAT and the basic features that comprise it. For a detailed guide on how to use each feature see section 7. Step-by-step guide.

SSWAT's operation can be divided into 3 parts which are built around 3 "main" scripts:

- measurement
- analysis
- visualization

Measurement: this part relies on the `run.sh` script. The command line argument to this script is the test name desired.

In order to start measurements, a command line must be provided in the `run.sh` script at the designated user editable part at the head of the script.

By using the settings provided in the configuration file [see section 5. Configuration] and running the script, a test directory is created which contains the oprofile output, raw data from measurements and an initial info file that is further enhanced in the analysis part.

Data gathered during this step are:

- oprofile measurements
- `/proc` entries selected in the configuration file
- iostat/vmstat information
- start/finish time stamps
- system information (machine name, kernel version, memory, number of cores)
- KVM counters (optional)

Some of the data gathered are sampled in intervals (iostat/vmstat). For proc entries, snapshots are taken before and after measurements.

Analysis: raw data gathered in the measurement step are analyzed in this part of the process. This analysis is based on the `start_analysis.sh` script, while helper scripts reside in the `sswat/analysis` directory. In order to process raw data, the name of the desired test must be provided as a command line argument to this script. Apparently, it must point to a test directory which was created by using the `run.sh` script in the previous step.

By running this script, the following information is added to the info file residing in the test directory specified:

- Execution time
- Oprofile information (total samples and sample rates)
- Aggregate CPU statistics (idle,user,system,iowait times and cpu utilization)
- Per-CPU statistics
- Total interrupts
- vmstat interrupt information
- Device information (average and maximum utilization, average queue depth)

- Disk information (number of write/read operations, number of sectors read/written)
- File-system performance counters information (currently supporting xfs and bladefs)
- KVM counter information

Visualization: in this step, graphs are created by using the data gathered and analyzed in the previous steps. In order to generate graphs using the tools residing in the `sswat/graphing` directory, the following data have to be available:

- Execution time (for execution time based graphs)
- CPU statistics (aggregate or per-CPU)
- Oprofile output

In the current version of SSWAT there are multiple scripts that can be used to generate the aforementioned graphs:

`autograph.sh`: generates a single graph in jgr format (jgraph)
`multigraph.sh`: generates per-CPU graphs in jgr format (jgraph)
`gnuplot_gen.sh`: generates up to 4 combined graphs in gnuplot format

For more information on the form of the graphs and the available utilities see section 6. Visualization.

5. Configuration

SSWAT is built around a central configuration file [`sswat/config`] which is read by the various scripts that comprise the tool. This design was chosen in order to minimize command line arguments needed by the scripts. In a sense it "enforces" the use of a relatively steady path to the working directory if convenience is in mind, but it can be changed by the user at will.

At minimum, the user must provide the following information in the config file:

`working directory`: this is the path to the working directory which will contain the separate test directories created by the tool.
e.g: `working_directory: /home/user/sswat`

`application`: the name of the application, which is used as an identifier for the test directory created by the tool. The term "application" is used in a loose manner, as this serves mainly as the first part of the test directory name in the naming convention used.
e.g: `application: DEDUP`

By combining the fields mentioned so far, a test directory created by the tool [see section 7. Step-by-step guide] would be `/home/user/sswat/DEDUP<test_name>`.

`oprof_vmlinux`: path to the uncompressed kernel image [see 2. Requirements]

`oprof_event`: the performance counter event used by oprofile. [see 2. Requirements]

`oprof_threshold`: the "rate" at which oprofile gathers samples. [see 2. Requirements] `oprof_mode`: possible values are `none` and `separate`. `none` option gives standard oprofile results (aggregate samples). `separate` option gives per-CPU oprofile results. Using an SSWAT utility (discussed later) per-cpu results can be converted to standard oprofile output so it is advised to use the separate option, unless per-CPU results are not needed.

Other important fields of the configuration file are:

`oprof_kernel_module_paths`: path to the custom kernel module directory. This is important if you wish to profile kernel modules, since oprofile cannot find the required symbols if it is not provided with paths to the ".ko" files. If multiple paths are to be provided the should be comma separated.

e.g: `oprof_kernel_module_paths: /home/user/devel/mod1,/home/user/devel/mod2`

The "standard" linux path to the kernel modules (`` /lib/modules/`uname -r`/build ``) is already integrated to the oprofile command line so there is no need to provide it.

`kvm_support`: turns on/off capturing and analysis of the kvm counter.

`proc_selection` and `proc_<num>` are part of the proc entry snapshot feature of SSWAT. Proc entries selected will be saved before and after taking measurements, named `<name>_before/after` and placed in the stats directory of the test directory.

e.g
```
proc_selection: 1
proc_1: /path/to/proc_entry name_of_entry
```

`graph_functions`: number of individual functions that will appear on the graphs generated by SSWATs scripts [see section 6. Visualization]

`oprof_result_step`: the number of columns in the oprofile output (eiter 4 or 5). This value is automatically recognized and updated by the graphing scripts, so there is no need to edit.

`mount_point`: this value determines the point on which a graph will be mounted by the `autograph.sh` script [see section 6. Visualization]. This has to be changed manually if the provided `merger.sh` utility [see section 8. Utilities] is not used.

6. Visualization

Using analyzed data, informative stacked bar graphs can be generated. Two different graph formats are supported by SSWAT: execution time aggregate graphs and per-cpu graphs.

Graphs are generated by combining CPU-statistics, oprofile results and in the case of execution time based graphs, execution time. Elements appearing on the stacked bar graphs are: (a) CPU time entities (idle/iowait/user/system etc), (b) individual functions attributed to either kernel spacer or user space that are above a certain threshold (typically 0.1%) and (\c) kernel subsystems.

6.1 Grouping results into subsystems

Individual kernel functions occupying the CPU in small percentages in runs without a top-load function can be misleading. In order to make analysis more fine-grained, kernel subsystem tags are added to the original oprofile results. By creating filters that contain functions or specific keywords from the kernel space and applying them on the oprofile results, we group percentages and add tags that bring otherwise hidden elements to the foreground.

The filters applied currently by SSWAT are: mm (Memory management), sched (scheduler), fs (file-system specific), radix (radix-tree), rb (ring-buffer). This are either important kernel subsystems we have found appearing frequently or heavily used data structures that reside in the kernel.

Additional filters can be applied at will, using the `name_extractor` utility discussed in section 8. Utilities.

6.2 Types of graphs

6.2.1 Aggregate execution time-based graphs

We “translate” CPU times to actual time by projecting the cpu percentages reported by `/proc/stat` to the execution time of the experiment. As a result abstract percentages are translated into actual time. The y axis represents time (in seconds) and each element appearing on the graph takes up space analogous to it's time “consumption”.

An example graph of this type can be seen in Figure 1.

6.2.2 Per-cpu graphs

In per-CPU graphs, the y axis represents the per-CPU time aggregate (100%) reported by `/proc/stat`. There is no notion of time in per-CPU graphs because we cannot accurately correlate execution time to individual CPU's.

6.3 Device utilization graphs

It is also possible to create a device utilization graph out of the iostat statistics gathered during execution. This functionality is provided by the `graph_device_util.sh` script [`graphing/graph_device_util.sh`]. This script generates a gnuplot script with the accompanying data file needed. Device utilization is graphed using a simple line-graph; multiple device utilization graphs can exist on the same chart.

7. Step-by-step guide

This section is aimed to guide you through each part of SSWAT by using detailed examples.

7.1 Configuration file

We first need to edit the config file [see section 5. Configuration]. Here is the configuration file settings we are going to use for this example:

```
#GENERAL
working_directory: /home/user/testing/sswat_output
application: TEST
kvm_support: off

proc_selection: 1 2 3 4
proc_1: /proc/diskstats dev
proc_2: /proc/stat CPU
proc_3: /proc/fs/xfs/stat xfs_counters
proc_4: /proc/interrupts ints

#OPROFILE_CONF
oprof_vmlinux_path: /usr/src/linux/vmlinux
oprof_kernel_module_paths: /home/user/dev/partitioned_io_stack
oprof_event: CPU_CLK_UNHALTED
oprof_threshold: 175000
oprof_mode: separate

#ANALYSIS
device_list: sda sdb sdc sdd

#GRAPHING
graph_functions: 5
mount_point: 4
```

7.2 Configuring and running the `run.sh` script

At this point we need to edit the `run.sh` script [`sswat/run.sh`] and add the `sswat` directory location and the command line we want to measure using SSWAT, at the user editable part. For this example we are going to measure the system idling for 10 seconds.

```
.
.
#USER: Enter path to the directory the tool resides and command line to measure
tool_path="/home/user/sswat"
cmd="sleep 10"
#/USER
.
.
```

Measurements start by running the `run.sh` script. A command line argument must be provided, which has to be the test name. This is the command line used:

```
$ ./run.sh example_case
```

After the run has finished the directory `/home/user/testing/sswat_output/TESTexample_case` contains all data gathered during measurements and is defined as the “test directory”.

At this point this test directory contains the following files/directories: `info_example_case`, `oprof_results_example_case` and the directory `stats`.

info_example_case:

```
info_example_case - 2012-06-24
```

```
      begin: 13-52-45
      end: 13-52-55

      System info:
      ------------
      Linux test_machine.ics.forth.gr 2.6.32-user #2 SMP Thu Sep 15 13:46:33 EEST    2011
x86_64 x86_64 x86_64 GNU/Linux
      Cpus: 8
      Memory: 12320984 kB
```

This is the initial state of the info file. At this point only date, time-stamps and system information (kernel info, number of cpus and availabled memory) are contained in the information file.

`oprof_results_example_case` contains the oprofile output.

The raw data files from the proc entries we selected reside in the stats directory. Iostat and vmstat output is also contained here. The files listed for this example:

```
      CPU_after_example_case
      CPU_before_example_case
      dev_after_example_case
      dev_before_example_case
      ints_after_example_case
      ints_before_example_case
      iostat_example_case
      vmstat_example_case
      xfs_counters_after_example_case
      xfs_counters_before_example_case
```

7.3 Analyzing the data

In this step we have to run the `start_analysis.sh` script which resides in the `sswat` directory (`sswat/start_analysis.sh`). Here is the command line:

```
      $ ./start_analysis.sh example_case
```

After running this script, the info file in the test directory is enhanced with the information gathered by analyzing the raw data. This is the final state of the info file:

```
info_example_case - 2012-06-24

begin: 13-52-45
end: 13-52-55

System info:
------------
Linux test_machine.ics.forth.gr 2.6.32-spyros #2 SMP Thu Sep 15 13:46:33 EEST 2011 x86_64 x86_64 x86_64
GNU/Linux
Cpus: 8
Memory: 12320984 kB

Execution time: 0:10
---------------

Oprofile sampling information
-----------------------------
Total samples: 7371
```

```
Samples per second: 737.10

CPU statistics:
---------------
user: .0616
system: .2979
idle: 99.5993
irq: 0
sirq: 0
iowait: .0410
cpu%: .3500

Per CPU statistics:
-------------------
                user   system   idle   irq   sirq   iowait  cpu_util
cpu0            0      .1642  99.8357  0     0      0       .1600%
cpu1            .0822  .5756  99.2598  0     0      .0822   .6500%
cpu2            0      .0821  99.9178  0     0      0       .0800%
cpu3            0      .0821  99.9178  0     0      0       .0800%
cpu4            .0821  .0821  99.8356  0     0      0       .1600%
cpu5            .2465  1.3968 98.2744  0     0      .0821   1.6400%
cpu6            0      .0823  99.8353  0     0      .0823   .0800%
cpu7            0      0      100.0000 0     0      0       0%

CPU0   CPU1   CPU2   CPU3   CPU4   CPU5   CPU6   CPU7
0      105    0      0      0      0      0      0      PCI-MSI-edgeeth0-rx-0
0      0      0      0      99     0      0      0      PCI-MSI-edgeeth0-tx-0
6      0      0      0      0      0      0      0      PCI-MSI-edgeeth1-rx-0
670    1379   495    541    669    2400   524    410    0      0      NMI
12170  12169  12169  12169  12169  12169  12169  12169  0      0      0      LOC
0      5      1      1      18     29     1      0      0      0      RES
8      1      3      3      2      3      3      3      0      0      0      CAL
2      51     23     7      0      28     10     3      0      0      TLB

Devices:
---------------
sda        avg=0 : max=1
       avg queue depth: .0005
sdb        avg=0 : max=0
       avg queue depth: 0
sdc        avg=0 : max=0
       avg queue depth: 0
sdd        avg=0 : max=0
       avg queue depth: 0


Disk stats:
-----------
Device: sda
       Write operations: 6
       Sectors written: 88
       Read operations: 2
       Sectors read: 48
-----
Device: sdb
       Write operations: 0
       Sectors written: 0
       Read operations: 0
       Sectors read: 0
-----
Device: sdc
       Write operations: 0
       Sectors written: 0
       Read operations: 0
       Sectors read: 0
-----
Device: sdd
       Write operations: 0
       Sectors written: 0
       Read operations: 0
       Sectors read: 0
-----
```

```
Total write ops: 6
Total sectors written: 88
Average write rq size: 14.6
Total read ops: 2
Total sectors read: 48
Average read rq size: 24.00
```

7.4 Generating graphs

At this point we have all data needed to generate graphs using the tools SSWAT provides. As mentioned in sections 4. Functional Overview and 6. Visualization there are two types of graphs supported. We will go through the available scripts to generate them.

7.4.1 Per-cpu graphs

Graphs of this format can be generated using the `multigraph.sh` script which resides in the `sswat/graphing` directory. The `oprof_mode: separate` option had to be specified before measurements in order to get per-cpu oprofile results.

The command line to generate the graph using `multigraph.sh` is:

```
     $ ./multigraph.sh example_case
```

A .jgr (jgraph file format) file will be created in the test directory specified. In this case the file is: `/home/user/testing/sswat_output/TESTexample_case/multigraph_example_case.jgr`

7.4.2 Execution time based graphs

There are two scripts available to generate execution time based graphs, which use different graphing utilities: `autograph.sh` which uses jgraph and `gnuplot_gen.sh` which uses gnuplot.

`autograph.sh`
This script receives the test name to be graphed as a command line argument. A sample command line to use this script is:

```
     $ ./autograph.sh example_case
```

As mentioned, this script creates a single graph. By using the `merger.sh` utility it is possible to merge multiple stacked bar graphs created with `autograph.sh` into a single .jgr file. Information on this utility will be given in section #Utilities.

`gnuplot_gen.sh`
In contrast to the previous script, `gnuplot_gen.sh` needs no command line arguments. There is a user editable part at the head of the script where we can configure the graphs that will be generated.

```
####USER####
test_names=( "example_case" "example_case0" )
top_title="Test_graph"
sub_titles=( “Example_sleep_10s" "Example0_sleep_20s" )
graph_name="guide"
###/USER####
```

`test_names`: the names of the tests to be graphed. Multiple tests names can be provided [up to **4** in the current version]. These tests have to be under the same `working_directory` path and have the same `application` name, which are fields provided in the `config` file.

`top_title`: the title appearing above the graphs.

`sub_titles`: the per-graph titles appearing below each graph. The subtitles have to be provided in the same order as the test names.

`graph_name`: name of the gnuplot script and data generated. In this case, the output will be `data_guide.dat` for the gnuplot raw data and `script_guide.gp` for the gnuplot script.

After configuring the script and executing it [`$ ./gnuplot_gen.sh`] the data and script files with be generated in the `sswat/graphing` directory. In order to generate the .ps file just run gnuplot in the following way:

```
$ gnuplot script_guide.gp
```

This concludes the step-by-step guide for the main features of SSWAT.

8. Utilities

Additional scripts and a parser can be found in the `sswat/utils` directory. Here is a small walkthrough for each utility provided.

8.1 `aggregate.sh`

The output mode (per-CPU or aggregate) of oprofile has to be setup before measurements. If aggregate results are obtained there is no way to generate per-CPU graphs for the same experiment, unless it is measured again with the per-CPU option.

In order to add more flexibility to the available graphs that can be created from oprofile output the `aggregate.sh` script can “merge” the per-CPU results into aggregate style output. As a result it is possible to get both types of graphs from a single mode of oprofile operation.

In order to convert a per-CPU oprofile output file into aggregate, the `aggregate.sh` script has to be provided with the name of the target test directory.

```
$ ./aggregate.sh example_case
```

This will create a file names `oprofile_results_example_case_AGGREGATE` in the test directory specified. This is done in order to avoid overwriting the original per-CPU oprofile output file. The original `oprofile_results_example_case` file has to be backed-up and the above file renamed for the graphing scripts to function properly.

8.2 `name_extractor.cpp` and oprofile filtering.

As mentioned in section 6. Visualization, graphs contain elements that represent kernel subsystems. In order to extract function names from the kernel sources the name_extractor.cpp parser is provided.

Though there are cases where this approach might be redundant, like kernel modules with distinct names or using the nm command on available .ko files, there are other cases where it is useful to use the name_exatractor.

This parser extracts kernel function names from source files that have been written using the standard kernel function definition style [`“type function_name (args) \n {“`].

A sample command line is the following:

```
$ ./fname_extract list_source_file output_file
```

> `list_source_file`: is a file that contains the paths to the source files to be processed. It can be generated easily using standard unix utilities, like `find`.
>
> `output_file`: is the name of the resulting filter.

Notice, that any special characters (“*”, “\”) and empty lines picked up must be eliminated because they introduce errors in the filtering process.

To add the new filter in the filtering process of the graphs copy the `output_file` in the `sswat/graphing/filters` directory. The `multifilter.sh` script has to be edited too:

```
filter[0]="mm_filter"
filter[1]="block_filter"
filter[2]="xfs_filter2"
filter[3]="radix_filter"
filter[4]="rb_filter"
       .
       .
       .
filter[N]=”new_filter”
```

Oprofile results will now be filtered for the function names contained in the new filter added and if the resulting percentage is above the threshold set (typically 1%), they will appear in the graphs generated.

8.3 `merger.sh`

By using `autograph.sh` it is possible to create only single stacked bar graphs, in contrast to `gnuplot_gen.sh` where multiple graphs can be automatically merged.

The `merger.sh` utility provided in the `sswat/graphing/merger` directory offers an automated way to create a graph consisting of multiple tests. A user editable area is located at the head of the script which requires the following information:

```
test_names=( "example_case" "example_case0" )
top_title="Example graphs"
sub_titles=( "Example case simple" "Example case0" )
merge_file="example_output.jgr"
```

> `test_names`: the names of the tests to be graphed. Multiple tests names can be provided. These tests have to be under the same `working_directory` path and have the same `application` name, which are fields provided in the `config` file.
>
> `top_title`: the title appearing above the graphs.
>
> `sub_titles`: the per-graph titles appearing above each graph. The subtitles have to be provided in the same order as the test names.
>
> `merge_file`: the name of the .jgr file to be used as the output file.

The `merger.sh` script update the mount points in the `config` file automatically and call `autograph.sh` for each test name provided. The resulting graph will be a merged graph which features all tests selected. After the `merger.sh` is configured, simply run:

```
      $ ./merger.sh
```

8.4 `oprof_comparator.sh`

This feature compares an arbitrary number of oprofile output files and outputs the union of the per-test functions.

There is a used editable part on the head of the script with the following fields:

```
#USER
output_file="indexer_comparison"
test_names=( bladefs_1cache_24GB_modIndexer bladefs_4caches_24GB_indexerMod )
function_threshold=0.1
#/USER
```

> `output_file`: name of the output file that will contain the results after the processing.
>
> `test_names`: names of the tests to be processed

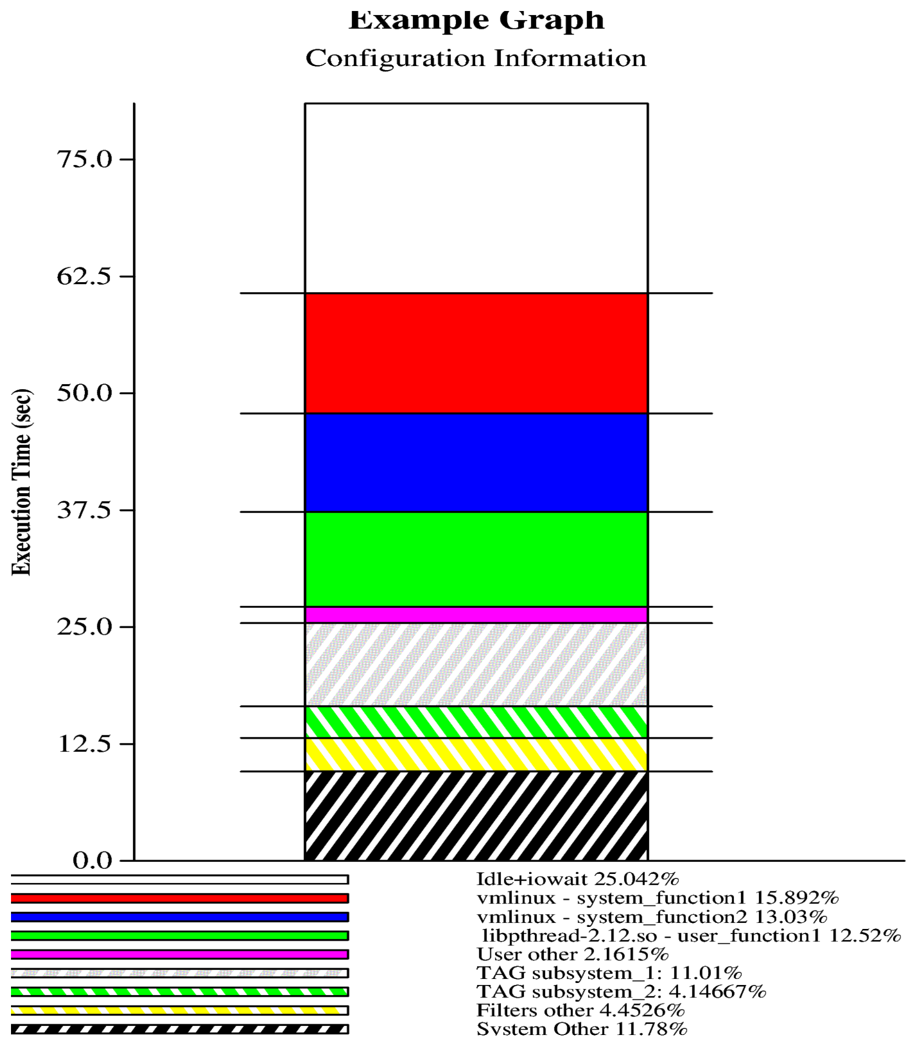
`function_threshold`: minimum percentage of the functions that will appear in the comparison output. Functions below that percentage are going to be excluded.

Functions not appearing in the oprofile output of a test or functions having percentages bellow the `function_threshold` specified have 0 value. If a specific function has 0 value in all tests it will be excluded from the final output file.

Here is a sample output of the script:

```
                    bladefs_1cache_24GB_modIndexer      bladefs_4caches_24GB_indexerMod
cbuf_writeback            5.6072                                 16.9712
copy_user_generic_string  1.4399                                 1.3817
clear_page_c              0.5448                                 0.4900
createNvdRequest          0.4614                                 0.2458
complete                  0.3354                                 0.2276
_cond_resched             0.1272                                 0
addElemToNvdRequest       0.1103                                 0.1101
```

As in the previous scripts, `oprof_comparator.sh` reads the working directory and application name information from the respective fields in the `config` file.



[Fig.1 example graph]