DCRAB

DIPC Control and Report Application in Bash

User Documentation

Document Revision 1.0
17 April 2018

DCRAB v2.0

CC-Staff
Contact: http://dipc.ehu.es/cc/computing_resources/staff.html
High Performance Computing
Donostia International Physics Center, Computing Center
Donostia - San Sebastin

Abstract

DCRAB is a tool to monitorize resource utilization in HPC environments. It works side-by-side with the job scheduler to collect runtime information about the processes generated in compute nodes.

Excluding a few cases, the data DCRAB collects is taken from the processes which the job has started, not from the entire node. The tool is able to collect the information listed below:

- CPU used
- Memory usage
- Infiniband statistics (of the entire node)
- Processes IO statistics
- NFS usage (of the entire node)
- Disk IO statistics

Contents

Installation

DCRAB has no installation at all, you have to take the lastest version tarball available in Github (currently v2.0) and expand it in a convenient location in your system:

```
tar xzvf ./DCRAB-2.0.tar.gz
```

This will create a directory called `DCRAB-2.0` with subdirectories `/src`, which contains the source files of the tool, and `/examples`, where are stored DCRAB's generated report examples of each version since it was created.

How to use DCRAB

Data Collection

DCRAB has two operation modes: normal report operation, which may be used by all users, and internal report operation, which is more focused to sysadmins. The second operation mode is going to be explained in **??** section, so let's introduce here the normal report operation which will be the commonly used one. The tool is straightforward to use, you have to add the following lines into your submission script:

```
export DCRAB_PATH=/PATH_TO_DCRAB/src/
export PATH=$PATH:$DCRAB_PATH
dcrab start

###################################
#    BLOCK OF CODE TO MONITOR     #
###################################

dcrab finish
```

Note that the variable `DCRAB_PATH` must be declared to run DCRAB, so do not omit it.

DCRAB will start a process in each compute node where the script runs, and will monitorize the processes started by it. The code will run as normal, and meanwhile DCRAB will generate a directory report called `dcrab_report_<jobid>` where `jobid` is the job number assigned by the scheduler. This reporting directory is generated in the same folder where the job was submitted.

Inside this reporting directory, DCRAB will create the reporting file called `dcrab_report.html` (which will be named as "the reporting file" is this document from now on), which is the main purpose of this monitorization, and were you will find statistics and plots to analyze visually the information collected. This report is continuous change (every 10 seconds) because the information collected is stored at the time it is taken. So, you could open it with a browser at the start of the job's execution and refresh to see what is going on with the job.

The reporting file is completely modular so you could copy only it and move to any location, because it does not need any other file to visualize or open it in a browser. Every plot and image is embedded in the report.

Furthermore, inside this reporting folder you will find some subdirectories, which normally are not relevant at all for the user, because they are generated to guarantee the correct behaviour of the tool. One will be `/data`, which contains for each computing node the files in charge of manage the processes asociated with the job. Another folder called `/auxFile`, which contains auxiliary files needed for the comunication between compute nodes, and `/log`, which contains the output generated by DCRAB process in each node and DCRAB's main process output also in a file called `dcrab.log` (used mostly for troubleshooting).

Execution Customization

On DCRAB version 2.0 is not included yet any configuration file to customize the execution, but it is included in the roadmap. However, there are some variables that could be changed inside the code until the configuration file is not released:

- `DCRAB_COLLECT_TIME`. This variable, inside `src/scripts/dcrab_config.sh` in `dcrab_init_variables()` function, configures the time between each data collection. By default it is set to 10 seconds.

- `DCRAB_NFS_MOUNT_PATH`. This variable, inside `src/scripts/dcrab_node_monitoring_functions.sh` in `dcrab_node_monitor_init_variables()` function, configures the path of the NFS filesystem to be monitorized. By defaut, its value is `/scratch`.

Crashed jobs

In the case where the job execution fails, as the report is continuosly upgrading every 10 seconds, it will ensure to have a report until the crash, which may contain relevant information about it.

On the other hand, there a couple of exceptions thrown by DCRAB when some situations occur:

- When some of the compute nodes try to write so many times in the reporting file with no success
- When the reporting directory has been deleted or moved

In this cases, a file called `DCRAB_ERROR_<computeNodeHostname>_<jobid>` will be generated, per each compute node involved in the calculation, in the same folder as the reporting folder: where the job was submitted. The `computeNodeHostname` refers to the hostname of the compute node that throws the error and `jobid` is the job number assigned by the scheduler to the job.

Design and Implementation

The main idea of DCRAB was to create a tool easy to use for the users. Notwithstanding that DCRAB is upgrading an HTML file continuosly, the runtime penalty is superfluous.

The continue upgrading of the report formed the scheleton of DCRAB's structure. The tool is modular relating to the data collection but no with the report upgrading. This upgrade is a critical section of the project because all the compute nodes need to write there at the same time, so there is a race condition we had to solve. In early versions we resolved the problem with a lock, to make the writing atomic, but there was a problem with the delay of parallel filesystems when the reporting file become bigger. The error occurs when a compute node that is waiting to write into the report some data is lost, because there is a little delay until the file is upgraded or refreshed with the new information for other nodes. We made a mark at the first line of the report to serialize the writes and solve the problem.

In its current implementation the tool makes a ssh from the master node to all other nodes. This connection starts a process in background that monitorizes the processes started by the job and collects information about them every 10 seconds (by default). The tool take advantage of the scheduler to know which processes are related to that job.

Statistics Collected

DCRAB collects different statistics and information which may be usefull for the user every 10 seconds by default. Here is a better explanation of the statistics:

- **CPU used**. The application reports the CPU usage for all the processes started by the job. This data is collected with `ps` command so it is a snapshot of the process at a concrete time. However, to have only a view of the main processes in the chart there is a threshold value defined to avoid trivial processes. This information is very useful for applications that use OpenMP.

- **Memory usage**. This value is obtained with the processes' `/proc/<pid>/status` file where `pid` is the PID of the process. The tool collect information about Virtual Memory (displayed as VmSize), Resident Memory and Max Resident Memory (displayed as VmRSS). Also is displayed the memory requested by the user for that job in the scheduler and the total memory available in the node.

  If there are more than one node involved in the execution a bigger pie chart is generated to display the amount of memory used for the scheduler (with one node calculation this value is the same as the usage in that node so there is no reason to generate that plot). With this information the users could revise the amount of memory requested into a more real value one to not waste resources, which may also help schedulers' algorithms such as Backfill of Moab.

- **Infiniband statistics (of the entire node)**. Number of packets and MB of data transfered and received over Infiniband of the entire node (this values can not be collected per processes). This information can be used to improve or revise certain parts of the code, reducing the amount of data transferred over the network, comparing this code section with high network activity levels. This values are taken from the counters available in `/sys/class/infiniband/mlx5_0/ports/1/counters`.

- **Processes IO statistics**. The I/O (Input/Output) made by the processes (no matters the filesystem type). This information can be useful to see if a process is writing more expected and is being a bottleneck in the program. This data is collected from `/proc/<pid>/io` file where `pid` is the PID of the process.

- **NFS usage (of the entire node)**. The I/O (Input/Output) made by the processes on the NFS fylesystem determined by `DCRAB_NFS_MOUNT_PATH` variable. This value, as the Infiniband value, can not be measured by process and must be a collected from the entire node statistics. The information is collected from `mountstats` command.
- **Disk IO statistics (of the entire node)**. The I/O (Input/Output) made by the processes on the local disks. This data is obtained from `/proc/diskstats` file so this statistic if related to the entire node.

Code Structure

This are the scripts that compose DCRAB without the internal report operation scripts, which are dcrab_internal_report_generation.sh and dcrab_internal_report_functions.sh.

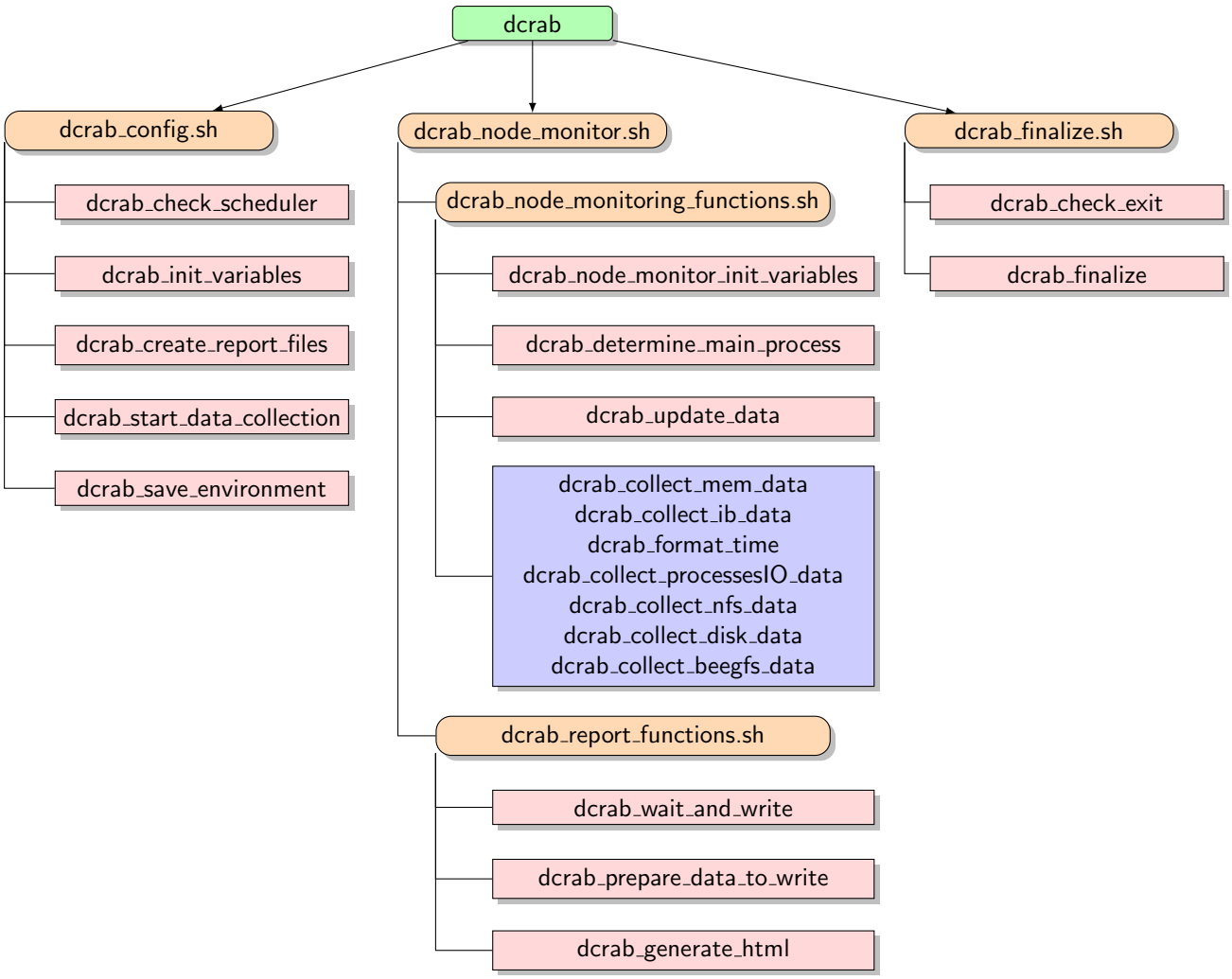

Figure 3.1: DCRAB's code structure

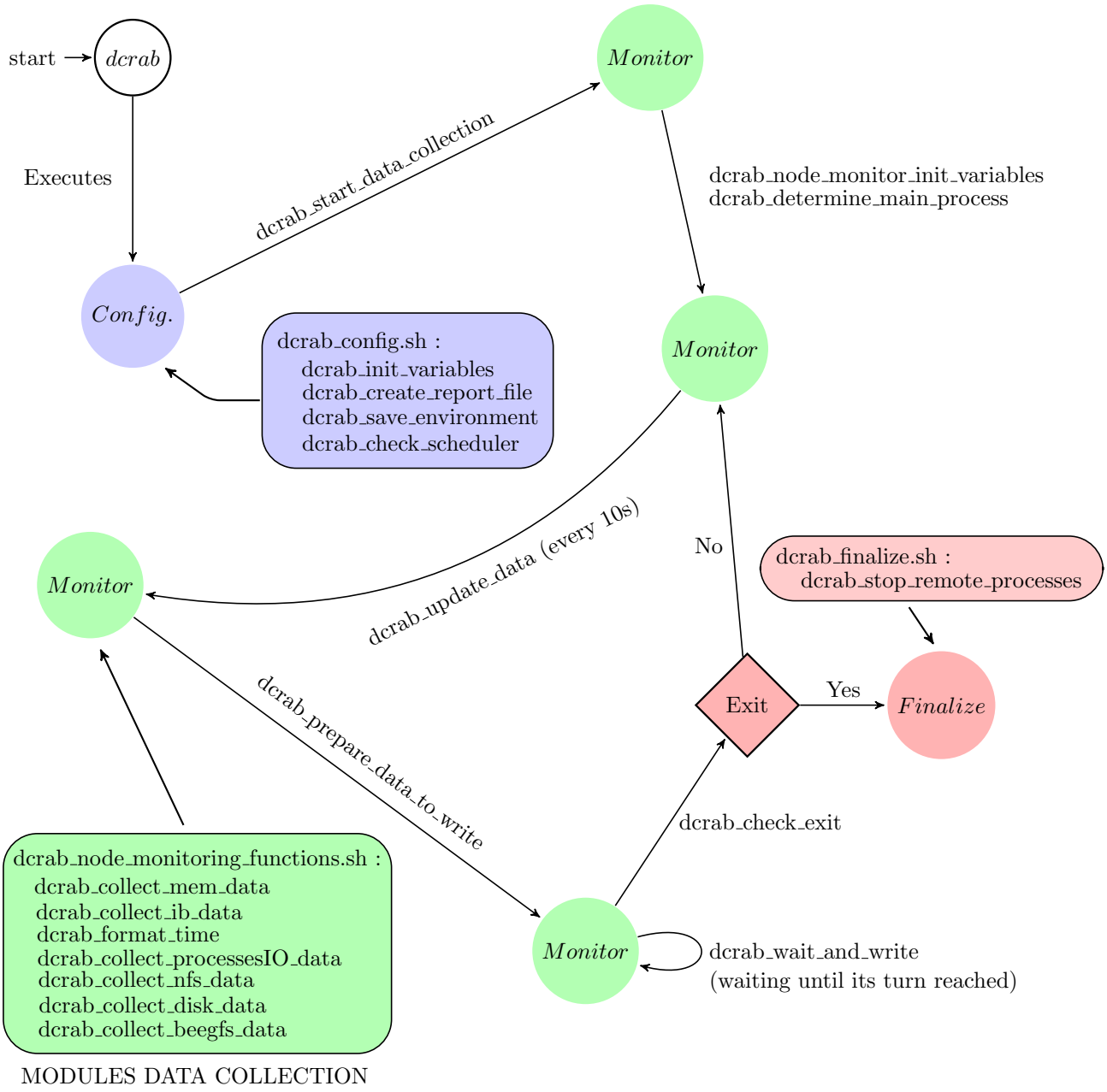

Figure 3.2: DCRAB's state machine

Above, in the figure 3.2, is presented a bit modificated state machine which may clarify the states where DCRAB passed through. Each node represents the main steps made by the tool and each edge an operation to change between states. There are boxes attached to some states to point the functions executed on them.

- The first node, which is named as `dcrab`, represent the script executed with `dcrab start` command.
- The second node, named as `Config.`, represents the configuration made by the tool where different functions are executed (attached box in blue color). It starts in each node the data collection.

- The next node `Monitor` is the first of four states that represent the process started in background on the compute nodes. This first node initialize some variables to execute correctly the module's data collection and creates the minimal environment to work. This means that it will find the main process of the job which is and will be the parent of all the processes started by the job on that compute node. The tool take advantage of the scheduler used to identify that main process.
- The next `Monitor` node represents the moment where the update function (which is made every 10 seconds by default) will be executed and goes to the next step.
- On this fiveth node the data of the different modules will be collected. In its box attached are written the names of the functions in charge of it. It prepares the data to go to the next step.
- In this last `Monitor` state, where the data will be collected and prepared to write, a wait will be done until the write turn of the node reaches. Normally the nodes do not wait more than a second to write into the report. It checks the exit of the tools in some scenarios as described in the section 2.3. If the conditions are satisfied it goes to the final state and if not it returns to the second `Monitor` state where another data collection will be done in the next 10 seconds.
- The last node, called `Finalize`, is reached when some conditions are satisfied as described in 2.3 section. Here DCRAB will finish and is going to kill all DCRAB instances inside the compute nodes (those processes launched in background).

Creating a new module

The process to create a new module is straightforward. It could be divided into two main steps, the first one the step where you must make the module to collect wanted data, and the second one modify the html report to display the information collected. The second step is not a must so you can decide not to do and store the collected data into a file. You need to add a few lines in `dcrab_node_monitoring_functions.sh` and `dcrab_report_functions.sh` for the first part, and other lines into `dcrab_report_functions.sh` script to configure the display.

The first thing to think about, in case you want to plot the collected data, is the type of chart you are going to use visualize it. DCRAB is going to upgrade the report file continuously by adding new data to the charts every loop, so you need to know how this data is going to be inserted in the report. The charts we used are made with the Google Visualization API so take a look to the documentation to choose one of them.

Once you have choosen one type of chart, you have to understand the way you could insert data into them. We normally store the information to insert in variables named `DCRAB_MODULENAME_DATA`.

Below are presented the steps to insert a new module and the best way to name variables and functions in charge of it, however, the names are only a recommendation to preserve the coherence in the code.

Step 1: Data collection

Changes in `dcrab_node_monitoring_functions.sh`:

- In the function `dcrab_node_monitor_init_variables()`: the variables needed for your module must be added where the `ADD NEW MODULES HERE` line is, that is to say, at the end of other variable declarations. We have used the same naming: all the variables starts with `DCRAB_` . A variable `DCRAB_NEWMODULENAME_DATA` should be created, which will be used to insert the data of the new module into the report file.
- Create a function, which should name as `dcrab_collect_newModuleName_data()`, after the last module function defined, where the line `ADD NEW MODULE'S FUNCTION HERE` stands. Collect the data and fill up the variable `DCRAB_NEWMODULENAME_DATA` must be the purpose of this function. Remember to generate properly the data on `DCRAB_NEWMODULENAME_DATA` variable. For example, in most of the cases we fill up the variable with a new point of the charts, which is translated to generate a string with the information of that point using the data collected (you could understand better looking at the examples in `/examples/normalReport/`).
- In the function `dcrab_determine_main_process()` you have to add at the beginning, where the string `ADD INITIALIZATION OF NEW MODULE VARIABLE HERE` is, the initialization of the `DCRAB_NEWMODULENAME_DATA` variable if needed. After that, at the bottom of the same function, add where the string `ADD THE CALL TO THE NEW FUNCTION HERE` stands, you will need to add the call to the function `dcrab_collect_newModuleName_data()` created.
- To finish with this file, you must repeat the previous step inside the `dcrab_update_data()` function as the same way you did.

Changes in `dcrab_report_functions.sh`:

-

Step 2: Display

Internal Report Operation