Documentation Framework DDM

Nicholas Ricci

25/09/2014

Indice

Capitolo 1

Introduction

This is a framework for **D**ata **D**istribution **M**anagement. Which this framework you can put your algorithms inside and you can test them with other algorithms with coherent input data.

In this document I'll explain how to configure this framework and what to do to edit your algorithms to make sure your algorithms works fine inside of it.

Before start you must be sure that your algorithms contain a makefile. If you haven't, do it!

Capitolo 2

Fix your algorithm

Before you can start to use framework you must fix your algorithm. Inside the root framework there's a folder called *C libraries for this framework* , within it there are two files: *DDM_input_output.c* and *DDM_input_output.h* .

You need to copy this two files into your sources folder. Remember to edit your makefile to compile also this library and to update your clean target.

Now you need to open your main file and include this library (for example):

```
#include "DDM_input_output.h"
```

Another thing you must do is to declare a new *DDM_Input* and a temporaneous bitmatrix variable, if you need to do an AND operation between different dimension:

```
//temporaneous result matrix, if your algorithm need an and
//operation between results of single dimensions
bitmatrix temp;
DDM_Timer *ddm_input;
```

Now you can get the number of extents, updates, subscriptions, alfa, dimensions and list of subscriptions, list of updates:

```
//Initialize variable of DDM
ddm_input = DDM_Initialize_Input(argc, argv);
//Check if the initialization of input was succesfully
if (ddm_input == NULL)
    exit(-1);
uint64_t extents = DDM_Get_Extents(*ddm_input);
uint64_t dimension = DDM_Get_Dimension(*ddm_input);
uint64_t updates = DDM_Get_Updates(*ddm_input);
uint64_t subscriptions = DDM_Get_Subscriptions(*ddm_input);
float alfa = DDM_Get_Alfa(*ddm_input);
DDM_Extent *list_updates = DDM_Get_Updates_List(*ddm_input);
DDM_Extent *list_subscriptions =
DDM_Get_Subscriptions_List(*ddm_input);

//create temporaneous matrix
bitmatrix_init(&temp, updates, subscriptions);
```

After you've gotten parameters it's time to start the timer, execute the matching algorithm and at the end stop the timer:

```
DDM_Start_Timer(ddm_input);
for (k = 0; k < dimensions; ++k){
    if (k > 0){
        //each time execute different dimension
        //reset the temp matrix
        bitmatrix_reset(temp, updates, subscriptions, zero);

        //Execute Algorithm Here **

        //Intersect temp matrix and ddm_input->result_mat and store
        //result in ddm_input->result_mat
        bitmatrix_and(ddm_input->result_mat, temp, updates,
        subscriptions);
    }else{
        //Execute algorithm and store result into
        //ddm_input->result_mat **
    }
}
DDM_Stop_Timer(ddm_input);
```

At the end you need to write result into a file then:

```
//if you want see the number of matches
printf("\nnmatches: %"PRIu64"\n", bitmatrix_count_ones(
ddm_input->result_mat, updates, subscriptions));

//Write result
DDM_Write_Result(ddm_input);
```

After you've fix your algorithm you can put it inside the framework and go to the next chapter!

Capitolo 3

How to use launcher

The script in bash inside of the root, launcher.sh, is a launcher to execute the framework. To execute the script you can open your terminal and digit this:

```
./launcher.sh <command>
```

Below will be indicated commands in order to execute for a proper functioning of the framework:

1. *–help* : will be shown what the framework can do and which commands can be run;
2. *configure* : will run an interactive part in which the user must configure the following parameters:
   - *START_EXTENTS* : indicating the initial number of extents by which the framework will execute the test;
   - *MAX_EXTENTS* : indicating the maximum number of extents to which the framework will come to run tests;
   - *STEP_SIZE* : indicating the number by which the framework will increase the number of initial extents until reaches the maximum;
   - *DIMENSION* : indicating the number of dimensions with which the framework will execute the algorithms;
   - *CORES* : in the case of parallelization in OpenMP this parameter must be set with the maximum number of cores that the user wants to use as the maximum number of cores. The initial number of cores is set to 2 by default;
   - *ALFAS* : is an array that indicates the degree of overlap of subscription extents and updates extents. You can insert multiple elements;
   - *ALFAS_PAR* : is an array similar to ALFAS, the only difference is that this is to indicate the values to use with algorithms that use OpenMP;

- *RUN* : indicating the number of executions that you want to perform for each algorithm.

This command will create a file in the root of the framework called **configure.sh** ;

3. *build* : will be compiled all algorithms in the folder *Algorithms* and *utils* where there are an avarager, a DDMInstanceMaker, a Bitmatrix-Comparator and an alfa_creator programs. At the end of the process there will be two text files in the root of each algorithm:

   - DDM_Sequential: in this file is necessary to write all names of sequential version of executables. One for each line;
   - DDM_Parallel: in this file is necessary to write all names of parallel version of executables. One for each line.

   Enter names of executables is a critical step, and even more fundamental is to insert only one name on each line;

4. *run* :

   - *alfa* : if you run this kind of test, the alfa test, you can reach some goals but isn't very accurate beacause values are very random and not created with some criteria, update and subscription extents are the same number:
     - *Results* : in this folder will be available all the runs performed with execution time, memory usage or/and bitmatrix file for distance calculation;

     You can use more precise tests, read below;
   - *<every elements inside TestsInstances>* : This tests are more accurate then alfa test. Values are always random but with more clauses.
   - plus options with run command:
     - without any option: the result is execution time in seconds;
     - mem: the result is memory usage in a .mem file. Memory usage in MB;
     - dist: the result is the distance from optimal solution;
     - all: the result is the all three options above in sequential.

This commands are listed in order of their executions.
There are other useful commands to clean framework:

- *cleanutils* : delete all objects files inside *utils* folder;
- *cleantestsinstances* : delete *TestsInstances* folder;

- *cleanalgorithms* : for each algorithm executes command *make clean* , then is foundamental define a clean target in your makefile;
- *cleanresults* : will delete *Results* and *Graphs* folders with all their files;
- *clean* : executes all previous clean commands.

There are other useful commands to create tests instances:

- *DDMInstanceMaker* : with this you can create a test instance;
- *DDMDefaultsTests* : with this you can create a set of default tests instances.
- *plotresult* : with this you can transform your .txt, .mem, .bin files into a graphics more readable for human in *Graphs* folder.