MRSG - MapReduce over SimGrid

Basic Usage
Version 1.1

July 27, 2012

Contents

0.1 Introduction

MRSG - MapReduce over SimGrid is a deterministic simulator, implemented over SimGrid, which provides to user the capability to simulate a Mapreduce environment. MRSG simulates the main features of the Mapreduce model, such as data distribution, map and reduce phases, speculative tasks, data locality and data skew.

Also, it provides some benefits, such as, no need of great number of resources to make tests, the ease of change the algorithms, and more.

0.2 Installation

In this chapter, will be presented the simulator dependencies and the process of installation.

0.2.1 Dependencies

The only dependency that the simulator have is the Simgrid. The download page of the Simgrid can be accessed at http://simgrid.gforge.inria.fr/download.php. The version 3.6.2 is the most recommended, but it should work at any stable version of Simgrid.

If you have trouble installing Simgrid, at http://simgrid.gforge.inria.fr/simgrid/3.6.2 /doc/installSimgrid.html you will find all the documentation that you need to solve it.

0.2.2 Installation of MRSG

The MRSG's installation process is quite simple. You can download the simulator at https://github.com/MRSG/MRSG/downloads. Then extract that in a folder and make sure that the *INSTALL_PATH* variable, which is defined inside the *Makefiles* points to your Simgrid Installation. The *Makefiles* must be at MRSG's directory and at the examples directory.

After check that, you just need to execute the `make` command at the MRSG's directory and at the examples directory, which must be found inside MRSG's directory. Now run the *hello.bin* example, which should work.

0.3 Input Files

The first step to run a simulation is to configure the environment. There are three input files. The first one is the *platform file*, which defines the characteristics of nodes (CPU power, number of cores) and the network configuration (bandwidth, latency, topology). The second one is the *deployment file*, which defines the characteristics of the environment, such as, master and worker nodes. The last one is the *configuration file*, which defines the characteristics of application, such as, number of reduces.

0.3.1 Platform File

The *platform file* is a XML file which defines the hardware configuration, the network capacity and the network topology of the environment. Code 1 shows an example of platform file.

Code 1: XML *platform file*

```
1  <?xml version='1.0'?>
2  <!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">
3  <platform version="3">
4    <AS id="AS0" routing="Full">
5
6          <host id="Host 0" power="1000000000.0" core="2" />
7          <host id="Host 1" power="1000000000.0" core="2" />
8          <host id="Host 2" power="1000000000.0" core="2" />
9                                    .
10                                   .
11                                   .
12
13         <link id="l1" bandwidth="125000000.0" latency="1e-4" />
14         <link id="l2" bandwidth="125000000.0" latency="1e-4" />
15         <link id="l3" bandwidth="125000000.0" latency="1e-4" />
16                                   .
17                                   .
18                                   .
19
20         <route src="Host 0" dst="Host 1">
21                 <link_ctn id="l1"/>
22         </route>
23         <route src="Host 0" dst="Host 2">
24                 <link_ctn id="l2"/>
25         </route>
26         <route src="Host 0" dst="Host 3">
27                 <link_ctn id="l3"/>
28         </route>
29         <route src="Host 0" dst="Host 4">
30                 <link_ctn id="l4"/>
31         </route>
32         <route src="Host 0" dst="Host 5">
33                 <link_ctn id="l5"/>
34         </route>
35                                   .
36                                   .
37                                   .
38   </AS>
39 </platform>
```

Lines 6-8, define the nodes' hardware configuration. Each node has a unique id, a computational power (in flops), and a number of cores.

Lines 13-15, define the configuration of the network links. Each one has a unique identificator (link id), a bandwidth (in B/s), and a latency (in seconds).

Lines 20-35, define the topology of the network.

0.3.2 Deployment File

The *deployment file*, which defines the function of each node, is also a XML file. To generate a deployment file, the user must know the id of the nodes in order to assign a master or worker function for each node. By definition, the Mapreduce has only one master, thus only one node in the *deployment file* should be defined as master. Code 2 show an example of *deployment file*.

Code 2: XML *deployment file*

```
1  <?xml version='1.0'?>
2  <!DOCTYPE platform SYSTEM "http://simgrid.gforge.inria.fr/simgrid.dtd">
```

```
<platform version="3">
        <process host="Host 0" function="master"/>
        <process host="Host 1" function="worker"/>
        <process host="Host 2" function="worker"/>
        <process host="Host 3" function="worker"/>
        <process host="Host 4" function="worker"/>
        <process host="Host 5" function="worker"/>
        <process host="Host 6" function="worker"/>
        <process host="Host 7" function="worker"/>
        <process host="Host 8" function="worker"/>
        <process host="Host 9" function="worker"/>
        <process host="Host 10" function="worker"/>
</platform>
```

0.3.3 Configuration File

The *configuration file*, which defines the properties of the application that will be simulated, is a .conf file, which is a text file. Code 3 shows an example of a configuration file.

Code 3: *configuration file*

```
master "Host 0"
reduces 100
chunk_size 64
input_chunks 100
dfs_replicas 3
map_slots 2
reduce_slots 2
```

The following attributes must be defined in the configuration file:

- *master* - Defines the master node. It **must be the same defined in the XML files**.
- *reduces* - Is the number of reduces of the application.
- *chunk_size* - Is the size (in MB) of the chunks.
- *input_chunks* - Is the number of chunks that will be simulated. The number of input chunks multiplied by the chunk size results in the size of the input data.
- *dfs_replicas* - Is the number of replicas of each chunk.
- *map_slots* - Is the number of map slots.
- *reduce_slots* - Is the number of reduce slots.

0.4 The MRSG API

The simulator's API, is a set of functions, written in C, which can be defined by the user. To use it, the user must include the MRSG library in a C-Program.

Actually, there are three functions that can be defined:

- The first one, is the function which sets the amount of data(in *bytes*) that will be emitted by the *map phase* to the *reduce phase*. It must receive two parameters, which are the id of the *map task* that will emit data and the id of the *reduce task* that will receive data.

- The second one, is the function that indicates the cost of a task. It receives three parameters: the phase, which can be *MAP* or *REDUCE*; the id of the task; the id of the *worker* that received the task. It must return the cost of the task, in *FLOPs*.
- The last one, is the function that defines the data distribution of the input. It's optional to define this function and the default definition is a uniform distribution. It receives four parameters: the simulator's internal matrix, which is a binary matrix that represents in which *workers* each *chunk* is located; the number of *chunks*; the number of *workers*; the number of *replicas* that a *chunk* will have.

Code 4 shows an example of a program with these three functions defined.

Code 4: *C program defining the three functions of mrsg*

```c
#include <mrsg.h>

/**
 * User function that indicates the amount of bytes
 * that a map task will emit to a reduce task.
 */
int my_map_output_function (size_t mid, size_t rid)
{
    return 6710886;
}


/**
 * User function that indicates the cost of a task.
 */
double my_task_cost_function (enum phase_e phase, size_t tid, size_t
    wid)
{
    switch (phase)
    {
        case MAP:
            return 16777216000;

        case REDUCE:
            return 67108864000;
    }
}


/**
 * User function which defines the data distribution
 */
void my_dfs_function (char** dfs_matrix, size_t chunks,
                      size_t workers, int replicas)
{
    size_t  c,w;

    for (c = 0; c < chunks; c++)
    {
        for (w = 0; w < workers; w++)
        {
            dfs_matrix[c][w] = 1;
        }
    }
}


```

```

int main (int argc, char* argv[])
{
    /* MRSG_init must be called before setting the user functions. */
    MRSG_init ();
        /* Set the data distribution function */
        MRSG_set_dfs_f (my_dfs_function);
    /* Set the task cost function. */
    MRSG_set_task_cost_f (my_task_cost_function);
    /* Set the map output function. */
    MRSG_set_map_output_f (my_map_output_function);
    /* Run the simulation. */
    MRSG_main ("platform.xml", "deploy.xml", "mrsg.conf");

    return 0;
}
```

In this simple example, the amount of data emitted by the map function is 6710886 Bytes. The costs of the map and reduce tasks worth 16777216000 FLOPs and 67108864000 FLOPs, respectively. The data distribution function is very simple and defines that each *worker* will have all *chunks*.

0.5 Contact

Nowadays, there is a list, where users can post any doubt that they have. That list can be accessed in https://groups.google.com/forum/?fromgroups#!forum/mrsg-user.