Simulating storage part of application with Simgrid

Cong Wang
Institute of High Energy Physics, Chinese Academy of Science, Beijing 100049, China

Abstract: Design of a file system simulation and visualization system, using simgrid API and visualization techniques to help users understanding and improving the file system portion of their application. The core of the simulator is the API provided by simgrid, cluefs tracks and catches the procedure of the I/O operation. Run the simulator simulating this application to generate the output visualization file, which can visualize the I/O action proportion and time series. Users can also change the parameters in the configuration file to change the parameters of the storage system such as reading and writing bandwidth, users can also adjust the storage strategy, test the performance, getting reference to be much easier to optimize the storage system. We have tested all the aspects of the simulator, the results suggest that the simulator performance can be believable.

1. Introduction

Simulation has been used for decades in various areas of computing science, such as network protocol design, microprocessor design. By comparison, current practice in storage simulation is in its infancy. So this article is trying to fulfill a simulator with Simgrid to simulate the storage part of application. There are several main concerns about simulating: simulation accuracy, simulation speed and simulation scalability. In the trace file caught by cluefs, each simulating process is described as a sequence of store events. This simulator will simulate these actions with simgrid API and users can check the performance of the storage application by the visualization part of the simulator and can also find out the suitable configuration parameters for the storage platform by simulating with different configuration files.

Simgrid is a counter-intuitive design approach. It is accepted wisdom that it must be highly specialized to a target domain and it can be both accurate, fast and scalable. It is also typical rational. To achieve scalability, users must “cut corners” and reduce accuracy in ways that are hopefully ok for the target domain, for example, when we want to simulating a cluster computing application, no need to simulate external load on the system. It is a powerful tool that can be accurate and scalable across domains.
Figure 1 shows the main components of SimGrid, and depicts some of the key concepts in its design. The top part shows the three APIs through which users can develop simulators. The MSG API allows users to describe an application as a set of concurrent processes. These processes execute user code and place MSG calls to simulate computation and communication activities. The SMPI API makes it possible to simulate unmodified MPI applications. The mechanisms for simulating the concurrent processes for these two APIs are implemented as part of SIMIX, which is a kernel (in the Operating Systems sense of the term) that provides process control, and synchronization abstractions. The set of concurrent processes synchronize on a set of condition variables. Each condition variable corresponds to a simulated activity, computation or communication, and is used to ensure that concurrent processes wait on activity completion to make progress throughout (simulated) time. The third API is SimDAG which does not use concurrent processes but instead specifies an abstract acyclic task graph of communicating computational tasks. The MSG API is the API which could be used mostly. Regardless of the API used, a simulated application consists of a set of activities which are to be executed on simulated hardware resources. Compute resources are defined in terms of compute capacities. The simulation core, the component that simulates the execution of activities on resources is called SURF. Each activity is defined by an amount of work to accomplish and a remaining amount of work. When its remaining amount of work reaches zero the activity completes, signaling the corresponding SIMIX condition variable. In our context, a file can be abstracted by its complete name, i.e., the absolute path from the mount point in the file system tree, its size, and the storage volume it is stored on. The size is the only file-related information that has to be handled by the simulation kernel. Indeed the file size allows the kernel to determine the time needed to complete a read or write operation and to manage the filling of a storage volume in a dynamic way. The operations on files that are exposed to users are: opening and closing a file, seeking into a file up to a given offset, reading or writing a certain amount of data, regardless of its meaning, from or to a file, and moving, copying, either locally or remotely, or deleting a file.

2. Fulfill of the simulator
In this section we detail how we use the proposed extension of the SimGrid toolkit to handle files and storage components. Simgrid added the necessary support in the SURF and SIMIX layers, and focused on the most used API (MSG). The following present here how it uses these API to fulfill the storage simulator.

2.1 Description of Storage components
Figure 2. storage components

It illustrates the declaration of storage components in the XML description format provided by SimGrid depicted in Figure 2. This platform comprises one machine named denise. Two disks are both attached and mounted by this machine. Firstly, it describes a storage type that corresponds to a single HDD hard drive disk whose capacity is of 500GB. Storage instances of this type will be simulated according to a linear_no_lat model whose parameters, a read bandwidth $B_{read}$ and a write bandwidth $B_{write}$. A list of files is associated to this storage instance, thanks to the content attribute, in a format given to the content_type attribute, here in a text file that follows a UNIX syntax for paths. There are two disk, one works as the HDD type and the other one works as the SSD type. It also need to describes the host denise. It mounts two disks. Disk 1 as the memory part where the host can only read the head of the file, Disk 2 as the normal disk where the host can read other part of the file, as normal, the size of the file reading from the quick disk depends on the size of the file head. Trace replay is one function of simgrid to replay some trace that has been gotten from the application. Firstly, the simulator need to transform the format of the original trace file to the files that needed by simgrid. The simulator includes a shell script configure.sh to parsing the original trace file. Configure.sh is the shell script which to produce some configuration files. It will read through the original trace file got by cluefs, and produce the configuration file needed by the simulator. When using SimGrid trace replay, it basically need your user code, a platform description, and something allowing to map your (simulated) process on your (simulated) platform. This is what deployment file is all about. So deployment file just consists of saying which process runs where and which arguments it should take as input. It can deploy as many processes as users want on the same host according to the original trace file, "start_time" is the time when the function will start. All the information are gotten by configure.sh according to the file.csv file. The simulator has one component describes the actions that each process executes. It is needed when simgrid replay the trace. The simulator will read through this file and simulate all the file action according to the action_trace file. Another component is storage_content which describes the initial state of the storage disk.

2.2 Tracing simulations
Tracing is widely used to observe and understand the behavior of parallel applications and distributed algorithms. Usually, this is done in a two-step fashion: the user instruments the application and the traces are analyzed after the end of the execution. The analysis can highlights unexpected behaviors, bottlenecks and sometimes can be used to correct distributed algorithms. The SimGrid team has instrumented the library in order to let users trace their simulations and analyze them. This part of the user manual explains how the tracing-related features can be enabled and used during the development of simulators using the SimGrid library.
The simulator can deal with the trace resource by categories. Categorized resource utilization tracing gives users the possibility to classify MSG and SimDAG tasks by category, tracing resource utilization for each of the categories. The tasks that are not classified according to a category are not traced. There are several options accepted by the tracing system of SimGrid.

2.3 Kernel part of the simulator

There are two modes of the simulator: sleep mode and action mode.

(a) Sleepmode

In the sleep mode, all the actions execute according to the trace file. It can reproduce the executing of users’ application and then users can get the output trace file and visualize it. All the actions are simulated by a sleeping task, that the task will sleep for specified time. This mode is mainly used for visualizing the executing of the application.

(b) Actionmode

In the action mode, read, creat, open, write action are simulated by specified function, other action are simulated by simsleep function. Users could change the configuration file such as the WriteBandwidth or the ReadBandwidth of the disk or other parameters in the platform.xml, and get the difference of the executing time. The simulator can also visualize the executing sequence.

3. Storage simulation use cases

The LSST is a large optical survey project funded by the National Science Foundation and the Department of Energy. It will continually image the sky, identify changes in near real time, and over a decade of operations collect tens of petabytes of data building up the deepest, widest, image of the Universe. Its data will enable a range of science goals from identification of Near Earth Asteroids to understanding the nature of Dark Energy. A survey of this scale requires significant computing resources but also a modern, high-performance, scalable, data processing and analysis system. It will do some file actions which we focus on. So what we want to do is: A: Simulate the application and visualize to make sure which kind of I/O action it does mostly. B: Change the parameter of the platform to find out whether it is useful if we put the head of the file in the SSD disk which read much faster, and how much the performance will improve.

After the simulating, we can get several files to help us know much more about the application. The log file in the configuration file folder. It includes all the executing information of the simulator. The parameters we set to the simulator and the start time, duration, process id of every action. The trace file which includes the space-time visualization information. PajeNG is a reimplementation and direct heir of the well-known Paje-visualization tool for the analysis of execution traces through trace visualization. We can use it to preprocess the trace file we got and then show the traces by the tool framesoc.

4. The accuracy and verification of the simulator

4.1 Verification of the simulator

After the executing of simulator, users can compare the executing time of each process in the simulator with the time in the original trace file, and get the time difference of each process. When users want to compare the start and end time of the original time in the trace file and the time got in the log file after executing of every process to make sure your simulator works well. They can get the file comparetime.txt which includes all the time difference of starttime, endtime and duration time. Then they can use python script to draw some picture to make it clear. For example, figure 3 4 5 are the time difference between 60_90_300_500 (Wbandwidth, Rbandwidth, Wbandwidth, Rbandwidth) platform in sleep mode and the original trace file. It shows that the difference is in millisecond and the whole executing time is more than 600s, so we can receive the difference and we can believe in the result of the simulator.
Figure 3. Start time difference

Figure 4. End time difference

Figure 5. Duration time difference
4.2 Executing time of different platform
When users want to draw pictures of the executing time of different platform. They can get the file gettime.txt which includes the platform information and the executing time.

5. In-depth study with the simulator
We want to use this simulator to find out how this application works with different platform file. As we have found that most of the fits file header are 4KB, so we put all the files to the slow and fast storage disk and attach both of them to the host. We fulfill the simulator as below: when we want to read something, the first 4096B block are read from the fast disk and others (the data) are read from the slow disk, and when we write something, all the data are written to the slow disk. With that, we can change the parameter of the bandwidth and check the executing time of different platform to find out which parameters works mostly and it can helps us to optimize our storage system. In this lsst application, when we simulate with the simulator. We found that the results are as figure 6.

![Figure 6. executing time of different platform](image1)

![Figure 7. executing time of different platform](image2)

If we read write data all from the slow disk, the result is figure 7. If we put the head of the file in the fast disk the result is figure 6, it will improves the performance but it doesn’t improve a lot, and when we change the bandwidth of the fast disk, the performance doesn’t change a lot. The cause of this perhaps is that described in figure 8.
As the figure 8 showed above, this application doesn’t include a lot of read, write action so the change of the disk parameter doesn’t affect a lot, so I will look for an application which includes much more read and write action and do some more test.

6. Conclusion

Users can use cluefs to get I/O action sequence of the application, and input all the sequence to the simulator, the simulator will simulate I/O action. The simulator has two mode, sleep mode to simulate all the action according to the time in the trace file, it will produce the log including exact executing information and will produce another trace file which is used for visualization. The visualization part can visualize all the I/O file action sequence of each process to make it clear how the application works on storage part. The second mode is action mode, it simulates all the I/O actions with a platform configuration file. In this file, users can change the storage structure and storage parameters such as Reading bandwidth of disk. And then users can get the executing time of different platform file to make sure whether it is worth to change some configuration of your real storage system. So the simulator is useful when you want to know how the storage part of you application works and when you want to know how your application will work with different disks, you can simulate it with the simulator and you will gain a lot.

References

[1] Bassat B, Levy R,Ben-AriM,Uronen PA. The Jeliot 2000 program animation system[J]. Computers & Education, 2003, 40(1): 1-15
[2] Yeh C L, Greyling J H, Cilliers C B. A framework proposal for algorithm animation systems [C]. Proceedings of the 2006 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists, (Somerset West, South Africa), South African Institute for Computer Scientists and Information Technologists, 2006: 155-163.

[3] Gogeshvili A. AVL tree visualization applet [EB/OL]. http://www.seanet.com/users/arsen/avltree.html/, 2001.

[4] Gustafson B, Kjenoli J. Binary Treesome [EB/OL]. http://www.binarytreesome.com/, 1998.

[5] Kolberg W, Marcos P, Anjos J, Miyazaki A, Geyer C, and Arantes L. “MRSG – A MapReduce Simulator over SimGrid” Parallel Computing, vol. 39, no. 4-5, pp. 233–244, Apr. 2013.

[6] Wang g, Butt A, Pandey P, and Gupta K. “A Simulation Approach to Evaluating Design Decisions in MapReduce setups” in Proc. of the 17th IEEE/ACM Intl. Symp. on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, 2009.

[6] Adrien L. Adding Storage Simulation Capacities to the SimGrid Toolkit: Concepts, Models, and API [C]. IEEE/ACM International Symposium on Cluster, Cloud & Grid Computing. Shenzhen, China, 2015.