CHAPTER 4

A STUDY ON SIMULATION CONCEPTS

4.1 INTRODUCTION

Researchers use simulators to evaluate new scheduling techniques under controllable and repeatable condition, which is impossible to reach in real cloud. Simulators are very useful as different setups and different data sets can be used to evaluate existing or proposed solutions as well as to compare their performance. In this chapter, some of the basic principles of simulation modeling are reviewed, and the simulation tool CSIM20 is introduced. Finally, the chapter gives a simple code example of a queuing system using CSIM20.

4.2 SIMULATION PRINCIPLES AND TOOLS

4.2.1 Simulation Principles

Simulation is the imitation of the operation of a real world process or system over time. Simulation involves the generation of an artificial history of the system, and the observation of that artificial history to draw inferences concerning the operating characteristics of the real system that is represented. There are quite a few application areas of the simulation, including simulation of manufacturing and material handling systems, simulation of automobile industry and transportation systems, simulation of healthcare and service systems, and even the simulation in the military field. In all these fields,
simulation is an indispensable tool to find all kinds of answers for the real world.

Simulation generally consists of three phases, namely the design of a model, model execution, and the analysis of the data obtained from the execution. In the phase of model design, one have to define a concept model according to the knowledge of the real system. Then one should consider the model execution where some mathematic languages are chosen to express the concept model. Also in this phase one should consider a proper simulation toolkit. There are many simulation toolkits, and one should decide the one most suitable for our purpose. After the model execution, some results may come up, and then we should start the final phase - the analysis. During the analysis phase, the data of simulation results are put together, maybe in some visualized way; and statistical analysis are made, which allows us to better understand the nature of the system and make further inferences. At this stage, the verification (process to make sure that the concept / mathematic model correspond precisely to the real system) and validation (the process to check whether the output of our concept model is exactly what we have expected in the real system) can also be made. The results are taken as feedback to further improve or correct the concept model.

4.2.2 Procedure of Simulation

Generally, there are some commonly followed steps in the simulation study (Banks 2003). The flow of these steps is shown in the Figure 4.1 and the brief explanation of each step is listed below.

- Problem formulation is the very first step in the simulation study, which provides a precise statement of the real problem.
• Setting of objectives and overall project plan is the preparation of the proposal, which states the goal, schedule, cost etc. of the simulation work.

• Model conceptualization defines an abstract conceptual model and mathematical relationship of the components of the real system.

• Data collection is the real system data that is required by the simulation model is collected in this step.

• Model translation is to translate the conceptual model to the operational model simulated on the computer.

• Verification is to determine previously built designed operational model performs properly.

• Validation is the comparison of the conceptual model and the real system is made to see if the conceptual model is the accurate representation of the real system.

• Experimental design is to design for each scenario the number of runs, the run length, the initial parameters of the run etc.

• Production runs and analysis is to estimate the performance of the scenarios

• More runs? - analysis from the previous production runs, to see if more runs are needed.

• Documentation and reporting - adequate documentation and reporting is clearly necessary for the simulation model reuse, and modification.
- Implementation is the documentation produced in the previous step help people to make implementation decisions for the real system.

**Figure 4.1 Procedure in Simulation Study (Banks 2003)**
To design the simulation model, the work start with analyzing the concept model of the real system, and break the whole system into a number of smaller abstract modules depending on its functionality. Finally, the work chooses a proper model to represent each of these modules. Since no rules can be followed to choose a model, sometimes the work have to use some heuristic approaches to make decisions.

### 4.2.3 Modeling Principles

A system is a collection of entities that act and interact together toward the accomplishment of some logical end. A model is a representation of an actual system. Immediately, there is a concern about the limits or boundaries of the model that supposedly represent the system. The model should be complex enough to answer the questions raised, but not too complex. In the simulation world, one of the most important concepts is the modeling. A model is actually a representation of the real-world system. Designing a model is more like an art than a technology because there can be many ways we can abstract the conceptual model from a real system. There is simply no best model, therefore any model complex enough to represent all the details of the system that are necessary for the problem under investigation is a good model. Models showing too many details, or not including factors that will alter the simulation results are not good ones.

Depending on the nature of the occurrence of the simulation event, the simulation models can be classified into three categories: namely discrete model, continuous model, and the combined model. A discrete model is a model with dependent variables that change only at distinct points in simulated time (so called event times). A continuous simulation model has dependent variables that change continuously over time (usually they can be represented by some forms of differential equations). A combined model simply consists of dependent variables that may change discretely or
continuously. The discrete model is good at modeling the system where the state of the system changes at discrete point of time, like the problem of resource management, queuing, and any problems that can be modeled by a finite state automata (FSA). The continuous model is usually used in modeling the problems, having variables that can be defined by some mathematic equation and changes continuously over time, like physical experiment, laws of nature etc (Banks 2003).

4.2.4 **Discrete-Event Simulations**

Discrete event model is a declarative model contains two primary components: states and events. It is especially suitable for mimicking the behavior of the real system whose action is considered to be the transition from one state to another. A simulation using a discrete-state model of the system is called a discrete event simulation. This is opposite of continuous-event simulations in which the state of the system takes continuous values. The continuous-state models are used in chemical simulations where the state of the system is described by the concentration of a chemical substance. In computer systems, discrete-event models are used since the state of the system is described by the number of jobs at various devices. Notice that the term “discrete” does not apply to the time values used in the simulation. A discrete event simulation may use discrete or continuous time values.

All discrete-event simulations have a common structure. Regardless of the system being modeled, the simulation will have some of the components described below. If a general-purpose language is used, all components have to be developed by the analyst. A simulation language provides some of the components and leaves others for the analyst. The components are as follows:
• Event Scheduler keeps a linked list of events waiting to happen. The scheduler allows the events to be manipulated in various ways like (a) Schedule event X at time T.(b) Hold event X for a time interval dt.(c) Cancel a previously scheduled event X.(d) Hold event X indefinitely (until it is scheduled by another event).(e) Schedule an indefinitely held event. Event scheduler is one of the most frequently executed components of the simulation. It is executed before every event, and it may be called several times during one event to schedule other new events.

• Simulation Clock and a Time-advancing Mechanism is a global variable representing simulated time. The scheduler is responsible for advancing this time.

• System State Variables are global variables that describe the state of the system. This is a global variable that is distinct from local variables such as CPU time required for a job, which would be stored in the data structure representing the job.

• Event Routines update the system state variables and schedule other events. For example, in simulating a CPU scheduling mechanism, one might need routines to handle the three events of job arrivals, job scheduling, and job departure.

• Input Routines get the model parameters, such as mean CPU demand per job, from the user. It is better to ask for all input at the beginning of a simulation and then free the user, since simulations generally take a long time to complete. The input routines typically allow a parameter to be varied in a specified manner.
Figure 4.2 Procedure in Discrete Event Simulation

- Report Generator are the output routines executed at the end of the simulation. They calculate the final result and print in a specified format.

- Initialization Routines set the initial state of the system state variables and initialize various random-number generation streams.

- Trace Routines print out intermediate variables as the simulation proceeds. They help debug the simulation program.
Dynamic Memory Management allows periodic garbage collection as new entities are generated and old ones are destroyed. Most simulation languages and many general-purpose languages provide this automatically.

Main Program brings all the routines together that calls input routines, initializes the simulation, executes various iterations, and finally, calls the output routines.

### 4.3 SIMULATION TOOLS

Openstack, Eucalyptus and Opennebulla are open source Cloud management infrastructures to set up small and medium scale private and public cloud. Users primarily deploy it as an IaaS solutions to create private and public cloud. However, Cloud management infrastructures are complex software stack solutions that require a deep understanding of different and heterogeneous aspects, spanning from optimization techniques to virtualization technologies and networking architectures. In addition, using real system on a test bed is costly affair. Furthermore, Openstack, Eucalyptus and Opennebulla, offer extremely primitive solutions as management infrastructures, and lack of more complex mechanisms to enable dynamic service provisioning and rescaling (Corradi et al 2014). These limitations turned the research work attention to choose simulation concepts. Using simulation concepts th above mentioned anamolies can easily be overcome.

Several simulation approaches for cloud systems have been proposed. Each differs in whether they focus on special applications or allow simulation of cloud systems in common. For an example, the simulation framework MRPerf (Wang 2009) instruments the discrete event network simulator NS-3 (2014) for studying performance and dependability of MapReduce (Dean & Ghemawat 2008). The framework models network,
node, and disk behavior in high aspect and thus allows evaluating the impact of network topology choices and network / node failures, but is limited to applications that use MapReduce.

Similarly, the cloudsim toolkit is a discrete event simulation engine provides simple implementation of common entities such as computational resources or users and also allows simulating virtual machines, VM scheduling, simple jobs, network topology, data storage and other useful functionality (Buyya et al 2009). However, provided implementations are too simple and it is necessary to extend these entities for more complex requirements. Further, it needs one to have in depth knowledge on cloudsim usage classes and java. In contrast to all the above, CSIM is a simulation model building toolkit, used by C/C++ programmers to implement process oriented, discrete event simulation model. These models mimic the operation of complex systems, to give modelers insight into the dynamic behavior of these systems. Because CSIM models are C/C++ programs, there are virtually no limits to the level of details, degree of complexity and size of the simulation models. CSIM processes are operated in an asynchronous parallel manner, mimicking the behavior of multiple entities which are active at the same time (Edwards & Sankar 1992; Schwetman 2001).

The CSIM20 simulation engine is very compact and efficient, and can be embedded into any code written in C/C++, so the users do not have to learn a particular programming language for CSIM20. Like C++, the simulation engine itself is object oriented, thus it provides a convenient and easy-to-use interfaces. It provides a library of classes, methods, and functions, which enable us to implement general simulation models. By inheriting the base class of the simulation engine, the user can easily modify and extend the behavior of the basic models to simplify the realization of more complicated systems. A CSIM program models a system as a collection of CSIM
processes which interact with each other by using the CSIM structures. The purpose of modeling a system is to produce estimates of time and performance. The model maintains simulated time, so that it can yield insight into the dynamic behavior of the modeled system. In CSIM, entities are represented by processes, and queues are represented by facilities and other simulated resources. In these models, the complete system is represented by a collection of simulated resources and a collection of processes that compete for use of these resources. A major benefit of using a standard programming language to implement simulation models is that these models can be combined easily with other software components.

4.3.1 Simulation Components (Classes) in CSIM20

There are a number of simulation components (classes) provided by CSIM20. The work now briefly introduce the most important ones.

4.3.1.1 Processes

A CSIM process is an independent thread (lightweight process), which can mimic certain activities of an entity; several processes can appear to be executing simultaneously, although they are actually executing sequentially on the processor. Just like a real process, a CSIM process can be in the states of ready, active, holding (allowing simulation time to pass), and waiting (for some event to happen). Their transitions are controlled by certain methods provided by the process class. Process has a priority for execution; different processes may have different priorities.

4.3.1.2 Facilities

A CSIM facility is a resource that is typically "used" by processes in the model; Usually a facility consists of a server and a queue used for the
processes waiting to be served by the server. A multi-server facility has a single queue for several servers. During the time of heavy load, the processes are queued up for access to a server. Processes with higher priorities are queued ahead of the process with lower priority. In CSIM, it is easy to model a CPU and multi-core CPU as a facility and facility_ms respectively. A facility is a server with a queue for the waiting processes. In operation, an arriving process reserves a facility. If the server at the facility is not busy, it is assigned to do the requesting process. If the server is busy, the arriving process is placed in the queue and it is suspended. Normally, when the process is given to the server, it does a hold (service time) and releases the server at the facility. When this happens the queue is checked; there is a waiting process, and so on. Now a multi-server operates in the same way only when there are multiple servers. The service time is drawn from a probability distribution function exponential (service Time). All the CSIM resources have “inspector functions, which let one to get properties and statistics from the resources. For an example, the mean response time of the server [i] is given by server[i]->resp (). Similarly, the statistics and counters for a resource is cleared by calling the reset () method.

4.3.1.3 Storages

Storage is a resource that can be allocated to the processes. It consists of a counter (amount of storage) and a queue used for queuing the processes waiting for storages. Storages can be set to be synchronous, which means several of them can be allocated in the same clock cycle. When the storage unit is insufficient to allocate to any process, the process will simply wait in line until other processes release the storage unit that is previously allocated.
4.3.1.4 Events

An event is used to synchronize the behavior of different processes, and it has two states: occurred or not occurred. A process can be suspended when waiting for another occurred event and it also can be resumed when that event occurs. The state of an event can be and usually is set by some other processes.

4.3.1.5 Mailboxes

A mailbox is used to exchange information between processes. Any process can send a message to or receive a message from a mailbox. A mailbox maintains two FIFO queue, one for incoming messages, and the other for waiting processes. When a message arrives and there is no process waiting for it, the message will go to the message queue waiting to be picked up. On the other hand, if a process execute a receive action while there is no message in the message queue or the mailbox is empty, the process will wait until there is some message coming in.

4.3.1.6 Tables

A table is an object that is used to collect individual data values and to report its statistical properties generated from that table. The properties of the report include mean, variance (and standard deviation), standard deviation, coefficient of variation, minimum, maximum, and the number of observations, etc. The report also support features like histogram, which reports the relative frequency of specified ranges of values, confidence intervals with which we can estimate the accuracy of some values collected, and moving window (which determine the sample size) etc.
4.3.1.7 Qtables

A Qtable is pretty much the same as a table described above, except that it is used solely to collect integer values (e.g. number of clients, queue lengths) and to report their statistical properties.

4.3.1.8 Meters

A Meter is used to measure the flow rate of entities passing a certain point in the system module and to keep track of the times between successive passages.

4.3.1.9 Boxes

A Box is used to collect data of time spent in a specified entity, and the number of processes inside the box. With these basic classes, the simulation modeling and the result data collection become an easy job. Users only need to focus their attention on the model itself rather than many tedious details.

4.3.1.10 Random number

Simulations are random number driven. In the sense, random numbers are used for inter arrival times, service times, allocation amounts, and routing probabilities. For each application of random numbers in a simulation, a distribution must be chosen. The distribution determines the likelihood of different values occurring. The CSIM20 simulation library provides both continuous (real) and discrete (integer) random numbers series from up to 18 distributions, including uniform, beta, exponential, gamma, erlang, weibull, normal, cauchy, poisson, geometric, and binomial etc. These functions make the simulation tool really handy in designing random aspect of the simulation models. The change of seeds can be realized by the “reseed”
function, which gives us a different series of random number still following the same distribution. Reseed enables us to find more stable and accurate result independent of any particular sequence of random numbers, which makes the simulation results more convincing.

4.3.2 An Example of CSIM 20 Model of M/M/1 Queue

To see how this tool works, it is shown in the following example of CSIM20 simulation engine used in a queuing system. This program simulates a queuing system with only one server (facility). There will be 5000 customers coming in and waiting to be served. The inter-arrival time of the customer follows an exponential distribution with the mean of 2 (IAR_TM) seconds, and the length of service time also follows an exponential distribution with the mean of 1 (SRV_TM) second. The function customer() mimics the behavior of a customer, coming in, being served and leaving. At the same time, the variable tbl (of type table) records how long this customer stays in the system or the customer’s response time (the time from when he enters the system to when he leaves), the variable f (of type facility) records how much time it takes for the server to serve the customers, and qtbl (of type qhistogram, almost the same as a qtable) counts the number of the customers in the system.

// C++/CSIM Model of M/M/1 queue
#include "cpp.h" // class definitions
#define NARS 5000
#define IAR_TM 2.0
#define SRV_TM 1.0
event done("done"); // the event named done
facility f("facility"); // the facility named f
table tbl("response time"); // table of response time
qhistogram qtbl("number in system", 10l); // qhistogram of number in system
int cnt; // count of remaining processes
void customer();
extern "C" void sim(int, char **);
void sim(int argc, char *argv[]) {
    set_model_name("M/M/1 Queue");
    create("sim");
    cnt = NARS;
    for(int i = 1; i <= NARS; i++) {
        hold(expntl(IAR_TM)); // interarrival interval
        customer(); } // generate next customer
done.wait(); // Synchronization
report(); // model report
mdlstat(); } // model statistics

void customer() { // arriving customer
double t1;
create("cust");
t1 = clock; // record start time
qtbl.note_entry(); // note arrival
f.reserve(); // reserve facility
hold(expntl(SRV_TM)); // service interval
f.release(); // release facility
tbl.record(clock - t1); // record response time
qtbl.note_exit(); // note departure
if(--cnt == 0)
done.set(); } // Synchronization

After run the simulation program, one can get the statistics of the facility summary of the server (like utilizations, response time etc), a table of response time, and a table of the customer number in the system (or the queue
length because the customers are served in the FCFS order) in the form of a histogram.

4.4 CONCLUSION

This chapter studied the basic principles of system and simulation modeling procedures and techniques. Further, it studied various simulator and why CSIM 20 is most preferable for the proposed system implementation.