Discrete Event Simulation

Master 2R SL module MD

Jean-Marc Vincent and Arnaud Legrand

Laboratory ID-IMAG
MESCAL Project
Universities of Grenoble
{Jean-Marc.Vincent,Arnaud.Legrand}@imag.fr

February 2, 2007

Outline

Outline

Large Scale Distributed Systems Research

Researchers often ask questions in the broad area of distributed computing

- Which scheduling algorithm is best for this application on a given Grid?
- Which design is best for implementing a distributed database?
- Which caching strategy is best for enabling a community of users that to distributed data analysis?
- What are the properties of several resource management policies in terms of fairness and throughput?
- What is the scalability of my publish-subscribe system under various levels of failures
- . . .

Analytical or Experimental?

Analytical research

- Some develop purely analytical / mathematical models.
  - makes it possible to prove interesting theorems
  - often too simplistic to convince practitioners
  - but generally useful for understanding principles in spite of dubious applicability
- One often uncovers NP-complete problems anyway e.g., routing, partitioning, scheduling problems
- And one must run experiments

$\rightsquigarrow$ Large Scale Distributed System research is based on experiments

LSDS as a science ?

You can tell you are a scientific discipline if

- You can read a paper, easily reproduce (at least a subset of) its results, and improve
- You can tell to a grad student: "Here are the standard tools, go learn how to use them and come back in one month".
- You can give a 1-hour seminar on widely accepted tools that are the basis for doing research in the area

We are not there today...

That is why you need to get familiar with most simulation issues.

Need for standard ways to run *Large Scale Distributed System (LSDS) experiments*

Large Scale Distributed System Experiments

Real-world experiments are good
- Eminently *believable*
- Demonstrates that proposed approach can be implemented in practice

But...

Can be time-intensive Execution of “applications” for hours, days, months, ...

Can be labor-intensive Entire application needs to be built and functional (including all design / algorithms alternatives include all hooks for deployment).
Is it a good engineering practice to build many full-fledge solutions to find out which ones work best?

What experimental testbed?

My own little testbed well-behaved, controlled, stable, often not representative of real Grids.

Real platforms
- (Still) challenging for many researchers to obtain
- Not built as a computer scientist's playpen (other users may disrupt experiments, other users may find experiments disruptive)
- Platform will experience failures that may disrupt the experiments
- Platform configuration may change drastically while experiments are being conducted
- Experiments are uncontrolled and unrepeatable: even if disruption from other users is part of the experiments, it prevents back-to-back runs of competitor designs / algorithms

LSDS Experiments are Limited and Non Reproducible

$\rightsquigarrow$ Difficult to obtain statistically significant results on an appropriate testbed

And to make things worse...

Experiments are limited to the testbed

- What part of the results are due to idiosyncrasies of the testbed?
- Extrapolations are possible, but rarely convincing
- Must use a collection of testbeds...
- Still limited explorations of "what if" scenarios (what if the network were different? what if we were in 2015? what if we had a different workload?)

Difficult for others to reproduce results This is the basis for scientific advances!

Simulation

Simulation can solve many (all) of these difficulties

- No need to build a real system
- Conduct controlled / repeatable experiments
- In principle, no limits to experimental scenarios
- Possible for anybody to reproduce results
- Does not cover all settings (wireless, high-performance networks, . . .)

Definition: Simulation.

Attempting to predict aspects of the behavior of some system by creating an approximate (mathematical) model of it.

Simulation $\approx$ Imitation

Key question: Validation (correspondence between simulation and real-world)

Simulation in Computer Science

Microprocessor Design A few standard "cycle-accurate" simulators are used extensively

http://www.cs.wisc.edu/~arch/www/tools.html

Possible to reproduce simulation results

Networking Network-guys are better at standards than we are... ☺

- A few standard "packet-level" simulators NS-2, DaSSF, OMNeT++
- Well-known data-sets for network topologies
- Well-known generators of synthetic topologies
- SSF standard: http://www.ssfnet.org/
- Possible to reproduce simulation results

Large Scale Distributed Systems None of the above up until a few years ago (most people built their own "ad-hoc" solutions).

Simulation of Distributed Computing Platforms?

Simulation of parallel platforms used for decades. Most simulations have made drastic assumptions

Simplistic platform model
- Processors perform work at some fixed rate (Flops)
- Network links send data at some fixed rate
- Topology is fully connected (no communication interference) or a bus (simple communication interference)
- Communication and computation are perfectly overlappable

Simplistic application model
- All computations are CPU intensive
- Clear-cut communication and computation phases
- Application is deterministic

Straightforward simulation in most cases Just fill in a Gantt chart with a computer rather than by hand. No need for a “simulation standard”

LSDS Simulations?

Simple models are perhaps justifiable for a switched dedicated cluster running a matrix multiply application.
They are hardly justifiable for grid or peer-to-peer platforms. . .

- Complex and wide-reaching network topologies (multi-hop networks, heterogeneous bandwidths and latencies, non-negligible latencies, complex bandwidth sharing behaviors, contention with other traffic)
- Overhead of middleware
- Complex resource access/management policies
- Interference of communication and computation

Grid Experiments

Recognized as a critical area

GRID 5000 A reconfigurable grid to enable reproducibility.

Grid eXplorer (GdX) project (INRIA) Build an actual scientific instrument

- Databases of "experimental conditions"
- 1000-node cluster for simulation/emulation
- Visualization tools

What simulation technology???

What kind of results in my paper?

- Results for a few "real" platforms
- Results for many synthetic platforms

Two main questions:

1. What does a "representative" Grid look like?
2. How does one do simulation on a synthetic representative Grid?

Types of simulation

Main types of simulation used to evaluate computer systems.

Emulation An emulator program is a simulation program that runs on some exiting system to make that system appear to be something else. As the real code is run, such simulations capture many overhead and are generally very precise.

It is a common belief that emulation are not concerned with validation (correspondence between simulation and real-world). However the relationship between virtual performances (the ones measured on the virtual system) and expected real performances (the ones that one would measure on the original system) is often very unclear.

Discrete-event Simulation A discrete-event simulator is used to model a system whose global state changes as a function of time.

The state may also be affected by events that are generated externally to the simulator as well as those that are spawned within the simulator by the processing of events. The global state is appropriately updated every time some event occurs.

Outline

Terminology

State variables The variables whose values define the state of the system (length of the job queue, list of pieces of a file, amount of computation performed for each task, ...). If a simulation is stopped in the middle, it can be restarted later if and only if values of all state variables are known.

Event A change in the system state (arrival of a job, beginning of a new execution, departure of a job, failure of device, ...).

Continuous-Time vs. Discrete-Time Depends whether the system state is defined at all times or not.

Continuous-state vs. Discrete-State Depending whether state variables are continuous (time spent by a packet in a queue) or discrete (number of jobs in a queue), a model is called continuous- or discrete-state model. Many models are hybrid.

Terminology

Deterministic vs. Probabilistic It the outputs of a model can be predicted with certainty, it is a deterministic model. A probabilistic model, on the other hand, may give a different result on repetitions for the same set of input parameters.

Linear and Nonlinear Models If the output parameters are a linear function of the input parameter, the model is linear.

Open vs. Closed If the input is external to the model and is independent of it, it is called an open model (packets entering a system). In a closed model, there is no external input.

Common structure

Each discrete-event simulator will require at least some of the following components:

- an event-scheduler,
- a global time variable and a method for updating this time,
- system state variables,
- event-processing routines,
- event-generation mechanisms,
- data-recording and summarization routines

The event-scheduler

- This is the heart of a d.-e. simulator. It generally maintains a list of all pending events in their global time order (if possible). In some particular cases, it may be interesting to compute completion times of events at the latest moment.
- It is the responsibility of the scheduler to process the next event on the list by removing it from the list and dispatching the event to the appropriate event-processing routine.
- The scheduler also inserts new events into the appropriate point in the list on the basis of the time at which the event is supposed to be executed. It also may have to reschedule or cancel events.
- This is one of the most frequently executed components of the simulation. It should thus be carefully optimized.
- Updates to the global time variable are also coordinated by the scheduler.

The global time variable

The global time variable records the current simulation time. It can be updated by the scheduler using one of two approaches.

fixed-increment
: The scheduler increments the global time variable by some fixed amount. It then checks the pending events on the event set. If the scheduled execution time for any of the pending events matches the current time, all of these events are dispatched for execution.
After all of the events scheduled for the current time have been processed, the scheduler again increments the global time variable.

event-driven
: the global time jumps to the value of the next event at the head of the pending-event set. With this approach, the value of the global time changes non-uniformly, sometimes jumping by a large amount, sometimes not changing from one event to another.

Event Processing and Generation

Each event in the system will typically have its own event-processing routine to simulate what happens when that event occurs in a real system.

These routines may update the global state and may generate additional events that must be inserted into the pending-event set by the scheduler. Efficient management of the event set is thus crucial.

Event generation is generally classified with the following categories:

Execution driven somewhat similar to an emulation. The simulator executes a benchmark program and uses its result. An emulation is concerned only with producing the appropriate output from the benchmark program whereas the execution-driven simulator is also concerned with how the output is produced.

Such simulations are often considered the most accurate type since they must model all of the details to actually execute a program. It is however generally expensive both in terms of simulation time and in term of time required to develop and verify the simulator.

Event Processing and Generation

Trace driven A trace is a record of the sequence of events that occurred when the system was traced. By using a trace, the simulator does not need to all of the functionality needed by the program, as is required in an execution-driven simulation. We will present more in details advantages and drawbacks of trace-driven simulations in the next two slides.

Distribution driven similar to trace-driven simulation, except that the input events are generated by the simulator itself to follow some predefined probabilistic function (e.g., sending messages over a communication networks would be modeled by using an exponential distribution).
Since there is no assurance that any real application program actually would produce this input sequence, the simulation should be run many times with several different input sequence. The outputs can then be averaged in some appropriate fashion.

Traces: Advantages

Main advantages:

Credibility The results of trace-driven simulation are easy to sell to other members of the design team. Traces have generally much more credibility than random sequences using an assumed distribution.

Easy Validation The first step is to monitor a real system to get the trace. Performance characteristics of the system can be measured at the same time and be used for comparison with the output of the simulator.

Accurate workload A trace preserves the correlation and interference effects in the workload. No simplification, such as those needed in getting an analytical model of the workload, is required.

Detailed Trade-offs Due to the high level of detail in the workload, it is possible to study the effect of small changes in the model or algorithms.

Traces: Advantages

Less Randomness A trace is a deterministic input, which reduces the variance of output results or even leads to fully deterministic outputs if all parts of the model are deterministic.

Fair Comparison A trace allows different alternatives to be compared under the same input stream, which is a fairer comparison than when random stream are used and are thus different for the various alternatives being simulated (this can be avoided though).

Similarity to the actual implementation trace-driven models are generally very similar to the system being modeled. Thus one gets a good feeling for the complexity of implementing a given algorithm.

Traces: Drawbacks

Complexity It generally requires a more detailed simulation of the system. Sometimes, the complexity of the model overshadows the algorithm being modeled.

Representativeness Traces taken on one system may not be representative of the workload on another system. Workload may also change over time and thus traces become obsolete faster than other forms of workload models that can be adjusted with time.

Finiteness A trace is a long sequence. A detailed trace of a few minutes of activity on a system may be enough to fill a disk pack. A result based on those few minutes may not be applicable to the activities during the rest of the day.

Traces: Drawbacks

Single Point of Validation An algorithm that is the best for one trace may not be the best for another. At least a few different traces should be used to validate the results.

Detail Traces are generally very long sequences that have to be read from the disk and fill your computer's memory.

Trade-off The workload characteristics is difficult to change. It is not possible to change a trace (a new trace for the changed workload is required).

Bias Sometimes, the trace is the consequence of some part of the system that is to be simulated. Studying and thus changing this part of the system invalidates the trace.

Outline

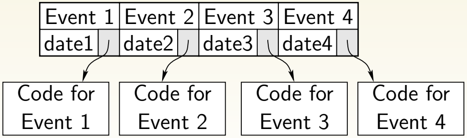
Array

As we have seen earlier, event management is crucial. Efficient event-set algorithms may save a large amount of processing time. This heavily depends on the kind of operations (and on the frequency) to be performed:

- Finding the next (earliest) event.
- Removing the next (earliest) event.
- Inserting new events in the set.
- Removing a given event (e.g., for an execution time update).
- Finding events that occur at a given time.

Depending on the cost of these operations, some data-structures may be more suited to a given usage than some others.

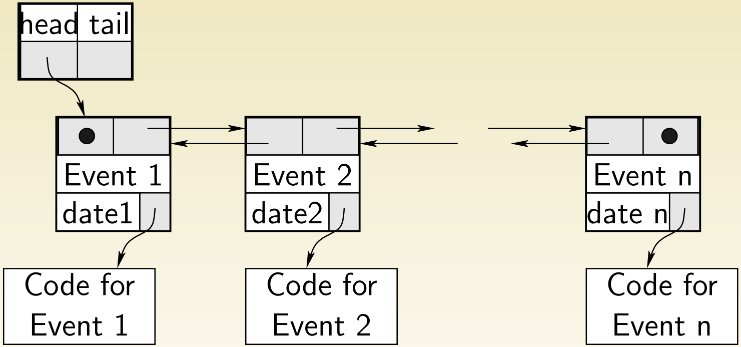
The simplest data-structure is an array or an ordered array.



Everything is $O(n)$.

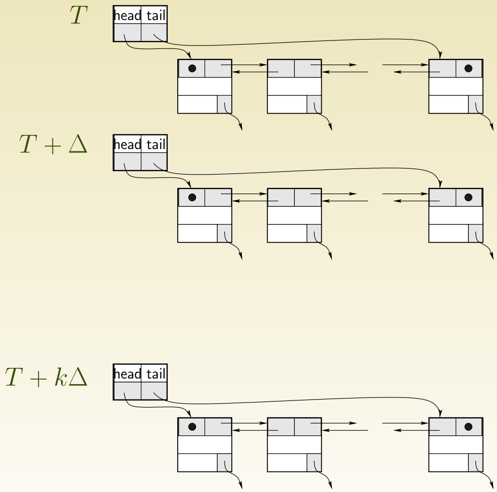
Doubly linked-list

This is the most commonly used data-structure.



Insertion is $O(n)$, other operations are $O(1)$. However, memory access are non-uniform.

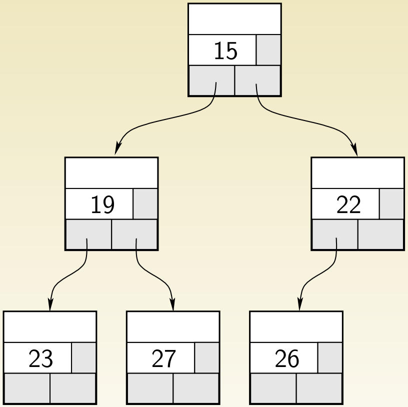
Indexed Linear List: Calendar



Speeds up Insertion and Search.

Many possible variations (non uniform sampling, fixed-size lists)

Tree structures



Most operations are $O(\log n)$.

Many possible variations (non uniform sampling, fixed-size lists)

Often, mixtures of data-structures are needed.

Outline

Random thoughts

Nobody will notice the core of your simulator is super-optimized if you spend 90% of your time collecting data or doing IO to store them.

The same principles as in monitor design apply. Use the same techniques as the ones you would use when designing a full-fledge monitor.

Most data need to be recorded in buffers, hence buffer management issues:

- buffer size is a function of input rate, input width and emptying rate,
- larger number of buffers allows to cope with variations in filling and emptying rates,
- buffer overflow management and tracking,
- data compression, on-line analysis,

Abnormal events should also be monitored and even handled at higher priority (low probability $\rightsquigarrow$ low overhead, possibility to take preventive action before the system becomes unavailable).

Summerization and Presentation

Once again, the same principles as in monitor design apply. You should provide hooks so that end-users can easily get the informations they need.

This layer is closely tied to the applications for which the monitor is used (performance monitoring, configuration monitoring, fault monitoring,. . .).

- Presentation frequency (for online monitors).
- Hierarchical representation/aggregation (space/time/states/values).
- Online vizualization of state variables.
- Alarm mode (thresholds, abnormal events).

Outline

Validation

Key question

How accurately does a simulation model reflects the actual system being simulated?

- Are the assumptions reasonable?
- Are the input distributions a good representation of what would be seen in practice?
- Are the outputs reasonable?
- Are the results explainable?

Different approaches:

Comparison with a real system run benchmarks.

Analytical results check that trends are correct.

Engineering judgment if you cannot explain why the simulator produces an unusual result, don't believe it.

Verification

Key question

Is the simulator implemented correctly?

Different approaches:

Deterministic verification
- Follow good software-engineering practices (structured programming, modular design, documentation, internal error checking, memory checking, ...).
- Step-by-step run to follow an execution and check that it proceeds through all desired states.
- Consistency checks.
- Simulate the system for special cases (corner conditions).

Stochastic verification How do we check that a sequence of pseudo-random numbers is "random enough"?...

Outline

Inappropriate level of detail

Simulation provides more detail than analytical modeling that relies on several simplifications and assumptions. In a simulation model, the level of detail is only limited by the time available for development.

More detailed simulations:

- require more times,
- increase the likelihood of bugs (and the difficulty to spot them),
- increase the simulation time,
- requires more input that may not be available.

It is a common belief that the more detailed the simulation, the more accurate the results.

FLASH vs. FLASH

FLASH vs. (Simulated) FLASH: Closing the Simulation Loop [GKOH00]

FLASH project at Stanford

- building large-scale shared-memory multiprocessors

  Went from conception, to design, to actual hardware (32-node)

  Used simulation heavily over 6 years

FLASH vs. FLASH

FLASH vs. (Simulated) FLASH: Closing the Simulation Loop [GKOH00]

- FLASH project at Stanford
  - building large-scale shared-memory multiprocessors
  - Went from conception, to design, to actual hardware (32-node)
  - Used simulation heavily over 6 years
- The authors went back and compared simulation to the real world!
  - Simulation error is unavoidable(30% error in their case was not rare; Negating the impact of “we got 1.5% improvement”)

FLASH vs. FLASH

FLASH vs. (Simulated) FLASH: Closing the Simulation Loop [GKOH00]

- FLASH project at Stanford
  - building large-scale shared-memory multiprocessors
  - Went from conception, to design, to actual hardware (32-node)
  - Used simulation heavily over 6 years
- The authors went back and compared simulation to the real world!
  - Simulation error is unavoidable(30% error in their case was not rare; Negating the impact of “we got 1.5% improvement”)
  - One should focus on simulating the important things

FLASH vs. FLASH

FLASH vs. (Simulated) FLASH: Closing the Simulation Loop [GKOH00]

- FLASH project at Stanford
  - building large-scale shared-memory multiprocessors
  - Went from conception, to design, to actual hardware (32-node)
  - Used simulation heavily over 6 years
- The authors went back and compared simulation to the real world!
  - Simulation error is unavoidable(30% error in their case was not rare; Negating the impact of "we got 1.5% improvement")
  - One should focus on simulating the important things
  - A more complex simulator does not ensure better simulation
    - Simple simulators worked better than sophisticated ones, which were unstable
    - Simple simulators predicted trends as well as slower, sophisticated ones

FLASH vs. FLASH

FLASH vs. (Simulated) FLASH: Closing the Simulation Loop [GKOH00]

- FLASH project at Stanford
  - building large-scale shared-memory multiprocessors
  - Went from conception, to design, to actual hardware (32-node)
  - Used simulation heavily over 6 years
- The authors went back and compared simulation to the real world!
  - Simulation error is unavoidable(30% error in their case was not rare; Negating the impact of “we got 1.5% improvement”)
  - One should focus on simulating the important things
  - A more complex simulator does not ensure better simulation
    - Simple simulators worked better than sophisticated ones, which were unstable
    - Simple simulators predicted trends as well as slower, sophisticated ones
  - It is key to use the real-world to tune/calibrate the simulator

FLASH vs. FLASH

FLASH vs. (Simulated) FLASH: Closing the Simulation Loop [GKOH00]

- FLASH project at Stanford
  - building large-scale shared-memory multiprocessors
  - Went from conception, to design, to actual hardware (32-node)
  - Used simulation heavily over 6 years
- The authors went back and compared simulation to the real world!
  - Simulati r case was not rare; Ne ement")
  - One sho gs
  - A more c simulation
    - Simple simulators worked better than sophisticated ones, which were unstable
    - Simple simulators predicted trends as well as slower, sophisticated ones
  - It is key to use the real-world to tune/calibrate the simulator

Conclusion

For FLASH, the simple simulator was all that was needed. . .

Improper Language/Environment

Selecting a proper language is probably the most important step in the process of developing a simulation model. An incorrect decision during this step may lead to long development times, incomplete studies and failures. There are three main choices:

A simulation language. SIMULA or SIMSCRIPT have built-in facilities for time-advancing, event scheduling, entity manipulation, random-variable generation, statistical data collection, report generation. You may not have time to learn such languages though and thus prefer:

A general-purpose language. Gives more flexibility by allowing some short-cuts prohibited in simulation languages.

A simulation package. The biggest advantage is the time savings that they provide. Using such a package, you could develop a model, solve it and get results in a few days. It takes generally several weeks (months ?) to do it from scratch.

The main problem is their inflexibility. They provide only flexibilities that were foreseen by their developers. In most practical situations, one has either to hack the package or make simplifications or use it in a tortured way.

Other very common mistakes

- Unverified Models (bugs)
- Invalid Models (incorrect assumptions in the model design)
  All simulation model results should be suspected until confirmed by analytical models, measurements, or intuition.
- Improperly handled initial conditions (is the input workload representative of the modeled system ?)
- Too short simulations
- Poor random-number generators and improper selection of seeds
- Inadequate time estimate (are you going for a 1 week, a 1 month, a 1 year project?)

Simulation is art and a science

Simulation is art and a science

Do not get fooled by nice talks about portability, modularity and nice GUIs. Ask about the simulation core. If you can't get answers, dive in the code and read how the engine works.

Outline

Outline

Outline

SimGrid

A few characteristics:

- SimGrid is a deterministic event-driven simulator.
- It uses hybrid state variables (discrete for process, continuous for tasks and resources).
- It is both Execution and Trace driven.
- It uses hybrid data-structures for traces and event lists.
- It uses "just-in-time" event completion time computation.
- There is almost no data-recording nor summarization routines as developers don't like it and are not good at it.

Jeff Gibson, Robert Kunz, David Ofelt, and Mark Heinrich.
FLASH vs. (simulated) FLASH: Closing the simulation loop.
In *Architectural Support for Programming Languages and Operating Systems*, pages 49–58, 2000.

R. K. Jain.
The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.
John Wiley & Sons Canada, Ltd., 1 edition, 1991.

David J. Lilja.
Measuring Computer Performance: A Practitioner's Guide - David J. Lilja - Hardcover.
Cambridge University Press, 2000.