



EEC 686/785
Modeling & Performance Evaluation of
Computer Systems

Lecture 14

Wenbing Zhao

Department of Electrical and Computer Engineering
Cleveland State University

wenbing@ieee.org

(based on Dr. Raj Jain's lecture notes)



Outline

- Review of lecture 13
 - 2^{k-p} Fractional Factorial Designs
 - One factor experiment
- Introduction to simulation



2^{k-p} Fractional Factorial Designs

- Large number of factors
 - Large number of experiments
 - Full factorial design too expensive
 - Use a fractional factorial design
- 2^{k-p} design allows analyzing k factors with only 2^{k-p} experiments
 - 2^{k-1} design requires only half as many experiments
 - 2^{k-2} design requires only one quarter of the experiments



Preparing Sign Table for 2^{k-p} Design

- Prepare a sign table for a full factorial design with $k-p$ factors
- Mark the first column I
- Mark the next $k-p$ columns with the $k-p$ factors
- Of the $(2^{k-p} - k + p - 1)$ columns on the right, choose p columns and mark them with the p factors which were not chosen in step 1

Algebra of Confounding

- Given just one confounding, it is possible to list all other confoundings
- Rules:
 - I is treated as unity
 - Any term with a power of 2 is erased

Design Resolution

- Order of an effect = number of factors included in it
- Order of ABCD=4, order of I=0
- Order of a confounding = sum of the order of two terms
 - E.g., AB=CDE is of order 5
- Resolution of a design = minimum of orders of confoundings
- Notation: $R_{III} = \text{Resolution III} = 2^{k-p}$

One Factor Experiments

- Used to compare alternatives of a single **categorical** variable
 - For example, several processors, several caching schemes
- Model: $y_{ij} = \mu + \alpha_j + e_{ij}$
 - r = number of replications y_{ij} = i th response with j th alternative
 - μ = mean response α_j = effect of alternative j
 - e_{ij} = error term
 - $$\sum \alpha_j = 0$$

Analysis of Variance (ANOVA)

- Importance \neq significance
- **Important** \Rightarrow explains a high percent of variation
- **Significance** \Rightarrow high contribution to the variation compared to that by errors
- Degree of freedom = number of independent values required to compute

$$SSY = SS0 + SSA + SSE$$

$$ar = 1 + (a - 1) + a(r - 1)$$

Note that the degrees of freedom also add up

F-Test

- Purpose: to check if SSA is *significantly* greater than SSE
 - Errors are normally distributed => SSE and SSA have chi-square distributions
 - The ratio $(SSA/v_A)/(SSE/v_e)$ has an F distribution.
Where $v_A = a - 1 =$ degree of freedom for SSA
 $v_e = a(r - 1) =$ degree of freedom for SSE
 - Computed ratio $> F_{[1-\alpha; v_A, v_e]}$
=> SSA is **significantly** higher than SSE
 - SSA/v_A is called mean square of A or (MSA).
Similarly, $MSE = SSE/v_e$

Simulation: Key Questions

- What language should be used for developing a simulation model?
- What are different types of simulations?
- How to schedule events in a simulation?
- How to verify and validate a model?
- How to determine that the simulation has reached a steady state?
- What are the common mistakes in simulation and why most simulations fail?

Simulation: Key Questions

- How long to run a simulation?
- How to generate uniform random numbers?
- How to verify that a given random number generator is good?
- How to select seeds for a random number generators?
- How to generate random variables with a given distribution?
- What distributions should be used and when?

Introduction to Simulation

- Simulation is a useful technique for computer systems performance analysis
- A simulation model provides an easy way to predict the performance or compare several alternatives
 - Applicable if the system to be characterized is not available, *e.g.*, during the design or procurement stage
- A simulation model allows the alternatives to be compared under a wider variety of workloads and environments
 - Useful even if a system is available for measurement
 - It may be preferred over measurements

Terminology

- Using an example of simulating CPU scheduling to describe the following terms
- **State variables**
 - The variables whose values define the state of the system
 - Can restart simulation from state variables (if their values are known)
 - E.g., in CPU scheduling simulation, the state variable is the length of the job queue
- **Event**
 - A change in the system state is called an event
 - E.g., arrival of a job, beginning of a new execution, departure of a job

Continuous Time and Discrete Time Models

- **Continuous-time model:** a model in which the system state is defined at all times
 - E.g., CPU scheduling model
- **Discrete time model:** a model in which the system state is defined only at particular instants in time
 - E.g., a model for a class that meets every Tuesday and Thursday

Continuous State and Discrete State Models

- A model is called **continuous-** or **discrete-state model** depending upon whether the state variables are continuous or discrete
 - E.g., continuous-state model: the state is described by the time spent on the subject by students
 - E.g., discrete-state model: the state is described by the number of students
 - E.g., in the CPU scheduling model, the state variable, i.e., the queue length, can assume only integer values => it is a discrete-state model

Continuous State and Discrete State Models

- Discrete-state model is often called discrete event model
- Continuous-state model is often called continuous event mode
- Continuity of time \neq continuity of state
- Four possible combinations:
 - Discrete state / discrete time; discrete state / continuous time
 - Continuous state / discrete time; Continuous state / continuous time

Deterministic and Probabilistic Models

- If the output (results) of a model can be predicted with certainty, it is a deterministic model
- A probabilistic model gives a different result on repetitions for the same set of input parameters

Static and Dynamic Models

- **Static model:** a model in which time is not a variable
 - E.g., matter-to-energy transformation model: $E=mc^2$
- **Dynamic model:** a model with a state that changes with time
 - E.g., CPU scheduling model

Linear and Nonlinear Models

- **Linear model:** the output parameters are a linear function of the input parameter
- **Nonlinear model:** the output parameters are a nonlinear function of the input parameter

Open and Closed Models

- **Open model:** the input is external to the model and is independent of it
- **Closed model:** there is no external input

Stable and Unstable Models

- **Stable model:** the dynamic behavior of the model settles down to a steady state, i.e., independent of time
- **Unstable model:** a model whose behavior is continuously changing

Computer System Models

- Continuous time
- Discrete state
- Probabilistic
- Dynamic
- Nonlinear
- Open or closed
- Stable or unstable

Selecting a Language for Simulation

- Simulation language
- General purpose language
- Extension of a general purpose language
- Simulation package

Simulation Languages

- Save development time
- Built-in facilities for time advancing, event scheduling, entity manipulation, random variate generation, statistical data collection, and report generation
- More time for system specific issues
- Very readable modular code

General Purpose Language

- A general purpose language is often chosen because of an analyst's familiarity with the language
- A general purpose language is more portable than simulation languages
- However, one has to spend time developing routines for event handling, random-number generation, and so forth

Extension of a General Purpose Language

- Extend a general purpose language with a collection of routines to handle simulation tasks
- Compromise for efficiency, flexibility, and portability
- Examples: GASP (for FORTRAN), JSIM for Java

Simulation Packages

- Example: QNET4, RESQ, NS2(?)
 - Input dialog
 - Library of data structures, routines, and algorithms
 - Big time savings
 - Inflexible => simplification

Types of Simulation Languages

- Continuous simulation languages:
 - CSMP, DYNAMO => popular in chemical system modeling
- Discrete-event simulation languages
 - SIMULA and GPSS
- Combined: SIMSCRIPT and GASP
 - Allow discrete, continuous, and well as combined simulations

Types of Simulations

- Emulation: using hardware or firmware
 - E.g., terminal emulator, processor emulator
 - Mostly hardware design issues
- Monte Carlo simulation
- Trace-driven simulation
- Discrete event simulation
- Process-oriented simulation

Monte Carlo Simulation

- Static simulation (no time axis)
- To model probabilistic phenomenon
- Need pseudorandom numbers
- Used for evaluating nonprobabilistic expressions using probabilistic methods

Monte Carlo: Example

- Integral to be evaluated: $I = \int_0^2 e^{-x^2} dx$
- Using Monte Carlo: generate uniformly distributed random numbers x and for each number compute a function y as follows:

$$x \sim \text{uniform}(0,2)$$

$$\text{density function } f(x) = 1/2 \quad \text{iff } 0 \leq x \leq 2$$

$$y = 2e^{-x^2}$$

Monte Carlo: Example

- The expected value of y is:

$$E(y) = \int_0^2 2e^{-x^2} f(x) dx = \int_0^2 2e^{-x^2} \frac{1}{2} dx = \int_0^2 e^{-x^2} f(x) dx = I$$
- Thus, the integral can be evaluated by generating uniformly distributed random numbers x_i , computing y_i , and then averaging as follows:

$$x_i \sim \text{uniform}(0,2)$$

$$y_i = 2e^{-x_i^2}$$

$$I = E(y) = \frac{1}{n} \sum_{i=1}^n y_i 2e^{-x_i^2}$$

Trace-Driven Simulation

- Trace = time ordered record of events on a system
- Trace-driven simulation = a simulation using a trace as its input
- Used in analyzing or tuning resource management algorithms
 - Paging, cache analysis, CPU scheduling, deadlock prevention, dynamic storage allocation

Trace-Driven Simulation

- Example: trace = page reference patterns
 - The trace can be used to find the optimal set of parameters for a given memory management algorithm or to compare different algorithms
- Should be independent of the system under study
 - E.g., trace of pages fetched depends upon the working set size and page replacement policy
 - Not good for studying other page replacement policies
 - Better to use pages referenced (in the trace)
 - An instruction trace obtained on one OS should not be used to analyze another OS

Advantages of Trace-Driven Simulation

- **Credibility:** e.g., a trace of page references has more credibility than references generated randomly using an assumed distribution
- **Easy validation:** compare simulation with measured
 - The measurement could be done while the trace is obtained
- **Accurate workload:** models correlation and interference
 - A trace preserve the correlation and interference effects in the workload
- **Detailed trade-offs:** detailed workload => can study small changes in algorithms

Advantages of Trace-Driven Simulation

- **Less randomness:** a trace is a deterministic input
 - The output has less variance => fewer number of repetitions are required to get a desired statistical confidence in the result
 - If other parts of the system are not random, it is possible to get absolute results in one execution of the model
- **Fair comparison:** a trace allows different alternatives to be compared under the same input stream
 - better than random input used to compare different systems
- **Similarity to the actual implementation:** trace-driven model is similar to the system => can understand complexity of implementation

Disadvantages of Trace-Driven Simulation

37

- **Complexity:** more detailed \Rightarrow complexity of the model might overshadow the algorithm being modeled
- **Representativeness:** workload changes with time, equipment \Rightarrow the trace taken may not be representative
- **Finiteness:** a trace is a long sequence. A detailed trace of a few minutes may fill up a disk

28 October 2005

EEEC686/785

Wenbing Zhao

Disadvantages of Trace-Driven Simulation

38

- **Single point of validation:** remember a trace offers only a single point of validation
 - An algorithm best for one trace may not be the best for another
- **Detail:** high level of details because a trace consists of very long sequences
- **Trade-off:** difficult to change workload \Rightarrow would need to obtain another trace. Also, if a trace contains resource demand characteristics of several jobs, it is difficult to study effects on individual jobs

28 October 2005

EEEC686/785

Wenbing Zhao

Discrete Event Simulations

39

- In computer systems, discrete event models are used
 - The state of the system is described by the number of jobs at various devices
- Discrete state \neq discrete time

28 October 2005

EEEC686/785

Wenbing Zhao

Components of Discrete Event Simulations

40

- **Event scheduler:** keeps a linked list of events waiting to happen. The scheduler allows the events to be manipulated in various ways, e.g.,
 - Schedule event X at time T
 - Hold event X for a time interval dt
 - Cancel a previously scheduled event X
 - Hold event X indefinitely
 - Schedule an indefinitely held event
- \Rightarrow Event scheduler is one of the most frequently executed components of the simulation. It is executed before every event
- \Rightarrow Event scheduling has a significant impact on the computational efficiency of a simulation

28 October 2005

EEEC686/785

Wenbing Zhao

Components of Discrete Event Simulations

41

■ Simulation clock and a time-advancing mechanism

- Each simulation has a global variable representing simulated time
- The scheduler is responsible for advancing this time
 - Unit-time approach: increments time by small increment and then checks to see if there are any events that can occur
 - Event-driven approach: increments the time automatically to the time of the next earliest occurring event => this approach is often used in computer simulations

28 October 2005

EEEC686/785

Wenbing Zhao

Components of Discrete Event Simulations

42

- **System state variables:** global variables that describe the state of the systems
 - E.g., in the CPU scheduling simulation, the system state variable is the number of jobs in the queue
 - This is a global variable
 - CPU time required for a job is a local variable => it is stored in the data structure representing the job
- **Event routines:** each event is simulated by its routine. One routine per each event
 - These routines update system state variables and schedule other events
 - E.g., job arrivals, job scheduling, and job departure

28 October 2005

EEEC686/785

Wenbing Zhao

Components of Discrete Event Simulations

43

- **Input routines:** used to get 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 so that the simulation can run non-interactively
 - Input routines allow parameters to vary in a range, e.g., mean CPU demand varying from 1 to 9 milliseconds in steps of 2 milliseconds
 - Each set of input values defines one iteration that may have to be repeated several times with different seeds
 - Each single execution of the simulation consists of several **iterations**, and each iteration consists of several **repetitions**

28 October 2005

EEEC686/785

Wenbing Zhao

Components of Discrete Event Simulations

44

- **Report generator:** consists of output routines executed at the end of the simulation
 - They calculate the final result and print in a specified format
- **Initialization routines:** they set the initial state of the system state variables and initialize various random-number generation streams (initialize seeds)
 - A good practice: have separate routines to init state at beginning of simulation, at the beginning of an iteration, and at the beginning of a repetition

28 October 2005

EEEC686/785

Wenbing Zhao

Components of Discrete Event Simulations

45

- **Trace routines:** they print out intermediate variables as simulation proceeds
 - They help debug the simulation program
 - The trace should have on/off feature so that it can be turned off for final production runs of the model
- **Dynamic memory management:** garbage collection is needed to handle large number of events
- **Main program:** brings all routines together
 - It call input routines, initializes the simulation, executes various iterations, and finally, calls the output routines

28 October 2005

EEEC686/785

Wenbing Zhao

Event-Set Algorithms

46

- In discrete-event simulations, it is necessary to ensure that events occur in proper order and at the proper time => **need event-set algorithms**
- **Event Set** = ordered linked list of future event notices
 - Each notice contains the time at which the event should occur and a pointer to the code that must be executed at that time
 - Two operations frequently performed:
 - Insert a new event in the set
 - Find the next event to execute and remove it from the set
 - Choice of data structures used affects overheads of insert / find and remove operations

28 October 2005

EEEC686/785

Wenbing Zhao

Event-Set Algorithms

47

- **Ordered linked list:** used in simulation languages such as SIMULA, GPSS, and GASP IV
 - First entry in the list is the next earliest event => removal is easy
 - To insert a new event, the list is searched to find the right place. There can be a number of different searching strategies

28 October 2005

EEEC686/785

Wenbing Zhao

Event-Set Algorithms

48

- **Indexed linear list:** the set of future events is divided into several subsets. Each subset spans a fixed interval of the time interval Δt and is maintained as a sublist
 - Use an array of indexes => no search to find the sublist
 - Fixed or variable Δt

28 October 2005

EEEC686/785

Wenbing Zhao

Event-Set Algorithms

- **Calendar queues:** a variation from the indexed linear list
 - All events of the same day on one page
 - Events for the same day of the next year can also be written on that page as long as
 - the year of occurrence is also written down with the event
 - the events are deleted after they have taken place

Event-Set Algorithms

- **Tree structures:**
 - binary tree $\Rightarrow \log_2 n$
- **Heap:** event is a node in binary tree

Event-Set Algorithms

- **Heap:** event is a node in binary tree
 - Up to two children
 - Event time for each node is smaller than that of its children
 - \Rightarrow root always has the earliest event time
 - Heap can be stored as arrays
 - Children of node in position i are in positions $2i$ and $2i+1$
- The heap can be extended to k -ary heaps that make use of k -ary trees
 - Each node has up to k children

Event-Set Algorithms

- **Guideline on selecting data structures**
 - 0-20 events: simple linked list
 - 20-120 events: index linear
 - 120+ events: heaps

Process-Oriented Simulation

- Event-driven simulation
 - Strength: it is general => any discrete-event simulation may be implemented
 - Weakness: it is not user-friendly in some respects
 - Each event stands alone <= by definition events maintain no context because they only exist for zero simulation time
- Process-oriented simulation: allows related state changes to be combined in the context of a process

Process-Oriented Simulation

- A sets of related event types are grouped together in a process, similar to a process in an operating system
- A process consists of
 - Body of code
 - Resources allocated
 - State, i.e., current point of execution in the code and the values of all the variables accessed by the code

Process-Oriented Simulation

- The simulation driver is more complicated: in addition to managing the event list, it has to manage
 - process creation and activation/scheduling
 - process suspension and process termination
 - interprocess communication/synchronization