



EEC 686/785
Modeling & Performance Evaluation of
Computer Systems

Lecture 16

Wenbing Zhao

Department of Electrical and Computer Engineering
Cleveland State University

wenbing@ieee.org

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



2

Outline

- Review of lecture 15
- Random number generation



3

Analysis of Simulation Results

- Model verification techniques
 - **Verification** => debugging, correct implementation of the model
- Model validation techniques
 - **Validation** => model = real world, valid assumption
- Transient removal
- Terminating simulations
- Stopping criteria: variance estimation
- Variance reduction



4

Model Verification Techniques

- Top down modular design
 - Antibugging
 - Structured walk-through
 - Deterministic models
 - Run simplified cases
 - Trace
 - Online graphic displays
 - Continuity test
 - Degeneracy tests
 - Consistency tests
 - Seed independence

Model Validation Techniques

- Aspects to validate
 - Assumptions
 - Input parameter values and distributions
 - Output values and conclusions
- Techniques
 - Expert intuition
 - Real system measurements
 - Theoretical results

Transient Removal

- General steady state performance is interesting
 - Remove the initial part. No exact definition => heuristics
- Transient removal methods
 - Long runs
 - Proper initialization
 - Truncation
 - Initial data deletion
 - Moving average of independent replications
 - Batch means

Truncation

- Assumes variability is lower during steady state
- Plot max-min of $n-l$ observation for $l=1,2,\dots$
- When $(l+1)$ th observation is neither the minimum nor maximum => transient state ended
- Example:
 - 1,2,3,4,5,6,7,8,9,10,11,10,9,10,11,10,9,10,11,10,9,...
 - At $l=9$, range = (9,11), next observation = 10
 - Sometimes incorrect result

Initial Data Deletion

1. Get a mean trajectory by averaging across replications
2. Get the overall mean
3. Delete the first l observations and get an overall mean from the remaining $n-l$ values
4. Compute the relative change
5. Repeat steps 3 and 4 by varying l from 1 to $n-1$
6. Plot the overall mean and the relative change
7. l at knee = length of the transient interval

Stopping Criteria: Variance Estimation

- Run until confidence interval is narrow enough

$$\bar{x} \pm z_{1-\alpha/2} \sqrt{\text{Var}(\bar{x})}$$
- Independence not applicable to most simulations. Large waiting time for i th job \Rightarrow large waiting time for $(i+1)$ th job
- For correlated observations: Actual variance $\gg \text{Var}(x)/n$
- Solutions:
 - Independent replications
 - Batch means
 - Method of regeneration

Independent Replications

- Assumes that means of independent replications are independent
- Conduct m replications of size $n+n_0$ each
 - 1. Compute a mean for each replications
 - 2. Compute an overall mean for all replications
 - 3. Calculate the variance of replicate means

$$\text{Var}(\bar{x}) = \frac{1}{m-1} \sum_{i=1}^m (\bar{x}_i - \bar{\bar{x}})^2$$
 - 4. Calculate the confidence interval for the mean response

Batch Means

- Also called method of subsamples
- Run a long simulation run. Discard initial transient interval, and divide the remaining observations run into several batches or subsamples
 - 1. Compute means for each batch
 - 2. Compute an overall mean

$$\text{Var}(\bar{x}) = \frac{1}{m-1} \sum_{i=1}^m (\bar{x}_i - \bar{\bar{x}})^2$$
 - 3. Calculate the variance of batch means
 - 4. Calculate the confidence interval for the mean response
- Check: compute the autocovariance of successive batch means. Double batch size until autocovariance is small

Method of Regeneration

- Behavior after idle period does not depend upon the past history
 - System takes a new birth
 - Regenerate point
- Regenerate cycle: between two successive regeneration points
- Use means of regeneration cycles
- Problems:
 - Not all systems are regenerative
 - Different lengths \Rightarrow computation complex
 - Overall mean \neq average of cycle means

Method of Regeneration

- Cycle means are given by:

$$\bar{x}_i = \frac{1}{n_i} \sum_{j=1}^{n_i} x_{ij}$$

- However, overall mean is not an arithmetic mean of cycle means

$$\bar{\bar{x}} \neq \frac{1}{m} \sum_{i=1}^m \bar{x}_i$$

Method of Regeneration

- Procedure to compute the overall mean and its confidence interval

- 1. Compute cycle sums: $y_i = \sum_{j=1}^{n_i} x_{ij}$
- 2. Compute overall mean: $\bar{\bar{x}} = \frac{\sum_{i=1}^m y_i}{\sum_{i=1}^m n_i}$

- 3. Calculate the difference between expected and observed cycle sums

$$w_i = y_i - n_i \bar{\bar{x}} \quad i = 1, 2, \dots, m$$

Method of Regeneration

- 4. Calculate the variance of the differences:

$$Var(w) = s_w^2 = \frac{1}{m-1} \sum_{i=1}^m w_i^2$$

- 5. Compute mean cycle length: $\bar{n} = \frac{1}{m} \sum_{i=1}^m n_i$

- 5. Confidence interval for the mean response is given by:

$$\bar{\bar{x}} \mp z_{1-\alpha/2} \frac{s_w}{\bar{n} \sqrt{m}}$$

- No need to remove transient observations

Problems of Regeneration Method

- The cycle lengths are unpredictable. Can't plan the simulation time beforehand
- Finding the regeneration point may require a lot of checking after every event
- Many of the variance reduction techniques can not be used due to variable length of the cycles

Variance Reduction

- Reduce variance by controlling random number streams
- Introduce correlation in successive observations
- **Problem:** careless use may backfire lead to increased variance
- For statistically sophisticated analysts only
- Not recommended for beginners

Common Mistakes in Simulation

- Inappropriate level of detail
 - More detail => more time => more bugs => more CPU => more parameters \neq more accurate
- Improper language
 - General purpose => more portable, more efficient, more time
- Unverified models
 - Bugs
- Invalid models
 - Model vs. reality

Common Mistakes in Simulation

- Improperly handled initial conditions
- Too short simulations
 - Need confidence intervals
- Poor random number generators
 - Safer to use a well-known generator
- Improper selection of seeds
 - Zero seeds, same seeds for all streams

Other Causes of Simulation Analysis Failure

- Inadequate time estimate
- No achievable goal
- Incomplete mix of essential skills
 - Project leadership
 - Modeling and statistics
 - Programming
 - Knowledge of the modeled system

Other Causes of Simulation Analysis Failure

21

- Inadequate level of user participation
- Obsolete or nonexistent documentation
- Inability to manage the development of a large complex computer program
 - Need software engineering tools
- Mysterious results

6 November 2005

EEC686/785

Wenbing Zhao

Random-Number Generation

22

- A key step in developing a simulation: have a routine to generate random values for variables with a specified random distribution
 - **Random number generation:** a sequence of random numbers distributed uniformly between 0 and 1
 - **Random variate generation:** the sequence is transformed to produce random values obeying the desired distribution

6 November 2005

EEC686/785

Wenbing Zhao

A Sample Generator

23

- Most common method: use recursive relation
$$x_n = f(x_{n-1}, x_{n-2}, \dots)$$
- For example: $x_n = 5x_{n-1} + 1 \pmod{16}$
- Starting with $x_0 = 5$:
$$x_1 = 5(5) + 1 \pmod{16} = 26 \pmod{16} = 10$$
- The first 32 numbers obtained by the above procedure are:
10,3,0,1,6,15,12,13,2,11,8,9,14,7,4,5,
10,3,0,1,6,15,12,13,2,11,8,9,14,7,4,5

Divide these by 16 to render them within (0,1)

6 November 2005

EEC686/785

Wenbing Zhao

Terminology

24

- **Seed:** x_0
- **Pseudo-random:** deterministic yet would pass randomness tests
- **Fully random:** not repeatable
- **Cycle length, Tail, Period:**

6 November 2005

EEC686/785

Wenbing Zhao

Desired Properties of a Good Generator

- It should be efficiently computable
- The period should be large
- The successive values should be independent and uniformly distributed

Types of Random-Number Generators

- Linear congruential generators
- Tausworthe generators
- Extended Fibonacci generators
- Combined generators

Linear-Congruential Generators

- Discovered by D. H. Lehmer in 1951: The residues of successive powers of a number have good randomness properties

$$x_n = a^n \bmod m$$

Equivalently,

$$x_n = a x_{n-1} \bmod m$$

a = multiplier, m = modulus

Linear-Congruential Generators

- Lehmer's choices: $a = 23$ and $m = 10^8 + 1$
 - Good for ENIAC, an 8-digit decimal machine
- Generalization: $x_n = ax_{n-1} + b \bmod m$
- Can be analyzed easily using the theory of congruences
- This type of generators are called **mixed linear-congruential generators**, or **linear-congruential generators (LCG)** for short
 - Mixed = both multiplication by a and addition of b

Selection of LCG Parameters

- a , b , and m affect the period and autocorrelation
- The modulus m should be large. The period can never be more than m
- For mod m computation to be efficient, m should be a power of 2 \Rightarrow mod m can be obtained by truncation (shift)

Selection of LCG Parameters

- If b is nonzero, the maximum possible period m is obtained if and only if:
 - Integers m and b are relatively prime, that is, have no common factors other than 1
 - Every prime number that is a factor of m is also a factor of $a - 1$
 - If integer m is a multiple of 4, $a - 1$ should be a multiple of 4
- All conditions are met if $m = 2^k$, $a = 4c + 1$, and b is odd. Here, c , b , and k are positive integers

Period vs. Autocorrelation

- A generator that has the maximum possible period is called a **fully-period generator**

$$x_n = (2^{34}+1) x_{n-1} + 1 \pmod{2^{35}}$$

$$x_n = (2^{18}+1) x_{n-1} + 1 \pmod{2^{35}}$$
- **Lower autocorrelation** between successive numbers are preferable
- Both generators have the same full period, but the first one has a correlation of 0.25 between x_{n-1} and x_n , whereas the second one has a negligible correlation of less than 2^{-18}

Multiplicative LCG

- Multiplicative LCG: $b=0$

$$x_n = ax_{n-1} \pmod{m}$$

- Two types:
 - $m = 2^k$
 - $m \neq 2^k$

Multiplicative LCG with $m = 2^k$

- $m = 2^k \Rightarrow$ trivial division
 \Rightarrow maximum possible period 2^{k-2}
- Period achieved if
 - Multiplier a is of the form $8i \pm 3$, and
 - The initial seed is an odd integer
- One-fourth the maximum possible may not be too small for some simulations
- Low order bits of random numbers obtained using multiplicative LCG's with $m = 2^k$ have a cyclic pattern

Example 26.1

- $x_n = 5x_{n-1} \pmod{2^5}$
- Using a seed of $x_0 = 1$:
 - 5, 25, 29, 17, 21, 9, 13, 1, 5, ...
 - Period = 8 = $32/4$
- With $x_0 = 2$:
 - The sequence is: 10, 18, 26, 2, 10,
 - Period is only 4
- Multiplier not of the form $8i \pm 3$:
 - $x_n = 7x_{n-1} \pmod{2^5}$
 - Using a seed of $x_0 = 1$, we get the sequence: 7, 17, 23, 1, 7, ... The period is only 4

Multiplicative LCG with $m \neq 2^k$

- Modulus $m =$ prime number
 - With a proper multiplier a , period = $m-1$
 - 0 can never be reached
 - Maximum possible = m
- If and only if the multiplier a is a *primitive root* of the modulus m
- a is a primitive root of m if and only if $a^n \pmod{m} \neq 1$ for $n = 1, 2, \dots, m-2$

Example 26.2

- $x_n = 3x_{n-1} \pmod{31}$
- Starting with a seed of $x_0 = 1$:
 - 1, 3, 9, 27, 19, 26, 16, 17, 20, 29, 25, 13, 8, 24, 10, 30, 28, 22, 4, 12, 5, 15, 14, 11, 2, 6, 18, 23, 7, 31, 1, ...
 - Period is 30
 - $\Rightarrow 3$ is a primitive root of 31
- With a multiplier of $a = 5$:
 - 1, 5, 25, 1, The period is only 3
 - $\Rightarrow 5$ is not a primitive root of 31: $5^3 \pmod{31} = 125 \pmod{31} = 1$
- Primitive roots of 31 = 3, 11, 12, 13, 17, 21, 22, and 24

Schrage's Method

- Assumes:
 - No round-off errors
 - Use integer arithmetic and no overflows
 - Product $ax_{n-1} >$ largest integer allowed in system \Rightarrow overflow
- Identity: $ax \bmod m = g(x) + mh(x)$
 - Where: $g(x) = a(x \bmod q) - r(x \operatorname{div} q)$
 - And: $h(x) = (x \operatorname{div} q) - (ax \operatorname{div} m)$
 - Here, $q = m \operatorname{div} a$, $r = m \bmod a$
 - 'A div B' = dividing A by B and truncating the result

Schrage's Method

- For all x 's in the range $1, 2, \dots, m-1$, computing $g(x)$ involves numbers less than $m-1$
- If $r < q$, $h(x)$ is either 0 or 1, and it can be inferred from $g(x)$; $h(x)$ is 1 if and only if $g(x)$ is negative

Example 26.3

- $x_n = 7^5 x_{n-1} \bmod (2^{31}-1)$
- $2^{31} - 1 = 2147483647 =$ prime number
- $7^5 = 16807$ is one of its 534,600,000 primitive roots
- The product ax_{n-1} can be as large as $16807 \times 2147483647 \approx 1.03 \times 2^{45}$
- Need 46-bit integers
 - $a = 16807$, $m = 2147483647$
 - $q = m \operatorname{div} a = 2147483647 \operatorname{div} 16807 = 127,773$
 - $r = m \bmod a = 2147483647 \bmod 16807 = 2836$
- For a correct implementation, $x_0=1 \Rightarrow x_{10000} = 1,043,618,065$

Tausworthe Generators

- Need long random numbers for cryptographic applications
- Generate random sequence of binary digits (0 or 1). Divide the sequence into strings of desired length
- Proposed by Tausworthe (1965):

$$b_n = c_{q-1} b_{n-1} \oplus c_{q-2} b_{n-2} \oplus c_{q-3} b_{n-3} \oplus \dots \oplus c_0 b_{n-q}$$
 where c_i and b_i are binary variables with values of 0 or 1, and \oplus is the exclusive-or (mod 2 addition) operation
- Uses the last q bits of the sequence \Rightarrow autoregressive sequence of order q or AR(q)
- An AR(q) generator can have a maximum period of $2^q - 1$

Tausworthe Generators

- D=delay operator such that $D^q b(n) = b(n+1)$:
 $D^q b(i-q) = c_{q-1}D^{q-1}b(i-q) + c_{q-2}D^{q-2}b(i-q) + \dots + c_0 b(i-q) \pmod 2$
 Or: $D^q - c_{q-1}D^{q-1} - c_{q-2}D^{q-2} - \dots - c_0 = 0 \pmod 2$
 Or: $D^q + c_{q-1}D^{q-1} + c_{q-2}D^{q-2} + \dots + c_0 = 0 \pmod 2$
 Characteristic polynomial
 $x^q + c_{q-1}x^{q-1} + c_{q-2}x^{q-2} + \dots + c_0$
- The period is the smallest positive integer n for which $x^n - 1$ is divisible by the characteristic polynomial
- The maximum possible period which a polynomial of order q is $2^q - 1$. The polynomials that give this period are called **primitive polynomials**

Example 26.4

- $x^7 + x^3 + 1$
- Using D operator in place of x :
 $D^7 b(n) + D^3 b(n) + b(n) = 0 \pmod 2$
 $\Rightarrow b_{n+7} + b_{n+3} + b_n = 0 \pmod 2 \quad n = 0, 1, 2, \dots$
 \Rightarrow using the exclusive-or operator
 $b_{n+7} \oplus b_{n+3} \oplus b_n = 0 \quad n = 0, 1, 2, \dots$
 $\Rightarrow b_{n+7} = b_{n+3} \oplus b_n \quad n = 0, 1, 2, \dots$
 \Rightarrow Substituting $n-7$ for n :
 $b_n = b_{n-4} \oplus b_{n-7} \quad n = 7, 8, 9, \dots$

Example 26.4

- Starting with $b_0 = b_1 = \dots = b_6 = 1$:
 $b_7 = b_3 \oplus b_0 = 1 \oplus 1 = 0$
 $b_8 = b_4 \oplus b_1 = 1 \oplus 1 = 0$
 $b_9 = b_5 \oplus b_2 = 1 \oplus 1 = 0$
 $b_{10} = b_6 \oplus b_3 = 1 \oplus 1 = 0$
 $b_{11} = b_7 \oplus b_4 = 0 \oplus 1 = 1$
 ...
- The complete sequence is:
 1111111 0000111 0111100 1011001 0010000 0010001 0011000 1011101
 0110110 0000110 0110101 0011100 1111011 0100001 0101011 1110100
 1010001 1011100 0111111 1000011 1000000
 Period = 127 or $2^7 - 1$ bits \Rightarrow The polynomial $x^7 + x^3 + 1$ is a primitive polynomial

Linear Feedback Shift Register

- Tausworthe sequence can be generated in hardware using Linear Feedback Shift Register
- $x^5 + x^3 + 1 \Rightarrow b_n = b_{n-2} \oplus b_{n-5}$