Performance Modeling of Computer Systems and Networks

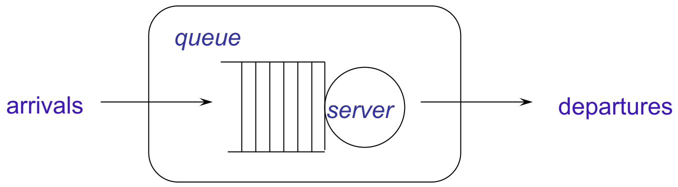
Prof. Vittoria de Nitto Personè

Discrete-Event Simulation
examples

Università degli studi di Roma Tor Vergata
Department of Civil Engineering and Computer Science Engineering

1

Single Server Queue



Arrival times: a_i

15 47 71 111 123 152 166 226 310 320

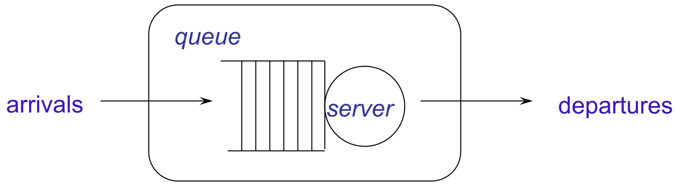
Pseudo-random generators

Service times: s_i

43 36 34 30 38 40 31 29 36 30

2

Single Server Queue



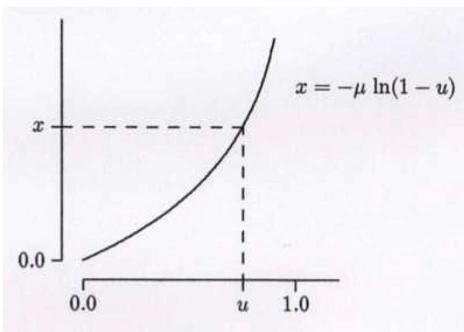
- assume **service times** are between 1.0 and 2.0 minutes
  - The distribution within this range is unknown
  - Without further knowledge, we assume no time is more likely than any other

→ Uniform(1.0, 2.0)

Exponential distribution

- In general, it is unreasonable to assume that all possible values are equally likely
- Frequently, small values are more likely than large values
- We need a non-linear transformation that maps $0.0 \rightarrow 1.0$ to $0.0 \rightarrow \infty$

this is the most frequently used function
$\mu > 0$ is a parameter that "control" the frequency of large values in respect of the small ones



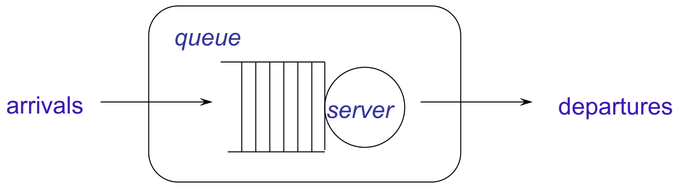
- the transformation is monotone increasing, one-to-one

$$
\begin{aligned}
0 < u < 1 &\Leftrightarrow 0 < (1 - u) < 1 \\
&\Leftrightarrow -\infty < \ln(1 - u) < 0 \\
&\Leftrightarrow 0 < -\mu \ln(1 - u) < \infty \\
&\Leftrightarrow 0 < x < \infty
\end{aligned}
$$

```
double Exponential(double μ)          /* use μ > 0.0 */
{
  return (-μ * log(1.0 - Random()));
}
```

- the parameter μ specifies the sample mean

Single Server Queue



Arrival times: a_i

- use the exponential function for the interarrival times

$a_i = a_{i-1} + \mathit{Exponential}(\mu)$; $i = 1, 2, 3, \ldots, n$

Service times: s_i

Uniform(1.0, 2.0)

- program ssq2 is an extension of ssq1
  - arrival times are drawn from *Exponential*(2.0)
  - service times are drawn from *Uniform*(1.0, 2.0)

trace-driven simulation

```
#include <stdio.h>

#define FILENAME   "ssq1.dat”    /* input data file */
#define START      0.0

  double GetArrival(FILE *fp)   /* read an arrival time */
{   double a;
    fscanf(fp, "%lf", &a);
    return (a);}

double GetService(FILE *fp)   /* read a service time */
{   double s;
    fscanf(fp, "%lf\n", &s);
    return (s);}
```

ssq2.c distribution-driven simulation

```
#include <stdio.h>
#include <math.h>
#include "rng.h”
#define LAST          10000L   /* number of jobs processed */
#define START         0.0

double Exponential(double m)                    /* ------*
{return (-m * log(1.0 - Random())); }             m > 0.0
                                                  ------ */
double Uniform(double a, double b)              /* ------*
{return (a + (b - a) * Random());    }              a < b
                                                * ------*/

 double GetArrival(void)
{static double arrival = START;
 arrival += Exponential(2.0);
 return (arrival);}

double GetService(void)
 {return (Uniform(1.0, 2.0));}
```


- the program generates all first-order statistics

$$\bar{r}, \bar{w}, \bar{d}, \bar{s}, \bar{l}, \bar{q}, \bar{x}$$

trace-driven simulation = distribution-driven simulation

```
int main(void)
{  FILE   *fp;                    /* input data file */
   long   index     = 0;         /* job index */
   double arrival   = START;     /* arrival time*/
   double delay;                 /* delay in queue*/
   double service;                /* service time*/
   double wait;                   /* delay + service*/
       double departure = START;     /* departure time*/
   struct {                       /* sum of ...   */
       double delay;      /*delay times */
       double wait;       /*wait times*/
       double service;    /*service times */
       double interarrival; /*   interarrival times */
   } sum = {0.0, 0.0, 0.0};
```


11

trace-driven simulation

```
fp = fopen(FILENAME, "r");
if (fp == NULL) {
   fprintf(stderr,"Cannot open input file %s\n",FILENAME);
   return (1);  }
while (!feof(fp)) {
```

distribution-driven simulation

```
PutSeed(123456789);
while (index < LAST) {
```


12

```
                     trace-driven: fp
while (index < LAST) {
  index++;
  arrival      = GetArrival();
  if (arrival < departure)
     delay  = departure - arrival; /* delay in queue  */
  else  delay      = 0.0;            /* no delay  */
  service = GetService();
  wait  = delay + service;
  departure    = arrival + wait;   /* time of departure */
  sum.delay   += delay;
  sum.wait    += wait;
  sum.service += service;
}
sum.interarrival = arrival - START;}
```



```
printf("\nfor %ld jobs\n", index);
printf("   average interarrival time = %6.2f\n", sum.interarrival / index);
printf("   average wait ............ = %6.2f\n", sum.wait / index);
printf("   average delay ........... = %6.2f\n", sum.delay / index);
printf("   average service time .... = %6.2f\n", sum.service / index);
printf("   average # in the node ... = %6.2f\n", sum.wait / departure);
printf("   average # in the queue .. = %6.2f\n", sum.delay / departure);
printf("   utilization ............. = %6.2f\n", sum.service / departure);
return (0);}
```

Example 1

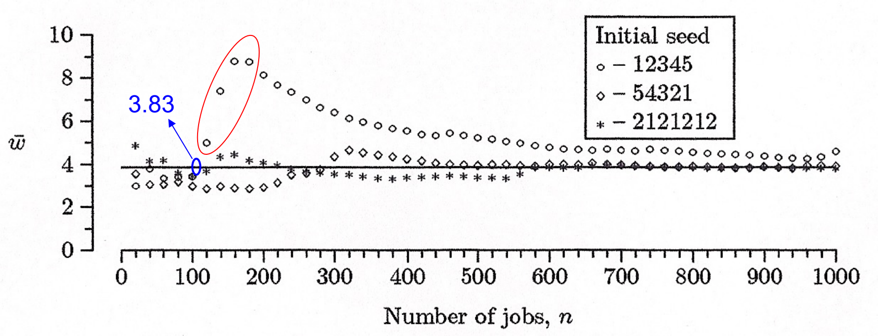
- The "theoretical" averages using *Exponential*(2.0) (rate 0.5 j/s) arrivals and *Uniform*(1.0, 2.0) (rate 0.67) service times are

$\bar{r}$	$\bar{w}$	$\bar{d}$	$\bar{s}$	$\bar{l}$	$\bar{q}$	$\bar{x}$
2.00	3.83	2.33	1.50	1.92	1.17	0.75

exact analytical results, No simulation!

- Although the server is busy 75% of the time, on average there are approximately 2 jobs in the service node
- A job can expect to spend more time in the queue than in service
- To achieve these averages, <u>many</u> jobs must pass through node

- The accumulated average wait was printed every 20 job



- The convergence of w is slow, erratic, and dependent on the initial seed

Discrete-Event Simulation
case study ssq

- the program can be used to study the *steady-state* behavior
  - Will the statistics converge independent of the initial seed?
  - How many jobs does it take to achieve steady-state behavior?
- the program can be used to study the *transient* behavior
  - Fix the number of jobs processed and replicate the program with the initial state fixed
  - Each replication uses a different initial rng seed

17

Steady-state analysis

Discrete-Event Simulation
case study ssq

	$\bar{r}$	$\bar{w}$	$\bar{d}$	$\bar{s}$	$\bar{l}$	$\bar{q}$	$\bar{x}$	
theoretical	2.00	3.83	2.33	1.50	1.92	1.17	0.75	seed=123456789
n=10	2.85	1.74	0.39	1.35	0.59	0.13	0.45	seed=123456789
n=100	2.06	3.16	1.67	1.48	1.50	0.80	0.71	seed=123456789
n=1000	2.03	3.44	1.94	1.50	1.69	0.96	0.74	seed=123456789
n=10000	2.02	3.86	2.36	1.50	1.91	1.17	0.74	seed=123456789
n=100000	2.00	3.85	2.35	1.50	1.92	1.17	0.75	seed=123456789
n=1000000	2.00	3.81	2.31	1.50	1.90	1.15	0.75	seed=123456789
n=1000000	2.00	3.84	2.34	1.50	1.92	1.17	0.75	seed=1
n=10	2.13	2.36	0.75	1.62	1.02	0.32	0.69	seed=1
n=10	1.48	2.37	0.87	1.50	1.24	0.46	0.79	seed=987654321
n=10	1.49	1.89	0.49	1.40	1.12	0.29	0.83	seed=2121212121

Transient analysis

18

Simulating an initial steady state

$\bar{d} = 2.33$

`departure=3`

$a_1=a_0+\text{expo}(2)=0+0.8=0.8$

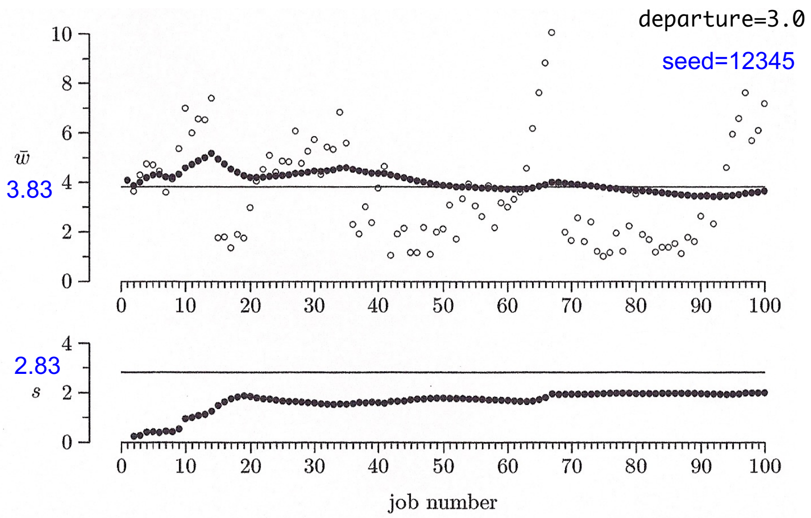
$0.8 < 3$

$d_1=3-0.8=2.2$

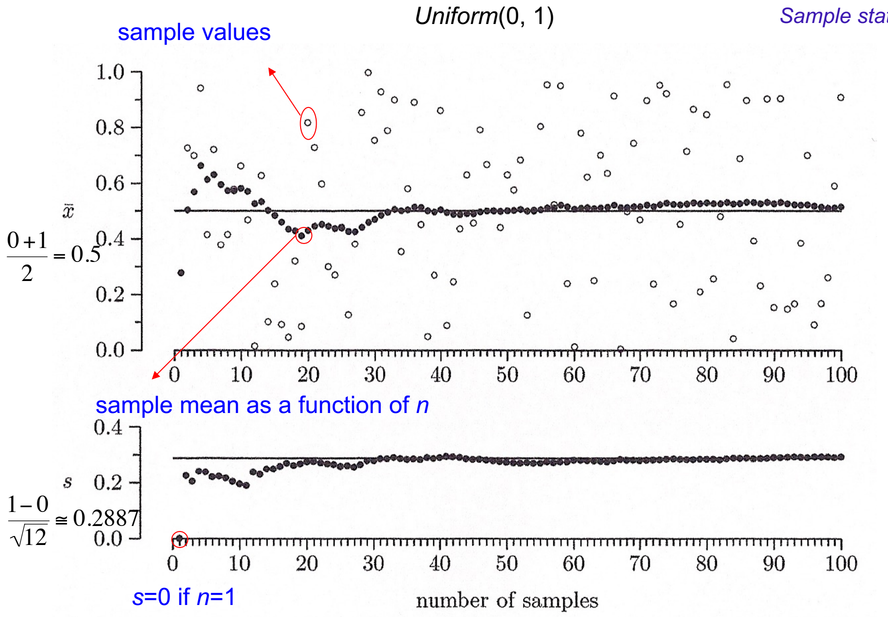
$s_1=\text{Uniform}(1,2)=1.3$

$w_1=2.2+1.3=3.5$

$c_1=0.8+3.5=4.3$



DE simulation
Sample statistics



DE simulation
Sample statistics

Serial correlation

- *Independence*: each x_i value does not depend on any other point
- Time-sequenced DES output is typically not independent
- E.g.: wait times of consecutive jobs have positive *serial correlation*
- Example: Consider output from ssq2
  - *Exponential*(2) interarrivals, *Uniform*(1,2) service
- wait times $w_1, w_2, \ldots, w_{100}$, have high positive serial correlation
  - The correlation produces a bias in the standard deviation

Example 2

- assume that jobs arrive at random with a steady-state arrival rate of 0.5 jobs per minute
- assume that Job service times are “composite” with two components:
  - the *number* of service tasks is 1 + *Geometric*(0.9)
  - the *time* (in minutes) per task is *Uniform*(0.1, 0.2)

```
double GetService(void)
{
long k;
double sum = 0.0;
long tasks = 1 + Geometric(0.9);
for (k = 0; k < tasks; k++)
    sum += Uniform(0.1, 0.2);
return (sum);
}
```


- The theoretical steady-state statistics for this model are

$\bar{r}$	$\bar{w}$	$\bar{d}$	$\bar{s}$	$\bar{l}$	$\bar{q}$	$\bar{x}$	
2.00	5.77	4.27	1.50	2.89	2.14	0.75	exact analytical results, No simulation!
	(3.83	2.33		1.92	1.17)		

- The arrival rate, service rate, and utilization are identical to the previous case (example 1)
- The other four statistics are significantly larger
- performance measures are sensitive to the choice of service time distribution

A simple inventory system

- Distributes items from current inventory to customers
- Customer demand is discrete
- Simple: one type of item

– Inventory review is periodic
– Items are ordered, <u>if necessary</u>, only at review times
– (s, S) are the min,max inventory levels, 0≤s<S

25

Simply Inventory System

- Program sis2 has randomly generated demands using an *Equilikely*(a, b) random variate
- Using random data, we can study transient and steady-state behaviors

sis1.c

```
#include <stdio.h>

#define FILENAME   "sis1.dat"
#define MINIMUM    20
#define MAXIMUM    80
#define STOP      100
#define sqr(x)     ((x) * (x))

long GetDemand(FILE *fp)
{
    long d;
    fscanf(fp, "%ld¥n", &d);
    return (d);}
```


26

sis2.c

```
#include <stdio.h>
#include "rng.h"

#define MINIMUM   20
#define MAXIMUM   80
#define STOP     100      /* 100 weeks = about 2 years*/
#define sqr(x)    ((x) * (x))


long Equilikely(long a, long b)
{  return (a + (long) ((b - a + 1) * Random()));}

long GetDemand(void)
{
        return (Equilikely(10, 50)); }
```

i	:	1	2	3	4	5	6	7	8	9	10	11	12
d_i	:	30	15	25	15	45	30	25	15	20	35	20	30

27

```
int main(void)
{
  long index     = 0;
  long inventory = MAXIMUM;
  long demand;
  long order;
  struct {
      double setup;
      double holding;  /*inventory hold (+) */
      double shortage; /*inventory short (-) */
      double order;
      double demand;
  } sum = { 0.0, 0.0, 0.0, 0.0, 0.0 };

   PutSeed(123456789);
```

Prof. Vittoria de Nitto Personè 28

28

```
while (index < STOP) {
  index++;
  if (inventory < MINIMUM) {
      order = MAXIMUM - inventory;
      sum.setup++;
      sum.order    += order;
  }
  else order         = 0;
  inventory += order;     /* there is no delivery lag */    demand  =
  GetDemand();
  sum.demand     += demand;
  if (inventory > demand)
      sum.holding  += (inventory - 0.5 * demand);
  else {
      sum.holding  += sqr(inventory) / (2.0 * demand);
      sum.shortage += sqr(demand - inventory) / (2.0 * demand);
  }
  inventory      -= demand;
}
```



```
if (inventory < MAXIMUM) {
      order = MAXIMUM - inventory;
      sum.setup++;
      sum.order+= order;
      inventory      += order;
  }

...
```

for 100 time intervals with an average demand of 27.68
and policy parameters (s, S) = (20, 80)

average order .... = 27.68
setup frequency .... = 0.36
average holding level .... = 44.81
average shortage level ... = 0.14

- trace file sis1.dat contains data for n=100 time intervals
- with (s, S)=(20, 80)

$\bar{o} = \bar{d} = 29.29 \qquad \bar{u} = 0.39 \qquad \bar{l}^+ = 42.40 \qquad \bar{l}^- = 0.25$


```c
int main(void)
{  long seed;
   long index     = 0;
   long inventory = MAXIMUM;
   long demand;
   long order;
   struct {
       double setup;
       double holding;  /*inventory held (+) */
       double shortage; /*   inventory short (-)   */
       double order;
       double demand;
   } sum = { 0.0, 0.0, 0.0, 0.0, 0.0 };

PutSeed(-1);
GetSeed(&seed);
printf(”\nwith an initial seed of %ld", seed);
```

for 100 time intervals with an average demand of 27.68
and policy parameters (s, S) = (20, 80)

average order .... = 27.68
setup frequency .... = 0.36
average holding level .... = 44.81
average shortage level ... = 0.14

with an initial seed of 1333437895
for 100 time intervals with an average demand of 31.00
and policy parameters (s, S) = (20, 80)

average order .... = 31.00
setup frequency .... = 0.40
average holding level .... = 43.39
average shortage level ... = 0.37

Simply Inventory System

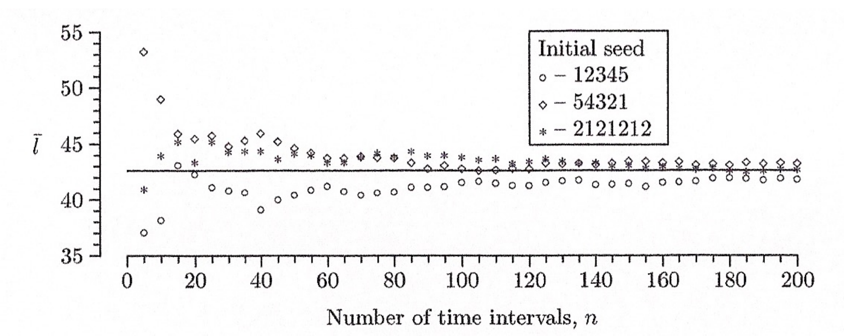
if (a, b) = (10, 50) and (s, S) = (20, 80), then the approximate steady-state satistics are

$\bar{d}$	$\bar{o}$	$\bar{u}$	$\bar{l}^+$	$\bar{l}^-$
30.00	30.00	0.39	42.86	0.26

(trace-driven

$\bar{o} = \bar{d} = 29.29$ $\quad \bar{u} = 0.39$ $\quad \bar{l}^+ = 42.40$ $\quad \bar{l}^- = 0.25$)

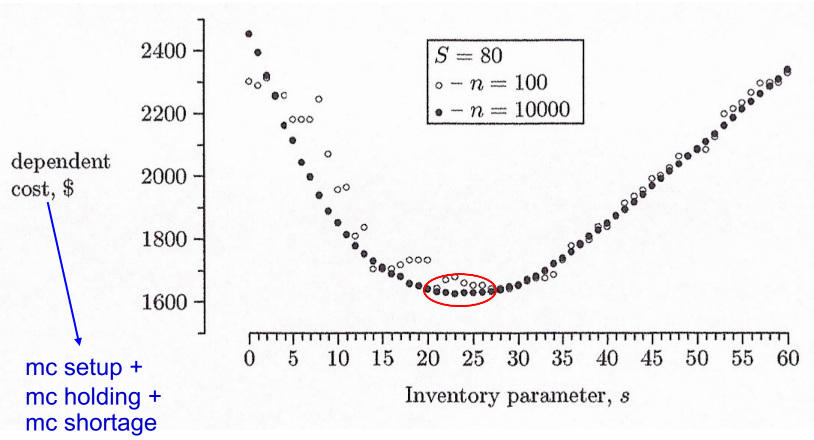
The average inventory level $\bar{l} = \bar{l}^+ - \bar{l}^-$
approaches steady state after several hundred time intervals



Convergence is slow, erratic, and dependent on the initial seed

- using a fixed initial seed guarantees the <u>exact</u> same demand sequence (in the example 12345)
  - any changes to the system are caused solely by the change of s
  - a steady state study of this system is unreasonable:
    - all parameters would have to remain fixed for many years
    - when n=100, we simulate approximately 2 years
    - when n=10000, we simulate approximately 192 years

- if we fix S, we can find the optimal cost by varying s
- recall that the dependent cost ignores the fixed cost of each item



Statistical Considerations
(sect. 3.1.3)

- *variance reduction:*

  (intuitive approach) use the same random numbers

  (We kept the same initial seed and changed only s)

NOTE:
transient behavior will always have some inherent uncertainty

- *Robust Estimation*:

  when a data point is not sensitive to small changes in assumptions

  - values of s close to 23 have essentially the same cost
  - Would the cost be more sensitive to changes in S or other assumed values?

Exercises

- derive analytical results on p.12
- study program ssq2.c; run it and compare output with the results on p.12
- Exercises: 3.1.1, 3.1.2, 3.1.4, 3.1.5, 3.1.6