



## Performance Modeling of Computer Systems and Networks

*Prof. Vittoria de Nitto Personè*

Next Event Simulation  
Examples

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

Copyright © Vittoria de Nitto Personè, 2021

<https://creativecommons.org/licenses/by-nc-nd/4.0/>



(CC BY-NC-ND 4.0)

1

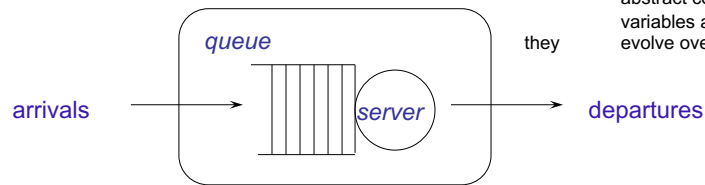
1. **Initialize** - set simulation clock and first time of occurrence for each event type
2. **Process current event** - scan event list to determine most imminent event; advance simulation clock; update state
3. **Schedule new events** - new events (if any) are placed in the event list
4. **Terminate** - Continue advancing the clock and handling events until termination condition is satisfied

Prof. Vittoria de Nitto Personè

2

2

## Single Server Queue



- **Conceptual model:**  
abstract collection of  
variables and how  
they evolve over time

- The state is number of jobs in the node at time  $t$ :  $l(t)$
- Its time-evolution is guided by arrival-departure events:
  - An arrival causes  $l(t)$  to increase by 1
  - A departure causes  $l(t)$  to decrease by 1

Prof. Vittoria de Nitto Personè

3

3

## Single Server Queue

- **Specification model:**  
collection of  
mathematical  
variables together  
with logic and  
equations

The state variable  $l(t)$  provides a complete characterization of the state of a ssq

$$\begin{aligned} l(t) = 0 &\Leftrightarrow q(t) = 0 \text{ and } x(t) = 0 \\ l(t) > 0 &\Leftrightarrow q(t) = l(t) - 1 \text{ and } x(t) = 1 \end{aligned}$$

Da  $l(t)$  posso effettivamente definire tutto.

Tutto ciò che è inutile/superfluo non va messo.

Posso scrivere queste formule solo se capisco bene il modello che sto definendo, se non me ne rendo conto allo 'step' prima posso tornare indietro e vedere cosa manca, ma idealmente a questo passo ci arrivo solo dopo aver catturato tutto.

Prof. Vittoria de Nitto Personè

4

4

$I(t)$  ci dice tutto, non ci serve altro.

- The initial state  $I(0)$  can have any non-negative value, typically 0. Ha senso partire da 0, se parto dal sistema vuoto. Se partissi da un sistema già stabile, potrei usare altri valori.
- terminal state: any non-negative value
  - Assume at time  $\tau$  arrival process stopped. Remaining jobs processed before termination
- some mechanism must be used to denote an event impossible
  - Only store possible events in event list
  - Denote impossible events with event time of  $\infty$


se volessi tornare allo stato iniziale, fisso un tempo 'tau' che, superato blocca i nuovi arrivi e fa servire gli ultimi job arrivati in ritardo.

13/04/2023

- The simulation clock (current time) is  $t$ . NextEvent guidata dal tempo, come detto è asincrono.
- The terminal ("close the door") time is  $\tau$ . istante terminale
- The next scheduled arrival time is  $t_a$
- The next scheduled service completion time is  $t_c$
- The number in the node (state variable) is  $I$  (in funzione di  $t$ )

## Next-Event Simulation

### Algorithm

1. **Initialize:** the clock  
the event list (e.g. ssq arrival)  
the system state
  2. **Remove** next event from the list
  3. **Advance** simulation clock
  4. **Process** current event
  5. **Schedule** new events (if any) generated from current event
  6. Go to 2. until **termination** condition is satisfied
- 

Nella coda singola, ad esempio, inizializziamo  $l(t=0) = 0$ , ovvero all'inizio non c'è nessuno.  
 5) ad esempio, se arriva un job e va nella coda, non schedulo il suo completamento, quello viene schedulato quando entra in servizio.  
 Se parto da centro vuoto vado subito in servizio, e quindi metto subito tempo completamento.

Prof. Vittoria de Nitto Personè

7

7

### ssq2.c

vecchio codice

```
int main(void)
{ long index = 0; /* job index */ conta i job
  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};
  PutSeed(123456789);
```

Prof. Vittoria de Nitto Personè

8

8

```
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; }
...
```

questa è la parte che cambia.

essendo sistema vuoto  
non posso mettere nella  
lista degli eventi un istante  
di completamento

continuo finchè non  
supero tau, oppure  
c'è qualcuno nella coda  
(che potrebbe esserci  
dopo tau)

il next event è  
quello più vicino a  
me:  
completamento o  
arrivo di un job

```
l = 0; t = 0.0;  inizializzazione del sistema.
t_c = ∞; t_a = GetArrival(); /* initialize the event list */
while ((t_a < τ) or (l > 0)) {
  t = min(t_a, t_c); /* scan the event list */
  if (t == t_a) { /* process an arrival */
    l++;
    t_a = GetArrival();
    if (t_a > τ)
      t_a = ∞;
    if (l == 1)
      t_c = t + GetService();
  }
  else { /* process a completion */
    l--;
    if (l > 0)
      t_c = t + GetService();
    else
      t_c = ∞;
  }
}
```

2 event types: arrival,  
departure

Algorithm 1

se il prossimo tempo è quello di completamento, decremento la popolazione,  
se c'è ancora qualcuno nel centro, devo generare il suo servizio, e quindi il suo istante di completamento. Se coda è vuota ( $l == 0$ ), prossimo tempo di completamento è infinito, perché non c'è nessuno.

"algoritmo" che gestisce gli arrivi, se c'è un arrivo incremento "l", (qui siamo in coda infinita), genero prossimo evento arrivo, sennò il programma terminerebbe. Controllo se sono oltre la simulazione, se eccedo il tempo tau allora posto il tutto a infinito, se partivo da sistema vuoto,  $l == 1$ , allora job sta per forza entrando in servizio, e quindi genero un evento di un altro tipo (il servizio), oltre a definire il prossimo tempo di completamento. NON SO ANCORA SE VIENE PRIMA DEL PROSSIMO  $t(a)$ .

Sopra avevamo uno pseudocodice, ora passiamo a qualcosa di più specifico.

Next-Event simulation  
ssq  
*computational model*

### Program ssq3

- `number` represents  $l(t)$  (system state)
- `struct t` represents time
  - `t.arrival, t.completion` event list  
( $t_a, t_c$  from algorithm 1)
  - `t.current` simulation clock ( $t$  from algorithm 1)
  - `t.next` next event time ( $\min(t_a, t_c)$  from algorithm 1)
  - `t.last` last arrival time
- `struct area` (time-averaged) statistics-gathering structure
  - $\int_0^t l(s) ds$  evaluated as `area.node`
  - $\int_0^t q(s) ds$  evaluated as `area.queue`
  - $\int_0^t x(s) ds$  evaluated as `area.service`

Prof. Vittoria de Nitto Personè

11

11

Next-Event simulation  
ssq

### ssq3.c NextEvent

```

#include <stdio.h>
#include <math.h>
#include "rngs.h" /* the multi-stream generator */
#define START 0.0
#define STOP 20000.0 /* terminal (close the door) time*/
#define INFINITY (100.0 * STOP) /* must be much larger than STOP */

double Min(double a, double c) /* semplice funzione che calcola il minimo.
{ if (a < c) return (a);
  else return (c);}

double Exponential(double m) ...
double Uniform(double a, double b) ... /* codice invariato
double GetArrival() ...
double GetService() ...
  
```

Prof. Vittoria de Nitto Personè

12

12

```

int main(void)
{ struct {
    double arrival; /* next arrival time */
    double completion; /* next completion time */
    double current; the clock! rrent time */
    double next; /* next (most imminent) event time */
    double last; /* last arrival time */
} t;
struct {
    double node; /* time integrated number in the node */
    double queue; /* time integrated number in the queue */
    double service; /* time integrated number in service */
} area = {0.0, 0.0, 0.0};
long index = 0; /* used to count departed job */
long number = 0; /* number in the node */ system state

```

number = numero job nel centro, index conta i job.  
Number è la nuova variabile introdotta.

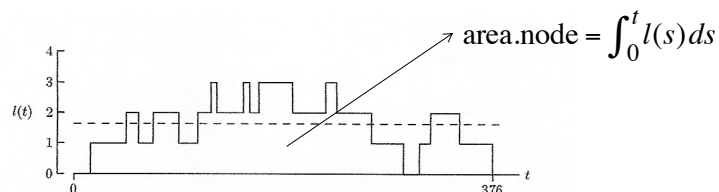
13

'traduzione' in C dello pseudocodice visto prima.

```

PlantSeeds(123456789);
t.current = START; /* set the clock */
t.arrival = GetArrival(); /* schedule the first arrival */
t.completion = INFINITY; /* the first event can't be a completion */
while ((t.arrival < STOP) || (number > 0)) {
    t.next = Min(t.arrival, t.completion); /* next event time */
    if (number > 0) { /* update integrals */
        area.node += (t.next - t.current) * number;
        area.queue += (t.next - t.current) * (number - 1);
        area.service += (t.next - t.current);
    }
    t.current = t.next; advance the clock!
}

```



14

Le varie 'area' lo calcolo come un integrale, ma essendo gradino è abbastanza semplice.  
Moltiplico "number" (la y) per quanto tempo è stato in quello stato (la x).

```

if (t.current == t.arrival) {
    number++;
    t.arrival= GetArrival();
    if (t.arrival > STOP) {
        t.last= t.current;
        t.arrival = INFINITY;
    }
    if (number == 1)
        t.completion = t.current + GetService();
}
else {
    index++;
    number--;
    if (number > 0)
        t.completion = t.current + GetService();
    else
        t.completion = INFINITY;
}
}

```

process an arrival

traduzione in C dello pseudocodice

process a completion

```

printf(" ... jobs", index);
printf(" average interarrival time ..", t.last / index);
printf(" average wait ...", area.node / index);
printf(" average delay ...", area.queue / index);
printf(" average service time ...", area.service / index);
printf(" average # in the node ... ", area.node / t.current);
printf(" average # in the queue .. ", area.queue / t.current);
printf(" utilization ....", area.service / t.current);

```

t.current alla fine sarà l'ultimo  
istante della simulazione



ssq2 ed ssq3, anche se basate su idee diverse, devono produrre stesse statistiche!  
Con 'ssq2' accumulavamo quelle sommatorie.

## World Views and Synchronization

### • ssq2 produces :

```
while (index < LAST) {
  index++;
  arrival = GetArrival();
  if (arrival < departure)
    delay = departure - arrival;
  else delay = 0.0;
  service = GetService();
  wait = delay + service;
  departure = arrival + wait;
  sum.delay += delay;
  sum.wait += wait;
  sum.service += service; }
```

Diagram illustrating the accumulation of statistics:

- $\sum_{i=1}^n d_i$  (Sum of delays)
- $\sum_{i=1}^n w_i$  (Sum of wait times)
- $\sum_{i=1}^n s_i$  (Sum of service times)

Prof. Vittoria de Nitto Personè

17

17

## World Views and Synchronization

### • ssq2 produces :

```
printf("... jobs", index);
printf("average interarrival time = ", sum.interarrival / index);
printf("average wait ..... = ", sum.wait / index);
printf("average delay ..... = ", sum.delay / index);
printf("average service time .... = ", sum.service / index);
printf("average # in the node ... = ", sum.wait / departure);
printf("average # in the queue .. = ", sum.delay / departure);
printf("utilization ..... = ", sum.service / departure);
```

Diagram illustrating the calculation of statistics:

- $\sum_{i=1}^n w_i$  (Sum of wait times)
- $\bar{w} = \frac{1}{n} \sum_{i=1}^n w_i$  (Average wait time)
- $\bar{l} = \frac{n}{c_n} \bar{w}$  (Average number in the node)
- $\bar{q} = \frac{n}{c_n} \bar{d}$  (Average number in the queue)
- $\bar{x} = \frac{n}{c_n} \bar{s}$  (Average number in the system)

$$\frac{\sum_{i=1}^n w_i}{c_n} = \frac{n \bar{w}}{c_n}$$

Prof. Vittoria de Nitto Personè

18

18

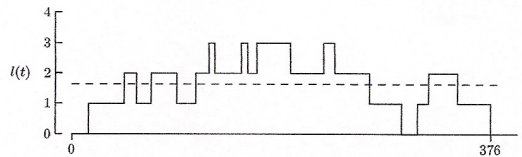
ssq3 fa il contrario, calcola le POPOLAZIONI, non i tempi, e dopo aver calcolato le aree divide per il tempo, per calcolare le medie.

Next-Event simulation  
ssq

• ssq3 produces :

```
PlantSeeds(123456789);
t.current = START;
t.arrival = GetArrival();
t.completion = INFINITY;
while ((t.arrival < STOP) || (number > 0)) {
    t.next= Min(t.arrival, t.completion);
    if (number > 0) {
        area.node+= (t.next - t.current) * number;
        area.queue+= (t.next - t.current) * (number - 1);
        area.service += (t.next - t.current);
    }
    t.current = t.next;
}
```

$\int_0^{\tau} l(t)dt \leftarrow$   
 $\int_0^{\tau} q(t)dt \leftarrow$   
 $\int_0^{\tau} x(t)dt \leftarrow$



Prof. Vittoria de Nitto Personè

19

19

Next-Event simulation  
ssq

$$\tau \bar{l} = \int_0^{\tau} l(t)dt$$

```
printf(" ... jobs", index);
printf(" average interarrival time ..", t.last / index);
printf(" average wait ...", area.node / index);
printf(" average delay ...", area.queue / index);
printf(" average service time ...", area.service / index);
printf(" average # in the node ... ", area.node / t.current);
```

$\bar{w} = \frac{\tau}{n} \bar{l}$

$$\bar{l} = \frac{1}{\tau} \int_0^{\tau} l(t)dt$$

```
printf(" average # in the queue .. ", area.queue / t.current);
printf(" utilization ....", area.service / t.current);
```

Prof. Vittoria de Nitto Personè

20

20

## 21

## 22

## World Views and Synchronization

- in ssq3 the order **cannot be known a priori**

```
while ((ta < τ) or (l > 0)) {
  t = min(ta, tc); /* scan the event list */
  if (t == ta) { /* process an arrival */
    l++;
    ta = GetArrival();
    if (ta > τ)
      ta = ∞;
    if (l == 1)
      tc = t + GetService();
  }
  else { /* process a completion */
    l--;
    if (l > 0)
      tc = t + GetService();
  }
  ....
}
```

genero all'inizio t(a) e t(c),  
perché le inizializzo.  
Dopo l'ordine dipende da chi è  
il minimo, quindi potrei  
generare più arrivi prima di  
produrre un tempo di servizio.

## World Views and Synchronization

The programs should produce exactly the same statistics solo se uso il multistream

quindi uno stream per gli arrivi, ed uno per i servizi. → to do so requires rngs

```
double GetArrival()
{ static double arrival = START;
  SelectStream(0);
  arrival += Exponential(2.0);
  return (arrival);}

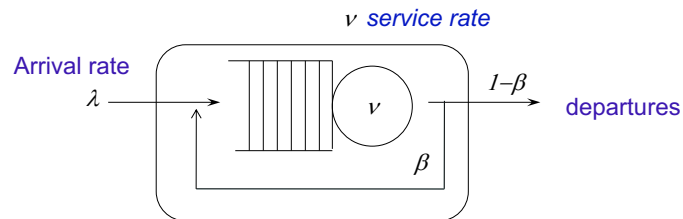
double GetService()
{ SelectStream(1);
  return (Uniform(0.0, 1.5)+Uniform(0.0, 1.5));}

con stesso stream non potrei analizzare come cambia la varianza.
```

## Model Extensions

Next-Event simulation  
ssq with feedback

### SSQ with immediate feedback



job index	1	2	3	4	5	6	7	8	9	...			
Arrival/feedback	1	3	4	7	10	13	14	15	19	24	26	30	...
service	9	3	2	4	7	5	6	3	4	6	3	7	...
departure	10	13	15	19	26	31	37	40	44	50	53	60	...

con l'approccio di prima, senza modellazione del tempo, avevo difficoltà a capire dove inserire un completamento che diventava feedback. Infatti dovevo mantenere questi feedback in 'attesa' aspettando che si creassero le condizioni idonee per inserirli.

Prof. Vittoria de Nitto Personè

25

25

## Model Extensions

Next-Event simulation  
ssq with feedback

### SSQ with immediate feedback

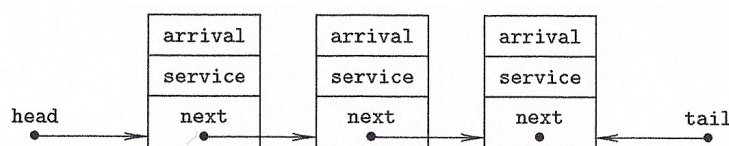
Con NextEvent è più semplice.

```

else {
    /* process a completion */
    if (GetFeedback() == 0) { /* this statement is new */
        index++;
        number--;
    }
    if (number > 0)
        t.completion = t.current + GetService();
    else t.completion = INFINITY;

```

- alternate queue disciplines
- it is necessary to add a dynamic-queue data structure



- +2 supporting queue functions: Enqueue, Dequeue

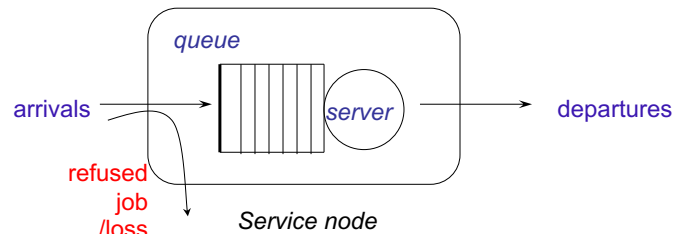
Prof. Vittoria de Nitto Personè

26

controllo se, completato un job, esso ha feedback o meno. Se non ha feedback, "index ++" perché un job esce, "number --" perché nel centro c'è un job in meno.

26

## Model Extensions



Assumption:  
for the finite *capacity* case,  
only the accepted jobs are considered

BUT  
the *loss probability* could be of interest!

## Model Extensions

### SSQ with finite capacity

```

if (t.current == t.arrival) {           /* process an arrival */

    if (number < CAPACITY) {
        number++;
        if (number == 1)
            t.completion = t.current + GetService();
    }
    else
        reject++;
    t.arrival = GetArrival();
    if (t.arrival > STOP) {
        t.last = t.current;
        t.arrival = INFINITY;
    }
}

```

se c'è spazio,  
incremento numero nodi nel centro  
se c'è solo un job, genero tempo di servizio.

altrimenti:  
incremento contatore di job rigettati,  
poi la dividerò per index (tutti i job trattati)  
e vedo probabilità di perdita.

## Random Sampling

- The structure of ssq3 facilitates adding sampling
- Add a sampling event to the event list
  - Sample deterministically, every  $\delta$  time units
  - Sample Randomly, every *Exponential*( $\delta$ ) time units

Adesso ci interessiamo al campionamento, introducendo l'evento di campionamento. Esso può essere deterministico (ogni  $x$  job, ogni  $y$  job), oppure randomicamente, che vuol dire Esponenziale. Perché il tempo che intercorre tra due eventi è esponenziale, essendo random.

Prof. Vittoria de Nitto Personè

29

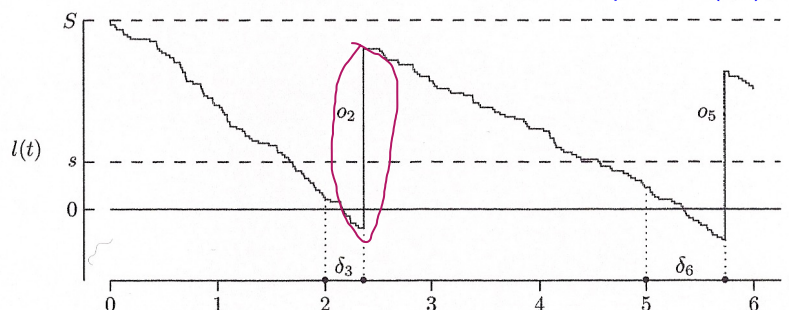
29

## A Simple Inventory System with Delivery Lag

Unif(0,1) perchè gli ordini mi devono arrivare entro la prossima settimana

Two changes relative to sis2

- *Uniform*(0,1) lag between inventory review and order delivery
- More realistic demand model
  - Demand instances for a single item occur *at random*
  - Average rate is  $\lambda$  demand instances per time interval
  - Time between demand instances is *Exponential*( $1/\lambda$ )



qui istantaneamente tornavo alla capacità richiesta.

Prof. Vittoria de Nitto Personè

30

30

Precedentemente, assumevano che le domande arrivassero spalmate sul tempo (7 domande in 7 giorni? 1 al giorno), ma non è detto sia per forza così. Per questo generiamo una esponenziale di media  $1/\lambda$  (inverso frequenza), che ci genera istanti di arrivo delle domande, e come vediamo in figura sono spazati in modo diverso, in modo random. LA MEDIA, RISPETTO A PRIMA, NON CAMBIA.

La domanda è sempre "unitaria", però arrivi in modo random nell'intervallo di tempo (cioè la settimana), con frequenza media  $\lambda$ .

## Comparison of Demand Models

sis2: used an **aggregate demand** for each time interval, generated as an *Equilike*(10,50) random variate

- Aggregate demand per time interval is random
- Within an interval, time between demand instances is constant
- Example: if aggregate demand is 25, inter-demand time is  $1/25=0.04$  se domanda è 25, allora tempo tra domande è sempre  $1/25$
- Now using *Exponential*( $1/\lambda$ ) inter-demand times
  - Demand is modeled as an arrival process
  - Average demand per time interval is  $\lambda$   
domande generate come un processo di arrivo stocastico.

## Specification Level: States and Notation

- The simulation clock is  $t$  (real-valued)
- The terminal time is  $\tau$  (integer-valued)
- Current inventory level is  $I(t)$  (integer-valued)
- Amount of inventory on order, if any, is  $o(t)$  (integer-valued)
  - Necessary due to delivery lag
- **$I(t)$  and  $o(t)$  provide complete state description** livello inventario, ordini, ad un certo istante di tempo.
- Initial state is assumed to be  $I(0)=S$  and  $o(0)=0$
- Terminal state is assumed to be  $I(\tau)=S$  and  $o(\tau)=0$
- **Cost to bring  $I(t)$  to  $S$  at simulation end (with no lag) must be included in accumulated statistics**



## Specification Level: Events

Three types of events can change the system state

- A *demand* for an item at time  $t$ 
  - $l(t)$  decreases by 1

richiesta di 1 articolo, che porta  $l(t)$  ad essere decrementata.
- An *inventory review* at integer-valued time  $t$ 
  - If  $l(t) \geq s \rightarrow o(t)=0$
  - If  $l(t) < s \rightarrow o(t)=s-l(t)$

a seconda delle scorte, definisco la quantità degli ordini.
- An *arrival* of an inventory replenishment order at time  $t$ 
  - $l(t)$  increases by  $o(t)$
  - $o(t)$  becomes 0

terzo evento è l'arrivo dell'ordine,  $o(t)$  incrementa le scorte, e poi  $o(t)$  diventa 0 perchè ho messo in magazzino tutti gli arrivi.

Prof. Vittoria de Nitto Personè

33

33

## Algorithm 2: initialization

Time variables used for event list:

- $t_d$ : next scheduled inventory *demand*
- $t_r$ : next scheduled inventory *review*
- $t_a$ : next scheduled inventory *arrival*

$\infty$  denotes impossible events

inizializzazione {

```

I = S;           /* initialize inventory level */
o = 0;           /* initialize amount on order */
t = 0.0;         /* initialize simulation clock */
td = GetDemand(); /* initialize event list */
tr = t + 1.0;     /* initialize event list */
ta = ∞;           /* initialize event list */

```

Prof. Vittoria de Nitto Personè

34

34

Next-Event simulation  
InvSys

### Algorithm 2: main loop

3 types of events:  
demand, review, arrival

```

while (t < τ) {
    t = min(td, tr, ta); /* scan the event list */
    if (t == td) { /* process an inventory demand */
        I--; demand
        td = GetDemand();
    }
    else if (t == tr) { /* process an inventory review */
        if (I < S) { review
            o = S - I;
            δ = GetLag();
            ta = t + δ;
        }
        tr += 1.0;
    }
    else { /* process an inventory arrival */
        I += o; arrival
        o = 0;
        ta = ∞;
    }
}
    
```

Se prossimo evento è la richiesta di un articolo, decremento gli oggetti nell'inventario, e genero la prossima domanda.

Se devo fare controllo inventario, vedo quante scorte richiedere, si genera il delivery lag, e si genera un nuovo arrivo, cioè l'arrivo dell'ordine delle merci.

se l'evento corrente è l'arrivo della merce, incremento le scorte, consumando le nuove scorte, e mettendo il prossimo arrivo di scorte a infinito, perchè non so quando avverrà.

Prof. Vittoria de Nitto Personè 35

35

Next-Event simulation  
InvSys

Scritto in C

### Program sis3

- implements algorithm 2

correspond to  $t_d, t_r, t_a$

```

while (t.current < STOP) {
    t.next = Min(t.demand, t.review, t.arrive);
    if (inventory > 0)
        sum.holding += (t.next - t.current) * inventory;
    else (inventory < 0)
        sum.shortage -= (t.next - t.current) * inventory; /* è la parte di mancanza perchè inventory è < 0 */
    t.current = t.next;
    if (t.current == t.demand) {
        sum.demand++; /* process an inventory demand */
        inventory--;
        t.demand = GetDemand();
    }
    else .....
}
    
```

Similmente procedo al calcolo dell'area, vedendo il livello di scorte per il tempo, in modo da ottenere l'holding.

Prof. Vittoria de Nitto Personè 36

36

## Program sis3

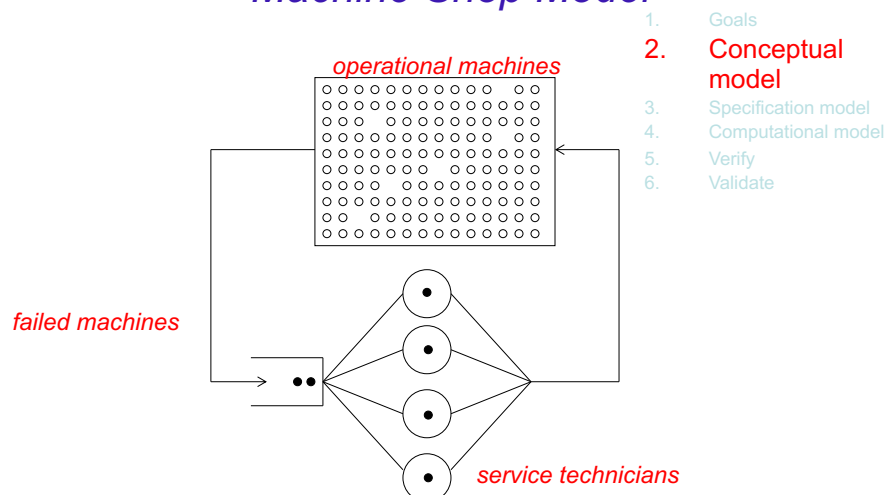
- State variables `inventory` and `order` correspond to  $l(t)$  and  $o(t)$

- `t.next`      next event instant ( $\min(t_d, t_r, t_a)$  in algorithm 2)
- `t.last`      last arrival instant

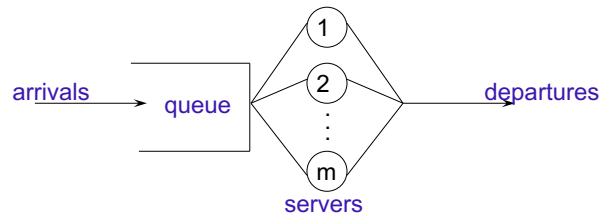
`sum.hold` and `sum.short` accumulate the time-integrated holding and shortage integrals

## Implementazione del multi server

### Machine Shop Model



## Conceptual model: MSQ



Servers in a multi-server service node are called *service channels*

- $m$  is the number of servers
- The *server index* is  $s = 1, 2, \dots, m$

The *state* includes:

the number of jobs in the node at time  $t$ :  $l(t)$

the state of each server at at time  $t$ :  $x_s(t)$

Se server fossero omogenei, non servirebbe  $x(t)$ , perché se sono tutti uguali basta sapere quanti sono e numeri di job nel sistema. Per trattare un caso più generale, usiamo però  $x(t)$ .

Prof. Vittoria de Nitto Personè

39

39

## Specification Model: States and Notation

$l(t)$  denotes the number of jobs in the service node at time  $t$

- If  $l(t) \geq m$ , all servers are busy and  $q(t) = l(t) - m$
- If  $l(t) < m$ , some servers are idle
- If servers are distinct, need to know which servers are idle

For  $s = 1, 2, \dots, m$  define

$x_s(t)$  : the number of jobs in service (0 or 1) at server  $s$  at time  $t$

ogni server o ha un job in servizio, o è idle.

The complete state description is  $l(t), x_1(t), x_2(t), \dots, x_m(t)$

$$q(t) = l(t) - \sum_{s=1}^m x_s(t)$$

job coda = job centro - serventi occupati

Prof. Vittoria de Nitto Personè

40

40

## Specification Model: Events

What types of **events** can change state variables

$l(t), x_1(t), x_2(t), \dots, x_m(t)$  ?

- **arrival at time  $t$** 
  - $l(t)$  increases by 1
  - If  $l(t) < m$ , an idle server  $s$  is selected, and  $x_s(t)$  becomes 1  
else all servers are busy
- **A completion of service by server  $s$  at time  $t$**  (scaturito da diversi server, non solo da uno)
  - $l(t)$  decreases by 1
  - if  $l(t) \geq m$ , a job is selected from the queue to enter service  
else  $x_s(t)$  becomes 0

→  $m+1$  event types

Prof. Vittoria de Nitto Personè

41

41

## Specification Model: Additional Assumptions

- The **initial state** is an empty node
  - $l(0) = 0$
  - $x_1(0) = x_2(0) = \dots = x_m(0) = 0$
  - The **first event** must be an arrival
- The arrival process is turned off at time  $\tau$ 
  - The node continues operation after time  $\tau$  until empty
  - The **terminal state** is an empty node
  - The **last event** is a completion of service

For simplicity, all servers are independent and *statistically identical*

- *Equity* selection is the server selection rule (**lowest-utilized**)

*All of these assumptions can be relaxed*

tutte queste assunzioni  
sono ipotesi, posso  
dunque cambiarle.

sistema continua per  
quei job arrivati prima di  
 $\tau$  ma che concludono  
dopo  $\tau$ , perchè devo  
svuotare il centro.

Prof. Vittoria de Nitto Personè

42

42

Solo il primo è un arrivo,  
gli altri sono tutti completamenti.  
Uso un array se m piccolo.

## Event list

t = tempo				
0	t	x	arrival	1 on, 0 off
1	t	x	completion by 1	1 busy, 0 idle
2	t	x	completion by 2	
m=4	3	t	x	completion by 3
4	t	x	completion by 4	

- can be organized as an array of  $m + 1$  event types
- field  $t$ : scheduled time of next occurrence for the event
- field  $x$ : current activity status of the event

Se parlo di server, struttura uguale, ma  $x$  assume significato diverso.

- for large  $m$  → alternate event-list structures  
userei una lista e non un array.

## Program msq

Implements this next-event multi-server service node simulation model

- number state variable  $l(t)$
- state variables  $x_1(t), x_2(t), \dots, x_m(t)$  are part of the event list
- area time-integrated statistic  $\int_0^t l(\theta) d\theta$
- sum array, records for each server
  - the sum of service times
  - the number served
- function NextEvent searches the event list to find the next event
- function FindOne searches the event list to find the longest-idle server (because equity selection is used)

## program msq.c

```

typedef struct {
    double t;
    int x;
} event_list[SERVERS + 1];  dimensione "m+1"
...

int NextEvent(event_list event)
{
    int e;
    int i = 0;
    while (event[i].x == 0)
        i++;
    e = i;
    while (i < SERVERS) {
        i++;
        if ((event[i].x == 1) && (event[i].t < event[e].t))
            e = i;
    }
    return (e);
}

```

*is the first active event, assume it is the next*

*look for the next active event*

*tempo selezionato < corrente?*  
*if it is previous, update e*

*prima cerco il primo evento ad "1", poi confronta il tempo.*

*server busy*

Prof. Vittoria de Nitto Personè

45

45

Seleziona il primo "1", se tempo simulazione non è finita, allora è l'arrivo.

Dopo confronta sui server che non sono vuoti, se l'istante di accadimento di quell'evento è minore dell'arrivo, poi continua il confronto con altri server. Perché devo trovare evento di tempo minimo.

All'inizio è sicuramente l'arrivo, perché i server stanno a 0, a 1 c'è solo processo degli arrivi.

## programma msq.c

```

int FindOne(event_list event)
{
    int s;
    int i = 1;
    while (event[i].x == 1)
        i++;
    s = i;
    while (i < SERVERS) {
        i++;
        if ((event[i].x == 0) && (event[i].t < event[s].t))
            s = i;
    }
    return (s);
}

```

*first server idle*

*look for the next idle*

*contrario di prima*

*if its completion is previous, it is idle since more time*

La findOne, similmente, cerca il primo servente libero e poi confronta su tutti quelli liberi. Cerca quello libero da più tempo, ovvero quello che dovrebbe avere utilizzazione più bassa.

Prof. Vittoria de Nitto Personè

46

46

# Exercises

- 5.1.1, 5.1.2, 5.1.3
- 5.2.1, 5.2.2,
- 5.2.8: modify program msq to allow for a finite capacity (max r jobs); a. draw a histogram of the time between lost jobs at the node; b. comment on the shape of this histogram.