NTNU | Norwegian University of Science and Technology

TPK 4191 - PRODUCTION OPTIMIZATION AND CONTROL

Discrete event simulation

Jørn Vatn/September 2024

Motivation

- There are many systems for which it is not easy to do probabilistic assessment
  - Repair times, failure times, service times etc. are not exponentially distributed
  - Resources depend on spare parts, weather window and so on
  - Prioritization rules determine which component should be repaired first
  - etc.
- Stochastic simulation is one way to establish system performance
- Discrete event simulation is a sub-set of stochastic simulation which is very flexible from a modelling point of view

What is Discrete-Event Simulation?

- In discrete-event simulation, the operation of a system is represented as a chronological sequence of events
- Each event occurs at an instant in time and marks a change of state in the system
  - For example, if the up- and down times of a system of components are simulated, an event could be "component 1" fails, and another event could be "component 2" is repaired
  - A third event could be that a customer arrives to a service desk
- Rather than explicitly assessing the performance of a system by the laws of probability, we just "simulate" the system, and calculate the relevant statistics to get the performance

Components of a Discrete-Event Simulation engine

- **Clock**: The simulation must keep track of the current simulation time. Time advances in discrete jumps, that is, the clock skips to the next event start time as the simulation proceeds
- **Event**: A change in state of a system
- **PES** = pending event set: The simulation maintains a list of simulation events to happen in the future. The pending event set is typically organized as a priority queue, sorted by event time. That is, regardless of the order in which events are added to the event set, they are removed in strictly chronological order.

Components of a DES, continued

- **Event notice**: An element in the PES describing when an event is to be executed. In our VBA the EventNotice(time,code,parameters) is used
- **Activity**: A pair of events, one initiating and the other completing an operation that transform the state of the entity. Time elapses in an activity. We usually assume that no continuous changes are taking place during the activity.
- **Random-Number Generator**
- **Statistics**: The simulation typically keeps track of the system's statistics
- **Ending Condition**

Example

Consider a system with two components

- Failure times of component 1 in days: 8, 4, 10
- Failure times of component 2 in days: 7, 6, 4
- Repair times of component 1 in hours: 4, 6, 2
- Repair times of component 1 in hours: 26, 3, 6

Visualization

- We may think of a “blackboard” as the container holding the pending event set
- We use “Post-It” sheets to hold the individual events
- The function `eventNotice()` is used to place an event to the blackboard, i.e.,
  - The timestamp for this event is given
  - An `onEvent()` function to call is given

Visualization

- We may think of a “blackboard” as the container holding the pending event set
- We use “Post-It” sheets to hold the individual events
- The function `eventNotice()` is used to place an event to the blackboard, i.e.,
  - The timestamp for this event is given
  - An `onEvent()` function to call is given
- The function `getNxtEvent()` is used to retrieve events from to the blackboard, i.e., the `onEvent()` function
- The events are retrieved in chronological order
- The clock is updated
- The `onEvent()` function is executed
- Required state variables are updated

Required built-in functions

```
def onFailure(compNo):
   compStatus[compNo] = Down
   eventNotice clock + rndRepTime(), onRepair(compNo)

def onRepair(compNo):
   compStatus[compNo] = Up
   eventNotice clock + rndFailureTime(), onFailure(compNo)
```

System status:

Comp1 = Up
Comp2 = Up
System = Up
Clock = 0

Initially, and empty blackboard, i.e., no events in PES

t = 8*24=192

onFailure(1)

System status:

Comp1 = Up
Comp2 = Up
System = Up
Clock = 0

Initializing, eventNotice: first failure component # 1

t = 7*24=168	t = 8*24=192
onFailure(2)	onFailure(1)

System status:

Comp1 = Up
Comp2 = Up
System = Up
Clock = 0

Initializing, eventNotice: first failure component # 2, inserted prior to failure of component # 2

t = 7*24=168

onFailure(2)

t = 8*24=192

onFailure(1)

System status:

Comp1 = Up
Comp2 = Up
System = Up
Clock = 0

Ready to go: getNxtEvent -> onFailure(2)

t = 8*24=192

onFailure(1)

System status:

Comp1 = Up
Comp2 = Up -> Down
System = Up
Clock = 168

onFailure(2); (i) change status of component # 2

t = 8*24=192

onFailure(1)

t = Clock+26 =194

onRepair(2)

System status:

Comp1 = Up
Comp2 = Down
System = Up
Clock = 168

onFailure(2); (ii) eventNotice for next repair of component # 2

t = 8*24=192

onFailure(1)

t = Clock+26 =194

onRepair(2)

System status:

Comp1 = Up
Comp2 = Down
System = Up
Clock = 168

getNxtEvent -> onFailure(1)

t = Clock+26 =194

onRepair(2)

System status:

Comp1 = Up -> Down
Comp2 = Down
System = Up -> Down
Clock = 192

onFailure(1); (i) change status of component #1

t = Clock+26 =194

onRepair(2)

t = Clock+4 =196

onRepair(1)

System status:

Comp1 =Down
Comp2 = Down
System = Down
Clock = 192

onFailure(1); (ii) eventNotice for next repair of component # 1

t = Clock+26 =194

onRepair(2)

t = Clock+4 =196

onRepair(1)

System status:

Comp1 =Down
Comp2 = Down
System = Down
Clock = 192

getNxtEvent -> onRepair(2)

t = Clock+4 =196

onRepair(1)

System status:

Comp1 =Down
Comp2 = Down -> Up
System = Down -> Up
Clock = 194

onRepair(2); (i) Change state of component #2 etc.

t = Clock+4 =196

onRepair(1)

System status:

Comp1 =Down
Comp2 = Up
System = Up
Clock = 194

... add new failure events, get next events and so on...

Simulation Engine Logic

- Start of simulation
- Initialize Ending Condition to FALSE
- Initialize system state variables
- Initialize Clock
- Schedule one or more events in the PES

Simulation Engine Logic

- Start of simulation
- Initialize Ending Condition to FALSE
- Initialize system state variables
- Initialize Clock
- Schedule one or more events in the PES
- While (Ending Condition is FALSE) then do the following:
  - Get the nextEvent from the PES
  - Set clock to nNextEvent time
  - Execute the nextEvent code and remove nextEvent from the PES
  - Update statistics

Simulation Engine Logic

- Start of simulation
- Initialize Ending Condition to FALSE
- Initialize system state variables
- Initialize Clock
- Schedule one or more events in the PES
- While (Ending Condition is FALSE) then do the following:
  - Get the nextEvent from the PES
  - Set clock to nNextEvent time
  - Execute the nextEvent code and remove nextEvent from the PES
  - Update statistics
- Generate statistical report
- End of simulation

Programming issues

- We will only consider the situation where we do the “programming” our selves
- An approach for Python is presented
- We need some standard functions like `eventNotice()` and `getNxtEvent()`
- If you write in C++, FORTRAN etc, you need to write those yourself, or find a library
- In all cases we need to write our own `onEvent()` functions

Programming issues - Callback's

- In the `eventNotice()` we need to pass a function to be called when the event is retrieved from the PES
- Such a function we would like to pass is often denoted a `Callback`-function
- In programs like C++ we may pass the (memory) `address`' of the `Callback`-function, and when the event is retrieved the function at this address is used for the execution
- In Python we use the `f = getattr(onEventLib, 'fName')`
  - where 'fName' is the name of the callback function
  - onEventLib is the nickname for a Python library imported
  - when calling the f-function we need to pass the appropriate number of parameters

`CallbackLib`-Simple

Below is an example of using the `callBack()`-function

```
Function Execute(onEvent As String)
Select Case onEvent
Case "onArrival"
  onArrival
Case "onServiceCompleted"
  onServiceCompleted
End Select
End Function
```

where `onEvent` is the name of the `callBack()`-function

`CallbackLib`-Refined

- Often we need to pass parameters to the `callBack()`-function. For example in the reliability example, we need to pass which component failed
- One way to accomplish this is to “wrap” both the name of the `callBack()`-function and parameters
- Below we assume that we only need one parameter, and the “wrapping” is typically:
  - `onEventData = Array("onFailure", CompNo)`

Library functions in: `DESlib.py`

```
InitPES()
eventID = eventNotice(time, onEventName, parameters)
onEventData = getNxtEvent()
releaseEventNotice(eventID)
execute(onEventData)
getClock()
getElapsedTime()
```

In addition you need `random.expovariate(rate)` from the `random` Python library

Example 1

- Customers are arriving to our service according to a Poisson process with parameter $\lambda = 3$
- The times between arrivals are then $\sim Exp(\lambda)$
- We have one server that can treat the arriving customers
- The service rate is $\mu = 4$, we assume exponentially distributed service times
- The maximum number of customers in the system is $N = 3$
- Objective: Find the mean number of customers in the system

Solution in Python

Task to do in the class

- Consider a workshop with one critical machine
- The machine has a constant failure rate equal to $\lambda = 0.001$ and there is one repair man that can repair a failed machine
- The rate of repair is $\mu = 0.1$ meaning that the mean repair time is $1/\mu = 10$
- Find the average portion of time the machine is not functioning
- DES lib for Python found here:
  - jvatn.folk.ntnu.no/eLearning/TPK4191/examples/Python/DESlib.py

Example 2

- Consider a workshop with three critical machines
- Each machine has a constant failure rate equal to $\lambda = 0.001$ and there is one repair man that can repair failed machines
- The rate of repair is $\mu = 0.1$ meaning that the mean repair time is $1/\mu = 10$
- The system state variable represents the number of functioning machines

▸ Solution in Python

Example 3

In this example we consider a system with one active and one passive (stand-by) pump

- The active pump has failure rate λ_{A}, where a failure will immediately be detected
- Mean time to repair for the active pump is MTTR
- The passive pump may fail in stand-by mode with a failure rate λ_{P}. Proof tests are carried out every τ time unit. If the stand-by pump is successfully started upon a demand, we assume that it will not fail while the active pump is being repaired
- When the active pump is repaired, the stand-by pump goes back to a stand-by mode
- Repair time of stand-by pump = 0, but we cannot start the system before the active pump is repaired

Solution in Excel VBA

Example 4

In this example we consider on offshore wind turbine. If the turbine fails, we have to wait for the weather window to open

- The wind turbine has failure rate $\lambda = 1/\mathrm{MTTF}$, where a failure will immediately be detected
- Mean repair time is MTTR
- Mean time for the weather window to open is gamma distributed, where the parameters depends on the season
- Winter season = 8760/2, and summer season = 8760/2, but we may change these...

▸ Solution in Excel VBA

Example 5

In this example we consider a component with a *hidden function*

- The failure rate is λ
- A proof test is carried out every τ time unit to reveal hidden failures
- If the component is in a fault state at the test, a repair action is initiated
- Mean time to repair is $1/\mu$

The objective is to find the probability that the system is in a fault state at a random point of time, i.e., the PFD = Probability of Failure on Demand

▸ Solution in Excel VBA

Thank you for your attention

NTNU | Norwegian University of Science and Technology