Advanced features for Multi-formalism modelling with Java Modelling Tools

Giuseppe Serazzi
Politecnico di Milano, Italy
giuseppe.serazzi@polimi.it

Joint work with:
Giuliano Casale, Lulai Zhu
Imperial College London, UK

28/11/19
Outline

- the JMT suite of tools
- Fork/Join (Queueing Networks, QN)
  - timeout of computing time
  - approximate computing
- Place/Transition (Petri Nets, PN)
- Multi-formalism models (QN + PN)
  - core allocation in HPC system
  - Adaptive Car Navigation Service
- What’s Next
Java Modelling Tools (JMT)

The project
- co-developed by Politecnico di Milano and Imperial College London

Supported models
- Queueing Systems
- Queueing Networks (QN)
  - Product-form
  - Extended (fork/join, blocking, priorities, ...)
- Petri Nets (PN)
  - Stochastic Petri Nets (SPN)
  - Generalized Stochastic Petri Nets (GSPN)
  - Coloured Petri Nets (CPN)
- Multiformalism models (QN + PN)
Java Modelling Tools (JMT) is a suite of applications developed by Politecnico di Milano and Imperial College London and released under GPL license.

The project aims at offering a comprehensive framework for performance evaluation, system modeling with analytical and simulation techniques, capacity planning and workload characterization studies.

The current stable version of the suite includes six Java applications:

1. JSImgraph - Queueing network and Petri net simulator with graphical user interface
2. JSImwiz - Queueing network and Petri net simulator with wizard-based user interface
3. JMWVA - Mean Value Analysis and Approximate solution algorithms for queueing network models
4. JWABA - Asymptotic Analysis and bottlenecks identification of queueing network models
5. JWAT - Workload characterization from log data
6. JMCH - Markov chain simulator

Manual, papers, video tutorials

Versions

Download: JMT 1.0.4 (Released 2019-Sep-04).
Release announcements: please subscribe to the jmt-announce list.
New users: new to JMT? Check out the manual and our video tutorials.

Reference

If you are using JMT for scientific papers, academic lectures, project reports, or technical documents, please cite:

M. Bertoli, G. Casale, G. Serazzi.
JMT start screen  http://jmt.sourceforge.net
Fork/Join (QN)
**Fork**
- jobs split into tasks (with same id of the parent job) executed in parallel
- **advanced fork policies**:
  - Branch prob.: randomize output links and no. of tasks per link
  - Random subset: choose n-out-of-k output links
  - Class Switch: assign new class to forked tasks

**Join**
- fires a job when all tasks forked from the same parent job are executed
- **advanced join policies**:
  - Quorum: wait a subset of tasks of the same parent job
  - Guard: like quorum but requires a given class mix (multiclass workload)

**F/J region**
- finite capacity on no. of jobs, multiple join
- Semaphore: block the first N tasks of the same job, when N is reached unblock all
timeout on computation time – the problem

- the service demands of the algorithms executed by a genetic program are highly variable
- a limit (timeout) to the computation time of each algorithm must be set in order to complete the program execution in a given amount of time
- the timeout can be described by a constant value or other statistical metrics: mean, std.dev, distribution (independence from distributions)
- investigate the behavior of the response time of each algorithm as a function of the timeout

- the model
  - F/J region with capacity limited to one request
  - two Delay stations: one for service demands, one for timeout
  - the Fork has two output links, one for each Delay
  - job split into two tasks, one for each output link
  - Join waits only the first task executed (Quorum=1)
timeout – the JSIM model

\[ \lambda = 0.5 \text{ req/s} \]

\[ S = 10 \text{ s constant} \]

\[ \text{max capacity} = 1 \text{ req} \]
timeout – Fork/Join parameters

- a req generate one task per link
- the max number of req in the F/J section is 1

Join Quorum=1: the first task that complete the execution exit the Join
timeout – service times of the two Delays

Service times: hyper-exp distr. $S=1\,s$  coeff.var.=$5$

timeout value
constant = 10 sec
timeout – service times of requests

service times without control

service times with timeout
timeout – response times Fork/Join vs timeout

Fork Join Response Time
Average response time for each selected class in each selected Fork/Join section.

<table>
<thead>
<tr>
<th>Description</th>
<th>Plot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Station Name:</td>
<td>Fork 1 (Fork Join)</td>
</tr>
<tr>
<td>Class Name:</td>
<td>All classes</td>
</tr>
<tr>
<td>Conf.Int. / Max Rel.Err. (0-1):</td>
<td>0.99 / 0.03</td>
</tr>
<tr>
<td>Models:</td>
<td>30</td>
</tr>
<tr>
<td>X min:</td>
<td>1</td>
</tr>
<tr>
<td>X max:</td>
<td>30</td>
</tr>
<tr>
<td>Y min:</td>
<td>0.539</td>
</tr>
<tr>
<td>Y max:</td>
<td>4.826</td>
</tr>
</tbody>
</table>

n.o of JSIM invocations

Simulation Results
<table>
<thead>
<tr>
<th>*</th>
<th>21.000 s</th>
<th>22.000 s</th>
<th>23.000 s</th>
<th>24.000 s</th>
<th>25.000 s</th>
<th>26.000 s</th>
<th>27.000 s</th>
<th>28.000 s</th>
<th>29.000 s</th>
<th>30.000 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Value (s)</td>
<td>3.0853</td>
<td>3.2729</td>
<td>3.3960</td>
<td>3.6276</td>
<td>3.7516</td>
<td>3.9673</td>
<td>4.1267</td>
<td>4.3109</td>
<td>4.4779</td>
<td>4.7207</td>
</tr>
</tbody>
</table>

timeout 1÷30 sec
approximate computing: Alternative Route Planning

• Alternative Route Planning: several algorithms are executed in parallel to compute N=6 alternative routes from source to destination in a Car Navigation System (highly variable computation times: travelling time, municipality policies: no city center, limited traffic areas, street closed, ...)

• to limit the computation time of the best route to $R \leq 3$ sec, sub-optimal solutions (approximate) are computed using a subset $k$ of the N routes

• impact on the best route computation time wrt variability of the computation times and the subset size $k$

• the model
  • F/J region with capacity limited to one request
  • 6 delay stations, one for each algorithm
  • a request is split into N=6 tasks in output from Fork (1 task per link)
  • Join waits the complete execution of only the first $k$ tasks
approx. comp.– the model

1 request \(\rightarrow\) 6 tasks

\(\lambda = 0.2 \text{ r/s exp}\)

Join Quorum=\(k\) tasks

max capacity = 1 req

\(S=1\ c=5\)
\(S=3\ c=3\)
\(S=1\ c=1\)
\(S=2\ c=1\)
\(S=4\ c=0.7\)
\(S=5\ c=0.5\)

Java Modelling Tools
approx. comp. - Fork/Join parameters

- each req generate one task per output link
- the max number of req in the F/J section is 1

Join Quorum=4: wait the computation of the first 4 routes then fire
approx. comp. – computation time of k routes

The graph illustrates the relationship between the number of alternative routes computed (x-axis) and the time to compute k routes (y-axis). A threshold is marked, which indicates the point at which the computation time for all 6 algorithms significantly increases. This threshold appears to be around 6 alternative routes computed.
Place – Transition (PN)
PN elements supported

- Places and Transitions (PN)

- Multi-formalism models (QN + PN)
PN sections & modes

- JMT design paradigm extends to PN elements
  - Enabling, Inhibiting, Timing, Firing sections
- Mode: a rule to activate and fire a transition
PN - transition parameters

Enable firing when 4 class1 requests are arrived (4 tokens of colour class1)

Inhibit the fire when 6 or more requests are arrived

When enabled, send 5 class1 requests to Sink1
Multi-formalism models (QN+PN)
a cloud provider offers a HPC service through a cluster infrastructure with two powerful HPC multicore systems (HPC Server1, HPC Server2) sharing a high performance storage and networking components

the HPC app considered (find nearest available parking space) is compute intensive and the number of cores it uses depends on the number of requests in execution (one request per core assuming a single-thread core)

there is a high variability of arrival traffic of requests and compute demands

arriving requests to the cluster are initially routed to HPC Server1

to avoid decreasing the performance of other apps running on HPC Server1, the number of cores that can be allocated to this app is limited to N

when this threshold is reached, new incoming requests are dynamically routed to HPC Server2 (where an instance of the same app is running)
HPC cluster on cloud: the infrastructure

• GOAL: evaluate response times and throughput of the requests executed on the HPC cluster \textit{wrt} the constraint on the maximum number of cores it can use in HPC Server1 (in the range 1-190)
HPC cluster: the model

- 2 classes of customers: requests (open) and cores (closed)

\[ \lambda = 5 \text{ r/s} \]

\[ c = 4 \]

\[ S = 0.18 \text{ s} \exp \]

\[ \text{standard route} \]

\[ \text{alternative route} \]

\[ \text{inhibiting arc} \]

\[ \text{throughput of Server 1} \]

\[ \text{throughput of Server 2} \]
HPC cluster: transitions enabling/inhibiting conditions

Join_Server 1

- Enabling Condition for Mode 1:
  - Arriving: 1, maxReq_Link1: 0
  - MaxServer1: 0, 1

- Inhibiting Condition for Mode 1:
  - Arriving: ∞
  - MaxServer1: ∞

Rel

- Enabling Condition for Mode 1:
  - Arriving: 1, maxReq_Link1: 0
  - Release: 1

- Inhibiting Condition for Mode 1:
  - Arriving: ∞
  - Release: ∞

Join_Server 2

- Enabling Condition for Mode 1:
  - Arriving: 1, maxReq_Link1: 0
  - MaxServer1: 0, 0

- Inhibiting Condition for Mode 1:
  - Arriving: ∞
  - MaxServer1: ∞

- Execution completed, exit the model

- Inhibit when ≠ 0 (cores are available)

never blocked
at least 1 core must be available
execution completed, exit the model
inhibit when ≠ 0 (cores are available)
HPC cluster: transitions firing outcome

1 request sent to Server 1 (when enabled)

1 request sent to Server 2 (when enabled)

1 request completed and exit + 1 core becomes available (when enabled)
HPC cluster: behavior of core allocation ($N_{1\text{max}}=20$)

- **Arriving requests**
- **Cores allocated of Server 1**
- **Cores allocated of Server 2**
HPC cluster: behavior of response times ($N_{1max}=20$)

Server 1

mean $R_{Server1}=2.20$ s

Server 2

mean $R_{Server2}=0.78$ s
HPC cluster: mean no. of cores allocated vs $N_{1\text{max}}$

**Server 1**

Plot

Show confidence interval range

- $N_{1\text{max}}$ vs max number of cores available on Server 1

**Server 2**

Plot

Show confidence interval range

- $N_{1\text{max}}$ vs max number of cores available on Server 1

$N_{1\text{max}}$: max number of cores available on Server 1
HPC cluster: response times of Server1 – Server2 vs N1_{\text{max}}

**Server 1**

\[ \text{Sink}_1 \text{-Arrv\_Req\_Response Time per Sink (s)} \]

\[ N_{1\text{max}} = 190 \times 10^2 \]

\[ \text{max number of cores available on Server 1} \]

**Server 2**

\[ \text{Sink}_2 \text{-Arrv\_Req\_Response Time per Sink (s)} \]

\[ N_{1\text{max}} = 190 \times 10^2 \]

\[ \text{max number of cores available on Server 1} \]
HPC cluster: throughput of Server1 vs Server2 vs $N_{1\text{max}}$

**Server 1**

Plot

- **Show confidence interval range**

$N_{1\text{max}}$

max number of cores available on Server 1

**Server 2**

Plot

- **Show confidence interval range**

$N_{1\text{max}}$

max number of cores available on Server 1

$N_{1\text{max}}$ max number of cores available on Server 1
Adaptive Car Navigation Service for Smart Cities

- provide optimal routes to hundred thousands of drivers operating in the city area

Requirements:
- intelligent routing based on calculation of traffic view state
- traffic global view calculation & optimization based on data fusion from several sources
- different layers of adaptivity

GOAL: evaluate the optimal number of cores of the HPC infrastructure to achieve the target performance of \( R \leq 1 \text{ sec} \).
Car Navig. Serv.: three stages of the application

**INPUT**
- **Input data**
  - GPS Coord.
  - Origin, Dest.
  - Dep. Time

**OUTPUT**
- **Route**
  - Waypoint 1
  - Waypoint 2
  - ...

**Global view elements**
1. Road Network Graph
2. Traffic Data
3. Municipal Policies

**QN + PN**
- ARP
- PTDR
- Reordering

**Best Route**
Car Navig. Serv.: Adaptivity controls

- the application may apply sub-optimal solutions when the system is overloaded, or when the computation time of a request is too long.
Car Navig. Serv.: ARP stage – Regular service

req. in queue < n

Resources

ing

Release

available
Car Navig. Serv.: ARP stage – requests with timeout

req. in queue < n

Resources

available

ing

Release

available

requests with timeout
Car Navig. Serv.: Fast algorithm when overloaded

- **Fast lane**: system privileges speed over quality, giving a better performance at the expense of the effectiveness of the suggested routes.
Car Navig. Serv.: transitions enab/inhib conditions

Serving

- **Enabling Condition for Mode1**
  - Incoming: 1
  - Cores: 0

- **Inhibiting Condition for Mode1**
  - Incoming: $n$
  - Cores: $\infty$

Overload

- **Enabling Condition for Mode1**
  - Incoming: $n$
  - Cores: 0

- **Inhibiting Condition for Mode1**
  - Incoming: $\infty$

- **Enabling when requests in queue are $\geq n$**
- **Block when requests in queue are $\geq n$**
- **At least 1 core available**
- **At least n core available**
Car Navig. Serv.: workload

- routing requests traffic in Milano Urban Area (1 day)
  2.5 Million trips per day
Car Navig. Serv.: Regular and Fast lane response time
## Car Navig. Serv.: some Results

<table>
<thead>
<tr>
<th>Time Slot</th>
<th>ARP</th>
<th>PTDR</th>
<th>Reordering</th>
<th>Total cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 - 06</td>
<td>51</td>
<td>10</td>
<td>1</td>
<td>62</td>
</tr>
<tr>
<td>07 - 10</td>
<td>584</td>
<td>111</td>
<td>2</td>
<td>697</td>
</tr>
<tr>
<td>11 - 16</td>
<td>373</td>
<td>71</td>
<td>1</td>
<td>445</td>
</tr>
<tr>
<td>17 - 20</td>
<td>393</td>
<td>75</td>
<td>1</td>
<td>469</td>
</tr>
<tr>
<td>21 - 24</td>
<td>200</td>
<td>38</td>
<td>1</td>
<td>239</td>
</tr>
</tbody>
</table>
What’s Next
What’s next

- Impatient customers (debugged)
- Parallel What-if (debugged)
- What-if – new control parameters
- New scheduling algorithms
- Blackbox modeling with predefined targets
  
  - With multiclass workload find the optimal load that minimize Response time or maximize throughput, Energy saving in a cloud, adaptive allocation of Virtual Machines, ...

- ...
  ...
  ...