DiffLQN — User Manual
sysma.imtlucca.it/difflqn

Tabea Waizmann and Mirco Tribastone
IMT — Institute for Advanced Studies Lucca

October 23, 2016

1 Introduction

Layered queuing networks (LQNs) are a popular technique in software performance engineering. DiffLQN is a tool for the analysis of LQNs using ordinary differential equations (ODEs). It estimates average performance indices such as throughputs, utilizations, and response times of software and hardware devices. The complexity of computing the solution is independent of the concurrency levels in the model (i.e., thread multiplicities and processing units) and the estimates are theoretically guaranteed to be asymptotically correct for large enough concurrency levels [1]. DiffLQN is designed having in mind compatibility with LQNS [2], which supports state-of-the-art analysis techniques based on mean value analysis.

1.1 Overview of LQNs

We start with a brief overview of LQNs using an example that exhibits all the features supported by DiffLQN. We refer to [3] for the details on LQNs. Our sample LQN is graphically depicted in Fig. 1 (although we remark that in DiffLQN this would be specified concretely in a text format). The basic computational resource is a *processor* (ovals) on which *tasks* (large parallelograms), for instance software services, are deployed. A task consists of different *entries* (smaller parallelograms) that represent distinct kinds of services. An entry can be a basic *activity*, the atomic unit of operation in LQN if it is not further specified. Otherwise, it points to a diagram of basic activities (rectangles) which are performed in sequence (linked by an arrow), through probabilistic choice via *decision/merge* nodes ('+' operator), or by means of fork/join synchronization ('&' operator).

Activities can call entries synchronously (closed arrowhead) or asynchronously (open arrowhead). Asynchronous requests just start the requested entry while continuing the current activity, without waiting for a reply. In synchronous requests, the activity is stopped until a reply arrives. An activity can send any number of requests to the same entry; the number of requests is written next to the call arrow in brackets. When called, an activity consumes time on the processor where the task is deployed. Time demands are shown below the activity name within square brackets. When two time demands are listed, as in write, second-phases are modeled. The second time demand happens after the activity has returned control to the caller, effectively executing asynchronously.

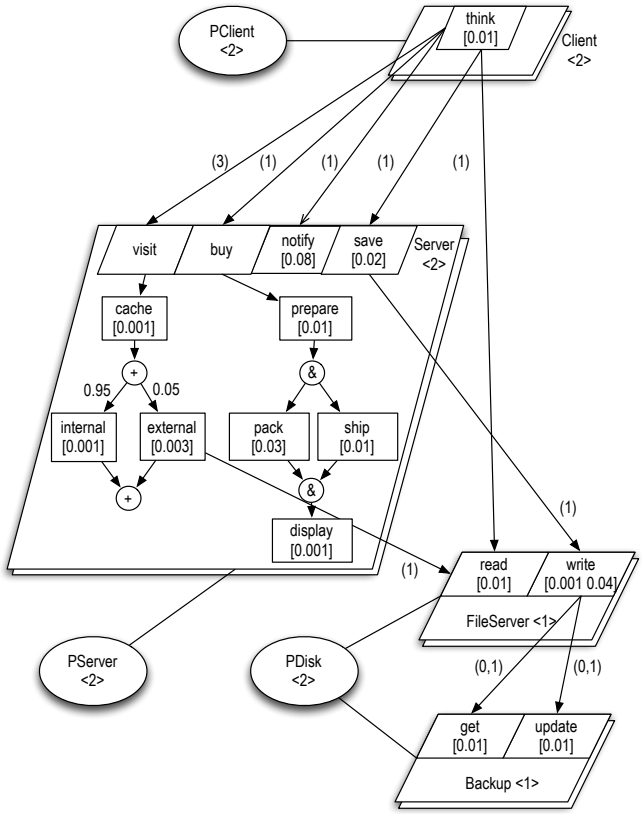

Figure 1: Graphical representation of an LQN, taken from [1].

2 Installation

DiffLQN is available as a single-jar Java executable compiled against Java 1.6. It can be downloaded as a zip archive from `http://sysma.imtlucca.it/tools/difflqn/`. All examples presented in this document can be found in the distribution.

DiffLQN executes fro the command line by running

```
java -jar DiffLQN.jar <filename>
```

where `filename` is the input LQN model, with `lqn` extension.

The additional argument `-s <solution>` can be used to get results from an externally computed steady state. The `<solution>` file has to simply contain a list of numbers in the right length.

3 Syntax

Model specification. To favour compatibility, DiffLQN accepts a slight variant of the text-based input format for LQNS. We refer to [4] for a complete documentation on the grammar. Some LQN features, e.g. loops and tasks with infinite multiplicity, are not currently supported. DiffLQN accepts LQN models containing unsupported features as valid syntax, but the solver either emits a warning, or explains the problem in an error message.

```
1  # This block is supported for backward compatibility
2  # but it is not used by DiffLQN
3  G
4  "simple-qn.lqn"
5  0.0001
6  500
7  1
8  0.5
9  -1
10
11 # Processors declaration, with multiplicity
12 P 0
13 p ClientP f m 10
14 p ServerP f m 2
15 -1
16
17 # Tasks declaration
18 T 0
19 t ClientT r client -1 ClientP m 10 # 10 client tasks running on client processors
20 t ServerT n serve -1 ServerP m 2   #  2 server tasks running on server processors
21 -1
22
23 # Entries declaration
24 E 0
25 s client 0.01 -1         # entry think has time demand 0.01 time units
26 y client serve 1.0 -1    # entry think makes a synchronous call to serve
27 s serve 0.02 -1          # entry serve has time demand 0.02 time units
28 -1
29
30 # DiffLQN settings, starting with #!
31 # These will be ignored by LQNS
32
33 # 1. Solver settings
34 #! v 1.0e5            # fast rate to approximate instantaneous events
35 #! solver ode         # ODE analysis - solver sim will run simulation
36 #! stoptime 50.0      # integration time horizon
37
38 # 2. Output settings
39 #! throughput: client serve
40 #! utilization: ServerP
41 #! responsetime: serve -1
42
43 # 3. Export settings
44 #! export csv
```

Figure 2: A minimal LQN model with the concrete syntax supported by DiffLQN.

Fig. 2 shows the concrete DiffLQN syntax for a minimal client/server example. (The file is available in the tool distribution.) The intent is to model a system with 10 clients which interpose an exponentially distributed local execution time demand with mean duration 0.01 time units before accessing a resource. This is modeled as a software server (an LQN task) with 2 threads running on a processor with 2 cores. After the local activity, the client synchronously calls the server once; the server takes 0.02 time units on average to process a request.

The `G`-block in lines 3–7 is ignored by DiffLQN because it provides parameter settings specific to LQNS. We show it to highlight that the same file can be run through LQNS. Processors are defined in lines 12–15 within the block delimited between `P 0` and `-1`. DiffLQN accepts only processors with FIFO discipline (keyword `f`); `m` declares their multiplicity. Tasks are defined in a similar fashion (lines 18–21). Each line declares the task name, its kind

(keywords `r`/`n`), followed by the list of entries running on the task, the processor on which the task is deployed, and the multiplicity. Keyword `r` declares *reference tasks*, i.e., tasks that do not accept requests. Entries are specified in lines 24–28. Service demands are given in lines starting with `s`, while synchronous calls are specified by the keyword `y`.

Directives specific to DiffLQN are provided at the end of the input file, in lines 34–34. These are backward compatible with LQNS since every line must start with '`#!`', which is treated by LQNS as a comment. DiffLQN directives consist of (in order):

1. Solver settings (required)
2. Output settings (optional)
3. Exporting settings (optional)

These are discussed in detail below.

Solver settings. ODE analysis is performed by solving an initial value problem numerically until convergence to steady state is detected (or if a threshold time horizon is reached, in which case a warning is issued if convergence has not been reached). The convergence criteria are based on *absolute* and *relative tolerances*. The former considers the Euclidean norm of the derivatives of the solution at the current time point, and absolute convergence is reached when this value is below a given threshold (formally, the norm must be equal to zero in the steady state); the latter compares the norm of the difference between the solutions at successive time points. By default, the analysis terminates successfully when both the absolute and the relative convergence criteria are met.

Stochastic simulation is performed using the method of batch means [5]: roughly speaking, a single simulation run is performed and statistics are collected across different non-overlapping parts of the run (the batches) which are assumed to be long enough that the system has reached steady state.

Below we list the solver settings that are currently supported.

- `v` specifies the value for a fast rate `v` that approximates the behavior of certain operations, such as forks and joins, that are assumed to be instantaneous in LQNs. This is the only mandatory setting.
- `solver [ode | sim]` specifies whether to use ODE analysis or stochastic simulation.
- `stoptime` specifies the maximum time horizon for the numerical ODE integration or the length of an initial transient simulation run that is removed before batch statistics are collected.
- `solver_abs_tol` and `solver_rel_tol` are the typical absolute and relative tolerances for the ODE numerical integration [6].
- `steady_abs_tol` and `steady_rel_tol` specify the tolerances for ODE steady-state detection, as discussed above.
- `[ absolute | relative ] steady state` is a flag for using only one of the two criteria of steady-state convergence.

Setting	*Default value*
`solver`	ode
`solver_abs_tol`	1E-10
`solver_rel_tol`	1E-11
`steady_abs_tol`	1E-6
`steady_rel_tol`	1E-6
`batch_length_factor`	2.0
`confidence_level`	0.95
`confidence_percent_error`	5.0

Table 1: Default values for the DiffLQN solver options.

- `batch_length_factor` specifies the length of a batch, relative to the initial transient defined with `stoptime`.
- `confidence_level` together with `confidence_percent_error` specify the usual termination criteria for stochastic simulation.

Solver settings `v` and `stoptime` must be provided in the input file. If not specified, DiffLQN runs with default values for all the other settings. These are shown in Table 1.

Output settings. DiffLQN provides the following LQN performance indices:

- *Throughput*, at different levels of granularity: it provides the average number of activities, entries, or tasks completed per unit time at steady state.
- *Utilization* for processors and tasks, giving the average number of busy entities at steady state. Utilization estimates are also provided per single entry and activity, giving their contribution to the utilization of the processor on which they are deployed.
- *Response time*, the average response time at steady state for the execution of entries or tasks.

By default, DiffLQN computes all possible performance measures. Optionally the user can explicitly choose which measures to track. This can be speed up the computation, especially for large networks analyzed using simulation [7]. This is done in a block with lines starting (in order) with keywords `throughput`, `utilisation`, and `responsetime`, followed by a list of desired elements for the respective performance index (closed by -1).

Export settings. By default, analysis results are outputted to the screen in a human-readable format. After solver-specific advisory, the output consists of line containing the following information.

- Type of performance measure;
- Type of element;
- Name of element;

- Result value;
- Confidence interval, in case of simulation.

However, the LQN model as well as the results can be exported in different formats. Each export command is specified in a new line with the `export` keyword, followed by the type of export requested (and an optional file path). Available export types are:

- `pepa`: Export of the PEPA encoding of the input LQN, in a format that is compatible with the PEPA Eclipse plug-in.
- `matlab`: A function file in Matlab-compatible form which can be used in conjunction with Matlab's ODE solvers (e.g., the stiff solver `ode15s`)
- `csv`: Results are saved to a comma-separated values file.

Figure 3 shows the input DiffLQN file corresponding to the model in Fig. 1 showing all the elements of the grammar as well as a selection of the optional directives.

References

[1] Mirco Tribastone. A fluid model for layered queueing networks. *IEEE Transactions on Software Engineering*, 39(6):744–756, 2013.

[2] Real-Time and Distributed Systems group, Department of Systems and Computer Engineering, University of Carleton. LQNS software package. `http://www.sce.carleton.ca/rads/lqns`.

[3] Greg Franks, Tariq Al-Omari, Murray Woodside, Olivia Das, and Salem Derisavi. Enhanced modeling and solution of layered queueing networks. *IEEE Transactions on Software Engineering*, 35(2):148–161, 2009.

[4] Greg Franks, Peter Maly, Murray Woodside, Dorina Petriu, Alex Hubbard, and Martin Mroz. Layered Queueing Network Solver and Simulator User Manual. `http://www.sce.carleton.ca/rads/lqns/LQNSUserMan-jan13.pdf`, 2013.

[5] William J. Stewart. *Probability, Markov Chains, Queues, and Simulation.* Princeton University Press, 2009.

[6] Uri M. Ascher and Linda R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations.* SIAM, 1988.

[7] Mirco Tribastone, Jie Ding, Stephen Gilmore, and Jane Hillston. Fluid rewards for a stochastic process algebra. *IEEE Transactions on Software Engineering*, 38:861–874, 2012.

```
G
"Example-LQN" 0.0001 500 1 0.5
-1

P 0
p PClient f m 2
p PServer f m 2
p PDisk f m 2
-1

T 0
t Client r think -1 PClient m 2
t Server n visit buy notify save -1 PServer m 2
t FileServer n read write -1 PDisk
t Backup n get update -1 PDisk
-1

E 0
s think 0.1 -1
y think visit 3.0 -1
y think save 1.0 -1
y think notify 1.0 -1
y think read 1.0 -1
y think buy 1.0 -1
A visit cache
A buy prepare
s save 0.02 -1
y save write 1.0 -1
s notify 0.08 -1
s read 0.01 -1
s write 0.001 0.04 -1
y write get 0.0 1.0 -1
y write update 0.0 1.0 -1
s get 0.01 -1
s update 0.01 -1
-1

# Activity definition block for task Server
A Server
 s prepare 0.01
 s pack 0.03
 s ship 0.01
 s display 0.001
 s cache 0.001
 s internal 0.001
 s external 0.003
 y external read 1.0
:
 prepare -> pack & ship;
 pack & ship -> display;
 cache -> (0.95)internal + (0.05)external;
 internal[visit];
 external[visit];
 display[buy]
-1

#! v 1.0e5
#! solver sim
#! confidence_level 0.98
#! confidence_percent_error 2.0
#! stoptime 1000.0
#! export csv
```

Figure 3: Input DiffLQN file corresponding to the model in Fig. 1.