Stabilization of Branching Queueing Networks

Tomáš Brázdil\textsuperscript{1}  Stefan Kiefer\textsuperscript{2}

\textsuperscript{1}Masaryk University, Brno, Czech Republic
\textsuperscript{2}University of Oxford, UK

STACS, Paris
02 March 2012
Queueing networks are simple and general.

Attractive for modeling parallelism:

- **Hardware**, especially large-scale multi-core systems:
  - Full-system performance simulators do not scale.
  - Queueing theory gives a more abstract analysis.

- **Software**, especially message passing:
  - asynchronous programs on multi-core computers
  - distributed programs on a network
Example: Router

Continuous-Time Markov Chain – Infinite-State!

Tomáš Brázdil, Stefan Kiefer
Stabilization of Branching Queueing Networks
Example: Router

Textual representation of the same thing:

\[ 0 \overset{1}{\rightarrow} A \]
\[ A \overset{0.25}{\rightarrow} S_1 \]
\[ A \overset{0.25}{\rightarrow} S_2 \]
\[ A \overset{0.5}{\rightarrow} D \]
\[ A \overset{0.25}{\rightarrow} S_1 \]
\[ D \overset{0.7}{\rightarrow} M \]
\[ D \overset{0.3}{\rightarrow} C \]
\[ S_1 \overset{1}{\rightarrow} D \]
\[ S_2 \overset{1}{\rightarrow} D \]
\[ C \overset{1}{\rightarrow} M \]
\[ M \overset{1}{\rightarrow} \varepsilon \]
Example: Router

Textual representation of the same thing:

\[
\begin{align*}
0 & \xrightarrow{1} A \\
A & \xrightarrow{0.25} S_1 \\
A & \xrightarrow{0.25} S_2 \\
A & \xrightarrow{0.5} D \\
D & \xrightarrow{0.7} M \\
D & \xrightarrow{0.3} C \\
S_1 & \xrightarrow{1} D \\
S_2 & \xrightarrow{1} D \\
C & \xrightarrow{1} M \\
M & \xrightarrow{1} \varepsilon
\end{align*}
\]

Also given: arrival rate $\alpha_A$
queue rates $\mu_A, \mu_{S_1}, \ldots$ of busy queues
Continuous-Time Markov Chain – Infinite-State!
Such networks are called **Jackson networks** (classical model).

Nice properties:

- **Stability** is easy to determine.
- **Product form solutions in “steady state”:**

  \[
  \Pr(S_1 = 3 \text{ and } D = 2) = \Pr(S_1 = 3) \cdot \Pr(D = 2)
  \]

But shortcomings in modeling:

- just one new task per one old task
- no (dynamic) control of the network
New model: Branching Queueing Networks (BQNs)

\[
\begin{align*}
0 & \xrightarrow{1} A \\
A & \xleftarrow{4/5} \varepsilon \\
B & \xleftarrow{1/6} A, B \\
A & \xrightarrow{1/5} B, B \\
B & \xleftarrow{5/6} \varepsilon
\end{align*}
\]

Rates as before: \(\alpha_A, \mu_A, \mu_B > 0\)

Similar models:
- a bit like pushdown systems, but parallel
- a bit like stochastic Petri nets, but of a special form
New model: Branching Queueing Networks (BQNs)

\[
0 \xrightarrow{1} A \\
A \xrightarrow{4/5} \varepsilon \\
B \xrightarrow{1/5} B, B \\
B \xrightarrow{1/6} A, B \\
B \xrightarrow{5/6} \varepsilon
\]

Rates as before: \( \alpha_A, \mu_A, \mu_B > 0 \)

Topic of this talk: Is a given BQN stable?

Stabilization of Branching Queueing Networks
New model: **Branching Queueing Networks (BQNs)**

\[ 0 \xrightarrow{1} A \]
\[ A \xleftarrow{4/5} \varepsilon \]
\[ A \xrightarrow{1/5} B, B \]
\[ B \xleftarrow{1/6} A, B \]
\[ B \xrightarrow{5/6} \varepsilon \]

Rates as before: \( \alpha_A, \mu_A, \mu_B > 0 \)
New model: Branching Queueing Networks (BQNs)

\[
\begin{align*}
0 & \xrightarrow{1} A \\
A & \xrightarrow{4/5} \varepsilon \\
B & \xrightarrow{1/5} B, B \\
A & \xrightarrow{1/6} B, B \\
B & \xrightarrow{5/6} \varepsilon \\
B & \xrightarrow{5/6} \varepsilon
\end{align*}
\]

Rates as before: \( \alpha_A, \mu_A, \mu_B > 0 \)

Topic of this talk: Is a given BQN stable??

\text{stable} \overset{\text{def}}{=} \text{expected return time to “completely empty” is finite}
Consider the BQN

\[
0 \xleftarrow{1} A \xrightarrow{1} B, C \quad B \xrightarrow{1} \varepsilon \quad C \xrightarrow{1} \varepsilon
\]

No product form: Take \( \alpha_A = 1, \mu_A = \mu_B = \mu_C = 3 \).
Then in steady-state:

\[
\Pr(C \geq 1) = \frac{1}{3} \\
\Pr(C \geq 1 \mid B \geq 1) \geq \frac{3}{7}
\]
Consider the BQN

\[
0 \xleftarrow{1} A \xrightarrow{1} B, C \xleftarrow{1} \varepsilon \xrightarrow{1} C
\]

No product form: Take \( \alpha_A = 1, \mu_A = \mu_B = \mu_C = 3 \).
Then in steady-state:

\[
\Pr(C \geq 1) = \frac{1}{3} \\
\Pr(C \geq 1 \mid B \geq 1) \geq \frac{3}{7}
\]

Analysis of BQNs harder than for Jackson networks
(Pre-)Suppose a form of balance:

\[
\text{# tasks processed at } X \text{ per sec} = \text{# tasks arriving at } X \text{ per sec}
\]

Here:

\[
\lambda_A = 0.2 + \lambda_B \cdot (1/6)
\]

\[
\lambda_B = \lambda_A \cdot (1/5) \cdot 2 + \lambda_B \cdot (1/6)
\]

We call the solution \( \lambda \) the “throughput”.

Note: the speeds of the queues \( \mu_A, \mu_B \) do not occur.

Here:

\[
\lambda_A = 0.22
\]

\[
\lambda_B = 0.10
\]
Proposition

Given a BQN.

- Let $\lambda$ be the **throughput**, i.e., $\lambda$ solves the balance equations $\lambda = \alpha + M\lambda$.
- Suppose $\lambda < \mu$ (in all components).
  Then the BQN is stable.

- Conversely, if the BQN is stable, then there is $\lambda$ with $\lambda = \alpha + M\lambda < \mu$. 
A Stability Result

Proposition

Given a BQN.

- Let $\lambda$ be the throughput, i.e., $\lambda$ solves the balance equations $\lambda = \alpha + M\lambda$.
  
  Suppose $\lambda < \mu$ (in all components).
  
  Then the BQN is stable.

- Conversely, if the BQN is stable, then there is $\lambda$ with $\lambda = \alpha + M\lambda < \mu$.

Theorem

Stability of a BQN can be decided in polynomial time.

Proposition

If a BQN is stable, it is “very much so”: in steady state there is an exponential moment of the total queue size, i.e., there is $\delta > 0$ such that $\sum_{x \in \mathbb{N}^n} \exp(\delta \|x\|) \Pr(x)$ exists.
The stability result requires a delicate proof (no product form).

Consider “drift” for all points:
The stability result requires a delicate proof (no product form).

Consider “drift” for all points:

For \((0, 0)\): \([0] = \begin{pmatrix} \alpha A \\ 0 \end{pmatrix}\)
Proof of the Stability Result

The stability result requires a delicate proof (no product form).

Consider “drift” for all points:

For \((0, 0)\): \([0] = \begin{pmatrix} \alpha A \\ 0 \end{pmatrix}\)

For \((+, 0)\): \([0A] = [0] + [A]\) with \([A] = \mu_A \cdot \begin{pmatrix} -1 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix}\)

\([0A] = \mu_A \cdot \begin{pmatrix} -1 & 2 \\ 0 & 0 \end{pmatrix}\)
Proof of the Stability Result

The stability result requires a delicate proof (no product form).

\[
\begin{array}{c|cccc}
0 & A & B, B & A, B & \varepsilon \\
0 & 1 & 1/5 & 1/6 & 4/5 & 5/6 \\
A & B & A & \varepsilon & \varepsilon \\
\end{array}
\]

Consider “drift” for all points:

For \((0, 0)\): \([0] = \begin{pmatrix} \alpha A \\ 0 \end{pmatrix}\)

For \((+, 0)\): \([0A] = [0] + [A]\) with

\([A] = \mu_A \cdot \begin{pmatrix} -1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ (1/5) \cdot 2 \end{pmatrix}\)

For \((0, +)\): \([0B] = [0] + [B]\)

Find a Lyapunov function w.r.t. which the drift is negative.
The stability result requires a delicate proof (no product form).

Consider “drift” for all points:

For $(0, 0)$: $[0] = \begin{pmatrix} \alpha_A \\ 0 \end{pmatrix}$

For $(+, 0)$: $[0A] = [0] + [A]$ with 

$[A] = \mu_A \cdot \begin{pmatrix} -1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ (1/5) \cdot 2 \end{pmatrix}$

For $(0, +)$: $[0B] = [0] + [B]$ 

For $(+, +)$: $[0AB] = [0] + [A] + [B]$
Proof of the Stability Result

The stability result requires a delicate proof (no product form).

Consider “drift” for all points:

For \((0, 0)\): 
\[
[0] = \begin{pmatrix}
\alpha A \\
0
\end{pmatrix}
\]

For \((+, 0)\): 
\[
[0A] = [0] + [A] \quad \text{with} \\
[A] = \mu_A \cdot \left( \begin{pmatrix}
-1 \\
0
\end{pmatrix} + \begin{pmatrix}
0 \\
(1/5) \cdot 2
\end{pmatrix} \right)
\]

For \((0, +)\): 
\[
[0B] = [0] + [B]
\]

For \((+, +)\): 
\[
[0AB] = [0] + [A] + [B]
\]

Find a Lyapunov function w.r.t. which the drift is negative.
Key steps:

- Construct **piecewise-linear Lyapunov function** w.r.t. which the drift is negative almost everywhere (hard).
  - use throughput $\lambda$ and apply Farkas’ lemma (wouldn’t work for general stochastic Petri nets)
- Smooth the Lyapunov function (standard)
- Derive (strong) stability using Foster’s criterion (standard)
Continuous-Time Markov Decision Process – Infinite-State

Generalize balance equations:

Subdivide $\lambda_B$ in $\lambda_B = \lambda_{B,1} + \lambda_{B,2}$.

Intention: $\lambda_{B,i}$ = rate of $B$-tasks processed according to $\sigma_i$.

$$\lambda_A = \alpha_A + \lambda_{B,1} \cdot (1/6) + \lambda_{B,2} \cdot (2/3)$$

$$\lambda_{B,1} + \lambda_{B,2} = \cdots$$

Similarly to the uncontrolled case:

$\exists$ stabilizing scheduler

$\iff \exists$ solution $\lambda$ with $\lambda_A < \mu_A$ and $\lambda_{B,1} + \lambda_{B,2} < \mu_B$
Continuous-Time Markov Decision Process – Infinite-State
Generalize balance equations:
Subdivide $\lambda_B$ in $\lambda_B = \lambda_{B,1} + \lambda_{B,2}$.
Intention: $\lambda_{B,i} =$ rate of $B$-tasks processed according to $\sigma_i$.

$$\lambda_A = \alpha_A + \lambda_{B,1} \cdot (1/6) + \lambda_{B,2} \cdot (2/3)$$

$$\lambda_{B,1} + \lambda_{B,2} = \cdots$$

Similarly to the uncontrolled case:

$\exists$ stabilizing scheduler

$\iff \exists$ solution $\lambda$ with $\lambda_A < \mu_A$ and $\lambda_{B,1} + \lambda_{B,2} < \mu_B$  

LP!!
General Main Result

Theorem

Given a controlled BQN.

1. It is decidable in polynomial time whether there exists an (arbitrary) stabilizing scheduler.

2. If it exists, one can compute in polynomial time a static randomized scheduler, which is stabilizing in a strong sense, i.e., in steady state there is an exponential moment of the total queue size.

The theorem implies:

Any stabilizing scheduler can efficiently be made

- static and – at the same time –
- “strongly” stabilizing.
Queueing networks can be used to model parallelism.

Classical Jackson networks lack branching and control.

→ New model: Branching Queueing Networks

Stability and existence of stabilizing schedulers can be determined in polynomial time.

If a stabilizing scheduler does exist, a static randomized scheduler suffices and can be computed in polynomial time.
Queueing networks can be used to model parallelism.

Classical Jackson networks lack branching and control.

New model: Branching Queueing Networks

Stability and existence of stabilizing schedulers can be determined in polynomial time.

If a stabilizing scheduler does exist, a static randomized scheduler suffices and can be computed in polynomial time.

Future work:

Performance beyond stability, e.g., long-term average queue size

Can non-static schedulers help to minimize it?
Thank you!