

A Continuous State Space Approximation for PEPA Queues

Ashok Argent-Katwala

Jeremy T. Bradley*

Abstract

PEPA Queues is a formalism that augments queueing networks with customers that have behavioural characteristics, as defined by PEPA components. PEPA Queues suffer from the traditional state-space explosion that affects both closed queueing networks and PEPA models. We discuss a possible solution to this with a continuous state-space approximation that allows us to derive an underlying mathematical model based on ordinary differential equations from a PEPA Queues model.

1 Introduction

PEPA Queues [1] is a behavioural extension of closed queueing networks. In PEPA Queues, the customers and servers are modelled as PEPA components and the servers can interact with queueing customers to determine how they are routed to other queues in the system.

2 A Rough Outline

In Hillston's original continuous state-space approximation of PEPA models [2], components which are grouped into self-synchronising sets:

$$P \parallel P \parallel \dots \parallel P \parallel P$$

are represented by a tuple which counts the number of P components that are in a given derivative state, that is:

$$(N(P), N(P_1), N(P_2), \dots, N(P_n))$$

where P has $(n + 1)$ derivative states (including itself). A set of ordinary differential equations (ODEs) can then be generated to show how the quantities $N(P_i)$ vary over time.

*Department of Computing, Imperial College London. Email: {ashok,jb}@doc.ic.ac.uk

In this continuous approximation of PEPA Queues we create a similar abstraction, by creating a continuous approximation of the number of components in the queue as well as the derivatives of those components.

So given a PEPA Queue with server process, S , and a buffer with components P and Q as customers, where P and Q have derivatives P' and Q' respectively:

$$[P, P, P, Q, Q, P] \underset{L}{\bowtie} S \quad (1)$$

We might approximate this with two tuples, one for P derivatives and one for Q :

$$[(N(P), N(P')), (N(Q), N(Q'))] \underset{L}{\bowtie} S$$

where the initial buffer state would look like:

$$\underbrace{[(3, 0)]}_P, \underbrace{[(2, 0)]}_Q \underset{L}{\bowtie} S$$

Clearly this could be made to work with the ODE generation of [2] (if there is no cooperation between customers in the queue) but does not maintain the spirit of queueing order (or service discipline, usually first-come-first-served, by default), since we lose the ordering of customers in the queue with such an abstraction.

We make this more queue-like by splitting the PEPA queue buffer into two parts or *buckets* (maybe more than two if it would improve the approximation). The first bucket or *server bucket* is of fixed size and is closest to the server. It has access to the server and customers are serviced from this bucket alone; the size of the server bucket can be thought to represent the number of customers that can be serviced simultaneously, as in an M/M/n queue. The second bucket or *feeder bucket*, also of fixed size making up the remainder of the buffer, cannot see the server directly but serves to fill up the first bucket as customers from the server bucket get serviced and leave the queue.

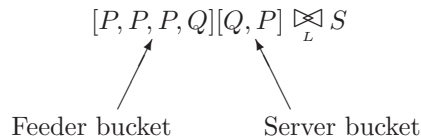


Fig. 1. A partition of the buffer queue into buckets

Thus the PEPA Queue buffer of Equation (1) might be partitioned as in Figure 1, where the front $[Q, P]$ components comprise the server bucket, and the back $[P, P, P, Q]$ bucket acts as the feeder bucket. Incoming customers of either P or Q fill up the server bucket if there is room or the second bucket if there is not (or are blocked if both buckets are full).

It is important to understand that there is no ordering of customers within a bucket (as to do so would necessitate keeping track of individuals, which is exactly what we try to avoid with continuous state-space representation). However, it may be appropriate to increase the number of buckets in a queueing buffer to achieve a better approximation of FCFS. For the moment we will keep it simple by having only two buckets, a server bucket and a feeder bucket.

3 A Simple Example

Take a simple PEPA Queue, *Coke*, with buffer size 10 which has only components of type *Coin* arriving. Initially, we have no arrival process (we will show how this is modelled later). The *Coin* process is specified as follows:

$$\begin{aligned} \textit{Coin} &\stackrel{\textit{def}}{=} (\textit{insert}, \mu).\textit{Coin}_1 \\ \textit{Coin}_1 &\stackrel{\textit{def}}{=} (\textit{roll}, \gamma).\textit{Coin}_2 \\ \textit{Coin}_2 &\stackrel{\textit{def}}{=} (\textit{accept}, \gamma).\textit{Coin}_1 + (\textit{clear}, \top).\textit{Coin} \end{aligned}$$

Let us specify that the server bucket is of size 3 and the *Coke* queue has the following format:

$$[-, -, -, -, -]_{fdr} [-, -, -]_{srv} \boxtimes_L S \xrightarrow{(\textit{clear}, \mu)}$$

where $[-]$ represents an empty buffer slot and $L = \{\textit{clear}\}$.

The server process S allows *Coin* components in the server bucket, $[\dots]_{srv}$, to leave the queue once a *clear* event occurs:

$$S \stackrel{\textit{def}}{=} (\textit{clear}, \top).S$$

Working on the same continuous state-space premise as PEPA, we count the number of components in each bucket with:

$$\begin{aligned} &[(N(\textit{Coin}, fdr, t), N(\textit{Coin}_1, fdr, t), N(\textit{Coin}_2, fdr, t))]_{fdr} \\ &[(N(\textit{Coin}, srv, t), N(\textit{Coin}_1, srv, t), N(\textit{Coin}_2, srv, t))]_{srv} \boxtimes_L S \quad (2) \end{aligned}$$

Based on the top level PEPA Queue model, we can construct coupled differential equations to describe how the number of each component, $N(\cdot, x, t)$ varies in each bucket x , with time, t . In general, this construction is generated by:

$$\begin{aligned} N'(X, B, t) = & \quad \text{rate of change of cpt. } X \text{ in bucket } B \\ & - \text{rate of departure of cpt. } X \text{ from bucket } B \\ & + \text{rate of arrival of cpt. } X \text{ into bucket } B \text{ from upstream bucket} \end{aligned}$$

The rate of change of a component type within a bucket is determined by the allowed component evolutions within the context of the PEPA Queue it is in. The rate of departure of a component type from a bucket is either the rate of departure of that component type from the queue if it is in the server bucket or it is the rate of transfer of that component type to a downstream bucket. For the server bucket, the component that may depart the queue is defined by the PEPA Queue process. For a component type in one of the feeder buckets, the rate of flow into the downstream bucket is a proportion of the total component exit rate from the queue – i.e. as components leave the queue (from the server bucket), this will trigger a flow of components from upstream to downstream buckets within the PEPA Queue. The exact proportion of the overall departure rate that governs this flow is given by the proportion of components of that type

that exist in that bucket, $R(X, B, t)$, defined below.

$$\begin{aligned}
N'(Coin, srv, t) &= -\mu N(Coin, srv, t) + \mu I(srv, t) R(Coin, fdr, t) \\
N'(Coin_1, srv, t) &= -\gamma N(Coin_1, srv, t) + \mu N(Coin, srv, t) \\
&\quad + \mu I(srv, t) R(Coin_1, fdr, t) \\
N'(Coin_2, srv, t) &= -(\gamma + \mu) N(Coin_2, srv, t) + \gamma N(Coin_1, srv, t) \\
&\quad + \mu I(srv, t) R(Coin_2, fdr, t) \\
N'(Coin, fdr, t) &= -\mu N(Coin, fdr, t) - \mu I(srv, t) R(Coin, fdr, t) \\
N'(Coin_1, fdr, t) &= -\gamma N(Coin_1, fdr, t) + \mu N(Coin, fdr, t) \\
&\quad - \mu I(srv, t) R(Coin_1, fdr, t) \\
N'(Coin_2, fdr, t) &= -\gamma N(Coin_2, fdr, t) + \gamma N(Coin_1, fdr, t) \\
&\quad - \mu I(fdr, t) R(Coin_2, fdr, t)
\end{aligned}$$

where $I(srv, t)$ is an indicator function which determines whether there is any space in the srv bucket for components in the fdr bucket to flow into; defined by:

$$\begin{aligned}
I(B, t) &= I(N(-, B, t) > 0) \\
N(-, B, t) &= n_B - \underbrace{\sum_i N(X_i, B, t)}_{\text{no. of blank slots available in } B \text{ at time, } t}
\end{aligned}$$

where n_B is the size of bucket B and $R(X, B, t)$ gives the ratio of component B that currently exists in the bucket B relative to all the other components in bucket B .

$$R(X, B, t) = \frac{N(X, B, t)}{\sum_i N(X_i, B, t)}$$

3.1 Adding an arrival process

Adding an arrival process for component X to the PEPA Queue above has the effect of adding an extra term to each occurrence of X (that an arrival is possible for) in the existing ODEs. Each term represents the possibility that that bucket might be capable of receiving the incoming X component from the sending queue.

Let's assume for simplicity, in the above case that we have an arrival process from a source of $Coin$ components that feeds in to the Coke queue at rate δ ; shown in Figure 2.



Fig. 2. An arrival process feeding into the 2-bucket Coke queue

This isn't strictly valid PEPA Queues notation but will serve at this stage to illustrate how arrivals from other queues can be handled.

The ODEs for the modified process to include an arrival source are identical except for the $N'(Coin, srv, t)$ and $N'(Coin, fdr, t)$ terms. These receive the incoming *Coin* flux conditional of their being room in the queue.

$$\begin{aligned} N'(Coin, srv, t) &= -\mu N(Coin, srv, t) + \mu I(srv, t) R(Coin, fdr, t) \\ &\quad + \delta I(srv, t) I(N(fdr, t) = 0) N(source, t) \\ N'(Coin, fdr, t) &= -\mu N(Coin, fdr, t) - \mu I(srv, t) R(Coin, fdr, t) \\ &\quad + \delta(1 - I(srv, t)) I(fdr, t) N(source, t) \end{aligned}$$

Entry to the *srv* bucket from the external source is conditional on there being an empty *fdr* bucket and space in the *srv* bucket. Whereas entry to the *fdr* bucket is conditional on there being a full *srv* bucket and space in the *fdr* bucket. Here, $N(B, t)$ is shorthand for $\sum_i N(X_i, B, t)$ and counts the number of components in the bucket B at time t .

The value $N(source, t)$ represents the number of concurrently serviced *source* components. For a simple source $N(source, t) = 1$ for all t , but in a realistic PEPA Queue, where the output from one queue feeds into the buffer of another, we would likely see a term comparable with the outgoing flux from our Coke queue.

The outgoing flux from any queue that feeds into another will also need conditioning on there being room in the last bucket of the downstream queue.

4 Conclusion

In this short paper, we have outlined some ideas as to how to apply the method of continuous state space approximation to PEPA Queues. A queue buffer can be approximated by multiple buckets where a continuous count of components is kept. Each bucket feeds into its downstream bucket in a given buffer determined by the rate of departure from the front of the buffer.

We have shown how ODEs can be constructed by hand for a queue with two buckets and an incoming source. Clearly work remains to extend this to multiple component-types, multiple buckets and a proper queueing network – not to mention automating the ODE construction process, itself.

References

- [1] A. Argent-Katwala, *A compositional, collaborative performance pipeline*. PhD thesis, Imperial College, London, United Kingdom, January 2006.
- [2] J. Hillston, “Fluid flow approximation of PEPA models,” in *QEST'05, Proceedings of the 2nd International Conference on Quantitative Evaluation of Systems*, (Torino), pp. 33–42, IEEE Computer Society Press, September 2005.