Application of Network Calculus Models to Heterogeneous Streaming Applications

Clayton J. Faber
Roger D. Chamberlain

Clayton J. Faber and Roger D. Chamberlain, “Application of Network Calculus Models to Heterogeneous Streaming Applications,” in *Proc. of IEEE International Parallel and Distributed Processing Symposium Workshops* (IPDPSW), May 2024, pp. 198-201.
DOI: 10.1109/IPDPSW63119.2024.00057

Presented at 26th Workshop on Advances in Parallel and Distributed Computational Models, San Fransisco, CA, USA.

Dept. of Computer Science and Engineering
Washington University in St. Louis

Application of Network Calculus Models to Heterogeneous Streaming Applications

Clayton J. Faber*
SimpleRose
clayton@simplerose.com

Roger D. Chamberlain
Dept. of Computer Science and Engineering
Washington University in St. Louis
roger@wustl.edu

***Abstract*—Network calculus has seen extensive use in the performance modeling of communications systems. Here, we apply network calculus techniques to the modeling of streaming data applications running on heterogeneous computing platforms. We quantitatively compare the performance predictions from network calculus with predictions from a discrete-event simulation model and a previously presented queuing theory model.**

***Index Terms*—performance modeling, network calculus, streaming data systems**

I. Introduction

In streaming data applications it can be difficult to reason about performance prior to deployment. Moreover, the application might need to be expanded to transform data into the appropriate input format or handle metadata processing. If one chooses to accelerate stages using heterogeneous hardware, data movement also becomes a concern when it is migrated to a new memory domain or the data movement takes place between physically separate networked compute resources.

When reasoning about performance in streaming applications it is often helpful to utilize analytical models to gain insights into how to spend time and development resources prior to a full deployment test. Queuing models have a long history for this purpose. One can readily apply queuing theory models to these streaming applications. Padmanabhan et al. [1] utilized queuing models to reason about streaming applications on heterogeneous architectures. These models utilize isolated measurements of individual compute stages and flow analysis to identify bottlenecks and identify where developers can focus their attention for performance improvements. While this technique provides mean flow analysis, without further effort to characterize both arrival and service distributions, it can be difficult to determine bounds on time and buffering requirements. It can also be difficult to capture the effects of data bundling between stages, and how the delay of waiting for jobs effects the overall throughput. To answer these shortcomings we turn to network calculus.

Network calculus is an analytic modeling technique that applies system theory concepts to computer communication networks utilizing min-plus algebra [2]–[4]. Although the technique was originally designed with network elements in mind, in this paper we propose additions to the network calculus modeling technique to reason about streaming data applications in a heterogeneous environment where the movement of data between memory domains and across network links is of vital importance. With modifications to support computational elements, deterministic network calculus models can give insights into delay due to data aggregation at nodes (packetization), end-to-end delay, and bounds on data flow through the overall application or through individual subsets of stages in the stream. Furthermore, the models retain the desirable property of being derived from measurements taken in isolation without a full deployment, similar to the aforementioned queuing theory models.

Here, we propose the use of network calculus modeling techniques to analyze the performance of streaming data computations. We include in the network calculus models elements of both data communication (which is typical of network calculus) and data computation (which is new). We compare the proposed network calculus models to a discrete-event simulation designed to mimic a pair of computation nodes with multiple stages in a streaming data application. In the example application we model two types of communication links: traditional network links and PCIe buses, in addition to discrete compute components as data streams though them. Through both modeled results and simulated results we show the utility of network calculus when considering performance in streaming data applications.

II. Background and Related Work

Network calculus is a modeling approach that is designed to analyze systems that utilize queues and has historically been primarily used to analyze bounds and model performance in networking systems. It relies on the min-plus and max-plus algebras. In min-plus algebra, addition is replaced by the infimum operator and multiplication is replaced with addition. Similarly in max-plus algebra, addition is replaced by the supremum and multiplication is replaced with addition. These two algebras are used in conjunction with the convolution operator to reason about data as it traverses a system.

In network calculus, data are modeled by a cumulative function with respect to time to represent the flow in and out of systems. Systems are modeled in a similar fashion with curves representing guarantees on flow into and out of the system, known as arrival and service curves, respectively.

Consider a data flow, in units of bits, $r(t)$, arriving at a system and let $\alpha(t)$ be a wide-sense increasing function with

*C.J. Faber's work was completed during PhD studies at Washington University in St. Louis.

979-8-3503-6460-6/24/$31.00 ©2024 IEEE
DOI 10.1109/IPDPSW63119.2024.00057

$\alpha(0) = 0$. The flow is constrained by $\alpha(t)$ and is an arrival curve if and only if for any $0 \leq s \leq t$:

$$r(t) - r(s) \leq \alpha(t - s).$$

Following a similar logic the system offers a service guarantee for an output flow $r^*(t)$. Allow $\beta(t)$ to be a wide-sense increasing function and $\beta(0) = 0$. $\beta(t)$ is a service curve given to the flow $r(t)$ with an output curve $r^*(t)$, defined by:

$$r^*(t) \geq \inf_{s \leq t} \{r(s) + \beta(t - s)\}.$$

Alternatively, this can be written as the min-plus convolution:

$$r^*(t) \geq r(t) \otimes \beta(t).$$

Furthermore, we can define an upper-bound on service provided defined as:

$$r^*(t) \leq r(t) \otimes \gamma(t),$$

where $\gamma(t)$ is the maximum (i.e., best case) service curve.

For arrival curves it is common to model the data flow using an affine curve known as the leaky bucket arrival curve:

$$\alpha(t) = \begin{cases} R_\alpha \cdot t + b & \text{if } t > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Here, R_α represents the rate of arrival and b is a burst, that is, how much data can be sent instantaneously. When considering the service curves, these are commonly represented as rate latency functions with an associated rate, R_β, and delay, T, associated with them:

$$\beta(t) = \begin{cases} R_\beta \cdot (t - T) & \text{if } t > T \\ 0 & \text{otherwise.} \end{cases}$$

By utilizing these models we can reason about bounds on a specific node such as the backlog generated by the flow entering the node, the delay data will experience at a given node, and what is the upper-bound output flow of the node. Figure 1 displays a data over time plot of a leaky-bucket arrival curve and two rate latency functions representing both a maximum and normal service curve adapted from [2]. Also in this figure horizontal and vertical lines are included that are meant to represent maximum virtual delay and backlog respectively. Finally from these two lines we can derive an output flow bound, $\alpha^*(t)$, which, along with the delay and backlog, will be further expanded upon below.

The use of network calculus is widespread in networking systems [5]–[7]. These applications, are mostly concerned with extensions to other models, such as network firewalls [8] and job scheduling [9]. Network calculus also has two subbranches; one that deals with probabilistic systems, called stochastic network calculus [10], and the other dealing with hard real-time deadlines, known as real-time network calculus [11]. In this particular work we use the standard, deterministic, network calculus and this is, as far as we know, the first application of these models to streaming computations that specifically target heterogeneous architectures.

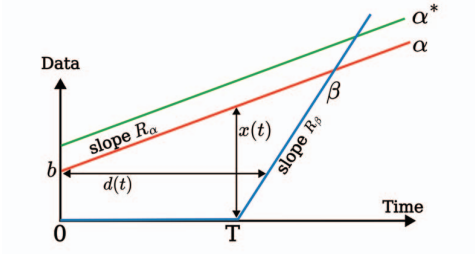

Fig. 1. Plot of a Leaky Bucket Arrival Curve, α, and a Rate-Latency Service Curve, β, showing the relation of the Backlog, $x(t)$, Virtual Delay, $d(t)$, and Output Flow, α^*, bounds. Adapted from [2].

There are of course other examples of modeling for heterogeneous architectures that use other types of models. Faber et al. [12] apply a queuing network model to perform flow analysis, estimating roofline performance for the overall application. In that study, the actual performance was almost 30% lower than what the roofline model predicted. When considering queuing theory models it is important to point out that both network calculus and queuing theory as mathematical models are designed to reason about queuing systems and there has been work to explain how one represents network calculus ideas in a queuing theory space [13], [14].

III. Network Calculus Modeling

As mentioned prior we want to use network calculus to reason about bounds on a given streaming application that utilizes heterogeneous architectures, however some additional assumptions and modifications to the standard model must be made in order to utilize it properly. Firstly network calculus in its original inception deals with continuous data flows that are bit-by-bit, however in the modern era a majority of network equipment work on a per-packet scheme and are similar to jobs flowing through a streaming application. This packetization does indeed have an effect on some of the properties that network calculus models [2] and needs to be accounted for in our final model as well. These adjustments come in the form modifying the arrival and service curves with a variable that describes the size of the maximum packet l_{max}. Consider a flow $r(t)$ and a packetizer, P^L, the packetized version of the arrival, service, and maximum service curves are [15]:

$$P^L(r(t)) \leq \alpha(t) + l_{max} 1_{t>0}$$

$$\beta'(t) = [\beta(t) - l_{max}]^+$$

$$\gamma'(t) = \gamma(t).$$

where $1_{t>0}$ is 0 for $t \leq 0$ and 1 for $t > 0$.

With these adjustments we can now talk about important bounds previously mentioned and shown in Figure 1, virtual delay and backlog. The virtual delay, $d(t)$, is a measure of the maximum amount of time it takes for a system to output the same amount of data sent to the system. For a leaky bucket

arrival curve α and a rate latency service curve β the virtual delay is given by:

$$d(t) \leq T + \frac{b}{R_\beta},$$

where T is the delay in the expression for β and b is the burst size in the expression for α. The backlog bound, $x(t)$, is a bound on the maximum amount of data that resides in the server before output is sent, and is calculated as the maximum deviation between α and β. In this example it is calculated as:

$$x(t) \leq b + R_\alpha \cdot T.$$

Finally we can make an estimation on the output bound on a system, $\alpha^*(t)$. This is known as the output flow bound. This is found by calculating both a min-plus convolution and a min-plus de-convolution utilizing the arrival curve of the node and both the maximum and normal service curves:

$$\alpha^* = (\alpha \otimes \gamma) \oslash \beta.$$

While these bounds are beneficial to have, it is important to know that these bounds assume that $R_\alpha \leq R_\beta$. If $R_\alpha > R_\beta$ it is noted in [2] that the bounds are infinite, which is the same result predicted by queuing theory if the arrival rate is greater than the service rate, resulting in an infinite bound on the queue. Taking this into account, there are three particular scenarios that we are interested in, when $R_\alpha < R_\beta$ or standard operation, when $R_\alpha = R_\beta$, and finally when $R_\alpha > R_\beta$. While the bounds are indeed infinite for backlog and virtual delay over the long run, we hypothesize that we can use values given by the model to understand estimates on required queue size for individual nodes as a job traverses a system implementing a streaming data application.

One important aspect of targeting heterogeneous architectures is the need to gather enough data to make dispatching a job worthwhile. The inherent overheads associated with initiating a computation on an attached accelerator, for example, can motivate the aggregation of a minimum data volume at the input to the accelerator prior to dispatching the job to the accelerator. We call this metric the job ratio. To reflect this in the service curve representations, we have made a modification to how initial delay is calculated at these nodes. For a node n that collects data of size b_n prior to initiation and b_n is larger than the burst rate of the previous node ($b_n > b^*_{n-1}$) then the latency at node n is:

$$T_n^{tot} = T_{n-1}^{tot} + \frac{b_n}{R_{\alpha_{n-1}}} + T_n.$$

Intuitively, total latency is the summation of initial delay of the previous nodes, T_{n-1}^{tot}, the time to collect a job from the previous node, $b_n/R_{\alpha_{n-1}}$, and finally the initial delay of the current node, T_n.

IV. Modeling and Simulation

Actual streaming data applications are often modeled as a chain of nodes interconnected into a directed acyclic graph. The model nodes in the chain might represent computation and/or communications (as described in [16]), especially given that data movement in a heterogeneous environment can be critical for performance. We believe that network calculus is well suited for capturing this type of data movement and can be a viable tool for measuring the effects of data channels in streaming environments.

Given a set of N nodes representing stages of a heterogeneous streaming application, we can create network calculus maximum and normal service curves to represent the guarantees on service at each node. Along with the actual compute nodes we can also create service curves to represent grantees on data movement. These nodes can be concatenated together to find the overall service curve of the full system. Going further, we can create models for intermediate systems by finding service curves for a subset of contiguous nodes.

To test this model we use the BLAST application [17] described in Faber et al. [12]. Figure 2 illustrates the setup of this application with nodes representing stages of the streaming data application.

Node	Function	Node	Function
Λ	data source	D	Network link
A	PCIe link	E	PCIe link
B	FPGA computation	F	GPU computation
C	PCIe link	G	PCIe link

Fig. 2. Data flow diagram for BLAST. Nodes represent computations or communications, and the job ratio is shown below each node. Node D decomposes large data blocks from the FPGA for delivery over the network, and Node E composes even larger data blocks for delivery to the GPU.

We take these nodes as described and model their execution time in a discrete-event simulator facilitated by the SimPy library [18] in Python3. Each node is given a maximum and minimum execution time, a data packet size to consume, and data packet size to emit when the execution time has completed. The time chosen for execution is chosen from a uniform random distribution using the minimum and maximum times as bounds. Following the approach of Timcheck and Buhler [19], we normalize the data volumes at each stage referred to the input, as some stages have a natural lossless data compression. In the results section we report all of the following: network calculus predictions on bounds, the results of the original $M/M/1$ queuing theory model and empirical measured performance (from [12]), and our simulated system performance.

V. Results

The predictions from our network calculus model and discrete-event simulation are depicted in Figure 3. The service curve, represented by $\beta(t)$, corresponds to the lower bound of predicted performance. The arrival curve, represented by $\alpha(t)$, corresponds to an upper bound on performance. The output

flow bound, represented by $\alpha^*(t)$, is a loose upper bound. The simulated data output is shown by the stairstep curve that stays between the two bounds.

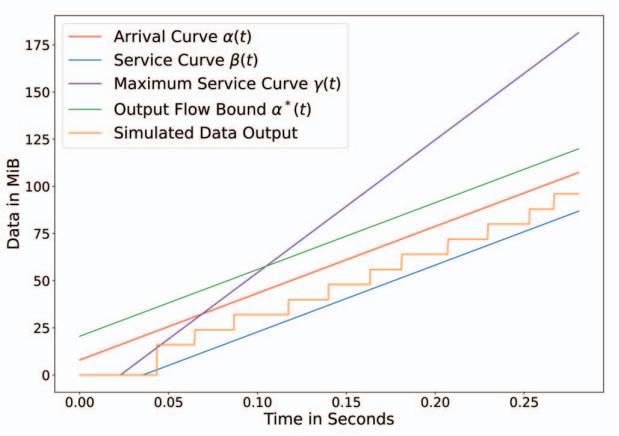

Fig. 3. Network calculus model results.

Throughput predictions from the various models and experiments are presented in Table I. As is apparent, the network calculus throughput predictions align well with both the discrete-event simulation results and the empirical results reported in [12].

TABLE I
STREAMING DATA APPLICATION THROUGHPUT.

Source	Value
Network calculus upper bound	704 MiB/s
Network calculus lower bound	350 MiB/s
Discrete-event simulation model	353 MiB/s
Queuing theory prediction [12]	500 MiB/s
Measured throughput [12]	355 MiB/s

While these throughput results are clearly of interest, they haven't yet demonstrated the power of network calculus, since they are merely confirming the conclusions from previous models. Additional information we can glean from the network calculus model include the following:

1) The maximum virtual delay, d, through the system is modeled to be 46.9 ms.
2) The maximum data occupancy resident in the system (or backlog bound), x, is modeled as 20.6 MiB.

Points (1) and (2) above are corroborated by the discrete-event simulation model.

Further capabilities of the network calculus models include the ability to analyze any desired subset of the streaming application separate from the rest of the application. For example, the contributions of the data occupancy bounds that are due to each node in Figure 2 can be determined analytically, which can assist a developer in allocating buffers.

VI. Conclusions and Future Work

This paper has presented the use of network calculus models to bound the performance of streaming data applications executing on heterogeneous execution platforms. The models provide both upper bounds and lower bounds for throughput as well as latency and data volume. For an example streaming application, the bounds are tight enough to be helpful in understanding the performance implications of candidate design changes, and appropriately bracket a discrete-event simulation model of the same application.

In future work, we would like to explore the use of these models on a wider set of streaming applications, and validate the models over a wider range of empirical experiments. In addition, it is possible to use network calculus to guide the sizing and allocation of buffers within the application.

Acknowledgments

This research was supported by NSF under grant CNS-1763503 and a gift from BECS Technology, Inc.

References

[1] S. Pamanabhan *et al.*, "Optimal design-space exploration of streaming applications," in *Proc. of IEEE Int'l Conf. on Application-specific Systems, Architectures and Processors*, Sep. 2011, pp. 227–230.
[2] J.-Y. Le Boudec and P. Thiran, *Network Calculus*. Springer, 2003.
[3] R. Cruz, "A calculus for network delay. I. Network elements in isolation," *IEEE Trans. Inf. Theory*, vol. 37, no. 1, pp. 114–131, 1991.
[4] ——, "A calculus for network delay. II. Network analysis," *IEEE Trans. Inf. Theory*, vol. 37, no. 1, pp. 132–141, 1991.
[5] S. Azodolmolky *et al.*, "An analytical model for software defined networking: A network calculus-based approach," in *Proc. of Global Comm. Conf.* IEEE, 2013, pp. 1397–1402.
[6] J.-Y. Le Boudec, "Application of network calculus to guaranteed service networks," *IEEE Trans. Inf. Theory*, vol. 44, no. 3, pp. 1087–1096, 1998.
[7] J. B. Schmitt and U. Roedig, "Sensor network calculus–a framework for worst case analysis," in *Proc. of Int'l Conf. on Distributed Computing in Sensor Systems*. Springer, 2005, pp. 141–154.
[8] Z. Wang, J. Zhang, and T. Huang, "Determining delay bounds for a chain of virtual network functions using network calculus," *IEEE Communications Letters*, vol. 25, no. 8, pp. 2550–2553, 2021.
[9] M. Li, G. Zhu, and Y. Savaria, "Delay bound analysis for heterogeneous multicore systems using network calculus," in *Proc. of 13th Conf. on Industrial Electronics and Applications*. IEEE, 2018, pp. 1825–1830.
[10] Y. Jiang and Y. Liu, *Stochastic Network Calculus*. Springer, 2008.
[11] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *Proc. of Int'l Symp. on Circuits and Systems*, vol. 4. IEEE, 2000, pp. 101–104.
[12] C. Faber, T. Plano, S. Kodali, Z. Xiao, A. Dwaraki, J. Buhler, R. Chamberlain, and A. Cabrera, "Platform agnostic streaming data application performance models," in *Proc. of IEEE/ACM Workshop on Redefining Scalability for Diversely Heterogeneous Architectures*, Nov. 2021.
[13] Y. Jiang, "Network calculus and queueing theory: Two sides of one coin," in *Proc. of Int'l Conf. on Perf. Eval. Meth. and Tools*, 2010.
[14] K. Pandit, J. Schmittt, and R. Steinmetz, "Network calculus meets queueing theory - a simulation based approach to bounded queues," in *Proc. of 12th Int'l Workshop on Quality of Service*, 2004, pp. 114–120.
[15] A. Van Bemten and W. Kellerer, "Network calculus: A comprehensive guide," Technical Univ. of Munich, Tech. Rep. 201603, 2016.
[16] J. Beard and R. Chamberlain, "Analysis of a simple approach to modeling performance for streaming data applications," in *Proc. of Int'l Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, Aug. 2013, pp. 345–349.
[17] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, 1990.
[18] Team SimPy, "SimPy: Discrete event simulation for Python," https://simpy.readthedocs.io, 2023, accessed Aug. 2023.
[19] S. Timcheck and J. Buhler, "Reducing queuing impact in streaming applications with irregular dataflow," *Parallel Computing*, vol. 109, p. 102863, Mar. 2022.