Analysis of a Simple Approach to Modeling Performance for Streaming Data Applications

Jonathan C. Beard
Roger D. Chamberlain

Jonathan C. Beard and Roger D. Chamberlain, "Analysis of a Simple Approach to Modeling Performance for Streaming Data Applications," in Proc. of IEEE Int'l Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), August 2013, pp. 345-349.

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

2013 IEEE 21st International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems

Analysis of a Simple Approach to Modeling Performance for Streaming Data Applications

Jonathan C. Beard and Roger D. Chamberlain
Dept. of Computer Science and Engineering
Washington University in St. Louis
Email: {jbeard,roger}@wustl.edu

***Abstract*—Current state of the art systems contain various types of multicore processors, General Purpose Graphics Processing Units (GPGPUs) and occasionally Digital Signal Processors (DSPs) or Field-Programmable Gate Arrays (FPGAs). With heterogeneity comes multiple abstraction layers that hide underlying complexity. While necessary to ease programmability of these systems, this hidden complexity makes quantitative performance modeling a difficult task. This paper outlines a computationally simple approach to modeling the overall throughput and buffering needs of a streaming application deployed on heterogeneous hardware.**

***Keywords*—computer performance; analytical models;**

I. Introduction

In search of ever higher performance, computer architectures have diversified to include a wide variety of heterogeneous hardware such as traditional multicore processors, GPGPUs, DSPs and FPGAs. Presented with multiple execution platforms, developers need reliable and computationally tractable models to predict performance. This paper explores an analytic model that is computationally simple and widely applicable to applications that are within the streaming data paradigm. Validation is performed across heterogeneous hardware resources, a pair of real and multiple synthetic streaming applications. When a model will fail is as important as when a model will succeed. To this end we seek to determine when this set of simple models can and cannot be trusted.

Stream processing is a computing paradigm that views applications as sets of pipelined kernels connected by streams of data. Streaming applications can be thought of as a series of queues and servers. Each compute kernel is a server which draws data from a queue. Many earlier works, including Schweitzer [11], describe how maximum throughput can be determined analytically for a finite-capacity open queueing network. These works have shown that queueing networks can be used for modeling throughput, however they assume that queue (buffer) capacity is known.

Queueing networks have a close relationship with flow networks. Work by Pourbabai [10] utilizes a maximum flow model to solve a queueing network with side constraints. Unlike the application studied in [10], computer programs have data-flow routing constraints that are critical to application correctness. Without additional constraints, a typical maximum flow problem formulation assumes that any path from source to sink can be taken; data-flow constraints are ignored. The flow model used here places data-flow constraints on the graph which are directly derived from the modeled application.

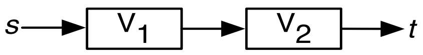

Fig. 1. Initial application graph G_A with two compute kernels V_1 and V_2, a data source s, and a data sink t.

Many applications exhibit some form of filtering, that is they either increase or decrease the volume of data as they process it. This phenomena is more formally termed gain or loss, respectively. Filtering presents a problem for standard maximum flow algorithms. This was solved by Jewell [8] and later with a polynomial solution by Goldfarb et al. [5]. Using the theoretical work of Jewell, the flow model presented here is a generalized flow network with a fixed branching probability.

II. The Model

A. Description

Given the throughput capacity into and out of each compute kernel within an application and the throughput achievable by each communications link, the model presented here calculates maximum flow of data through the overall network. Using a constrained generalized maximum flow network the model determines maximum flow through an application topology given a set of constraints. Utilizing a simple $M/M/1$ queueing model, it attempts to estimate the minimum required buffering capacity for each communication edge within the application.

An application graph topology G_A (Figure 1) is a connected directed graph consisting of each compute kernel within an application as a vertex V_i and every data-flow dependency as an edge $\overrightarrow{V_iV_j}$. An application topology also defines a (pseudo-) data source s and sink t. Since application topologies can have more than one actual data source and sink, the model inserts s with out-edges to all application kernels that have zero in-edges and t with in-edges from all application kernels with zero out-edges. Nodes s and t are modeled as having infinite capacity.

Every communications link is a distinct resource with its own service rate. This necessitates transforming G_A by adding additional vertices for each communications link which can be directly modeled as a queueing network G_Q (Figure 2). Every application kernel and communications link is a queue and server pair in G_Q. Formally G_Q is defined by the 4-tuple:

$$G_Q = (V_Q, E_Q, s \in V_Q, t \in V_Q)$$

where s is the source node and t is the termination (sink) node.

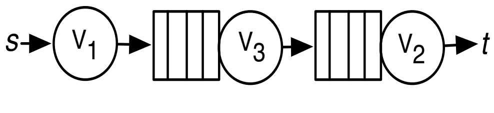

Fig. 2. The queueing network G_Q that arises from addition of a communications vertex, V_3, modeling the edge from V_1 to V_2 in G_A.

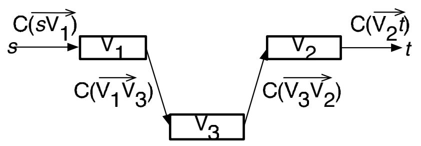

Fig. 3. The overall flow graph G_F with capacities C at each edge.

In a queueing network, two main parameters characterize the performance of the network: $\lambda(V_i)$ (the arrival rate of data at node V_i) and $\mu(V_i)$ (the service rate at node V_i). Nodes in V_Q that represent compute kernels have their service rates determined by measurement in isolation and are assumed to have non-blocking read and write behavior. At equilibrium, with no gain or loss, $\mu(V_i)$ is equal to the aggregate data ingest rate (with units of Bytes/s). The service rates of nodes in V_Q that represent communication links are determined from first principles (i.e., from performance specifications provided). The arrival rates $\lambda(V_i)$ will be derived from the flow model described below.

A flow graph is defined as a directed acyclic graph G_F (Figure 3) where each server in the queueing network (Figure 2) is represented as a vertex. Since G_Q is a case of an open Jacksonian network, G_F is constructed from G_Q by removing the queues on each edge $\overrightarrow{V_iV_j} \in E_Q$. Formally the flow graph is defined as a 7-tuple:

$$G_F = (V_F, E_F, s, t, C, \gamma, R)$$

$$V_F = V_Q, \quad E_F = E_Q,$$

where $C : E_F \to \Re_+$ represents the flow capacity of each edge (determined as described below), and $\gamma : V_F \to \Re_+$ represents the data volume gain or loss at each node. It is defined as the ratio of the mean data volume out of a node relative to the mean data volume in. A $\gamma < 1$ represents data loss (e.g., data compression) and a $\gamma > 1$ represents data gain (e.g., data expansion). For nodes representing compute kernels, these values are determined empirically, and for nodes representing communication links, $\gamma = 1$. $R : E_F \to (0, 1)$ represents the routing fraction associated with each out-edge $\overrightarrow{V_iV_j}$ of node V_i.

Given $\mu(V_i)$, $\gamma(V_i)$, and $R(\overrightarrow{V_iV_j})$ for each vertex and edge, the capacity C associated with each edge can be computed using Equation (1).

$$C(\overrightarrow{V_iV_j}) = \mu(V_i) \times \gamma(V_i) \times R(\overrightarrow{V_iV_j}). \tag{1}$$

Each edge in a flow graph is constrained by $C(\overrightarrow{V_iV_j})$. Note that the implicit assumption has been made that each compute kernel is mapped to a dedicated compute resource. Extensions for resource sharing are in the section below.

To calculate the maximum stable throughput the model maximizes Γ (the overall application throughput) and f (flow at every graph edge) subject to:

$$\sum_{j|(i,j)\in E_F} f(\overrightarrow{V_iV_j}) - \sum_{j|(j,i)\in E_F} f(\overrightarrow{V_jV_i}) = \begin{cases} + & i = s \\ 0 & i = \text{circulation} \\ - & i = t \end{cases} \tag{2}$$

$$f(\overrightarrow{V_iV_j}) \leq C(\overrightarrow{V_iV_j}) \tag{3}$$

$$\frac{f(\overrightarrow{V_iV_j})}{\sum_{x=1}^{N} f(\overrightarrow{V_iV_x})} = R(\overrightarrow{V_iV_j}). \tag{4}$$

Equation (2) states that flow must be conserved across all edges and that the only edges with positive or negative flow are adjacent to s and t respectively. Flow must be less than or equal to the capacity as shown in Equation (3). Equation (4) ensures that the data-routing is maintained across each edge.

To bound queue size, the model can be further constrained by ϕ, ensuring a smaller server utilization ($\rho = \lambda/\mu$) at each queueing station. This corresponds to maximizing Γ with the following constraint:

$$\rho(V_i) \leq \phi. \tag{5}$$

The value assigned to ϕ can be any value < 1. Once maximal values of $f(\overrightarrow{V_iV_j})$ have been calculated for every $\overrightarrow{V_iV_j} \in E_F$, these values can be used within the queueing model to determine the necessary buffering for the system at the calculated flow. To do this the relationship must be shown between $f(\overrightarrow{V_iV_j})$ and the queueing model parameters $\lambda(V_i)$. For queueing stations with multiple in-edges, these queues are treated as sub-queues of one larger queue. The relationship between maximized flows along each edge and λ is therefore

$$\lambda(V_j) = \sum_i f(\overrightarrow{V_iV_j}). \tag{6}$$

Our hypothesis is that the $M/M/1$ model gives an upper estimate of the queue occupancy, we expect the actual service time distributions to have a lower coefficient of variation than an exponential distribution. To estimate buffering capacity, we solve for queue occupancy K at a probability P_K that is close to zero as in Equation (7).

$$K(V_i) = \frac{\log(\frac{P_K}{1-\rho(V_i)})}{\log(\rho(V_i))} - 1. \tag{7}$$

The parameter K is very sensitive at low and high values of ρ and is also influenced by P_K (Figure 4). Values of P_K should be chosen based on overall throughput of the system, for our experiments we set $P_K = 10^{-7}$.

B. Sharing Models

In order to account for resource sharing, Equation (1) is modified to substitute μ_s for μ, reflecting the shared capacity. The sharing model for multicore processors is a fair sharing model:

$$\mu_s(V_i) = \mu(V_i)/n_p, \quad n_p = \text{ \# processes.} \tag{8}$$

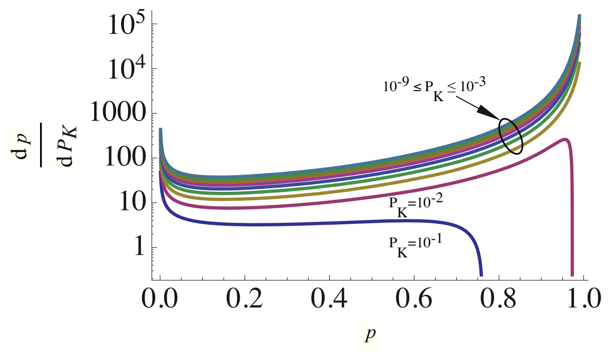

Fig. 4. This figure shows the change of slope for values of $10^{-9} \le P_K \le 10^{-1}$, demonstrating the relative stability of K for values of P_K with $.1 \le \rho < .7$. Once $\rho \ge .7$ a value of $0 < P_K \le 10^{-3}$ gives a much tighter bound on K. Also seen is the convergence of the slope towards $+\infty$.

FPGAs are assumed to be shareable in area, but not temporally. The sharing equation reflects that by giving each compute kernel mapped to an FPGA its full μ until all available gates are exhausted:

$$\mu_s(V_i) = \mu(V_i) \times a_i \tag{9}$$

where $a_i = 1$ if $\sum_{i=1}^{N} Area_i \le$ Available Area , else $a_i = 0$.

A PCI-X bus is used for multicore to FPGA communication. The PCI-X sharing model is also a fair sharing policy, but only until the bandwidth limit is reached:

$$\mu_s(V_i) = \mu(V_i)/n_c \tag{10}$$

where n_c is equal to the number of communication links sharing the bus.

C. Modeling Assumptions

The model presented above makes the following assumptions about the applications, graph topology and underlying hardware. (1) The application is assumed to be in equilibrium: the streaming computation paradigm is typically used in application domains that require high-throughput, high volume computation. On initial startup and termination non-steady state behavior is exhibited, however during the majority of the execution steady state behavior is typical. (2) The data volume into and out of each edge is measurable on the compute kernel in isolation (i.e., separated from the rest of the application topology). (3) Only non-blocking behavior exists, i.e. servers are allowed to process data as soon as it is present on its queue. (4) Data routing is independent of the state of the system, i.e. external signals don't influence removal of items from a queue, nor $R(\overrightarrow{V_iV_j})$. (5) All compute kernels are work conserving: when two compute kernels are mapped to the same resource, the work that is done by the compute kernel does not decrease. If two compute nodes are combined in such a way that overall work is less for the combined kernel than the two separate nodes then this is non-work conserving.

III. Model Evaluation Approach

In order to evaluate the model, two paths are taken. First a pair of real applications are used and second, a set of synthetic applications of varying topologies are generated. For each application, both real and synthetic, random mappings of application kernels to compute resources are generated and run on the hardware enumerated in Table I. Unless noted, the multi-level queue scheduler is used. The paragraphs below describe the tools, hardware, and methods used for evaluation.

TABLE I. Hardware used for empirical measurement

Name	Machine 1	Machine 2
CPU	12 x 2.4GHz AMD Opteron	4 x 3.1GHz Intel Xeon E3
FPGA	2 x Virtex-4 LX100	None
RAM	32GB DDR2	8GB DDR3

The Auto-Pipe [4] development environment is used for all experiments. The TimeTrial [9] performance monitor provides accurate measurements of queue occupancies and edge throughput. All applications and compute kernels (both real and synthetic) are expressed in combinations of C and VHDL and compiled with the GNU C compiler or synthesized with Synopsys Synplify Premier DP respectively. The GraphModeler [6] tool generates synthetic kernels, maps compute kernels, and executes the model.

The model uses measurements of each compute kernel running on its assigned hardware as input. To accomplish this, each compute kernel is instantiated in isolation using a test bench produced by GraphModeler that provides input to each in-edge and consumes all data on each out-edge. Throughput is measured using the TimeTrial monitoring system and recorded.

For a given application, each compute kernel can be run on many potential resources. GraphModeler uses a uniform random process to select hardware resources from the set in Table I, producing a set Ω of chosen resources. Once Ω is selected, the set of application compute kernels (V_A) must be mapped to it. To map V_A to Ω, kernels $\chi \in V_A$ (drawn uniformly from V_A) are selected and assigned to resources $\omega \in \Omega$ (again, drawn uniformly from Ω), $\forall \omega \in \Omega$. The mapping algorithm then assigns remaining kernels $\chi \in V_A$ to $\omega \in \Omega$ by randomly walking the in- and out-edges of previously mapped compute kernels until all compute kernels are mapped.

Whenever the verification of a model is based principally on empirical evidence, a primary consideration is the extent to which the test sets used are truly representative of the overall universe of possibilities. That concern is addressed through the use of several synthetic benchmarks generated by GraphModeler using topologies from [2] with parameters as specified in [1]. In addition, two real applications (JPEG encode and DES encrypt) are used for model evaluation. The JPEG encode application is implemented according to the specifications in [7]. The DES encrypt application is implemented according to the FIPS (46-3) standard [3]. The topology of each application is specified in the X language [4] which serves as input for GraphModeler.

IV. Empirical Results

A test application designed to run multiple processes on a single core is used to validate the processor sharing model. Each process is synchronized to start concurrently and runs for 2 minutes. Each time quantum is consumed by looping for 200 *no-op* instructions. Tests were run on both machines

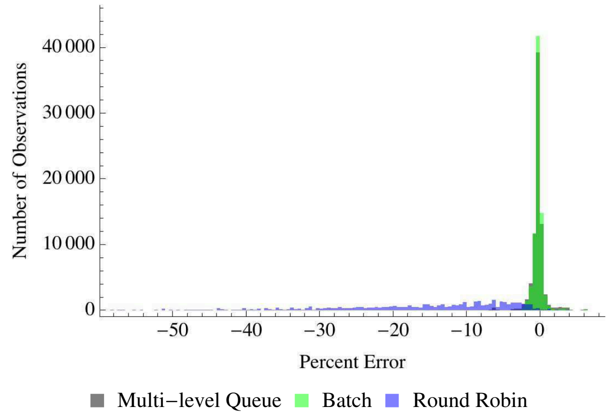

Fig. 5. Percent error for processor sharing model using three scheduling algorithms. All metrics are over 1 through 40 processes on one processor core. Model predicts executions per second. Error is calculated as $\frac{(\text{modeled rate} - \text{observed rate})}{\text{observed rate}}$. R^2 values for each scheduler are .999854, .999952, and .766693 for multi-level queue, batch, and round robin respectively.

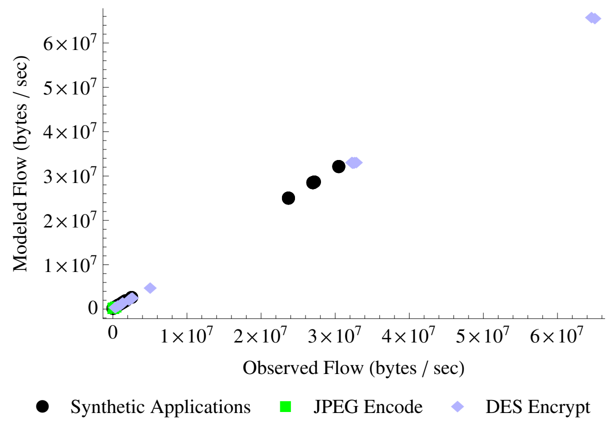

Fig. 6. Plot of modeled versus observed flow for all three application sets. Kernels were executed on FPGA and multicore processors.

listed in Table I. Three different scheduling algorithms (multi-level queue, batch and round robin) were chosen as they are representative of most modern systems.

In Figure 5 the processor sharing model validation percent error distribution is shown for predicted executions per second. The overall model vs. observed fit is quite good for the batch and multi-level queue scheduler. As we might expect, the round robin scheduler resulted in more variation than the other two scheduling algorithms due to fixed quantum sizing.

The flow model is validated with the set of applications described in Section III. Forty synthetic applications with 3 through 82 compute nodes were tested on Machines 1 and 2 (see Table I). Linear regression of the modeled versus observed flow rates across each for the combined synthetic, JPEG encode and DES encrypt gives an $r^2 = 0.999$; Figure 6 shows this relationship, and a histogram of relative error is shown in Figure 7.

As important as where the flow model succeeds is where it could fail. Given the percent error shown in Figure 7, it is assumed that the model is generally correct for the applications tested. To find some instances where the model might fail, the variation in percent error is examined. We hypothesize that some of the variation could be explained by compounding error as sharing increases for a given compute resource. Observing the correlation coefficient between the number of compute kernels assigned to each resource and the percent error could give an indication that this explanation is at least plausible. While the overall predictive quality of the flow model is high, the hypothesis that its imperfections are dominated by the sharing model isn't supported by the evidence given a relatively weak correlation of 0.317 for the combined set of synthetic and JPEG encode applications.

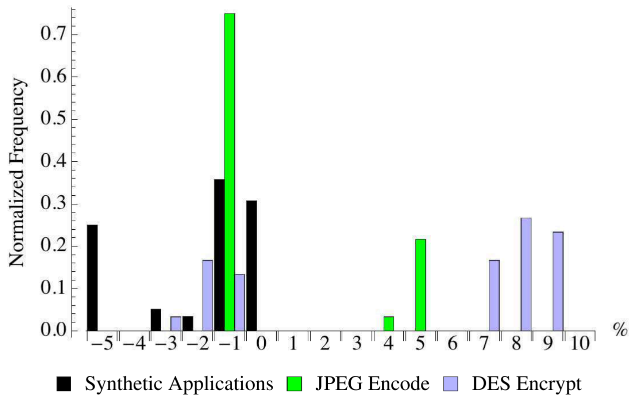

Fig. 7. Percent error for flow model for all three application sets, calculated as $\frac{(\text{modeled flow} - \text{observed flow})}{\text{observed flow}} \times 100$. Histogram bin size is 1%.

The $M/M/1$ queueing model assumes exponentially distributed arrival rates and service rates, while real service distributions are often closer to deterministic (i.e., have a much lower coefficient of variation than an exponential), even if not fully deterministic. This distinction is the basis for our hypothesis that the $M/M/1$ model will produce conservative estimates for the actual queue occupancy. Figure 8, which plots percent error of modeled maximum queue occupancy, implies that the model is excessive when predicting queue occupancies across the board.

Figure 4 suggests that solving for queue occupancy when $\rho < .9$ should produce a more stable result owing to the sensitivity of K to ρ. To better understand where the queueing model fails, a separate tandem queue micro-benchmark was constructed (see Figure 9). The micro-benchmark was run on multicore processors with a multi-level queue scheduler.

Figure 10 compares the observed queue occupancy for the micro-benchmark to modeled predictions.We conclude that not only does the model not match the empirical results, the errors are negative (i.e., the measured max queue occupancy exceeds the model predictions). While the results for low ρ shown in Figure 10 have negative errors, the even larger positive errors present in Figure 8 are for values of ρ near 1. As a percentage, these values are large because they are normalized to the smaller, measured quantity. When the errors are negative, they are often significantly negative, with the percent error bounded at -100% simply by the normalization. The take-home message is that the $M/M/1$ queueing model is simply

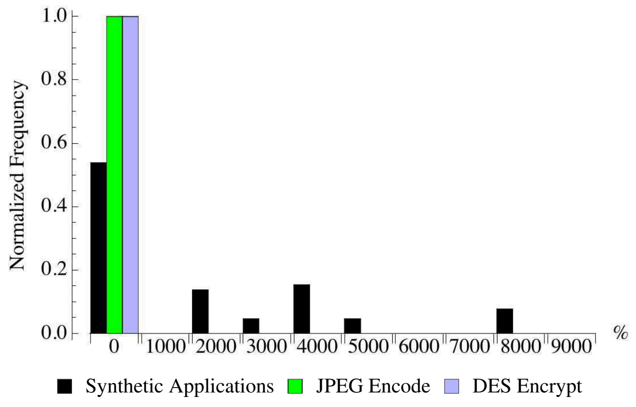

Fig. 8. Percent error for modeled maximum queue occupancy at each buffer vs. measured occupancy. Percent error calculated as $\frac{(\text{modeled occupancy} - \text{observed occupancy})}{\text{observed occupancy}} \times 100$. Histogram bin size is 1000%.

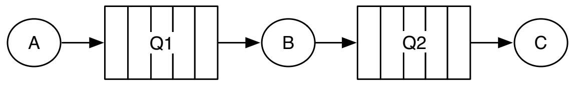

Fig. 9. Tandem queue micro-benchmark used to explore properties of M/M/1 queueing model in a more controlled setting than full applications. Two queues, labeled "Q1" and "Q2" are instrumented for profiling while varying the ρ of servers "A", "B" and "C."

inadequate to reasonably explain the queueing requirements of these applications, and an alternative model is needed.

V. Conclusions

With multicore chips, FPGAs, GPGPUs and other resources to choose from, application designers have a very difficult set of choices when selecting the best execution platform for a given application. A metric that is of particular interest to "big-data" applications is throughput. The analytic model presented in this paper aims to provide an easy to use method for application developers to find the throughput for an application on a particular set of hardware resources while placing a relatively conservative upper bound on queueing capacity necessary. It does a good job of the former, but a poor job of the latter.

The empirical measurements show how the model performs under several conditions and how it can be used to solve for throughputs that are typically within 10% of reality and frequently much closer. In addition to showing where the model performs well, we've shown that for estimating buffering capacity an $M/M/1$ queueing model is often significantly incorrect. A micro-benchmark was constructed to analyze this behavior which points out even further the inability of the model to be effective for maximum queue occupancy estimation purposes.

Overall the results are quite reasonable for a set of models that are explicitly trying to stay simple. The flow model is positioned well to be quite useful. Future work includes further testing the boundaries of where these models succeed and where they fail, exploring alternative models for determining buffering bounds, and exploring the applicability of the models to automated mapping strategies.

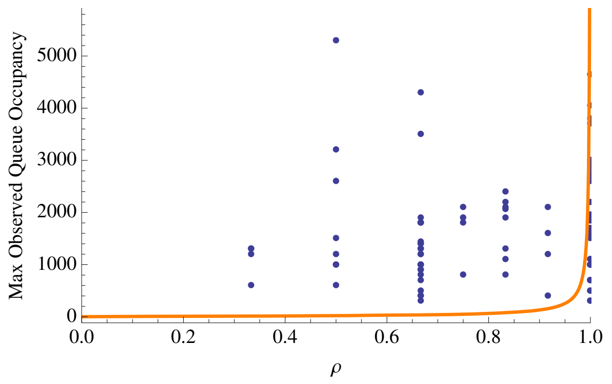

Fig. 10. Measured maximum queue occupancy for the tandem queue micro-benchmark at varying levels levels of ρ. Equation (7), which predicts queue occupancy as a function of ρ, is plotted as a continuous line.

Acknowledgments

This work was supported by NSF grants CNS-0905368 and CNS-0931693 and by Exegy, Inc. R. Chamberlain and Washington University receive income based on a license of technology by the university to Exegy, Inc.

References

[1] J. C. Beard *et al.*, "Simple analytic performance models for streaming data applications deployed on diverse architectures," Dept. of CSE, Washington Univ., Tech. Rep. WUCSE-2013-2, Feb. 2013.

[2] R. Dick *et al.*, "TGFF: Task graphs for free," in *Proc. of 6th Int'l Workshop on Hardware/Software Codesign*, 1998, pp. 97–101.

[3] FIPS PUB 46-3, "Data Encryption Standard (DES)," National Institute of Standards and Technology, 1999.

[4] M. Franklin *et al.*, "Auto-Pipe and the X language: A pipeline design tool and description language," in *Proc. of Int'l Parallel and Distributed Processing Symp.*, Apr. 2006.

[5] D. Goldfarb, Z. Jin, and J. Orlin, "Polynomial-time highest-gain augmenting path algorithms for the generalized circulation problem," *Mathematics of Operations Research*, vol. 22, no. 4, pp. 793–802, 1997.

[6] GraphModeler. http://sbs.wustl.edu/GraphModeler. Accessed June 2013.

[7] International Telegraph and Telephone Consultative Committee *et al.*, "Information technology-digital compression and coding of continuous-tone still images-requirements and guidelines," *Rec. T*, vol. 81, 1992.

[8] W. Jewell, "New methods in mathematical programming–optimal flow through networks with gains," *Operations Research*, vol. 10, no. 4, pp. 476–499, 1962.

[9] J. M. Lancaster *et al.*, "TimeTrial: A low-impact performance profiler for streaming data applications," in *Proc. of Int'l Conf. on Application-specific Systems, Architectures and Processors*, Sep. 2011, pp. 69–76.

[10] B. Pourbabai, J. Blanc, and F. Van der Duyn Schouten, "Optimizing flow rates in a queueing network with side constraints," *European Journal of Operational Research*, vol. 88, no. 3, pp. 586–591, 1996.

[11] P. Schweitzer, "Maximum throughput in finite-capacity open queueing networks with product-form solutions," *Management Science*, vol. 24, no. 2, pp. 217–223, Oct. 1977.