Automated Reliability Classification of Queueing Models for Streaming Computation

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

ABSTRACT

When do you trust a model? More specifically, when can a model be used for a specific application? This question often takes years of experience and specialized knowledge to answer correctly. Once this knowledge is acquired it must be applied to each application. This involves instrumentation, data collection and finally interpretation. We propose the use of a trained Support Vector Machine (SVM) to give an automated system the ability to make an educated guess as to model applicability. We demonstrate a proof-of-concept which trains a SVM to correctly determine if a particular queueing model is suitable for a specific queue within a streaming system. The SVM is demonstrated using a micro-benchmark to simulate a wide variety of queueing conditions.

Categories and Subject Descriptors

D.4.8 [**Performance**]: Measurements, Modeling and Prediction, Queuing Theory

1. INTRODUCTION

Stochastic modeling is essential to the optimization of high performance stream processing systems. The optimization of streaming systems can require the application of several different stochastic models within a single system. Each one must be carefully selected so that assumptions inherent to the model do not make the model diverge significantly from reality. Some streaming systems (such as RaftLib [9]) can spawn tens to hundreds of queues, each potentially with a unique environment and characteristics to model. Approximating an optimal queue size at run-time is clearly not possible manually when microsecond-level decisions are required. This paper outlines a proof-of-concept for an approach when fast modeling decisions are necessary. We will briefly outline the approach to training and using a SVM for deciding when and when not to apply a simple $M/M/1$ queueing model [7] to a particular queue in the context of a streaming computation. Evaluation is given for selection of this queueing model for a variety of conditions simulated via micro-benchmark on a multitude of hardware platforms.

ICPE'15, Jan. 31–Feb. 4, 2015, Austin, Texas, USA.

http://dx.doi.org/10.1145/2668930.2695531.

Stream processing is a compute paradigm that views an application as a set of compute kernels connected via communications links or "streams" (example shown in Figure 1). Streaming languages include StreamIt [12], S-Net [6], and others. Stream processing is increasingly used by multidisciplinary fields with names such as computational-x and x-informatics (e.g., biology, astrophysics) where the focus is on safe and fast parallelization of a specific application [8, 13]. Many of these applications involve real-time or latency sensitive big data processing necessitating usage of many parallel kernels on several compute cores.

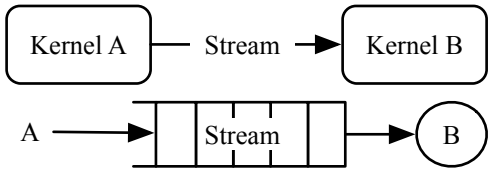

Figure 1: The top image is an example of a simple streaming application with two compute kernels (labeled A & B). Each compute kernel could be assigned to any number of compute resources (e.g., processor core, graphics engine). The communications stream connecting A & B could be allocated to a variety of resources depending on the nature of A & B (e.g., heap, shared memory or network interface). We are interested in the queueing behavior that results from the communications between compute kernels A & B. The bottom image is the resulting queue with arrival process A (emanating from compute kernel A) and server B. For more complex systems this becomes a queueing network.

Optimizing or reducing the communication overhead within a streaming application is often a non-trivial task, it is however central to making big data and stream processing successful. When viewing each compute kernel as a "blackbox," a major tuning knob at the application's disposal is the buffer (queue) size between kernels. A classic way to size a buffer is to run each compute kernel in isolation with the expected workload, derive the service rate and distribution, then use this data to select a queueing model to inform the correct buffer size. Modern streaming systems such as RaftLib support online queue optimization. Param-

eters such as service process distribution are difficult to determine online. Clearly, the classic method is inadequate for this circumstance. Another approach could use branch and bound searching, but it can consume significant time (much spent reallocating buffers). A more efficient online approach is to make an educated guess as to which model to use for each queue and solve for the buffer size. If the service rate of the kernel is known, we contend (and show evidence for) that a sufficiently trained SVM can select the correct model quickly, avoiding bounding search and possibly negating the need for process distribution determination.

A SVM is a method to separate a multi-dimensional set of data into two classes by a separating hyperplane. Theoretical details are covered by relevant texts on the subject [14] A SVM labels an observation (represented by a set of attributes to identify it) with a learned class label based on the solution to Equation (1) [5] (note: dual form given, $\mathbf{e}$ is a vector of all ones of length l, Q is an $l \times l$ matrix defined by $Q_{i,j} \leftarrow y_i y_j K(x_i, x_j)$ and K is a kernel function, specific symbolic names match those of [3]). A common parameter selected to optimize the performance of the SVM is the penalty parameter for the error term C (value discussed in Section 3.1).

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T \mathbf{Q}\boldsymbol{\alpha} - \mathbf{e}^T\boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \leq \boldsymbol{\alpha}_i \leq C, i = 1, \ldots, l, \end{aligned} \tag{1}$$

$$K(x, y) = e^{-\gamma||x-y||^2}, y > 0. \tag{2}$$

Utilizing 76 features (shown in Figure 2) easily extracted from a system, we show that a machine learning process can identify where a model can and cannot be used. Each one of the input features can be determined *a priori*. We also assume that the mean service rate (i.e., rate of data consumption by the kernel) can be determined either statically or online, while the application is executing. In order to map all 76 attributes into features we use a Radial Basis Function (RBF, [10], Equation (2)) represented as K. The parameter, γ is commonly optimized separately in order to maximize the performance of the SVM and kernel combination (value of γ also discussed in Section 3.1).

Figure 2: Word cloud depicting features used for machine learning process with the font size representing the significance of the feature as determined by [4].

Kernel learning methods such as SVM have been utilized in diverse pattern recognition fields such as handwriting recognition and protein classification [2]. This work does not seek to add new techniques to SVM or machine learning theory, rather we are focused on expanding the application of these methods to judging the applicability of stochastic performance models to real-world scenarios in streaming computation. Results are shown that demonstrate that our SVM approach appears to be applicable with respect to multiple operating systems and hardware types, at least for the micro-benchmark conditions.

2. ASSUMPTIONS & PITFALLS

The stochastic mean queue occupancy models our SVM will be trained for are steady state models. Applications whose performance is not well characterized by mean behavior are not good candidates without modifications of our approach. First, we assume that the applications under test have executed long enough so that the queue occupancies observed are at steady state. Second, our investigation will be constrained to applications for which the computational behavior is independent of the input set or does not vary dramatically as a function of the input. In future work we plan to modify our approach that will perhaps enable characterization of some non-steady state behavior.

Architectural features of a specific platform such as cache size are used as features for this work. As such we assume that we can find them either via literature search or directly by querying the hardware. Platforms where this information is unknown are avoided, and there is a surfeit of platforms where this information is knowable.

Implicit within most stochastic queueing models (save for the circumstance of a deterministic queue) is the strict relegation of server utilization ρ to be less than one for a finite queue occupancy. It is expected that the SVM should be able to find this relationship based upon the training process. It is shown in Section 3.2 that this is indeed the case. We also assume that the SVM is not explicitly told what the actual service time distributions are of the compute kernels modulating data arrival and service.

Changes in service time distribution can drastically effect the queue occupancy characteristics of an individual queue. Traditionally a change in distribution would necessitate a quantification of that distribution before moving forward with a different type of model. We hypothesize that by training the SVM with a variety of distributions that the SVM can identify regions where the models under consideration can be used and where they cannot. This is not to say that the SVM as trained will be successful in all cases. We'll examine some edge cases in the results. It is possible that the SVM as trained might not generalize to distributions that did not exist in the training data.

3. EVALUATION

In order to train the model with as much data as possible and evaluate the method with some known parameters, a set of micro-benchmarks (see topology in Figure 1) with synthetically generated workloads is used. A synthetic workload for each compute kernel is composed of a simple busy-wait loop whose looping is dependent on a random number generator (either exponential, Gaussian, deterministic or a mixture of multiple distributions). Upon completion of

work, a nominal "job" (data element) is emitted from "A" to "B" via the connecting stream. Each job is the same size for each execution of the micro-benchmark (8-bytes). The micro-benchmark applications have been authored in C/C++ using the RaftLib framework and are compiled with `g++` using the `-O1` optimization flag for their respective platforms. These applications are executed on a variety of x86 commodity hardware with a range of Linux and OS X operating systems.

3.1 Methodology

Our SVM will be used to classify the micro-benchmark queues with "use" or "don't use" for the $M/M/1$ analytic queueing model. Before the SVM can be trained as to which set of attributes to assign to a class, a label must be provided. The labeling function is described by Algorithm 1. For our labeling function $l \leftarrow 5$. A percentage based function for l could also have been used. Empirically measured mean queue occupancy is used for labeling purposes. Observation data is divided into two distinct sets via a uniform random process. Approximately 20% of the data is used for training the SVM, the remainder is used for testing. The set training set has the following specifications: server utilization ranges from close to zero to greater than one and distributions vary widely (a randomized mix of Gaussian, deterministic, and the model's expected exponential distribution as well as some mixture distributions).

Algorithm 1 Class assignment algorithm

```
if |observed occupancy − predicted occupancy| ≤ l then
    class ← use
else
    class ← don't use
end if
```

The micro-benchmark data (and attributes) are linearly scaled in the range $[-1000, 1000]$ (see [11]). This tends to cause a slight loss of information, however it does prevent extreme values from biasing the training process. It also has the added benefit of reducing the precision necessary for the representation. Once all the data are scaled, there are a few SVM specific parameters that must be optimized in order to maximize classification performance (γ and C). We use a branch and bound search for the best parameters for both the RBF Kernel ($\gamma \leftarrow 4$) and for the penalty parameter ($C \leftarrow 32768$). The branch and bound search is performed by training and cross-validating the SVM using various values of γ and C for the 20% set of training data discussed above. The SVM framework is sourced from LIBSVM [3].

3.2 Results

The SVM classifies a set of platform and compute kernel specific attributes with one of two labels, "use" or "don't use" with respect to a $M/M/1$ stochastic queueing model. The SVM is trained on data labeled with these two binary categories via Algorithm 1. To evaluate how well the SVM classifies a queueing system, we'll compare the known (but not to the SVM) class labels compared to those predicted by the SVM. If the queueing model is usable and the predicted class is "use" then we have a true positive (TP). Consequently the rest of the error types true negative (TN), false positive (FP) and false negative (FN) follow the obvious corollaries.

As enumerated in Table 1, the SVM correctly predicts (TP or TN) 88.1% of the test instances for the $M/M/1$ model. Overall these results are quite good compared to manual selection [1]. Not only do these results improve the mean queue occupancy predictions, they are far faster than manually interpreting the parameters of the queueing system to select a model. The average per example classification time is in the microsecond range. Our results suggest that this process can be effective for online model selection for the $M/M/1$ model.

Table 1: Overall classification predictions for micro-benchmark data.

Model	# obs.	TP	TN	FP	FN
$M/M/1$	39392	66.60%	21.60%	11.70%	0.20%

Server utilization (ρ) informs a classic, simple test to determine if a mean queue length model is suitable. At high ρ it is likely that the $M/M/1$ models will diverge widely from reality. It is assumed that the SVM should be able to discern this intuition from its training without being given the logic via human intervention (a key point in training the SVM to take the place of human intervention). Figure 3 shows a box and whisker plot for the error types separated by ρ. As expected the middle ρ ranges offer the most true positive results. Also expected is the correlation between high ρ and true negatives. Slightly unexpected was the relationship between ρ and false positives.

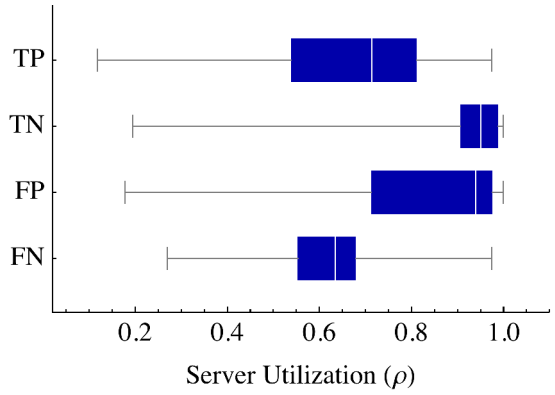

Figure 3: Summary of true positive (TP), true negative (TN), false positive (FP), false negative (FN) classifications for the $M/M/1$ model of the micro-benchmark data by server utilization ρ.

Directly addressing the performance and confidence of the SVM is the probability of class assignment. Given the high numbers of TP and TN it would be useful to know how confident the SVM is in placing each of these feature sets into a category. Probability estimates are not directly provided by the SVM, however there are a variety of methods which can generate a probability of class assignment [15]. We'll use the median class assignment probability for each error category as it is a bit more robust to outliers than the mean. This results in the following median probabilities: TP = 99.5%, TN = 99.9%, FP = 62.4% and FN = 99.8%. The last number must be taken with caution given that there are only 79 observations in the FN category. For the FP

it is good to see that these were low probability classifications on average, perhaps with more training and refinement these might be reduced. Calculating probabilities is expensive relative to simply training the SVM and using it. It could however lead to a way to reduce the number of false positives. Placing a limit of $p = .65$ for positive classification reduces false positives by an additional 95% for the micro-benchmark data. Post processing based on probability has the benefit of moving this method from slightly conservative to very conservative if high precision is required, albeit at a slight increase in computational cost.

In Section 2 we noted that one potential pitfall of this method is the training process. What would happen if the model is trained with too few distributions? To test this hypothesis a set of the training data from a single distribution (the exponential) is used and divided into two sets, 20% for training and 80% for testing. The exponential only training data is used to train a SVM which is then tested on the testing exponential only data (results shown in Table 2). Second, the SVM trained with only exponential micro-benchmark data is used to classify data from distributions other than exponential (note: no data used to train is in the test set). These results are shown in Table 2. Two trends are apparent: training with a single distribution increases the accuracy when attempting to classify data with that distribution and lack of training diversity increases the number of false positives for the distributions not seen during the training process. Unlike the false positives seen earlier, these are high confidence predictions meaning that post processing for probability will not significantly improve predictions. Training with as many distributions as possible is essential to improving the generalizability of our method.

Table 2: % for SVM predictions with SVM trained only with servers having an exponential distribution.

Dist.	# obs.	Model	TP	TN	FP	FN
exp.	3249	$M/M/1$	53.0	31.2	15.7	.092
many	6297	$M/M/1$	55.8	0.0	44.2	0.0

4. CONCLUSIONS & FUTURE WORK

We have shown an approach for using a SVM to classify a stochastic queuing model's reliability in the context of a streaming system (varying hardware platform, application, operating system and environment). Across multiple hardware types, operating systems and micro-benchmark applications it has been shown to produce fairly good reliability estimates for the $M/M/1$ stochastic queueing model.

This work does not assume the availability of knowledge of the actual distribution of each compute kernel. Manually determining these distributions and retraining the SVM improves the classification rate to 96.6%. One obvious path for future work is faster and lower overhead process distribution estimation. As a work in progress paper, we did not explore the limits of generalization or the impact of online model selection to the efficiency (and performance) of the RaftLib framework. In conclusion we have shown a proof-of-concept for automated stochastic model selection using a SVM. We have shown that it can be done, and our limited testing suggests it works relatively well.

5. ACKNOWLEDGMENTS

This work was supported by Exegy, Inc., and VelociData, Inc. Washington University in St. Louis and R. Chamberlain receive income based on a license of technology by the university to Exegy, Inc., and VelociData, Inc.

6. REFERENCES

[1] J. C. Beard and R. D. Chamberlain. Analysis of a simple approach to modeling performance for streaming data applications. In *Proc. of IEEE Int'l Symp. on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 345–349, Aug. 2013.

[2] C. J. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[3] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Trans. on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[4] Y.-W. Chen and C.-J. Lin. Combining SVMs with various feature selection strategies. In *Feature Extraction*, pages 315–324. Springer, 2006.

[5] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[6] C. Grelck, S.-B. Scholz, and A. Shafarenko. S-Net: A typed stream processing language. In *Proc. of 18th Int'l Symp. on Implementation and Application of Functional Languages*, pages 81–97, 2006.

[7] L. Kleinrock. *Queueing Systems. Volume 1: Theory.* Wiley-Interscience, New York, NY, 1975.

[8] W. Liu, B. Schmidt, G. Voss, and W. Muller-Wittig. Streaming algorithms for biological sequence alignment on GPUs. *IEEE Trans. on Parallel and Distributed Systems*, 18(9):1270–1281, Sept 2007.

[9] RaftLib. `http://www.raftlib.io`. Accessed November 2014.

[10] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, 2002.

[11] D. M. Tax and R. P. Duin. Support vector data description. *Machine learning*, 54(1):45–66, 2004.

[12] W. Thies, M. Karczmarek, and S. Amarasinghe. StreamIt: A language for streaming applications. In R. Horspool, editor, *Proc. of Int'l Conf. on Compiler Construction*, volume 2304 of *Lecture Notes in Computer Science*, pages 49–84. 2002.

[13] E. J. Tyson, J. Buckley, M. A. Franklin, and R. D. Chamberlain. Acceleration of atmospheric Cherenkov telescope signal processing to real-time speed with the Auto-Pipe design system. *Nuclear Inst. and Methods in Physics Research A*, 585(2):474–479, Oct. 2008.

[14] V. N. Vapnik and V. Vapnik. *Statistical Learning Theory*, volume 2. John Wiley & Sons, New York, NY, 1998.

[15] T.-F. Wu, C.-J. Lin, and R. C. Weng. Probability estimates for multi-class classification by pairwise coupling. *The Journal of Machine Learning Research*, 5:975–1005, 2004.