Machine Learning for Performance Prediction of Data Distribution Service (DDS)

Kaleem Peeroo
City, University of London
London, UK
Kaleem.Peeroo@city.ac.uk

Peter Popov
City, University of London
London, UK
P.T.Popov@city.ac.uk

Vladimir Stankovic
City, University of London
London, UK
Vladimir.Stankovic.1@city.ac.uk

Tillman Weyde
City, University of London
London, UK
T.E.Weyde@city.ac.uk

***Abstract*—Networking middleware following the Data Distribution Service (DDS) specification is used in real-time mission-critical systems such as autonomous vehicles, energy management systems, and air traffic control. DDS follows the publish-subscribe communication pattern and offers a set of Quality of Service (QoS) parameters, allowing the users to align the data communication to the needs of the application.**

Configuring DDS to achieve the required performance is a difficult task, given the large space of QoS parameter values. Experimental evaluation of performance levels with a real DDS system for different QoS configurations can be complex and require substantial time and resources.

We propose the use of Machine Learning (ML) models to predict the performance metric distribution of DDS under different configurations. This is done by using performance measurements of some configurations to train an ML model. The trained model can then be used to predict the performance distribution of DDS under other system configurations. Since the prediction is computationally inexpensive, we can predict the performance of many different configurations to find a suitable one for given requirements. To the best of our knowledge, this is the first time this approach has been applied to DDS performance evaluation.

We used *random forests* (RF) as an ML method and *linear regression* (LR) as a baseline. We selected thirteen performance metrics, and for each, we trained an RF model and tuned its hyperparameters. We tested the final models on system configurations unseen during training, both for parameter values within the training range (interpolation) and outside the training range (extrapolation).

The RF models show better predictive accuracy than the LR baseline. This paper focuses on the models for *throughput* and *latency* - the two well-established performance metrics. The models demonstrate coefficients of determination greater than 0.9 and 0.8, respectively, for different unseen system configurations in interpolation, but work less well in extrapolation cases.

We conclude that the proposed ML models offer a way of predicting the performance distribution of a range of configurations when interpolation is used. Since model prediction is computationally much cheaper than relying on experimentation, it is a useful tool to guide DDS system parametrisation and design.

***Index Terms*—Data Distribution Service, Machine Learning, Linear Regression, Random Forests, Performance Prediction**

I. Introduction

A. Data Distribution Service and its Configuration Parameters

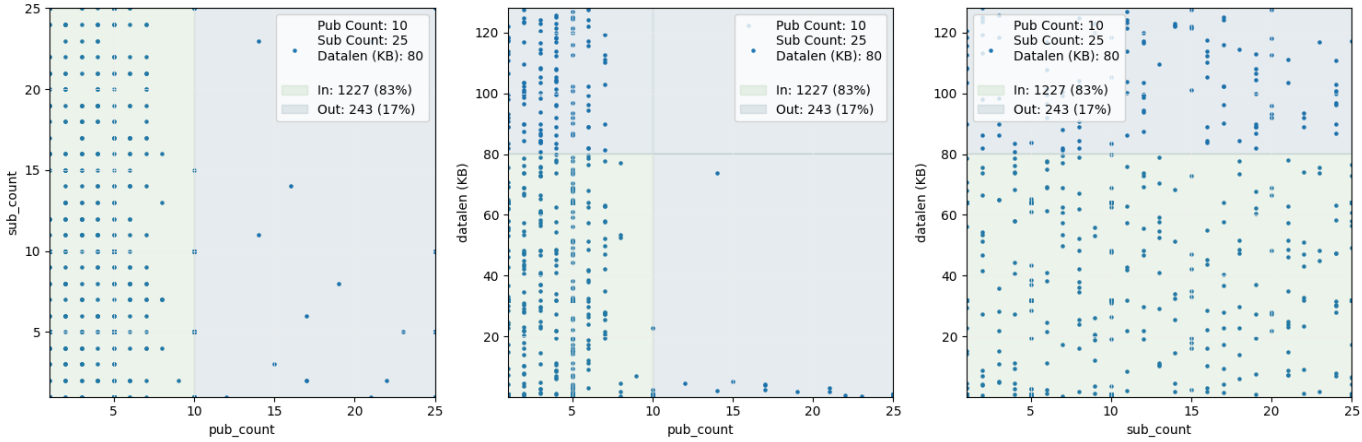
The Data Distribution Service (DDS) [1] specification addresses networking middleware employed in real-time, mission-critical systems such as autonomous vehicles and air traffic control. DDS follows the publish-subscribe paradigm, where publishers distribute data and subscribers receive it, with data organised into topics. The DDS specification is maintained by the Object Management Group (OMG) and delineates 22 Quality-of-Service (QoS) parameters for communication. There are also non-QoS system parameters, such as message data length, the number of publishers/subscribers, and multicast usage.

Performance measures like latency or throughput are essential for determining if real-time requirements are met for DDS implementations. Different combinations of configuration parameters, including both QoS and non-QoS, can significantly impact performance. However, the selection of an ideal setup is challenging due to the millions of possible configurations. Conducting experiments for each possible configuration is prohibitive in terms of time and computational cost. This paper presents a solution to efficiently evaluate DDS performance amidst this vast array of combinations. We propose using a small set of measurements to train a Machine Learning (ML) model, allowing for cost-effective exploration of the DDS configuration design space. This approach enables the identification of configurations that are likely good enough or even best in some sense. The actual performance of the chosen configurations can then be confirmed with a small set of experiments.

We extended Real-Time Innovation's (RTI) Perftest [2], a widely used performance testing benchmark, to evaluate DDS performance. While performance could be described using just average and variance, our approach involves characterising distributions using multiple statistics, including mean, standard deviation, lower/upper quartiles, median, all deciles, and specific percentiles that capture the head and tail of the distribution. This analysis comprehensively captures the behaviour of the system, particularly worst-case performance, such as the tail of the latency distribution, which is essential for meeting real-time requirements. Although further analysis may be necessary for determining upper bounds in worst-case scenarios, our method can identify relevant configurations for in-depth study. Our work is distinctive in accounting for the entire distribution, unlike other studies, empirical or otherwise, that typically focus only on predicting the expected value and variance.

Our study concentrates on two types of predictions: *interpolation*, predicting inside the range of training values, and

Fig. 1. Interpolation vs Extrapolation within the gridspace



extrapolation, predicting for configurations outside the training value range. This paper presents selected results for predicting DDS performance, with comprehensive details available in the accompanying technical report [3].

B. Perftest

As mentioned above, we employ Perftest to assess DDS performance. While Perftest is beneficial, it is not ideal due to its benchmark nature; evaluating DDS performance using a real-life application would provide a more meaningful assessment of the middleware. However, our focus is to demonstrate our method for obtaining performance estimates for various configurations, which applies to any workload, with Perftest used here merely for demonstration purposes. Notably, Perftest allows manipulation of only two of the 22 QoS parameters: durability and reliability.

C. Machine Learning

ML methods aim to learn complex patterns for making accurate predictions and are used here to predict the performance of DDS configurations. We employ *random forests* and compare it with *linear regression* as a baseline. Linear regression assumes a linear relationship between variables, which can be enhanced to fit the data better using various transformation functions like logarithmic or exponential. In contrast, random forests, an ensemble of decision trees, can directly model non-linear patterns. All ML models are trained on sample data aiming to minimise an error metric and evaluated on unseen test data. We use two prediction metrics: RMSE (Root Mean Square Error) and R^2 [4]. RMSE, the square root of the average squared difference between actual and predicted values, is more intuitive as it is in the same units as the target variable; a lower value indicates better predictions. R^2, indicating the proportion of variance in the target variable explained by the model, typically ranges from 0 to 1; a higher number signifies better fit and allows for comparing the goodness of fit of models across different scenarios.

The remainder of this paper is structured as follows: Section II looks at related works, Section III explains our approach, including how the performance tests were carried out and how the data was processed and used to train the ML models. Section IV provides an overview of the prediction results. Section V summarises the findings and possible future work.

II. Related Work

To the best of our knowledge, ML models have not been applied to predict DDS performance. We discovered, however, potentially relevant works under a different umbrella term: *Performance Prediction of Configurable Systems using ML.*

Much work is done on the performance prediction of systems, which have configurable parameters whose values affect the system's performance. Most of these works do rely on ML techniques, but only predict a single statistic characterising the chosen performance metric (e.g. the mean value).

In [5], a comparison-based model based on random forests is used to find the optimal configuration that maximises the throughput. A combination of ML techniques and sampling strategies to predict the performance of configurable systems is given in [6] and [7]. These sampling strategies determine which data to train on regarding the configuration locations within an n-dimensional space determined by the values of the configuration parameters. A performance prediction of multiple configurations using a deep feed-forward neural network is given in [8]. This approach results in just a single performance statistic.

The ML techniques used in our work originate from [9], a survey paper looking at works that involve a combination of ML techniques and sampling strategies for predicting the performance of configurable systems. The survey paper recommends using 3 main ML techniques: random forests, multiple linear regression, and classification and regression trees.

III. Method

A. Configuration Space Exploration

We conducted 1470 tests with predefined and random configurations where the publisher/subscriber count ranged from 1 to 25 and the data length from 100 bytes to 64KB. The results were divided for interpolation and extrapolation purposes,

TABLE I
PREDICTION PERFORMANCE PER DDS METRIC FOR RANDOM FORESTS AND LINEAR REGRESSION (INT = INTERPOLATION, EXT = EXTRAPOLATION).

Metric	LR Int R2	LR Ext R2	RF Int R2	RF Ext R2	LR Int RMSE	LR Ext RMSE	RF Int RMSE	RF Ext RMSE	Metric Avg.
Latency (us) Mean	0.26	0.03	0.83	0.27	1,228,465	8,106,697	583,673	6,285,378	1,080,019
Latency (us) 99th Percentile	0.32	0.04	0.83	0.07	3,010,601	19,932,462	1,498,101	19,550,963	1,080,019
Avg. Throughput (mbps) Mean	0.57	0.16	0.99	0.94	27.22	32	9	4	39
Avg. Throughput (mbps) 1st Percentile	0.52	0.38	0.93	0.93	29	28	10	11	39

between inside and outside a parameter value boundary. The split is 83%-17% between inside and outside the boundary, as shown in Figure 1. Training for both interpolation and extrapolation occurred within the boundary, whereas testing for interpolation and extrapolation used data from inside and outside the boundary, respectively.

B. Experimental Setup (Testbed)

Our experiment used 3 Raspberry Pis: a main controller (Raspberry Pi 4 Model B, 2018) and two endpoints (Raspberry Pi 3 Model B V1.2, 2015) communicating with each other. Tests were conducted using 'Autoperf', our bespoke tool, derived from Perftest, that varies parameter values, running different test configurations in sequence with reboots between tests to avoid interactions between configurations. Each test configuration ran for 10 minutes, measuring latency every 100 messages to balance measurement frequency and performance impact. Latency was measured on publishers by timestamping messages before sending and upon return, avoiding clock skew. Throughput was measured by each subscriber every second.

C. Dataset Preprocessing

Latency measurements in our tests were taken only from the first publisher (a Perftest characteristic). On each subscriber, we measured throughput, sample rate, lost and received sample counts, and percentages every second. Our dataset includes both configuration details and statistics for distribution recreation per performance metric. Thirteen performance metrics were recorded: latency, average and total throughput, sample rates, received and lost sample counts and percentages, averaged or totalled over all subscribers. Transient periods observed in most tests led to the removal of the initial 20% of raw observations.

D. Model Training & Evaluation

Linear regression and random forests models were trained individually for each DDS metric (such as latency or average throughput). The models receive 6 input parameters (data length, publisher count, subscriber count, durability, reliability, and multicast usage) and predicts for each performance metric 20 distribution statistics: mean, standard deviation, median, min, max, and the following percentiles: 1, 2, 5, 10, 20, 25, 30, 40, 60, 70, 75, 80, 90, 95, and 99. We tested the following input data standardisation methods: none, z-score [10], min-max [11], and robust scaler [12]. For both models, we tested the following transformations of the label data: log, $log(1+x)$, and square root. We hypothesised that this would mitigate the impact of very high latency values on overall predictions. These transformations were inverted before the calculation of prediction metrics. Our focus here is on R^2 and RMSE, though MSE, MAE, MAPE, and MedAE, and explained variance were also recorded [3].

IV. RESULTS

A. Overview

Different models were trained for each combination of standardisation and transformation for each DDS metric. This section discusses the results of interpolation and extrapolation for both linear regression and random forests models. Table I presents the prediction RMSE and R^2 for two relevant statistics (mean and first/last percentile) for throughput and latency. The metric average column has been included in the table to provide context for the RMSE values. The full set of results is available in the technical report [3].

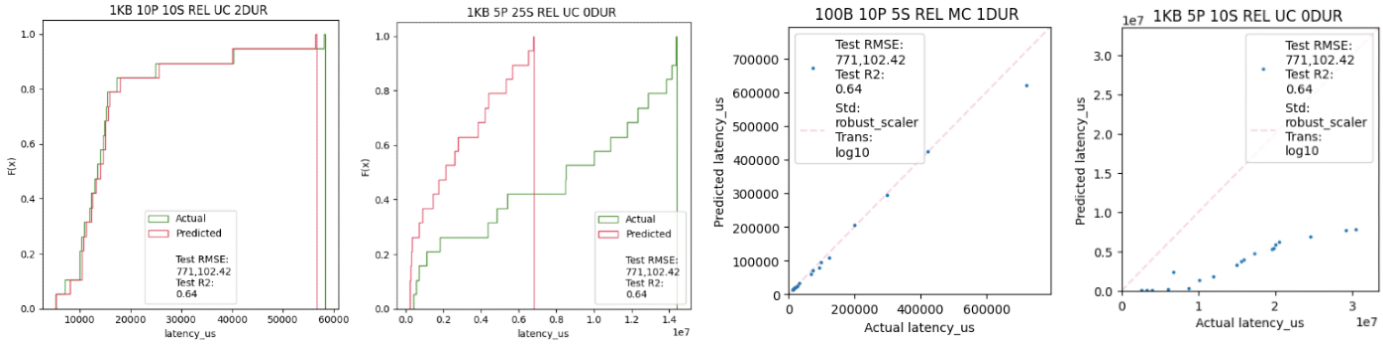
B. Linear Regression vs Random Forests

The results show that random forest models consistently outperform linear regression. Linear regression shows worse interpolation predictions than random forests and notably poor extrapolation, especially for latency (R^2 values are 0.03 and 0.04). On the other hand, random forests show consistently high R^2 for interpolation, while extrapolation of latency is less successful. The RMSE values for all throughput models are lower than their respective metric averages. In contrast, the prediction of latency is less reliable, in particular for extrapolation. For the 99th percentile, this effect is even more pronounced. To avoid high errors in latency prediction, it will be necessary to train models on the full range of relevant parameter values.

C. Latency Model Analysis

In this section, we focus on the latency metric, which is essential for assessing if the DDS application meets real-time requirements. Figure 2 shows four tested configurations, two good and two bad, with their results depicted in empirical cumulative distribution functions (CDFs, on the left) and scatter plots (on the right) comparing actual and predicted values. The model shown in plot 1, has an average R^2 of 0.64 (over all R^2 values for the twenty statistic) and RMSE of 770,000, closely matches actual and predicted values, with only a 3% difference in the highest latency value. The second plot shows a systematic left shift of the latency distribution. The third plot is a scatter plot showing a majority of points aligning well, albeit with slightly optimistic predictions for the highest value.

Fig. 2. Examples of actual vs predicted cumulative distribution functions for good (first) and bad (second) cases and examples of actual vs predicted scatter plots for good (third) and bad (fourth) cases of the best-performing interpolation model for predicting latency distributions. 1KB = 1 kilobyte, 10P = 10 publishers, 10S = 10 subscribers, REL = Reliable, UC = Unicast, MC = Multicast, 1DUR = Durability level 1.



In contrast, plot 4 shows consistently lower predictions than actual values. Overall interpolation performance is better than extrapolation. However, specific cases can deviate, as in plot 1 (on the border for the publisher number) and plot 3 (similar location) show good predictions, while plot 4, with parameters well inside the boundary of the training data, shows poor prediction. Further error analysis and exploration of different transformation techniques are necessary to understand these varied results and improve prediction accuracy, especially for extrapolation. However, extrapolation can be avoided in practice by using training examples that include the range boundaries for all parameters. This has not been tested here but is expected to reduce the high latency prediction error for some configurations.

V. Conclusion

This research introduces, to our knowledge for the first time, a method for predicting DDS performance based on QoS and non-QoS parameters using machine learining. The models predict twenty statistics to characterise the the distribution of a performance metric according to six configuration parameters.

Our study compares linear regression and random forests for interpolation and extrapolation, finding as expected that prediction in interpolation scenarios generally outperforms extrapolation, with a few exceptions. Random forests are more effective than linear regression, especially for extrapolation. The throughput predictions by random forests are generally effective, with high R^2 and low RMSE values relative to the metric average. Latency, a crucial metric for real-time constraints, is harder to predict, especially for the higher end of the distribution. While the latency model provides accurate predictions in the interpolation case, further error analysis is needed to enhance its performance for extrapolation. In conclusion, our findings demonstrate that ML models offer a cost-effective means of estimating DDS performance in untested conditions.

Our experiments use a particular performance benchmark, RTI Perftest. However, this method can be applied to different applications by collecting specified data and training ML models. We acknowledge, however, that results may vary in different scenarios. Extensions of this work include training *neural networks* and experimenting with *different sampling strategies* as mentioned in related works. Also, we aim to investigate how the location of configurations within the gridspace impacts the accuracy of predictions.

References

[1] "Data Distribution Service (DDS)."

[2] "RTI Perftest." [Online]. Available: https://github.com/rticommunity/rtiperftest

[3] K. Peeroo, P. Popov, V. Stankovic, and T. Weyde, "Machine learning for performance prediction of data distribution service," Tech. Rep. 1. [Online]. Available: https://openaccess.city.ac.uk/id/eprint/31554/

[4] *Coefficient of Determination*. New York, NY: Springer New York, 2008, pp. 88–91. [Online]. Available: https://doi.org/10.1007/978-0-387-32833-1_62

[5] L. Bao, X. Liu, Xin Liu, Xin Liu, X. Ziheng, and B. Fang, "AutoConfig: automatic configuration tuning for distributed message systems," pp. 29–40, Sep. 2018, mAG ID: 2888705583.

[6] J. Guo, D. Yang, N. Siegmund, S. Apel, A. Sarkar, P. Valov, K. Czarnecki, A. Wasowski, Huiqun Yu, Huiqun Yu, and H. Yu, "Data-efficient performance learning for configurable systems," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1826–1867, Jun. 2018, mAG ID: 2769879317.

[7] N. Siegmund, A. Grebhahn, S. Apel, and C. Kästner, "Performance-influence models for highly configurable systems," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. Bergamo Italy: ACM, Aug. 2015, pp. 284–294. [Online]. Available: https://dl.acm.org/doi/10.1145/2786805.2786845

[8] H. Ha, H. Ha, and H. Zhang, "DeepPerf: performance prediction for configurable software with deep sparse neural network," pp. 1095–1106, May 2019, mAG ID: 2954141573.

[9] A. Grebhahn, N. Siegmund, and S. Apel, "Predicting Performance of Software Configurations: There is no Silver Bullet." *arXiv: Software Engineering*, Nov. 2019, mAG ID: 2991131477.

[10] "sklearn.preprocessing.StandardScaler." [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[11] "sklearn.preprocessing.MinMaxScaler." [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[12] "sklearn.preprocessing.RobustScaler." [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.preprocessing.RobustScaler.html