A Framework for Scalable Data Analysis and Model Aggregation for Public Bus Systems

Mayuri A. Morais[1], **Raphael Y. de Camargo**[1]

[1]Centro de Matemática, Computação e Cognição – Universidade Federal do ABC (UFABC)
Rua Arcturus, 03 – 09606-070 – São Bernardo do Campo – SP – Brazil

{mayuri.morais,raphael.camargo}@ufabc.edu.br

Abstract. *Urban mobility through quality public transportation is one of the major challenges for the consolidation of smart cities. Researchers developed different approaches for improving bus system reliability and information quality, including travel time prediction algorithms, network state evaluations, and bus bunching prevention strategies. The information provided by these approaches are complementary and could be aggregated for better predictions. In this work, we propose the architecture and a present a prototype implementation of a framework that enables the integration of several approaches, which we call models, into scalable and efficient composite models. For instance, travel time prediction models can use estimators of bus position, network state, and bus headways to deliver more accurate and reliable predictions. We evaluate the scalability of the framework, the CPU usage of the framework components, and the predictions of the travel time models. We show that real-time predictions using this framework can be feasible in large metropolitan areas, such as São Paulo city.*

1. Introduction

Providing efficient urban mobility is one of the major challenges facing large metropolitan centers today. An efficient way to reduce congestion is with the provision of quality public transport systems. Rail transport systems using trains and subways are very efficient, cover long distances, can connect cities close to capitals, are fast and generally suffer few delays and loss of time. However, the investments required for the creation or expansion of a railway line are high, with long maturation and grace periods (until the first line operation occurs). Thus, although rail transport is very efficient, its cost of implementation and extension is very high, so that most of the demand for travel in public transport is served by urban bus systems, which are more straightforward to implement and with lower cost.

Public transport systems using buses in Metropolis, such as São Paulo, are complex systems that continuously interact with the dynamics of the city [Cascetta 2009]. Buses are delayed due to congestion and overcrowding, as well as having their flow interrupted by traffic lights. Understanding the behavior of this system in different contexts, such as weekdays, hours of the day and holidays, is vital for better

This research is part of the INCT of the Future Internet for Smart Cities funded by CNPq proc. 465446/2014-0, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001, FAPESP proc. 14/50937-1, and FAPESP proc. 15/24485-9.

planning of these systems. Dealing with this complexity is difficult and may require sophisticated models that encompass different aspects of the bus system, such as bus network state [Zhang et al. 2016], bus position estimation [Adachi et al. 2015], travel time predictions [Mori et al. 2015, Choudhary et al. 2016] and bus headway evolution [Yu et al. 2017]. Existing studies evaluate these aspects separately, despite the clear interactions between them, such as the influence of headway evolution on travel time predictions. There are only a few efforts in this direction of combining aspects of bus system analysis [Mazloumi et al. 2011].

One way to improve this interaction is by providing a framework for bus data collection, analysis, and visualization, which contains a set of models, each contemplating a specific aspect of the bus system. Such a system contain some lower level models, such as the graph representing the bus network and the estimation of the network link states (congestion level), bus positions and bus headways. On top of these, higher level predictions models, such as for travel time, bus headway evolution, and network link state evolution, can be built. Moreover, the framework should allow the deployment of different types of models for each aspect, with different computational complexities, which can be selected based on available computational resources and processing time requirements.

In this work we propose the architecture and a prototype implementation of this framework, containing a network model of one hundred bus routes from São Paulo city, an estimator for bus positions, and three travel time prediction models, which use historical and real-time data on bus positions. We evaluate the scalability of the framework with three workstations, the CPU usage of the framework components, and the predictions of the travel time models. We show that real-time predictions using this framework can be feasible even for large metropolitan areas, such as São Paulo city.

2. Related Work

In the literature we find two distinct types of predictions for bus travel time: predictions of total travel time and predictions of time / arrival time at a specific point, where efforts are more concentrated in this second type of prediction. Proposals for the prediction of total bus travel time were developed using Kalman filter [Chen et al. 2004], non-parametric regression [Chang et al. 2010] and SVM [Yu et al. 2010]. One of the first methodologies for predicting bus arrival time was proposed by [Lin and Zeng 1999] using regression and real-time GPS data. After this study, several other studies developed methods and models for the prediction of arrival time, using neural networks [Chien et al. 2002, Jeong and Rilett 2004], Kalman filter [Chen et al. 2004, Shalaby and Farhan 2004], *Support Vector Machine* (SVM) [Bin et al. 2006, Yu et al. 2008], *crowdsourcing* [Zhou et al. 2012] and *Finite State Machine* [Zuo and Wang 2013, Fu et al. 2014].

Some works propose combining different methods to achieve better predictions. [Mazloumi et al. 2011] proposed a framework using data from different sources to predict mean travel time and its variability with two neural networks, using these values to compose a Gaussian distribution. [Zhenliang et al. 2011] combined SVM and H-Filter to combine historical and real-time data to predict bus arrival time. [Kumar et al. 2017a] used the k-NN classification algorithm to identify typical patterns in historical data. The results of this algorithm were used in a Kalman filter to predict travel times. Similarly,

[Kumar et al. 2017b] used k-NN and Kalman filter for travel time predictions, but now applying an exponential smoothing factor to the model used in the Kalman filter. The smoothing factor parameters were dynamically estimated and updated using recent measures. Still extending the latter work, [Kumar et al. 2017c] developed a model using equations of velocity conservation in terms of the flow theory, applying a spatiotemporal discretization to these equations. The discretized model is applied in the Kalman filter to predict the speed in a connection.

All of the above works use a single bus line to carry out predictions. [Yu et al. 2011] proposed a model considering eight lines that go through a common link to make the prediction using SVM and data only in real time. [Yin et al. 2017] used historical travel time information from two bus lines on a common link to predict arrival time at a bus stop. [Gal et al. 2017] investigated the combination of methods of the Theory of Queues and Decision Trees for the prediction of travel times considering historical data and real-time for four lines simultaneously. They also considered dividing the bus route into links according to the bus stops.

These works all focus on travel time predictions and use ad-hoc integration of different types of models. Recently, [Zhang et al. 2016] proposed a complex citywide network of 261 bus lines but focusing their work on analyzing the static topological properties of the city bus network, and without travel time prediction method.

In our approach, we do not focus on a specific model for bus network modeling or travel time prediction. Instead, we propose a general framework for the integration of models, which allows easy development of new models that can be built over existing ones and that can scale over multiple machines.

3. Framework architecture

We propose a general framework for scalable and distributed processing of large amounts of historical and real-time data from thousands of bus lines. This framework would be useful for different applications, such as predicting bus travel times, evaluation of the flow along the bus routes, and preventing the occurrence of bus bunching.

3.1. Models

The core framework concept are `Models` (Figure 1), which can represent: (i) different views of the bus system state, pre-processed by some data analysis algorithm, such as estimated bus position, estimated links states, estimated bus headways; or (ii) predictions of future developments, such as travel time predictions in each link, evolution of link states, and likelihood of bus bunching occurrences.

There can be interdependence between models, such as the bus bunching prevention algorithm, which requires a model of the evolution of bus headways and estimation of bus positions and distances. In these cases, the model that requires the result from other models can use the results of the last execution of these models, or request a new execution of these models and wait for the results.

Models maintain information that can be used by other models and keeps this data in shared databases. After finishing its estimation or prediction algorithm, the model updates its state and writes the new state in shared databases, from which other models

can read. These shared databases improve processing scalability since it decouples the executions of different models.

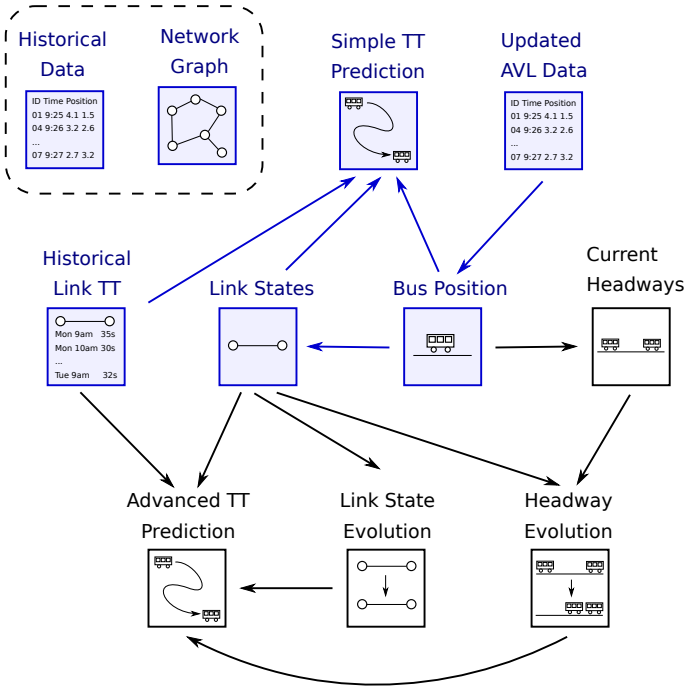

Figure 1. Set of models with direct data dependencies depicted with arrows. Historical Data and Network Graph models are used by several models and are showed separately. The models in blue are part of a possible sequence of model executions, triggered by an update in the AVL Data model.

3.2. Model Execution

The execution of each model is independent of other models, as each can have different execution schedules. This independence permits the execution of models of different complexities. For instance, there can be bus travel time prediction using simple heuristics, such as the mean travel time of the last three buses in the line, to more computationally demanding ones, such as executing deep neural networks models for predicting the evolution of link states in the following two hours.

The framework updates these models periodically, respecting some updating schedule, which can be periodical, for example, every 10 minutes, or started by some event, such as the retrieval of updated bus position information.

Large cities such as São Paulo may have tens of thousands of buses and to generate real-time predictions and estimations requires large computational power. The organization of the framework as independent tasks that updates models allows the processing to occur in distributed machines, which can also access distributed databases. The types of models that can be executed in real-time will depend on the hardware available, but the framework permits a natural expansion of processing capabilities by adding new computational resources, located in a private cluster or the cloud.

3.3. Bus System as a Graph

The `Network Graph` model maintains the representation of the bus system as a graph, with vertices representing points of interest, such as bus stops, and edges (links) representing the path between these points of interest. It keeps a list of links and their properties, such as length, position, and list of bus lines that use it.

Most models use this graph representation. For instance, the `Bus Position` model maps position estimations into graph links. Other models establish link properties, such as the mean historical travel time, the travel time of the last k-buses, the current link state, among others.

3.4. Types of Models

There are two main types of models: estimation and prediction models. Estimation models deal with the evaluation of the bus system current state and include models on bus positions, bus headways, and link states. Estimation models usually are fast to evaluate and, if required, can be computed whenever the bus positions are updated.

Predictions models deal with the prediction of future states of the model, such as bus travel times, bus bunching predictions and link state evolution. The framework can schedule the update of some simpler prediction models, such as the prediction of travel times from the current link state or from mean travel times, whenever bus positions are updated. However, for other models, such as predictions on the evolution of the link states or more complex bus travel time predictions, using neural networks or Hidden Markov Models, the framework may schedule their updates less frequently.

Additional models can also be added, such as trip planning models, which could generate trip proposals based on bus routes and schedules and travel time predictions, and data analysis models, which can gather bus bunching information and generates useful statistics of these events.

3.5. Composite Model Dependency Graph

Composite models can be constructed from simple models using shared databases. The framework updates the models when updated position data arrives. It then follows a dependency graph of model updates, respecting the dependencies in processing. It first updates the bus position estimation, then the models that depend on it, such as the link states and the bus headways. It then updates other models, such as bus travel time estimates that depend on this information.

The dependency graph can be long and contain branches, but not all models must update their state on every bus position update. For instance, complex models can generate new predictions about the evolution of links states in the next hours or about the city-wide evolution of bus headways. The framework can schedule these models for updating with a minimum interval of 10 minutes or more. If these models appear in a dependency graph but have updated their state more recently than the minimum update interval, the downstream models in the dependency graph use the data from the last update of these models. This design provides more flexibility for constructing dependency graph with models of different complexities and different update frequencies.

3.6. Input and Output Data

Real-time data can be obtained directly from APIs available at transit authorities and fed into the framework models. These data can generate updates in the bus position estimation model, which in turn can generate new updates to the states of the links, which causes new travel time estimates for buses.

Historical data usually is available as data files or in databases formats and contain several inconsistencies, missing fields, and other errors. At this moment, we are not dealing with the initial treatment of this data, but instead, consider that they have been already cleaned.

The framework can be accessed by external clients, which can request, for instance, predictions of travel times for bus lines, or the state of all the links from the city for visualization purposes. However, the purpose of the proposed framework is not to deal with individual requests from millions of users. In this case, we consider that an external system would perform this task, by periodically requesting the full model data to the framework and treating the individual requests locally.

4. Implementation

4.1. Distributed Execution using Dask

For the framework implementation we used Python with Dask.distributed, a lightweight library for distributed computing. This library consists of a centralized scheduler and distributed workers which can communicate with each other. Dask provides scalability by permitting the inclusion of more workers as demanded by the framework and supports complex workflows dependency graphs beyond those of map/filter/reduce.

The scheduler sends each task to a different worker which can be running on one or more machines (Figure 2a). Dask manages data transfer from the coordinator process to each worker. Each model stores the data it generates in a local or remote database, and models that use this data can access it directly on the databases. The databases are the primary interaction point among models, and we are using MongoDB non-relational database in the current implementation. Writes to single documents are atomic in MongoDB, preventing race conditions.

We implemented dependency graphs using *future* objects, which hold a promise of a result from a task. Each task executes a complete model or parts of it (Figure 2b). Tasks that depend on the results from other tasks wait for all upstream tasks to complete and then start its computation. Notice that the framework can execute many dependency graphs concurrently, exploiting the available computational resources. Moreover, when constructing the dependency graph, multiple workers can be assigned to a single task, improving the parallelization of the execution. Finally, models that have a minimum interval between executions can, if necessary, skip their execution, and the downstream tasks then get the results from the last execution from the shared database.

4.2. Implemented Models

We already implemented some models in the framework, which we used as a prototype implementation to validate the framework architecture.

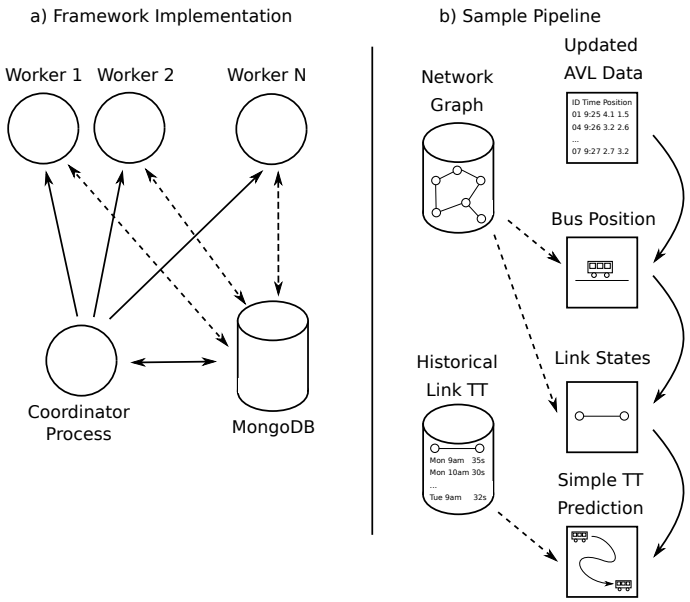

Figure 2. Distributed task execution and management using Dask. a) A coordinator process launches the tasks that are executed by workers (solid lines). The worker read data from other models and write their updated results in the MongoDB database (dashed lines). b) A sample graph of models. Models downstream can wait for other models to finish their execution (solid lines) or obtain data from other models that were already present in the database (dashed lines).

Bus Position: This model estimates the distance traveled by each bus in its predetermined route, interpolate this data and estimate the passing time at any arbitrary point of the bus route.

Graph Model: This model transform general transit feed specification (GTFS) files into a citywide graph containing all bus routes from a city. It defines the midpoints between bus stops of a bus route as vertices and the paths between consecutive vertices as links. The model first defines subgraphs for individual bus routes and then concatenates them into a citywide graph, identifying common links to multiple bus routes.

K-last Buses Model: This model determines the mean travel time of the last k buses that crossed a link in the city graph. The model retrieves data from the Bus Position model, querying for the positions of all buses in the last few hours. It then finds the last k travels on each link and calculates their mean travel times. Each call to this model evaluates a different link in the graph.

Mean Travel Time Model: This model evaluates the historical mean travel time of buses in all graph links. The model retrieves data from the Bus Position model, querying for the positions of all buses for a specific time range (e.g., from 9 am to 10 am), a specific weekday (e.g., Mondays), and a specific calendar range (e.g., from January 2017 to July of 2017). From this data, it estimates the time required for each bus to transverse the path

between the vertices of a link and then evaluates the mean travel time on the link for all buses. Each call to this model evaluates a different link in the graph.

Combined Model: This model combines the results of the Mean Travel Time and K-last Buses models by calculating a weighted mean between these two values, giving weight 1 do the historical mean as weight 2 to the K-last mean. In the current implementation, it waits for the *futures* from the Mean Travel Time and K-last Buses models.

Total Travel Time Model: This model predicts the total travel time in a bus route by summing the individual link travel times from the complete route. Each call to this model evaluates a single bus line, and in the current implementation, it waits for the *futures* from the corresponding models used for link travel time estimation.

4.3. Extending the Framework

One can include new models by writing new Dask tasks that perform the required model computations and access data from other models through the shared databases. Developers can create dependency graphs for composite models by making downstream tasks wait for the results of previous tasks in the graph using Dask *futures*.

The current framework prototype does not yet contain a model to collect real-time AVL data, although we easily implement an AVL Data model task that accesses an online API, such as SPTrans Olho Vivo[1]. We could then construct a dependency graph, which would call the tasks that execute the Bus Position model, the link travel time models, and the Total Travel Time model.

We will also implement in the framework an API that permits external clients to perform queries to obtain model results, including travel time predictions and bus arrival times. The responses to the queries will be asynchronous to the model updates since they will only return the most up to date results.

5. Experimental Setup

We performed an experimental evaluation over the implemented prototype. We simulated a real-time travel time prediction scenario, using three travel time prediction algorithms, implemented in the K-last Buses Model, Mean Travel Time Model, and Combined Model, described in the Implementation section. We used $k = 3$ buses in the K-last Buses Model.

In the experiments, we did not execute the online retrieval of AVL data and the Bus Position estimation model. The models obtained the interpolated bus position data directly from a shared MongoDB database. These steps of obtaining AVL data and updating the bus positions should not take longer than the travel time predictions steps.

We executed a simulated real-time prediction of the total travel time of each bus, considering a set of 110 routes. We consider a single business weekday, from 6:00 am. to 11:50 pm., and generated a new prediction every 10 minutes. We used three machines, each with 2 CPUs model *Intel(R) Xeon(R) CPU E5-2620v2@2.10GHz* with 6 (12) physical (virtual) cores per CPU and 128GB of RAM. One machine held the shared database

[1] http://www.sptrans.com.br/desenvolvedores/

and the dask scheduler, with the workers distributed in up to the three machines (A, B, and C), depending on the evaluated scenarios.

First experiment: Execution time. We executed the framework with different setups to evaluate the processing time, CPU usage, and scalability of the framework. In all setups, we executed MongoDB and the scheduler in machine A. We considered scenarios with 6 to 72 workers, distributed as follows:

1. Machine A, with 6 workers;
2. Machine A, with 12 workers;
3. Machine A, with 24 workers;
4. Machines A and B with 48 workers, 24 on each;
5. Machines B and C with 48 workers, 24 on each;
6. Machines A, B, and C with 72 workers, 24 on each.

Note that in configuration 5 we used 3 machines since machine A was executing MongoDB and the scheduler.

We measured the execution time for the combined prediction of the bus travel from all lines. We evaluated predictions for a single time, an hour interval (6 predictions), and a complete day. The last two were used to evaluate the scalability better. We repeated the execution of each setup five times. During the execution, we also collected the total machine CPU usage and the MongoDB Server CPU usage.

Second Experiment: total travel time prediction accuracy. Besides the execution times, we also collected the predictions generated by the framework and compared with the actual bus travel times to measure the quality of the predictions. We collected the following predictions for 110 bus lines (to a total of 1977 links):

- Mean Travel Time Model: the model estimated the mean travel time (MTT) on each link using historical data from 7 months (January to July of 2017). For each link, the model gathered data from the corresponding weekday and full hour from MongoDB, and generated a mean predictor for every hour from 6:00 am. to 11:00 pm. for all graph links.
- K-last Buses Model: the model estimate the travel time on each link by obtaining the last $k = 3$ bus travels to cross each link in a 2-hour time frame. It then evaluated the k-last mean travel time (KTT) of all links. Different from the MTT, we generate a KTT prediction every 10 minutes from 6:00 am. to 11:50 pm.
- Combined Model: this model combines both MTT and KTT predictions using a weighted mean, with $mtt_weight = 1$ and $ktt_weight = 2$ to calculate the final prediction for every link.

We compared all bus travel time predictions to actual bus travel times from 110 bus lines in the August, 1st, chosen as the first weekday in August.

6. Experimental Results

6.1. Execution Time and Scalability

Our evaluations show that the processing time decreased as we increased the number of workers, except for 72 workers (Figure 3). When evaluating travel times at full hours (e.g.,

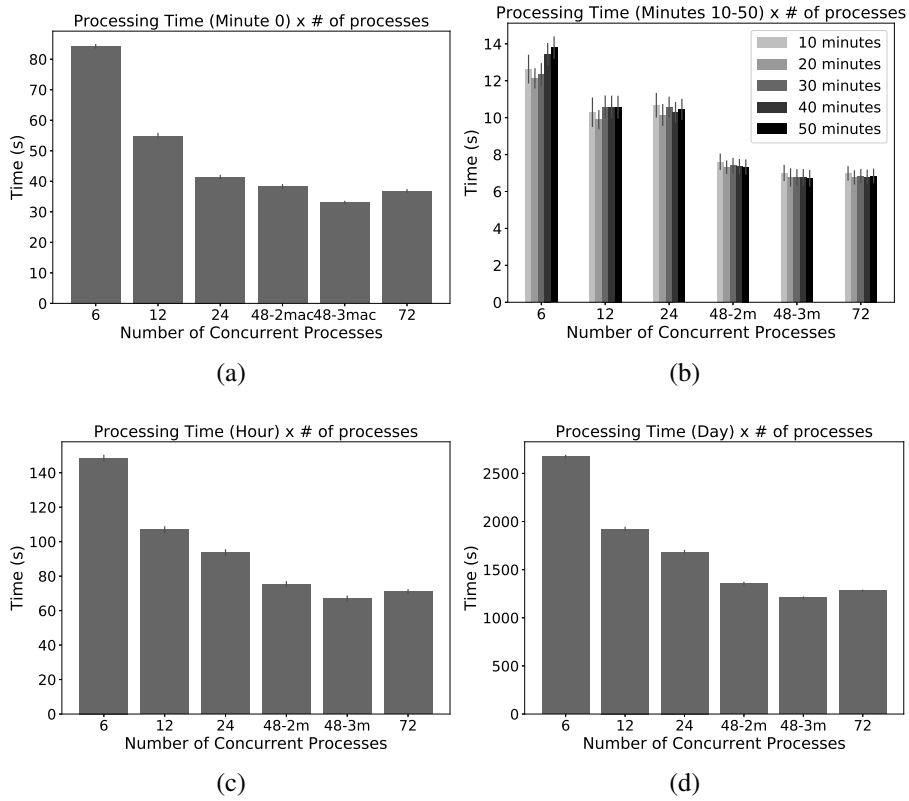

Figure 3. Execution time for predicting the travel times of 110 bus lines, using one machine (6, 12, and 24 workers), two machines (48 workers), and three machines (48 and 72 workers).

at 7 am), the framework executed both MTT and KTT models and required 40 seconds with using 24 workers (Figure 3a) or 30 seconds with 48 workers and three machines (48-3mac). When evaluating minutes 10 to 50, it executes only KTT and the processing time can be lower than 8 seconds for every 10 minutes (Figure 3b). We also evaluated the total time to calculate a whole hour (Figure 3c) and day (Figure 3d). We can see that it follows the same trend as the smaller computations.

The improvement in processing time is less evident when using more than 24 workers. To evaluate this phenomenon, we monitored the overall CPU usage on Machine A and Mongo CPU usage. The collected data shows that with up to 24 workers, running in the same machine, the CPU usage is slightly below 100% (Figure 4c), where 100% represent full usage of the 24 virtual cores in the machine, with MongoDB using about 75

The most interesting case is when machine A only executes MongoDB and the scheduler (Figure 4f), which was the configuration with the smallest total execution time. In this case, MongoDB used almost all CPU resources. When executing 72 workers (Figure 4d), MongoDB had to divide the CPU of machine A with workers and the total execution time increased.

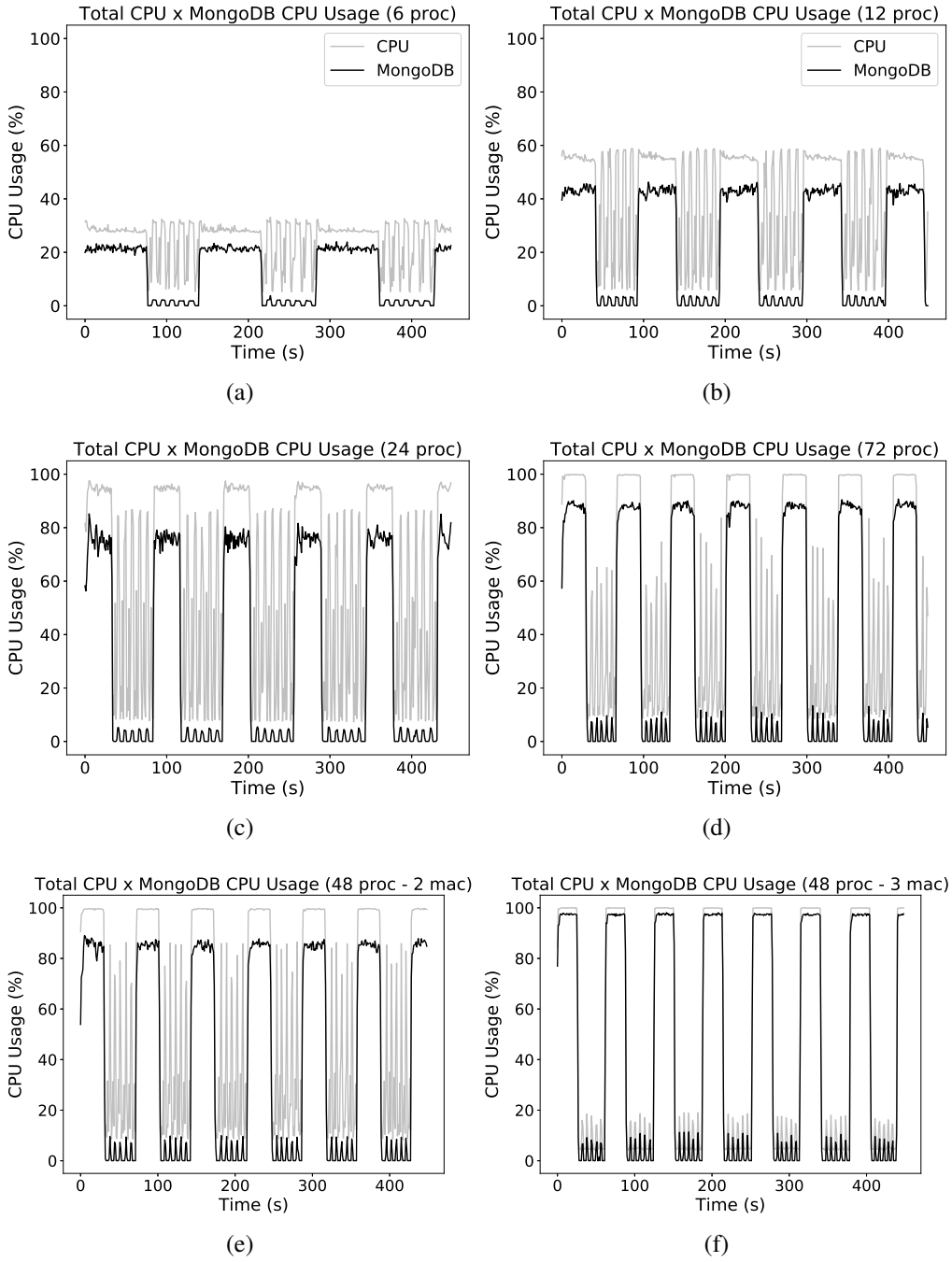

Figure 4. CPU usage measurement for machine A in the six experiments setups: (a) 6 workers, (b) 12 workers, (c) 24 workers, (d) 72 workers, (e) 48 workers in 2 machines, and (f) 48 workers in 3 machines

6.2. Travel Time Prediction Accuracy

In the third experiment, we collected the predictions generated by our prediction algorithms and compared to real travel time data, to evaluate the accuracy of these models. Note that our focus in this paper is the architecture and prototype implementation of the distributed framework and the prediction algorithms used here are rather simple. We calculated three measures: Rooted Mean Squared Error (RMSE), Mean Absolute Error

(MAE) (in minutes) and Mean Absolute Percentage Error (MAPE).

The model KTT performed slightly worse than MTT and combined models (Table 1). This performance probably happened because KTT used a few data points to make each prediction (only the latest three travels before the target travel). Although it captures better the real-time state of each link, it is more sensitive to errors in GPS measurements and unexpected behaviors, for instance, from an unusual delay in one of the latest three buses. Combining both MTT and KTT models with a weighted mean improved RMSE the predictions by a minimal margin.

Table 1. Error Measures

Algorithm	RMSE (min.)	MAE (min.)	MAPE (%)
MTT	9.21	6.70	11.12
KTT	12.05	8.37	13.95
COMBINED	8.97	6.68	11.22

7. Discussion

We presented a framework for the scalable execution of composite models of public bus systems, describing its general architecture and a prototype implementation using Dask and MongoDB. We showed that it could successfully process data from more than one hundred routes in São Paulo in near real time, using a single node with 12 physical cores.

The experimental results indicate that the database can be a significant bottleneck in data dependent models, such as determining the historical mean travel times on network links. However, on more CPU-bound models, such as neural networks or hidden Markov models, the database may not be a critical bottleneck.

There are several possible solutions for improving the scalability of the database. A simpler one is to replicate MongoDB in multiple machines, with multiples copies of data that is highly accessed. For static data this extension is simple, but dynamic data may require updates on multiple sites to guarantee consistency. However, since the results of model processing generate smaller amounts of data, this may not be an issue. Another approach is to use columnar databases for storing data for some models that depend on the data aggregation, such as the mean travel times model. We are evaluating these approaches for applying in the framework.

References

Adachi, H., Suzuki, H., Asahi, K., Matsumoto, Y., and Watanabe, A. (2015). Estimation of bus traveling section using wireless sensor network. In *2015 Eighth International Conference on Mobile Computing and Ubiquitous Networking (ICMU)*, pages 120–125. IEEE.

Bin, Y., Zhongzhen, Y., and Baozhen, Y. (2006). Bus arrival time prediction using support vector machines. *Journal of Intelligent Transportation Systems*, 10(4):151–158.

Cascetta, E. (2009). *Transportation Systems Analysis: Models and Applications*, volume 29. Springer US.

Chang, H., Park, D., Lee, S., Lee, H., and Baek, S. (2010). Dynamic multi-interval bus travel time prediction using bus transit data. *Transportmetrica*, 6(1):19–38.

Chen, M., Liu, X., Xia, J., and Chien, S. I. (2004). A dynamic bus-arrival time prediction model based on apc data. *Computer-Aided Civil and Infrastructure Engineering*, 19(5):364–376.

Chien, S. I.-J., Ding, Y., and Wei, C. (2002). Dynamic bus arrival time prediction with artificial neural networks. *Journal of Transportation Engineering*, 128(5):429–438.

Choudhary, R., Khamparia, A., and Gahier, A. K. (2016). Real time prediction of bus arrival time: A review. *Proceedings on 2016 2nd International Conference on Next Generation Computing Technologies, NGCT 2016*, (October):25–29.

Fu, J., Wang, L., Pan, M., Zuo, Z., and Yang, Q. (2014). Bus arrival time prediction and release: System, database and android application design. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 404–416. Springer.

Gal, A., Mandelbaum, A., Schnitzler, F., Senderovich, A., and Weidlich, M. (2017). Traveling time prediction in scheduled transportation with journey segments. *Information Systems*, 64:266–280.

Jeong, R. and Rilett, R. (2004). Bus arrival time prediction using artificial neural network model. In *Intelligent Transportation Systems, 2004. Proceedings. The 7th International IEEE Conference on*, pages 988–993. IEEE.

Kumar, B. A., Jairam, R., Arkatkar, S. S., and Vanajakshi, L. (2017a). Real time bus travel time prediction using k-NN classifier. *Transportation Letters*, 7867(August):1–11.

Kumar, B. A., Vanajakshi, L., and Subramanian, S. C. (2017b). A hybrid model based method for bus travel time estimation. *Journal of Intelligent Transportation Systems*, 2450(December):1–17.

Kumar, B. A., Vanajakshi, L., and Subramanian, S. C. (2017c). Bus travel time prediction using a time-space discretization approach. *Transportation Research Part C: Emerging Technologies*, 79:308–332.

Lin, W.-H. and Zeng, J. (1999). Experimental study of real-time bus arrival time prediction with gps data. *Transportation Research Record: Journal of the Transportation Research Board*, 1666:101–109.

Mazloumi, E., Rose, G., Currie, G., and Sarvi, M. (2011). An integrated framework to predict bus travel time and its variability using traffic flow data. *Journal of Intelligent Transportation Systems*, 15(2):75–90.

Mori, U., Mendiburu, A., Álvarez, M., and Lozano, J. A. (2015). A review of travel time estimation and forecasting for Advanced Traveller Information Systems. *Transportmetrica A: Transport Science*, 11(2):119–157.

Shalaby, A. and Farhan, A. (2004). Prediction model of bus arrival and departure times using avl and apc data. *Journal of Public Transportation*, 7(1):3.

Yin, T., Zhong, G., Zhang, J., He, S., and Ran, B. (2017). A prediction model of bus arrival time at stops with multi-routes. *Transportation Research Procedia*, 25:4627–4640.

Yu, B., Lam, W. H., and Tam, M. L. (2011). Bus arrival time prediction at bus stop with multiple routes. *Transportation Research Part C: Emerging Technologies*, 19(6):1157–1170.

Yu, B., Yang, Z. Z., and Wang, J. (2010). Bus travel-time prediction based on bus speed. In *Proceedings of the Institution of Civil Engineers-Transport*, volume 163, pages 3–7. Thomas Telford Ltd.

Yu, B., Yang, Z.-Z., and Zeng, Q.-C. (2008). Bus arrival time prediction model based on support vector machine and kalman filter. *China Journal of Highway and Transport*, 2:016.

Yu, H., Wu, Z., Chen, D., and Ma, X. (2017). Probabilistic prediction of bus headway using relevance vector machine regression. *IEEE Transactions on Intelligent Transportation Systems*, 18(7):1772–1781.

Zhang, X., Chen, G., Han, Y., and Gao, M. (2016). Modeling and analysis of bus weighted complex network in qingdao city based on dynamic travel time. *Multimedia Tools and Applications*, 75(24):17553–17572.

Zhenliang, M., Jianping, X., Shihao, Y., and Yubing, W. (2011). An Aggregation Method for Dynamic Bus Arrival Time Prediction. *Traffic*, 3(July):37–48.

Zhou, P., Zheng, Y., and Li, M. (2012). How long to wait?: predicting bus arrival time with mobile phone based participatory sensing. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 379–392. ACM.

Zuo, Z. and Wang, L. (2013). Bus arrival time forecasting and real-time information publication technology. *Journal of Transportation Systems Engineering and Information Technology*, 1:012.