Accelerated on-line calibration of dynamic traffic assignment using distributed stochastic gradient approximation


The MIT Faculty has made this article openly available. Please share how this access benefits you. Your story matters.
Accelerated On-line Calibration of Dynamic Traffic Assignment using Distributed Stochastic Gradient Approximation

Enyang Huang, Constantinos Antoniou, Jorge Lopes, Yang Wen and Moshe Ben-Akiva

Abstract—Dynamic Traffic Assignment (DTA) system [Ben-Akiva et al., 1991] [Mahmassani, 2001] benefits travelers by providing accurate estimate of current traffic conditions, consistent anticipatory network information as well as reliable route guidance. Over the years, two types of model adjustment schemes have been studied - DTA off-line calibration [Balakrishna, 2006] [Toledo et al., 2003] [van der Zijpp, 1997] and DTA on-line calibration [Antoniou et al., 2007] [Wang et al., 2007] [Ashok and Ben-Akiva, 2000]. The on-line calibration of DTA system allows real-time model self-corrections and has proven to be a useful complement to off-line calibration. In this paper, we explore distributed gradient calculations for the speed-up of on-line calibration of Dynamic Traffic Assignment (DTA) systems. Extended Kalman Filter (EKF) and Stochastic Gradient Descent (SGD) are examined and their corresponding distributed versions (Para-EKF and Para-GD) are proposed. A case study is performed on a 25-km expressway in Western Portugal. We empirically show that the application of distributed gradient calculation significantly reduce the computational time of on-line calibration and thus provide attractive alternatives for speed-critical real-time DTA systems.

I. INTRODUCTION

Calibration of DTA systems can be grouped into two categories. Off-line calibration of DTA system typically results in a historical database of model parameters that reflect average or expected traffic conditions over a period of time. The on-line calibration of DTA system is a useful complement of the off-line procedure by allowing adjustment of model parameters in real-time. Because of this nature, modern DTA are time critical systems. On-line calibration of such systems at their operational time dictates efficient algorithms.

Over the years, application of parallelization in DTA on-line calibration has been proposed. However, literature lacks relevant real-world implementations and case studies. In this paper, we explore efficient implementation of the on-line calibration process. In particular, distributed implementations of two classic on-line calibration algorithms - EKF and GD are studied. We first present their single-processor serial implementation and then identify their bottle-neck, namely procedures that perform numerical gradient approximations. We show that it is possible to parallelize the computation of the gradient and we propose the lossless Para-EKF and Para-GD algorithms. A case study on a real-world expressway network is used to demonstrate the feasibility of our approach. Empirical results support our expectation that the speed of Para-EKF and Para-GD significantly outperform their serial versions. We conclude with future research directions.

II. STATE-OF-THE-ART

A. Notation

Dividing time period $T$ into time interval $1, 2, 3, ..., N$ of size $t$, the network is represented as a directed graph with a set of links and nodes. Each link is composed of a set of links and OD pairs. Furthermore, each segment can be equipped with multiple types of sensors. For example, $n^\text{count}_g$ segments are equipped with count sensors and $n^\text{speed}_g$ segments are equipped with speed sensors.

Let $\pi_h$ denote the vector of OD flows and model parameters subject to on-line calibration at time interval $h$. $(\pi_h = [x_h, y_h])$, $x_h$ is the OD flows at time $h$ and $y_h$ is the model parameters at time $h)$. Let $S$ denote a DTA simulator and $M^\text{obs}_h$ denote a vector of observed traffic conditions for time interval $h$ (For example, travel time, segment flow counts). Let $M^\text{sim}_h$ denote a vector of corresponding simulated traffic conditions from $S$.

B. State-Space Formulation and Algorithm

1) State-Space Formulation: The classical approach to model dynamic systems is to use the state-space formulation [Antoniou et al., 2007]. The state-space model consists of transition and measurement equations. The first step in defining the state-space formulation is the definition of state. For the on-line calibration of DTA systems, a state consists of model parameters and OD flows. Explicitly, let $\pi_h$ denote the state for time interval $h$, so that:

$$\pi_h = [x_h, y_h]^T$$

The transition equation captures the evolution of the state vector over time. The general formulation is that: $\pi_{h+1} = \pi(\pi_h, \pi_h, \ldots, \pi_{h-p}) + \epsilon^\text{auto}_h$, where $\pi$ is a function that describes the dependence of $\pi_{h+1}$ in its previous $p$ states, $\epsilon^\text{auto}_h$ is a vector of random errors. In this context, an autoregressive function for $\tau$ is used. The measurement equations are in two parts. The direct measurement equations capture the error between the state vector and its a priori values. The indirect measurement equation links the state vector with...
the sensory observations. [Ashok and Ben-Akiva, 1993] proposed to write the state-space model in its deviation from its historical. The recommendation stems from two main reasons. Firstly, the deviation form implicitly incorporates the wealth of information contained from the offline calibrated parameters and OD flows. Secondly, the deviation form allows the normality assumption to hold for the error terms in the model. Without using the deviation form, the state variables, such as the OD flows, will have a skewed distribution. Normality assumption is useful in the application of Kalman Filtering techniques. For these reasons, [Ashok and Ben-Akiva, 1993] write: \( \Delta \pi_h = \pi_h - \pi_h^\text{hist} \), and our final state-space model in the deviation form is:

\[
\Delta \pi_h = \Delta \pi_h + \epsilon_h^\text{obs} + \epsilon_h^\text{auto} \\
\Delta M_h = S(\pi_h^H + \Delta \pi_h) - M_h^H + \epsilon_h^\text{obs} \\
\Delta \pi_{h+1} = \sum_{q=h-p}^{h+1} \Delta \pi_{q} + \epsilon_{h}^\text{auto} \tag{4}
\]

Where the eq. 2 implements the direct measurement equation using a priori observation \( \Delta \pi_h^0 \), eq. 3 implements the indirect measurement equation through the use of DTA simulator \( S \) and sensory observations \( M_h^H \). Eq. 4 implements a q degree auto-regressive process \( F_{h+1}^h \) that captures the relationship between state at time \( h+1 \) and past state at time \( h \).

2) Extended Kalman Filter: The Extended Kalman Filter comes naturally for the state-space model. Alg. 1 presents the details of EKF for on-line DTA calibration.

The EKF algorithm starts by initializing state vector \( X \) and the variance-covariance matrix \( P \). This is followed by a loop indexed on the time interval \( h \) during the DTA model simulation.

At each iteration, a number of tasks are performed in sequence. The time update models the transition process of the state vector, as in eq. 7 and the variance-covariance matrix, as in eq. 8. This is followed by the numerical approximation of the Jacobian Matrix for time interval \( h \), computed in eq. 9. Then the Kalman Gain \( G_h \) is calculated in eq. 10. The state vector, as computed in eq. 11, is thus the sum of two components - the vector output from time update (its time evolution) and the difference between sensory observation \( Y_h \) and its simulated counter-part \( h(X_{h|h-1}) \), but magnified by the quantity of the Kalman Gain. Finally, the variance-covariance matrix is updated as in eq. 12.

C. Direct Optimization Formulation and Algorithm

1) Direct Optimization Formulation: A state-space equivalent formulation is the so-called direct optimization formulation [Balakrishna, 2006] [Vaze et al., 2009] [Huang, 2010]. [Ashok, 1996] discussed the connection between the state-space and the Generalized Least Square (GLS) direct minimization formulations in general, and, Kalman Filter and least square estimation as their respective solutions in specific. It was argued that the application of the results obtained by the classical GLS to discrete stochastic linear processes leads to precisely the Kalman Filter. Therefore an alternative way of formulating the on-line calibration framework is to adopt the GLS direct optimization. In the direct optimization formulation, we jointly minimize the three errors from the three state-space equations: \( \epsilon_h^a, \epsilon_h^\text{obs} \) and \( \epsilon_h^\text{auto} \). Because the historicals cancel out when we subtract each equation for the error terms, the direct optimization can be formulated as:

\[
\pi_h^* = \arg\min[N_1(\epsilon_h^a) + N_2(\epsilon_h^\text{obs}) + N_3(\epsilon_h^\text{auto})] \tag{13}
\]

If we assume that \( \epsilon_h^a, \epsilon_h^\text{obs} \) are normally distributed and uncorrelated, and, \( N_i \) signifies squared distance, the above can be translated into the following generalized least square (GLS) formulation:

\[
\pi_h^* = \arg\min[(\pi_h^0 - \pi_h^0)^TV^{-1}(\pi_h^0 - \pi_h^0) + (M_h^im - M_h^im)^TW^{-1}(M_h^im - M_h^im) + (\pi_h - \sum_{q=h-p}^{h-1} F_{q+1}^q \pi_q) Z^{-1}(\pi_h - \sum_{q=h-p}^{h-1} F_{q+1}^q \pi_q)] \tag{14}
\]

Where, \( V, W \) and \( Z \) are the variance-covariance matrix of \( \epsilon_h^a, \epsilon_h^\text{obs} \) and \( \epsilon_h^\text{auto} \) respectively.

2) Stochastic Gradient Descent: GD is a collection of simple yet elegant ideas that can be used to solve the direct optimization framework. The GD procedure requires O(N)
number of function evaluations for state size of N. However, GD is matrix free - no matrix inversion is performed. Thus it significantly outperforms EKF in memory complexity. GD requires slightly more function evaluations in its line search routine. This incremental cost is largely constant irrespective of the size of the state variables. The main routine of the algorithm is presented in Alg. 2.

Algorithm 2 The Gradient Descent Algorithm

Initialization

Let $P_0$ be the starting point in the minimization routine

for time interval $h = 0$ to Infinity do

Direction Search

$$G = \frac{\partial f(X^*)}{\partial X} \bigg|_{X^* = x_{0|h-1}}$$ (15)

Step Size Search

$$(\lambda) = \arg \min ((X_{h|h-1} + \lambda G))$$ (16)

Update

$$X_{h|h} = X_{h|h-1} + \lambda G$$ (17)

End for loop

III. PARA-EKF AND PARA-GD

A. Background

The most computationally burdensome part of the EKF and the GD algorithms lie in their derivative approximations. In the case of EKF, the burden lies in equation 9. In the case of GD, the overheads come from equation 15. With state vector of size $n$ and assuming forward finite differential approximation, both equations require exactly $n + 1$ function evaluations. (Each function evaluation incurs a full DTA model operation). As $n$ grows, the total overheads are extremely significant.

However, the $n + 1$ evaluations can be distributed to multiple processors and thereby reduce the overall computation time in real-time operations. We notice the following characteristics of the derivative approximation routines:

- Independence - The $n + 1$ function evaluations are strictly independent from one another. In addition, the order in which the evaluations are performed does not matter. The $i^{th}$ evaluation may be performed before or after the $j^{th}$ evaluation and vice versa.

- Similar Processing Time per Evaluation - The time required to perform each function evaluation is largely identical. Small fluctuations on the running time may exist due to the perturbation of the state vector in the finite differential approximation. This property is attractive since it allows for the design of much simpler load balancing schemes. Because each evaluation takes roughly the same amount of time, the load level can be simply determined by the processing speed of each processor. This idea will come back in the case study.

- Minimal Inter-Processor Communication (IPC) - The required communications between processors are low. Depending on the design of the scheme, IPC may occur at the end of the EKF algorithm to synchronize the state vector $X$ and variance-covariance matrix $P$. The situation is even simpler for the GD algorithm. IPC may occur at the end of the GD algorithm to synchronize the state vector $X$. The amount of data transferred over the IPC is relatively moderate. With a state vector’s size of $N$ and the above scheme in operation, the amount of data transferred (payload) for the EKF algorithm would include $4N$ bytes for state vector $X$ and $4N^2$ bytes for the variance-covariance matrix $P$ (Using double precision). Lower precision may be used for the scalability requirement). This amounts to approximately 4MBs of data with $N = 1000$. With 100Mbs LAN connection, this is not a significant overhead. If the synchronization occur at the Jacobian step in equation 9, the amounts of data reduces to $4NN_{ob}$ bytes, where $N_{ob}$ is the number of surveillance sensors in the system (Usually $N > N_{ob}$). The case for the GD algorithm is similar to the analysis above. In general, with any moderate size of $N$, this process is entirely a trivial exercise.

- Lossless Decomposition - The distributed implementation of gradient approximation is lossless. The reconstructed Jacobian/Directive vector is identical to the serial implementation. The random seeds can be synchronized before calculations start for each time interval.

In the next section, we will describe the architecture of the distributed computing in detail. The implementation details will follow and a summary will be given at the last part of this section.

B. Architecture

While there are many schemes for distributed computing, this paper employs the most fundamental one - the Server-Client scheme (or sometimes called Master-Slave scheme). In this scheme, one machine takes the role of “Server” and is responsible for 1) Coordinating clients for their derivative computations 2) completing the rest of the algorithm routines before and after the computation of derivatives and 3) Initializing synchronization with clients at the appropriate point in time. The exact internal state for both Server and Clients is controlled by a finite state automaton.

The general architecture of the implementation is given below:

The master is responsible for distributing clients’ work load, receiving derivative portions from clients, centralized processing on the remaining routines, and synchronizing clients at the end of each time interval.

At the beginning of a new time interval, the server first performs iteration/loop initializations. The derivative is then calculated in the distributed fashion. The server allocates portions of the derivative calculations to every client. The derivative will be estimated using the finite differential method stated above. The server first calculates
and conducts meso-scopic traffic simulation. While traveling, drivers’ behaviors are constantly updated in response to the latest guidance. The outputs of DynaMIT-R from demand and supply interactions are network characteristics such as link speed, average travel time, densities as well as drivers’ benchmarks such as trip travel time, trip length, fuel consumption, etc.

Off-line calibration is first performed using archived surveillance data for a typical normal week day on 10-Dec-2009. OD flow levels, speed-density relationships and auto-regressive parameters are calibrated. Next, sensory data from another week day (11-Jan-2010) is used for on-line algorithm validation. The state vector has length of 187, which consists of OD flows as well as supply model parameters which include segment capacities and speed-density relationships.

**IV. EXPERIMENT**

The network used in this analysis is the Brisa A5 motorway (A5 - Auto-estrada da Costa do Estoril). It is a 25-km inter-urban expressway section between Lisbon and Cascais. The motorway consists of 85 road segments and 56 nodes representing significant points in the network. The A5 motorway is located in western Portugal.

The motorway is primarily equipped with toll collection systems, surveillance camera counters and inductive loop detectors. All these sensors provide time dependent vehicle counts at specific parts of the network. The network also deploys point-to-point sensors such as the Via-verde sensor. The sequential identification of vehicles between consecutive toll plazas using Via-verde offers measurements of average segment speeds in real-time.

DynaMIT-R (stands for DynaMIT real-time module) [Ben-Akiva et al., 1991] [Ben-Akiva. et al., 2002] is chosen as the candidate DTA system in the experiment. DynaMIT-R is a state-of-the-art DTA system for traffic network estimation and prediction developed by the MIT Intelligent Transportation Systems Laboratory. The system is composed of two major components 1) Demand simulation and 2) Supply simulation. The demand simulation conducts pre-trip behavior modeling using historical OD demand matrix.

This includes a simulation of the choice of departure time, routes and whether the trip is canceled, while considering the latest guidance information. The supply simulation takes the listed drivers with choices from the demand simulator,
The on-line calibration experiment is performed for the period of [15:00-16:30]. We use root mean square normalized (RMSN) as the performance indicator for accuracy. The RMSN is defined as:

$$RMSN = \sqrt{\frac{\sum_{n=1}^{N} (y_{sim} - y_{obs})^2}{\sum_{n=1}^{N} y_{obs}^2}}$$

We compare RMSN of our proposed calibration DTA model with a reference implementation, which performs a single iteration of OD estimation at run time. The results of the comparison is presented in table II.

The total running time and total number of function evaluations (runs of DTA) for the single-processor EKF and GD are presented in table III.

<table>
<thead>
<tr>
<th>Trial</th>
<th>Processor</th>
<th>Function Evaluations</th>
<th>Running Time (sec)</th>
<th>Improvement in Percentage Over the Base Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 CPU</td>
<td>Trial 1</td>
<td>149.21</td>
<td>3.056 × 10⁻²</td>
<td>87.05%</td>
</tr>
<tr>
<td></td>
<td>Trial 2</td>
<td>149.73</td>
<td>2.604 × 10⁻²</td>
<td>88.58%</td>
</tr>
<tr>
<td></td>
<td>Trial 3</td>
<td>152.49</td>
<td>2.651 × 10⁻²</td>
<td>89.72%</td>
</tr>
<tr>
<td></td>
<td>Trial 4</td>
<td>151.95</td>
<td>2.642 × 10⁻²</td>
<td>90.42%</td>
</tr>
<tr>
<td></td>
<td>Trial 5</td>
<td>152.38</td>
<td>2.650 × 10⁻²</td>
<td>90.47%</td>
</tr>
<tr>
<td>Mean</td>
<td>151.52</td>
<td>2.628 × 10⁻²</td>
<td>89.25</td>
<td>89.38%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trial</th>
<th>Processor</th>
<th>Function Evaluations</th>
<th>Running Time (sec)</th>
<th>Improvement in Percentage Over the Base Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 CPU</td>
<td>Trial 1</td>
<td>86.58</td>
<td>3.073 × 10⁻²</td>
<td>44.66%</td>
</tr>
<tr>
<td></td>
<td>Trial 2</td>
<td>83.93</td>
<td>2.979 × 10⁻²</td>
<td>43.71%</td>
</tr>
<tr>
<td></td>
<td>Trial 3</td>
<td>85.81</td>
<td>3.046 × 10⁻²</td>
<td>44.26%</td>
</tr>
<tr>
<td></td>
<td>Trial 4</td>
<td>86.05</td>
<td>3.055 × 10⁻²</td>
<td>44.34%</td>
</tr>
<tr>
<td></td>
<td>Trial 5</td>
<td>85.14</td>
<td>3.022 × 10⁻²</td>
<td>44.19%</td>
</tr>
<tr>
<td>Mean</td>
<td>85.5</td>
<td>3.035 × 10⁻²</td>
<td>44.23</td>
<td>45.32%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trial</th>
<th>Processor</th>
<th>Function Evaluations</th>
<th>Running Time (sec)</th>
<th>Improvement in Percentage Over the Base Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 CPU</td>
<td>Trial 1</td>
<td>65.59</td>
<td>3.343 × 10⁻²</td>
<td>29.71%</td>
</tr>
<tr>
<td></td>
<td>Trial 2</td>
<td>66.85</td>
<td>3.407 × 10⁻²</td>
<td>30.09%</td>
</tr>
<tr>
<td></td>
<td>Trial 3</td>
<td>66.85</td>
<td>3.407 × 10⁻²</td>
<td>30.53%</td>
</tr>
<tr>
<td></td>
<td>Trial 4</td>
<td>67.64</td>
<td>3.448 × 10⁻²</td>
<td>31.36%</td>
</tr>
<tr>
<td></td>
<td>Trial 5</td>
<td>65.71</td>
<td>3.349 × 10⁻²</td>
<td>29.79%</td>
</tr>
<tr>
<td>Mean</td>
<td>66.53</td>
<td>3.391 × 10⁻²</td>
<td>30.30</td>
<td>32.88%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Trial</th>
<th>Processor</th>
<th>Function Evaluations</th>
<th>Running Time (sec)</th>
<th>Improvement in Percentage Over the Base Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 CPU</td>
<td>Trial 1</td>
<td>55.63</td>
<td>4.040 × 10⁻²</td>
<td>20.89%</td>
</tr>
<tr>
<td></td>
<td>Trial 2</td>
<td>55.31</td>
<td>4.017 × 10⁻²</td>
<td>20.56%</td>
</tr>
<tr>
<td></td>
<td>Trial 3</td>
<td>54.15</td>
<td>3.932 × 10⁻²</td>
<td>20.85%</td>
</tr>
<tr>
<td></td>
<td>Trial 4</td>
<td>56.80</td>
<td>4.125 × 10⁻²</td>
<td>22.24%</td>
</tr>
<tr>
<td></td>
<td>Trial 5</td>
<td>55.76</td>
<td>4.049 × 10⁻²</td>
<td>21.66%</td>
</tr>
<tr>
<td>Mean</td>
<td>55.53</td>
<td>4.033 × 10⁻²</td>
<td>21.24</td>
<td>24.80%</td>
</tr>
</tbody>
</table>

The extended state now includes additional variables in DynaMIT-R that were not included in the previous state vector.
experiment are summarized in Table I. The same results are plotted in Fig. 2.

These experiment results are intuitive and support our expectation. As the gradient approximation gets distributed across more processors, the number of function evaluations performed by one processor (master for example) decreases, and thus the overall time spent on DTA function evaluation and algorithms’ total running time decrease.

V. CONCLUSION AND FUTURE WORK

A. Conclusions

Application of parallelization in the domain of DTA has been proposed in the literature. However, there lacks real-world implementations and case studies. In this paper we propose distributed gradient calculation in classical DTA online calibration algorithms. In particular, Para-EKF and Para-GD are proposed, implemented, and tested against their serial versions - EKF and GD. The key idea of this proposal is to defray the computational bottlenecks by spreading the calculation of the Jacobian in EKF, as well as the gradient approximation in GD, onto multiple processors. We show empirically that this approach is viable and beneficial if additional processing resources are available.

B. Future Works

The proposed Para-EKF and Para-GD are demonstrated on a non-trivial but relatively small real-world network. Further verification of the effectiveness and computational performance may need to be performed on larger case studies. Moreover, the load given to each processors are determined off-line and no load balancing is performed on-line. Further work will research and address these issues.

REFERENCES


