Large-Scale Estimation in Cyberphysical Systems Using Streaming Data: A Case Study With Arterial Traffic Estimation

Timothy Hunter, Tathagata Das, Matei Zaharia, Pieter Abbeel, and Alexandre M. Bayen

Abstract—**Controlling and analyzing cyberphysical and robotics systems is increasingly becoming a Big Data challenge. We study the case of predicting drivers' travel times in a large urban area from sparse GPS traces. We present a framework that can accommodate a wide variety of traffic distributions and spread all the computations on a cluster to achieve small latencies. Our framework is built on Discretized Streams, a recently proposed approach to stream processing at scale. We demonstrate the usefulness of Discretized Streams with a novel algorithm to estimate vehicular traffic in urban networks. Our online EM algorithm can estimate traffic on a very large city network (the San Francisco Bay Area) by processing tens of thousands of observations per second, with a latency of a few seconds.**

Note to Practitioners—**This work was driven by the need to estimate vehicular traffic at a large scale, in an online setting, using commodity hardware. Machine Learning algorithms combined with streaming data are not new, but it still requires deep expertise both in Machine Learning and in Computer Systems to achieve large scale computations in a tractable manner. The Streaming Spark project aims at providing an interface that abstracts out all the technical details of the computation platform (cloud, HPC, workstation, etc.).**

As shown in this work, Streaming Spark is suitable for implementing and calibrating nontrivial algorithms on a large cluster, and provides an intuitive yet powerful programming interface. The readers are invited to refer to the source code referred in this article for more examples.

This paper presents algorithms to sample and compute densities for Gamma random variables restricted to a hyperplane (i.e., distributions of the form $T_i | \sum_j \alpha_j T_j = d$ with T_j independant Gamma distributions). It is common in this case to use Gaussian random variables because of closed-form solutions to solve. If one considers positive valued distributions with heavy tails, our formulas using gamma distributions may be more suitable.

Manuscript received June 30, 2013; accepted July 17, 2013. Date of publication August 19, 2013; date of current version October 02, 2013. This paper was recommended for publication by Associate Editor D. Song and Editor K. Lynch upon evaluation of the reviewers' comments. This work was supported in part by Google, Amazon Web Services, Ericsson, IBM, Mark Logic, NEC Labs, Oracle, VMWare, National Sciences and Engineering Research Council of Canada, California Department of Transportation, NEC Labs, Network Appliance, Oracle, Splunk and VMWare, by DARPA (contract #FA8650-11-C-7136), and by the National Sciences and Engineering Research Council of Canada, the U.S. Department of Transportation, the California Department of Transportation, and Nokia and NAVTEQ for the ongoing partnership and support through the *Mobile Millennium* Project.

The authors are with the Department of Electrical and Computer Science, University of California at Berkeley, Berkeley, CA 94709 USA (e-mail: tjhunter@eecs.berkeley.edu; tdas,@eecs.berkeley.edu; matei@eecs.berkeley.edu; pabbeel@cs.berkeley.edu; bayen@berkeley.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASE.2013.2274523

Index Terms—**Arterial traffic, arterial traffic estimation, expectation-maximization, large-scale estimation, streaming, streaming spark, travel times.**

I. Introduction

TRAFFIC congestion affects nearly everyone in the world due to the environmental damage and transportation delays it causes. The 2007 Urban Mobility Report [33] states that traffic congestion causes 4.2 billion hours of extra travel in the United States every year, which accounts for 2.9 billion extra gallons of fuel and an additional cost of $78 billion. Providing drivers with accurate traffic information reduces the stress associated with congestion and allows drivers to make informed decisions, which generally increases the efficiency of the entire road network [9]. Researchers on Traffic Information Systems (TIS) broadly agree that accurate information is critical to increase their usage [12]. So far however, it seems only a small fraction of the drivers uses TIS [16]. In this scenario, we can consider that revealing the true state of the traffic to the participants will not change the dynamics of the overall phenomenon: the informed drivers will optimize their routes or schedule without changing (much) the global equilibrium of the other road users.[1]

Modeling highway traffic conditions has been well-studied by the transportation community with work dating back to the pioneering work of Lighthill *et al.* [25]. Recently, researchers demonstrated that estimating highway traffic conditions can be done using only GPS probe vehicle data [34]. Arterial roads, which are major urban city streets that connect population centers within and between cities, provide additional challenges for traffic estimation. Recent studies focusing on estimating real-time arterial traffic conditions have investigated traffic flow reconstruction for single intersections using dedicated traffic sensors. Dedicated traffic sensors are expensive to install, maintain and operate, which limits the number of sensors that governmental agencies can deploy on the road network. The lack of sensor coverage across the arterial network thus motivates the use of GPS probe vehicle data for estimating traffic conditions.

The specific problem we address in this use case is how to extract travel time distributions from *sparse, noisy* GPS measurements collected *in real-time* from vehicles, and over a *very large network*. A probabilistic model of travel times on the arterial network is presented along with an online Expectation Maximization (EM) algorithm for learning the parameters of this model (Section II). The algorithm is expensive due to the

[1]A working draft of this paper was presented in [22]

large dimension of the network and the complexity inherent to the evolution of traffic. Furthermore, our EM algorithm has no closed-form expression and requires sampling and nonlinear optimization techniques. This is why the use of a distributed system is appropriate.

This EM algorithm is the core of an estimation pipeline deployed inside the *Mobile Millennium* traffic information system [2], [23]. This engine gathers GPS observations from participating vehicles and produces estimates of the travel times on the road network. *Mobile Millennium* is intended to work at the scale of large metropolitan areas: the road network considered in this work is a real road network (a large portion of the greater Bay Area, comprising 506,685 road links) and the data for this work is collected from thousands of vehicles that generate millions of observations per day. As a consequence of these specifications and requirements, we employ highly scalable traffic algorithms. Furthermore, *Mobile Millennium* is a research platform and can be used with various models of travel times. The fundamental unit of estimation is the probability distribution of travel times over a single link of the road network. As we will see, our framework can accommodate *any* distribution of travel times that provided they expose a few functionalities (sampling, parameter estimation from observations). This should be of interest to the traffic researchers and practitioners since our framework solves all the issues of using raw GPS samples to build traffic estimates at a very large scale with low latency. Our framework has been released under an open-sources license and is available for download [7].

More generally, our system presents a way to cope with large amounts of data from automation systems in a principled way. Industries such as genomic and astronomy have learned to cope with extremely large datasets over the last decade. What makes cyberphysical systems stand out amongst these applications is the fast decay of the value of information: in robotics systems for example, the data collected from sensors is usually fed into a control system. Past information is often of limited or no value, sometimes as fast as in the span of a few minutes or tens of seconds. This is unlike genomic records which, rather than being processed immediately, need to be stored reliably for a long time. In essence, the incoming information in cyberphysical systems needs to be considered as a stream, and not so much as an ever-growing dataset. In this setting, the design of the computing platform becomes critical to achieve both *scalability* (which implies robustness to computer failures) and *latency*, which is all the more important as estimates are usually part of a larger decision system. In this paper, we investigate the use of Discretized Streams (D-Streams) [35], a novel computing technique that process flows of incoming data on a cluster.

The present work is novel for three reasons.

- Our framework can work with a very large class of travel time distributions proposed in the literature [17], [20], [26]. If a travel time distribution can perform some elementary operations described in Section II-D (conditional sampling, maximum-likelihood estimation), then it can be used in our highly scalable architecture. Thus, transportation researchers can focus on designing good travel time distributions, leaving the system aspects aside should they wish to deploy it on a real system.
- Our framework can accommodate complex distributions and spread the computations across a large cluster. The natural baseline is Gaussian distributions, because a number of important operations have close-formed solutions, unlike other distributions. However, Gaussian distributions are not adapted to represent traffic distributions: they are not limited to the positive reals, and they lack a heavy tail (i.e., they are sensitive to noise and outliers). These considerations drive our use of another simple distribution: the Gamma distribution. We present novel results regarding the sampling from conditioned Gamma distributions.
- Building such a system is at the forefront of research in large scale systems. Our algorithm (an EM algorithm on streaming data) is representative of a large class of Machine Learning algorithms used in robotics and automation, and our overall design could be used as well for these other algorithms and applications. This is why we present and explain the design of our system, as we find there are valuable lessons for practitioners.

We start by presenting our general approach to traffic estimation and the *Mobile Millennium* framework in Section II. We then present in details how a nontrivial distribution of travel times (a Gamma distribution) can be used on the system. We then present the system aspects and the overall design of the system in Section IV. We finally evaluate our implementation in Section V from the perspective of scalability (Section V-B) and accuracy (Section V-C).

II. Scalable Traffic Estimation From Streaming Data

Most GPS data available today is generated at low frequencies due to energy and bandwidth constraints. This data is extremely noisy and provides indirect observations of the travel time distributions on each link of the road network. We introduce here *Mobile Millennium*, a traffic information system that is designed to process such data at low latencies and for very large urban areas. In this section, we will discuss the overall architecture of the arterial estimation pipeline. This pipeline make few assumptions about the actual travel time distribution considered. This lets us define a highly scalable algorithm that is amenable to distribution on cloud computing. In the next section, we will present a particular choice of travel time distribution, along with some algorithms to perform the different steps. It should be noted that the choice of the distribution can be made independently from our framework: most distributions for travel times can be plugged into our framework and yield a very scalable algorithm.

We define the road network as a graph $\mathcal{D} = (\mathcal{V}, \mathcal{E})$, where the set $\mathcal{E}$ will be referred to as the "links" of the road network (streets) and $\mathcal{L}$ as the "nodes" (road intersections). For each link $l \in \mathcal{E}$, the algorithm outputs X_l^t, the time it takes at time index t to traverse link l. This time is described as a probability distribution parametrized by a vector ν_l. Our goal is then to estimate X^t, the joint distribution of all link travel times across all links in $\mathcal{E}$, for each time index t. We assume that the traffic is varying slowly enough that it can be considered a steady state between each evaluation: our algorithm will consider that all the observations between two consecutive time steps have been generated

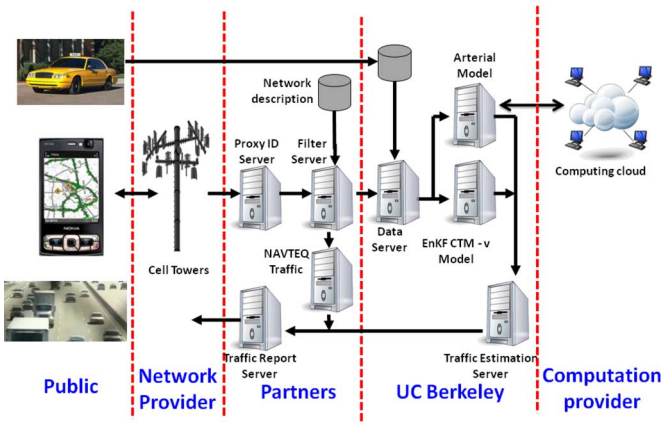


Fig. 1. Schematic architecture of the *Mobile Millennium* system.

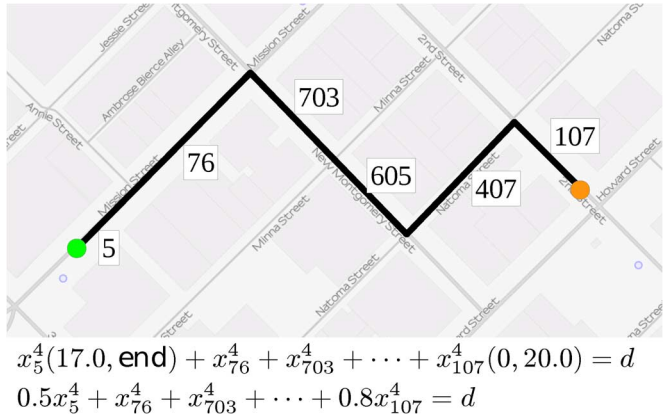


Fig. 2. Example of observation. The green mark represents an initial GPS reading, the orange mark represents a subsequent reading. The black line marks the path of the vehicle, as reconstructed by the PIF between the two GPS points and the numbers are the indexes of each road link covered by this observation. Given a realization x^4 of the travel time distribution at time $t = 4$, all the information on travel times encoded by this observation is summarized in the equation above.

according to the same state. To simplify notations, we will consider a single time interval and drop the reference to time: the joint distribution of travel time is the multidimensional variable X.

We will first give an overview of the GPS data that is commercially available today, and an algorithm that converts raw GPS points to map-matched trajectories with high accuracy: the Path Inference Filter (PIF) [21]. We will then present our modeling approach to infer the traffic conditions from these GPS observations. Then we will explain how the *Mobile Millennium* [23] pipeline implements this algorithm using a computing cloud as a computation backend.

A. Overview of the Mobile Millennium Pipeline

The *Mobile Millennium* system incorporates a complete pipeline for receiving probe data, filtering it, distributing it to estimation engines and displaying it, all in real-time. This software stack, written in Java and Scala, evaluates *probabilistic distribution of travel times* over the road links, and uses as input the *sparse, noisy* GPS measurements from probe vehicles.

The most computation-intensive parts of this pipeline have all been ported to a cloud environment. We briefly describe the operations of the pipeline, pictured in Fig. 1.

The observations are grouped into time intervals and sent to a traffic estimation engine, which runs the learning algorithm described in the next section and returns distributions of travel times for each link (Fig. 3).

The travel time distributions are then stored and broadcast to clients and to a web interface. Examples of means of travel times are shown in Fig. 6.

It is important to point out that *Mobile Millennium* is intended to work at the scale of large metropolitan areas. The road network considered in this work is a real road network (a large portion of San Francisco downtown and of the greater Bay Area, comprising 506,685 road links) and the data is collected from the field (as opposed to simulated). A consequence of this setting is the scalability requirement for the traffic algorithms we employ. Thus, from the outset, our research has focused on designing algorithms that could work for large urban areas with hundreds of thousands of links and millions of observations.

B. Map-Matching GPS Probe Data With the PIF

In order to reduce power consumption and transmission costs, probe vehicles do not continuously report their location to the base station. A high temporal resolution gives access to the complete and precise trajectory of the vehicle, but this causes the device to consume more power and communication bandwidth. Also, such data is not available at large scale today, except in a very fragmented portion of the private sector. A low temporal resolution carries some uncertainty as to which trajectory was followed. In the case of a high temporal resolution (typically, a frequency greater than an observation per second), some highly successful methods have been developed for continuous estimation [15], [27], [32]. However, most data collected at large scale today is generated by commercial fleet vehicles. It is primarily used for tracking the vehicles and usually has a low temporal resolution (1 to 2 min) [3], [10], [24], [31]. In the span of a minute, a vehicle in a city can cover several blocks (see Fig. 2 for an example). Information on the precise path followed by the vehicle is lost. Furthermore, due to GPS localization errors, recovering the location of a vehicle that just sent an observation is a nontrivial task: there are usually several streets that could be compatible with any given GPS observation. Simple deterministic algorithms to reconstruct trajectories fail due to misprojection or shortcuts. The PIF [21] is a probabilistic framework that recovers trajectories and road positions from low-frequency probe data in real time, and in a computationally efficient manner.

This algorithm first projects the raw points onto candidate projections on the road network and then builds candidate trajectories to link these candidate projections. An observation model and a driver model are then combined in a Conditional Random Field to find the most probable trajectories, using the Viterbi algorithm. More precisely, the algorithm performs the following steps.

- We map each point of raw (and possibly noisy) GPS data to a collection of nearby *candidate projections* on the road network [21, Fig. 2–1].

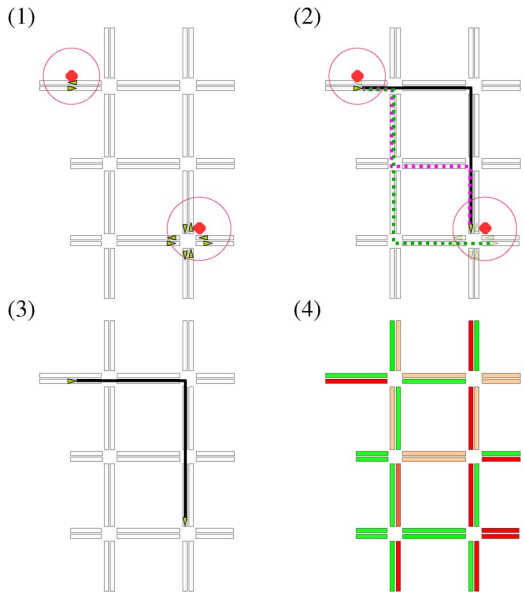


Fig. 3. Map-matching algorithm: the raw GPS readings a first projected onto candidate points on the road network (Step 1). Then, all feasible paths between each pair of candidate points are computed (Step 2). A dynamic programing algorithm then finds the most likely trajectory, using a Conditional Random Field (Step 3). Trajectory measurements are the input to the EM algorithm. This algorithm outputs distributions of travel times (Step 4).

- For each vehicle, we reconstruct the most likely trajectory using a Conditional Random Field [21, Fig. 2–2].
- Each segment of the trajectory between two GPS points is referred as an *trajectory measurement* [21, Fig. 2–3]. A trajectory measurement consists in a start time, an end time and a route on the road network. This route may span multiple road links, and starts and ends at some offset within some links.

At the output of the PIF, we have transformed sequences of GPS readings into sequences of trajectory readings. These readings are the input for our travel time estimation algorithm.

C. Fundamental Generative Model

Estimating the travel time distributions is made difficult by the fact that we do not observe travel times for individual links. Instead, each reading only specifies the total travel time for an entire list of links traveled. We formally describe our estimation task as a maximum likelihood estimation problem.

We consider one reading, described by an offset on a first road link o_{start}, an offset on a last link o_{end}, a list of m visited links $l_1 \cdots l_m$, a start time, and a travel duration d (see Fig. 2 for an example of a reading). The observed travel time is a sum of (unobserved) travel times on the visited links

$$X_{l_1}\left(o_{\text{start}}, L(l)\right) + X_{l_2} + \cdots X_{l_{m-1}} + X_{l_m}(0, o_{\text{end}}) = d.$$

We are going to introduce two assumptions: independence and scaling. The first one is critical to our framework, and is widely used in practice. The scaling assumption is not necessary, but offers some convenience for the development of the discussion. When working with more sophisticated distributions and with enough data, it could easily be dispensed with.

Independence Assumption: To make the inference problem tractable, we model the link travel times for each link l as a univariate distribution with parameter vector $\nu_l = (k_l, \theta_l)$, and we assume these distributions are pairwise independent. The independence assumption is standard in the transportation literature [18], [19] and it also leads to a highly scalable estimation algorithm. We will discuss the validity of this assumption in Section V-C.

Scaling Assumption: The distribution of travel time $X_l(a, b)$ between two offsets a and b of a road link l can be significantly different in shape from the distribution X_l over the full link. We simplify the problem by assuming that the partial travel time from the start of a road link to some offset o is proportional to the distribution over the full link

$$X_{\text{partial}}(o_{\text{start}}, o_{\text{end}}) = f_l(o_{\text{start}}, o_{\text{end}}) X_l$$

where f_l is a function in values between 0 and 1. In our implementation, we make the use of the following function:

$$f\left(o_{\text{start}}, o_{\text{end}}, L(l)\right) = \left(\frac{o_{\text{end}}}{L(l)}\right)^r - \left(\frac{o_{\text{start}}}{L(l)}\right)^r$$

for some $r > 0$. It is well-known that the travel time on a part of a road link is not proportional to the travel time over the complete link [19]. The function f captures some of this nonlinearity. The factor r is selected by cross-validation and was set to be 2.1. In all generality, this assumption may not be representative of empirical data, and it is a convenient way to consider partial travel times without introducing additional parameters to the model. However, in our streaming setting, the updates happen at high frequency (every few second), and there is not enough observation to fully update the distributions, which may lead to some overfitting. As we will see in the next section, this assumption may be dispensed with when more complex traffic distributions are considered.

Using the two assumptions outlined above, the duration d of a travel is the linear combination of realizations of travel times on the different links of the road network

$$\begin{aligned} d &= \sum_{l \in \mathcal{E}} \alpha(l) x_l \\ &= \sum_{l \in p} \alpha(l) x_l \end{aligned}$$

where the vector $\alpha \in [0, 1]^n$ is defined as follows[2]:

$$\begin{aligned} \alpha(l_1) &= f_l\left(o_{\text{start}}, L(l_1)\right) \\ \alpha(l_m) &= f_l(0, o_{\text{end}}) \\ \alpha(l_i) &= 1 \text{ for } i \in [2 \ldots m-1] \end{aligned}$$

and $\alpha(l) = 0$ for all other links l. The vector α is called the *path activation vector* for this reading. Note that fewer than ten links are covered in a typical trajectory measurement, so the path activation vectors are extremely sparse (the fill-in factor is less than 0.001%). We will use this fact to achieve very good scaling of our algorithm.

[2]In practice, the definition of α needs to be adapted when an observation spans a only single link.

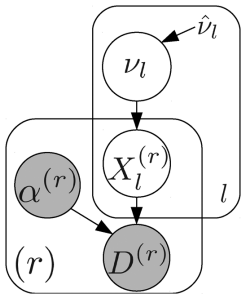


Fig. 4. Directed (Bayesian) graph of the travel time model. Grey nodes are observed variables, white nodes are hidden variables. The arrows represent conditional dependencies between the variables. Boxes encode *plates*, i.e., a factorization of repeating variables. (Right) an expansion of a few elements of the plates.

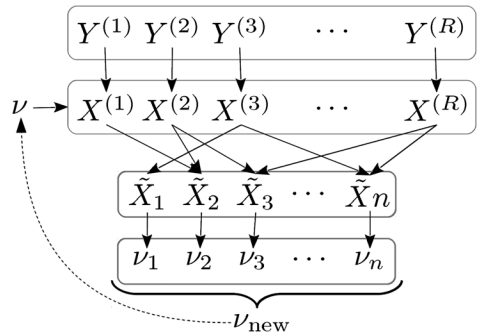


Fig. 5. System workflow of the EM algorithm. In the E-step, we generate per-link travel time samples from whole observations; specifically, for each observation $Y^{(r)} = (\alpha^{(r)}, d^{(r)})$, we produce a set of U weighted samples $\mathbf{X}^{(r)} = \{(x^{(r,u)}, w^{(r,u)})\}_{u=1\cdots U}$, each sample $x^{(r,u)}$ produced by randomly dividing travel time $d^{(r)}$ among its constituent links (producing a travel time $x_{l_i}^{(r,u)}$ for each edge $l_i \in \alpha^{(r)}$). We assign a weight $w^{(r,u)}$ as the likelihood of travel time $x^{(r,u)}$ according to the current distribution parameters ν. In the shuffle step, we regroup the samples $\mathbf{X}^{(r)}$ by link, so that each link l now has samples $\tilde{\mathbf{X}}_l = \{(s_l^{(r,u)}, w_l^{(r,u)})\}_{r,u}$ from all the observations r that go over it. In the M-step, we recompute the parameters ν_l to fit link l's travel time distribution to the samples $\tilde{\mathbf{X}}_l$.

For a given time interval, we can completely represent a trajectory reading by an *observation* $Y = (\alpha, d) \in (\mathbb{R}^+)^n \times \mathbb{R}_+^*$. Each observation $Y^{(r)} = (\alpha^{(r)}, D^{(r)})$ describes the rth trajectory's travel time $D^{(r)}$ and path $\alpha^{(r)}$ as inferred by earlier stages of the *Mobile Millennium* pipeline. The travel time $D^{(r)}$ is the time interval between consecutive GPS observations and is roughly one minute for our source of data.

The dependencies between the observations and the parameter vector ν can be represented as a Bayesian graphical model, which encodes all the dependencies between the variables in a very compact form (Fig. 4). We now formalize the problem of estimating the set of parameters $\nu = (\nu_l)_l$ for a set of observations $(Y^{(r)})_{r=1\cdots R}$ as a learning problem. We consider that the current estimate of the traffic is completely described by independent distributions (parametrized by some vectors ν_l) of the travel times over each road link. These travel times are indirectly observed through a set of observations $Y^{(r)} = (\alpha^{(r)}, d^{(r)})$. The set of parameters that maximizes the likelihood of these observations is solution to the *maximum likelihood problem*

$$\max_{\nu} \quad ll(Y;\nu) = \sum_r \log \pi\left(D^{(r)}|\alpha^{(r)};\nu\right) \tag{1}$$

with $\pi(D^{(r)}|\alpha^{(r)};\nu)$ the probability of observing the duration $d = \sum_l (\alpha(l))x_l$ when x_l is generated according to the distribution $\pi(\cdot;\nu_l)$ of the variable X_l. This likelihood can be decomposed using the relations of independence between variables

$$\begin{aligned}
\pi\left(D^{(r)}|\alpha^{(r)};\nu\right) &= \int_X \pi\left(D^{(r)}|X,\alpha^{(i)}\right)\pi(X;\nu)\mathrm{d}X \\
&= \int_X \pi\left(D^{(r)}|X,\alpha^{(i)}\right) \\
&\quad \times \left(\prod_{l:\alpha^{(r)}(l)>0} \pi(X_l;\nu_l)\mathrm{d}X_l\right) \\
&= \int_X \pi\left(D^{(r)}|X,\alpha^{(i)}\right) \\
&\quad \times \left(\prod_{l:\alpha^{(r)}(l)>0} \pi(X_l;\nu_l)\mathrm{d}X_l\right).
\end{aligned}$$

Estimating the travel time distributions is made difficult by the fact that we do not observe travel times X for individual links. Instead, each observation only specifies the total travel time D for an entire list α of links traveled. To get around this problem, we use the EM algorithm [13], [29]. The EM algorithm operates in two phases: In the E-step it considers each travel time measurement and computes a distribution over allocations of travel time to each of the links. In the M-step it computes the link parameters that maximizes the likelihood of the travel times for the allocations made in the E-step. By iterating this process the EM algorithm converges to a set of link parameters that are a local maximum of the likelihood of the data. In our setting, we run the EM algorithm in an online fashion: for each time step, we use the previous time step as a value, perform a few (iterations and we monitor the convergence through the expected complete log-likelihood. This form of online EM gives good results for our application (Section V).

D. Dataflow of the Algorithm

Fig. 5 shows the data flow in the algorithm in more detail. In the E-step, we generate per-link travel time samples from whole observations; specifically, for each observation $Y^{(r)} = (\alpha^{(r)}, d^{(r)})$, we produce a set of U weighted samples $\mathbf{X}^{(r)} = \{(x^{(r,u)}, w^{(r,u)})\}_{u=1\cdots U}$, each sample $x^{(r,u)}$ produced by randomly dividing travel time $d^{(r)}$ among its constituent links (producing a travel time $x_{l_i}^{(r,u)}$ for each edge $l_i \in \alpha^{(r)}$). We assign a weight $w^{(r,u)}$ as the likelihood of travel time $x^{(r,u)}$ according to the current distribution parameters ν. In the shuffle step, we regroup the samples $\mathbf{X}^{(r)}$ by link, so that each link l now has samples $\tilde{\mathbf{X}}_l = \{(s_l^{(r,u)}, w_l^{(r,u)})\}_{r,u}$ from all the observations r that go over it. In the M-step, we recompute the parameters ν_l to fit link l's travel time distribution to the samples $\tilde{\mathbf{X}}_l$.

So far, we have not introduced a particular distribution for the travel times X_l. This will be the subject of the next section. We require only few operations on a distribution of travel times:

- Sampling under a constraint: The Expectation step involves sampling $X_1 \cdots X_U$ under some constraint $a^T X = d$. We will show in the next section how this operation can be performed efficiently in the case of Gamma distributions. For more complicated distributions, importances sampling may be used. This technique in turn only requires the knowledge of the unnormalized pdf of the distribution, which is usually not a concern.
- solving the maximum-likelihood problem: in the M-step, for each of the links, we solve a problem of the form $\max_\nu \sum \tilde{w}^{(i)} \log \pi(\tilde{x}^{(i)}; \nu)$ given a set of weighted samples $(\tilde{w}^{(i)}, \tilde{x}^{(i)})_i$. This problem can be solved approximately, using gradient descent for example. If the number of parameters is small (typically less than 4), grid search may even yield an appropriate solution in a reasonable time.

As one can see, the minimum requirements for travel time distributions are very mild and may satisfy complex, multi-modal distributions. However, using complex distributions involves learning a large number of parameters and may lead to overfitting when data is scarce. Since our goal is to study the validity of the framework for very large networks and datasets, we will consider in our discussion simple distributions. It will be interesting to compare in the future with some other more realistic distributions.

III. Modeling Travel Times With Gamma Distributions

The Gamma distribution is a simple unimodal, heavy-tailed distribution with a mean and a scale parameter. In this section, we show how it can be used as a travel time distribution and how it fits into our framework.

Gamma distributions substantially complicate the Expectation step, because sums of independent Gamma distributions have no closed form. This is why we approximate the conditional expectation $X^{(r)}|(\alpha^{(r)})^T X^{(r)} = D^{(r)}$ by sampling from this distribution. As we will see, there is a surprisingly simple algorithm in the case of Gamma distributions. While not strictly necessary, it is also useful to compute the value of the marginal likelihood of each observation $\mathbb{P}((\alpha^{(r)})^T X^{(r)} = D^{(r)})$ in order to track the converge of the EM algorithm. Since the computations are independent for each of the observations, they are good candidates for cloud computing.

This section is self contained and does not make use of concepts from the other sections. We introduce it to show how a nontrivial distribution can be incorporated into the rest of the framework, and what computations are necessary to fit in the framework presented is Section III-B. We present the main results in Section III-A. Since the justification of these results requires some measure-theoretic technicalities, the proofs are derived in the subsequent sections. These justifications will require introducing a new statistical distribution: the Gamma-Dirichlet distribution.

A. Learning With Gamma Distributions

Consider a set of n independent Gamma distributions $T_i \sim \Gamma(k_i, \theta_i)$ with $k \in (\mathbb{R}_+^*)^n$ and $\theta \in (\mathbb{R}_+^*)^n$, a n-dimensional vector of positive numbers $\alpha \in (\mathbb{R}_+^*)^n$ and $d > 0$.[3] Call T the joint distribution of all T_is. The purpose of this paragraph is to present some practical formulas to sample and compute the density function of the sum $\sum_i \alpha_i T_i$, which is a univariate distribution.

Marginal Likelihood: Call $U = \sum_i \alpha_i T_i$. Call $\underline{\theta} = \min_i \alpha_i \theta_i$ and $\underline{k} = \sum_i k_i$. The probability density function of U is an infinite series [8], [28]

$$f_U(d) = \underline{\theta}^{\underline{k}} \prod_i (\alpha_i \theta_i)^{-\underline{k}} \sum_{l=0}^{\infty} \delta_l f_\Gamma(d; \underline{k} + l, \underline{\theta})$$

in which f_Γ is the density function of the Gamma distribution $f_\Gamma(x; a, b) = \Gamma(a)^{-1} b^{-a} x^{a-1} e^{-b^{-1}x}$ and $(\delta_j)_j$ a series defined by the recursive formula

$$\begin{cases} \delta_0 = 1 \\ \delta_l = \frac{1}{l} \sum_{m=0}^{l} \delta_l \left(\sum_{i=1}^{n} k_i \left(1 - \alpha_i \theta_i^{-1} \underline{\theta}\right)^{l-m} \right). \end{cases}$$

This result is a direct application of [28], using the scaling property of the Gamma distribution: $\alpha_i T_i \sim \Gamma(k_i, \alpha_i \theta_i)$ for $\alpha_i > 0$.

Sampling From Conditional Gamma Distributions:

Algorithm 1: Sampler for Gamma distributions conditioned on a hyperplane

Given $\alpha \in (\mathbb{R}_+^*)^n$ and $d > 0$.

Generate n independent samples $a_i \sim \Gamma(k_i, d^{-1}\alpha_i \theta_i)$

$z_i = d\alpha_i^{-1}(a_i / \sum_k a_k)$

Then $z \sim T|\alpha^T T = d$

Our algorithm must provide values from the conditional distribution $Z \sim T|\alpha^T T = d$. While this distribution has a complex shape (in particular, it is defined over a zero-measure hyperplane of the space of variable), there happens to exist a remarkably simple procedure to sample values from the conditional Gamma distribution Z: sample n independent values from Gamma distributions

$$A_i \sim \Gamma\left(k_i, \frac{\alpha_i \theta_i}{d}\right). \tag{2}$$

Then, a suitably rescaled value of A_i follows the distribution of Z:

$$Z_i = \frac{d}{\alpha_i} \frac{A_i}{\sum_l A_l}. \tag{3}$$

To our knowledge, this is a new result regarding Gamma distributions. We present the proof of correctness in the next section. The proof requires some technical arguments that may be skipped in a first reading. The algorithm is presented in Algorithm 1.

B. The Gamma-Dirichlet Distribution

In order to show that (2) and (3) give the correct distribution for Z, we formally introduce a generalization of the Dirichlet

[3]The bivariate function $\Gamma(\cdot, \cdot)$ will refer to the Gamma distribution and the univariate function $\Gamma(\cdot)$ will refer to the Gamma function. Which one is used should be clear from the context.

distribution, which we call the *Gamma-Dirichlet distribution*.[4] This distribution can be sampled using a closed for solution. We show in a second step that its pdf is the same as the pdf of the conditional distribution $T|\sum_i T_i = 1$. We then generalize to arbitrary positive combinations.

The Gamma-Dirichlet Distribution: The regular simplex $\mathcal{S}^n \subset \mathbb{R}^n$ is the convex hull of the elementary vertices $(\mathbf{e}_i)_{i\in[1,n]}$. Given a vector $k \in (\mathbb{R}^+)^n$ and a vector $\theta \in (\mathbb{R}^+)^n$, we define the *Gamma-Dirichlet distribution*

$$X \sim \Gamma D(k, \theta)$$

with values over the regular simplex $\mathcal{S}^n$ as the normalized sum of n elements drawn from independent Gamma distributions

$$X_i = \frac{Y_i}{\sum_j Y_j}$$

with

$$Y_i \sim \Gamma(k_i, \theta_i) \tag{4}$$

and Y_i all pairwise independent. The Gamma-Dirichlet distribution is a simple of the Dirichlet distribution: if $\theta = a\mathbf{1}$ for some $a > 0$, this is the Dirichlet distribution of the nth order. The definition gives a straightforward procedure to sample some values from X.

Density: Note first that one needs to be careful in defining the underlying σ-algebra of our probability space, as the values of X are located in an embedding of $\mathbb{R}^n$ of measure 0 (a hyperplane). Consider the n-dimensional hyperplane $\mathcal{H}^n = \{x|x^T\mathbf{1} = 1\}$. This hyperplane includes the simplex $\mathcal{S}^n$. The Lebesgue measure of this set in $\mathbb{R}^n$ is zero. However, we can consider the Lebesgue measure $\tilde{\mu}$ defined over $\mathbb{R}^{n-1}$ and the transform: $\phi : \mathbb{R}^{n-1} \to \mathcal{H}^n$ defined by $\phi(u) = (u - \mathbf{1}^T u)^T$. This transform is a linear mapping and it defines a new measure $\hat{\mu}$ for the space $\mathcal{H}^n$ based on $\tilde{\mu}$. Under this measure, the measure of the simplex $\mathcal{S}^n$ is positive. Call μ the measure defined over $\mathcal{S}^n$ by $\mu(\cdot) = \hat{\mu}(\mathcal{S}^n)^{-1}\hat{\mu}(\cdot)$. Consider the conditional distribution $Z_i = Y_i|\sum_j Y_j = 1$ with Y_i defined in (4). The density function of this distribution is 0 over $\mathbb{R}^n$ nearly everywhere, however, it has nonzero measure over the regular simplex $\mathcal{S}^n$.

Call $f_{\Gamma D}(x)$ the pdf of the Gamma-Dirichlet distribution over $\mathcal{S}^n$. Then, we have

$$f_{\Gamma D}(x) \propto \prod_{i=1}^{n} f_\Gamma(x_i; k_i, \theta_i)$$

with $f_\Gamma(x; k, \theta)$ defined above.

a) Proof: This proof is adapted from a similar proof [14] for the Dirichlet distribution. Using the same notations as above, define $Y = \sum_j Y_j$ and $X_i = Y_i/Y$ for $i \leq n$. The joint density for the Y's is

$$f(y) \propto \prod_i y_i^{k_i-1} e^{-\sum \theta_i^{-1} y_i}.$$

[4]To our knowledge, the following results have not been presented in the literature so far.

Define the transform $\tilde{\varphi} : \mathbb{R}^n \to \mathbb{R}^n$ by: $\tilde{y} = \sum_i \theta_i^{-1} y_i$ and $\tilde{x}_i = \tilde{y}^{-1}\theta_i^{-1} y_i$ for $i \leq n-1$. We are going to show that the joint density of $\tilde{x}$ and $\tilde{y}$ can be written as $g(\tilde{x}, \tilde{y}) = a(\tilde{x})b(\tilde{y})$, which implies that the variables $\tilde{x}$ and $\tilde{y}$ are independent. This mapping is invertible and its Jacobian at $\tilde{y}$ is $(\prod_i \theta_i^{-1})\tilde{y}$. Thus, the joint density of $(\tilde{y}, \tilde{x})$ is

$$g(\tilde{y}, \tilde{x}) \propto \left[\left(1 - \sum_{i=1}^{n-1} \tilde{x}_i\right)^{k_n - 1} \prod_{i=1}^{n-1} \tilde{x}_i^{k_i - 1}\right] \left[\tilde{y}^{\sum_i k_i - 1} e^{-\tilde{y}}\right].$$

This shows that the variables $\tilde{y}$ and $\tilde{x}$ are independent. Furthermore, from the expression above, the distribution of $\tilde{x}$ is a Dirichlet distribution and the distribution of $\tilde{y}$ is a Gamma distribution. The marginal for $\tilde{x}$ writes

$$g(\tilde{x}) \propto \left(1 - \sum_{i=1}^{n-1} \tilde{x}_i\right)^{k_n - 1} \prod_{i=1}^{n-1} \tilde{x}_i^{k_i - 1}.$$

Now, we consider a change of variables for the joint variables y to remove the conditional constraint $\sum_i y_i = 1$. Define the transform $\varphi : \hat{y} = \sum_i y_i$ and $\hat{x}_i = \hat{y}^{-1} y_i$ for $k \leq n-1$. This mapping is also invertible, with Jacobin $\hat{y}^k$. The joint density of $(\hat{y}, \hat{x})$ is

$$g(\hat{y}, \hat{x}) \propto \left[\left(1 - \sum_{i=1}^{n-1} \hat{x}_i\right)^{k_n - 1} \prod_{i=1}^{n-1} \hat{x}_i^{k_i - 1}\right] \left[\hat{y}^{\sum_i k_i - 1} e^{-\theta_n \hat{y}}\right].$$

By identification, we get: $g(x|y = 1) = g(\Delta^{-1}\tilde{x})$ with Δ the diagonal matrix defined by $\Delta_{ii} = \theta_i$. Since $\tilde{\varphi}^{-1}((\Delta^{-1}\tilde{x}, 1)^T) = \tilde{x}$, the result ensues. ∎

The previous result proves that the sampling procedure in Algorithm I is correct for the simplex $\mathcal{S}^n$ (i.e. $d = 1$). We can then use the scaling transform of the Gamma distribution to show it is also correct for other values of d. Indeed, the constraint $\alpha^T T = d$ is also equivalent to $\sum Y_i = 1$ with $Y_i = d^{-1}\alpha_i T_i \sim \Gamma(k_i, d^{-1}\alpha_i\theta_i)$. We can perform the conditional sampling on the variables Y_i, and then rescale the values obtained to get the correct distribution.

b) Proof: Consider a set of n independent Gamma distributions $T_i \sim \Gamma(k_i, \theta_i)$, a n-dimensional vector of positive numbers $\alpha \in (\mathbb{R}_+^*)^n$ and $t > 0$. The purpose of this section is to present some practical formulas to sample and compute the density function of the conditional distribution

$$Z = T|\sum_i \alpha_i T_i = d$$

We define this distribution over the n-dimensional simplex

$$\mathcal{S}_{\alpha,d} = \left\{x \in (\mathbb{R}^+)|\alpha^T x = d\right\}.$$

As before, we define a new measure over the hyperplane defined by $\alpha^T x = d$ by an isomorphism from $\mathbb{R}^{n-1}$, and use it as our base measure $\mathrm{d}z$ for Z. We call f the probability density function of variable Z with respect to this measure. With respect to this measure, the probability density function of Z is that of a Gamma-Dirichlet distribution

$$f(z) \propto f_{\Gamma D}(y; k, \hat{\theta}).$$

with

$$\hat{\theta}_i = t^{-1}\alpha_i\theta_i$$

■

IV. Discretized Streams: Large-Scale Real-Time Processing of Data Streams

We now describe the design of the data processing pipeline of the *Mobile Millennium* system. This framework takes as an input streams of raw GPS data from various sources, and outputs states of traffic, in the form of parameters of travel time distributions. We had the followivsk —12ng requirements, which are shared with most automation system.

- *Low latency*: We wanted to investigate update rates as high as every few seconds.
- *Scalability and high throughput*: The system should be able to handle tens or hundreds of thousands of measurements per second
- *Robustness to failures*: The failure of machines should have a limited impact on the performance of the system
- *Testing multiple scenarios*: The GPS data needs to go through some complex filtering process (map-matching and trajectory reconstruction) before it can be used for estimation, and the ability to perform cross-validation is a prerequisite to tuning these algorithms. Thus, we needed to rerun or even run multiple instances of our algorithm in parallel.
- *Flexible deployment solution*: From the outset, our goal was to deploy our framework on a generic cloud solution such as Amazon EC2. This implies limited control over the topology of the computer network and over the characteristics of the computers.

Streams of hundreds of thousands of elements per second are common in cloud-based environment. However, in our application, the Expectation step of the EM algorithm generates a large number of samples for each observation. This multiplies the internal throughput rate by a factor of 1000 to 10,000. Single computers cannot work at this rate, and a cloud-based solution is required. Furthermore, these samples are ephemeral and can be deterministically recreated from an observation: there is no need to store them. The novelty of our application lies in combining the competing requirements of the different stages: the initial and final steps have average throughput and their respective inputs and outputs require fault-tolerant storage, while the intermediate computation steps must have high throughput and be resilient to individual node failures.

In particular, since this is a research platform, we needed the ability to test and monitor complex iterative algorithms. In this section, we will present the notion of Discretized Streams, a concept recently introduced [35] and implemented in the Spark computing framework. We will present how the design of D-Stream lets users build cyberphysical systems at scale.

A. Limitations of Current Techniques

Current techniques to process large amounts of live streaming data can be broadly classified into the following two categories.

- *Using traditional streaming processing systems*: Streaming databases like StreamBase [6] and Telegraph [11], and stream processing systems like Storm [30] have been used to meet such processing requirements. While they do achieve low latencies, they either have limited fault-tolerance properties (data lost on machine failure) or limited scalability (cannot be run on large clusters).
- *Using traditional batch processing systems*: The live data is stored reliably in a replicated file system like HDFS [1] and later processed in large batches (minutes to hours) using traditional batch processing frameworks like Hadoop [1]. By design, these systems can process large volumes of data on large clusters in a fault-tolerant manner, but they can only achieve latencies of minutes at best. Furthermore, the processing model is too low level to conveniently express complex stream computations.

B. D-Streams—A Programming Model for Stream Processing

D-Streams execute deterministic computations similar to those in MapReduce for fault tolerance, but they do so at a much lower latency than previous systems, by keeping state *in memory*, as opposed to saving it on disk between each step. The input data received from various input sources (e.g., web-services, sensors, etc.) during each interval is stored reliably across the cluster to form an input dataset for that interval. Once the time interval completes, this dataset is processed via deterministic parallel operations (like mapping transformations or filtering) to produce new datasets representing program outputs or intermediate states. Finally, these datasets can be saved to external source such as databases. The advantage of this model is that it provides the developer a convenient high-level programming model to easily express complex stream computations while allowing the underlying system to process the data in small batches thus achieving excellent fault-tolerance properties. D-Streams support the same stateless transformations available in typical batch frameworks, including map, reduce, groupBy, and join. In addition, D-Streams also provide stateful operators like windowing and moving average operators that share data across time intervals. All the intermediate data computed using D-Streams are by design fault-tolerant, that is, no data is lost if any machine fails. This is achieved by treating each batch of data (and each new batch derived through transforms of the original dataset) as a dataset distributed across the machines. Each dataset maintains a *lineage* of operations that was used to create it from the raw input data (stored reliably by the system by automatic replication) [36]. Hence, in the event of computer failure, if any partition is lost, it can be recomputed from raw input data using the lineage. As these operations are deterministic, the recomputation can be done using fine-grained tasks in parallel. This ensures fast recovery minimizing the effect of the failure on the stream processing system. This novel technique is called *parallel-recovery* and sets this abstraction apart from existing stream processing systems, that need to write intermediate steps to disk or implement complex recovery mechanisms.

To implement D-Stream, we use Spark, an existing open-source, batch processing framework, to create Spark Streaming. Spark is a fast, in-memory batch processing framework, and we naturally extend this framework to implement D-Streams. Both these systems are implemented in Scala [4] (a language based on the Java Virtual Machine), which allows them integrate well with existing Java and Scala libraries for linear algebra, machine

learning, etc. Furthermore, the compact syntax of the Scala language hides all the complexities of distribution, replication and data access pattern behind an intuitive programming interface. A relevant portion of the code of the algorithm is provided in the Appendix. This code instantiates a D-Stream with the raw data and derives some other D-Streams that correspond to each step of the algorithm. As can be seen, this code leverages the functional API of Spark and Scala to express stream transformations in a very natural way. Spark Streaming can scale to hundreds of cores while achieving latencies as low as hundreds of milliseconds. We use this system to implement our traffic estimation algorithms, which we shall explain next.

C. Lessons Learned

In the process of designing this framework, we experienced several challenges and we made several design decisions that we feel should be taken into consideration when considering building a large system for cyberphysical computations.

Computing in memory: Keeping the computations in memory leads to dramatic performance improvements, as the processes do not need to seek or write data on the disk. We found that this improves both the throughput and the latency. The fault recovery is provided by the lineage information, and is typically fast enough (a few seconds) for our experiments.

Interfacing with databases: One of the bottlenecks that came to us as a surprise is the interaction with our primary storage (an Oracle database). After map-matching, the data is written to the database. The nodes of the computing cloud then open a connection periodically to read the new data at every beat. However, the database could not keep up with hundreds of queries at the same time. Replicating the database in the cluster did not lead to much improvement. Our solution was to use the database as a end point that would not be queried by the cloud computers: at the end of the map-matching step, the stream of map-matched observations is not only pushed to the database, but also to the Hadoop filesystem (HDFS) that is spread on the cluster. We found the insertion mechanism to be fast enough for this pattern.

Using immutable, stateless transforms: Working with immutable distributed datasets called for a different programming style that emphasizes function-based transforms of data. In this style, a filter would be implemented as a function that takes a state and some observations, and creates a new object that contains the updated state: the original state object is not modified. We found this style to be a significant departure from the common practices in control and automation. Thus, a stream is defined as a sequence of transforms on a dataset. The Scala programming language provides an elegant interface for expressing these transformations (see Annex). The immutability is a key factor in performing operations in memory and in a fault-tolerant way. We found that in practice, we did not experience performance bottleneck from using this programming style.

V. A Case Study: Taxis in the San Francisco Bay Area

Having now described an algorithm for computing travel time distributions in real time on a road network, we describe our validation experiments. These experiments explore two settings.

- The raw performance of the machine learning algorithm, given a limited amount of data and a computational budget,
- The performance of the Streaming Spark framework in distributing computations across a cluster, and the computational performance improvement gained by additional hardware.

The performance of the algorithm is computed by asking the model to give travel time distributions on unseen trajectories, slightly in the future. The observed travel time of the trajectory is then compared with the distribution provided by the model. We measured the L1 and L2 losses between the observed travel time and the distribution mean, and the likelihood of the observed travel time with respect to the predicted travel time distribution. This is done with different amount of data and different time horizons.

The computational efficiency of the algorithm is validated in two steps. First, we demonstrate that our algorithm scales well: given twice as many computation nodes, it perform the same task about twice as fast. We also see that this algorithm is bounded by computations. Then, we demonstrate that it can sustain massive data flow rates under strict scheduling constraints: we fix a completion time of a few seconds for each time step, and we find the maximum flow rate under a given computational budget.

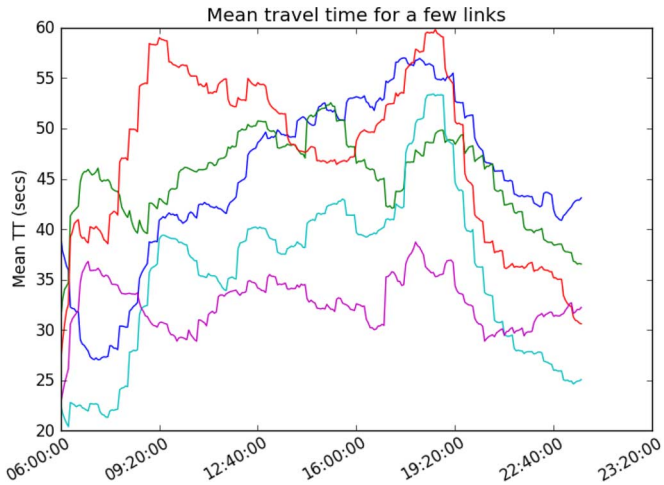


Fig. 6. An example output of traffic estimates from our algorithm: mean travel times on different road links in the business district of San Francisco during a complete day.

A. Taxis in San Francisco

Our implementation was run on a road network that corresponds to the greater San Francisco Bay Area (506,685 road links), using some taxi data provided by the Cabspotting project [10]. This dataset contains GPS samples of a few thousand taxicabs emitted every minute, for more than a year. All in all, it represents hundred of millions of GPS points. We ran our algorithm on a typical day (August 12, 2010, a Tuesday) with different settings. An example of input data is given in Fig. 7. A typical output of travel times provided by the algorithm is given in Fig. 6.

B. Good Scalability Results Using a Large Cluster

In this section, we evaluate how much the cloud implementation helped with scaling the *Mobile Millennium* EM traffic

Fig. 7. An example of dataset available to *Mobile Millennium* and processed by the PIF: taxicabs in San Francisco from the Cabspotting program. Large circles in red show the position of the taxis at a given time and small dots (in black) show past positions (during the last 5 h) of the fleet. The position of each vehicle is observed every minute.

estimation algorithm. Distributing the computation across machines provides a twofold advantage: each machine can perform computations in parallel, and the overall amount of memory available is much greater.

Scaling: First, we evaluated how the runtime performance of the EM job could improve with an increasing number of nodes/cores. The job was to learn some historical traffic estimate for San Francisco downtown for a half-hour time-slice, using a large portion of the data split in one-hour intervals. This data included 259,215 observed trajectories, and the network consisted of 15,398 road links. We ran the experiment on two cloud platforms: the first was using Amazon EC2 `m1.large` instances with 2 cores per node, and the second was a cloud managed by the *National Energy Research Scientific Computing Center* (NERSC) with 4 cores per node. Fig. 8 (bottom) shows near-linear scaling on EC2 until 80–160 cores. Fig. 8 (top) shows near-linear scaling for all the NERSC experiments. The limiting factor for EC2 seems to have been network performance. In particular, some tasks were lost due to repeated connection timeouts.

Scaling on Streaming Spark: After having found the bottlenecks in the Spark program, we wrote another version in Streaming Spark. The two programs are strikingly similar (see program listing in Appendix B). We then benchmarked the application. We ported this application to Spark Streaming using an online version of the EM algorithm that merges in new data every five seconds. The implementation was about 200 lines of Spark Streaming code, and wrapped the existing map and reduce functions in the offline program. In addition, we found that only using the real-time data could cause overfitting, because the data received in 5 s is so sparse. We took advantage of D-Streams to also combine this data with historical data from the same time during the past ten days to resolve this problem. Fig. 9 shows the performance of the algorithm on up to 80 quad-core EC2 nodes. The algorithm scales almost perfectly, largely because it

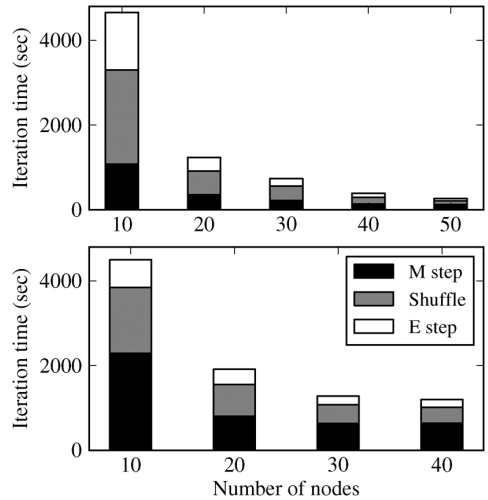


Fig. 8. Experiments with Spark to build historical estimates of traffic, on NERSC (top) and Amazon EC2 (bottom).

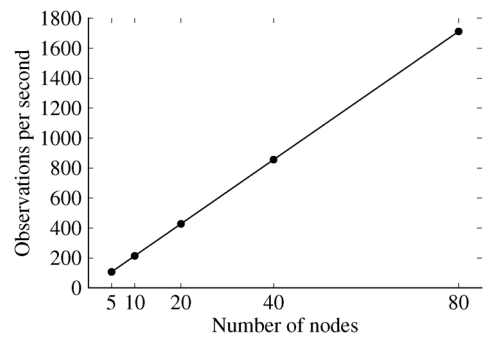


Fig. 9. Experiments with Streaming Spark: rate of observation processing for different cluster sizes.

is so CPU-bound, and provides answers an order of magnitude faster than the previous implementation.

C. Our Algorithm Can be Adjusted for Tradeoffs Between Amount of Data, Computational Resources, and Quality of the Output

We now study the accuracy of our algorithm in estimating the traffic. Even if we receive a large number of observations per day, this number is not sufficient to cover properly in real time all the road network: indeed, some sections of the road network are much less traveled than the busy downtown areas. We use several strategies to mitigate this spatial discrepancy.

- We use a prior on the Gamma distribution, itself a Gamma distribution since the Gamma is in the exponential family and conjugate with itself. The parameters of this prior are to 70% of the speed limit in mean and 1 min or 50% of the travel time in standard deviation, whichever is greater.
- We incorporate some data from the same day before the current time step, weighted by an exponential decay scheme: the traffic in the arterial network is assumed to change slowly enough.
- We also incorporate some data from previous days, corresponding to the same day of the week (Monday, Tuesday, etc.). Traffic is expected to follow a weekly pattern during the same month.

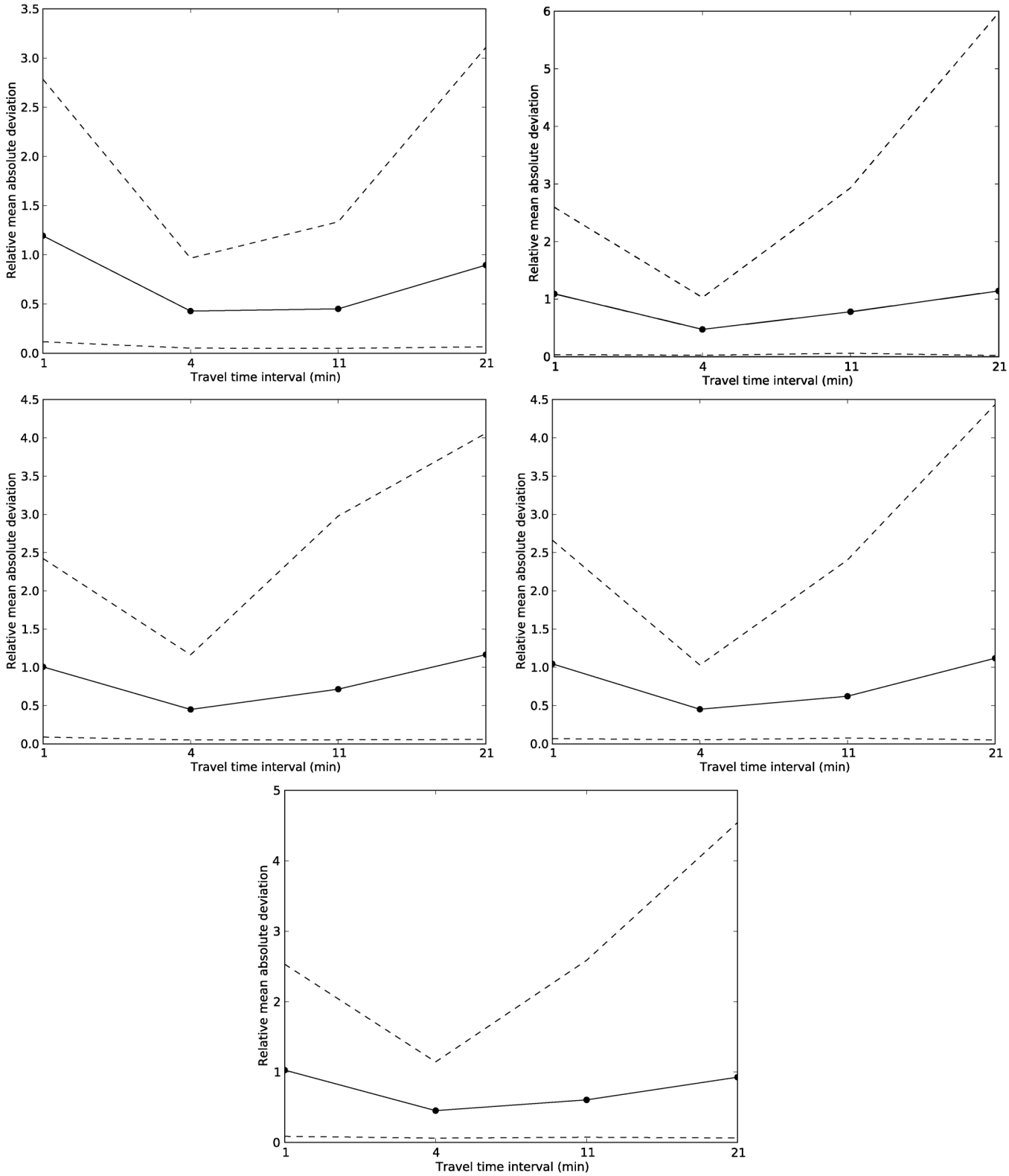


Fig. 10. L1 residuals for different settings and for different travel times. The dashed lines indicate the 95% confidence interval. On the x axis, the travel time of the observations considered for this metric.

To summarize, a large number of observations are lumped together and weighted according to the formula

$$w = e^{-\Delta t_{\text{day}}^{-1}(t_{\text{obs}} - t_{\text{current}})} e^{-\Delta t_{\text{week}}^{-1}(week_{\text{obs}} - week_{\text{current}})}.$$

The half-time decaying factors Δt_{day} and Δt_{week} are set so that the corresponding weight is 0.2 at the end of the window.

Since the EM learning algorithm is not linear in the observations, we cannot reduce each observation to some sufficient

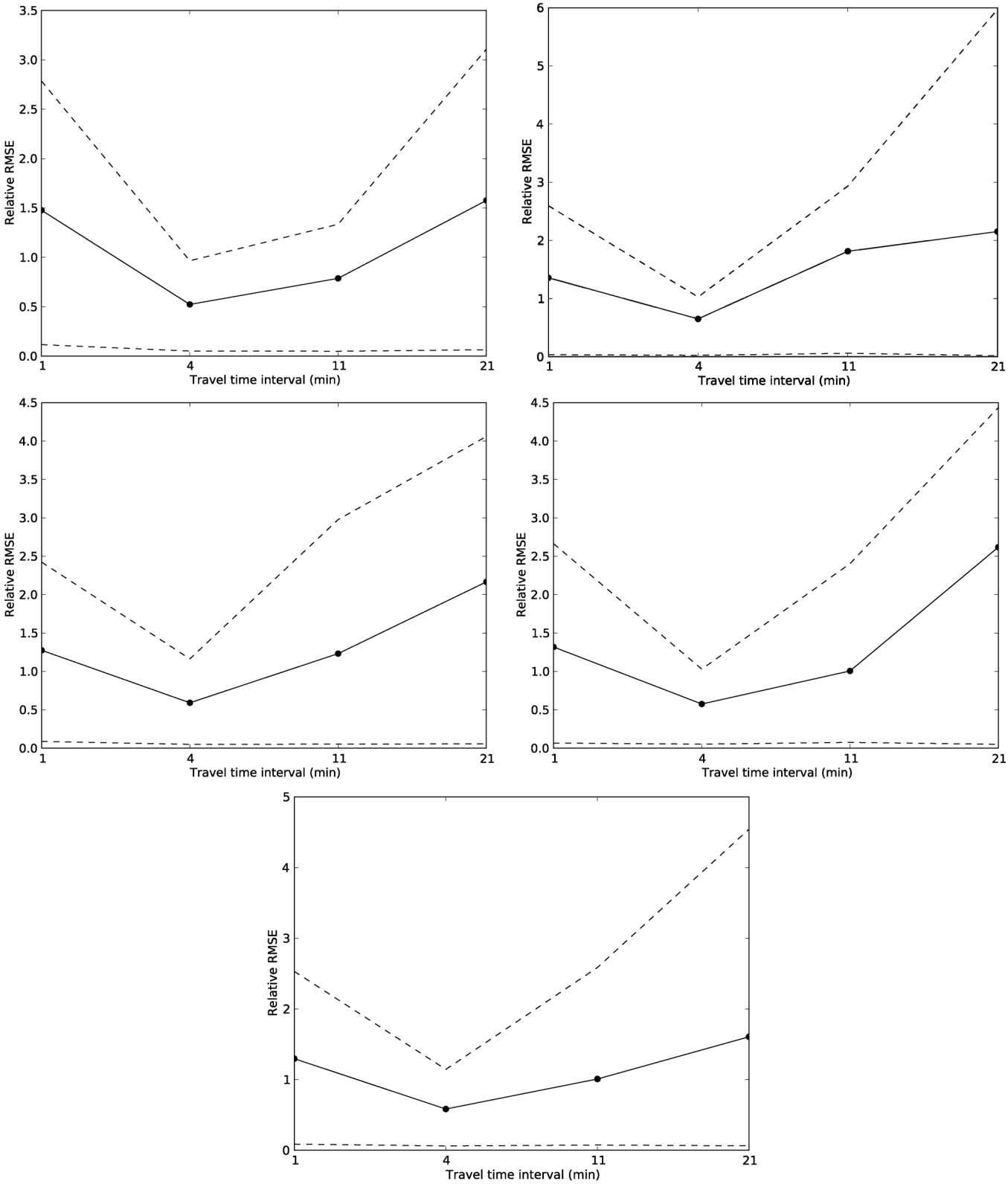


Fig. 11. L2 residuals for different settings and for different travel times. The dashed lines indicate the 95% confidence interval.

statistics. As the algorithm moves forward in time, each observation will appear at different time steps with a different weight and needs to be reprocessed. This is a significant limitation from this approach, but it makes for a good testing ground of Streaming Spark.

Our EM algorithm can be adjusted in several ways.

- The number of weeks of data to look back (between 1 and 10).
- The time window to consider before the current observation (between 20 min and 2 h).

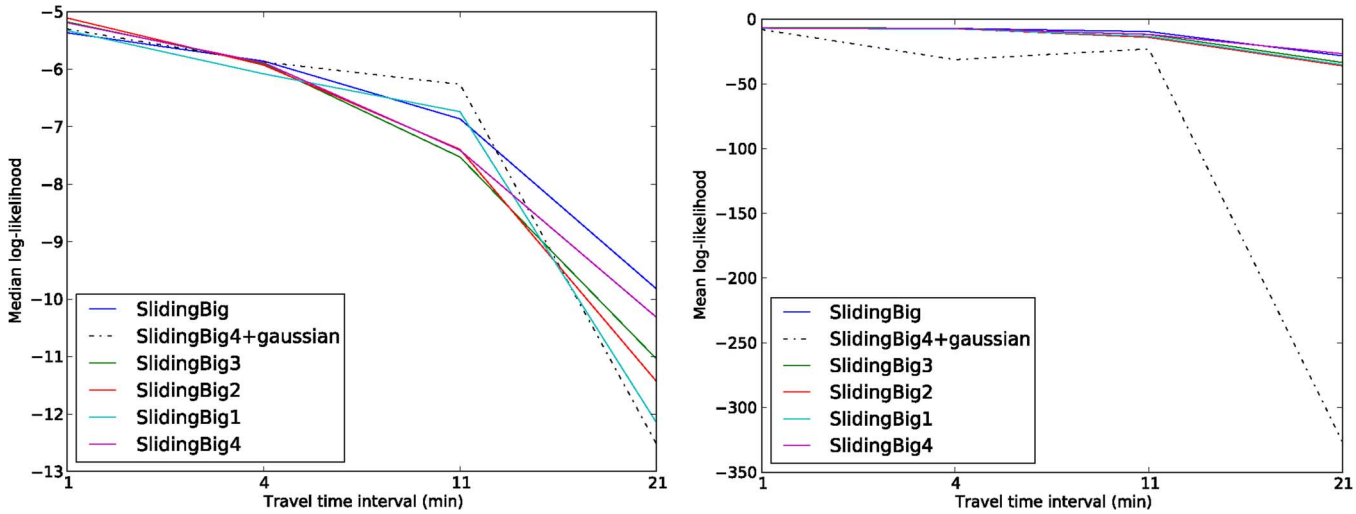


Fig. 12. Log-likelihood of unobserved trajectories, for different trajectory lengths and different settings. The median is presented on the left, while the mean is presented on the right.

- The number of samples generated during the E-steps (10–100).
- The number of EM iterations (1–5).
- The duration of each time step (5 s–15 min)

The observations we process all have a duration of one minute, but travel times experienced by users are usually much longer (10 min to a few hours). As such, a good metric for assessing the quality of a model should not be on predicting travel times for 1-min observations, but on longer distances. Hour-long travels are very likely to go be spent mostly on highways, which is not the scope of this study, and taxicabs usually make small trips (10–30 min). This is why we focus our attention to travel times between one minute (the observations) and 30 min (typical durations for taxi rides). As far as we know, this study of different durations is seldom done in the study of traffic, which limits any attempt to compare the performance between different algorithms.

The longer trajectories are obtained from the PIF. They are then cuts into different pieces of the same length (1 min, 5 min, 10 min, 20 min). Each piece of trajectory is considered as an independent piece of trajectory for the purpose of travel time prediction.

We ran the algorithm with four different settings.

- SlidingBig: the most expensive setting (10 weeks of data, 2 h of data, 100 EM samples, 5 EM iterations, 15 time steps), used as the baseline for comparison. Travel time estimates are produced every 20 min.
- SlidingBig: uses less data (40 min of data).
- SlidingBig2: uses less data (10 days).
- SlidingBig3: uses the same amount of data, but performs only a single EM iteration every 4 min instead of 5 EM iterations every 20 min.
- SlidingBig4: uses the same amount of data, but generates only ten EM samples for each observation.

For all these experiments, the prior was fixed.

We now compare the results obtained with the different experiments. We first turn our attention to the L1 loss in Fig. 10. As expected, the best performance is obtained for experiment SlidingBig, which uses the most data. Interestingly enough, the best performance is obtained for travels of medium length (4–11 min), and not for short trajectories. This can be explained by the conversion step that transforms trajectory readings on partial links into weighted observations on complete links. The relation between link travel time and location on a link is more complex than a linear weighting. Nevertheless, the model gives relatively good performance by this simple transform. When a vehicle is stopped at a red light, it does not travel along the link but still has a nonzero travel time. In this case, the weight of an observation is taken to be half of the total travel time of the link. In particular, the relative error increases as the duration (and the length) of travels increases. Performance is not too different between experiments, which suggests some even smaller amount of data could be considered.

The results for the L2 loss, presented in Fig. 11, provide some similar, if more acute, results. The RMSE is lowest for small to medium travels (in the range of 3–10 min).

A probabilistic metric (the log-likelihood) gives a different insight, as shown in Fig. 12. The model best explains the data for very short travel times (similar to what is was trained on) but its precision falls down as the length of trajectories increases. All in all, this results should not be unexpected: this model with independent links cannot take into account the correlations that occur due to light synchronizations or drivers' behavior. As such, the probability density of a longer travel rapidly dilutes as the number of links increases As we saw with our study of L1 and L2 errors, the mean travel time becomes the only significant value of interest for longer travel times. In the light of this result, there seems to be little to gain by modeling travel time with physically realistic, link-based, independent distributions, as the independence assumption will strongly weigh on the quality of the travel time for longer travels. Instead, we recommend focusing effort on simpler models of travel times that take into account the correlations between links. We also present in Fig. 12 (right) the mean of the log-likelihood. As one can see

in this plot, the Gaussian distribution is significantly worse than the Gamma distribution: from inspection of the data, we could conclude that it performs very badly with long tail observations (unusually long stops at red lights). Furthermore, its symmetric nature causes a significant portion of the probabilistic mass to be assigned to negative travel times: a Gaussian distribution cannot at the same time have a small mean, a high standard deviation and mostly positive values. This experiment should serve as another confirmation that modelling travel time noise with a normal distribution is not only unrealistic, but also leads to erratic results in the face of real data.

VI. Conclusion

As datasets grow in size, some new strategies are required to perform meaningful computations in a short amount of time. We explored the implementation of a large-scale state estimation in near-real-time using D-Streams, a recently proposed streaming technique. Our traffic algorithm is an EM algorithm that computes travel time distributions of traffic by incremental online updates. This approach was validated with a large dataset of GPS traces. This algorithm seems to compare favorably with the state of the art and shows some attractive features from an implementation perspective. When distributed on a cluster, this algorithm scales to very large road networks (half a million road links, tens of thousands of observations per second) and can update traffic state in a few seconds.

In order to foster research in systems and in traffic, the authors have released the code of Spark Streaming [5], the code of the EM traffic algorithm [7], and the dataset used for these experiments [7].

Acknowledgment

bThe authors would like to thank S. Blandin, J. Reilly, and S. Samarayanake for helpful discussions. They would also like to thank Ghada Elabed for her insightful comments on the draft.

References

[1] Apache Hadoop [Online]. Available: http://hadoop.apache.org
[2] The Mobile Millennium Project [Online]. Available: http://traffic.berkeley.edu
[3] Navteq [Online]. Available: http://navteq.com
[4] Scala Programming Language [Online]. Available: http://scala-lang.org
[5] Spark Project [Online]. Available: http://spark-project.org/
[6] StreamBase [Online]. Available: http://www.streambase.com
[7] The Berkeley Open Traffic Stack (BOTS), Streaming Arterial Module [Online]. Available: http://github.com/calpath/bots-arterial-streaming/
[8] M. S. Alouini, A. Abdi, and M. Kaveh, "Sum of gamma variates and performance of wireless communication systems over Nakagami-fading channels," *IEEE Trans. Veh. Technol.*, vol. 50, no. 6, pp. 1471–1480, Nov. 2001.
[9] X. Ban, R. Herring, J. Margulici, and A. Bayen, "Optimal sensor placement for freeway travel time estimation," in *Proc. 18th Int. Symp. Transportation and Traffic Theory*, Jul. 2009.
[10] The Cabspotting Program [Online]. Available: http://cabspotting.org/
[11] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. R. Madden, V. Raman, F. Reiss, and M. Shah, "TelegraphCQ: Continuous dataflow processing," in *Proc. ACM SIGMOND Int. Conf. Manage. Data*, 2003, p. 668.
[12] C. G. Chorus, E. J. E. Molin, and B. Van Wee, "Use and effects of advanced traveller information services (ATIS): A review of the literature," *Transport Rev.*, vol. 26, no. 2, pp. 127–149, 2006.
[13] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *J. Royal Stat. Soci. Series B (Methodological)*, pp. 1–38, 1977.
[14] L. Devroye and L. Devroye, *Non-Uniform Random Variate Generation*. New York, NY, USA: Springer-Verlag, 1986, vol. 4.
[15] J. Du, J. Masters, and M. Barth, "Lane-level positioning for in-vehicle navigation and automated vehicle location (AVL) systems," in *Proc. IEEE 7th Int. Conf. Intell. Transport. Syst.*, 2004, pp. 35–40.
[16] S. Gao, E. Frejinger, and M. Ben-Akiva, "Cognitive cost in route choice with real-time information: An exploratory analysis," *Procedia-Social and Behav. Sci.*, vol. 17, pp. 136–149, 2011.
[17] B. R. Hellinga and L. Fu, "Reducing bias in probe-based arterial link travel time estimates," *Transport. Res. Part C: Emerging Technologies*, vol. 10, no. 4, pp. 257–273, 2002.
[18] A. Hofleitner and A. Bayen, "Optimal decomposition of travel times measured by probe vehicles using a statistical traffic flow model," in *Proc. 14th IEEE Intell. Transport. Syst. Conf.*, Oct. 2011, pp. 815–821.
[19] A. Hofleitner, R. Herring, and A. Bayen, "Probability distributions of travel times on arterial networks: A traffic flow and horizontal queuing theory approach," in *Proc. 91st Transport. Res. Board Annu. Meeting*, Washington, DC, USA, Jan. 2012, number 12-0798.
[20] A. Hofleitner, R. Herring, and A. Bayen, "Arterial travel time forecast with streaming data: A hybrid approach of flow modeling and machine learning," *Transport. Res. Part B: Methodological*, vol. 46, no. 9, pp. 1097–1122, 2012.
[21] T. Hunter, P. Abbeel, and A. M. Bayen, "The path inference filter: Model-based low-latency map matching of probe vehicle data," *Algorithmic Foundations of Robotics X*, pp. 591–607, 2013.
[22] T. Hunter, T. Das, M. Zaharia, A. Bayen, and P. Abbeel, "Large-scale online expectation maximization with spark streaming," *NIPS Workshop Parallel Large-Scale Machine Learning*, 2012.
[23] T. Hunter, T. Moldovan, M. Zaharia, S. Merzgui, J. Ma, M. J. Franklin, P. Abbeel, and A. M. Bayen, "Scaling the mobile millennium system in the cloud," in *Proc. SOCC*, 2011, p. 28.
[24] Inrix Inc. [Online]. Available: http://www.inrix.com
[25] M. Lighthill and G. Whitham, "On kinematic waves. II. A theory of traffic flow on long crowded roads," in *Proc. Royal Soc. London Series A, Math. Phys. Sci.*, May 1955, vol. 229, no. 1178, pp. 317–345.
[26] W.-H. Lin, A. Kulkarni, and P. Mirchandani, "Short-term arterial travel time prediction for advanced traveler information systems," in *Intelligent Transportation Systems*. New York, NY, USA: Taylor & Francis, 2004, vol. 8, pp. 143–154.
[27] T. Miwa, T. Sakai, and T. Morikawa, "Route identification and travel time prediction using probe-car data," *International J.*, 2004.
[28] P. G. Moschopoulos, "The distribution of the sum of independent gamma random variables," *Ann. Inst. Stat. Math.*, vol. 37, no. 1, pp. 541–544, 1985.
[29] R. Neal and G. Hinton, "A view of the EM algorithm that justifies incremental, sparse, and other variants," in *Learning in Graphical Models*. Cambridge, MA, USA: MIT Press, 1999, pp. 355–368.
[30] Storm [Online]. Available: https://github.com/nathanmarz/storm/wiki
[31] Telenav Inc. [Online]. Available: http://www.telenav.com
[32] S. Thrun, "Probabilistic robotics," *Commun. ACM*, vol. 45, no. 3, pp. 52–57, 2002.
[33] TTI, Texas Transport. Inst.: Urban Mobility Information, 2007 Annual Urban Mobility Report, 2007. [Online]. Available: http://mobility.tamu.edu/ums/
[34] D. B. Work, S. Blandin, O. P. Tossavainen, B. Piccoli, and A. M. Bayen, "A traffic model for velocity data assimilation," *Appl. Math. Res. eXpress*, vol. 2010, no. 1, p. 1, 2010.
[35] M. Zaharia, T. Das, H. Li, S. Shenker, and I. Stoica, USENIX Association, "Discretized streams: An efficient and fault-tolerant model for stream processing on large clusters," in *Proc. 4th USENIX Conf. Hot Topics Cloud Ccomput.*, 2012, p. 10-10.
[36] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. 9th USENIX Conf. Networked Syst. Design Implementation*, 2011.

Timothy Hunter received the Engineering degree in applied mathematics from the Ecole Polytechnique, Palaiseau, France, in July 2007, and the M.S. degree in electrical engineering from Stanford University, Stanford, CA, USA, in 2009. He is currently working towards the Ph.D. degree at the Department of Electrical Engineering and Computer Science and in the AMPLab, University of California at Berkeley, Berkeley, CA, USA.

His research interests include new programming models for machine learning with big data, and some applications to estimation and transportation.

Tathagata Das is working towards the Ph.D. degree as a second year graduate student in computer science at the University of California Berkeley, Berkeley, CA, USA.

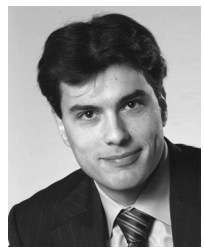
He is currently working with Prof. Scott Shenker. He is interested in cloud computing and datacenters.

Matei Zaharia is working towards finishing the Ph.D. degree at the University of California Berkeley, Berkeley, CA, USA, where he works on computer systems, networks, and cloud computing. Afterwards, he will start a position with the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology (MIT), Cambridge, MA, USA.

Pieter Abbeel received the B.S. and M.S. degrees in electrical engineering from KU Leuven, Leuven, Belgium, and the Ph.D. degree in computer science from Stanford University, Stanford, CA, USA, in 2008.

He joined the faculty at the University of California Berkeley, Berkeley, CA, USA, in Fall 2008, with an appointment in the Department of Electrical Engineering and Computer Sciences. He has developed apprenticeship learning algorithms which have enabled advanced helicopter aerobatics, including maneuvers such as tic-tocs, chaos and auto-rotation, which only exceptional human pilots can perform. His group has also enabled the first end-to-end completion of reliably picking up a crumpled laundry article and folding it.

Dr. Abbeel has won various awards, including Best Paper Awards at ICML and ICRA, the Sloan Fellowship, the Okawa Foundation Award, and 2011's TR35. His work has been featured in many popular press outlets, including BBC, MIT Technology Review, Discovery Channel, SmartPlanet and Wired.



Alexandre M. Bayen received the Engineering degree in applied mathematics from the Ecole Polytechnique, Palaiseau, France, in July 1998, the M.S. and Ph.D. degrees in aeronautics and astronautics from Stanford University, Stanford, CA, USA, in June 1999 and December 2003, respectively.

He was a Visiting Researcher at NASA Ames Research Center from 2000 to 2003. He worked as the Research Director of the Autonomous Navigation Laboratory at the Laboratoire de Recherches Balistiques et Aerodynamiques, (Ministere de la Defense, Vernon, France), where he holds the rank of Major. He is an Associate Professor of Electrical Engineering and Computer Sciences at the University of California Berkeley, Berkeley, CA, USA. He has authored one book and over 100 articles in peer reviewed journals and conferences.

Dr. Bayen is the recipient of the Ballhaus Award from Stanford University in 2004, of the CAREER Award from the National Science Foundation in 2009, and is a NASA Top 10 Innovators on Water Sustainability, 2010. His projects Mobile Century and Mobile Millennium received the 2008 Best of ITS Award for Best Innovative Practice, at the ITS World Congress and a TRANNY Award from the California Transportation Foundation, 2009. He is the recipient of the Presidential Early Career Award for Scientists and Engineers (PECASE) Award from the White House in 2010. Mobile Millennium has been featured more than 100 times in the media, including TV channels and radio stations (CBS, NBC, ABC, CNET, NPR, KGO, the BBC), and in the popular press (*Wall Street Journal*, *Washington Post*, *LA Times*).