Using Storm for scaleable sequential statistical inference

Simon Wilson[1] Arnab Bhattacharya[1] Gernot Roetzer[1]
Séan Ó'Ríordáin[1] Tiep Mai[2] Peter Cogan[3]
Oscar Robles Sánchez[4] Louis Aslett[5]

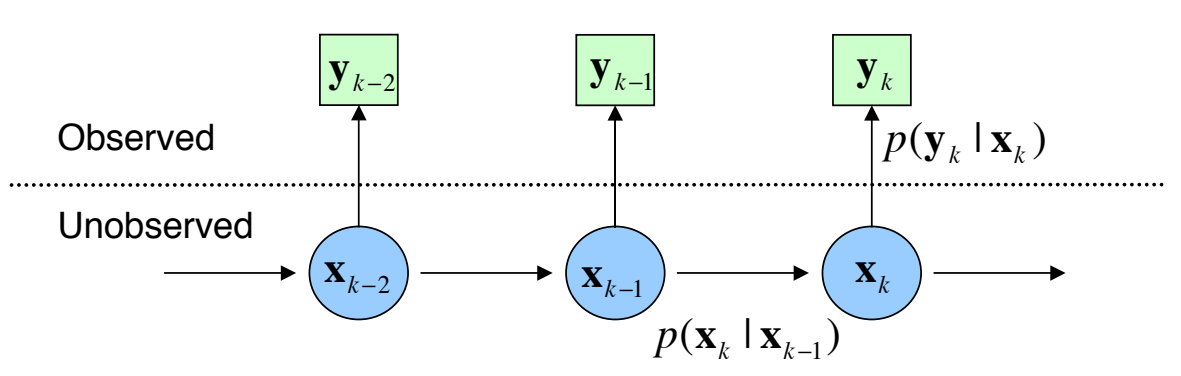
[1]Trinity College Dublin, Ireland
[2]Bell Labs, Dublin, Ireland
[3]Amdocs, Dublin, Ireland
[4]Universidad Rey Juan Carlos, Madrid, Spain
[5]Oxford University, UK

Overview

- A sequential learning algorithm and their 'topology';
- What is Storm?
- Two illustrations of using Storm for sequential data analysis;
- Challenges in using Storm for typical sequential learning algorithms.

A SEQUENTIAL LEARNING ALGORITHM

Dynamic state space/HMM model



Model specified by $p(y_k \mid x_k, \theta)$, $p(x_0)$, $p(x_k \mid x_{k-1})$ and prior $p(\theta)$:

$$p(\mathbf{y}_{1:t}, \mathbf{x}_{1:t}, \theta) = \left(\prod_{k=1}^{t} p(y_k \mid x_k, \theta)\, p(x_k \mid x_{k-1}, \theta) \right) p(x_0)\, p(\theta),$$

where $\mathbf{y}_{1:t} = (y_1, \ldots, y_t)$, etc.

The Problem

- Usual inference tasks with these models:

  Filtering Compute $p(x_t|\mathbf{y}_{1:t}, \theta)$ (also possibly $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t}, \theta)$)

  Estimation Compute $p(\theta|\mathbf{y}_{1:t})$ $\Leftarrow$ We will be concentrating on this.

  Prediction Compute $p(x_{t+1}|\mathbf{y}_{1:t}, \theta)$ and $p(y_{t+1}|\mathbf{y}_{1:t}, \theta)$

- In this work we want to do this quickly and sequentially:

$$
\begin{aligned}
p(x_{t-1}|\mathbf{y}_{1:t-1}, \theta) &\longrightarrow p(x_t|\mathbf{y}_{1:t}, \theta) \\
p(\theta|\mathbf{y}_{1:t-1}) &\longrightarrow p(\theta|\mathbf{y}_{1:t}) \\
p(y_t|\mathbf{y}_{1:t-1}, \theta) &\longrightarrow p(y_{t+1}|\mathbf{y}_{1:t}, \theta)
\end{aligned}
$$

The Problem

- Usual inference tasks with these models:

  Filtering Compute $p(x_t|\mathbf{y}_{1:t}, \theta)$ (also possibly $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t}, \theta)$)

  Estimation Compute $p(\theta|\mathbf{y}_{1:t})$ $\Leftarrow$ We will be concentrating on this.

  Prediction Compute $p(x_{t+1}|\mathbf{y}_{1:t}, \theta)$ and $p(y_{t+1}|\mathbf{y}_{1:t}, \theta)$

- In this work we want to do this quickly and sequentially:

$$\begin{aligned} p(x_{t-1}|\mathbf{y}_{1:t-1}, \theta) &\longrightarrow p(x_t|\mathbf{y}_{1:t}, \theta) \\ p(\theta|\mathbf{y}_{1:t-1}) &\longrightarrow p(\theta|\mathbf{y}_{1:t}) \\ p(y_t|\mathbf{y}_{1:t-1}, \theta) &\longrightarrow p(y_{t+1}|\mathbf{y}_{1:t}, \theta) \end{aligned}$$

The Problem

- Usual inference tasks with these models:

  Filtering Compute $p(x_t|\mathbf{y}_{1:t}, \theta)$ (also possibly $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t}, \theta)$)

  Estimation Compute $p(\theta|\mathbf{y}_{1:t}) \Leftarrow$ We will be concentrating on this.

  Prediction Compute $p(x_{t+1}|\mathbf{y}_{1:t}, \theta)$ and $p(y_{t+1}|\mathbf{y}_{1:t}, \theta)$

- In this work we want to do this quickly and sequentially:

$$
\begin{aligned}
p(x_{t-1}|\mathbf{y}_{1:t-1}, \theta) &\longrightarrow p(x_t|\mathbf{y}_{1:t}, \theta) \\
p(\theta|\mathbf{y}_{1:t-1}) &\longrightarrow p(\theta|\mathbf{y}_{1:t}) \\
p(y_t|\mathbf{y}_{1:t-1}, \theta) &\longrightarrow p(y_{t+1}|\mathbf{y}_{1:t}, \theta)
\end{aligned}
$$

The Principle

- Simple manipulation of probability laws yields:

$$p(\theta \,|\mathbf{y}_{1:t}) \;\propto\; \left.\frac{p(\mathbf{y}_{1:t} \,|\, \mathbf{x}_{1:t}, \theta)\, p(\mathbf{x}_{0:t} \,|\, \theta)\, p(\theta)}{p(\mathbf{x}_{0:t} \,|\, \mathbf{y}_{1:t}, \theta)}\right|_{\mathbf{x}_{0:t}=\mathbf{x}^*(\theta)},$$

  for any $\mathbf{x}^*(\theta)$ such that $p(\mathbf{x}^*(\theta) \,|\, \mathbf{y}_{1:t}, \theta) > 0$;

- Further manipulation yields a sequential version:

$$p(\theta \,|\, \mathbf{y}_{1:t}) \propto p(\theta \,|\, \mathbf{y}_{1:t-1}) \left.\frac{p(y_t \,|\, x_t, \theta)\, p(x_t \,|\, \mathbf{y}_{1:t-1}, \theta)}{p(x_t \,|\, \mathbf{y}_{1:t}, \theta)}\right|_{x_t=x^*(\theta)}$$

- For many models the dimension of θ is small enough to allow $p(\theta \,|\mathbf{y}_{1:t})$ to be computed on a discrete grid $\Theta = \{\theta_j \,|\, j = 1, \ldots, J\}$;

The Principle

- The former requires a filtering density $p(\mathbf{x}_{0:t} \mid \mathbf{y}_{1:t}, \theta)$ of dimension $t + 1$;
  - Computation time grows with t;
- The latter requires both filtering and prediction densities but only of fixed dimension (those of x_t and y_t);
  - Any algorithm that outputs the filtering and prediction densities can be used to implement it;
  - Computation time constant with t;
  - This is the basis of our approach.

Doing this sequentially

Observations:

- Can update from $p(\theta \mid \mathbf{y}_{1:t-1})$ to $p(\theta \mid \mathbf{y}_{1:t})$ on grid Θ;
- Typical choice for $x^*(\theta)$ is $\arg\max_\theta p(x_t \mid \mathbf{y}_{1:t}, \theta)$;
- Normalising constant quick to compute (sum over Θ);
- Trivial parallelisation of the computation over Θ:
  - Important for the rest of the talk!

Non-Sequential Method — INLA

- The integrated nested Laplace approximation:

$$\tilde{p}_{\text{INLA}}(\theta \,|\mathbf{y}_{1:t}) \;\propto\; \left.\frac{p(\mathbf{y}_{1:t} \,|\, \mathbf{x}_{1:t}, \theta)\, p(\mathbf{x}_{0:t} \,|\, \theta)\, p(\theta)}{\tilde{p}_G(\mathbf{x}_{0:t} \,|\, \mathbf{y}_{1:t}, \theta)}\right|_{\mathbf{x}_{0:t}=\mathbf{x}^*(\theta)},$$

  where $\tilde{p}_G(\mathbf{x}_{0:t} \,|\, \mathbf{y}_{1:t}, \theta)$ is a Gaussian approximation;
- Computed on grid Θ which INLA also provides;
- Very accurate for Gaussian X_t so can be computed until t is too large for fast computation.

Sequential method with approximate filtering and prediction densities

- More typically have approximations $\tilde{p}(x_t \mid \mathbf{y}_{1:t-1}, \theta)$ and $\tilde{p}(x_t \mid \mathbf{y}_{1:t}, \theta)$;
- So sequential update approximation to $p(\theta \mid \mathbf{y}_{1:t})$ is:

$$\tilde{p}(\theta \mid \mathbf{y}_{1:t}) \propto \tilde{p}(\theta \mid \mathbf{y}_{1:t-1}) \left. \frac{p(y_t \mid x_t, \theta)\, \tilde{p}(x_t \mid \mathbf{y}_{1:t-1}, \theta)}{\tilde{p}(x_t \mid \mathbf{y}_{1:t}, \theta)} \right|_{x_t = x^*(\theta)},$$

  for any $x^*(\theta)$ such that $\tilde{p}(x^*(\theta) \mid \mathbf{y}_{1:t}, \theta) > 0$.
- Use this when t has got too big for INLA;
- Dynamic updating of the grid?

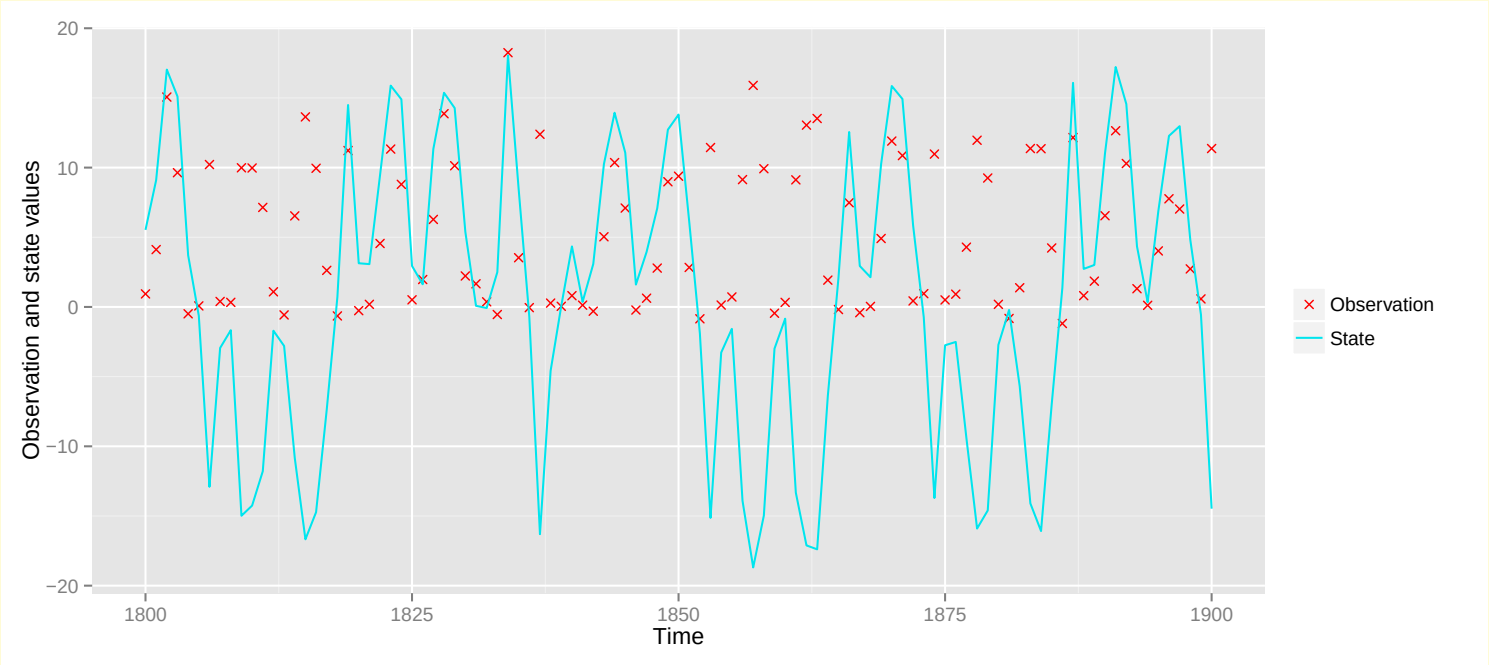
Example: non-stationary growth model of Kitagawa (JCGS, 1996)

$$
\begin{aligned}
y_t &= 0.05 x_t^2 + w_t, \\
x_t &= 0.5 x_{t-1} + \frac{25 x_{t-1}}{1 + x_{t-1}^2} + 8 \cos(1.2(t-1)) + v_t.
\end{aligned}
$$

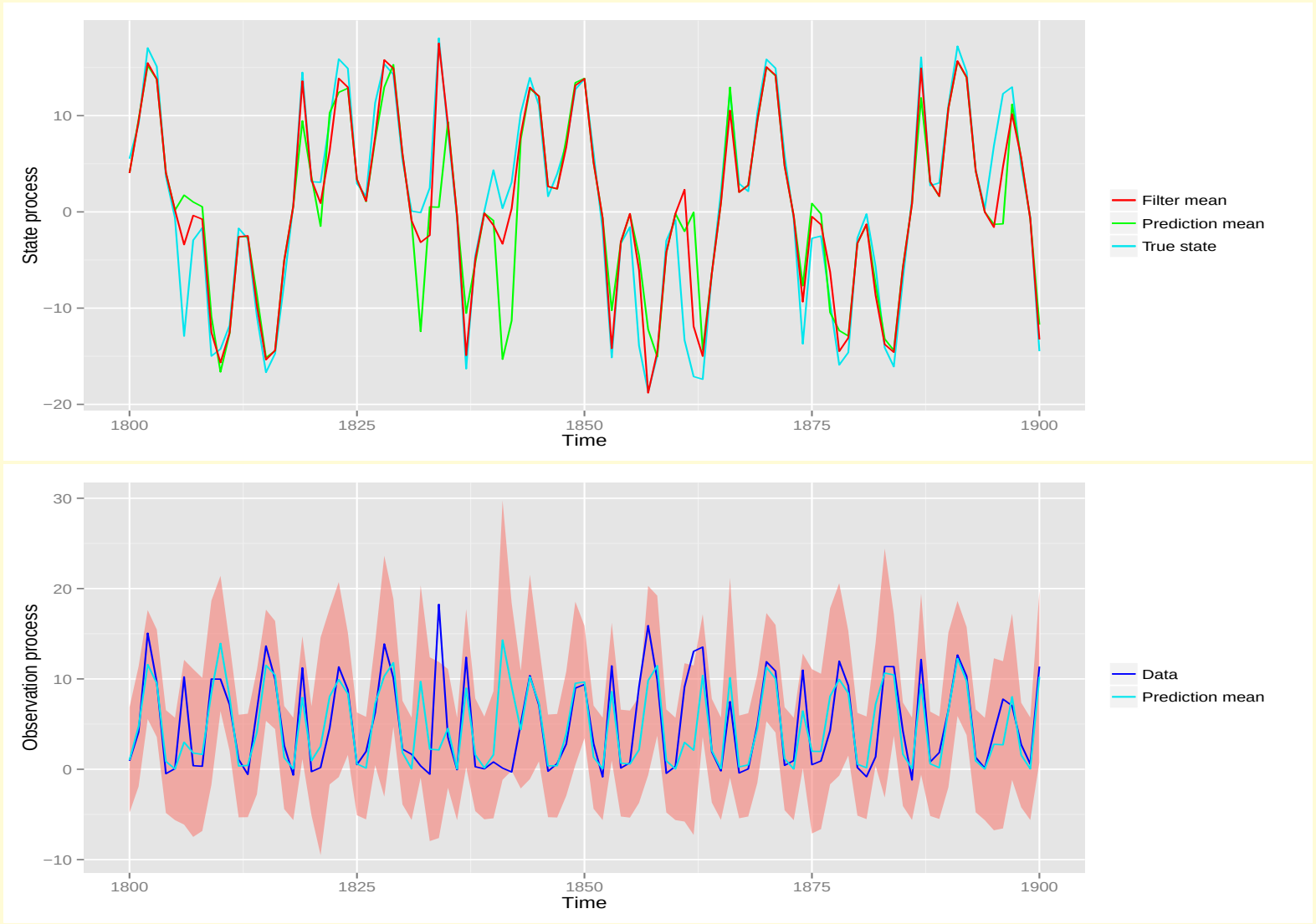
$$
w_t \sim N(0, W), \quad v_t \sim N(0, V)
$$

- Inference on $\theta = (V, W)$.
- Used unscented Kalman filter.
- Model first seen in Andrede Netto et al. (1978).

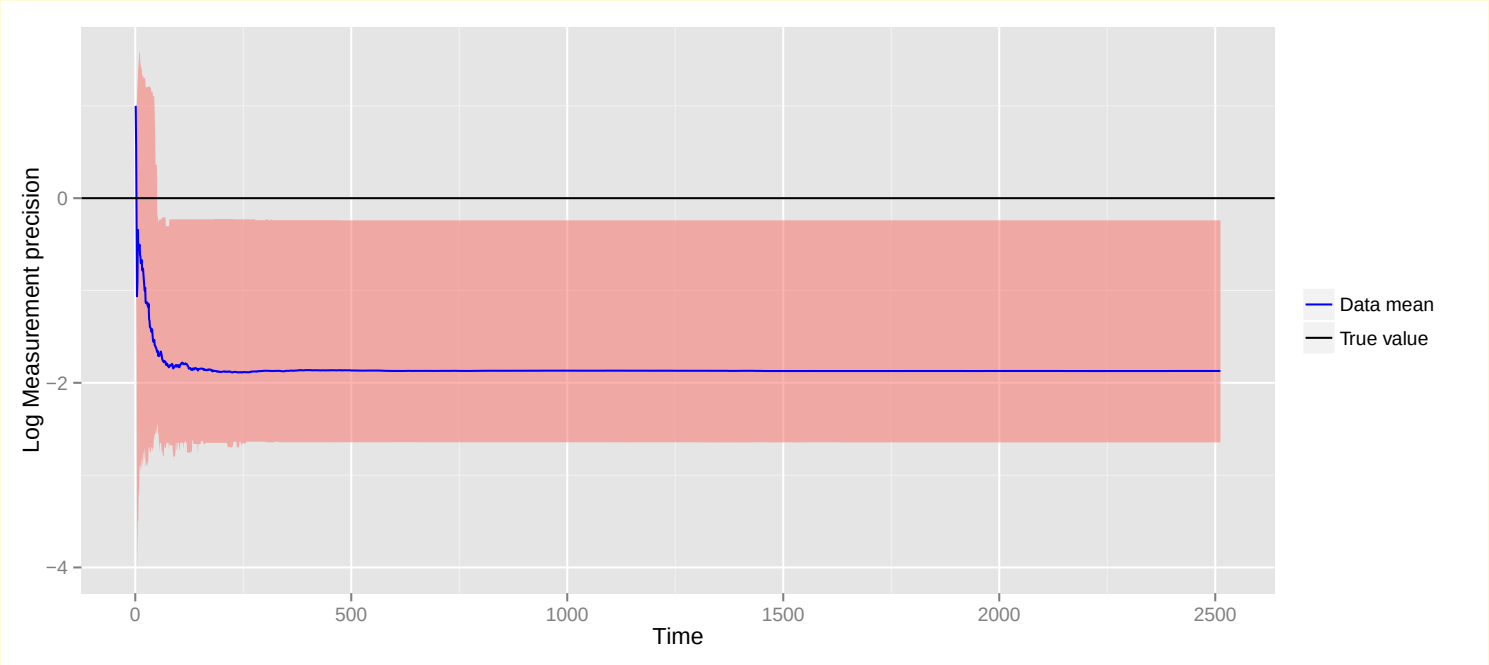
Kitagawa model: simulated state and observations



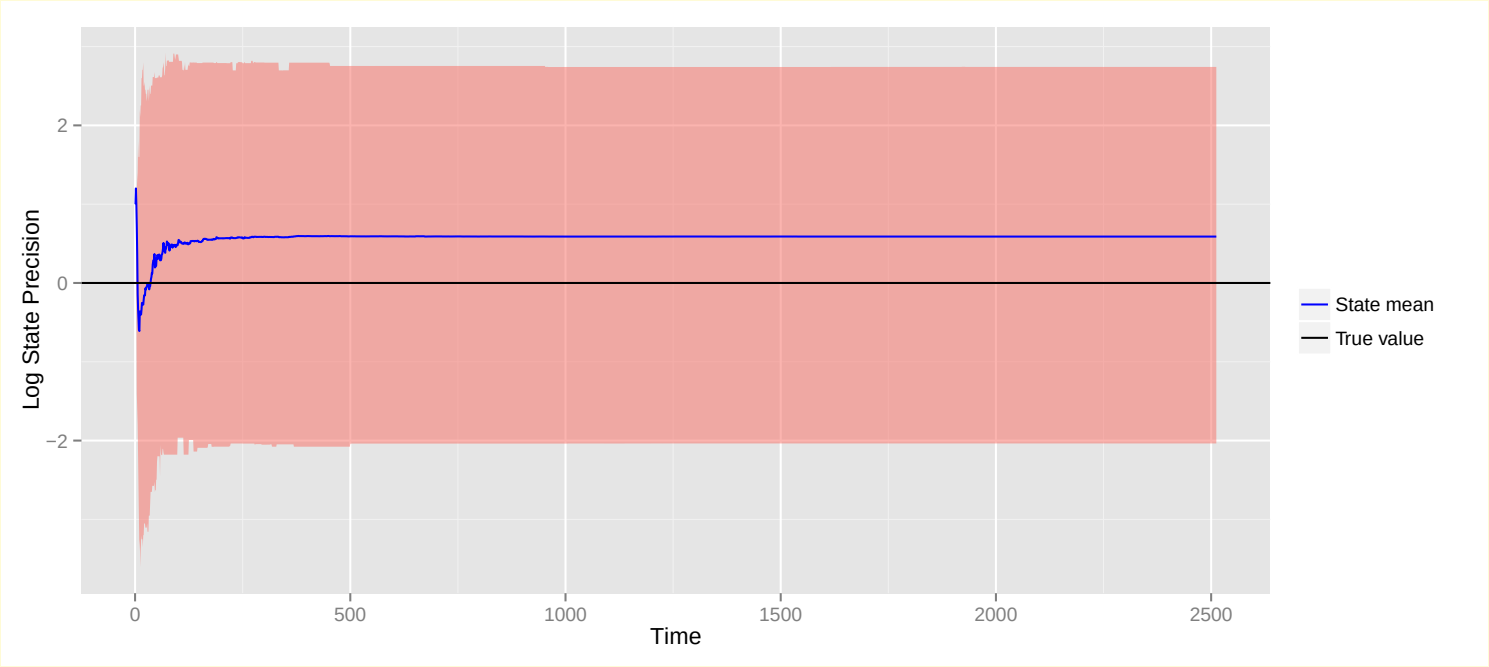
Kitagawa model: UKF filter and prediction



Kitagawa model: sequential inference on W



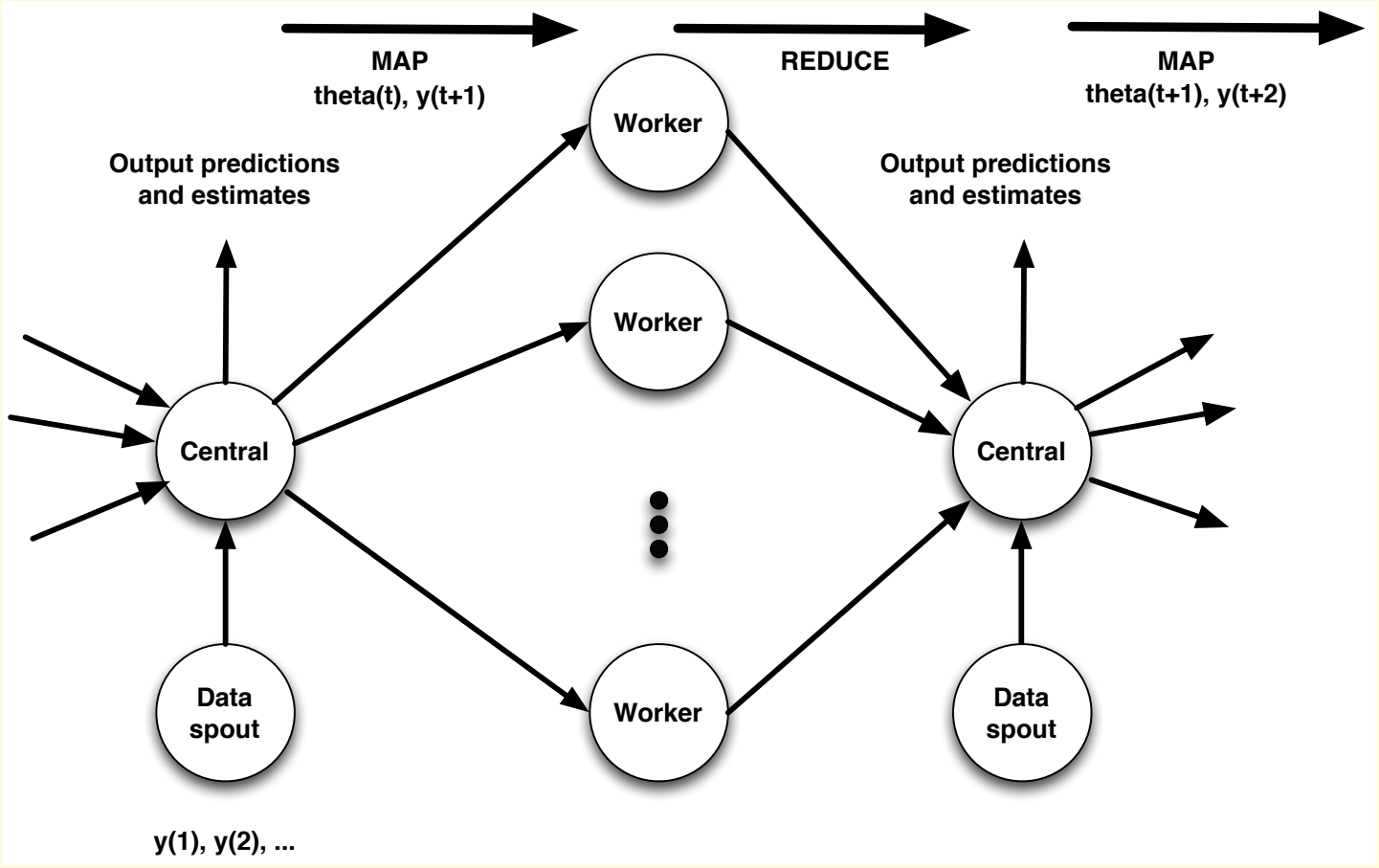
Kitagawa model: sequential inference on V



Comments

- Why the estimation bias?
- Even happens if we fix one of the variances to its true value;
- Might expect V to be overestimated because of the two solutions?

Computation topology: MapReduce



STORM

What is Storm?

- It's a parallel computing environment for doing streaming data analysis in a scaleable and fault tolerant way;
- Originally developed by a company called BackType — acquired by Twitter in 2011 — Twitter made it open source the same year;
- It's easy to install and program in Java (but you can code in other languages like Python);
- There are even some crude ways to link it to `R` (and hopefully these will be easier to use soon);
- Have you heard of Hadoop?
  - If yes then Storm is like Hadoop but for streaming data (and ignore the next 2 slides);
  - If no then see 2 next slides!

What is Storm? More details

- All large IT companies run large servers consisting of *many* processing cores networked together;
- These systems are set up to do parallel computing using the *MapReduce* paradigm. This means that:
  - Any operation (e.g. a web search) can be split into many essentially identical operations that can be done independently at the same time;
  - The operating system tries to detect if any processor has failed. If it thinks this happens then that job is assigned to another processor.
- These are *batch* computations e.g. there is a task to do, you do it, report the result and it's finished.

What is Storm? More details

- What about streaming data e.g.
  - Sentiment analysis from a (never-ending) Twitter feed;
  - Accident detection from a (never-ending) video stream from a highway;
  - Object tracking from a (never-ending) radar or IR camera feed.
- Storm is designed to implement the MapReduce idea but for streaming data (and not batch);
- In principle, the computation never ends (in practice, it ends when you manually kill it).

Streaming data analysis

- We assume a never-ending stream of data (called *tuples* in Storm) $x_1, x_2, \ldots$;
- The task is to sequentially do some analysis as the data streams to us and output it
- Examples:
  - Calculate a running mean a stream of numbers, so we output $\bar{x}_1, \bar{x}_2, \ldots$ where $\bar{x}_n = \sum_{i=1}^{n} x_i/n$;
  - Report market sentiment from a stream of tweets, so output 'positive' or 'negative' after every new tweet or after every minute, etc.

Topologies, bolts and spouts

- A Storm program starts with a graph that describes how data flows from input to output (the *topology*);
- Nodes in the topology are either *bolts* or *spouts*:
  - Spouts are sources of data;
  - Bolts are functions that process data; they have an input and an output.
- You write code to implement the spouts and bolts (in Java, Python, etc.);
- You specify where the input and output from each spout and bolt is to go, and how much parallelism you want;
- Storm does the rest of the work to manage the running of this on a cluster.
- *A lot more* to be said about how Storm works!

Simple example: computing a posterior distribution sequentially

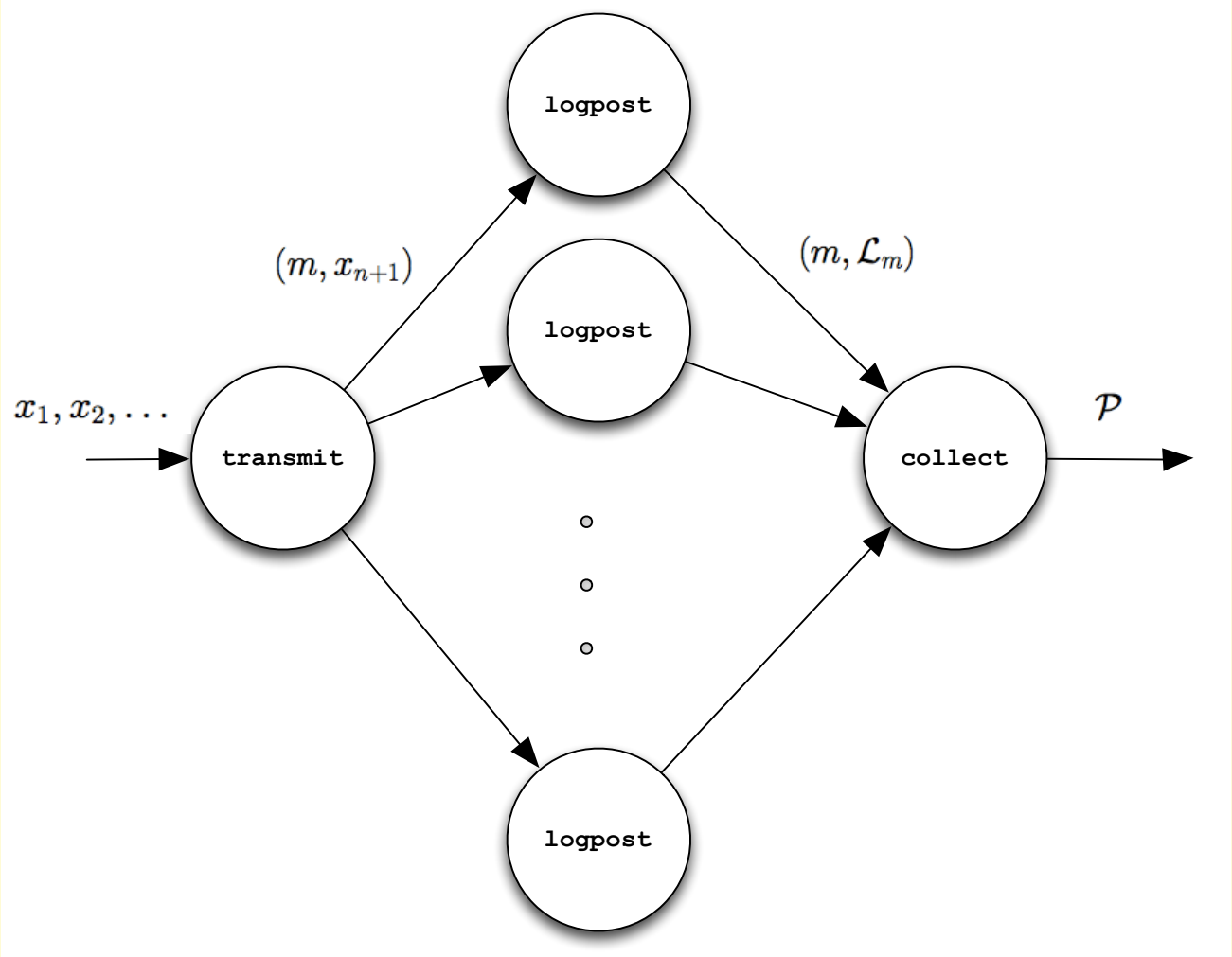
- We have a stream of Gaussian data $x_1, x_2, \ldots$ with unknown mean μ, precision τ;
- Goal is to sequentially compute the posterior distribution $p(\mu, \tau \,|\, x_1, \ldots, x_n)$;
- When x_{n+1} is streamed then we update the posterior by Bayes':

$$p(\mu, \tau \,|\, x_1, \ldots, x_n, x_{n+1}) \propto p(x_{n+1} \,|\, \mu, \tau) p(\mu, \tau \,|\, x_1, \ldots, x_n).$$

- We compute the posterior on a discrete grid

$$\Theta = \{(\mu_i, \tau_j) \,|\, i = 1, \ldots, I;\ j = 1, \ldots, J\}.$$

Computing a posterior distribution sequentially: a topology



Computing a posterior distribution sequentially: a topology

- We partition the grid into M sub-grids $\Theta_1, \ldots, \Theta_M$;
- We have M replications of a bolt `logpost` that computes the unnormalised log posterior:

$$l_n(\mu, \tau) = \log(p(\mu, \tau)) + \sum_{k=1}^{n} \log(p(x_k \,|\, \mu, \tau)).$$

- Each bolt is assigned to compute this over one of the Θ_m;
- When x_{n+1} is streamed, it is transmitted to *all* these bolts that then update l_n to l_{n+1} by computing $\log(p(x_{n+1} \,|\, \mu, \tau))$ and adding it to $l_n(\mu, \tau)$;

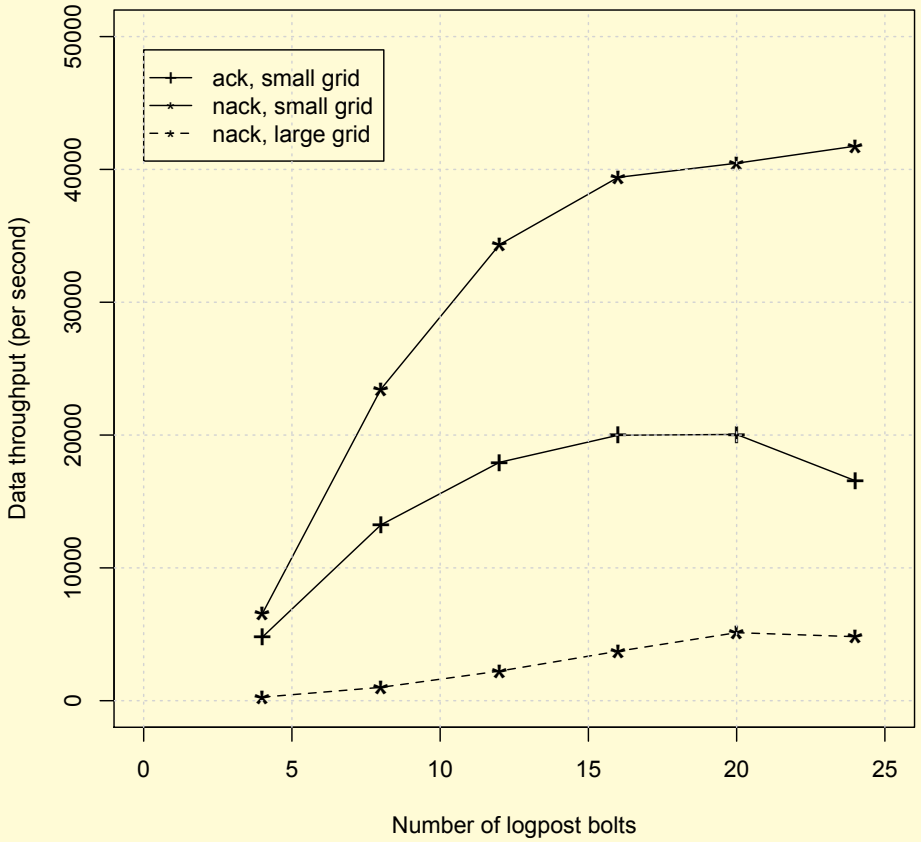
Computing a posterior distribution sequentially: a topology

- After a certain number of data points have been streamed, the `logpost` bolts transmit the l_n values to a `collect` bolt that merges and normalises them:

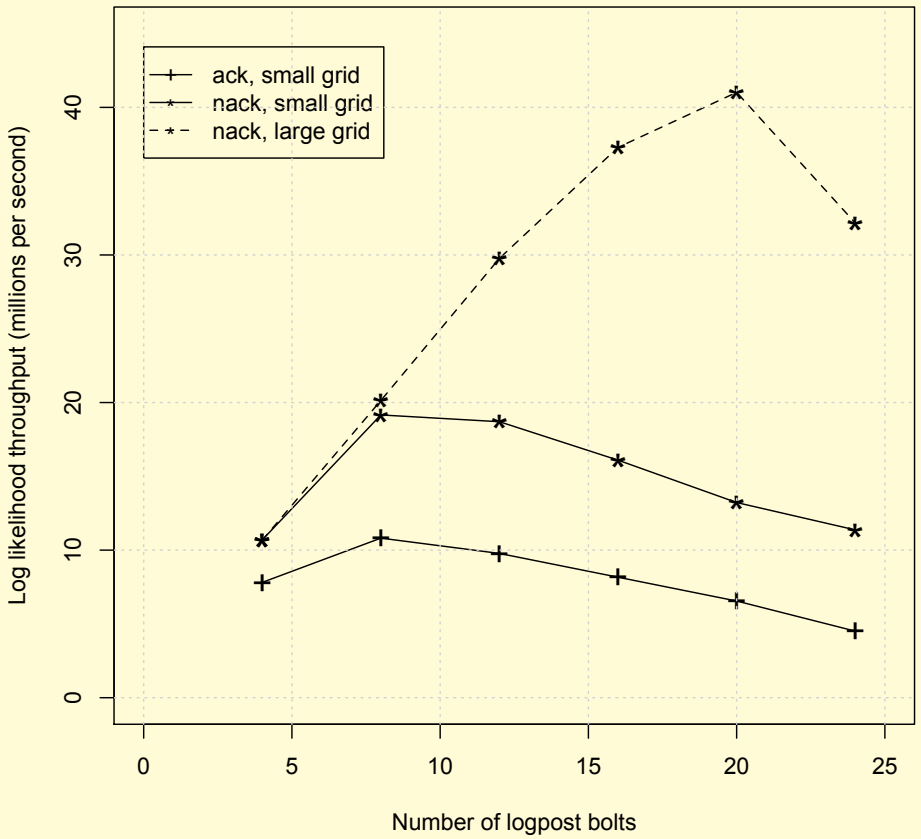
$$p(\mu, \tau \,|\, x_1, \ldots, x_n) \;\approx\; \frac{\exp(l_n(\mu, \tau))}{\sum_{(\mu,\tau)\in\Theta} \exp(l_n(\mu, \tau))\, \Delta\mu\, \Delta\tau}.$$

- This was implemented on a cluster of 5 machines, each with a 4 core processor;
- Run on 2 grids: one with 6,500 points, another with 160,000 points;
- Posterior distribution was computed by the `collect` bolt every 50,000 observations.

Computing a posterior distribution sequentially: data throughput



Computing a posterior distribution sequentially: likelihood throughput



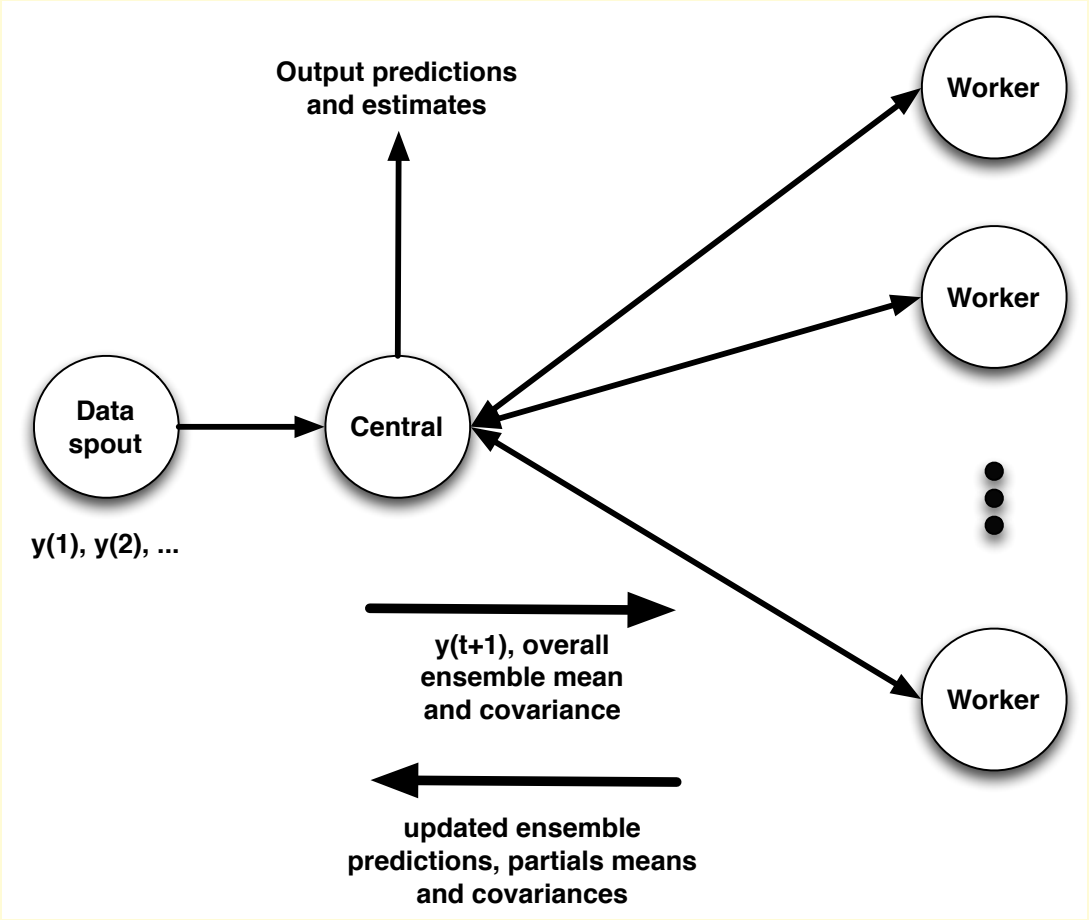
More complicated example: the ensemble Kalman filter

- A Monte Carlo version of the Kalman filter:

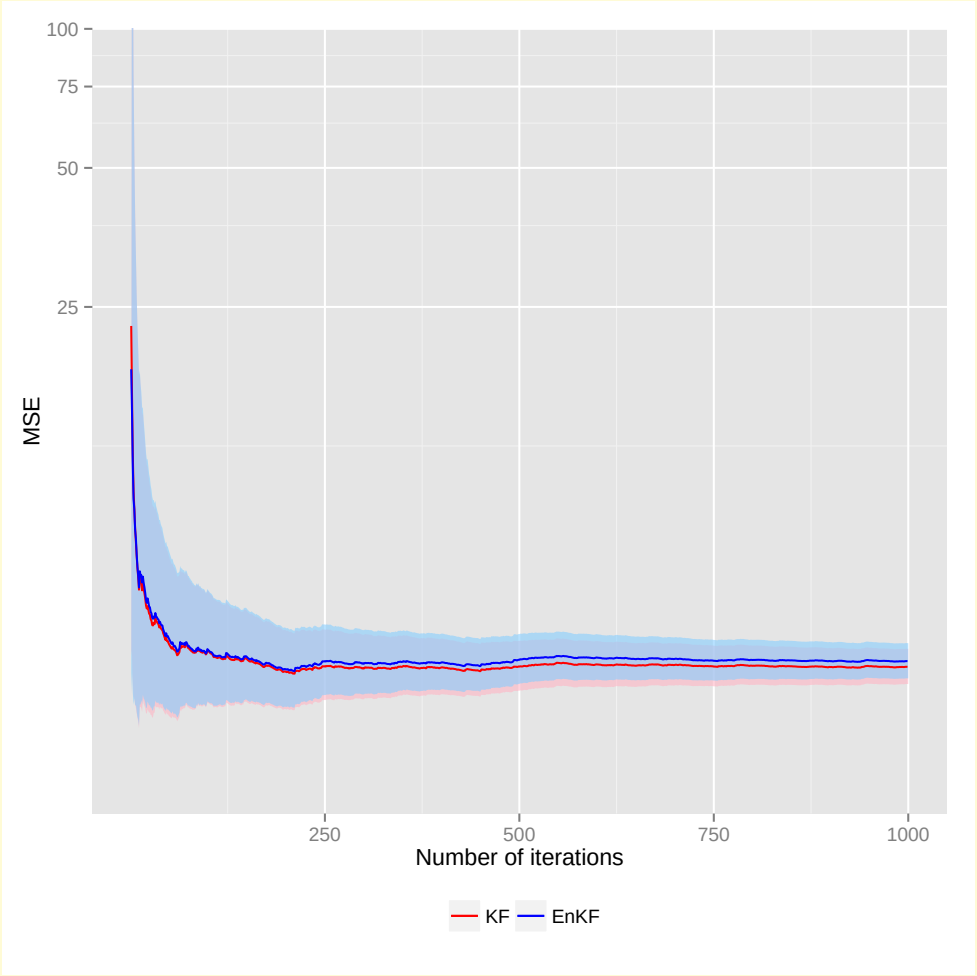
$$\begin{aligned} y_t &= Hx_t + \epsilon_t, \quad \epsilon_t \sim N(0, R); \\ x_t &= Kx_{t-1} + \eta_t, \quad \eta_t \sim N(0, Q). \end{aligned}$$

- Basic operation is to maintain an ensemble $X = (x_1, \ldots, x_M)$ of values that approximate $p(x_t \mid y_1, \ldots, y_t)$;
- Ensemble is updated on observation of y_{t+1} by reweighting and recomputing ensemble mean and covariance.
- Is applicable to more general non-linear state space models;
- ... also as an approximation to non-Gaussian models;

Ensemble Kalman filter topology

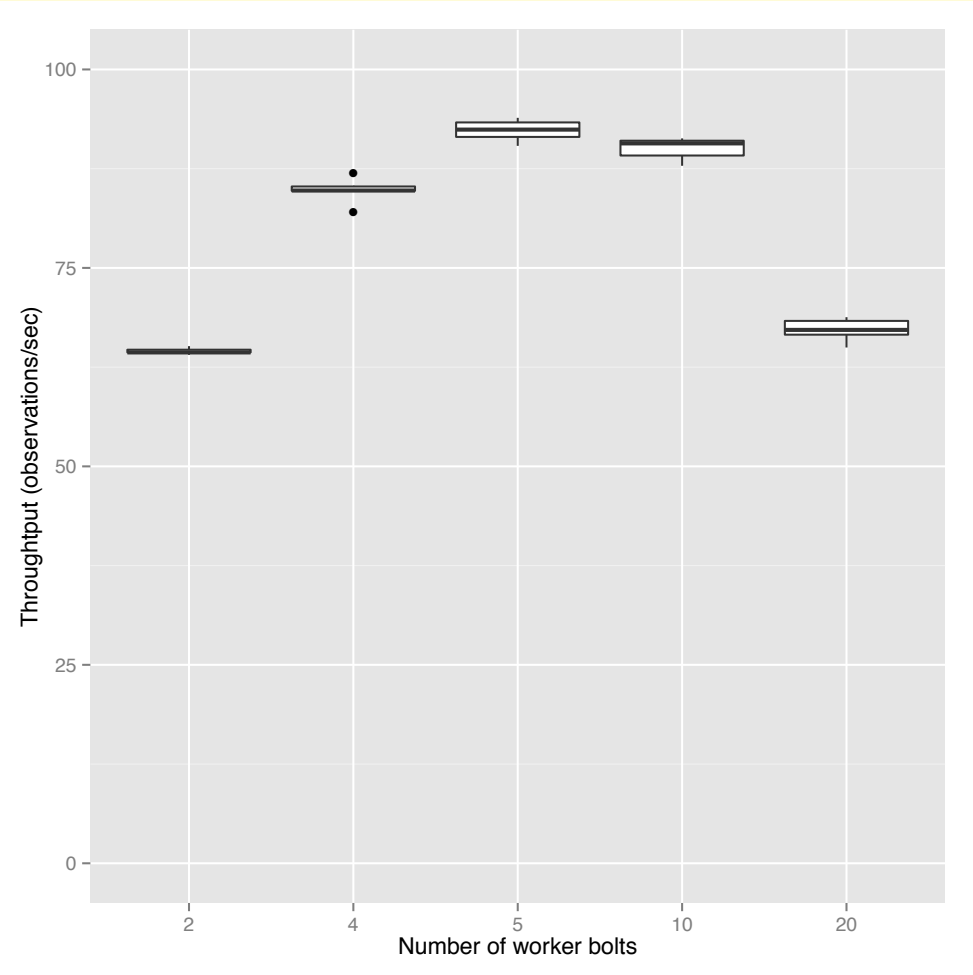


Applying to the linear model case: compared to KF



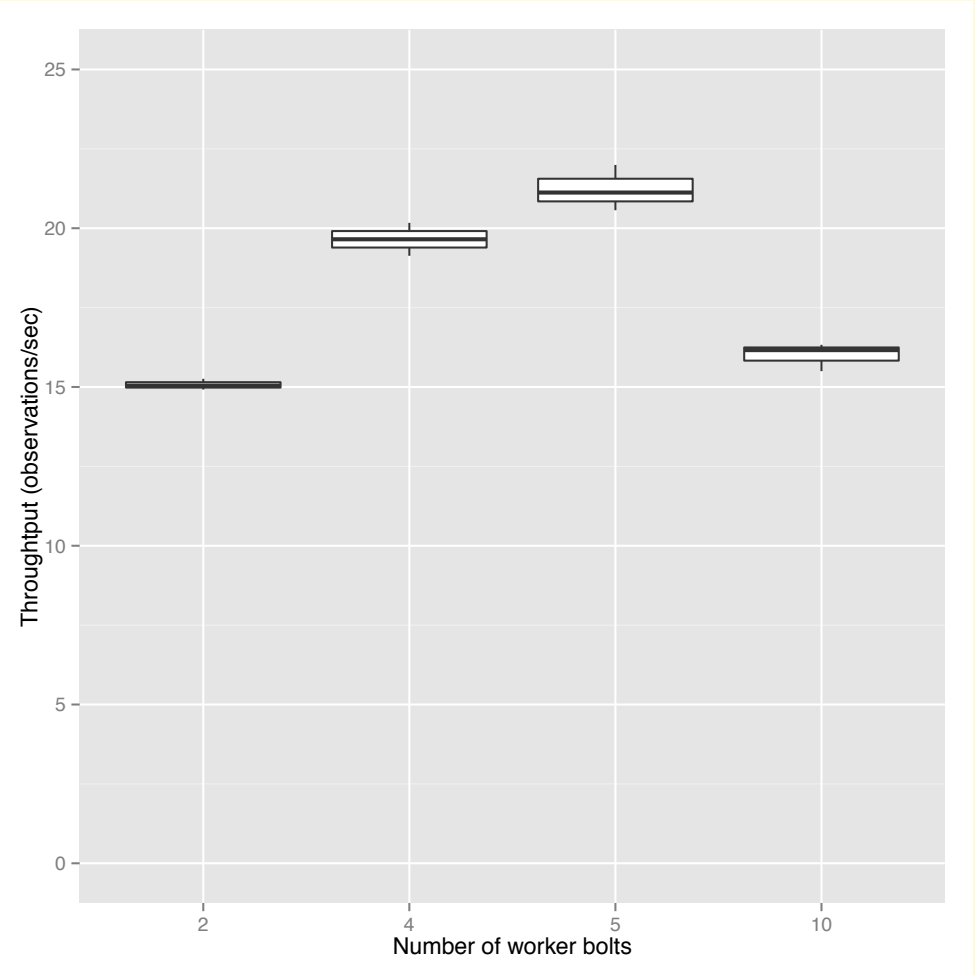
Applying to the linear model case: throughput

$\dim(y_t) = 15$, $\dim(x_t) = 200$.



Applying to the linear model case: throughput

$\dim(y_t) = 25$, $\dim(x_t) = 500$.



Some discussion

- Storm has several nice properties:
  - Not too difficult to program;
  - Easily scales (Storm handles all of the management of parallelization);
  - Fault tolerant;
  - Starting to be linked to things like `R`.
- Of course its performance depends a lot on the cluster that you use;
- The second example is quite a common topology for sequential inference methods:
  - Kalman filter and its extensions;
  - Particle filters?
- Principal practical difficulties:
  - Most sequential learning algorithms require synchronisation between data arrival and computation;
  - Most require that bolts will store a state.

References

- Apache Software Foundation (2013). MapReduce tutorial. `http://hadoop.apache.org/docs/stable/mapred_tutorial.html`.
- Bedini, I., S. Sakr, B. Theeten, A. Sala, and P. Cogan (2013). Modeling performance of a parallel streaming engine: bridging theory and costs. In *Proceedings of the International Conference on Performance Engineering*, pp. 173–184.
- Boyd, S., N. Parikh, E. Chu, B. Peleato, and J. Eckstein (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning 3*, 1–122.
- Dean, J. and S. Ghemawat (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM 51*, 107–113.

References

- Andrede Netto, M.L., Gimeno, L. and Mendes, K.J. (1978). On the optimal and suboptimal nonlinear filtering problem for discrete-time systems. *IEEE Transactions on Automatic Control*, **23**: 1062–1067.
- Gates, A. (2011). *Programming Pig: Dataflow Scripting with Hadoop*. O'Reilly Media.
- Kitagawa, G. (1996). Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*. **5**: 1–25.

References

- Leibiusky, J., G. Eisbruch and D. Simonassi (2012). *Getting started with Storm*. O'Reilly Media.
- White, T. (2012). *Hadoop, the Definitive Guide* (Third ed.). Yahoo Press, O'Reilly Media.
- Zhao, J. and J. Pjesivac-Grbovic (2009). Mapreduce: The programming model and practice. `http://research.google.com/archive/papers/mapreduce-sigmetrics09-tutorial.pdf`. Sigmetrics 2009 tutorial.