Asynchronous Gibbs sampling

Alexander Terenin
Imperial College London

Daniel Simpson
University of Toronto

David Draper
University of California, Santa Cruz

Abstract

Gibbs sampling is a Markov Chain Monte Carlo (MCMC) method often used in Bayesian learning. MCMC methods can be difficult to deploy on parallel and distributed systems due to their inherently sequential nature. We study *asynchronous Gibbs sampling*, which achieves parallelism by simply ignoring sequential requirements. This method has been shown to produce good empirical results for some hierarchical models, and is popular in the topic modeling community, but was also shown to diverge for other targets. We introduce a theoretical framework for analyzing asynchronous Gibbs sampling and other extensions of MCMC that do not possess the Markov property. We prove that asynchronous Gibbs can be modified so that it converges under appropriate regularity conditions – we call this the *exact asynchronous Gibbs* algorithm. We study asynchronous Gibbs on a set of examples by comparing the exact and approximate algorithms, including two where it works well, and one where it fails dramatically. We conclude with a set of heuristics to describe settings where the algorithm can be effectively used.

1 Introduction

Bayesian methods have found increased application during the last two decades in the scientific community, as well as in technology, business, public policy, and other settings. Unfortunately, Bayesian computation has become increasingly difficult as data sets and models have grown in size and complexity.

In particular, one of the standard approaches – Markov Chain Monte Carlo (MCMC) (Metropolis et al., 1953; Hastings, 1970) – often does not scale well, either with data size or model complexity. This has been addressed in recent work by deploying MCMC on parallel and distributed systems such as GPUs and compute clusters – here we focus on the latter setting.

To use MCMC on a compute cluster efficiently, recent work has focused both on modifying the system's architecture to better suit MCMC (Wei et al., 2015; Ho et al., 2013) and on modifying MCMC to better suit the system (Newman et al., 2009). The simplest approach to parallelization – running multiple parallel chains – requires each chain to burn in individually, which limits its performance. One way to make Gibbs sampling (Geman and Geman, 1984) better suited to a compute cluster is to run it *asynchronously*, by sampling the next full conditional without waiting for previous ones to finish – this is illustrated in Figure 1. Similar approaches have recently been proposed for distributed optimization (Niu et al., 2011). Unfortunately, for Gibbs sampling this creates a stochastic process that does not possess the Markov property: empirical results regarding the behavior of such processes have largely outpaced their theoretical understanding.

Asynchronous Gibbs has found widespread use and industrial deployment, especially in the natural language processing community for the Latent Dirichlet Allocation model (Blei et al., 2003), where it was first proposed by Newman et al. (2009), and analyzed by Ihler and Newman (2012). However, Johnson et al. (2013) exhibited an explicit counterexample demonstrating that asynchronous Gibbs sampling can diverge, and analyzed its behavior for Gaussian targets. Other authors have analyzed the algorithm in settings where conditional independence allows the Markov property to be recovered, thereby making asynchronous and synchronous execution equivalent (Gonzalez et al., 2011; Neiswanger et al., 2013; Terenin et al., 2019).

The work most related to ours, and which originally appeared concurrently to our own, is that of De Sa et al. (2016). They showed – assuming Dobrushin's condition (Pedersen, 2007) – that the asymptotic bias and mixing time of asynchronous Gibbs can be bounded. This is the

Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS) 2020, Palermo, Italy. PMLR: Volume 108.

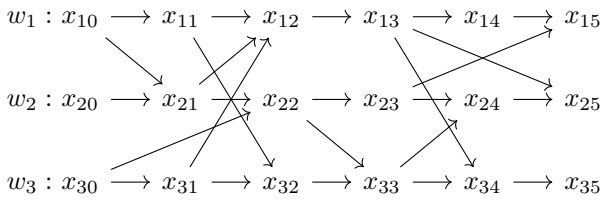

Figure 1: One possible sampling path of asynchronous Gibbs sampling. Here, workers w_1, w_2, w_3 sample values and transmit them to one another. Communication is not instantaneous – not all samples are transmitted, and not all transmissions are received.

only result we are aware of that holds for general targets with fully asynchronous execution which doesn't reduce to the synchronous case. We discuss the relationship between this important and largely complementary viewpoint with our approach in Section 4.

In this work, we analyze asynchronous Gibbs by defining the *exact asynchronous Gibbs* algorithm that includes a correction step, which we prove converges even if executed asynchronously. This allows study of the approximate algorithm in use by practitioners, through comparison with its convergent counterpart. Our framework allows for convergence analysis of MCMC algorithms in settings that do not possess the Markov property, and gives a general technique for the construction of convergent algorithms. Our contribution and focus is primarily theoretical, but we also showcase the method on a number of examples.

2 Asynchronous Gibbs Sampling

Asynchronous Gibbs sampling is an algorithm that modifies Gibbs sampling to make its implementation on a compute cluster more efficient. We present it in this section informally using an *actor model* (Hewitt, 1973) definition of parallelism – for an overview of the algorithm in a simpler setting, see Terenin and Xing (2017). Formal analysis is given in Appendix A. We now introduce notation.

(1) w_i: a worker capable of (a) performing computations, (b) transmitting messages to other workers, and (c) receiving messages from other workers.

(2) π: a probability measure with density $f(x)$ defined on $X = \mathbb{R}^d$ from which we wish to sample.

(3) $\mathbf{X}^k$: a matrix where each entry x_{ij} represents component j of the parameter vector $\boldsymbol{x}_i$ on worker w_i. This gives the total state of computation on all workers at time k. We generally suppress k from the notation.

The algorithm proceeds as follows.

Algorithm 1 (Asynchronous Gibbs sampling)**.** *For all workers, repeat the following.*

(i) Select a variable x_{ij} from some subset of $\boldsymbol{x}_i$ at random with fixed probability and update it using the full conditional distribution $x_{ij} \mid \boldsymbol{x}_{i,-j}$ of Gibbs sampling.

(ii) Transmit x_{ij} to other workers, if possible given network limitations.

(iii) Update the state $\boldsymbol{x}_i$ using any transmissions received from other workers by overwriting previous parameter values in arbitrary order.

This algorithm is illustrated in Figure 1. We assume that every worker can either update or receive each component of $\boldsymbol{x}_i$, and that each worker choses to do so in a random-scan fashion. Each worker's state is based both on values it has sampled and on values it has received from other workers – because of network delays, these may be out of date. Thus asynchronous Gibbs with network transmission that includes the possibility of delays is not a Markov chain.

The algorithm allows for messages to be dropped or not sent entirely, making it fault-tolerant with respect to network traffic. Since the number of possible messages grows quadratically with the number of workers, most messages will not be sent or received. We focus here on the case where all transmitted variables are sampled via Gibbs steps.

Can anything be proven about such a process? To study this question, we now consider two ways in which workers process updates received from other workers.

(a) *Asynchronous Gibbs*: accept all updates.

(b) *Exact asynchronous Gibbs*: worker w_i with current state $\boldsymbol{x}_i$ accepts updated parameter x'_j proposed by worker w_s, whose state when x'_j was randomly updated was $\boldsymbol{x}_s$, with Metropolis-Hastings (MH) probability

$$\min\left\{1, \frac{f(\boldsymbol{x}'_i)f(\boldsymbol{x}_i \mid \boldsymbol{x}_s)}{f(\boldsymbol{x}_i)f(\boldsymbol{x}'_i \mid \boldsymbol{x}_s)}\right\} \tag{1}$$

where $\boldsymbol{x}'_i$ is defined to be $x'_{ij} = x'_{sj}$ for component j transmitted by w_s and $x'_{ij} = x_{ij}$ for all other components.

Note that this update rule entails transmitting both the sender's newly updated parameter, and its full state given at the time this parameter was updated.

Here, (a) is the algorithm considered by Ihler and Newman (2012), De Sa et al. (2016), and other authors

– since it does not always converge, we refer to it as the *approximate algorithm*. We study it by examining its relationship with (b), which we call *exact* because it turns out that, under appropriate regularity conditions, it converges.

Theorem 2 (Asynchronous convergence, informal)**.** *Let $\mathbf{X}^k$ be an exact asynchronous Gibbs sampler. Assume temporarily that communication is instantaneous, and suppose in this setting that $\mathbf{X}^k$ converges sufficiently quickly on every worker as $k \to \infty$. Then $\mathbf{X}^k$ converges asynchronously in the sense of Baudet (1978), Bertsekas (1983), and Frommer and Szyld (2000).*

Proof. Appendix A. □

The argument proceeds by first studying distributed Gibbs sampling in the simpler setting where communication is instantaneous, there are no asynchronous delays, and $\mathbf{X}^k$ again possesses the Markov property. Here, we assume that both the underlying Gibbs sampler on each worker and communication between workers are sufficiently regular that $\mathbf{X}^k$ converges to stationarity at sufficient rate. We then use the general theory of Baudet (1978), Bertsekas (1983), and Frommer and Szyld (2000) to show that convergence under instantaneous communication implies asynchronous convergence, defined in an appropriate sense, irrespective of most details involving the asynchronous delays.

Having established this result, we proceed to study circumstances under which exact asynchronous Gibbs can be implemented. We then study asynchronous Gibbs sampling by studying how its trajectories differ from those of exact asynchronous Gibbs. To do so, we implement the algorithm in *Scala*, a compiled language interoperable with *Java* and well suited to parallel and distributed environments. Network communication is handled by *Akka*, an *actor model* framework designed for large-scale distributed applications.

2.1 Distributed computation in exchangeable latent variable models

If we are interested in sampling from an exchangeable latent-variable hierarchical Bayesian model, the posterior ratio used in the MH acceptance test in exact asynchronous Gibbs simplifies to an expression involving only one data point – this means that this ratio can be evaluated locally to each worker in a parallel environment. In particular, this allows us to partition the data and latent variables among the workers, and update a copy of the upper-level non-latent variables locally on each worker. To illustrate, consider the model

$$y_i \mid \nu_i \sim A(\nu_i) \qquad \nu_i \mid \theta \sim B(\theta) \qquad \theta \propto \tau(\theta) \qquad (2)$$

in which A and B are arbitrary distributions and y_i are data points. We can define a Gibbs sampler of the form

$$\nu_i \mid \theta, y_i \sim C(\theta, y_i) \quad \theta \mid \nu_1, .., \nu_n \sim D(\nu_1, .., \nu_n) \quad (3)$$

where C and D are some distributions and n is the number of data points. Assume that we can sample from C directly. Now define an asynchronous Gibbs sampler in which all workers transmit the values of their corresponding ν_i but never transmit θ. Consider a transmitted update from ν_j to ν_j'. Let q be the full conditional proposal distribution on the worker that sent ν_j', and assume that this worker transmits the parameters of that distribution along with ν_j'. Since q is a full conditional distribution, it does not depend on ν_j or the previous value of ν_j' on the transmitting worker. The MH acceptance probability takes the form

$$\min\left\{1, \frac{f(\theta, \nu_j' \mid y_j)\left[\prod_{i\neq j} f(\theta, \nu_i \mid y_i)\right] q(\nu_j)}{f(\theta, \nu_j \mid y_j)\left[\prod_{i\neq j} f(\theta, \nu_i \mid y_i)\right] q(\nu_j')}\right\} = \quad (4)$$

$$= \min\left\{1, \frac{f(\theta, \nu_j' \mid y_j)\, q(\nu_j)}{f(\theta, \nu_j \mid y_j)\, q(\nu_j')}\right\} \quad (5)$$

where f is the density of the full conditional distribution in question and q is the Hastings term. Thus we can carry out the evaluation using only one data point.

If data is stored in a distributed fashion and y_j is not available on other workers, we can transmit it over network along with ν_j'. If ν_j is also not available on other workers, but the latent variables ν_j form a non-overlapping partition among the workers, then we can transmit (ν_j', ν_j, y_j, q), because ν_j can only be updated on other workers through communication. This situation occurs in some problems where parameters – such as θ in Equation (2) – that are located at the top of a hierarchical model may depend on ν_j only through sufficient statistics, and where storing ν_j for all j on every worker is thus unnecessary.

These details illustrate the flexibility that asynchronous Gibbs sampling gives the user in handling large distributed data sets. They also show that implementing exact asynchronous Gibbs comes with substantial additional communication costs. If the data points y_j are sufficiently large, transmitting them may be too expensive. As a compromise, we instead recommend computing and storing the MH ratios at random with small probability, and using them as a convergence diagnostic. Remarkably, we find for many models the MH ratio is close enough to 1 sufficiently often that the MH correction does nothing the vast majority of the time, and so the approximate algorithm yields good numerical results. This is not always the case: a target distribution where this fails is showcased in Section 3.3. We now illustrate the algorithm on a set of examples.

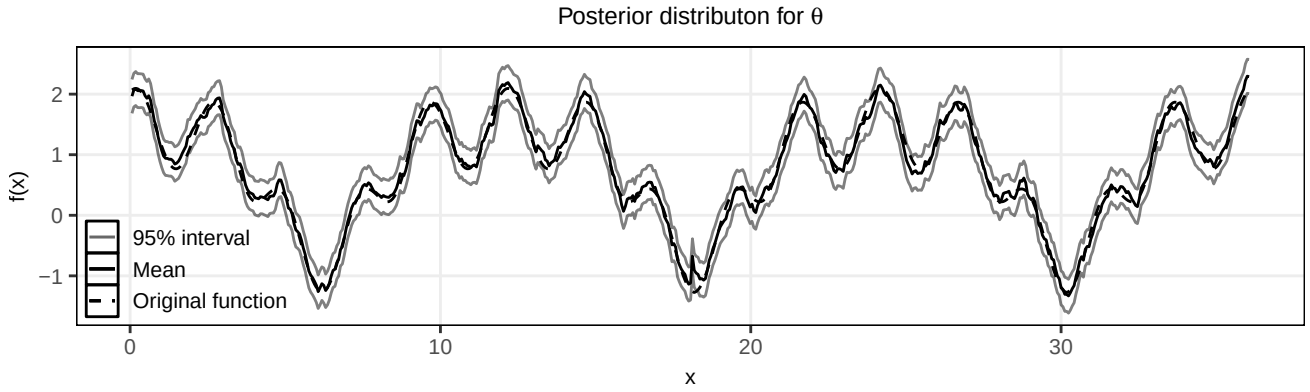

Figure 2: Partial subset of $\boldsymbol{\theta}$ for two workers, split at center, in the Gaussian process example of Section 3.1.

3 Examples

3.1 Gaussian process regression: a highly simplified spatial model with $n = 71,500$

In this example, originally due to Neal (1997), we used the algorithm to sample from the posterior distribution arising from a simple Gaussian process regression problem. This example is far too simple for use in a real spatial Bayesian learning problem – rather, we present it as a way to study how approximate asynchronous Gibbs sampling can be used for computation at scale. Our goal was to reconstruct the function

$$\tilde{f}(x) = 0.3 + 0.4x + 0.4\sin(2.7x) + \frac{1.1}{1+x^2} \quad (6)$$

defined for $x \in [-3, 3]$, and reflected and copied around the lines $x = 3, 9, ..,$ and $x = -3, -9, ..,$ in such a way that $\tilde{f}(x)$ becomes periodic with period 6 and is continuous everywhere. To simplify our example, we assumed that our data lives on a grid with equal spacing of 0.06 (i.e., $x_1 = 0, x_2 = 0.06, x_3 = -0.06, ..$). To generate the data, we added Gaussian white noise with standard deviation 0.2. Our model for reconstructing this function is

$$y_i = f(x_i) + \varepsilon_i \quad f(x_i) \sim \mathrm{GP}(0, k) \quad \varepsilon_i \sim \mathrm{N}(0, \sigma^2). \quad (7)$$

Here $i = 1, .., n =$ 71,500 with x on [−2,145, 2,145). For simplicity, we selected a Gaussian process with constant mean function μ and exponential covariance function $k(x, x') = \tau^2 \exp(-\phi|x - x'|)$ with hyperiors

$$\mu \sim \mathrm{N}(a_\mu, b_\mu) \quad \sigma^2 \sim \mathrm{IG}(a_\sigma, b_\sigma) \quad \tau^2 \sim \mathrm{IG}(a_\tau, b_\tau).$$

By introducing latent variables θ_i corresponding to each data point, the model can be expressed as

$$y_i \mid \theta_i, \sigma^2 \sim \mathrm{N}(\theta_i, \sigma^2) \qquad \boldsymbol{\theta} \sim \mathrm{N}(\mu\mathbf{1}, \mathbf{K}(\phi)) \quad (8)$$

where $K_{ij}(\phi) = k(x_i, x_j)$. By conjugacy, this yields inverse gamma posteriors for σ^2 and τ^2, a Gaussian posterior for μ, and a multivariate Gaussian posterior for $\boldsymbol{\theta}$. Since ϕ is non-conjugate and unidentifiable in the presence of τ^2, to simplify our example we fixed it at 0.5, which gives an interpretable length scale for the given problem.

If n is large, block sampling from this posterior is intractable because it requires the frequent inversion of two $(n \times n)$ matrices. It is possible to integrate $\boldsymbol{\theta}$ out of the model, but this does not avoid large matrix inversion. To avoid these difficulties and focus attention on those aspects of the computational problem most relevant to asynchronous Gibbs sampling, we use the assumption of an evenly-spaced grid together with special properties of the exponential covariance to develop a scheme for approximate closed-form analytic matrix inversion. Details may be found in Appendix B

With standard Gibbs, there are too many full conditionals to sample for the chain to produce useful output in reasonable time. Asynchronous Gibbs lets us parallelize this computation. In this example we used 143 workers with 1 CPU each. Each worker was responsible for 500 values of $\boldsymbol{\theta}$, each different from those handled by the other workers, and for (μ, σ^2, τ^2). We started the algorithm from low-probability initial values $\mu = 10, \sigma^2 = 10, \tau^2 = 10, \boldsymbol{\theta} = \mathbf{0}$. The algorithm converged rapidly, producing approximately 10,000 samples per worker in around 20 minutes.

In Figure 2 we plot a slice of the data, together with the correct solution. As noted above, our matrix inversion approximation scheme is inaccurate around the edges of each slice of $\boldsymbol{\theta}$ – this can be seen in the middle of Figure 2 – and hence these values are not as accurate as those elsewhere. The algorithm converged in an analogous fashion for all other slices of the data. We conclude that asynchronous Gibbs sampling produces reasonable output for the given large-scale Gaussian process model.

3.2 Mixed-effects regression: a complex hierarchical model with $n = 1,000,000$

The following model, due to von Brzeski et al. (2015), was used in a large-scale decision-theoretic analysis of product updates at eBay Inc. Because users choose when to update to the latest version of the product, analysis of product updates is done not by controlled experiment but by observational study, and causal inference is difficult. In particular, it is necessary to control for the *early-adopter effect*, in which the behavior of the response is correlated with how quickly a user adopts the treatment after release. To adjust for this effect, a Bayesian hierarchical mixed-effects regression model was selected. Since we are primarily interested in the computational aspects of this problem, we omit further discussion of the particular model and evaluation of its results – such discussion can be found in the original publication (von Brzeski et al., 2015).

A variety of different data sets have been used with this model – the data set that we employed consists of $n = 1,000,000$ users. The model can be written as

$$\boldsymbol{y}_i = \mathbf{F}_i\boldsymbol{\beta}_i + \mathbf{W}_i\boldsymbol{\gamma} + \boldsymbol{\varepsilon}_i \tag{9}$$

$$\boldsymbol{\beta}_i \mid \boldsymbol{\mu}, \boldsymbol{\Sigma} \sim \mathrm{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \tag{10}$$

$$\boldsymbol{\varepsilon}_i \mid \nu \sim \mathrm{N}(\mathbf{0}, \nu\,\mathbf{I}). \tag{11}$$

The data set consists of $\boldsymbol{y}_i : (T-p)\times 1$, $\mathbf{F}_i : (T-p)\times d$, and $\mathbf{W}_i : (T-p)\times(T-p)$. The parameters are $\boldsymbol{\beta}_i : (d\times 1)$, $\boldsymbol{\gamma} : (T-p)\times 1$, $\boldsymbol{\mu} : (d\times 1)$, $\boldsymbol{\Sigma} : (d\times d)$, and $\nu : (1\times 1)$, with the following priors:

$$\boldsymbol{\mu} \sim \mathrm{N}(\mathbf{0}, \kappa_\mu\mathbf{I}_d) \qquad \boldsymbol{\Sigma} \sim \mathrm{IW}(d+1, \mathbf{I}) \tag{12}$$

$$\boldsymbol{\gamma} \sim \mathrm{N}(\mathbf{0}, \kappa_\gamma\mathbf{I}_{T-p}) \qquad \nu \sim \mathrm{IG}(\epsilon/2, \epsilon/2). \tag{13}$$

Here $i = 1, .., n$ indexes individual data points (eBay users), $\boldsymbol{y}_i$ is a vector of values representing customer satisfaction for user i over time (aggregated to the weekly level), $\mathbf{F}_i$ and $\mathbf{W}_i$ are user-specific matrices of known constants (fixed effects), d is the length of the random-effects vector, $T = 52$ is the number of weeks of data for each user, p is the number of lags of autoregression in the model (typically no more than 5), and $\kappa_\mu, \kappa_\gamma, \epsilon$ are fixed hyperparameters.

The full conditionals for $\boldsymbol{\mu}$, $\boldsymbol{\gamma}$, $\boldsymbol{\Sigma}$, and ν involve the full conditional sufficient statistics

$$\bar{\boldsymbol{\beta}} = \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{\beta}_i \qquad \mathbf{S} = \sum_{i=1}^{n}(\boldsymbol{\beta}_i - \boldsymbol{\mu})(\boldsymbol{\beta}_i - \boldsymbol{\mu})^T \tag{14}$$

$$\boldsymbol{g} = \sum_{i=1}^{n}\mathbf{W}_i(\boldsymbol{y}_i - \mathbf{F}_i\boldsymbol{\beta}_i) \tag{15}$$

$$l = \sum_{i=1}^{n}(\boldsymbol{y}_i - \mathbf{F}_i\boldsymbol{\beta}_i - \mathbf{W}_i\boldsymbol{\gamma})^T(\boldsymbol{y}_i - \mathbf{F}_i\boldsymbol{\beta}_i - \mathbf{W}_i\boldsymbol{\gamma}) \tag{16}$$

which need to be calculated in a distributed setting and broadcast to all workers.

Approximate asynchronous Gibbs can enable this computation to be performed fully in parallel by an arbitrarily large cluster, while reducing synchronization costs and improving fault tolerance.

To avoid calculations over the full data, we maintain a *cache* of $\bar{\boldsymbol{\beta}}$, $\mathbf{S}$, $\boldsymbol{g}$ and l. To illustrate this, consider a new update of a single $\boldsymbol{\beta}_i$. When it is generated or received, the cache is updated by subtracting the portion of the sum corresponding to the old $\boldsymbol{\beta}_i$ and adding the portion corresponding to the new value. This significantly speeds up computation, but results in higher memory use.

Each worker updates $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}$, $\boldsymbol{\gamma}$, ν with the same probability as each individual element $\boldsymbol{\beta}_i$. With 20 workers and 1,000 iterations for each $\boldsymbol{\beta}_i$, the algorithm generates 20(1,000) = 20,000 total samples for each variable. This helps with mixing, improving accuracy.

For a fair performance comparison between approximate asynchronous Gibbs and standard Gibbs with multithreaded sampling of $\boldsymbol{\beta}_{i=1,..,n}$, we implemented a simple sequential-scan Gibbs sampler in *Scala*, using the exact same multithreaded numerical routines as in our cluster sampler. We used a data set size $n = 1,000,000$, and ran for 1,000 Monte Carlo iterations per worker. Runtime for the sequential-scan and asynchronous Gibbs samplers can be seen in Table 1, from which we see that asynchronous Gibbs was much faster, and scaled effectively to 20 worker nodes.

Figure 3 gives the distribution of the MH acceptance probabilities. The probability of rejecting a random update is about 0.02, indicating that the behavior of the approximate algorithm is close to what the exact algorithm would have done – this diagnostic is developed further in Section 3.3. Both chains yielded similar diagnostic plots, which indicated issues with slow mixing. Overall, the sequential-scan Gibbs sampler and asynchronous Gibbs appeared to have produced similar results.

It took substantially longer for ν to reach equilibrium with the asynchronous Gibbs sampler: this is a result of caching. Before we implemented caching, the

Algorithm	Workers	Threads	Runtime
Sequential-scan	1	8	12 hours
Asynchronous	20	160	1 hour

Table 1: Runtime and degree of parallelism for 1000 Monte Carlo iterations per worker, in the hierarchical mixed-effects regression example of Section 3.2.

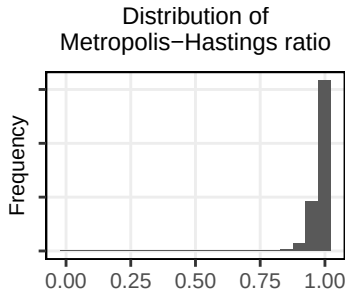

Figure 3: Distribution of MH acceptance probability, in the hierarchical mixed-effects regression example of Section 3.2.

asynchronous Gibbs trace plot for ν looked similar to the sequential-scan trace plot, but the algorithm ran substantially slower due to time spent computing the relevant sum. Trace plots for ν can be seen in Figure 6 in Appendix C. Note also that caching helps to ensure that all variables take a similar amount of time to sample, which is needed to ensure that the asynchronous chain is time-homogeneous – an assumption of our analysis in Appendix A.

To evaluate whether the output produced by asynchronous Gibbs was meaningful, we compared it to the output produced by standard sequential-scan Gibbs sampling. Both algorithms mix poorly, but produce similar distributional estimates for $\boldsymbol{\mu}$ and $\boldsymbol{\gamma}$, which were the primary unknowns of interest – an important outcome in an unsupervised setting where cross-validation is not immediately available. Thus the output of asynchronous Gibbs sampling was sufficient for these purposes, and in this problem the benefits of parallelism outweighed the additional implementation complexity.

3.3 Jacobi sampling and approximate asynchronous Gibbs

We now illustrate one way that asynchronous Gibbs sampling without a Metropolis-Hastings correction can fail. This is example is originally due to Johnson et al. (2013) – we study it in the setting of exact asynchronous Gibbs, under simulated parallelism with fixed deterministic communication schemes. Let $\pi(\boldsymbol{x}) \sim \mathrm{N}(\mathbf{0}, \boldsymbol{\Sigma})$ be the target distribution.

Consider the following sampler with workers $(w_1, .., w_m)$, each of which updates one coordinate. Initialize arbitrary starting values and perform the following updates.

(1) Each w_i: update $x_{ii} \mid \boldsymbol{x}_{i,-i}$ in parallel.

(2) Each w_i: broadcast x_{ii} to all other workers.

Here, no Metropolis-Hastings step is performed and all transmitted updates are accepted. Johnson et al. (2013) has shown that this sampling scheme does not converge for all $\boldsymbol{\Sigma}$: it can diverge if the precision matrix $\boldsymbol{\Sigma}^{-1}$ is not diagonally dominant. In cases where it does converge, it's also possible for the algorithm to converge to the wrong target distribution. We call this algorithm *Jacobi sampling*, because the mean vector at each update is an iteration of the Jacobi algorithm for solving linear systems (Saad, 2003) – for the corresponding linear system, diagonal dominance suffices to ensure stability of the iterations.

We analyze the following target with $m = 8$

$$\boldsymbol{\Sigma}^{-1} = 1 + 0.01\mathbf{I} \qquad \boldsymbol{\Sigma} = \begin{cases} 87.5 & i = j \\ -12.5 & i \neq j. \end{cases} \tag{17}$$

which we call the *near-singular* covariance. This is clearly a difficult target from the parallel sampling perspective, due to strong dependence between components. Here, Jacobi sampling with a target that has the near-singular covariance matrix of Equation (17) diverges. For comparison, consider a correlated mean-zero 8-dimensional Gaussian, with unit exponential covariance $\Sigma_{ij} = \exp(-\phi|i - j|)$ with $\phi = 0.5$.

To study the approximate algorithm for this target, we modify the communication scheme so that the approximate algorithm numerically converges to the incorrect target, rather than diverging. This ensures that the difference between approximate and exact algorithms is large enough to be interesting, but not so large that nothing can be said about it.

Suppose that there are 4 workers, each with 2 full conditionals assigned to them from our 8-dimensional Gaussian target. Each worker selects one of its full conditionals at random, performs a Gibbs step, and transmits the resulting draw to each other worker with probability 0.75. For the exact algorithm, the other workers then perform a Metropolis-Hastings calculation and either accept or reject the transmitted value.

We implemented both the exact and approximate versions of this variation with the near-singular covariance matrix on a single machine with simulated parallelism. For comparison, we also ran the variation with an exponential covariance matrix. Trace plots are given in Figure 4. Clearly the algorithm does far better with the exponential covariance. The exact algorithm the near-singular covariance matrix mixes poorly, but ends up yielding a Monte Carlo mean and covariance matrix that are not too far away from the correct answer. The approximate algorithm with the near-singular covariance matrix roughly yields the correct sample mean, but vastly incorrect sample covariance.

To further understand the differences between the exact

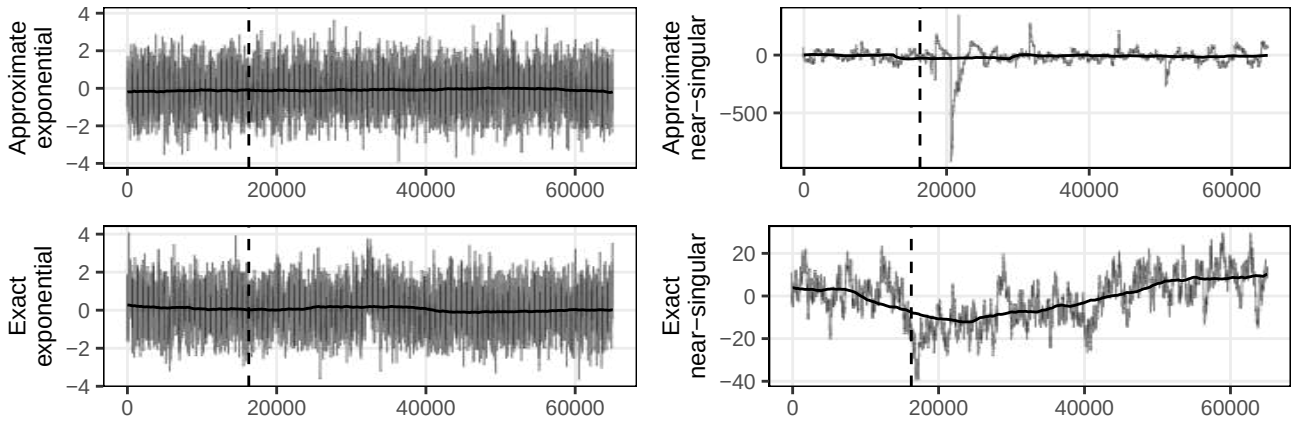

Figure 4: Trace plots for the first component of $\boldsymbol{x}$ for the exact and approximate variations of the asynchronous Gibbs sampler of Secton 3.3 with simulated parallelism, under exponential and near-singular covariance matrices.

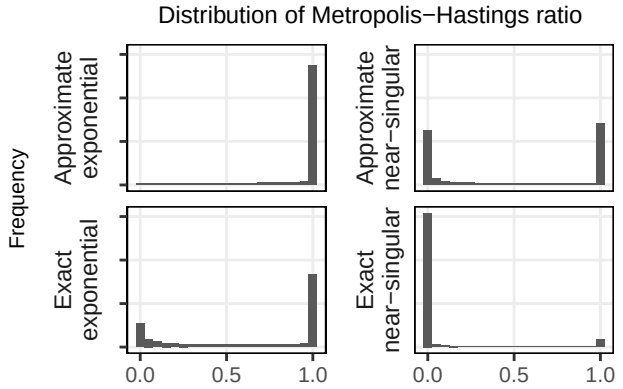

Figure 5: Distribution of the Metropolis-Hastings acceptance ratio for both approximate and exact asynchronous Gibbs, with the algorithm of Section 3.3, and exponential and near-singular covariance matrices.

and approximate algorithm, we examined the distribution of the MH acceptance ratios in all four examples – these are shown in Figure 5. In the case of the approximate algorithm this was accomplished by calculating and storing the MH probabilities and then ignoring them by accepting all updates. This distribution was concentrated around 1 for the approximate exponential case. It was substantially lower – bimodal near 0 and 1 – for the approximate Jacobi case that yielded the wrong answer. Interestingly, the MH ratio distributions were also different when comparing both exact algorithms to their approximate counterparts. This appears to be because the approximate chain undergoes *phase transition* in the sense of Diaconis (2011), making its behavior more akin to an optimization algorithm in those regions of the state space.

The intuition suggested by this example leads to the following diagnostic.

Diagnostic A. *Approximate asynchronous Gibbs is reasonable if the distribution of the Metropolis-Hastings acceptance ratio in the approximate algorithm is concentrated around 1.*

If the condition in Diagnostic A is satisfied, the behavior of the approximate algorithm will be similar to that of the exact algorithm in the posterior regions that it explores. Further work on approximate Markov chain theory is needed to formalize this intuition – see Section 4 for additional discussion. To conclude, we provide the following heuristic for describing problems in which this is likely to occur.

Heuristic B. *Asynchronous Gibbs without Metropolis-Hastings correction produces a good approximation to the exact algorithm if all of the following hold:*

(i) The target density π does not possess too much dependence between its components.

(ii) The dimensionality of π is significantly larger than the number of workers.

(iii) All transmitted variables are drawn via Gibbs steps.

We propose Heuristic B for the following reasons: (i) suggests that full conditional distributions in nearby posterior regions are similar, (ii) suggests that there is not too much movement happening at once, and (iii) suggests, given the previous two conditions, that the algorithm will consist of moves that are approximately Gibbs steps and hence should be accepted often.

Both Diagnostic A and Heuristic B are intuitive tools designed to help practitioners use approximate asynchronous Gibbs in situations where it is likely to work well. Future work is necessary to formalize these intuitions within the framework of Markov chain theory.

4 Discussion

Asynchronous Gibbs sampling can allow Bayesian learning to be effectively implemented in parallel and distributed environments, and has been a popular choice in the topic modeling community (Newman et al., 2009; Ihler and Newman, 2012). It can work well for models with a structure similar to the one found in the hierarchical mixed-effects regression example of Section 3.2 – in which each data point maps to a parameter – because the dependence in the posterior between almost all dimensions, for instance two vectors $\boldsymbol{\beta}_i, \boldsymbol{\beta}_j$ for $i \neq j$, is weak. Models with strong posterior dependence will likely remain difficult for any Gibbs-based algorithm, because even in the sequential case, we expect poor mixing in that context. One way around this would be to tailor blocking of the Gibbs sampler to the problem at hand. For example, performance in the Gaussian process model in Section 3.1 could be improved by considering an overlapping block scheme, such as the *additive Schwartz* method (Saad, 2003).

The theory of asynchronous Gibbs sampling can be further expanded. While we focused on convergence, it would be useful to quantify the degree to which asynchronous delays affect the performance of the algorithm. This is especially true for approximate asynchronous Gibbs – we have found, surprisingly, that reducing communication latency can in some cases make performance worse. The right amount of latency involves a balance: too much delay slows down mixing, but too little delay increases the bias introduced by ignoring the Metropolis-Hastings step.

Asynchronous Gibbs is not a Markov chain, which makes analysis non-trivial. However, we believe that a more detailed understanding of the interplay between the convergence behavior of the asynchronous Gibbs stochastic process and its dependence on past states will be a useful step toward developing *partially asynchronous* MCMC methods, which may mix better or possess other useful properties, and could potentially use asynchronous steps to hide latency during the global operations required for synchronization. This would mirror recent advances in massively parallel iterative algorithms for solving linear systems (Ghysels and Vanroose, 2014) and parameter-server-based distributed optimization (Ho et al., 2013).

Our analysis is largely complementary to the approach taken by De Sa et al. (2016). That work is based on assuming Dobrushin's condition (Pedersen, 2007), which limits their analysis to target distributions that do not exhibit too much dependence. In contrast, our approach depends on Assumption 10, which ensures all workers to converge to the target sufficiently quickly. Both perspectives are useful: further work is needed to connect the two approaches, perhaps weakening these regularity requirements in the process.

Further work is also needed in understanding the quality of the algorithm's output, for instance by extending the standard effective sample size calculation to multiple dependent chains. This would help verify the algorithm's output, particularly since Bayesian models are often used in unsupervised settings, such as our example in Section 3.2, where algorithm-independent approaches to evaluating model quality and uncertainty, such as cross-validation, are difficult to deploy.

Implementation of asynchronous Gibbs is specific both to the problem being solved and to the hardware used – in particular, it is necessary to decide how to divide the workload among all of the workers. We found that different choices produced widely different mixing efficiencies – in extreme cases, one worker can bottleneck the entire algorithm if it is sampling, at too slow a rate, a dimension upon which all other workers have strong dependence. Similar issues can occur with respect to network traffic control: if one worker is producing output too fast, it can flood the network, preventing other workers from communicating with each other. This is not solely an issue in complex problems – at one point in time, due to a default *Akka* configuration poorly suited to distributed computation, this difficulty manifested itself in a simple problem involving an 8-dimensional Gaussian. Thus care was required to properly tune the algorithm in the problems we studied.

Our implementation is nowhere near optimal. *Akka* is designed for large-scale distributed web applications rather than high-performance computing. This makes for convenient development, but does not yield the kind of low-level hardware control available in a framework such as *MPI*. Our cluster also was selected for convenience rather than performance – indeed, the machines we used were physically located in data centers in three different US states. This is an extremely high-latency environment from a high-performance computing perspective, and illustrates the algorithm's robustness.

These challenges are common to any nontrivial parallel computation scheme, where fully generic solutions are difficult. Here, we have focused on studying asynchronous Gibbs under a common class of big-data Bayesian problems, for which the number of latent variables grows with the number of data points. We find the results for hierarchical Bayesian models can mirror those of in LDA and topic modeling (Newman et al., 2009; Ihler and Newman, 2012), where the approach has long been popular. Our construction with exact asynchronous Gibbs provides a view on why this algorithm has been successful in these areas.

Acknowledgments

We are grateful to Chris Severs, Joseph Beynon, and Matt Gardner for many useful discussions and much help with the coding. DS (and perhaps to a lesser extent AT and DD) would like to thank Christian Robert for sending him an early version of this paper that allowed him, during an otherwise uneventful trip to buy trousers, to remember a strategy of proof for asynchronous algorithms. We are grateful to Matt Johnson for sending us the *Jacobi sampling* example and thereby demonstrating that approximate asynchronous Gibbs is not exact, the subsequent analysis of which made much of this work possible. We thank Christopher De Sa for sending us a different counterexample to exactness of the approximate algorithm on $\{0, 1\}^3$. We thank Murray Pollock for sending us an example demonstrating that violating time-homogeneity systematically can greatly reduce performance of the approximate algorithm. We are grateful to Cyril Chimisov for pointing out a subtle mathematical error in our original proof. We thank Herbie Lee for his ideas, which simplified the arguments used in our proof. We are grateful to Gareth Roberts, Murray Pollock, Radford Neal, Daniele Venturi, Rajarshi Guhaniyogi, Tatiana Xifara, and Torsten Erhardt for interesting and useful discussions. We also thank eBay, Inc. for providing the computational resources used for running our examples. Membership on this list does not imply agreement with the ideas given here, nor are any of these people responsible for any errors that may be present.

References

A. Banerjee, D. B. Dunson, and S. T. Tokdar. Efficient Gaussian process regression for large datasets. *Biometrika*, 100(1):75–89, 2012. Cited on page 15.

G. M. Baudet. Asynchronous iterative methods for multiprocessors. *Journal of the Association for Computing Machinery*, 25(2):226–244, 1978. Cited on pages 3, 11, 13, 14.

D. P. Bertsekas. Distributed asynchronous computation of fixed points. *Mathematical Programming*, 27(1):107–120, 1983. Cited on pages 3, 11, 13, 14.

D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3(1):993–1022, 2003. Cited on page 1.

C. De Sa, K. Olukotun, and C. Ré. Ensuring rapid mixing and low bias for asynchronous Gibbs sampling. In *International Conference on Machine Learning*, pages 1567–1576, 2016. Cited on pages 1, 2, 8.

P. Diaconis. The mathematics of mixing things up. *Journal of Statistical Physics*, 144(3):445–458, 2011. Cited on page 7.

M. Dow. Explicit inverses of Toeplitz and associated matrices. *Australian and New Zealand Industrial and Applied Mathematics Journal*, 44(E):185–215, 2003. Cited on page 15.

A. Frommer and D. B. Szyld. On asynchronous iterations. *Journal of Computational and Applied Mathematics*, 123(1):201–216, 2000. Cited on pages 3, 13, 14.

R. Furrer, M. G. Genton, and D. Nychka. Covariance tapering for interpolation of large spatial datasets. *Journal of Computational and Graphical Statistics*, 15(3):502–523, 2006. Cited on page 15.

S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, 1984. Cited on page 1.

P. Ghysels and W. Vanroose. Hiding global synchronization latency in the preconditioned conjugate gradient algorithm. *Parallel Computing*, 40(7):224–238, 2014. Cited on page 8.

J. Gonzalez, Y. Low, A. Gretton, and C. Guestrin. Parallel Gibbs sampling: from colored fields to thin junction trees. In *International Conference on Artificial Intelligence and Statistics*, pages 324–332, 2011. Cited on page 1.

W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970. Cited on page 1.

C. Hewitt. A universal modular actor formalism for artificial intelligence. In *International Joint Conference on Artificial Intelligence*, pages 235–245, 1973. Cited on page 2.

Q. Ho, J. Cipar, H. Cui, S. Lee, J. K. Kim, P. B. Gibbons, G. A. Gibson, G. Ganger, and E. P. Xing. More effective distributed machine learning via a stale synchronous parallel parameter server. In *Advances in Neural Information Processing Systems*, pages 1223–1231, 2013. Cited on pages 1, 8.

G. Hu and R. F. O'Connell. Analytical inversion of symmetric tridiagonal matrices. *Journal of Physics A: Mathematical and General*, 29(7):1511–1513, 1996. Cited on page 15.

A. Ihler and D. Newman. Understanding errors in approximate distributed latent Dirichlet allocation. *IEEE Transactions on Knowledge and Data Engi-*

neering, 24(5):952–960, 2012. Cited on pages 1, 2, 8.

M. Johnson, J. Saunderson, and A. Willsky. Analyzing Hogwild parallel Gaussian Gibbs sampling. In *Advances in Neural Information Processing Systems*, pages 2715–2723, 2013. Cited on pages 1, 6.

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953. Cited on page 1.

R. M. Neal. Monte Carlo implementation of Gaussian process models for Bayesian regression and classification. Technical report, University of Toronto, 1997. Cited on page 4.

W. Neiswanger, C. Wang, and E. Xing. Asymptotically exact, embarrassingly parallel MCMC. In *Uncertainty in Artificial Intelligence*, pages 623–632, 2013. Cited on page 1.

D. Newman, A. Asuncion, P. Smyth, and M. Welling. Distributed algorithms for topic models. *Journal of Machine Learning Research*, 10:1801–1828, 2009. Cited on pages 1, 8.

F. Niu, B. Recht, C. Re, and S. Wright. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011. Cited on page 1.

K. Pedersen. Dobrushin Conditions for Systematic Scan with Block Dynamics. In *Mathematical Foundations of Computer Science*, pages 264–275, 2007. Cited on pages 1, 8.

G. Roberts and J. Rosenthal. Coupling and ergodicity of adaptive Markov chain Monte Carlo algorithms. *Journal of Applied Probablity*, 44(2):458–475, 2007. Cited on page 13.

Y. Saad. *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, 2003. Cited on pages 6, 8.

M. L. Stein, Z. Chi, and L. J. Welty. Approximating likelihoods for large spatial data sets. *Journal of the Royal Statistical Society, Series B (Statistical Methodology)*, 66(2):275–296, 2004. Cited on page 15.

R. H. Swendsen and J.-S. Wang. Replica Monte Carlo simulation of spin-glasses. *Physical Review Letters*, 57(21):2607–2609, 1986. Cited on page 11.

A. Terenin, S. Dong, and D. Draper. GPU-accelerated Gibbs sampling: a case study of the horseshoe probit model. *Statistics and Computing*, 29(2):301–310, 2019. Cited on page 1.

A. Terenin and E. P. Xing. Techniques for proving asynchronous convergence results for Markov chain Monte Carlo methods. In *Workshop on Advances in Approximate Bayesian Inference, Neural Information Processing Systems*, 2017. Cited on pages 2, 11.

V. von Brzeski, M. Taddy, and D. Draper. Causal inference in repeated observational studies: a case study of eBay product releases. *arXiv:1509.03940*, 2015. Cited on page 5.

J. Wei, W. Dai, A. Qiao, Q. Ho, H. Cui, G. R. Ganger, P. B. Gibbons, G. A. Gibson, and E. P. Xing. Managed communication and consistency for fast data-parallel iterative analytics. In *ACM Symposium on Cloud Computing*, pages 381–394, 2015. Cited on page 1.