Bayes and Big Data: The Consensus Monte Carlo Algorithm

Steven L. Scott[1], Alexander W. Blocker[1], Fernando V. Bonassi[1], Hugh A. Chipman[2], Edward I. George[3], and Robert E. McCulloch[4]

[1]Google, Inc.
[2]Acadia University
[3]University of Pennsylvania, The Wharton School
[4]University of Chicago, Booth School of Business

October 31, 2013

Abstract

A useful definition of "big data" is data that is too big to comfortably process on a single machine, either because of processor, memory, or disk bottlenecks. Graphics processing units can alleviate the processor bottleneck, but memory or disk bottlenecks can only be eliminated by splitting data across multiple machines. Communication between large numbers of machines is expensive (regardless of the amount of data being communicated), so there is a need for algorithms that perform distributed approximate Bayesian analyses with minimal communication. Consensus Monte Carlo operates by running a separate Monte Carlo algorithm on each machine, and then averaging individual Monte Carlo draws across machines. Depending on the model, the resulting draws can be nearly indistinguishable from the draws that would have been obtained by running a single machine algorithm for a very long time. Examples of consensus Monte Carlo are shown for simple models where single-machine solutions are available, for large single-layer hierarchical models, and for Bayesian additive regression trees (BART).

1 Introduction

This article describes a method of performing approximate Monte Carlo simulation from a Bayesian posterior distribution based on very large data sets. When the data are too large for a single processor, the obvious solution is to divide them among multiple processors. There are two basic methods of computing on the divided data. The first is to divide the work among multiple cores on the same chip, either on a multi-core central processing unit (CPU), or on a massively parallel graphics processing unit (GPU). The multi-core approach can be extremely effective (Suchard, Wang, Chan, Frelinger, Cron, and West, 2010; Lee, Yao, Giles, Doucet, and Holmes, 2010), but it has two limitations. The first is that multi-core computing cannot alleviate bottlenecks related to memory or disk. The second issue is programming. The multi-threaded code necessary for multi-core computing can be difficult to write, even for expert programmers, because it is subject to race conditions that are poorly understood and difficult to debug. GPU programming in particular

requires very low level memory management that is difficult to abstract to higher level programming languages.

The alternative to multi-core computing is multi-machine computing, where data are divided among many different machines. Multi-machine computing provides scalable memory, disk, and processing power, so it can eliminate bottlenecks in all three traditional computing resources. However, multi-machine computing comes with a very high cost (in terms of efficiency) of communicating between machines. The multi-machine and multi-core approaches are complementary. If a multi-core solution is available it can be embedded in a multi-machine strategy, making each machine faster.

The primary difference between the multi-machine and multi-core computing models is communication, which is fast in multi-core systems and slow in multi-machine systems. Different communication costs imply that different algorithms are necessary for multi-core and multi-machine environments. On a multi-core system one can effectively parallelize a sequential algorithm by distributing the work required to implement an expensive step. For example, Suchard *et al.* (2010) achieve substantial speed increases by implementing a data augmentation algorithm on a GPU. Their algorithm is a standard data augmentation with the computationally intensive loop over the data done in parallel. Section 2 of this article presents a case study demonstrating the inefficiency of that algorithm in a multi-machine environment. It illustrates a fact which is well known to parallel computing experts, but often surprising to novices: passing messages among a large number of machines is expensive, regardless of the size of the messages being passed.

For Monte Carlo algorithms to succeed in a multi-machine environment, they must avoid regular communication between machines. Consensus Monte Carlo attacks the communication problem by dividing the data across multiple machines, with each machine independently sampling from the posterior distribution given its data. Posterior draws from each machine are then combined to form a consensus, system-wide belief about the model unknowns. The consensus Monte Carlo algorithm is embarrassingly parallel, as defined by the mapreduce framework (Dean and Ghemawat, 2008), so it can be run on virtually any system for parallel computing, including Hadoop (White, 2012), multi-core systems, or networks of workstations (Anderson *et al.*, 1995).

Consensus Monte Carlo is one of several attempts to scale traditional statistical computation. Zhang, Duchi, and Wainwright (2012), and references therein, describe the literature on averaging frequentist point estimates for parameters of statistical models. Guha, Kidwell, Hafen, and Cleveland (2009) discuss distributed methods of visualizing large data sets. Kleiner, Talwalkar, Sarkar, and Jordan (2011) extend the bootstrap to distributed data with the "bag of little bootstraps." To the extent that bootstrap samples can be viewed as approximating posterior samples, this work can be considered an alternative to the work presented here. Returning to Bayesian inference, Huang and Gelman (2005) proposed a consensus Monte Carlo approach that is similar to ours, but with a different rule for combining posterior draws. For certain classes of models (and certain classes

of model summaries), Bayesian inference can be conducted without Monte Carlo through clever approximations such as INLA (Rue, Martino, and Chopin, 2009), or variational Bayes (e.g. Hinton and Van Camp, 1993; Ghahramani and Beal, 2001; Jaakkola and Jordan, 2000). These approaches are effective, but there can be reasons to prefer Monte Carlo. For example, the computational cost of INLA is exponential in the dimension of the parameter space (or hyperparameter space in the case of hierarchical models), and the approximate posteriors delivered by variational Bayes give good approximations to individual marginal distributions, but not to the joint distribution as a whole. Posterior simulation by Monte Carlo is considerably more expensive than deterministic approximation algorithms, but it delivers the full posterior distribution. Although the approach presented here is limited to continuous parameter spaces, Monte Carlo methods can generally be applied to arbitrarily complex models.

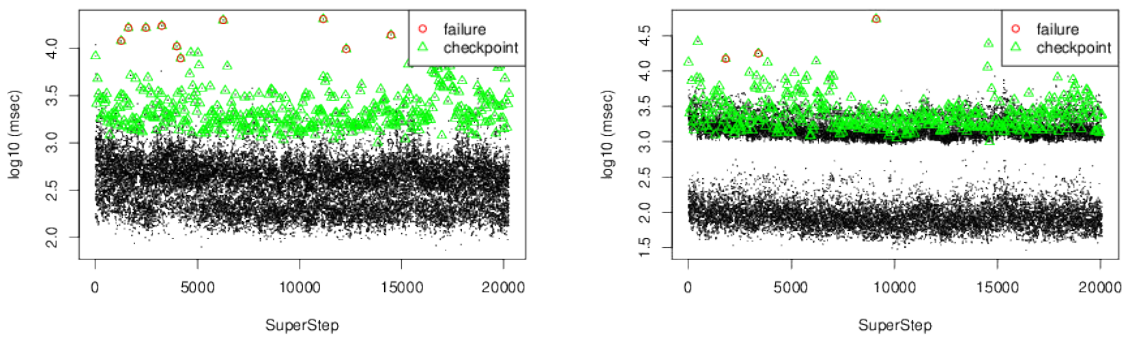
The remainder of the article is structured as follows. Section 2 contains the case study, mentioned above, illustrating the high cost of communicating between processes in a multi-machine environment. Section 3 describes the consensus Monte Carlo algorithm, and then Section 4 provides a series of examples illustrating consensus Monte Carlo on a variety of simulated and real data sets. Section 5 is a concluding discussion.

2 Coordinating many machines is expensive

This Section presents timings from a multi-machine MCMC algorithm, with the aim of illustrating just how expensive multi-machine communication can be. We focus on a single layer hierarchical logistic regression model

$$\begin{aligned} y_{ij} &\sim \text{Binomial}(n_{ij}, p_{ij}) \\ \text{logit}(p_{ij}) &= \mathbf{x}_{ij}^T \beta_i \\ \beta_i &\sim \mathcal{N}(\mu, \Sigma) \\ \mu|\Sigma &\sim \mathcal{N}(0, \Sigma/\kappa) \\ \Sigma^{-1} &\sim W(I, \nu), \end{aligned} \tag{1}$$

where $W(I, \nu)$ is the Wishart distribution with sum of squares matrix I and scale parameter ν. The precise numerical values of ν and κ used in the hyperprior are unimportant for this example. This model was applied to an internet advertising data set, where y_{ij} is the number of clicks on an advertisement with characteristics $\mathbf{x}_{ij}$ that was shown n_{ij} times. The hierarchy corresponds to different internet domains (e.g. `espn.com` vs. `nytimes.com`). There are slightly more than 600,000 domains in our data set, each with its own β_i. The vector $\mathbf{x}_{ij}$ contains an intercept and 7 dummy variables describing the color, font, and similar characteristics of the advertisement, so that the dimension of $\mathbf{x}_{ij}$ is 8. The number of distinct configurations of $\mathbf{x}_{ij}$ in each domain is small. It



(a) (b)

Figure 1: *Step times for the naive MCMC algorithm in Section 2 with (a) 500 and (b) 50 machines.*

is bounded above by $2^7 = 128$, but it is rarely more than a few dozen, and often just a single observation.

The obvious parallel MCMC algorithm for this model partitions the complete data by domain across a set of worker machines, and assigns one worker to be the "master node" responsible for the prior distribution. The algorithm then alternates between drawing each β_i given current values of μ and Σ, and then drawing μ and Σ given the set of β_i's. During the parallel data augmentation step, each machine simulates from $p(\beta_i|\mu, \Sigma, \mathbf{y})$ for its domains, and adds the draw to a set of low dimensional sufficient statistics $(\sum_i \beta_i, \sum_i \beta_i \beta_i^T)$. When a machine finishes drawing β_i for all of its domains, it sends its sufficient statistics to the master node, which accumulates sufficient statistics from different machines as they become available. Once all machines have reported, the master node draws μ and Σ, broadcasts them to the workers, and the cycle repeats. The algorithm was implemented in Pregel (Malewicz, Austern, Dehnert, Horn, Leiser, and Czajkowski, 2010), a system for distributed graph computing written by professional software engineers who are experts in parallel computation. The statistical computations were done in optimized C++ code.

There are potential issues with the adequacy of this model, the convergence rate of the MCMC algorithm, and the specific method used to sample the logistic regression coefficients. Set all these aside and focus on the amount of time required to perform an MCMC iteration. Figure 1 plots the time taken by each step in the algorithm from two separate runs with the same data on 500 and 50 machines. The draws of β_i and (μ, Σ) are separate "SuperSteps" in Pregel, so each panel in Figure 1 represents 10,000 MCMC iterations, with two SuperSteps per MCMC iterate. The 500-machine run completed in 2.75 hours, which is very close to 1 iteration per second. The 50-machine run completed in around 5 hours. Notice that a ten-fold reduction in computing resources only produced a two-fold increase in compute time, so there is at least a factor of 5 inefficiency at play.

Job failures are a fact of life in multi-machine computing and are often discussed as a source

of delay. Jobs can fail either because of faulty hardware or because of evictions in favor of higher priority jobs. To protect against job failures, Pregel periodically saves its state so that it can recover from a failure without completely restarting. The state-saving iterations are highlighted with triangles in Figure 1. Both runs lose about the same amount of time saving state. The iterations containing job failures are highlighted with circles. Each recovery from a failure takes a few tens of seconds. The 500 machine run experienced more failures, but not enough to materially factor into the efficiency difference between the two runs.

The source of the inefficiency can be seen in the lower of the two black bands in Figure 1. The bands represent even and odd numbered SuperSteps corresponding to the expensive β_i draws in the even steps and the cheap (μ, Σ) draws in the odd steps. As expected, increasing the number of machines leads to a decrease in the amount of time spent on the expensive step. The surprising thing is how much more expensive the "cheap" steps become. In the 50 machine run, the (μ, Σ) draw takes a median of roughly $10^2 = 100$ milliseconds. In the 500 machine run it takes roughly $10^{2.4} \approx 250$ milliseconds. Logging in the C++ code shows that the actual time spent drawing μ and Σ is less than 1 millisecond, with the rest lost to communication overhead. The inefficiency is not coming from rare, catastrophic machine failures, but the consistently higher communication cost in iterations where no failures occurred. Ironically, CPU was the limiting resource in the single machine version of this problem, but on a relative scale we wound up devoting almost no CPU time to our job!

We wish to emphasize that time is not being lost in what one might think are the obvious places: an imbalanced workload among the machines, inhomogeneous hardware, an inefficient broadcast of the parameters, or reduction of the sufficient statistics, or a long transmission time on the network. All these possibilities were investigated and found to be performing efficiently. The problem is that we are communicating between machines at all. There is simply an overhead involved in getting a set of machines ready to send and receive messages, even if those machines are dedicated to your job. The cost is high enough that, in this case, a 500 machine job could achieve at most 4 iterations per second, on average, even if the machines had no data to process. The clear lesson is that if Bayesian posterior simulation is to be distributed across multiple machines, it must be done in a way that uses minimal communication.

3 Consensus Monte Carlo

The idea behind consensus Monte Carlo is to break the data into groups (called "shards"), give each shard to a worker machine which does a full Monte Carlo simulation from a posterior distribution given its own data, and then combine the posterior simulations from each worker to produce a set of global draws representing the consensus belief among all the workers. Let $\mathbf{y}$ represent the full data, let $\mathbf{y}_s$ denote shard s, and let θ denote the model parameters. For models with the appropriate

independence structure the system can be written

$$p(\theta|\mathbf{y}) \propto \prod_{s=1}^{S} p(\mathbf{y}_s|\theta)p(\theta)^{1/S}. \tag{2}$$

Notice that the prior distribution $p(\theta) = \prod_s p(\theta)^{1/S}$ is broken into S components to preserve the total amount of prior information in the system. Equation (2) assumes that batches of observations are conditionally independent across shards, given parameters, but it allows for arbitrary dependence within the elements of $\mathbf{y}_s$. Thus, for example, it can be used with hierarchical models with hyperparameters θ, as long as data in a single level of the hierarchy are not split across two different machines.

If each worker returned its posterior distribution as a mathematical function, then the worker-level distributions could be multiplied to form the overall "consensus" posterior given the complete data. The difficulty is that the workers-level distributions are reported as Monte Carlo draws, so we need a way of combining the draws.

3.1 Combining draws by weighted averages

Suppose worker s generates draws $\theta_{s1}, \ldots, \theta_{sG}$ from $p(\theta|\mathbf{y}_s) \propto p(\mathbf{y}_s|\theta)p(\theta)^{1/S}$. One way to combine draws across workers is to simply average them. Suppose each worker is assigned a weight, representable as a matrix W_s. The consensus posterior for draw g is

$$\theta_g = \left(\sum_s W_s\right)^{-1} \sum_s W_s \theta_{sg}. \tag{3}$$

When each $p(\theta|\mathbf{y}_s)$ is Gaussian, the joint posterior $p(\theta|\mathbf{y})$ is also Gaussian, and equation (3) can be made to yield exact draws from $p(\theta|\mathbf{y})$. To see this, suppose $S = 2$, and that $\theta|\mathbf{y}_1 \sim \mathcal{N}(\mu_1, \Sigma_1)$, and $\theta|\mathbf{y}_2 \sim \mathcal{N}(\mu_2, \Sigma_2)$. Then standard Bayesian calculations for the normal distribution give

$$\begin{aligned} p(\theta|\mathbf{y}) &\propto p(\theta|\mathbf{y}_1)p(\theta|\mathbf{y}_2) \\ &\propto \mathcal{N}(\theta|\tilde{\mu}, V) \end{aligned} \tag{4}$$

where $V^{-1} = \Sigma_1^{-1} + \Sigma_2^{-1}$ and $\tilde{\mu} = V(\Sigma_1^{-1}\mu_1 + \Sigma_2^{-1}\mu_2)$. Now let θ_1 be a draw from $\mathcal{N}(\mu_1, \Sigma_1)$ and let θ_2 be a draw from $\mathcal{N}(\mu_2, \Sigma_2)$. It is easily verified that

$$V\left(\Sigma_1^{-1}\theta_1 + \Sigma_2^{-1}\theta_2\right) \sim \mathcal{N}(\tilde{\mu}, V). \tag{5}$$

Taken together, equations (4) and (5) suggest the algorithm in Figure 2. Although the algorithm is exact only for Gaussian posteriors, there are two reasons to believe that it is broadly useful. The first is that under standard regularity conditions for asymptotic theory, posterior distributions tend

1. Divide $\mathbf{y}$ into shards $\mathbf{y}_1, \ldots, \mathbf{y}_S$.
2. Run S separate Monte Carlo algorithms to sample $\theta_{sg} \sim p(\theta|\mathbf{y}_s)$ for $g = 1, \ldots, G$, with each shard using the fractionated prior $p(\theta)^{1/S}$.
3. Combine the draws across shards using weighted averages: $\theta_g = \left(\sum_s W_s\right)^{-1} \left(\sum_s W_s \theta_{sg}\right)$.

Figure 2: *The consensus Monte Carlo algorithm.*

towards a Gaussian limit in large samples (Le Cam and Yang, 2000). The second is that we will see examples in Section 4 where applying the method to non-Gaussian posteriors (without theoretical justification) works surprisingly well.

3.2 Choosing Weights

The weight $W_s = \Sigma_s^{-1}$ is optimal (for Gaussian models), where $\Sigma_s = Var(\theta|\mathbf{y}_s)$. An obvious Monte Carlo estimate of Σ_s is sample variance of $\theta_{s1}, \ldots, \theta_{sG}$. If the dimension of θ is very large, or if the model is very simple, one may prefer to weight sub-optimally in order to simplify the computation. If θ is high dimensional then it may be preferable to ignore the covariances in Σ_s, and simply weight each scalar element of θ by the reciprocal of its marginal posterior variance.

In many "big data" problems the analyst is in control of the sharding process, so data can be sharded by randomly assigning each observation (or cluster of correlated observations) to a shard. One is tempted to argue that the posterior distributions in each shard will have roughly the same shape, and so the draws from different shards can be combined with equal weighting. However, large samples on each shard are necessary for the preceding argument to apply. In particular, each worker must have enough information to estimate all components of θ. Almost by definition, models that are complex enough to require very large data sets will have subsets of parameters that cannot be estimated on every shard. Section 4.3 provides an example where shard-level posterior distributions can be very different, even with IID data, because some observations are more informative than others. In practice, information-based weighting will usually be necessary.

3.3 Other potential consensus strategies

Strategies other than averaging could be used to form consensus posterior estimates. We focus on averages because they are simple and stable. For example, one could empirically estimate each worker-level posterior density using a kernel density estimate and then apply equation (2) directly. This approach might work well in small models, but as the dimension of θ increases the kernel density estimates will become unreliable.

Huang and Gelman (2005) suggest four alternative methods for consensus posteriors, all of which are variants of either an explicit normal approximation or importance resampling. The consensus

Monte Carlo method from Figure 2 is also rooted in normal theory, but it has the potential to capture non-normal features of the posterior such as fat tails or skewness that would be lost under an explicit normal approximation. We have not explored importance resampling methods because of the potential for importance resampling to collapse to a single point in high dimensions, and because of the potential difficulty in evaluating the likelihood $p(\mathbf{y}_s|\theta)$ in complex models, like hierarchical generalized linear models similar to equation (1). On the other hand, consensus distributions based on averaging are obviously limited to continuous parameter spaces, while importance resampling methods could be applied more generally (e.g. to mixtures, or regression models with spike-and-slab priors).

3.4 More complex models

3.4.1 Nested hierarchical models

If the data have nested structure, where $y_{ij} \sim f(y|\phi_j)$ and $\phi_j \sim p(\phi|\theta)$, then consensus Monte Carlo can be applied to the hyperparameters in a straightforward way. If the data are partitioned so that no group is split across multiple shards, then the model satisfies equation (2). One can run the consensus Monte Carlo algorithm, storing the draws of θ and discarding the draws of ϕ_j. Combining the draws of θ_{sg} based using empirical weights W_s estimated from the within-shard Monte Carlo variance produces a set of draws $\theta_1, \ldots, \theta_G$ approximating $p(\theta|\mathbf{y})$.

Conditional on the simulated draws of θ, sampling $\phi_j \sim p(\phi_j|\mathbf{y}, \theta) = p(\phi_j|\mathbf{y}_j, \theta)$ is an embarrassingly parallel problem. The simulated draws of θ can be broadcast back to the workers, which can simulate each ϕ_j in parallel. Each worker machine simulates ϕ_{jg} given the corresponding θ_g, with no need to communicate with other workers. Because $\theta \sim p(\theta|\mathbf{y})$ the second set of ϕ_j draws will have marginal distribution $p(\phi_j|\mathbf{y})$, whereas the first, discarded set follows $p(\phi_j|\mathbf{y}_s)$.

This procedure can be recursively applied as needed if the data are nested more than two levels deep.

3.4.2 Nonparametric regression

In a nonparametric regression model $y \sim \mathcal{N}\left(f(\mathbf{x}), \sigma^2\right)$ the goal is to compute $p(f(\mathbf{x})|\mathbf{y})$ where the form of f is unknown. Many nonparametric models either have no parameters to average or else the "model parameters" are non-numerical constructs like decision trees (see Denison, Mallick, and Smith, 1998; Chipman, George, and McCulloch, 2010). However, the predictive distributions from these models are often nearly Gaussian (possibly after conditioning on latent variables), so consensus Monte Carlo can be used to combine their predictive distributions at particular values of $\mathbf{x}$. Given training data $\mathbf{y}$ and a set of locations $\mathbf{x}_1, \ldots, \mathbf{x}_K$, where predictions are desired, one can run S separate Monte Carlo simulations to produce draws from $p(f(\mathbf{x}_k)|\mathbf{y}_s)$. At each $\mathbf{x}_k$, each worker's draws can be weighted by the inverse of the Monte Carlo estimate of the the within-

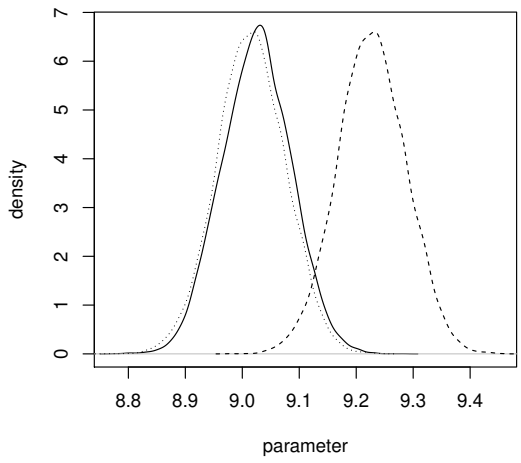


Figure 3: *Example of small sample bias. The solid line is the posterior distribution of μ^2 as described in Section 3.5. The dashed line is the consensus Monte Carlo estimate. The dotted line is the consensus Monte Carlo estimate with a jackknife bias correction applied.*

worker posterior variance $s_{sk}^2 = \sum_{g=1}^{G}(f_{sg}(\mathbf{x}_k) - \bar{f}_s(\mathbf{x}_k))^2/(G-1)$, and averaged to form a consensus posterior distribution.

3.5 The potential for small-sample bias

Oddly enough, small sample bias can play a role when analyzing large data sets. When the data are divided among many machines, bias that vanishes in the full data may still be present in each shard. Zhang, Duchi, and Wainwright (2012) discuss the issue of small sample bias in the context of frequentist point estimation. A similar effect can exist in Bayesian analysis, where the consensus posterior needs to be shifted because of an accumulation of small sample biases.

Figure 3 shows the effect of small sample bias in a toy simulation where $y_i \sim \mathcal{N}(\mu, 1)$, with $p(\mu) \propto 1$, where the parameter of interest is μ^2. The Figure is based on a simulation of $n = 10,000$ observations from $y \sim \mathcal{N}(3, 1)$, with 10 equally weighted shards. The solid line in the Figure is the posterior distribution based on all $10,000$ observations, while the dashed line is the consensus Monte Carlo estimate.

Not all problems will exhibit meaningful small sample bias. In some cases the shard-level models will have sufficiently large sample sizes that small-sample bias is not an issue. In others, models can be unbiased even in small samples. When small sample bias is a concern jackknife bias correction can be used to mitigate it through subsampling. The idea behind jackknife bias correction is to shift the draws from a distribution by a specified amount B determined by subsampling. Suppose $E(\theta|\mathbf{y}) = \theta + B/n$, so that the posterior mean is a biased estimate of θ. On each worker machine,

take a subsample of size αn, where $0 < \alpha < 1$. Let $\mathbf{y}_{\text{sub}}$ denote the resulting distributed subsample, and construct consensus Monte Carlo draws from $p(\theta|\mathbf{y}_{\text{sub}})$. Then $E(\theta|\mathbf{y}_{\text{sub}}) = \theta + \frac{B}{\alpha n}$. Simple algebra gives an estimate of the bias term

$$\frac{B}{n} \approx (E(\theta|\mathbf{y}_{\text{sub}}) - E(\theta|\mathbf{y})) \left(\frac{\alpha}{1-\alpha}\right).$$

Subtracting the estimated value of B/n from each draw in the original consensus distribution gives a set of first order bias corrected draws. The dotted line in Figure 3 shows the effect of the bias correction in our example.

4 Examples

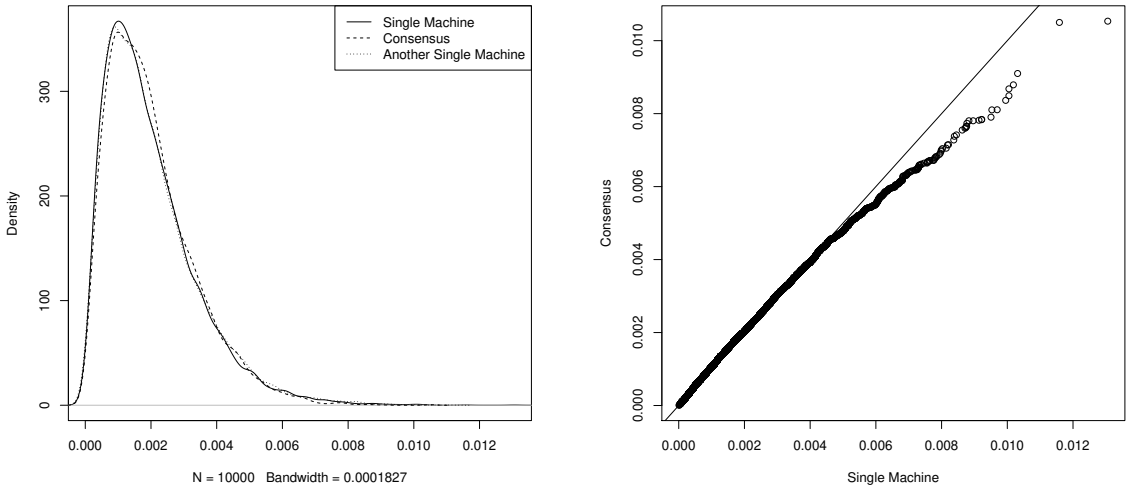
This section proceeds through a set of examples illustrating the consensus Monte Carlo algorithm. In each case, the goal is to see how the algorithm compares to a single machine algorithm run on the same data. Thus the examples will have to be simple enough (or else the data small enough) for a single machine run to be possible.

4.1 Binomial data with a beta prior

Figure 4(a) shows three Monte Carlo approximations to the posterior distribution of a binomial success probability. The data are 1000 Bernoulli outcomes with a single success, and the assumed prior is uniform. Two of the distributions in Figure 4 are made by directly drawing from the $Be(2, 1001)$ distribution. The distribution marked "Consensus" was constructed by distributing the data across 100 independent Monte Carlo samplers, each with a $Be(.01, .01)$ prior distribution. One of the samplers is assigned the shard of data containing 9 failures and the single success. The remaining 99 samplers each observe 10 failures. The posterior draws were then combined using an unweighted average. This example is interesting because even though the posterior is highly non-Gaussian, consensus Monte Carlo does an effective job capturing the distribution. Figure 4(b) highlights that the approximation is imperfect, in that it slightly under-weights the tails.

This example also illustrates that the procedure can be sensitive to the scale of the analysis. The shard-level prior $Be(.01, .01)$ is not the same as the full $Be(1, 1)$ prior raised to the .01 power. If the 100 workers had each been given the uniform prior then the estimated distribution would be seriously incorrect, as shown by Figure 5(a). The prior is very important because only a single success was observed. Giving each worker an additional "prior success" corresponding to the uniform prior adds a substantial amount of prior information to the system. The influence of the prior, and thus the need for prior adjustment, becomes less dramatic when the data are more informative.

Figure 5(b) shows a different simulation where the workers are given different amounts of work



(a) (b)

Figure 4: *(a) Posterior draws from binomial data. (b) A qq plot showing that the tails of the consensus Monte Carlo distribution in panel (a) are slightly too light.*

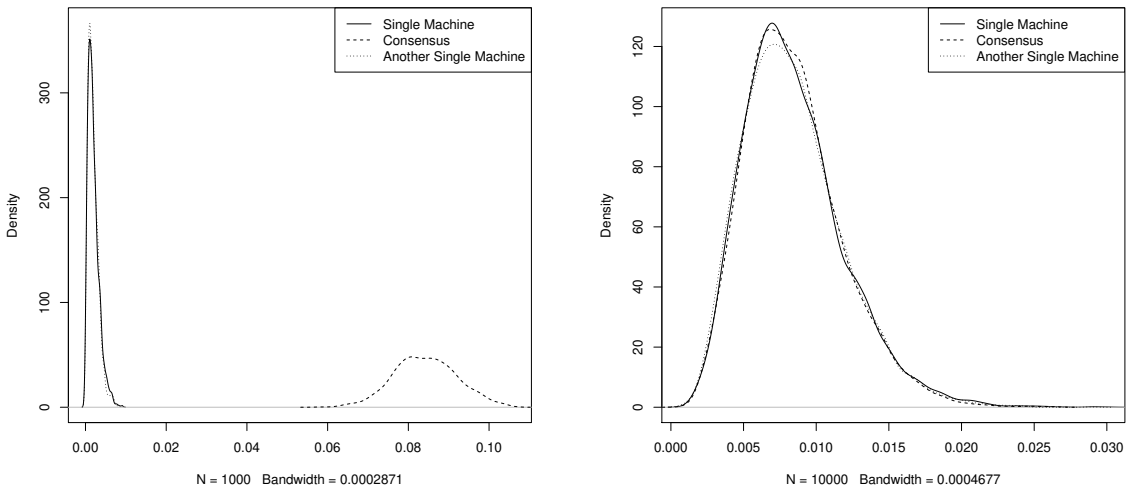
to do. There were 5 workers each assigned 100, 20, 20, 70, and 500 binomial observations. The data were simulated from the binomial distribution with success probability .01. The workers were each given a $Be(1/5, 1/5)$ prior to match the $Be(1, 1)$ uniform prior used in the single machine analysis. This distribution is slightly less skewed than the one in Figure 4, because the simulation produced more than one success. The distribution is still noticeably non-Gaussian, and the consensus Monte Carlo estimate is very similar to the two single machine analyses.

4.2 Gaussian Data

The next test case is a multivariate normal model with unknown mean and variance. The test case has $\mu = (1, 2, 3, 4, 5)$ and

$$\Sigma = \begin{pmatrix} 1.00 & 0.99 & 0.98 & 0.00 & -0.70 \\ 0.99 & 1.00 & 0.97 & 0.00 & -0.75 \\ 0.98 & 0.97 & 1.00 & 0.00 & -0.60 \\ 0.00 & 0.00 & 0.00 & 1.00 & 0.00 \\ -0.70 & -0.75 & -0.60 & 0.00 & 1.00 \end{pmatrix}.$$

The model includes three deviates that are highly correlated with one another, one that is independent of everything else, and one that is anticorrelated with the first three. We consider a problem with 100 workers and examine how consensus Monte Carlo handles draws of Σ. We focus on Σ because the marginal posterior of μ is multivariate T, which is sufficiently close to normal for the



(a) (b)

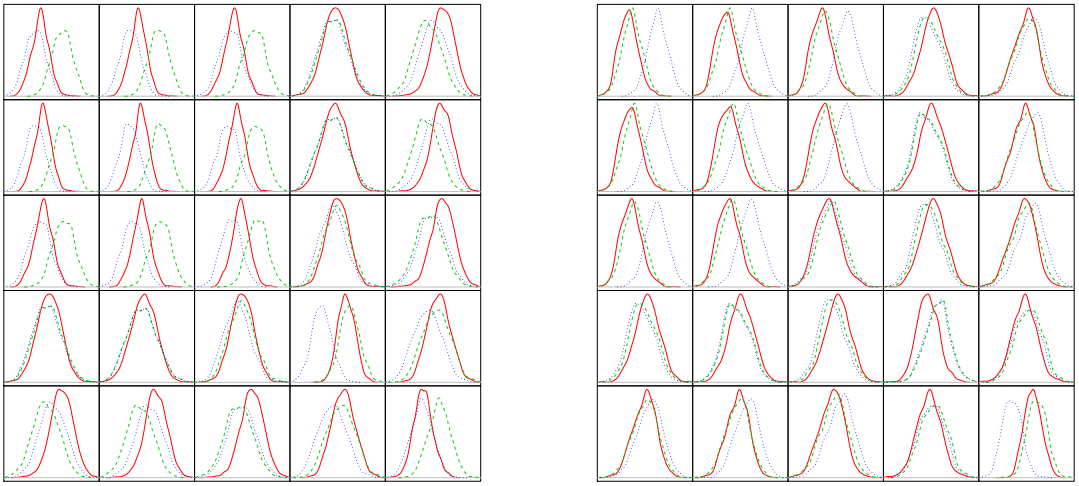
Figure 5: *The consensus Monte Carlo distribution (a) performs badly when each worker receives a uniform prior, and (b) performs well with imbalanced shards, when properly weighted.*

consensus draws to be nearly exact.

Figure 6 shows posterior draws of Σ based on simulated data with 50 observations per worker. The scale is different in each of the panels (which are arrayed as the elements of Σ) to maximally highlight differences in the consensus and single-machine draws. When viewed on the same scale, the differences between parameter values are far more important than the differences between methods. The consensus Monte Carlo estimates (shown as green dashed lines) of the posterior appear to have a bias away from zero when compared to the single machine algorithm (the solid lines).

The dotted blue lines are a jackknife bias corrected density estimate based on a 20% subsample. With 50 observations per worker, a 20% subsample is 10 observations. When paired with corrected prior distribution (obtained by dividing both the prior sum of squares and prior sample size by S), a much smaller sub-sample would lead to an improper posterior. The bias correction generally seems to help in Figure 6(a), though for the variance term in element 4 it provides an adjustment were none is necessary.

Interestingly, when we double the sample size to 100 in Figure 6(b) the unadjusted consensus Monte Carlo estimates match the single machine algorithm quite closely, while the jackknife bias adjustment winds up introducing a bias towards zero similar to the one it removed in Figure 6(a). We tried one more run with 1000 observations per worker, which we chose not to display because all distributions matched sufficiently closely that it was difficult to visually distinguish between them.



(a) (b)

Figure 6: *(a) Posterior distribution of* Σ *based on 100 workers, with 50 observations per worker. Red (solid) line is the single machine algorithm. Green (dashed) line is the consensus Monte Carlo estimate. Blue (dotted) line is the bias corrected consensus estimate. (b) Same, but with 100 observations per worker. With 1000 observations per worker all plots overlap essentially perfectly.*

4.3 Logistic regression

Logistic regression is a widely used model for internet data because of the interest in whether a particular stimulus results in a user action such as a click, mouse-over, or similar observable event. Table 1(a) is a set of simulated data based on a hypothetical set of binary predictors that occur independently with the frequencies shown in Table 1(b). The first variable is an intercept, and the last is a rarely occurring variable that is highly predictive of an event when it occurs.

Figure 7(a) shows the posterior distribution of the coefficients in a logistic regression of y on x_1 through x_5. The single machine "overall" MCMC estimate is compared to consensus Monte Carlo estimates under three different weighting schemes. The "matrix" scheme weights each worker's draws using the inverse of the sample variance matrix for β draws from that worker. The "scalar" weighting scheme weights each individual component of β by the inverse of its scalar sample variance, ignoring covariances. The "equal" scheme simply averages all draws with equal weights.

Despite the fact that all 100 workers faced identical sampling distributions, the equal weighting scheme fails badly because the workers are only "equal" *a priori*. They don't see the same data. Out of 10,000 total observations, there are only 104 in which x_5 is active, and only 71 of those produce an event. Thus, many workers (roughly 1/3) have no information with which to measure the effect of x_5, and roughly another third have only a single observation, so a small subset of workers has all the information about this coefficient. The different subsets of workers can be seen

y	n	x_1	x_2	x_3	x_4	x_5
266	2755	1	0	0	1	0
116	2753	1	0	0	0	0
34	1186	1	0	1	0	0
190	717	1	1	0	1	0
61	1173	1	0	1	1	0
37	305	1	1	1	0	0
68	301	1	1	1	1	0
119	706	1	1	0	0	0
18	32	1	0	0	0	1
13	17	1	0	1	1	1
18	24	1	0	0	1	1
8	10	1	1	0	1	1
2	2	1	1	1	0	1
7	13	1	0	1	0	1
2	2	1	1	1	1	1
3	4	1	1	0	0	1

(a)

	x_1	x_2	x_3	x_4	x_5
frequency	1	.2	.3	.5	.01
coefficient	-3	1.2	-.5	.8	3

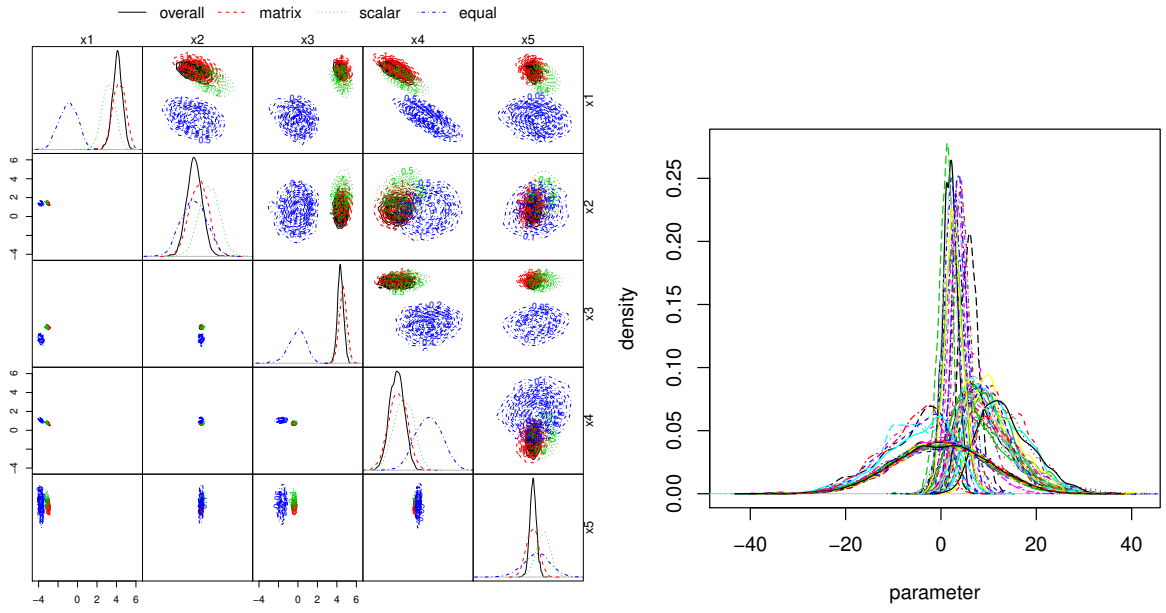
(b)

Table 1: *(a) Data for the logistic regression in Figure 7. (b) The probability that each variable is active, and the true logistic regression coefficients used in the simulation.*

in Figure 7(b). Those with no information about β_5 sample from the prior. Those that only get to see a single observation sample from one of the the skewed distributions, depending on whether the single observation corresponded to a success or a failure. The subset of informed workers able to see multiple observations sample from the nearly normal distributions centered around 3. The empirical weighting schemes ("matrix" and "scalar") are aware of the information asymmetry and can use it to place more weight on workers that have more information.

The information asymmetry is the main reason the equal weighting scheme misses so badly in Figure 7(a). The Figure also shows substantial agreement between the single-machine overall MCMC algorithm and the "matrix" and "scalar" weighting schemes. Both schemes do a good job capturing not only the location and spread of the marginal distribution of the coefficients, but also the correlations between coefficients. The "scalar" scheme tends to be over-dispersed relative to the "matrix" scheme, but not terribly so.

Figure 8 shows how the various schemes perform as the sample size grows to 1,000 and 10,000 observations per worker. Both the matrix and scalar weighting methods do about equally well combining the worker draws. The equal weighting scheme is not shown in these plots. It still missed badly with 1000 observations per worker, but it performs about as well as the other methods with 10,000 observations per worker. With so many observations, the differences between pairwise marginal distributions are vanishingly small when seen on a common scale.



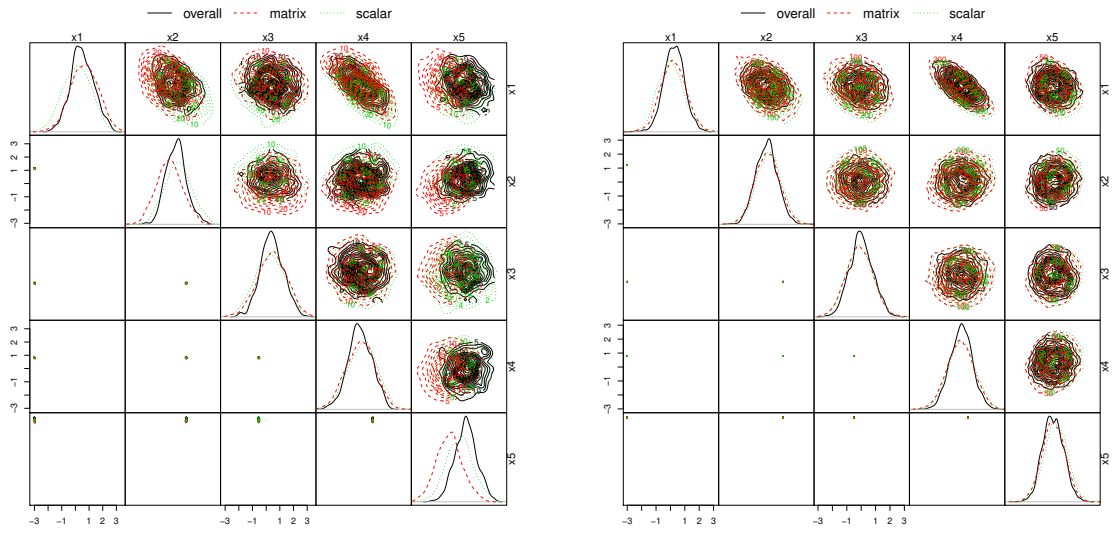
(a) (b)

Figure 7: *Logistic regression with 100 observations per worker. Panel (a) is a pairs plot comparing the single machine MCMC estimate of the posterior distribution with the consensus Monte Carlo estimate under three different weighting schemes. Plots on and above the diagonal show the distributions on a scale just large enough to fit the plots in each dimension. Plots below the diagonal are on a common scale. The diagonal shows the marginal distribution of each coefficient. Panel (b) shows the worker-level marginal distributions for* β_5.

4.4 Hierarchical models

We now consider fitting a hierarchical model to a large distributed data set of internet advertising data. The model is the hierarchical Poisson regression described in equation (6), where y_{ij} is the number of times advertisement i from advertiser j was clicked, E_{ij} is the number of times it was shown, and $\mathbf{x}_{ij}$ is a small set of predictor variables, including dummy variables describing the type of ad format the ad was shown in (e.g. part of a vertical vs. horizontal configuration, along the top vs. along the side, etc.), and a continuous "quality score" assigned by another model.

$$\begin{aligned} y_{ij} &\sim Po(E_{ij}\lambda_{ij}) \\ \log \lambda_{ij} &= \beta_j^T \mathbf{x}_{ij} \\ \beta_j &\sim \mathcal{N}(\mu, \Sigma) \\ \mu &\sim \mathcal{N}(0, \Sigma/\kappa) \\ \Sigma^{-1} &\sim W(I, \nu) \end{aligned} \tag{6}$$



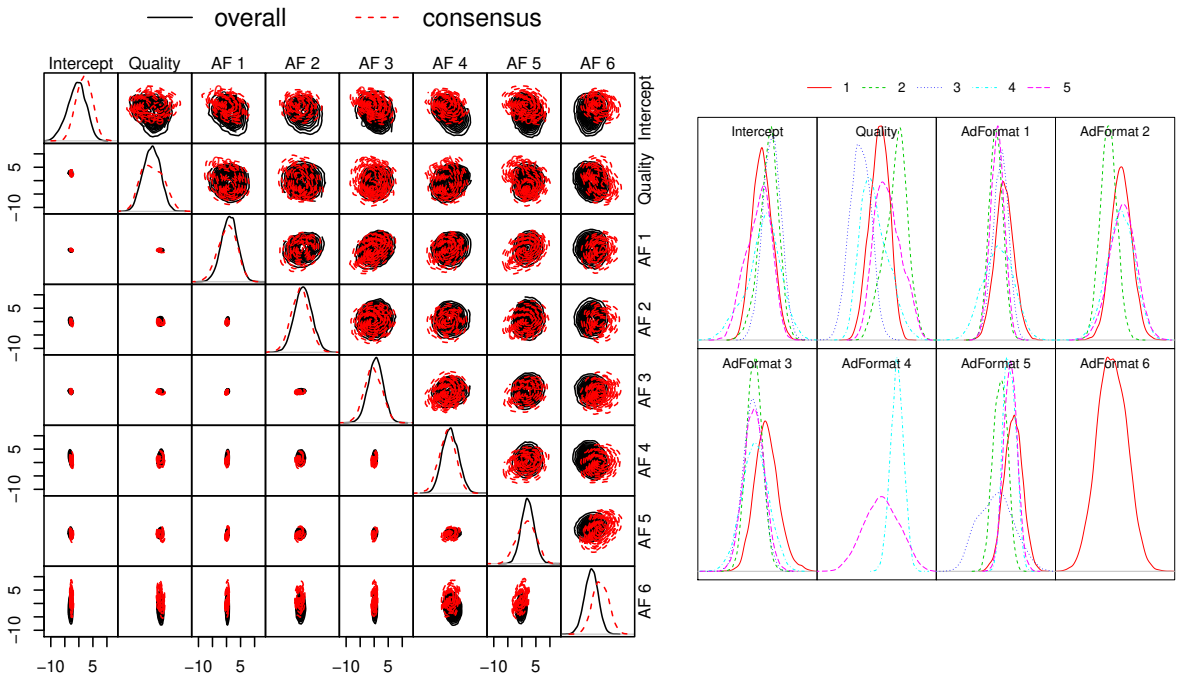
(a) (b)

Figure 8: *Logistic regression with (a) 1,000 and (b) 10,000 observations per worker. Compare to Figure 7(a). The equal weighting scheme is not shown here, to better focus on matrix and scalar schemes.*

The data consist of nearly 24 million observations regarding the performance of individual advertising creatives nested within around 11,000 advertisers using a particular Google advertising product. The data were divided into 867 shards, so that no shard contained more than 50,000 observations. A few shards contained between 10,000 and 20,000 observations, with a median shard size of about 27,000 observations. Most shards were close to the median.

The model from equation (6) was run on a single machine for 10,000 MCMC iterations using the MCMCglmm package from CRAN (Hadfield, 2010). Figure 9 shows the posterior draws of μ based on the first 5 shards of data under a single machine run and the consensus Monte Carlo algorithm with 5 workers. We used only 5 of the 867 shards because of the long compute time for the single machine run. The shards were combined by averaging draws based on the scalar precisions of each element of μ. There is general agreement between the single machine and consensus algorithms. The largest disagreement is on *AdFormat 6*, which only one of the 5 shards had any information about. There was one shard that exhibited slow mixing on the *AdFormat 5* coefficient, leading to a somewhat overdispersed consensus distribution. That could presumably be improved through a longer run time or by using a more efficient Monte Carlo algorithm on the worker machine. There were only two shards with information about *AdFormat 4*, but its single machine and consensus estimates agree closely. The posterior correlations between elements of μ are not particularly strong, but to the extent that non-spherical errors are present (e.g. between *AdFormat 4* and the Intercept), they appear to be well captured by consensus Monte Carlo.

Figure 10 shows what happens to the computing time as the number of shards increases. The



(a) (b)

Figure 9: *Posterior draws of μ based on the first 5 shards described in Section 4.4. (a) Posterior draws from the single-machine and consensus Monte Carlo algorithms. (b) Draws from the five worker machines.*

single machine algorithm scales linearly up to about 20 shards, at which point it encounters a resource limit and begins slowing down. We stopped experimenting with the single machine run after 30 shards, because resource bottlenecks caused the machine to stop functioning, and we could not get the run to finish. With consensus Monte Carlo the job completes within about 20-30 minutes, regardless of the number of shards. These times are with off-the-shelf R code. Optimized code would improve both the single machine behavior and the run time on each worker machine.

In this application, the estimands of interest require the advertiser specific coefficients β_j. As mentioned in Section 3.4.1, these can be obtained with independent MCMC runs on each advertiser's data, given the draws of μ and Σ.

4.5 Nonparametric regression

We consider nonparametric regression models of the form

$$y_i \sim \mathcal{N}\left(f(\mathbf{x}_i), \sigma^2\right) \tag{7}$$

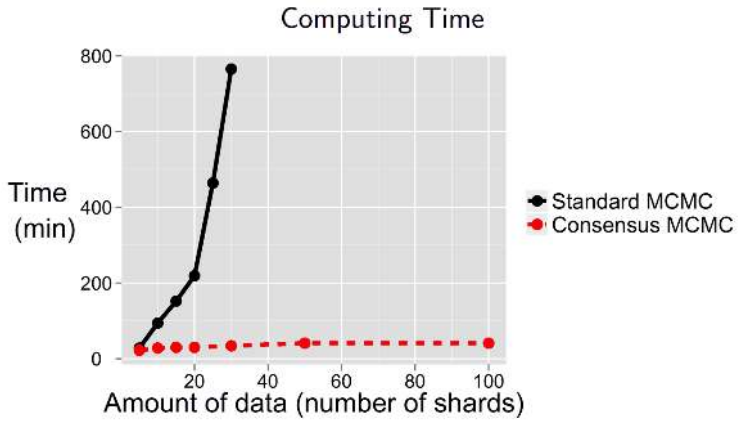


Figure 10: *Time required to complete 10,000 MCMC draws with different numbers of shards under the single machine and consensus Monte Carlo algorithm.*

independently across observations. Such models often lack interpretable sets of model parameters, so Bayesian inference focuses on the posterior distribution $p(f(\mathbf{x})|\mathbf{y})$ at a particular value of $\mathbf{x}$. As mentioned in Section 3.4.2, Consensus Monte Carlo can be applied to nonparametric regression models by fitting a separate model on each worker machine and averaging the draws from the worker-level predictive distributions.

As an example, consider BART (Chipman, George, and McCulloch, 2010), a tree-based algorithm for nonparametric regression, where

$$f(\mathbf{x}) = \sum_{t=1}^{T} f_t(\mathbf{x}) \tag{8}$$

and each $f_t(\mathbf{x})$ is a decision tree. Each interior node j of tree t compares a specific element ν_{tj} of $\mathbf{x}$ to a specific cutpoint χ_{tj}. Starting at the root of the tree, if node j has children and $x_{\nu_{tj}} \leq \chi_{tj}$ then $\mathbf{x}$ falls to the left child, otherwise it falls to the right. If node j is a leaf then it contains a mean parameter M_{tj} giving the contribution of tree t to $f_t(\mathbf{x})$. BART is an additive model that combines contributions from many trees.

The prior distribution for BART recommended by Chipman *et al.* (2010) consists of a prior over the topology of each tree, a prior over the parameters M_{jt}, and a prior on the residual variance σ^2. The prior probability that a node at depth d splits into children at depth $(d+1)$ is $a/(1+d)^b$. If there is a split, a variable ν_{tj} is chosen uniformly from the set of available variables, and a cutpoint χ_{tj} uniformly from the set of available cutpoints. If there is no split, then the node is a leaf containing mean parameter M_{tj} with prior distribution $M_{tj} \sim \mathcal{N}\left(0, \tau^2\right)$, independently across t and j. A conditionally conjugate inverse gamma prior is assigned to σ^2.

We tested the consensus Monte Carlo algorithm with BART using simulated data from Fried-

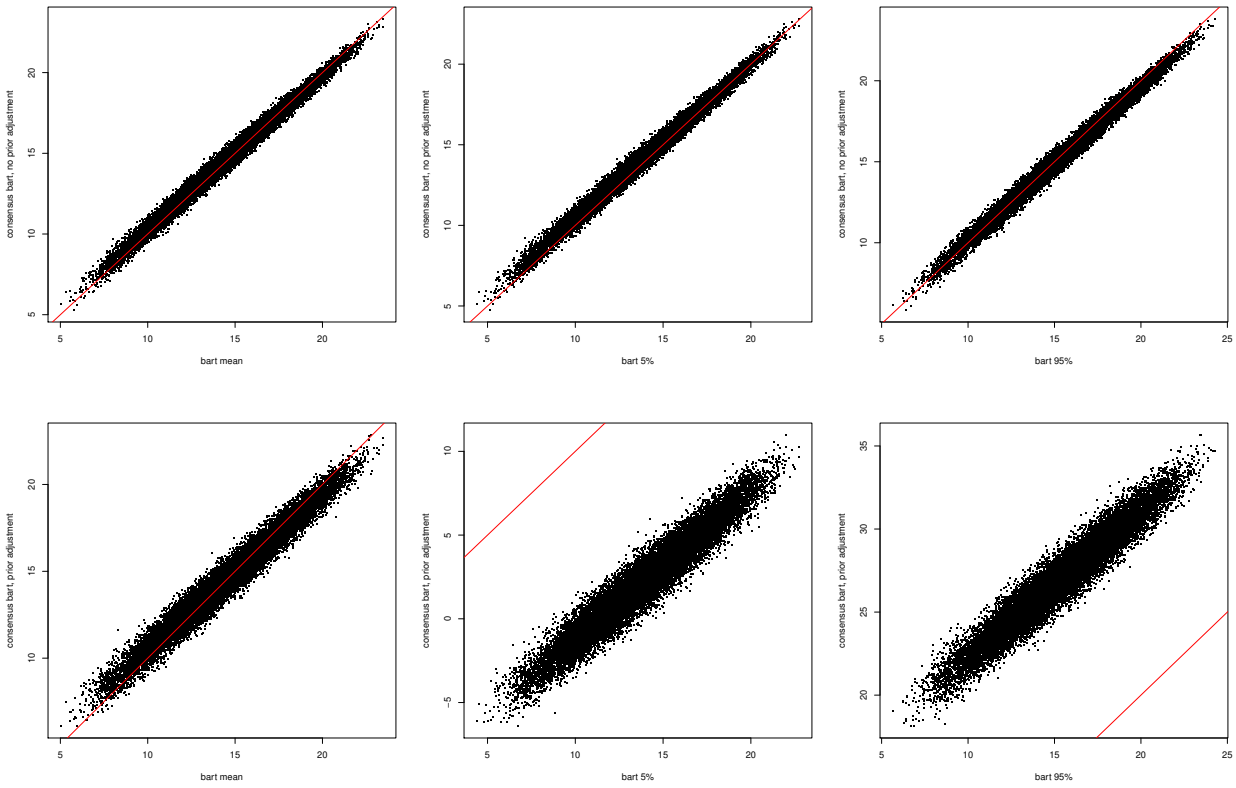


Figure 11: *Comparing consensus BART with the single machine algorithm. The three columns compare posterior means (left) and posterior 5% (center) and 95% (right) percentiles from the single-machine and consensus posterior distributions. In the top row the 30 workers were run with the same prior as the single machine. In the bottom row the prior was raised to the 1/30 power.*

man's test function (Friedman, 1991)

$$f(\mathbf{x}) = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \epsilon \tag{9}$$

where $\epsilon \sim \mathcal{N}\left(0, 3^2\right)$ is an independent error term. Our full data set contained 20,000 observations where each component of $\mathbf{x}_i$ was independently simulated from $\mathcal{U}(0,1)$. We also simulated $x_6, \ldots, x_{10}$ as additional superfluous predictors. We chose 20,000 observations because that is close to the largest data set that the single-machine BART algorithm could handle in a reasonable time period. The consensus Monte Carlo algorithm was implemented using the *parallel* package from R on a 30 processor machine.

Figure 11 summarizes the results of two model runs on the same set of simulated data. In the top row, the single machine algorithm and the worker machines in consensus Monte Carlo each used the recommended prior from Chipman *et al.* (2010). In the bottom row the prior was raised to the 1/30 power to account for the fact that there are 30 workers. With no prior adjustment the single

machine and consensus algorithms essentially agree on both the posterior mean and the upper and lower posterior quantiles. When the prior is adjusted, the consensus Monte Carlo posterior becomes overdispersed. The prior distribution plays a significant role in determining the size of each tree in the BART algorithm. Weakening the prior results in much larger trees, and generally more diffuse posterior distributions. Averaging over shards mitigates the overdispersion, but not enough to match the single machine algorithm. Alternative strategies for prior adjustment, such as only modifying the prior on M_{tj} and σ^2, might still prove to be effective, and could potentially be necessary in other applications.

Interestingly, the consensus Monte Carlo algorithm actually fits the data slightly better than the single machine algorithm. The correlation between the posterior means and the true function values for the single machine run was .992. The same correlation for consensus Monte Carlo was .995. The reason is that the Monte Carlo algorithm used to fit BART tends to have trouble mixing as sample sizes grow large. The consensus Monte Carlo posterior is based on 30 rapidly mixing workers, while the single machine algorithm is based on one slow mixing worker.

5 Discussion

The idea of distributing a Bayesian calculation to a group of independent workers is a natural one, justified by equation (2). This article has shown that there are models, some with non-Gaussian posteriors, for which the draws from workers can be combined using simple weighted averages.

Consensus Monte Carlo scales to very large numbers of machines in ways that multi-core algorithms cannot. It can be implemented with existing code, whether single or multi-threaded, and it is agnostic to the specific algorithm used to generate the worker-level draws. It is similarly agnostic to the method used to parallelize the computation. It can be used on a cluster of computers, a single multi-core or multi-processor computer, or an arbitrary collection of machines that need not be linked by a high speed network.

A fundamental weakness of the algorithm is its limitation to unknowns of fixed dimension in continuous parameter spaces. For example, label switching and changing numbers of dimensions would make it difficult to use consensus Monte Carlo to cluster observations based on infinite Dirichlet process mixtures. Similar issues make consensus Monte Carlo ill suited to model averaging using spike and slab priors.

There is a long list of open questions that needs to be addressed on consensus Monte Carlo. We need to better understand how the algorithm behaves as posterior distributions move away from Gaussianity. To the extent that averaging fails in non-Gaussian models other methods of consensus should be explored. Improvements to the algorithm will probably involve a small number of between-machine communications, a handful of which would not fundamentally change the character of the method. Consensus Monte Carlo is likely to be useful with models that violate equation (2), such as hierarchical models with crossed random effects, or Gaussian processes with

non-trivial covariances functions, but these directions need to be explored.

The examples in Section 4 provide useful but sometimes contradictory lessons for practitioners. Consensus Monte Carlo was found to perform well on a variety of models, but subtle adjustments to the prior were sometimes necessary to achieve good results. In Section 4.1 adjusting the prior was critical to the method's success, while in Section 4.5 it was detrimental. Until a more solid theory can be established, the best advice we can give practitioners is to try consensus Monte Carlo on a simulated data set to determine conditions under which it performs well in a particular problem.

References

Anderson, T. E., Culler, D. E., and Patterson, D. (1995). A case for now (networks of workstations). *Micro, IEEE* **15**, 54–64.

Chipman, H. A., George, E. I., and McCulloch, R. E. (2010). Bart: Bayesian additive regression trees. *The Annals of Applied Statistics* **4**, 266–298.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM* **51**, 107–113.

Denison, D. G. T., Mallick, B. K., and Smith, A. F. M. (1998). Automatic Bayesian curve fitting. *Journal of the Royal Statistical Society, Series B, Methodological* **60**, 333–350.

Friedman, J. H. (1991). Multivariate adaptive regression splines. *The Annals of Statistics* **19**, 1–67.

Ghahramani, Z. and Beal, M. J. (2001). Propagation algorithms for variational Bayesian learning. *Advances in neural information processing systems* 507–513.

Guha, S., Kidwell, P., Hafen, R., and Cleveland, W. S. (2009). Visualization databases for the analysis of large complex datasets. In *International Conference on Artificial Intelligence and Statistics*, 193–200.

Hadfield, J. D. (2010). Mcmc methods for multi-response generalized linear mixed models: The MCMCglmm R package. *Journal of Statistical Software* **33**, 1–22.

Hinton, G. E. and Van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, 5–13. ACM.

Huang, Z. and Gelman, A. (2005). Sampling for Bayesian computation with large datasets. Tech. rep., Columbia University Department of Statistics.

Jaakkola, T. S. and Jordan, M. I. (2000). Bayesian parameter estimation via variational methods. *Statistics and Computing* **10**, 25–37.

Kleiner, A., Talwalkar, A., Sarkar, P., and Jordan, M. I. (2011). A scalable bootstrap for massive data. *arXiv preprint arXiv:1112.5016* .

Le Cam, L. M. and Yang, G. L. (2000). *Asymptotics in Statistics: Some Basic Concepts*. Springer-Verlag.

Lee, A., Yao, C., Giles, M. B., Doucet, A., and Holmes, C. C. (2010). On the utility of graphics cards to perform massively parallel simulation of advanced Monte Carlo methods. *Journal of Computational and Graphical Statistics* **19**, 769–789.

Malewicz, G., Austern, Matthew H. Bik, A. J. C., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. (2010). Pregel: A system for large-scale graph processing. In *SIGMOD'10*, 135–145.

Rue, H., Martino, S., and Chopin, N. (2009). Approximate bayesian inference for latent gaussian models by using integrated nested laplace approximations. *Journal of the royal statistical society: Series b (statistical methodology)* **71**, 319–392.

Suchard, M. A., Wang, Q., Chan, C., Frelinger, J., Cron, A., and West, M. (2010). Understanding GPU programming for statistical computation: Studies in massively parallel massive mixtures. *Journal of Computational and Graphical Statistics* **19**, 419–438.

White, T. (2012). *Hadoop: the definitive guide*. O'Reilly.

Zhang, Y., Duchi, J. C., and Wainwright, M. J. (2012). Communication-efficient algorithms for statistical optimization. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, 6792–6792. IEEE.