

Workstation Idle Period Duration Prediction for Distributed Computing

Justin Talbot
Brigham Young University

April 21, 2004

Abstract

We consider the problem of predicting idle period duration on workstations used in DOGMA, a distributed computing architecture developed and deployed at Brigham Young University. BYU provides students with open access to nearly 900 workstations throughout the main campus for use in homework and research pursuits. When idle, these computers are used by DOGMA to perform time consuming research computing. Predicting the lengths of future idle periods could improve the performance of DOGMA. We develop a hierarchical Bayesian model for idle period lengths. We then use Markov Chain Monte Carlo (MCMC) computation to create an idle duration predictive distribution for each computer. We demonstrate that a Bayesian approach produces reasonable predictive distributions for this application.

1 Introduction

The DOGMA system [1] was developed at Brigham Young University (BYU) to take advantage of nearly nine hundred computers on campus that are available to students for both academic and personal use. These computers sit idle a large portion of each day, largely during classes and at night. During our study period, this idle time averaged nearly 5,400 CPU hours per day. The DOGMA system seeks to reclaim this idle computing time by running computationally intensive research on available computers.

The DOGMA system works by running a simple program that is attached to the Windows screen saver. When the Windows screen saver is launched, the DOGMA program contacts the main DOGMA server and requests a job. The DOGMA server assigns

the available workstation the first job waiting on the queue. When the workstation completes computation on the job, the results are returned to the server which stores them for later user access. However, if the Windows screen saver is interrupted at any point during computation, then the job is immediately terminated and any completed work is lost. The job is returned to the DOGMA queue and is reassigned later. This greatly increases the computation time of each job. To reduce the number of jobs that are terminated prematurely, the DOGMA administrators would like to assign jobs only to workstations that have a high probability of remaining idle long enough to complete the job.

Our goal is to estimate the distribution of future idle period durations for each computer in the DOGMA network. This distribution can then be used to guide the assignment of jobs in a way that reduces job interruption. In section 2 we describe the data collected over a 43 day period from the DOGMA network. We discuss problems with the data and the solutions used in this paper. Section 3 describes the hierarchical Bayesian model used in modelling the data. Section 4 presents the posterior results and assesses the model. Finally, Section 5 concludes the paper and outlines various directions for future model improvements.

2 Data

The Parallel Processing Research Group in the BYU Computer Science Department administers DOGMA and has provided the data for this analysis. The raw data, taken from the DOGMA server logs, includes simply the date, time, and IP address of idle notifications from available workstations. When workstations running the DOGMA software are idle (either doing nothing or executing a DOGMA job) they will contact the DOGMA server periodically. This lets the server know 1) that the machine is idle and 2) that the job, if one is assigned to that workstation, has not been prematurely terminated. The time interval between subsequent idle notifications can vary due to computation load and network delays.

Since DOGMA does not currently record the contents of these idle notifications, we must reconstruct the length of the idle period by examining the time and IP address recorded by the DOGMA server. In discussion with the DOGMA administrators, they indicated that the interval between idle notifications from any single idle machine should rarely be above 5 minutes.¹

¹The author would like to thank Patrick Mullen and Thomas Warne for their help in obtaining and processing this data.

Thus, we derive idle periods as follows. The first idle notification from a previously busy machine indicates the start of an idle period. The idle period continues as long as idle notifications are received no more than 5 minutes apart. Once there is a span of more than 5 minutes, the computer is considered to have been busy at some point during that interval. Since we do not know at what point in that interval the computer became busy, we assume the worst case—that the idle period terminated immediately after the last idle notification was received. This summary of the raw data gives a list of approximate idle periods for each DOGMA workstation. Each idle period has a starting time and a duration in minutes.²

Our data is taken from 882 workstations that were available at some point during the 43 day test period. The distribution of idle period durations for all computers during this time is shown in Figure 1. The mean idle period duration across all computers in the study is 97 minutes with a 90% interval of (3, 374) minutes.

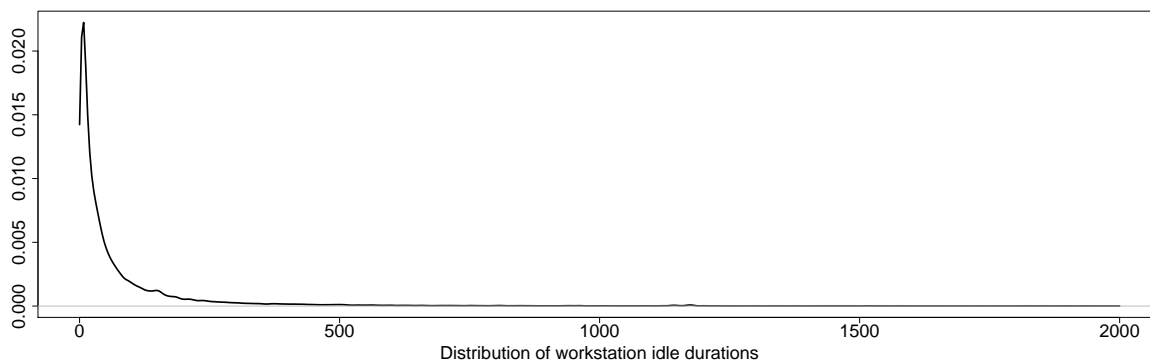


Figure 1: Distribution of idle period durations for all workstations

The duration data is subject to many inaccuracies. In addition to truncation, the 5 minute cutoff point may also be incorrect and may create duration estimates that are substantially longer or shorter than the true duration. We do not treat these issues in this paper.

²This data is available online at <http://students.cs.byu.edu/~jtalbot/byuidletimedata.txt>

3 Bayesian Model

3.1 Model

A number of models for predicting idle time duration have been described in the literature. Gelernter et al. [2] try to predict future idle time using a number of different ad hoc strategies. They find that the best predictors of future idle time are the idleness in the last few minutes—a computer that has been idle is more likely to stay idle—and the idleness at the same time the day before. Unlike our problem, they are more interested in finding the total idle time during some time span, rather than the length of individual idle periods.

Acharya et al. [3] use the mean of past idle period durations for each computer in their system to guide job placement.

Camenisch and Russ [4] try fitting their idle duration data to six different standard distributions. They find that both the Weibull and the Pearson 5 distributions provide reasonable approximations.

We choose to model idle period durations using a Weibull distribution. Importantly, the Weibull distribution is continuous and maintains support on the positive real numbers. The Weibull distribution comes in one-, two-, and three-parameter versions. Since the shape of the distribution is unknown, we cannot use the one-parameter model. The three-parameter version allows for minimum values other than zero. In our model, this would represent a minimum idle period between workstations busy times. There is no practical reason why the minimum time would be non-zero for the computers in our study. Thus, we use the two-parameter Weibull distribution.

In the first stage of our hierarchy, we model Y_i , the next idle period duration for workstation i , as

$$Y_i | \beta_i, \nu_i \sim Weibull(\beta_i, \nu_i)$$

Notice that β_i and ν_i are unique for each computer. This results in $2n$ parameters at the first stage of the hierarchy, where n is the number of workstations.

3.2 Priors

We next specify prior distributions for β_i and ν_i . Both β_i and ν_i have support on the positive reals. Without any other prior knowledge, we choose the following prior distributions.

$$\beta_i | \kappa, \lambda^2 \sim LN(\kappa, \lambda^2) \quad i = 1, \dots, n$$

and

$$\nu_i | \sigma, \tau^2 \sim LN(\sigma, \tau^2) \quad i = 1, \dots, n$$

where LN is the log normal distribution.

At the next stage, we specify prior distributions for the parameters κ , λ , σ , and τ :

$$\kappa | m_\kappa, s_\kappa^2 \sim N(m_\kappa, s_\kappa^2)$$

$$\lambda | t_\lambda \sim Exp(t_\lambda)$$

$$\sigma | m_\sigma, s_\sigma^2 \sim N(m_\sigma, s_\sigma^2)$$

and

$$\tau | t_\tau \sim Exp(t_\tau)$$

where N is the normal distribution and Exp is the exponential distribution. These distributions are chosen to maintain the proper support for the log normal distributions.

Finally, we choose values for the hyperparameters— m_κ , s_κ^2 , t_λ , m_σ , s_σ^2 , and t_τ . Since β_i is the shape parameter of the Weibull distribution, we expect that it will be relatively small. Thus we chose $m_\kappa = 0$, $s_\kappa = 1$, and $t_\lambda = 2$.

Parameter	Hyperparameters
κ	$m_\kappa = 0 \quad s_\kappa = 1$
λ	$t_\lambda = 2$
σ	$m_\sigma = 4.5 \quad s_\sigma = 1$
τ	$t_\tau = 2.5$

Table 1: Prior hyperparameter values

ν_i should be quite large. From our own experience, we estimated that a mean of 100 minutes for the idle time distribution would be reasonable. Choosing $m_\sigma = 4.5$, $s_\sigma = 1$, and $t_\tau = 2.5$ gives a plausible prior.

4 Results

Since the posterior distribution is not closed form, we use an MCMC algorithm to generate observations from the posterior. The candidate distributions were tuned by hand to ensure proper mixing. Details of the implementation can be found in Appendix A. Figure 2 shows the posterior means of β_i and ν_i for all 882 workstations. Table 2 shows the posterior means and standard deviations for κ , λ , σ , and τ . Density plots of these parameters can be found in Appendix B.

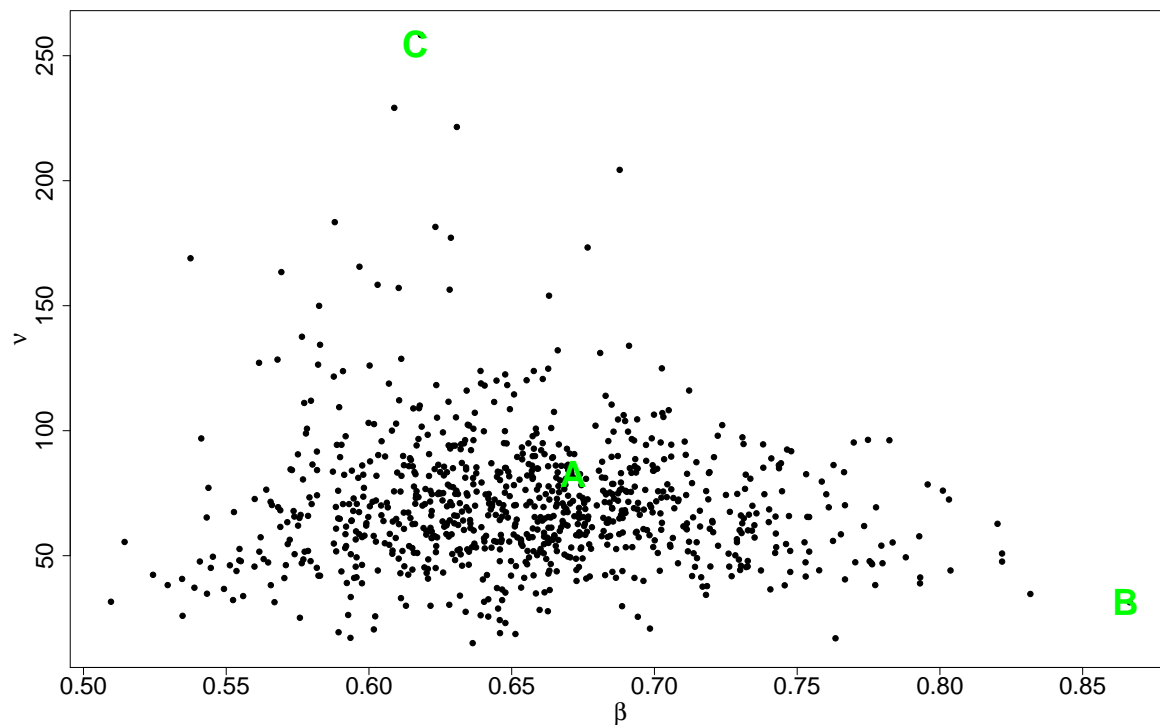


Figure 2: Joint posterior distribution of means of β_i and ν_i

Figures 3, 4, and 5 show the posterior predictive distribution for the three computers marked A, B, and C in Figure 2. The posterior distribution appears to fit the data reasonably well. However, it does appear to overestimate the probability for very small idle times. This is especially true for computer C. Figure 6 shows the predictive distribution for the next computer that is added to the DOGMA system.

Parameter	Mean	Standard Deviation
κ	-0.427	0.00395
λ	0.0984	0.00333
σ	4.183	0.0136
τ	0.386	0.0111

Table 2: Posterior parameter estimates

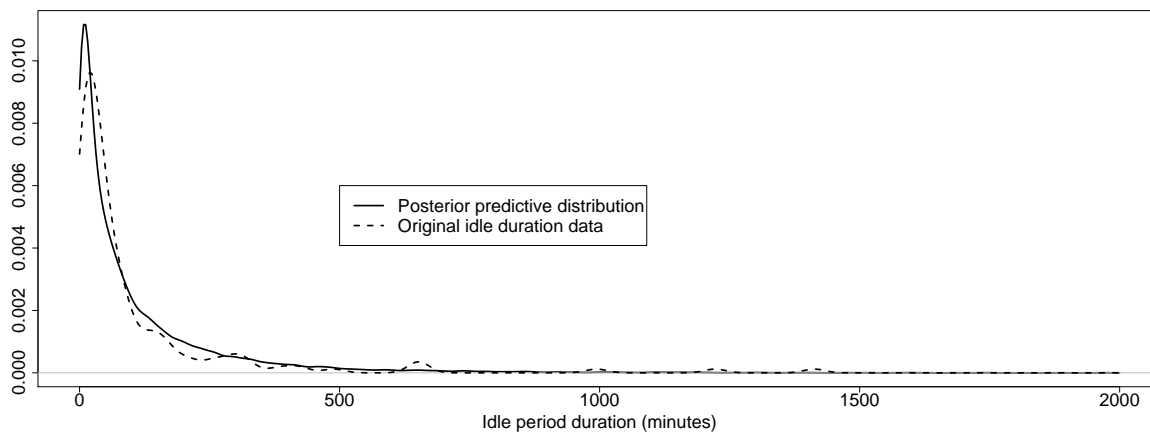


Figure 3: Posterior predictive distribution for computer A

5 Discussion

This approach falls short in two areas. First, our method for recovering idle period duration data from the raw data could be improved. A better approach would model the truncation inherent to the data. Also, we use a fixed 5 minute cutoff point for determining idle period continuance. A more flexible cutoff policy could be used instead. Perhaps some probability could be assigned to each idle period that reflects how ‘sure’ we are that that idle period is correct. A Bayesian approach could also be used to find a better cutoff point.

Second, our model produces a single distribution for each workstation. A more complete model would produce distributions for the time of day, for the day of the week, or for the physical location of each workstation. Each of these represents a large increase in the number of parameters that must be computed. However, since there is a large

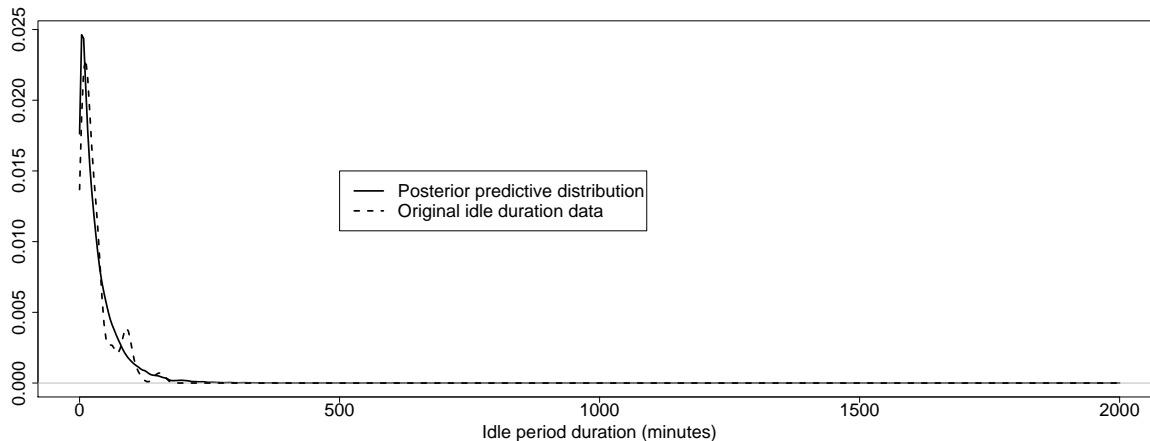


Figure 4: Posterior predictive distribution for computer B

amount of data available, this should not be a problem.

The overestimation of the probability of small idle period durations is problematic. This may indicate a problem with the model. Perhaps a likelihood distribution other than Weibull would be more appropriate.

Finally, it remains to be seen if this can be used in an actual distributed system. Although the approach presented in this paper generates reasonable distributions, it is very time consuming. In a distributed system, the time necessary to make a job assignment decision must be very short. If this approach is used, the posterior distribution would probably have to be computed infrequently and cached. Questions remain on how to efficiently store the posterior distribution and on how often the posterior distribution would have to be recomputed to achieve good results.

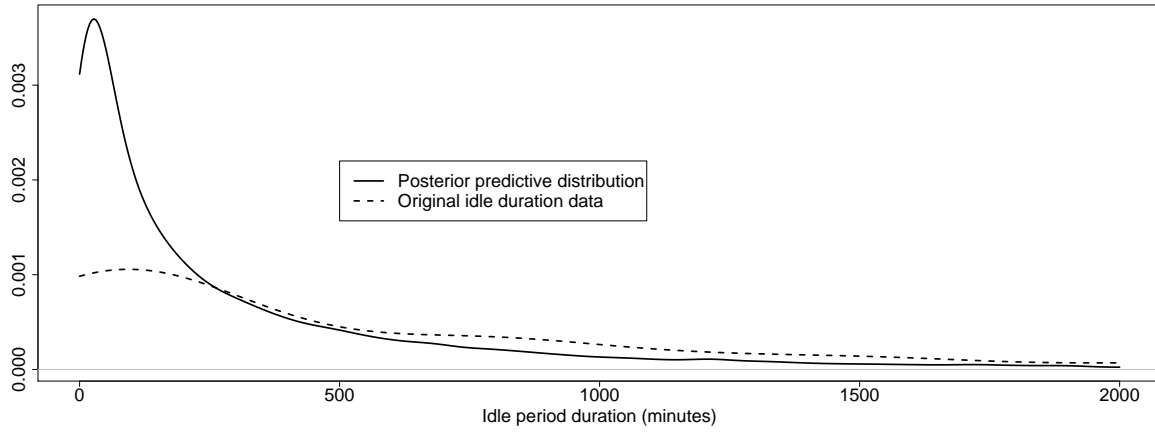


Figure 5: Posterior predictive distribution for computer C

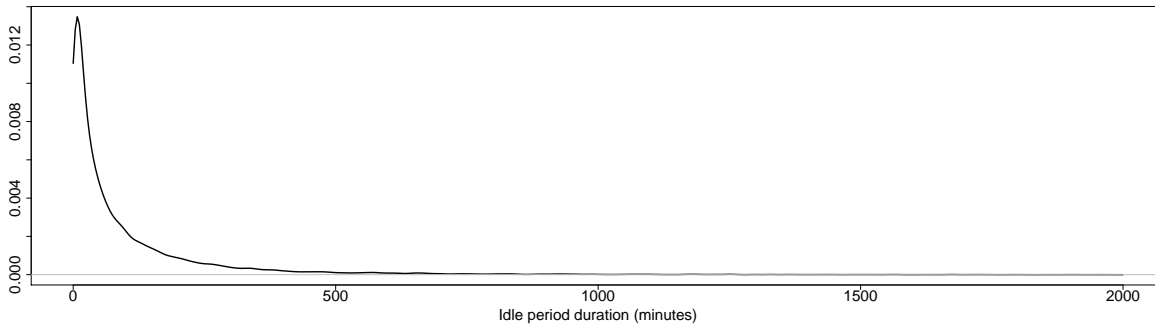


Figure 6: Posterior predictive distribution for the next computer added to DOGMA

References

- [1] Glenn Judd, Mark Clement, and Quinn Snell. “DOGMA: Distributed Object Group Metacomputing Architecture”, *Concurrency: Practice and Experience*. Vol. 10(11-13), pp 977-983, 1998.
- [2] David Gelernter, Marc J. Jourdenais, et al. “Piranha Scheduling: Strategies and Their Implementation”, 1993, Technical Report: New Haven. <http://citeseer.nj.nec.com/carriero95piranha.html>
- [3] Anurag Acharya, Guy Edjlali, Joel Saltz. “The Utility of Exploiting Idle Workstations for Parallel Computation”, *Proceedings of 1997 Sigmetrics International Conference on Measurement and Modeling of Computer Systems*. June 1997, Seattle, Washington. <http://citeseer.nj.nec.com/acharya97utility.html>
- [4] Joel Camenisch and Samuel H. Russ. “A Statistical Approach to Predicting Workstation Idle Times”, An Abstract Presented for Consideration to the *Conference on High Performance Distributed Computing*. Not yet published. <http://citeseer.nj.nec.com/110705.html>

A MCMC implementation details

We use Gibbs sampling which requires complete conditionals from which to generate. The complete conditionals are as follows:

$$[\beta_i|\cdot] \sim \prod_k^n \left[\beta_i \left(\frac{Y_{ik}}{\nu_i} \right)^{(\beta_i-1)} e^{-\left(\frac{Y_{ik}}{\nu_i} \right)^{\beta_i}} \right] e^{-\frac{(\log(\beta_i)-\kappa)^2}{2\lambda^2}}$$

$$[\nu_i|\cdot] \sim \prod_k^n \left[\frac{1}{\nu_i} \left(\frac{Y_{ik}}{\nu_i} \right)^{(\beta_i-1)} e^{-\left(\frac{Y_{ik}}{\nu_i} \right)^{\beta_i}} \right] e^{-\frac{(\log(\nu_i)-\sigma)^2}{2\tau^2}}$$

where n is the number of idle periods observed on workstation $_i$, and,

$$[\kappa|\cdot] \sim \prod_k^m \left[e^{-\frac{(\log(\beta_k)-\kappa)^2}{2\lambda^2}} \right] e^{-\frac{(\kappa-m\kappa)^2}{2s_\kappa^2}}$$

$$[\lambda|\cdot] \sim \prod_k^m \left[\frac{1}{\lambda} e^{-\frac{(\log(\beta_k)-\kappa)^2}{2\lambda^2}} \right] e^{-t_\lambda \lambda}$$

$$[\sigma|\cdot] \sim \prod_k^m \left[e^{-\frac{(\log(\nu_k)-\sigma)^2}{2\tau^2}} \right] e^{-\frac{(\sigma-m\sigma)^2}{2s_\sigma^2}}$$

$$[\tau|\cdot] \sim \prod_k^m \left[\frac{1}{\tau} e^{-\frac{(\log(\nu_k)-\sigma)^2}{2\tau^2}} \right] e^{-t_\tau \tau}$$

where m is the number of workstations.

Since these complete conditionals are not closed form we use Metropolis-Hastings to generate observations from these distributions. Our proposal distribution for all the parameters was a normal distribution with the mean at the previous value and standard deviations as shown in Table 3.

We generated 25,000 observations from the distribution. 5,000 were thrown out as part of the burn period, leaving us with 20,000 draws from the posterior.

Parameter	Proposal Distribution	Standard Deviation
β_i		0.06
ν_i		12
κ		0.007
λ		0.003
σ		0.005
τ		0.005

Table 3: Standard Deviations for Proposal Distributions

B Posterior distributions of parameters

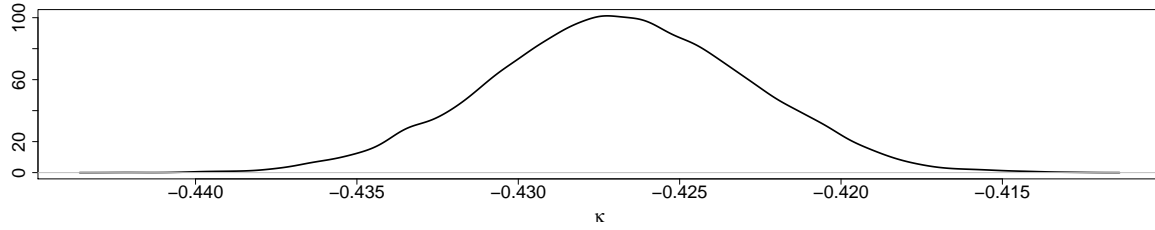


Figure 7: Posterior distribution of κ

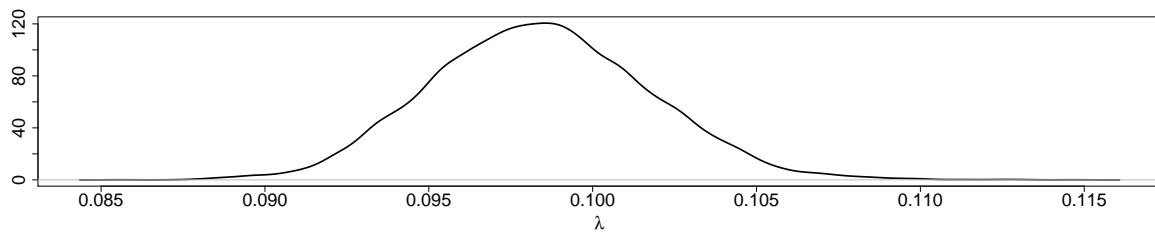


Figure 8: Posterior distribution of λ

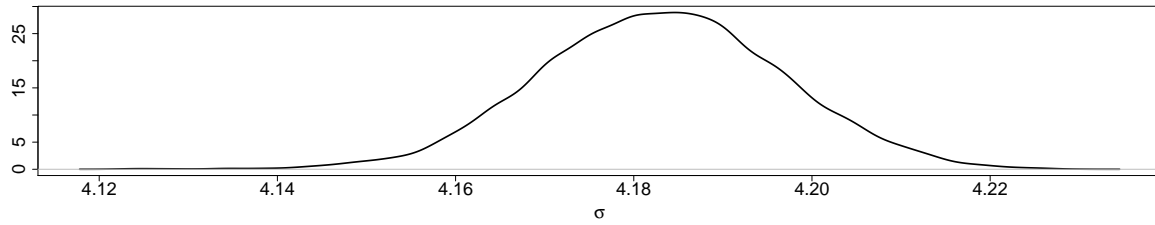


Figure 9: Posterior distribution of σ

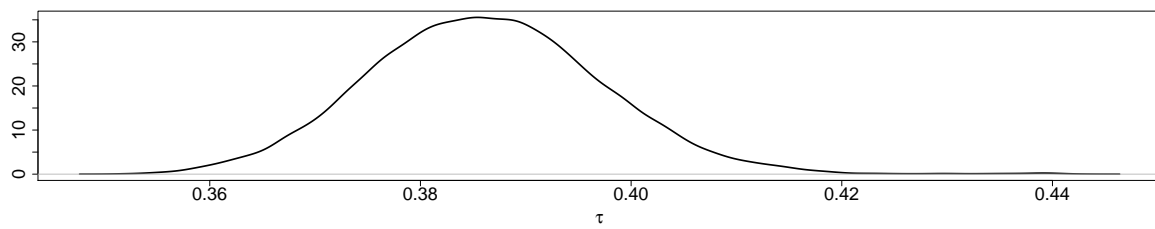


Figure 10: Posterior distribution of τ

C MCMC Code

This is the relevant portion of the C++ code used to produce the results in this paper.

```
/* Justin Talbot 2004 */

#include "Prediction.h"

#include <iostream> #include <fstream>

int IdleTimePredictor::burn = 25;

double IdleTimePredictor::betaCS = 0.06; double
IdleTimePredictor::nuCS = 12;

double IdleTimePredictor::kappaCS = 0.007; double
IdleTimePredictor::lambdaCS = 0.003; double
IdleTimePredictor::sigmaCS = 0.005; double
IdleTimePredictor::tauCS = 0.005;

double IdleTimePredictor::beta = 1; double IdleTimePredictor::nu =
90;

double IdleTimePredictor::kappa = 0; double
IdleTimePredictor::lambda = 0.5; double IdleTimePredictor::sigma =
4.5; double IdleTimePredictor::tau = 0.4;

double IdleTimePredictor::km = 0; double IdleTimePredictor::ks =
1; double IdleTimePredictor::lb = 2; double IdleTimePredictor::sm
= 4.5; double IdleTimePredictor::ss = 1; double
IdleTimePredictor::tb = 2.5;

double IdleTimePredictor::CCbeta( int i, double beta, double nu,
double kappa, double lambda ) {

    double o = beta;
```

```

double n = gennor( o, betaCS );
double result = o;
double gold = 0, gnew = 0;
double lognu = log( nu );
int k = 0;

if( n > 0 ) {

    gold += machines[i].idles.size() * ( log( o ) - (o-1)*lognu );
    for( k = 0; k < machines[i].idles.size(); k++ )
        gold += (o-1)*( machines[i].idles[k].logIdle ) - pow( ( machines[i].idles[k].idle
gold -= pow( (log(o) - kappa), 2 )/(2*lambda*lambda);

    gnew += machines[i].idles.size() * ( log( n ) - (n-1)*lognu );
    for( k = 0; k < machines[i].idles.size(); k++ )
        gnew += (n-1)*( machines[i].idles[k].logIdle ) - pow( ( machines[i].idles[k].idle
gnew -= pow( (log(n) - kappa), 2 )/(2*lambda*lambda);

    if( log( genunf( 0, 1 ) ) < (gnew-gold) )
        result = n;
}

return result;
}

```

```

double IdleTimePredictor::CCnu( int i, double beta, double nu,
double sigma, double tau ) {

    double o = nu;
    double n = gennor( o, nuCS );
    double result = o;
    double gold = 0, gnew = 0;
    double logo = log(o), logn = log(n);

    int k = 0;

```

```

if( n > 0 ) {

    gold += machines[i].idles.size() * (log( 1.0/o ) - (beta-1)*logo);
    for( k = 0; k < machines[i].idles.size(); k++ )
        gold += (beta-1)*( machines[i].idles[k].logIdle ) - pow( ( machines[i].idles[k].i
    gold -= pow( (log(o) - sigma), 2 )/(2*tau*tau);

    gnew += machines[i].idles.size() * (log( 1.0/n ) - (beta-1)*logn);
    for( k = 0; k < machines[i].idles.size(); k++ )
        gnew += (beta-1)*( machines[i].idles[k].logIdle ) - pow( ( machines[i].idles[k].i
    gnew -= pow( (log(n) - sigma), 2 )/(2*tau*tau);

    if( log( genunf( 0, 1 ) ) < (gnew-gold) )
        result = n;
}

return result;
}

```

```

double IdleTimePredictor::CCkappa( double* beta, double kappa,
double lambda ) {

    double o = kappa;
    double n = gennor( o, kappaCS );
    double result = o;
    double gold = 0, gnew = 0;
    int k = 0;

    for( k = 0; k < machines.size(); k++ )
        gold += -pow( log( beta[k] ) - o, 2 )/(2*lambda*lambda);
    gold += -pow( o - km, 2 )/(2*ks*ks);

    for( k = 0; k < machines.size(); k++ )
        gnew += -pow( log( beta[k] ) - n, 2 )/(2*lambda*lambda);
    gnew += -pow( n - km, 2 )/(2*ks*ks);
}

```

```

    if( log( genunf( 0, 1 ) ) < (gnew-gold) )
        result = n;

    return result;
}

double IdleTimePredictor::CCLambda( double* beta, double kappa,
double lambda ) {

    double o = lambda;
    double n = gennor( o, lambdaCS );
    double result = o;
    double gold = 0, gnew = 0;
    int k = 0;

    if( n > 0 ) {

        for( k = 0; k < machines.size(); k++ )
            gold += -log( o ) - pow( log( beta[k] ) - kappa, 2 )/(2*o*o);
        gold += -(lb * o);

        for( k = 0; k < machines.size(); k++ )
            gnew += -log( n ) - pow( log( beta[k] ) - kappa, 2 )/(2*n*n);
        gnew += -(lb * n);

        if( log( genunf( 0, 1 ) ) < (gnew-gold) )
            result = n;
    }

    return result;
}

double IdleTimePredictor::CCsigma( double* nu, double sigma,
double tau ) {

    double o = sigma;
    double n = gennor( o, sigmaCS );
    double result = o;

```

```

double gold = 0, gnew = 0;
int k = 0;

for( k = 0; k < machines.size(); k++ )
    gold += -pow( log( nu[k] ) - o, 2 )/(2*tau*tau);
gold += -pow( o - sm, 2 )/(2*ss*ss);

for( k = 0; k < machines.size(); k++ )
    gnew += -pow( log( nu[k] ) - n, 2 )/(2*tau*tau);
gnew += -pow( n - sm, 2 )/(2*ss*ss);

if( log( genunf( 0, 1 ) ) < (gnew-gold) )
    result = n;

return result;
}

double IdleTimePredictor::CCTau( double* nu, double sigma, double
tau ) {

double o = tau;
double n = gennor( o, tauCS );
double result = o;
double gold = 0, gnew = 0;
int k = 0;

if( n > 0 ) {

    for( k = 0; k < machines.size(); k++ )
        gold += -log( o ) - pow( log( nu[k] ) - sigma, 2 )/(2*o*o);
    gold += -(tb * o);

    for( k = 0; k < machines.size(); k++ )
        gnew += -log( n ) - pow( log( nu[k] ) - sigma, 2 )/(2*n*n);
    gnew += -(tb * n);

    if( log( genunf( 0, 1 ) ) < (gnew-gold) )
        result = n;
}
}

```

```

    }

    return result;
}

void IdleTimePredictor::predict( unsigned int numSamples,
std::vector< double >& samples ) {

    std::ofstream result( "result.txt", std::ios::out );

    MCMCData first, next;

    int i;

    int numMachines = machines.size();

    printf("Machines: %d\n", numMachines);

    //initialize chains
    for( int j = 0; j < numMachines; j++ ) {
        first.beta[j] = beta;
        first.nu[j] = nu;
    }

    first.kappa = kappa;
    first.lambda = lambda;
    first.sigma = sigma;
    first.tau = tau;

    int burnTime = numSamples*0.25;
    if( burnTime < 25 ) burnTime = 25;

    //initialize the chain
    for( i = 1; i < burnTime; i++ ) {

        for( int j = 0; j < numMachines; j++ ) {
            next.nu[j] = CCnu( j, first.beta[j], first.nu[j], first.sigma, first.tau );

```

```

    next.beta[j] = CCbeta( j, first.beta[j], next.nu[j], first.kappa, first.lambda );
}

next.kappa = CCkappa( next.beta, first.kappa, first.lambda );
next.lambda = CClambda( next.beta, next.kappa, first.lambda );

next.sigma = CCSigma( next.nu, first.sigma, first.tau );
next.tau = CCTau( next.nu, next.sigma, first.tau );

first = next;
}

//generate predictions
for( i = 0; i < numSamples; i++ ) {

    if( i%1000 == 0 )
        printf("%d\n", i );

    for( int j = 0; j < numMachines; j++ ) {
        next.nu[j] = CCnu( j, first.beta[j], first.nu[j], first.sigma, first.tau );
        next.beta[j] = CCbeta( j, first.beta[j], next.nu[j], first.kappa, first.lambda );
    }

    next.kappa = CCkappa( next.beta, first.kappa, first.lambda );
    next.lambda = CClambda( next.beta, next.kappa, first.lambda );

    next.sigma = CCSigma( next.nu, first.sigma, first.tau );
    next.tau = CCTau( next.nu, next.sigma, first.tau );

    first = next;

    chain.push_back( first );

    if( chain.size() > 100 ) {

        for( int n = 0; n < chain.size(); n++ ) {

            for( int k = 0; k < machines.size(); k++ ) {

```

```

        result << chain[n].beta[k] << " ";
    }

    for( int m = 0; m < machines.size(); m++ ) {
        result << chain[n].nu[m] << " ";
    }

    result << chain[n].kappa << " " << chain[n].lambda << " " << chain[n].sigma << " "
}
chain.clear();
}

//samples.push_back( genweibull( next.beta[0], next.nu[0] ) );
}

if( chain.size() > 0 ) {

    for( int n = 0; n < chain.size(); n++ ) {

        for( int k = 0; k < machines.size(); k++ ) {
            result << chain[n].beta[k] << " ";
        }

        for( int m = 0; m < machines.size(); m++ ) {
            result << chain[n].nu[m] << " ";
        }

        result << chain[n].kappa << " " << chain[n].lambda << " " << chain[n].sigma << " "
    }
    chain.clear();
}
}

```