Exact Sampling in Machine Scheduling Problems

Sung-woo Cho Ashish Goel

Department of Computer Science
University of Southern California
{sungwooc,agoel}@cs.usc.edu

Abstract. Analysis of machine scheduling problem can get very complicated even for simple scheduling policies and simple arrival processes. The problem becomes even harder if the scheduler and the arrival process are complicated, or worse still, given to us as a black box. In such cases it is useful to obtain a *typical* state of the system which can then be used to deduce information about the performance of the system or to tune the parameters for either the scheduling rule or the arrival process. We consider two general scheduling problems and present an algorithm for extracting an *exact* sample from the stationary distribution of the system when the system forms an ergodic Markov chain. We assume no knowledge of the internals of the arrival process or the scheduler. Our algorithm assumes that the scheduler has a natural monotonic property, and that the job service times are geometric/exponential. We use the Coupling From The Past paradigm due to Propp and Wilson to obtain our result. In order to apply their general framework to our problems, we perform a careful coupling of the different states of the Markov chain.

1 Introduction

Consider a system consisting of multiple machines where jobs arrive according to some probabilistic process, get scheduled on a machine using some scheduling rule, and leave when they have been serviced. Analysis of such a system can get very complicated even for simple scheduling policies and simple arrival processes [4]. The problem becomes even harder if the scheduler and the arrival process are complicated, or worse still, given to us as a black box. In such cases it is useful to obtain a *typical* state of the system. More formally, if the entire system forms an ergodic Markov chain [11], then we would like to have a sample from the stationary distribution of the system. A set of such samples could then be used to deduce information about the performance of the system or to tune the parameters for either the scheduling rule or the arrival process.

In this paper we study scheduling problems where the system forms an ergodic Markov chain. We will not assume any knowledge of the internal structure of the scheduler or the arrival process; we will treat them as black-boxes that we are allowed to query. However we do need to assume that the scheduling rule is monotonic, where a monotonic scheduler must have the following property. Suppose the scheduler assigns a job i to machine j when the system is in state s. Consider another state s' where machine j has the same load as in state s and all other machines have at least as much load as in state s. Then the scheduler must assign job i to the same machine j even if the system is in state s' instead of s. Intuitively, we can not force the scheduler to change its decision to assign a job to machine j by increasing the load on the other machines. We will further assume that each job gets executed on a single machine, that the service times for all jobs are i.i.d. geometric or exponential variables, and that each machine has a finite buffer size. Under these fairly general assumptions we exhibit an efficient algorithm to find a sample from the stationary state of the system. The algorithm simulates the Markov process for an expected time $O(T_{mix} \ln C)$ where T_{mix} is the mixing time for the Markov chain and C is the total buffer size of the machines. It is important to note that the algorithm does not need to know the mixing time.

Our algorithm uses an elegant tool called Coupling From The Past (CFTP) due to Propp and Wilson [12]. CFTP is essentially a variant of the Markov Chain Monte Carlo method [5, 9] and a natural extension of the work on approximate and exact sampling for specific Markov chains [2, 10, 9]. In order to use CFTP efficiently, we exhibit a coupling that allows different states of the Markov chain to evolve in a *monotonic* fashion [12]. A complete survey of work in this area is beyond the scope of this paper. While Markov Chain Monte Carlo methods have been used widely for problems arising from combinatorics (e.g. [3, 1]), physics (e.g. [6, 8]), and statistics (e.g. [2]), their use for traditional optimization problems such as scheduling has been limited. We hope that our work will lead to more applications of this flavour.

Section 2 presents a formal definition of the problem. Section 3 presents our algorithm and does a running time analysis. Section 4 presents simulation results for our algorithm, and illustrates its use for optimizing a simple scheduler.

2 Problem Definition

We are given m machines, $M_1, \cdots, M_m$. Let c_i denote the capacity of machine M_i. We define C as the sum of the capacities of all machines, that is $C = \sum_{i=1}^{m} c_i$. There are t types of jobs. Different job types may have a different arrival process, but we assume that the service times for the jobs are i.i.d. geometric random variables with mean $1/\mu$.

We are also given a scheduler that assigns arriving jobs to machines. However some machines cannot process certain types of jobs.[1] If a job of type i cannot be processed by M_j, then the scheduler cannot assign that job to M_j. This assignability can be represented as a matrix $V_{(t \times m)}$ as follows.

$V_{i,j} = 1$ if a job of type i can be assigned to machine M_j.
$\quad = 0$ otherwise.

If there is no space available in any machine for the arriving job, scheduler must discard the job. The scheduler may also do admission control. We will not assume any knowledge of the arrival process or the scheduler and will merely use them as black boxes.

[1] This is analogous to the 1-∞ case in machine scheduling [7].

The state s_t of the system at time t is defined as the number of jobs in every machine; let $l_i(s_t)$ denote the number of jobs in machine M_i. The state s_t is an m-dimensional vector, i.e. $\{l_1(s_t), l_2(s_t), \cdots, l_m(s_t)\}$. The state evolves only when a job arrives or leaves, and we define a random mapping $\phi(s_t)$ such that $s_{t+1} = \phi(s_t)$. In this paper, we will assume that the random mapping ϕ defines an Ergodic Markov chain. It is important to note that ϕ depends on the arrival process, the scheduler, and the mean service time $1/\mu$.

Our goal is to efficiently extract a sample from the stationary distribution of this Ergodic Markov chain. We will consider the following two scenarios.

Scenario 1: Each machine has a FIFO buffer. That is, only the job in the front of queue is processed.
Scenario 2: Jobs in each machine are processed simultaneously.

Our algorithm will work for a large class of schedulers that we call *monotonic schedulers.*

Definition 2.1 *We define $\leq$ over state space S such that for x and y in S, $x \leq y$ iff $l_i(x) \leq l_i(y)$ for all machines M_i.*

Definition 2.2 *Let $\Gamma_1(i)$ and $\Gamma_2(i)$ represent the state of the system after arrival of a job of type i in state s_1 and s_2 respectively. A scheduler is said to be monotonic if $s_1 \leq s_2 \Rightarrow \Gamma_1(i) \leq \Gamma_2(i)$ for all job types i.*

We claim that this is a natural class of schedulers to study for the following reason. Suppose that there are two states $\hat{L}$ and $\hat{U}$ such that $\hat{L} \leq \hat{U}$. Consider the situation that a job k is assigned to a machine M_j in the state $\hat{L}$. If $l_j(\hat{L}) < l_j(\hat{U})$, the monotonicity property will be maintained because $l_j(\Gamma_1) \leq l_j(\Gamma_2)$. Suppose that $l_j(\hat{L}) = l_j(\hat{U})$. It seems natural to require that if the load on machine j in $\hat{L}$ is kept the same and all other loads are either kept the same or increased, then the scheduler must still assign the job to machine j. This would result in monotonicity of the scheduler.

3 Applying CFTP to Our Scheduling Problems

3.1 Coupling the Evolution of Different States

We would like to couple the evolution of all the states in the Markov Chain to allow an efficient application of the CFTP technique. A state evolves only when a job arrives or leaves. The other factors – number of machines, capacities of machines, scheduling strategy, and so on – are identical for any state. Hence we just have to distribute job arrivals and departures carefully. Let the random mapping $\phi(s_t)$ be described as a function of the state s_t and a hidden random seed U_t. The random seed affects both the arrival and departure of jobs. If we use the same random seed for evolution from different states, we will obtain the same job arrivals at time t for all sample paths. Therefore all we need to take care of are the job-departure events. Recall (from section 2) that the service times for the jobs are geometrically distributed. We will couple the departure events by associating departure events with machines rather than with jobs, as outlined below.

Scenario 1: Suppose that a departure event happens from the machine M_i at time t. Let $\delta^{(i)}(t)$ be a geometric random variable with mean $1/\mu$. The next departure event for machine i happens at time $t+\delta^{(i)}(t)$. At this time, if the queue in M_i is not empty then the job in the front of the queue leaves the machine, otherwise nothing happens. This ensures that the service times of the jobs are geometrically distributed with mean $1/\mu$ because of the memoryless property of geometric random variables. The evolution of different states is coupled by providing them with identical values of $\delta^{(i)}(t)$.

Scenario 2: In scenario 1, we made m Poisson processes. In scenario 2, we make C Poisson processes. Suppose that a departure event happens from the j-th "slot" of machine M_i at time t. Let $\delta^{(i,j)}(t)$ be a geometric random variable with mean $1/\mu$. The next departure event from this slot happens at time $t+\min_j\{\delta^{(i,j)}(t)\}$. Suppose that there are k jobs in machine M_i at time t. If $j \leq k$ then one job in the buffer leaves the machine, otherwise nothing happens. This process ensures that the service times of the jobs are geometrically distributed with mean $1/\mu$. Like scenario 2, the evolution of different states is coupled by providing them with identical values of $\delta^{(i,j)}(t)$.

The above coupling has been carefully devised to allow us to prove monotonicity of our Markov chain, as explained in the next section.

3.2 Monotonicity of the Coupled Markov Chain

CFTP is particularly efficient for monotonic Markov chains as defined by Propp and Wilson [12].

Definition 3.1 *Suppose that the state space S of our problem admits a natural partial ordering $\leq$, with a minimum element $\hat{0}$ and a maximum element $\hat{1}$. A Markov chain is defined over S in terms of the transition probability matrix P. For elements x and y in S, let the function $\phi(i)$ be a random mapping of S to S itself with $Pr[\phi(x) = y] = P_{x,y}$. We say that the Markov chain is monotonic iff $x \leq y \Rightarrow \phi(x) \leq \phi(y)$.*

The following theorem is now immediate; we sketch the proof of the theorem for completeness.

Theorem 1. *Given a monotonic scheduler and the coupling described above, our Markov chain is monotonic in both scenarios.*

Proof. We will state the proof for the first scenario; the proof for the other scenario is similar. Let $\hat{L}$ and $\hat{U}$ be two states such that $\hat{L} \leq \hat{U}$. Let Γ_1 and Γ_2 be the next states starting from $\hat{L}$ and $\hat{U}$, respectively, after one event. We need to consider

1. Departure Events: Let the departure event correspond to the j-th machine. We know that $l_j(\hat{L}) \leq l_j(\hat{U})$. If $l_j(\hat{L}) = 0$ then $l_j(\Gamma_1) = 0 \leq l_j(\Gamma_2)$. On the other hand, if $l_j(\hat{L}) > 0$ then so is $l_j(\hat{U})$ i.e. a job will depart from machine j in both steps. Now $l_j(\Gamma_2) = l_j(\hat{U}) - 1 \geq l_j(\hat{L}) - 1 = l_j(\Gamma_1)$. Thus, in both cases, $l_j(\Gamma_2) \geq l_j(\Gamma_1)$. Since the loads on the other machines are unchanged, this implies that $\Gamma_1 \leq \Gamma_2$.
2. Arrival Events: Since the scheduler is monotonic, we are guaranteed that $\Gamma_1 \leq \Gamma_2$.

We can now use the CFTP paradigm for monotonic Markov chains as defined by Propp and Wilson [12] to obtain a sample from the stationary distribution. In this scheme, the evolutions of the states $\hat{0}$ and $\hat{1}$ in the Markov chain are simulated from time $-\tau$ to 0. If the two processes are in the same state at time 0, then this common state is output as the sample from the stationary distribution. If not, then τ is doubled and the process is repeated. It is important to reuse the same random numbers at time $-t$ during all the iterations of the algorithm.[2] The main intuition is that when the sample paths starting from the bottom state $\hat{0}$ and the top state $\hat{1}$ converge, they sandwich the entire state space in between. The reader is referred to an excellent description of the process by Propp and Wilson [12] for more detail.

We are going to use the state where all machines are empty as $\hat{0}$ and the state where all machines are filled to capacity as $\hat{1}$. Since departures are associated with machines rather than jobs, the random number reuse is easy to ensure. The following theorem follows from a general theorem due to Propp and Wilson [12] regarding the correctness of the CFTP paradigm.

Theorem 2. *The algorithm outlined above samples* exactly *from the stationary distribution of the ergodic Markov chain defined by the scheduler and the arrival process.*

3.3 Running Time Analysis

Theorem 3. *Define*

$$\Phi_{t_1}^{t_2}(x) = \phi_{t_2-1}(\phi_{t_2-2}(\cdots(\phi_{t_1}(x, U_{t_1}), U_{t_1+1}), \cdots, U_{t_2-2}), U_{t_2-1}).$$

Let T^* be the smallest t such that $\Phi_{-t}^{0}(\hat{0}) = \Phi_{-t}^{0}(\hat{1})$. Define $\bar{d}(k) = \max_{\pi_1,\pi_2} ||\pi_1^k - \pi_2^k||$ for particular k, where π^k is the distribution governing the Markov chain at time k when started at time 0 in a random state governed by the distribution π. The mixing time threshold T_{mix} is defined to be the smallest k for which $\bar{d}(k) \leq 1/e$. Then,

$$E[T^*] \leq 2T_{mix}(1 + \ln C).$$

[2] More formally, we have to use the same instance F_{-t} of the random mapping ϕ at time $-t$ during all the iterations of the algorithm.

Proof. Propp and Wilson proved that $E[T^*] \leq 2T_{mix}(1 + \ln l)$ where l is the length of the longest chain in the partially ordered state space [12]. Suppose that there are l distinct states such that $s_0 \leq s_1 \leq \cdots \leq s_l$. For l to be the length of the longest chain, s_0 should be $\hat{0}$ and s_l should be $\hat{1}$. The number of jobs of each machine in s_{i+1} is greater than or equal to the number of jobs in corresponding machine of s_i. Since the two states are distinct, s_{i+1} must have at least one more job than s_i. Recall that the difference between the number of jobs in $\hat{0}$ and $\hat{1}$ is C. Thus there can be at most C distinct states between $\hat{0}$ and $\hat{1}$, and l cannot exceed C.

We will now explore the implications of the above theorem for the running time of our scheme for scenario 1 (the implications for scenario 2 can be similarly obtained). Let λ be the total arrival rate for all the job types. Since there are m machines, there are an average of μm departure events and λ arrival events during each step. We can store all the arrival/departure events in a priority queue, which can be implemented using heaps. This allows us to find the next event to be processed in time $O(\log n)$. Using the fact that T^* has an exponential tail, we can claim that the expected running time is $\tilde{O}(T_{mix}(1 + \ln C)(\lambda + \mu m))$ – we omit the details from this abstract.

Note that we do not need to define Markov chains to make a random sample using our scheme. In fact it is very hard to define Markov chains sometimes because
1) State space may be too large ($\Pi_{i=1}^{m} c_i$).
2) The scheduler may be too complicated, or given as a black box.

It is interesting (and a testament to the efficacy of the CFTP paradigm) that we can get an *exact* sample from the stationary distribution of the Markov chain without any knowledge of or assumptions about the structure of the Markov chain beyond ergodicity and monotonicity of the scheduler, and that we can prove good bounds on the expected running time of our algorithm.

3.4 Extensions

The above algorithm can be adapted in a straight forward fashion to

The continuous case: This is the case where the arrival process is continuous and the service times are exponential rather than geometric.

Randomized Schedulers: The notion of monotonicity that we require is that of vector stochastic dominance. More formally, for each state s and job type i there must be a random variable $X(i, s)$ such that

1. $\mathbf{Pr}[X(i, s) = t]$ is the same as the probability that the state changes from s to t as a result of the scheduling of job i.
2. $s_1 \leq s_2 \Rightarrow X(i, s_1) \leq X(i, s_2)$.

Such a scheduler will allow us to couple the evolution of states so that the resulting Markov chain is monotonic.

4 Simulation Results

In this section we will present simulation results only for the first scenario. As mentioned in the introduction, we can tune a black box scheduler using the extracted samples. We will also present an example to tune a black box random scheduler.

We will not show the detail of our code in this paper. However let us explain the simulation setting. We have used a simple(Min-load) scheduler for the first scenario. When a job arrives, the scheduler looks at the loads of all machines to which the job can be assigned, and assigns the job to the minimum loaded machine among them. This is a simple monotonic scheduling strategy. We need some input parameters as follows.

1. average arrival rate
2. average processing time
3. number of machines
4. capacities of machines
5. job types distribution
6. job type assignability matrix

We will not specify the job types in detail in this abstract. We simulated the case where there are 10 machines and 10 job types. We set the total arrival rate equal to 10 times the service rate. Figure 1 shows the running time of our program in seconds as the capacities of machines are increased. The basic case is that each machine has a capacity between 1 and 15. We multiplied the capacities by n where n is between 1 and 500. The running time seems almost linear in the multiplication constant n.

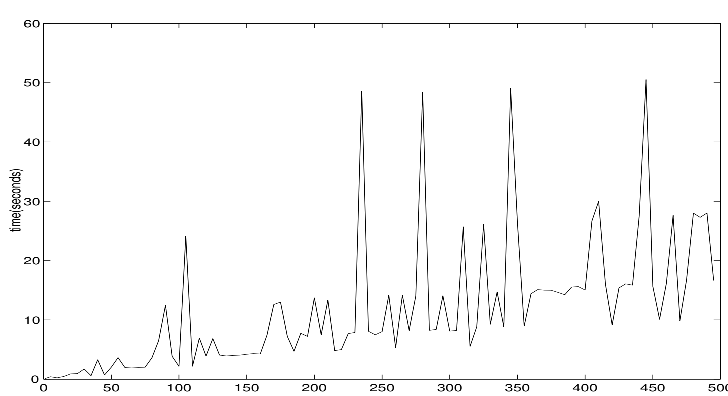

Fig. 1. Running time of our program for scenario 1

Since we can extract a random sample efficiently using our scheme, we can use it to tune a random scheduler. We illustrate this process for a simple random scheduler that assigns a job according to assigning probability matrix $\tilde{V}$ such that $\tilde{V}_{i,j}$ is the probability that a job type i is assigned to a machine j. Of course $\tilde{V}_{i,j} = 0$ if job type i can not be scheduled on machine j. We illustrate a very simple optimizer in figure 2. Assuming that there are not too many arriving jobs and each machine has enough capacity, this optimizer will tune the assigning probability matrix for random scheduler. We simulated the case that we have 10 machines, 10 job types, [1-15] capacities and 10% tolerance rate. Figure 3 shows the evolution of the random scheduler. As the tuning process progresses, the number of machines with completely filled up buffers goes down, indicating an improvement in the performance of the scheduler. Notice that the tuning process does not involve any knowledge of the arrival rates.

Tune_Rand_Schd($\tilde{V}$: assigning probability matrix, t: tolerance rate)

while True
    Extract k samples.
    $f =$ total number of full machines in the k samples
    if $f/k > t$
        Improve $\tilde{V}$.
    else
        break

Fig. 2. A simple optimizer for random scheduler

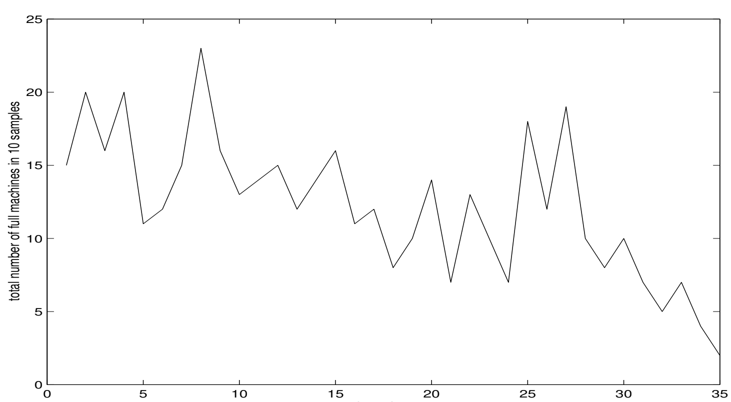

Fig. 3. Evolution of the random scheduler

Acknowledgements

Ashish Goel would like to thank Chandra Chekuri for introducing him to the problem and for several useful discussions.

References

1. D.J. Aldous. A random walk construction of uniform spanning trees and uniform labelled trees. *SIAM Journal on Discrete Mathematics*, 3(4):450–465, 1990.
2. S. Asmussen, P. Glynn, and H. Thorisson. Stationary detection in the initial transient problem. *ACM trans. on modeling and comp. simulation*, 2(2):130–157, 1992.
3. A. Broder. Generating random spanning trees. *30th Annual Symposium on Foundations of Computer Science*, pages 442–447, 1989.
4. C. Chekuri, K. Ramanan, P. Whiting, and L. Zhang. Blocking probability estimates in a partitioned sector TDMA system. *Fourth International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (Dial M for Mobility)*, 2000.
5. P. Diaconis and L. Saloff-Coste. What do we know about the metropolis algorithm? *Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 112–129, 1995.
6. J. Fill. An interruptible algorithm for perfect sampling via Markov chains. *Annals of Applied Probability*, 8(1):131–162, 1998.
7. D. Hochbaum. *Approximation algorithms for NP-hard problems.* PWS Publishing Company, 1997.
8. M. Huber. Exact sampling and approximate counting techniques. *30th ACM Symposium on the Theory of Computing*, pages 31–40, 1998.
9. M. Jerrum and A. Sinclair. The Markov chain Monte Carlo method: an approach to approximate counting and integration. *In "Approximation Algorithms for NP-hard Problems," D.S.Hochbaum ed.*, 1996.
10. L. Lovsz and P. Winkler. Exact mixing in an unknown Markov chain. *Electronic Journal of Combinatorics, 2, paper #R15*, 1995.
11. R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.
12. J.G. Propp and D.B. Wilson. Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structure & Algorithms*, 9:223–252, 1996.