NUMERICAL SOLUTION of MARKOV CHAINS, p. 99–120

Hypergraph-based Parallel Computation of Passage Time Densities in Large Semi-Markov Models

* Jeremy T. Bradley[1], Nicholas J. Dingle[1], William J. Knottenbelt[1], Helen J. Wilson[2]

[1] *Department of Computing, Imperial College London, South Kensington Campus, London SW7 2AZ, United Kingdom, {jb,njd200,wjk}@doc.ic.ac.uk*
[2] *Department of Applied Mathematics, University of Leeds, Woodhouse Lane, Leeds LS2 9JT, United Kingdom, h.j.wilson@leeds.ac.uk*

KEY WORDS: Semi-Markov chains, passage time densities, hypergraph partitioning

ABSTRACT

Passage time densities and quantiles are important performance and quality of service metrics, but their numerical derivation is, in general, computationally expensive. We present an iterative algorithm for the calculation of passage time densities in semi-Markov models, along with a theoretical analysis and empirical measurement of its convergence behaviour. In order to implement the algorithm efficiently in parallel, we use hypergraph partitioning to minimise communication between processors and to balance workloads. This enables the analysis of models with very large state spaces which could not be held within the memory of a single machine. We produce passage time densities and quantiles for very large semi-Markov models with over 15 million states and validate the results against simulation.

1. Introduction

A rapid response time is an important performance criterion for almost all computer-communication and transaction processing systems. Examples of systems with stringent response time requirements include mobile communication systems, stock market trading systems, web servers, database servers, flexible manufacturing systems, communication protocols and communication networks. Typically, response time targets are specified in terms of quantiles – for example "95% of all text messages must be delivered within 3 seconds".

*Correspondence to: Jeremy Bradley, Department of Computing, Imperial College London, South Kensington Campus, London SW7 2AZ, United Kingdom, jb@doc.ic.ac.uk

Response time quantiles are frequently specified as key quality of service metrics in Service Level Agreements and industry standard benchmarks such as TPC.

In the context of the high-level models used by performance analysts (e.g. queueing networks and stochastic Petri nets), response times can be specified as *passage* times – that is, the time taken to enter any one of a set of target states starting from a specified set of source states in the underlying Markov or semi-Markov chain. Traditional steady-state performance analysis of such models is adequate to predict standard resource-based measures such as utilisation and throughput, but is inadequate to predict passage time densities and quantiles.

The focus of the present study is on semi-Markov processes [24], a generalisation of Markov processes which support arbitrary state holding times. Techniques already exist for the practical extraction of passage time densities and quantiles from large purely Markovian systems [11, 14, 17]. Until now, however, only relatively small semi-Markov systems have been analysed for passage time quantities [5, 12, 15].

In this paper, we present a scalable iterative passage time density extraction algorithm for very large semi-Markov processes (SMPs). This extends our preliminary work on passage time extraction [6] with significant theoretical and empirical convergence results for the iterative algorithm. Our approach is based on the calculation and subsequent numerical inversion of the Laplace transform [1, 2] of the desired passage time density. In [12, 15], the time complexity of the numerical derivation of passage time and transient quantities for a semi-Markov system with N states, using the Laplace domain, is $O(N^4)$. This approach is dominated by the complexity of maintaining the Laplace transforms of state holding time distributions in closed form. We solve this problem by characterising a distribution by the samples from its Laplace transform that are ultimately required for the inversion process. The result is an iterative algorithm which calculates arbitrary semi-Markov passage times in $O(N^2 r)$ time, for r iterations.

The data partitioning strategy employed is key to the scalability (in terms of both efficiency and capacity) of all parallel algorithms. We exploit recent advances in the application of hypergraph data structures and their partitioning [8] to minimise inter-processor communication while balancing computational load.

The rest of this paper is organised as follows: Section 2 summarises the theory relating to the computation of passage time densities in semi-Markov processes. Our iterative passage time algorithm and its theoretical convergence analysis are detailed in Section 3. Section 4 addresses the problem of general distribution representation, while Section 5 discusses the application of hypergraph partitioning techniques. Section 6 presents a complete parallel passage time analysis pipeline. Section 7 presents numerical results showing passage time densities extracted from two very large semi-Markov models, as well as scalability and convergence results.

2. Definitions and Background Theory

2.1. Semi-Markov Processes

Consider a Markov renewal process $\{(X_n, T_n) : n \geq 0\}$ where T_n is the time of the nth transition ($T_0 = 0$) and $X_n \in \mathcal{S}$ is the state at the nth transition. Let the kernel of this process be:

$$R(n, i, j, t) = \mathbb{P}(X_{n+1} = j, T_{n+1} - T_n \leq t \mid X_n = i) \tag{1}$$

for $i, j \in \mathcal{S}$. The continuous time semi-Markov process (SMP), $\{Z(t), t \geq 0\}$, defined by the kernel R, is related to the Markov renewal process by:

$$Z(t) = X_{N(t)} \tag{2}$$

where $N(t) = \max\{n : T_n \leq t\}$, i.e. the number of state transitions that have taken place by time t. Thus $Z(t)$ represents the state of the system at time t. We consider time-homogeneous SMPs, in which $R(n, i, j, t)$ is independent of any previous state except the last. Thus R becomes independent of n:

$$\begin{aligned} R(i, j, t) &= \mathbb{P}(X_{n+1} = j, T_{n+1} - T_n \leq t \mid X_n = i) \quad \text{for any } n \geq 0 \\ &= p_{ij} H_{ij}(t) \end{aligned} \tag{3}$$

where $p_{ij} = \mathbb{P}(X_{n+1} = j \mid X_n = i)$ is the state transition probability between states i and j and $H_{ij}(t) = \mathbb{P}(T_{n+1} - T_n \leq t \mid X_{n+1} = j, X_n = i)$, is the sojourn time distribution in state i when the next state is j.

2.2. First passage times

Consider a finite, irreducible, continuous-time semi-Markov process with N states $\{1, 2, \ldots, N\}$. Recalling that $Z(t)$ denotes the state of the SMP at time t ($t \geq 0$), the first passage time from a source state i at time t into a non-empty set of target states $\vec{j}$ is:

$$P_{i\vec{j}}(t) = \inf\{u > 0 : Z(t + u) \in \vec{j}, N(t + u) > N(t), Z(t) = i\} \tag{4}$$

For a stationary time-homogeneous SMP, $P_{i\vec{j}}(t)$ is independent of t and we have:

$$P_{i\vec{j}} = \inf\{u > 0 : Z(u) \in \vec{j}, N(u) > 0, Z(0) = i\} \tag{5}$$

This formulation of the random variable $P_{i\vec{j}}$ applies to an SMP with no immediate (that is, zero-time) transitions. If zero-time transitions are permitted in the model then the passage time can be stated as:

$$P_{i\vec{j}} = \inf\{u > 0 : N(u) \geq M_{i\vec{j}}\} \tag{6}$$

where $M_{i\vec{j}} = \min\{m \in \mathbb{Z}^+ : X_m \in \vec{j} \mid X_0 = i\}$ is the transition marking the terminating state of the passage.

$P_{i\vec{j}}$ has an associated probability density function $f_{i\vec{j}}(t)$ such that the passage time quantile is given as:

$$\mathbb{P}(t_1 < P_{i\vec{j}} < t_2) = \int_{t_1}^{t_2} f_{i\vec{j}}(t)\,\mathrm{d}t \tag{7}$$

In general, the Laplace transform of $f_{i\vec{j}}$, $L_{i\vec{j}}(s)$, can be computed by solving a set of N linear equations:

$$L_{i\vec{j}}(s) = \sum_{k \notin \vec{j}} r^*_{ik}(s) L_{k\vec{j}}(s) + \sum_{k \in \vec{j}} r^*_{ik}(s) \qquad : \text{for } 1 \leq i \leq N \tag{8}$$

where $r^*_{ik}(s)$ is the Laplace-Stieltjes transform (LST) of $R(i,k,t)$ from Section 2.1 and is defined by:

$$r^*_{ik}(s) = \int_0^\infty e^{-st}\, \mathrm{d}R(i,k,t) \tag{9}$$

Eq. (8) has a matrix-vector form, $A\tilde{x} = \tilde{b}$, where the elements of A are general complex functions; care needs to be taken when storing such functions for eventual numerical inversion (see Section 4). For example, when $\vec{j} = \{1\}$, Eq. (8) yields:

$$\begin{pmatrix} 1 & -r^*_{12}(s) & \cdots & -r^*_{1N}(s) \\ 0 & 1 - r^*_{22}(s) & \cdots & -r^*_{2N}(s) \\ 0 & -r^*_{32}(s) & \cdots & -r^*_{3N}(s) \\ \vdots & \vdots & \ddots & \vdots \\ 0 & -r^*_{N2}(s) & \cdots & 1 - r^*_{NN}(s) \end{pmatrix} \begin{pmatrix} L_{1\vec{j}}(s) \\ L_{2\vec{j}}(s) \\ L_{3\vec{j}}(s) \\ \vdots \\ L_{N\vec{j}}(s) \end{pmatrix} = \begin{pmatrix} r^*_{11}(s) \\ r^*_{21}(s) \\ r^*_{31}(s) \\ \vdots \\ r^*_{N1}(s) \end{pmatrix} \tag{10}$$

When there are multiple source states, denoted by the vector $\vec{i}$, the Laplace transform of the passage time density at steady-state is:

$$L_{\vec{i}\vec{j}}(s) = \sum_{k \in \vec{i}} \alpha_k L_{k\vec{j}}(s) \tag{11}$$

where the weight α_k is the probability at equilibrium that the system is in state $k \in \vec{i}$ at the starting instant of the passage. If $\tilde{\pi}$ denotes the steady-state vector of the embedded discrete-time Markov chain (DTMC) with one-step transition probability matrix $P = [p_{ij}, 1 \leq i,j \leq N]$, then α_k is given by:

$$\alpha_k = \begin{cases} \pi_k / \sum_{j \in \vec{i}} \pi_j & \text{if } k \in \vec{i} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

The vector with components α_k is denoted by $\tilde{\alpha}$.

3. Iterative Passage Time Analysis

3.1. Overview

In this section, we describe an algorithm for generating passage time densities that creates successively better approximations to the SMP passage time quantity of Eq. (8). Our technique considers the rth transition passage time of the system, $P^{(r)}_{i\vec{j}}$. This is the conditional passage time of the system having reached any of the specified target states within r state-transitions. The unconditioned passage time random variable, $P_{i\vec{j}}$, is then obtained in the limit as $r \to \infty$.

We calculate the Laplace transform of $P_{i\vec{j}}^{(r)}$, $L_{i\vec{j}}^{(r)}(s)$, and pick a sufficiently high value of r to give an approximation to $L_{i\vec{j}}(s)$ to within a specified degree of accuracy. $L_{i\vec{j}}(s)$ can then be numerically inverted to obtain the desired passage time density $f_{i\vec{j}}(t)$.

This iterative method bears a loose resemblance to the well-known *uniformisation* technique [4, 22, 23] which can be used to generate transient-state distributions and passage time densities for Markov chains. However, as we are working with semi-Markov systems, there can be no *uniformising* of the general distributions in the SMP. The general distribution information has to be maintained as precisely as possible throughout the process, which we achieve using the representation technique described in Section 4.

3.2. Technical Description

Recall the semi-Markov process $Z(t)$ of Section 2.1, where $N(t)$ is the number of state transitions that have taken place by time t. We formally define the rth transition first passage time to be:

$$P_{i\vec{j}}^{(r)} = \inf\{u > 0 : Z(u) \in \vec{j}, 0 < N(u) \leq r, Z(0) = i\} \tag{13}$$

which is the time taken to enter a state in $\vec{j}$ for the first time having started in state i at time 0 and having undergone up to r state transitions[†]. $P_{i\vec{j}}^{(r)}$ is a random variable with associated Laplace transform $L_{i\vec{j}}^{(r)}(s)$. $L_{i\vec{j}}^{(r)}(s)$ is, in turn, the ith component of the vector:

$$\tilde{L}_{\vec{j}}^{(r)}(s) = (L_{1\vec{j}}^{(r)}(s), L_{2\vec{j}}^{(r)}(s), \ldots, L_{N\vec{j}}^{(r)}(s)) \tag{14}$$

representing the passage time for terminating in $\vec{j}$ for each possible start state. This vector may be computed as:

$$\tilde{L}_{\vec{j}}^{(r)}(s) = U(I + U' + U'^2 + \cdots + U'^{(r-1)})\,\tilde{e}_{\vec{j}} \tag{15}$$

where U is a matrix with elements $u_{pq} = r_{pq}^*(s)$ and U' is a modified version of U with elements $u'_{pq} = \delta_{p\notin\vec{j}}\, u_{pq}$, where states in $\vec{j}$ have been made absorbing. We include the initial multiplication with U in Eq. (15), so as to generate cycle times for cases such as $L_{ii}^{(r)}(s)$ which would otherwise register as 0 if U' were used instead. The column vector $\tilde{e}_{\vec{j}}$ has entries $e_{k\vec{j}} = \delta_{k\in\vec{j}}$.

From Eq. (5) and Eq. (13):

$$P_{i\vec{j}} = P_{i\vec{j}}^{(\infty)} \quad \text{and thus} \quad L_{i\vec{j}}(s) = L_{i\vec{j}}^{(\infty)}(s) \tag{16}$$

We can generalise to multiple source states $\vec{i}$ using the vector $\tilde{\alpha}$ of Eq. (12):

$$\begin{aligned} L_{\vec{i}\vec{j}}^{(r)}(s) &= \tilde{\alpha}\tilde{L}_{\vec{j}}^{(r)}(s) \\ &= \textstyle\sum_{k=0}^{r-1} \tilde{\alpha} U U'^k\, \tilde{e}_{\vec{j}} \end{aligned} \tag{17}$$

[†]If we have immediate transitions in our SMP model (as in Eq. (6)) then the rth transition first passage time is $P_{i\vec{j}}^{(r)} = \inf\{u > 0 : M_{i\vec{j}} \leq N(u) \leq r\}$.

The sum of Eq. (17) can be computed efficiently using sparse matrix-vector multiplications with a vector accumulator, $\tilde{\mu}_r = \sum_{k=0}^{r} \tilde{\alpha} U U'^k$. At each step, the accumulator (initialised as $\tilde{\mu}_0 = \tilde{\alpha} U$) is updated as $\tilde{\mu}_{r+1} = \tilde{\alpha} U + \tilde{\mu}_r U'$. The worst-case time complexity for this sum is $O(N^2 r)$ versus the $O(N^3)$ of typical matrix inversion techniques. In practice, for a sparse matrix with constant number of non-zeros per row, this can be as low as $O(Nr)$.

3.3. Analytic Convergence and Truncation Error Analysis

In the iterative passage time calculation of Eq. (17), we approximate the Laplace transform of the passage time density $L_{\vec{i}\vec{j}}(s)$ by a sum of the form $\sum_{i=0}^{k} \tilde{x} A^i$. In this section, we demonstrate that this sum converges onto the analytic solution of Eq. (8); also, by using the Power method to provide an expression for the dominant eigenvalue, we derive an approximation for the truncation error after k iterations.

3.3.1. Limit and Convergence Let us consider the finite sum $\sum_{i=0}^{k} \tilde{x} A^i$ where A has dominant eigenvalue λ_1. If we post-multiply this expression by $(I - A)$, we obtain:

$$\sum_{i=0}^{k} \tilde{x} A^i (I - A) = \tilde{x} - \tilde{x} A^{k+1} \tag{18}$$

and so, if the modulus of the dominant eigenvalue of A, $|\lambda_1| < 1$, then $\lim_{k \to \infty} \tilde{x} A^{k+1} = 0$, and we obtain:

$$\sum_{i=0}^{\infty} \tilde{x} A^i (I - A) = \tilde{x} \quad \text{and thus} \quad \sum_{i=0}^{\infty} \tilde{x} A^i = \tilde{x} (I - A)^{-1}. \tag{19}$$

We can relate the above analysis to the iterative passage time algorithm by taking $\tilde{x} = \tilde{\alpha} U$ and $A = U'$. We know that $|\lambda_1| < 1$ for U' as the sum of Eq. (15) converges as $r \to \infty$ according to Eq. (16). This demonstrates that our iterative scheme will converge in the limit to the analytic solution of Eq. (8).

3.3.2. Truncation Error Analysis Let A have eigenvalues λ_j and associated eigenvectors $\tilde{v}_j$, with dominant eigenvalue λ_1 as above. In general, the $\tilde{v}_j$ form a basis and so for an arbitrary vector $\tilde{x}$:

$$\begin{aligned} \tilde{x} &= \sum_j a_j \tilde{v}_j \\ \tilde{x} A^i &= \sum_j a_j \lambda_j^i \tilde{v}_j \\ &= a_1 \lambda_1^i \tilde{v}_1 + \sum_{j>1} a_j \lambda_j^i \tilde{v}_j \\ \tilde{x} A^k &\sim a_1 \lambda_1^k \tilde{v}_1 \qquad : \text{as } k \longrightarrow \infty \end{aligned} \tag{20}$$

Thus the quantity $\tilde{x} A^k$ is dominated by the largest eigenvalue, with the rate of asymptotic convergence being governed by the ratio of the moduli of the dominant and sub-dominant eigenvalues, $|\lambda_1|/|\lambda_2|$. Eq. (20) yields $\tilde{x}A^k \simeq \lambda_1 \tilde{x}A^{k-1}$ for large k; so, by right-multiplying both sides by $(\tilde{x}A^k)^*$, we may approximate λ_1 with:

$$\lambda_1 \simeq \frac{|\tilde{x}A^k|^2}{(\tilde{x}A^{k-1}) \,.\, (\tilde{x}A^k)^*} \tag{21}$$

where for $\tilde{\omega}$, a vector with complex elements ω_i, the complex conjugate $\tilde{\omega}^*$ has elements ω_i^*.

The error incurred in truncating the sum at the kth term is $\sum_{i=k+1}^{\infty} \tilde{x}A^i$. Using the approximation $\tilde{x}A^i \simeq a_1 \lambda_1^i \tilde{v}_1$ for $i \geq k$ yields:

$$\begin{aligned} \sum_{i=k+1}^{\infty} \tilde{x}A^i &= \sum_{i=1}^{\infty} \tilde{x}A^{k+i} \\ &\simeq \sum_{i=1}^{\infty} \lambda_1^i \tilde{x}A^k \\ &= \frac{\lambda_1}{1-\lambda_1} \tilde{x}A^k \end{aligned} \tag{22}$$

providing $|\lambda_1| < 1$ as in Section 3.3.1.

Thus an approximate convergence condition is to find the minimum k such that:

$$\left| \frac{\lambda_1}{1-\lambda_1} \tilde{x}A^k \right| < \epsilon \tag{23}$$

where λ_1 is approximated as in Eq. (21).

3.4. Practical Convergence

In practice, convergence of the sum $L_{\vec{i}\vec{j}}^{(r)}(s) = \sum_{k=0}^{r-1} \tilde{\alpha} U U'^k$ can be said to have occurred if, for a particular r and s-point:

$$|\mathrm{Re}(L_{\vec{i}\vec{j}}^{(r+1)}(s) - L_{\vec{i}\vec{j}}^{(r)}(s))| < \epsilon \quad \text{and} \quad |\mathrm{Im}(L_{\vec{i}\vec{j}}^{(r+1)}(s) - L_{\vec{i}\vec{j}}^{(r)}(s))| < \epsilon \tag{24}$$

where ϵ is chosen to be a suitably small value, say $\epsilon = 10^{-16}$. We present empirical observations on the convergence behaviour of this technique (i.e. the order of r) in Section 7.

An optimisation which makes the sum converge more quickly (assuming the approximation of Eq. (21) for λ_1 is a good one), is to use the approximate truncation error to improve the accuracy of the calculation. For each iteration we can take:

$$L_{\vec{i}\vec{j}}^{(r)}(s) = \sum_{k=0}^{r-1} \tilde{\alpha} U U'^k + \frac{\lambda_1}{1-\lambda_1} \tilde{\alpha} U U'^{(r-1)} \tag{25}$$

We can then compare successive $L_{\vec{i}\vec{j}}^{(r)}(s)$ in the real and imaginary parts as before. If $|\lambda_1| = |\lambda_i| < 1$ for one or more values of $i \neq 1$ then convergence of $L_{\vec{i}\vec{j}}^{(r)}(s)$ will still occur, but the error dynamics are more complicated and we are forced to resort to the stricter notion of convergence above.

4. Distribution Representation and Laplace Inversion

The key to practical analysis of semi-Markov processes lies in the efficient representation of their general distributions. Without care the structural complexity of the SMP can be recreated within the representation of the distribution functions. This is especially true with the manipulations performed in the iterative passage time calculation of Section 3.

Many techniques have been used for representing arbitrary distributions – two of the most popular being *phase-type distributions* and *vector-of-moments* methods. These methods suffer from, respectively, exploding representation size under composition, and containing insufficient information to produce accurate answers after large amounts of composition.

As all our distribution manipulations take place in Laplace-space, we link our distribution representation to the Laplace inversion technique that we ultimately use. Our tool supports two Laplace transform inversion algorithms, which are briefly outlined below: the Euler technique [3] and the Laguerre method [1] with modifications summarised in [14].

Both algorithms work on the same general principle of sampling the transform function $L(s)$ at n points, $s_1, s_2, \ldots, s_n$ and generating values of $f(t)$ at m user-specified t-points $t_1, t_2, \ldots, t_m$. In the Euler inversion case $n = km$, where k can vary between 15 and 50, depending on the accuracy of the inversion required. In the modified Laguerre case, $n = 400$ and, crucially, is independent of m (see Section 4.2).

The process of selecting a Laplace transform inversion algorithm is discussed later; however, whichever is chosen, it is important to note that calculating $s_i, 1 \leq i \leq n$ and storing all our distribution transform functions, sampled at these points, will be sufficient to provide a complete inversion. Key to this is the fact that matrix element operations, of the type performed in Eq. (17), (i.e. convolution and weighted sum) do not require any adjustment to the array of domain s-points required. In the case of a convolution, for instance, if $L_1(s)$ and $L_2(s)$ are stored in the form $\{(s_i, L_j(s_i)) \ : \ 1 \leq i \leq n\}$, for $j = 1, 2$, then the convolution, $L_1(s)L_2(s)$, can be stored using the same size array and using the same list of domain s-values, $\{(s_i, L_1(s_i)L_2(s_i)) \ : \ 1 \leq i \leq n\}$.

Storing our distribution functions in this way has three main advantages. Firstly, the function has constant storage space, independent of the distribution-type. Secondly, each distribution has, therefore, the same constant storage requirement even after composition with other distributions. Finally, the function has sufficient information about a distribution to determine the required passage time (and no more).

4.1. Summary of Euler Inversion

The Euler method is based on the Bromwich contour inversion integral, expressing the function $f(t)$ in terms of its Laplace transform $L(s)$. Making the contour a vertical line $s = a$ such that $L(s)$ has no singularities on or to the right of it gives:

$$f(t) = \frac{2e^{at}}{\pi} \int_0^\infty \mathrm{Re}(L(a + iu)) \cos(ut) \, \mathrm{d}u \tag{26}$$

This integral can be numerically evaluated using the trapezoidal rule with step-size $h = \pi/2t$ and $a = A/2t$ (where A is a constant that controls the discretisation error), which results in the nearly alternating series:

$$f(t) \approx f_h(t) = \frac{e^{A/2}}{2t}\mathrm{Re}(L(A/2t)) + \frac{e^{A/2}}{2t}\sum_{k=1}^{\infty}(-1)^k \mathrm{Re}\left(L\left(\frac{A+2k\pi i}{2t}\right)\right) \tag{27}$$

Euler summation is employed to accelerate the convergence of the alternating series infinite sum, so we calculate the sum of the first n terms explicitly and use Euler summation to calculate the next m. To give an accuracy of 10^{-8} we set $A = 19.1$, $n = 20$ and $m = 12$ (compared with $A = 19.1$, $n = 15$ and $m = 11$ in [3]).

4.2. Summary of Laguerre Inversion

The Laguerre method [1] makes use of the Laguerre series representation:

$$f(t) = \sum_{n=0}^{\infty} q_n l_n(t) \qquad : t \geq 0 \tag{28}$$

where the Laguerre polynomials l_n are given by:

$$l_n(t) = \left(\frac{2n-1-t}{n}\right) l_{n-1}(t) - \left(\frac{n-1}{n}\right) l_{n-2}(t) \tag{29}$$

starting with $l_0 = e^{t/2}$ and $l_1 = (1-t)e^{t/2}$, and:

$$q_n = \frac{1}{2\pi r^n}\int_0^{\pi} Q(re^{iu})e^{-iru}\,\mathrm{d}u \tag{30}$$

where $r = (0.1)^{4/n}$ and $Q(z) = (1-z)^{-1}L((1+z)/2(1-z))$.

The integral in Eq. (30) can be approximated numerically by the trapezoidal rule, giving:

$$q_n \approx \bar{q}_n = \frac{1}{2nr^n}\left(Q(r) + (-1)^n Q(-r) + 2\sum_{j=1}^{n-1}(-1)^j \mathrm{Re}\left(Q(re^{\pi ji/n})\right)\right) \tag{31}$$

As described in [14], the Laguerre method can be modified by noting that the Laguerre coefficients q_n are independent of t. This means that if the number of trapezoids used in the evaluation of q_n is fixed to be the same for every q_n (rather than depending on the value of n), values of $Q(z)$ (and hence $L(s)$) can be reused after they have been computed. Typically, we set $n = 200$. In order to achieve this, however, the scaling method described in [1] must be used to ensure that the Laguerre coefficients have decayed to (near) 0 by $n = 200$. If this can be accomplished, the inversion of a passage time density for any number of t-values can be achieved at the fixed cost of calculating 400 truncated summations of the type shown in Eq. (17). This is in contrast to the Euler method, where the number of truncated summations required is a function of the number of points at which the value of $f(t)$ is required.

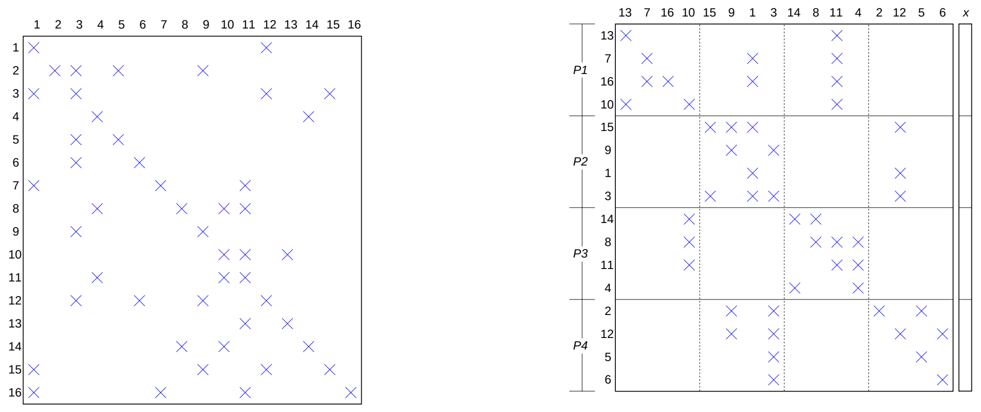
Fig. 1. A 16 × 16 non-symmetric sparse matrix (left), with corresponding 4-way hypergraph partition (right) and corresponding partitions of the vector

4.3. Automatic Inversion Algorithm Selection

We have implemented a distributed Laplace transform inverter which automatically selects which inversion algorithm is the most suitable for a given model in the following manner. Firstly, it attempts to scale the Laguerre coefficients (as described in the previous section) and if this scaling is successful the modified Laguerre method is used to invert the Laplace transform $L_{\vec{i}\vec{j}}(s)$ and yield the required passage time density. If, however, it proves impossible to scale the Laguerre coefficients – typically when the density function contains discontinuities – then the tool switches to the Euler inversion method. This ensures that the Laguerre method is used whenever possible as it minimises the amount of computation needed, while the ability to calculate pathological passage time densities using the Euler method is preserved at the cost of requiring more computation.

5. Hypergraph Partitioning

As discussed in Section 3.2, the core operation in our algorithm is the repeated sparse matrix-vector multiplication of Eq. (17). In order to exploit the combined processing power and memory capacity of several processors to compute passage time densities for very large systems (with state spaces of the order of 10^7 states or more), it is necessary to partition the sparse matrix U' across the processors. Such a scheme will necessitate the exchange of data (vector elements and possibly partial sums) after every iteration in the solution process. The objective in partitioning the matrix is to minimise the amount of data which needs to be exchanged while balancing the computational load (as given by the number of non-zero elements of U' assigned to each processor).

Hypergraph partitioning is an extension of graph partitioning. Its primary application to date has been in VLSI circuit design, where the objective is to cluster pins of devices such that interconnect is minimised. It can also be applied to the problem of allocating the non-zero elements of sparse matrices across processors in parallel computation [8].

Formally, a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined by a set of vertices $\mathcal{V}$ and a set of nets (or hyperedges) $\mathcal{N}$, where each net is a subset of the vertex set $\mathcal{V}$ [8]. In the context of a row-wise decomposition of a sparse matrix A, matrix row i $(1 \leq i \leq n)$ is represented by a vertex $v_i \in \mathcal{V}$ while column j $(1 \leq j \leq n)$ is represented by net $N_j \in \mathcal{N}$. The vertices contained within net N_j correspond to the row numbers of the non-zero elements within column j, i.e. $v_i \in N_j$ if and only if $a_{ij} \neq 0$. The weight of vertex i is given by the number of non-zero elements in row i, while the weight of a net is its contribution to the edge cut, which is defined as one less than the number of different partitions spanned by that net. The overall objective of a hypergraph sparse matrix partitioning is to minimise the sum of the weights of the cut nets while maintaining a balance criterion. A column-wise decomposition is achieved in an analogous fashion.

The matrix on the right of Fig. 1 shows the result of applying hypergraph-partitioning to the matrix on the left in a four-way row-wise decomposition. Although the number of off-diagonal non-zeros is 18 the number of vector elements which must be transmitted between processors during each matrix-vector multiplication (the communication cost) is 6. This is because the hypergraph partitioning algorithms not only aim to concentrate the non-zeros on the diagonals but also strive to line up the off-diagonal non-zeros in columns. The edge cut of the decomposition is also 6, and so the hypergraph partitioning edge cut metric exactly quantifies the communication cost. This is a general property and one of the key advantages of using hypergraphs – in contrast to graph partitioning, where the edge cut metric merely approximates communication cost. Optimal hypergraph partitioning is NP-complete but there are a small number of hypergraph partitioning tools which implement fast heuristic algorithms, for example PaToH [8] and hMeTiS [16].

6. Parallel Pipeline

The process of calculating a passage time density (shown in Fig. 2) begins with a high-level model specified in an enhanced form of the DNAmaca interface language [18, 19]. This language supports the specification of queueing networks, stochastic Petri nets and stochastic process algebras. Next, a probabilistic, hash-based state generator [21] uses the high-level model description to produce the transition probability matrix P of the model's embedded Markov chain, the matrices U and U', and a list of the initial and target states. Normalised weights for the initial states are determined by the solution of $\tilde{\pi} = \tilde{\pi}P$, which is readily done using any of a variety of steady-state solution techniques (e.g. [10, 20]). U' is then partitioned using a hypergraph partitioning tool.

Control is then passed to the distributed passage time density calculator, which is implemented in C++ using the Message Passing Interface (MPI) [13] standard. This employs a master-slave architecture with groups of slave processors. The master processor computes in advance the

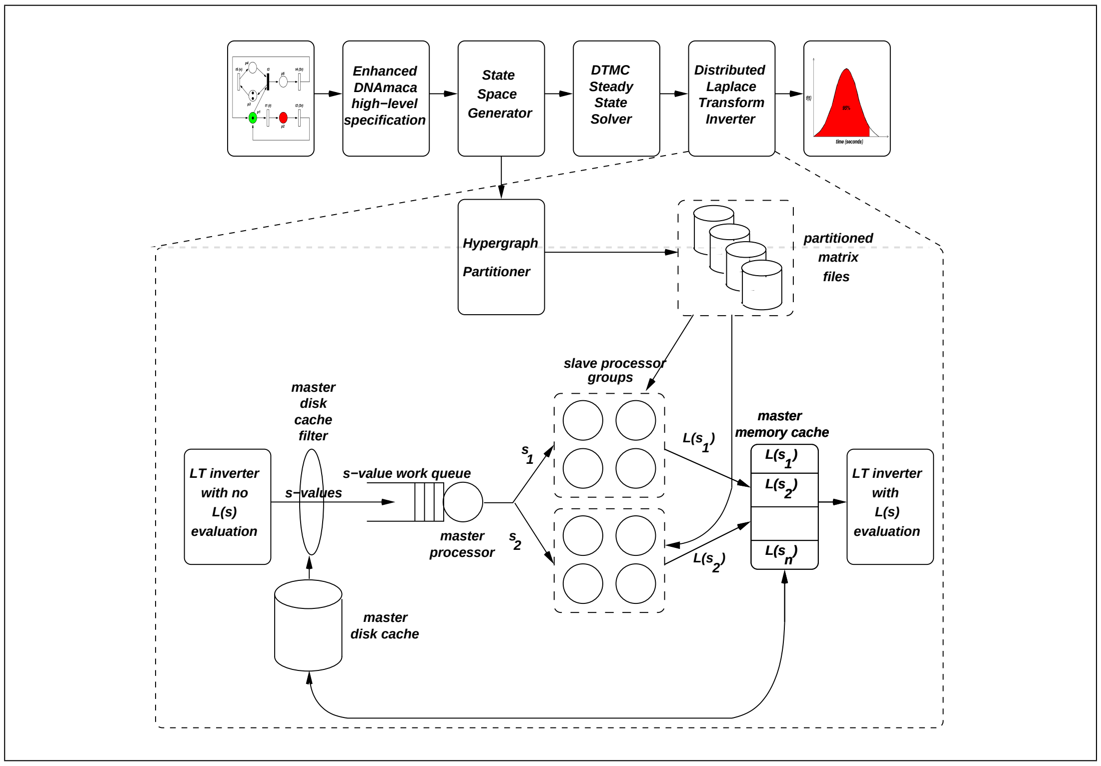

Fig. 2. Parallel hypergraph-based passage time density calculation pipeline

values of s at which it will need to know the value of $L_{\vec{i}\vec{j}}(s)$ in order to perform the inversion. As described in Section 4, this can be done irrespective of the inversion algorithm employed. The s-values are then placed in a global work-queue to which the groups of slave processors make requests.

The highest ranking processor in a group of slaves makes a request to the master for an s-value and is assigned the next one available. This is then broadcast to the other members of the slave group to allow them to construct their columns of the matrix U' for that specific s. Each processor reads in the columns of the matrix U' that correspond to its allocated partition into two types of sparse matrix data structure and also reads in the initial source-state weighting vector $\tilde{\alpha}$. *Local* non-zero elements (i.e. those elements in diagonal matrix blocks that will be multiplied with vector elements stored locally) are stored in a conventional compressed sparse column format. *Remote* non-zero elements (i.e. those elements in off-diagonal matrix blocks that must be multiplied with vector elements received from other processors) are stored in an ultrasparse matrix data structure – one for each remote processor – using a coordinate format. Each processor then determines which vector elements need to be received from and sent to every other processor in the group on each iteration, adjusting the row indices in the ultrasparse matrices so that they index into a vector of received elements. This ensures that a minimum amount of communication takes place and makes multiplication of off-diagonal blocks with received vector elements efficient.

For each step in our iterative algorithm, each processor begins by using non-blocking communication primitives to send and receive remote vector elements, while calculating the product of local matrix elements with locally stored vector elements. The use of non-blocking operations allows computation and communication to proceed concurrently on parallel machines where dedicated network hardware supports this effectively. The processor then waits for the completion of non-blocking operations (if they have not already completed) before multiplying received remote vector elements with the relevant ultrasparse matrices and adding their contributions to the local vector-matrix product cumulatively.

Once the calculations of a slave group are deemed to have converged, the result is returned to the master by the highest-ranking processor in the group and cached. When all results have been computed and returned for all required values of s, the final Laplace inversion calculations are made by the master, resulting in the required t-points.

7. Numerical Results

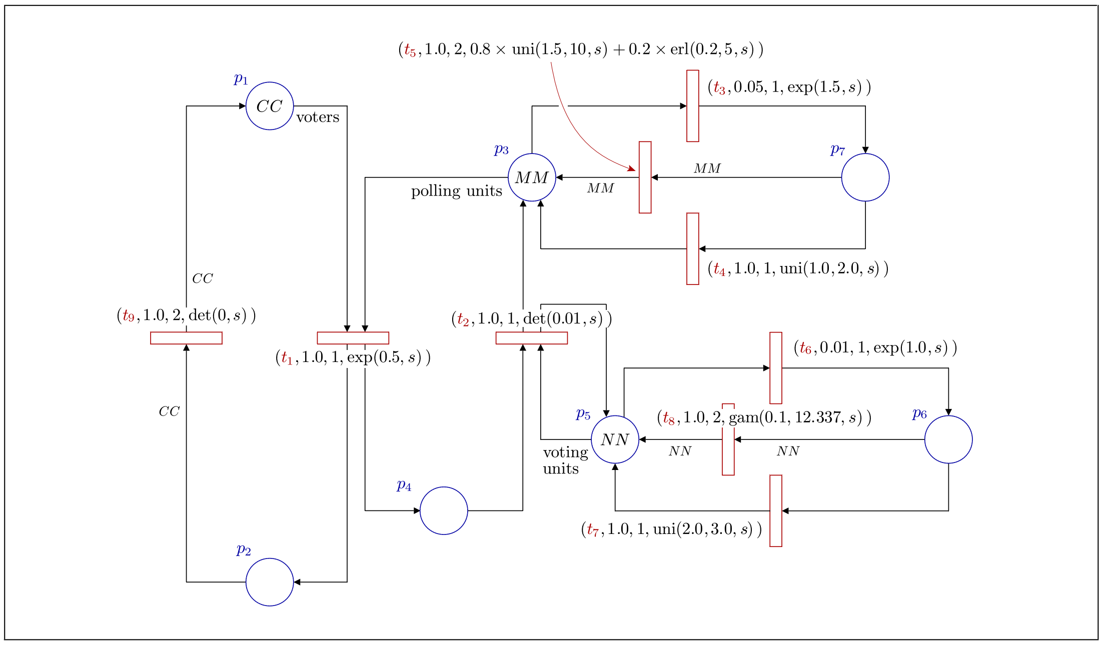
(t_5, 1.0, 2, 0.8 × uni(1.5, 10, s) + 0.2 × erl(0.2, 5, s))
p_1
CC
voters
(t_3, 0.05, 1, exp(1.5, s))
p_3
p_7
MM
MM
MM
polling units
(t_4, 1.0, 1, uni(1.0, 2.0, s))
CC
(t_9, 1.0, 2, det(0, s))
(t_2, 1.0, 1, det(0.01, s))
(t_1, 1.0, 1, exp(0.5, s))
(t_6, 0.01, 1, exp(1.0, s))
p_5
(t_8, 1.0, 2, gam(0.1, 12.337, s))
p_6
CC
NN
NN
NN
voting units
p_4
(t_7, 1.0, 1, uni(2.0, 3.0, s))
p_2

Fig. 3. A semi-Markov stochastic Petri net of a voting system with breakdowns and repairs

The results presented in this section were produced on a Beowulf Linux cluster with 64 dual processor nodes. Each node has two Intel Xeon 2.0GHz processors and 2GB of RAM. The

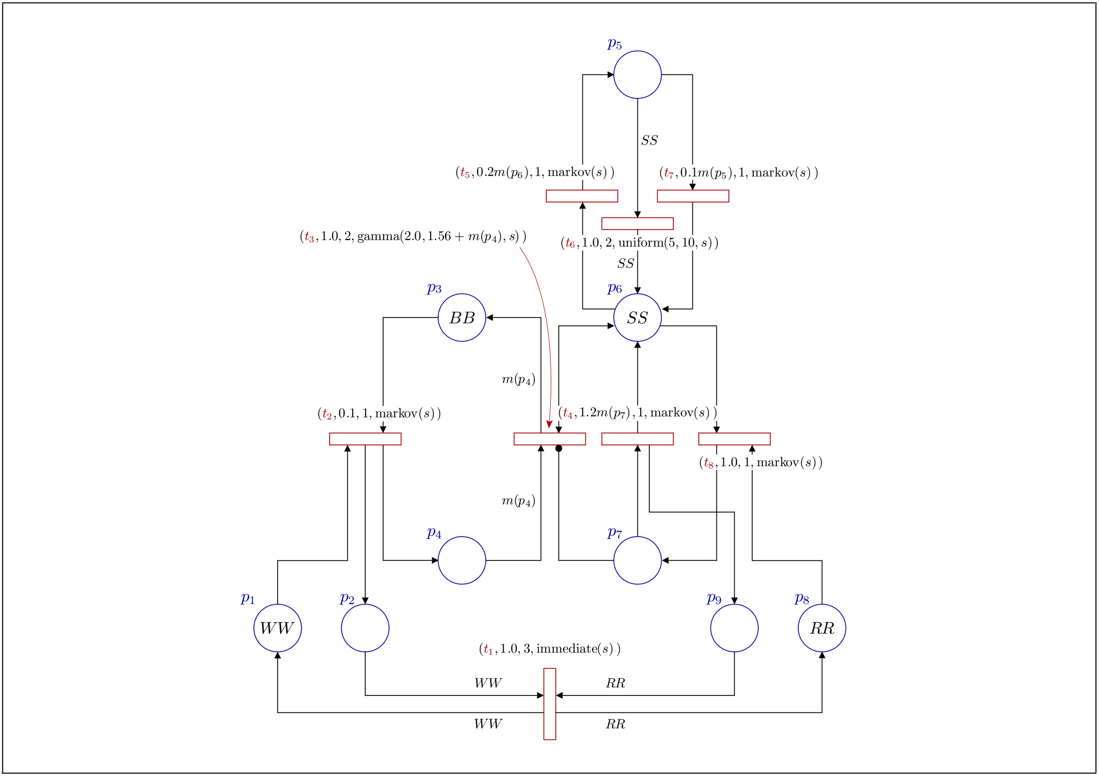
p_5
SS
$(t_5, 0.2m(p_6), 1, \text{markov}(s))$
$(t_7, 0.1m(p_5), 1, \text{markov}(s))$
$(t_3, 1.0, 2, \text{gamma}(2.0, 1.56 + m(p_4), s))$
$(t_6, 1.0, 2, \text{uniform}(5, 10, s))$
SS
p_3
BB
p_6
SS
$m(p_4)$
$(t_2, 0.1, 1, \text{markov}(s))$
$(t_4, 1.2m(p_7), 1, \text{markov}(s))$
$(t_8, 1.0, 1, \text{markov}(s))$
$m(p_4)$
p_4
p_7
p_1
WW
p_2
p_9
p_8
RR
$(t_1, 1.0, 3, \text{immediate}(s))$
WW
RR
WW
RR

Fig. 4. A semi-Markov Petri Net of a parallel web server

nodes are connected by a Myrinet network with a peak throughput of 250 Mb/s.

We demonstrate the SMP passage time analysis techniques of the previous sections with large semi-Markov models of a distributed voting system (Fig. 3) and a distributed web server (Fig. 4). The models are specified in a semi-Markov stochastic Petri net formalism (see [7] for full SM-SPN semantics) using an extension of the DNAmaca Markov chain modelling language [18]. Generally distributed transitions, if simultaneously enabled, are selected by probabilistic choice; in this way, we are guaranteed an underlying semi-Markov state space. This probabilistic selection can be seen as an approximation of the more expressive concurrency provided by Generalised Semi-Markov Process (GSMP) models. However, semi-Markov processes are an accurate model in many useful cases; for example when modelling mutual exclusion or probabilistic task switching on a uniprocessor computer system. Concurrent Markovian execution is, however, fully supported (and used in Fig. 4). Transition firing time densities are specified in terms of their Laplace transforms, with macros provided for common distributions (e.g. uniform, gamma, deterministic), and can be made marking dependent by use of the $m(p_i)$ function (which returns the current number of tokens on place p_i). Support for inhibitor arcs is also provided.

System	CC	MM	NN	States
1	60	25	4	106 540
2	100	30	4	249 760
3	125	40	4	541 280
4	150	40	5	778 850
5	175	45	5	1 140 050
6	300	80	10	10 999 140

Tab. I. Configurations of the voting system as used to present convergence results and passage times

System	RR	WW	SS	BB	States
1	45	22	4	8	107 289
2	52	26	5	10	248 585
3	60	30	6	12	517 453
4	65	30	7	13	763 680
5	70	35	7	14	1 044 540
6	100	50	18	20	15 445 919

Tab. II. Configurations of the web server system as used to present convergence results and passage times

Fig. 3 represents a voting system with CC voters, MM polling units and NN central voting servers. In this system, voters cast votes through polling units which in turn register votes with all available central voting units. Both polling units and central voting units can suffer breakdowns, from which there is a soft recovery mechanism. If, however, all the polling or voting units fail, then, with high priority, a failure recovery mode is instituted to restore the system to an operational state.

Fig. 4 represents a web server with RR clients, WW web content authors, SS parallel web servers and a write-buffer of BB in size. Clients can make read requests to one of the web servers for content (represented by the movement of tokens from p_8 to p_7). Web content authors submit page updates into the write buffer (represented by the movement of tokens from p_1 onto p_2 and p_4), and whenever there are no outstanding read requests all outstanding write requests in the buffer (represented by tokens on p_4) are applied to all functioning web servers (represented by tokens on p_6). Web servers can fail (represented by the movement of tokens from p_6 to p_5) and institute self-recovery unless all servers fail, in which case a high-priority recovery mode is initiated to restore all servers to a fully functional state. Complete reads and updates are represented by tokens on p_9 and p_2 respectively.

For the voting system, Tab. I shows how the size of the underlying SMP varies according to the configuration of the variables CC, MM, and NN. Similarly, for the web server model, Tab. II shows state space sizes in relation to the configuration of the variables RR, WW, SS and BB.

In the following, we consider the rate of convergence of the iterative passage time algorithm and the extraction of passage time densities and cumulative distribution functions for the example semi-Markov systems.

7.1. Convergence of iterative passage time algorithm

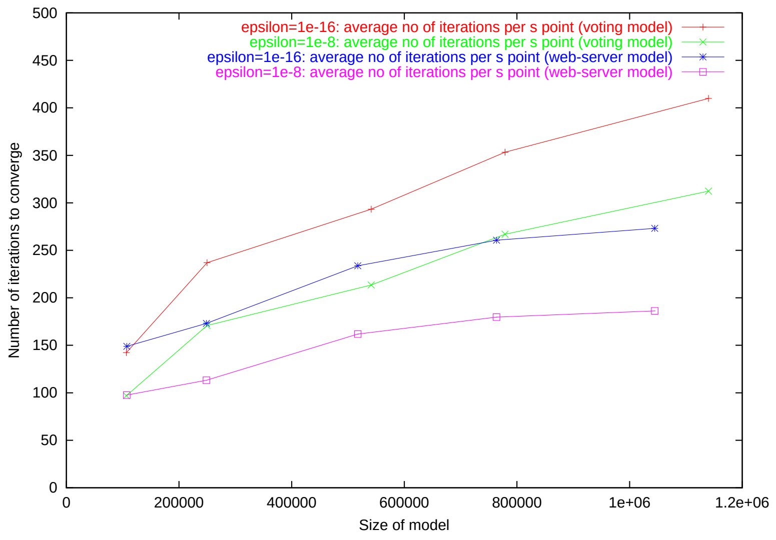

Fig. 5. Average number of iterations to converge per s point for two different values of ϵ over a range of model sizes.

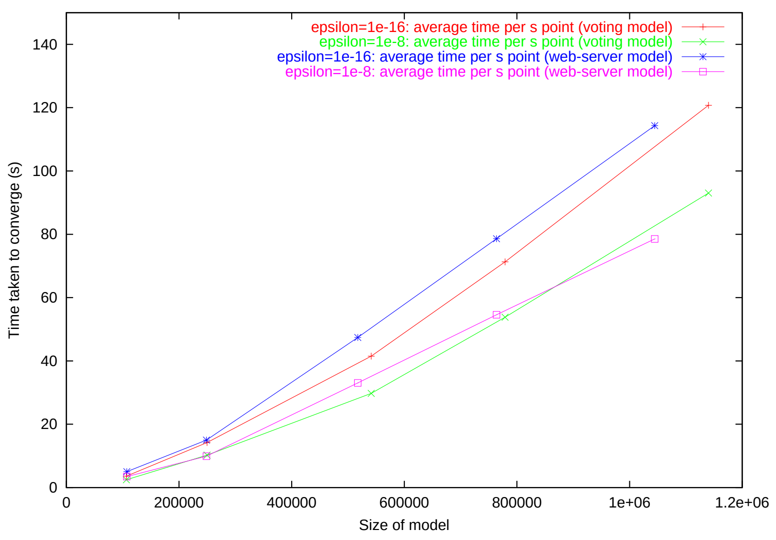

Fig. 6. Average time to convergence per s point for two different values of ϵ over a range of model sizes.

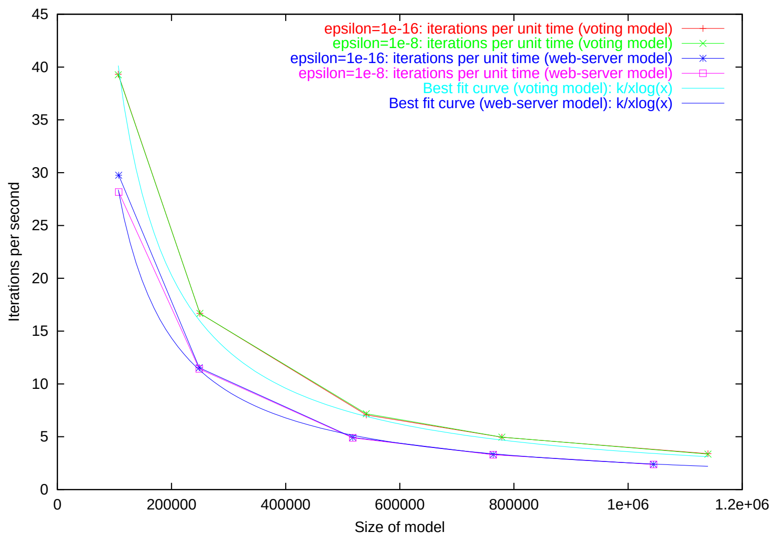

Fig. 7. Average number of iterations per unit time over a range of model sizes.

Our iterative algorithm terminates when two successive iterates are less than ϵ apart (cf. Eq. (24)), for some suitably small value of ϵ. Fig. 5 shows the average number of iterations the algorithm takes to converge per s-point for both models for two different values of ϵ (10^{-8} and 10^{-16}).

It is noted that the number of iterations required for convergence as the model size grows is sub-linear; that is, as the model size doubles the number of iterations less than doubles. This suggests the algorithm has good scalability properties. Fig. 6 shows the average amount of time to convergence per s-point, while Fig. 7 shows how the number of iterations per unit time decreases as model size increases. The curves are almost identical for both values of ϵ, suggesting that the time spent per iteration remains constant, irrespective of the number of iterations performed. The rate of computation (iterations per unit time) is $O(1/(N\log(N)))$ for system size N. This gives a time per iteration of $O(N\log(N))$, suggesting an overall practical complexity of better than $O(N^2\log(N))$ (given the better than $O(N)$ result for the number of iterations required).

7.2. Passage time densities and quantiles

In this section, we display passage time densities produced by our iterative passage time algorithm and validate these results by simulation.

Fig. 8 shows the density of the time taken to process 300 voters (as given by the passage of 300 tokens from place p_1 to p_2) in system 6 of the voting model. Calculation of the analytical density required 15 hours and 7 minutes using 64 slave processors (in 8 groups of 8) for the 31 t-points plotted. Our algorithm evaluated $L_{\vec{i}\vec{j}}(s)$ at 1 023 s-points, each of which involved

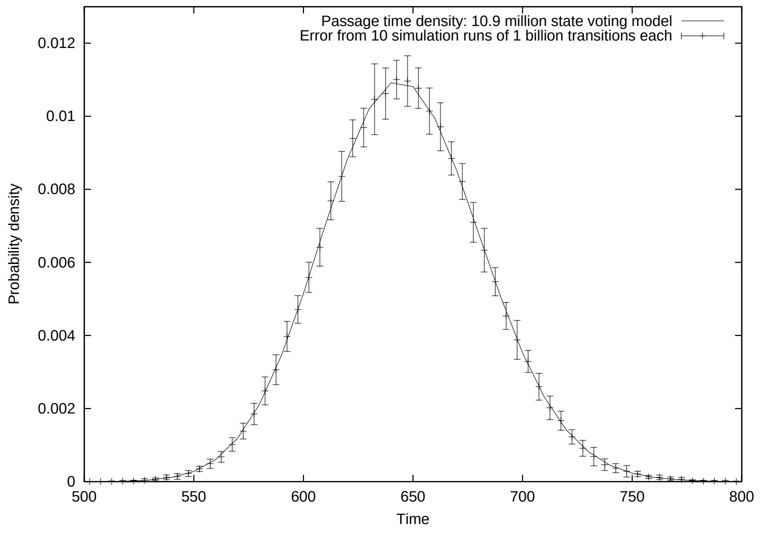

Fig. 8. Analytic and simulated (with 95% confidence intervals) density for the time taken to process 300 voters in the voting model system 6 (10.9 million states).

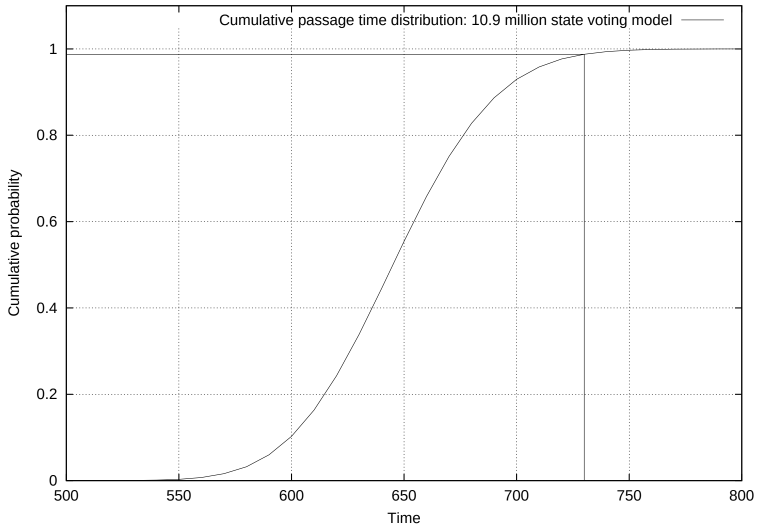

Fig. 9. Cumulative distribution function and quantile of the time taken to process 300 voters in the voting model system 6 (10.9 million states).

manipulating sparse matrices of rank 10 999 140. The analytical curve is validated against the combined results from 10 simulations, each of which consisted of 1 billion transition firings. Despite this large simulation effort, we still observe wide confidence intervals (probably because of the rarity of source states).

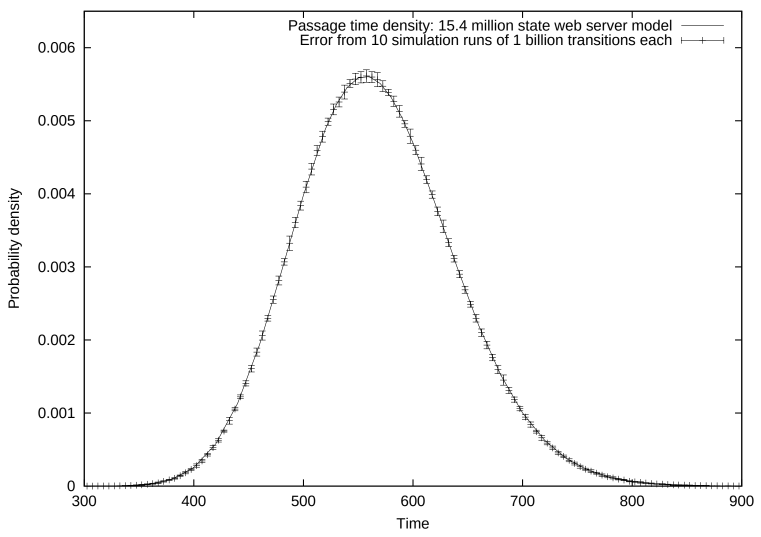

Fig. 10. Analytic and simulated (with 95% confidence intervals) density for the time taken to process 100 reads and 50 page updates in the web-server model system 6 (15.4 million states).

Fig. 9 is a cumulative distribution for the same passage as Fig. 8 (easily obtained by inverting $L_{\vec{i}\vec{j}}(s)/s$ from cached values of $L_{\vec{i}\vec{j}}(s)$). It allows us to extract reliability quantiles, for instance:

$$\mathbb{P}(\text{system 6 can process 300 voters in less than 730 seconds}) = 0.9876$$

Fig. 10 shows the density of the time taken to perform 100 reads and 50 page updates in the web server model 6. Calculation of the 35 t-points plotted required 2 days, 17 hours and 30 minutes using 64 slave processors (in 8 groups of 8). Our algorithm evaluated $L_{\vec{i}\vec{j}}(s)$ at 1 155 s-points, each of which involved manipulating sparse matrices of rank 15 445 919. Again, the analytical result is validated against the combined results from 10 simulations, each of which consisted of 1 billion transition firings. We observe excellent agreement.

7.3. Scalability Results

Tab. III and Fig. 11 display the performance of the hypergraph-partitioned sparse matrix-vector multiplication operations. They show good scalability with a linear speedup trend, which is unusual in problems of this nature. This is because the hypergraph partitioning minimises the amount of data which must be exchanged between processors. The efficiency is not 100% in all cases, however, as even this reduced amount of inter-node communication imposes an overhead and computational load is not perfectly balanced.

Tab. IV and Fig. 12 show the hypergraph scalability in the case where 32 slave processors were divided into various size sub-clusters (32 groups of 1, 16 groups of 2, 8 groups of 4, and so on). This was to measure the benefit to be gained from adding extra groups to draw s-points from the global work queue versus doing the computation across larger groups of slave processors

Processors	Time (s)	Speedup	Efficiency
1	3968.07	1.00	1.00
2	2199.98	1.80	0.902
4	1122.97	3.53	0.883
8	594.07	6.68	0.835
16	320.19	12.39	0.775
32	188.14	21.09	0.659

Tab. III. Speedup and efficiency of doing hypergraph-partitioned sparse matrix-vector multiplication across 1 to 32 processors. Calculated for the 249 760 state voting model for 165 s-points.

Processors	Time (s)	Speedup	Efficiency
32×1	150.13	26.43	0.830
16×2	159.55	24.87	0.777
8×4	162.13	24.47	0.765
4×8	165.24	24.01	0.750
2×16	173.76	22.84	0.714
1×32	188.14	21.09	0.659

Tab. IV. Speedup and efficiency using 32 slave processors divided into various different size sub-clusters.

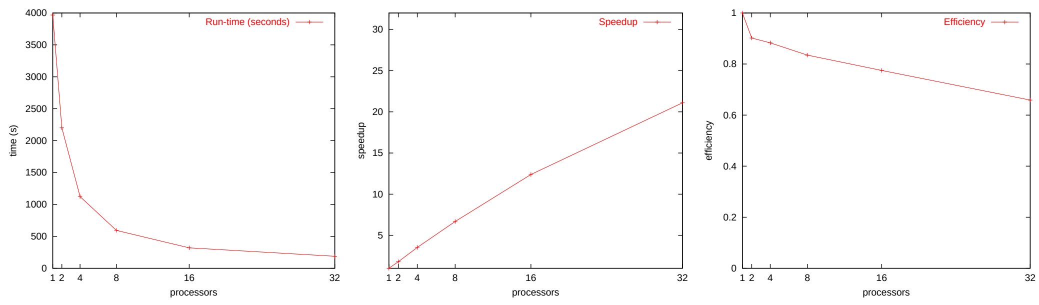

Fig. 11. Runtime, speedup and efficiency using hypergraph partitioning.

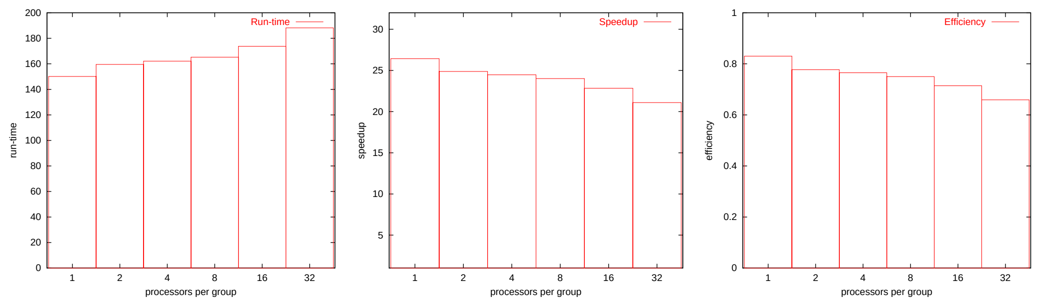

Fig. 12. Runtime, speedup and efficiency for using 32 processors in groups of varying sizes.

(which may be necessary when the state space of the model under analysis is very large). The efficiency decreases as the number of groups decreases. Note, however, that with runtimes of between 150 and 188 seconds, there is still a dramatic improvement over the runtime on a single slave processor (3 968 seconds) regardless of the group size employed. These results suggest that, given a fixed number of slave processors, it is best to allocate them into the smallest size subgroups (that is, to maximise the number of groups drawing from the global work-queue) subject to the constraints imposed by the size of the model and the memory available on each processor.

8. Conclusion

In this paper, we have derived passage time densities and quantiles from semi-Markov models with over 15 million states. To achieve this, we have developed a parallel iterative algorithm for computing passage time densities. The key to its scalability is the hypergraph data partitioning used to minimise communication and balance load. We represent the general distributions found in SMPs efficiently by using a constant-space representation based on the evaluation demands of a numerical Laplace transform inverter.

We have analysed the truncation error of our algorithm and derived an optimised convergence test that may be applied provided that the dominant eigenvalue of the system can be found. Finally, we have demonstrated the applicability, scalability and capacity of the method on several semi-Markov systems derived from stochastic Petri nets, and have observed practical complexity of better than $O(N^2 \log N)$ processing times for systems with N states.

As future work, we are planning to extend the iterative scheme to generate transient distributions while maintaining a similar complexity profile.

Acknowledgements: The authors would like to thank the Imperial College Parallel Computing Centre for the use of the Viking Beowulf Cluster. We are particularly grateful to Steve Newhouse and Keith Sephton for their help and technical advice.

REFERENCES

1. J. Abate, G.L. Choudhury, and W. Whitt. On the Laguerre method for numerically inverting Laplace transforms. *INFORMS Journal on Computing*, 8(4):413–427, 1996.
2. J. Abate and W. Whitt. The Fourier-series method for inverting transforms of probability distributions. *Queueing Systems*, 10(1):5–88, 1992.
3. J. Abate and W. Whitt. Numerical inversion of Laplace transforms of probability distributions. *ORSA Journal on Computing*, 7(1):36–43, 1995.
4. G. Bolch, S. Greiner, H. de Meer, and K.S. Trivedi. *Queueing Networks and Markov Chains.* Wiley, August 1998.
5. J.T. Bradley and N.J. Davies. A matrix-based method for analysing stochastic process algebras. In *ICALP Workshops 2000, Process Algebra and Performance Modelling Workshop*, pages 579–590. University of Waterloo, Ontario, Canada, Carleton Scientific, Geneva, July 2000.

6. J.T. Bradley, N.J. Dingle, P.G. Harrison, and W.J. Knottenbelt. Distributed computation of passage time quantiles and transient state distributions in large semi-Markov models. In *Performance Modelling, Evaluation and Optimization of Parallel and Distributed Systems (PMEO 2003)*, Nice, April 2003.
7. J.T. Bradley, N.J. Dingle, W.J. Knottenbelt, and P.G. Harrison. Performance queries on semi-Markov stochastic Petri nets with an extended Continuous Stochastic Logic. In *PNPM'03, Proceedings of Petri Nets and Performance Models*, 2003. To appear.
8. U.V. Catalyürek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Transactions on Parallel and Distributed Systems*, 10(7):673–693, July 1999.
9. L. de Alfaro and S. Gilmore, editors. *PAPM-PROBMIV 2001, Proceedings of Process Algebra and Probabilistic Methods*, volume 2165 of *Lecture Notes in Computer Science*. Springer-Verlag, Aachen, September 2001.
10. D.D. Deavours and W.H. Sanders. An efficient disk-based tool for solving very large Markov models. In *TOOLS 1997, Computer Performance Evaluation: Modelling Techniques and Tools*, volume 1245 of *Lecture Notes in Computer Science*, pages 58–71, St. Malo, June 1997. Springer-Verlag.
11. N.J. Dingle, P.G. Harrison, and W.J. Knottenbelt. Response time densities in Generalised Stochastic Petri Net models. In *Proceedings of the 3rd International Workshop on Software and Performance (WOSP 2002)*, pages 46–54, Rome, July 24th–26th 2002.
12. R. German, D. Logothetis, and K.S. Trivedi. Transient analysis of Markov regenerative stochastic Petri nets: A comparison of approaches. In *PNPM'95, Proceedings of Petri Nets and Performance Models*, pages 103–112, Durham, North Carolina, 1995.
13. W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. MIT Press, Cambridge, Massachussetts, 1994.
14. P.G. Harrison and W.J. Knottenbelt. Passage-time distributions in large Markov chains. In *Proceedings of ACM SIGMETRICS 2002*, pages 77–85, Marina Del Rey, USA, June 2002.
15. G.G. Infante López, H. Hermanns, and J.-P. Katoen. Beyond memoryless distributions: Model checking semi-Markov chains. In de Alfaro and Gilmore [9], pages 57–70.
16. G. Karypis and V. Kumar. Multilevel k-way hypergraph parititioning. Technical Report #98-036, University of Minnesota, 1998.
17. J.-P. Katoen, M. Kwiatkowska, G. Norman, and D. Parker. Faster and symbolic CTMC model checking. In de Alfaro and Gilmore [9], pages 23–38.
18. W.J. Knottenbelt. Generalised Markovian analysis of timed transitions systems. MSc thesis, University of Cape Town, South Africa, July 1996.
19. W.J. Knottenbelt. *Parallel Performance Analysis of Large Markov Models*. PhD thesis, Imperial College, London, United Kingdom, February 2000.
20. W.J. Knottenbelt and P.G. Harrison. Distributed disk-based solution techniques for large Markov models. In *NSMC'99, Proceedings of the 3rd International Meeting on the Numerical Solution of Markov Chains*, pages 58–75, Zaragoza, September 1999.
21. W.J. Knottenbelt, P.G. Harrison, M.A. Mestern, and P.S. Kritzinger. A probabilistic dynamic technique for the distributed generation of very large state spaces. *Performance Evaluation*, 39(1–4):127–148, February 2000.
22. B. Melamed and M. Yadin. Randomization procedures in the computation of cumulative-time distributions over discrete state Markov processes. *Operations Research*, 32(4):926–944, July–August 1984.
23. J.K. Muppala and K.S. Trivedi. Numerical transient analysis of finite Markovian queueing systems. In *Queueing and Related Models*, pages 262–284. Oxford University Press, 1992.
24. R. Pyke. Markov renewal processes: Definitions and preliminary properties. *Annals of Mathematical Statistics*, 32(4):1231–1242, December 1961.