Performance Trees: Implementation and Distributed Evaluation

Darren K. Brien[1], Nicholas J. Dingle[2], William J. Knottenbelt[3], Harini Kulatunga[4], Tamas Suto[5]

Department of Computing, Imperial College London
180 Queen's Gate, London SW7 2AZ

Abstract

In this paper, we describe the first realisation of an evaluation environment for Performance Trees, a recently proposed formalism for the specification of performance properties and measures. In particular, we present details of the architecture and implementation of this environment that comprises a client-side model and performance query specification tool, and a server-side distributed evaluation engine, supported by a dedicated computing cluster. The evaluation engine combines the analytic capabilities of a number of distributed tools for steady-state, passage time and transient analysis, and also incorporates a caching mechanism to avoid redundant calculations. We demonstrate in the context of a case study how this analysis pipeline allows remote users to design their models and performance queries in a sophisticated yet easy-to-use framework, and subsequently evaluate them by harnessing the computing power of a Grid cluster back-end.

Keywords: Performance Trees, Quantitative Performance Analysis, Parallel and Distributed Computing, Grid Computing

1 Introduction

In today's fast-paced world great importance is attached to performance. To help engineers design efficient systems, it is essential to provide the means to (a) quantify performance measures (e.g. *"In a hospital waiting room, what is the steady-state distribution of the number of patients waiting to be treated?"*) and (b) verify conformance to Quality of Service (QoS) requirements (e.g. *"In a mobile communications network, is the time taken to send an SMS message between two handsets less than 5 seconds with more than 95% probability?"*).

[1] Email: dkb03@doc.ic.ac.uk
[2] Email: njd200@doc.ic.ac.uk
[3] Email: wjk@doc.ic.ac.uk
[4] Email: hkulatun@doc.ic.ac.uk
[5] Email: suto@doc.ic.ac.uk

In [1] and [2] we proposed Performance Trees as a unifying framework for the quantification and verification of performance properties. The concepts expressible in Performance Tree queries are intended to be familiar to engineers and include steady-state and passage time distributions and densities, their moments, transition firing rates, convolutions and arithmetic operations. An important concern during the development of Performance Trees was ease of use, resulting in a formalism that can be straightforwardly visualised and manipulated as hierarchical tree structures.

In this paper, we describe major extensions to the open-source PIPE Petri net editor which establish the first tool support for Performance Trees. Specifically, we describe the architecture and implementation of a user-facing front-end module for performance query design and a Grid cluster-based back-end for distributed performance query evaluation. Together with the original model specification capabilities of PIPE, the result is a comprehensive system modelling, performance query design and performance query evaluation pipeline.

The structure of the remainder of this paper is as follows. Section 1.1 provides a brief overview of Performance Trees. Sections 1.2 to 1.5 summarise some fundamental analysis techniques underlying the evaluation of Performance Tree operators in the context of Generalised Stochastic Petri Nets. Section 2 describes the architecture of the Performance Tree Evaluation Engine – the computational back-end of the evaluation architecture. Section 3 gives details of the techniques used for the distributed evaluation of Performance Tree queries. Section 4 demonstrates the application and evaluation of Performance Trees on a case study of a hospital's Accident and Emergency unit. Section 5 concludes with a summary of the paper and a discussion of future work.

1.1 Performance Trees

As mentioned above, Performance Trees are a recently proposed formalism for the representation of performance-related queries. They combine the ability to specify performance requirements – i.e. queries aiming to determine whether particular properties hold on system models – and to extract performance measures – i.e. quantifiable performance metrics of interest.

A Performance Tree query is represented as a tree structure, consisting of nodes and interconnecting arcs. Nodes can have two kinds of roles within queries: *operation* nodes represent performance-related concepts, such as the calculation of a passage time density for instance, while *value* nodes represent the inputs to these operations. Value nodes identify attributes such as a set of states, a function on a set of states, an action, or simply numerical or boolean constants. Operation nodes can be interpreted like functions in a programming language, which perform some operation on the supplied inputs in order to obtain a result that is provided as output. In this way, complex queries can be easily constructed from basic concepts by connecting nodes together. Table 1 provides an overview of the currently available operation nodes.

Performance Trees are an extensible formalism in the sense that every operation node encapsulates a single self-contained concept, and hence new nodes can be pro-

Textual	Graphical	Description
?		The overall result of a performance query.
;		Represents a vector of results of independent queries that are joined together.
PTD		Represents a passage time density, calculated from a given set of start and target states, as well as optional additional constraints on excluded states.
Dist		Represents a passage time distribution that is obtained from a passage time density.
Conv		Represents a convolution of two passage time densities.
ProbInInterval		Represents the probability with which a passage takes place in a certain amount of time.
ProbInStates		Represents the transient probability of the system being in a given set of states at a given instant in time.
Moment		Represents a raw moment of a passage time density or distribution.
FR		Represents the mean occurrence of an action / firing rate of a transition.
SS:P		Represents the steady-state probability distribution for a given set of states.
SS:S		Represents a set of states that have a certain steady-state probability.
StatesAtTime		Represents the set of states that the system can occupy at a given time.
InInterval		A boolean operator that determines whether a numerical value is within an interval or possibly within multiple intervals.
$\subseteq$		A boolean operator that determines whether a set is included in or corresponds to another set.
$\vee$, $\wedge$		Represent a boolean disjunction or conjunction of two logical expressions.
$\neg$		Represents boolean negation of a logical expression.
$>$, $\geq$, $==$, $\leq$, $<$		Represent arithmetic comparisons of two numerical values.
$+$, $-$, $*$, $/$, $\hat{}$		Represent arithmetic operations on two numerical values.

Table 1
Description of Performance Tree operation nodes

gressively added as long as evaluation support for the new operations is integrated into the analysis engine at the same time. The formalism also supports macros, which allow new concepts to be created with the use of existing operators.

Performance Trees can be used with many different modelling formalisms, enabled by an abstract state specification mechanism, details of which can be found in [2]. This paper will consider Generalised Stochastic Petri nets (GSPNs) as the modelling formalism and will present a GSPN-based application case study.

1.2 Generalised Stochastic Petri Nets

Petri nets were originally devised as a graphical formalism for describing concurrency and synchronisation in distributed systems. In their simplest form (not containing any timing information) they are also known as Place-Transition nets [3].

Definition 1.1 *A Place-Transition net is a 5-tuple* $PN = (P, T, I^-, I^+, M_0)$ *where:*

- $P = \{p_1, ..., p_n\}$ *is a finite and non-empty set of places.*
- $T = \{t_1, ..., t_m\}$ *is a finite and non-empty set of transitions.*
- $P \cap T = \emptyset$
- $I^-, I^+ : P \times T \to \mathbb{N}_0$ *are the backward and forward incidence functions, respectively.*
- $M_0 : P \to \mathbb{N}_0$ *is the initial marking.*

A marking (or state) is a vector of integers representing the number of tokens on each place of the model. A transition can fire if the input places of the transition contain at least the number of tokens specified by the backward incidence matrix. In so firing, a number of tokens are removed from the transition's input places and a number of tokens added to the transition's output places according to the backward and forward incidence matrices respectively.

Generalised Stochastic Petri Nets (GSPNs) extend Place-Transition nets by incorporating timing information [3, 4].

Definition 1.2 *A GSPN is a 4-tuple* $GSPN = (PN, T_1, T_2, W)$ *where*

- $PN = (P, T, I^-, I^+ M_0)$ *is the underlying Place-Transition net.*
- $T_1 \subseteq T$ *is the set of timed transitions.*
- $T_2 \subset T$ *is the set of immediate transitions, where* $T_1 \cap T_2 = \emptyset$ *and* $T = T_1 \cup T_2$.
- $W = \big(w_1, ..., w_{|T|}\big)$ *is an array whose entry* $w_i \in \mathbb{R}^+$ *is a (possibly marking dependent)*
  - *rate of a negative exponential distribution specifying the firing delay, when transition* t_i *is a timed transition, or*
  - *firing weight, when transition* t_i *is an immediate transition.*

Timed transitions have an exponentially distributed firing rate λ_i. Immediate transitions fire in zero time. Markings that only enable timed transitions are known as *tangible*, while markings that enable both timed and immediate transitions are called *vanishing*. We denote the set of tangible markings $\mathcal{T}$ and the set of vanishing markings $\mathcal{V}$. The sojourn time in a tangible marking M_i is exponentially distributed with parameter $\mu_i = \sum_{k \in en(M_i)} \lambda_k$ where $en(M_i)$ is the set of transitions enabled by marking M_i. The sojourn time in vanishing markings is zero.

1.3 Steady-State Calculations for GSPNs

The stochastic process described by a GSPN's reachability graph is a continuous-time Markov chain (CTMC) if $\mathcal{V} = \emptyset$ and semi-Markovian otherwise. It is possible, however, to reduce the reachability graph of a GSPN where $\mathcal{V} \neq \emptyset$ to a CTMC using vanishing-state elimination techniques [5, 6].

An homogeneous N-state $\{1, 2, \ldots, N\}$ CTMC has state at time t denoted $\chi(t)$. Its evolution is described by an $N \times N$ generator matrix $\mathbf{Q}$, where q_{ij} is the infinitesimal

rate of moving from state i to state j ($i \neq j$), and $q_{ii} = -\sum_{i \neq j} q_{ij}$.

Where it exists, the steady-state distribution of CTMC, $\{\pi_j\}$, is given by [3]:

$$\pi_j = \lim_{t \to \infty} \mathbb{P}(\chi(t) = j \mid \chi(0) = i)$$

For an finite, irreducible and homogeneous CTMC, the steady-state probabilities $\{\pi_j\}$ always exist and are independent of the initial state distribution. They are uniquely given by the solution of the equations:

$$-q_{jj}\pi_j + \sum_{k \neq j} q_{kj}\pi_k = 0 \quad \text{subject to} \quad \sum_i \pi_i = 1$$

This can be expressed in matrix vector form (in terms of the vector $\boldsymbol{\pi}$ with elements $\{\pi_1, \pi_2, \ldots, \pi_N\}$ and the matrix $\mathbf{Q}$ defined above) as $\boldsymbol{\pi}\mathbf{Q} = \mathbf{0}$. We define p_{ij} to be the probability that j is the next state to be entered after state i.

1.4 Passage Time Density Calculations for GSPNs

The Laplace transform is an integral transform which is widely used in the solution of problems which are hard to solve in (real-valued) t-space. It transforms such problems into (complex-valued) s-space where they can be solved more easily; this solution is then inverted to bring it back into t-space. We adopt this approach for the passage time analysis of GSPNs [7,8].

For a GSPN where $\mathcal{V} \neq \emptyset$, we define the passage time from a single source marking i to a non-empty set of target markings $\vec{j}$:

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

where $M_{i\vec{j}} = \min\{m \in \mathbb{Z}^+ : \chi_m \in \vec{j} \mid \chi_0 = i\}$; here χ_i is the state of the system after the ith transition firing [9].

To find this passage time we must convolve the state sojourn time densities for all paths from i to $j \in \vec{j}$. We exploit the convolution property of the Laplace transform which states that the convolution of two functions is equal to the product of their Laplace transforms. We perform a first-step analysis to find the Laplace transform of the relevant density; that is, we first find the probability density of moving from state i to its set of direct successor states $\vec{k}$ and then convolve it with the probability density of moving from $\vec{k}$ to the set of target states $\vec{j}$. Vanishing markings have a sojourn time density of 0, with probability 1, which results in their Laplace transform equalling 1 for all values of s. If $\mathcal{L}_{i\vec{j}}(s)$ is the Laplace transform of the density function $f_{i\vec{j}}(t)$ of the passage time variable $T_{i\vec{j}}$, then we can express this Laplace transform as a system of linear equations given by:

$$\mathcal{L}_{i\vec{j}}(s) = \begin{cases} \displaystyle\sum_{k \notin \vec{j}} \left(\frac{q_{ik}}{s - q_{ii}}\right) \mathcal{L}_{k\vec{j}}(s) + \sum_{k \in \vec{j}} \left(\frac{q_{ik}}{s - q_{ii}}\right) & \text{if } i \in \mathcal{T} \\ \sum_{k \notin \vec{j}} p_{ik}\mathcal{L}_{k\vec{j}}(s) + \sum_{k \in \vec{j}} p_{ik} & \text{if } i \in \mathcal{V} \end{cases} \tag{1}$$

If we wish to calculate the passage time from multiple source states, denoted by the vector $\vec{i}$, the Laplace transform of the passage time density is given by:

$$\mathcal{L}_{\vec{i}\vec{j}}(s) = \sum_{k \in \vec{i}} \alpha_k \mathcal{L}_{k\vec{j}}(s)$$

where α_k is the steady-state probability that the SMP is in state k at the starting instant of the passage.

Now that we have the Laplace transform of the passage time, we must invert it to get the distribution in the real domain. To do this we use the Laguerre method, which makes use of the Laguerre series representation of $f(t)$ [10]:

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

where the Laguerre polynomials l_n are given by:

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

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

$$q_n = \frac{1}{2\pi r^n} \int_0^{2\pi} Q(re^{iu}) e^{-inu}\, du \tag{2}$$

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

The integral in the calculation of Eq. 2 can be approximated numerically using the trapezoidal rule, giving:

$$q_n \approx \frac{1}{2nr^n} \left(Q(r) + (-1)^n Q(-r) + 2 \sum_{j=1}^{n-1} (-1)^j \mathrm{Re}\left(Q(re^{\pi ji/n})\right) \right) \tag{3}$$

As described in [7], the Laguerre method can be modified by noting that the Laguerre coefficients q_n are independent of t. Since $|l_n(t)| \leq 1$ for all n, the convergence of the Laguerre series depends on the decay rate of q_n as $n \to \infty$ which is in turn determined by the smoothness of $f(t)$ and its derivatives [10]. Slow convergence of the q_n coefficients can often be improved by exponential dampening and scaling using two real parameters σ and b [11]. Suitable values for these parameters can be automatically determined using the algorithm described in [7].

Each q_n coefficient is computed as in Eq. 3, using the trapezoidal rule with $2n$ trapezoids. However, if we apply scaling to ensure that q_n has decayed to (almost) zero by term p_0 (say $p_0 = 200$), we can instead make use of a constant number of $2p_0$ trapezoids when calculating each q_n. This allows us to calculate each q_n with high accuracy while simultaneously providing the opportunity to cache and re-use values of $Q(z)$. Since q_n does not depend on t, and each evaluation of $Q(z)$ involves a single evaluation of $f^*(s)$, we obtain $f(t)$ at an arbitrary number of t-values at the fixed cost of evaluating $Q(z)$ (and hence $f^*(s)$) $2p_0$ times.

1.5 Moments Calculation for GSPNs

As well as calculating passage-time densities, we can also compute the moments of such densities. The n^{th} (raw) moment $M_{i\vec{j}}(n)$ of the passage time density from state i into a vector of target states $\vec{j}$ $\mathcal{L}_{i\vec{j}}(s)$ is obtained by differentiating $\mathcal{L}_{i\vec{j}}(s)$ n

times and evaluating the resulting expression at $s = 0$:

$$M_{i\vec{j}}(n) = (-1)^n \left. \frac{d^n L_{i\vec{j}}(s)}{ds^n} \right|_{s=0}$$

Moments in GSPNs can therefore be calculated by repeated differentiation of Eq. 1 and evaluation of the result at $s = 0$:

$$M_{i\vec{j}}(n) = \begin{cases} \sum_{k \notin \vec{j}} \frac{q_{ik}}{-q_{ii}} M_{k\vec{j}}(n) + \frac{1}{-q_{ii}} n M_{i\vec{j}}(n-1) & \text{if } i \in \mathcal{T} \\ \sum_{k \notin \vec{j}} p_{ik} M_{k\vec{j}}(n) & \text{if } i \in \mathcal{V} \end{cases}$$

2 Performance Query Evaluation Architecture

The architecture we have implemented for the evaluation of Performance Tree queries on GSPN models is shown in Fig. 1.

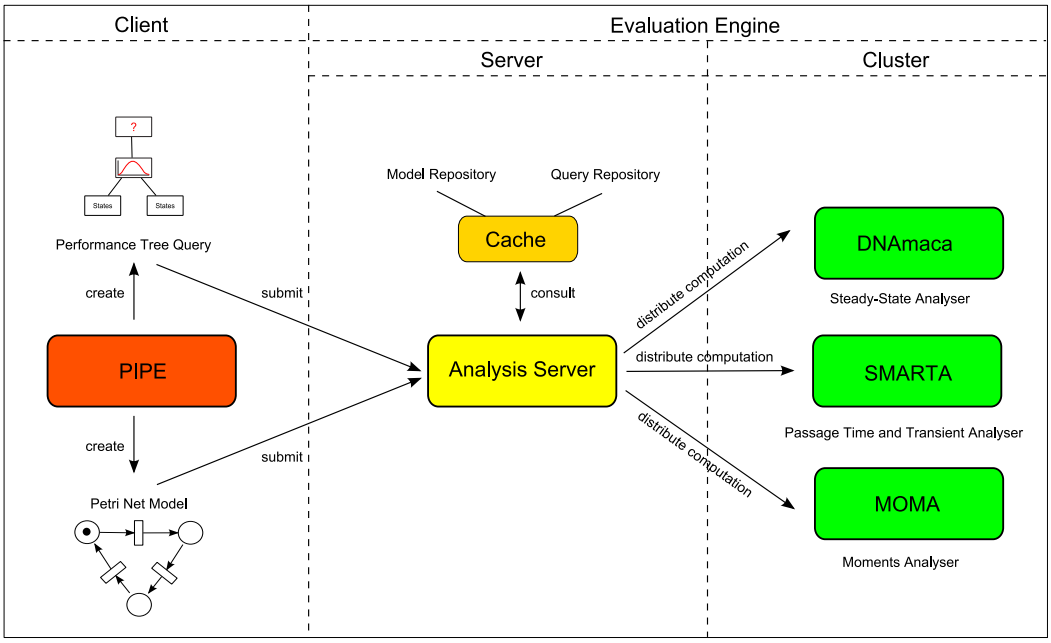


Fig. 1. Performance Query Evaluation Architecture

2.1 Client

The user-facing front-end is the open-source Platform Independent Petri net Editor (*PIPE*) [12]. Its basic functionality is the graphical design and animation of GSPN models. Beyond this, *PIPE* is equipped with pluggable analysis modules which perform tasks such as invariant analysis, steady state analysis, and so on.

PIPE provides support for Performance Trees through its Performance Query Editor module, which implements a graphical interface for the composition of Performance Tree queries. These are constructed on a canvas using a toolbar that enables the placing, connection and manipulation of operation and value nodes.

The creation process is aided by a natural language query translation mechanism, which continuously informs the user of the natural language equivalent of the current query. Figure 2 depicts a performance requirement verification query being built with the query designer interface. It also shows how a query is represented in natural language while being constructed.

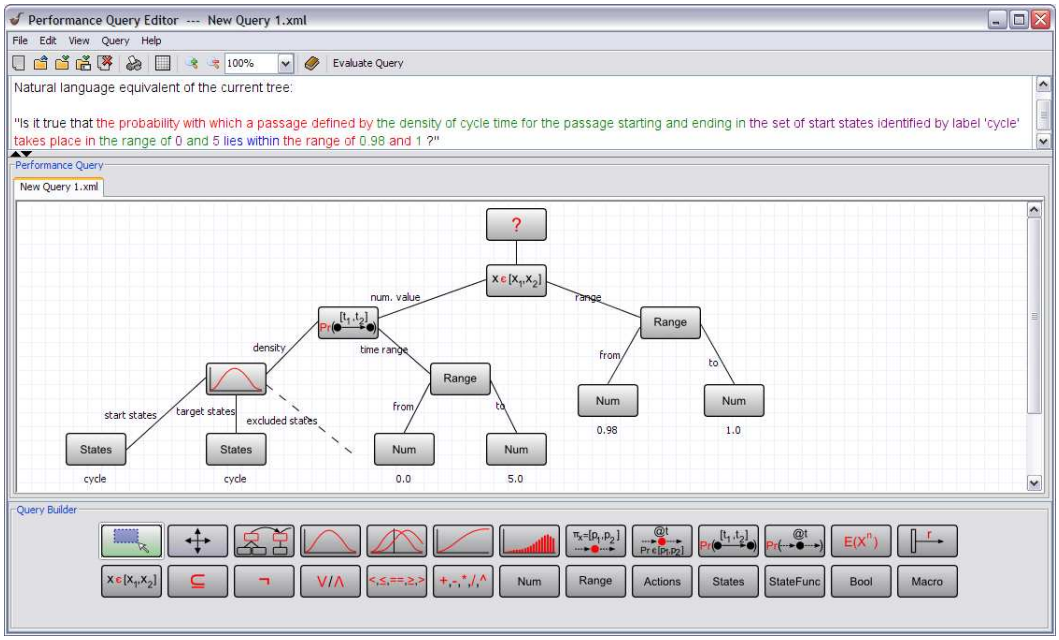


Fig. 2. Performance Query Editor module showing a performance requirement verification query

For added convenience, a macro mechanism supports the creation of new concepts through the use of existing operators. Whole subtrees can be conveniently represented by a single parameterisable node and reused within the same or even among different queries.

When a GSPN model of a system and an applicable Performance Tree query have been specified, *PIPE* converts the graphical information into a tool-specific XML format (through Java object serialisation) that is used for the communication with the analysis back-end. Both the GSPN model and the Performance Tree query are converted into XML files, which are sent via a socket connection to the Analysis Server, the coordinating entity of the Evaluation Engine, for further processing. Upon initialisation of a socket connection with the Analysis Server, a dedicated thread is made available to receive the incoming data and to commence the evaluation process.

2.2 Evaluation Engine

2.2.1 Analysis Server

The Analysis Server's main responsibility is the inspection of performance queries and the appropriate distribution of work among the dedicated analysis tools hosted on the Grid cluster. Once the server has obtained the two XML documents describing the system model and the performance query, it constructs its own representation of the data, based on input formats required by the analysis tools. Firstly, a

translation to the *DNAmaca* [6] model specification language is performed, which describes the structure of the model Then, the performance query XML document is inspected to determine what kinds of calculations will have to be carried out and what their dependencies are. Subsequently, the analysis tools invoked by the server parse the model and relevant parts of the query data into C++ code and compile the output with a probabilistic hash-based state space generator library. This is linked with a pre-compiled performance analysis library and executed in order to compute the requested performance measures. Once the server has distributed the jobs amongst the analysis tools, it awaits the results of the calculations, which it then forwards to the client.

2.2.2 Distributed Analysis Tools

The evaluation of quantitative performance measures laid out in the Performance Tree query generated by the client is carried out by a set of distributed analysis tools coordinated by the Analysis Server. Currently, tools are available for the calculation of steady-state measures, first passage time and transient distributions, as well as higher-order moments based on the underlying Markov chain of the GSPN model [6, 13]. The integration of these tools into the analysis pipeline has enabled the evaluation of the majority of Performance Tree operator nodes and has set the framework for future development in implementing support for the few remaining operators (such as convolution).

DNAmaca [6] is a Markov chain steady-state analyser that can solve models with up to $O(10^7)$ states. It features model and performance measure specification in its input language, functional and steady-state analysis and the computation of performance statistics, such as the mean, variance and standard deviation of expressions computed on states in the model and the mean rate at which particular events occur. *DNAmaca* is used for the direct evaluation of the **SS:P** and **FR** nodes, and the indirect evaluation of the **ProbInInterval** and **ProbInStates** nodes.

SMARTA [13], the Semi-MArkov Response Time Analyser, is a parallel MPI-based analysis pipeline for the iterative numerical analysis of passage time and transient measures in very large semi-Markov models (including GSPNs). It is used to evaluate the **PTD** and **Dist** nodes.

MOMA is a performance analyser that calculates n^{th} order (raw) moments using a Laplace transform-based method. It is used to evaluate the **Moment** node.

2.3 Hardware Infrastructure

Camelot, the computational cluster forming the backbone of the evaluation engine, consists of 16 dual-processor dual-core nodes, each of which is a Sun Fire x4100 with two 64-bit Opteron 275 processors and 8GB of RAM. Nodes are connected with both Gigabit Ethernet and Infiniband interfaces; the Infiniband fabric runs at 2.5Gbit/s, managed by a Silverstorm 9024 switch. Job submission is handled by Sun Grid Engine, a middleware that configures and exposes the cluster as a computational Grid resource. Clients submit sequential and parallel (MPI) jobs to Grid Engine via the Distributed Resource Management Application API (DRMAA).

3 Distributed Evaluation of Performance Trees

Performance Tree evaluation occurs on the Analysis Server where the query tree is decomposed into sub-trees for evaluation. The Analysis Server then sends queries derived from each sub-tree to the appropriate distributed analysis tools and waits to collate results to deliver back to the client upon availability.

3.1 Query Processing and Distribution

Before a query is submitted to the Analysis Server, the client performs validation. This is designed to prevent users from building illegal or incomplete performance queries, for instance by connecting incompatible operation nodes (causing a type violation) or by not supplying required arguments to operation nodes.

A single Performance Tree query often requires the calculation of many potentially dependent measures, giving rise to an ordering in which results must be evaluated. Therefore, the scheduling of node evaluation requires an understanding of the dependencies of the sub-trees of the current node. The evaluation of independent nodes takes place in parallel, so that multiple elements in the tree are submitted for evaluation to the cluster at the same time. Dependent nodes are queued and scheduled for evaluation once their dependents have been evaluated. The results from nodes being evaluated usually become available in an unordered fashion and the client is made aware of the availability of results as soon as they arrive at the Analysis Server.

For enhanced speed and efficiency, the Analysis Server incorporates a disk-based cache mechanism that stores the results of performance queries evaluated by the analysis tools. In order to differentiate between different queries on the same model, a hash of the model description and the performance query specification is done separately, using a hashing algorithm with a very low probability of clashes (e.g. MD5). This is used to create a two-level structure in which the computed performance measures can be stored. Before any computation takes place, a cache look-up is performed for the given model. If a match is found, the hash of the current query is compared to all existing hashes of queries on that particular model in the cache. If a further match is found, the query has already been evaluated on the model and the results are already available. If not, the query has to be evaluated, with the results being stored in the cache.

For *DNAmaca*, each performance measure is hashed separately and the corresponding cache location contains the computed steady-state result (either a state measure or count measure). For *SMARTA* and *MOMA*, the cache is particularly useful as calculating the density or distribution of a passage time requires the solution of at least 400 sets of linear equations of the form shown in Eq. 1. Recalling Eq. 3, the values at which the Laplace transform of the passage time must be evaluated are independent of the range of t-values at which the final answer is required. Thus, for a given model and set of initial and target states, results can be computed at any value of t using the same values of the Laplace transform of the passage time. We therefore store the computed values of this Laplace transform indexed by the value of s to which they correspond.

3.2 Query Analysis

Using *DNAmaca*, steady-state performance statistics for the GSPN model are derived by generating and then solving a continuous-time Markov chain (see Section 1.3). From the Markov chain's steady-state probability distribution, high-level performance measures such as throughput and mean buffer occupancy can be derived. These correspond to the **FR** and **SS:P** Performance tree operators.

Passage time analysis on a GSPN model is performed as described in Section 1.4. If the operation nodes **PTD** and **Dist** are part of a Performance Tree query, the user can expect a probability density function (pdf) and a cumulative distribution function (cdf) as a result after the passage time calculation has been completed by *SMARTA* (see Fig. 1).

An interesting issue that arises in displaying the results of the above nodes is the automatic determination of the time range over which a pdf or cdf should be plotted. In both cases, we do this by establishing at what time value t the cdf approaches 1 within some ϵ bound. The pdf or cdf is then plotted between 0 and t.

We consider the complementary cumulative distribution function (ccdf) $F^c(t) = 1 - F(t)$, since it has a structure more suited to numerical inversion (since $F^c(t)$ is a non-negative decreasing function with $F^c(t) \to 0$ as $t \to \infty$). Using Laguerre series representation,

$$F^c(t) = \sum q'_n l_n(t) \tag{4}$$

with $Q'(z) = q'_n z^n$. It can be shown that

$$Q'(z) = \frac{-2(1-z)}{(1+z)} \left(Q(z) - (1-z)^{-1}\right) \tag{5}$$

Thus, the Laguerre coefficients of the ccdf q'_n can be computed based on the Laplace transform $f^*(s)$ of the density function. Furthermore, based on the following relationship, the Laguerre coefficients of the pdf $f(t)$ can also be recursively computed from q'_n:

$$q_n - q_{n-1} = -\frac{1}{2}(q'_n + q'_{n-1}) \tag{6}$$

Finally, the cdf and pdf are obtained as:

$$F(t) = 1 - \sum q'_n l_n(t) \text{ and } f(t) = \sum q_n l_n(t) \tag{7}$$

The time range of interest for the user is given by the time at which $F^c(t) \to \epsilon$ where ϵ is chosen as an arbitrarily small, positive value. Eqs. 5 and 6 are described in more detail in [14].

If the **Moment** operation node is included in a query then *MOMA* is invoked to calculate the required moments using the formula in Section 1.5.

3.3 Results Processing

Once the user has created a valid Performance Tree and submitted the query and model for analysis, they are presented with an evaluation progress window. This shows a replicated version of the Performance Tree query, augmented with an eval-

uation status indicator associated with each node. Initially, the status indicator of every node is red, signaling that evaluation of the nodes has not started yet, but as soon as a node's evaluation has been scheduled (that is, forwarded to the Analysis Server), the respective status indicator turns yellow. Once results for a node have been returned to the client by the Analysis Server, the status indicator turns green and the user can access individual results by clicking on the node.

4 Case Study

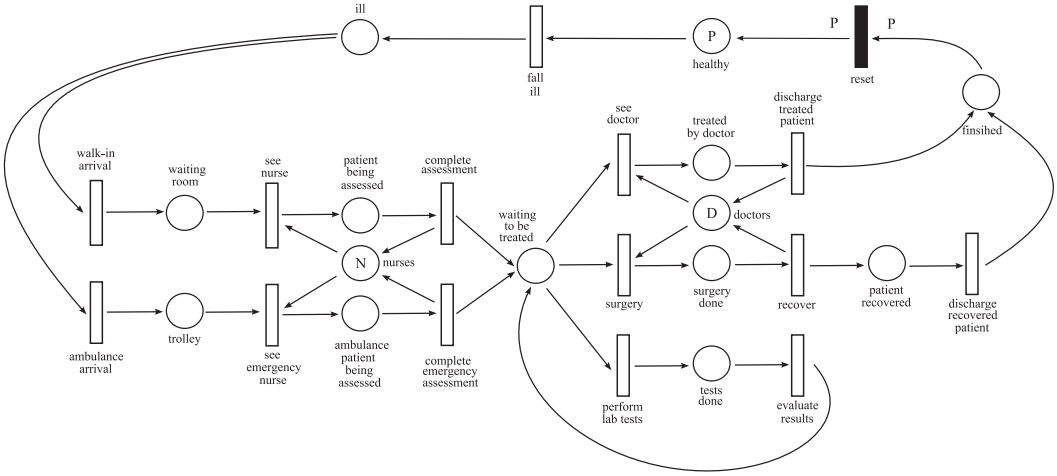


Fig. 3. GSPN model of a Hospital A&E Department

To demonstrate our performance analysis pipeline, we will design queries and calculate relevant results for two example case studies based on a small modified version of the Accident and Emergency (A&E) department model introduced in [2]. In the GSPN model of Figure 3 there is an initial group of healthy people who fall ill and go to a hospital – arriving either as walk-in patients or by ambulance. Walk-in patients await assessment by a nurse, while ambulance patients are given prioritised attention. Patients are subsequently either seen by a doctor for treatment, sent for lab tests, or taken for surgery. Once a patient is discharged from the hospital, (s)he is assumed to be healthy again. The model is parameterised with P, N and D, which denote the number of tokens on the places *healthy* (people), *nurses* and *doctors*, respectively. In the following examples, we set $P = 5$, $N = 2$ and $D = 2$, yielding an underlying Markov chain with $3,815$ states. In order to illustrate better the types of passage time query that can be asked, we have modified the model to include an extra place ("finished" on the diagram) which collects patients as they leave the hospital. Only when the entire population P has arrived at this place will the immediate transition "reset" become enabled and repopulate the system with healthy individuals. This allows us to measure the time taken to process all patients in the system without the need to identify individuals (i.e. by tagging tokens).

Example 1 : *What is the cumulative distribution function of the time taken for all patients in the system to fall ill, complete treatment and be discharged from the hospital?*

The screen shots of the Performance Tree user interface for the query in Example 1 and its evaluation window are given in Figures 4 and 5, respectively. Relevant state labels for this query are:

$$\textit{all patients healthy} := (\#(\textit{healthy}) = P)$$
$$\textit{all patients treated} := (\#(\textit{finished}) = P)$$

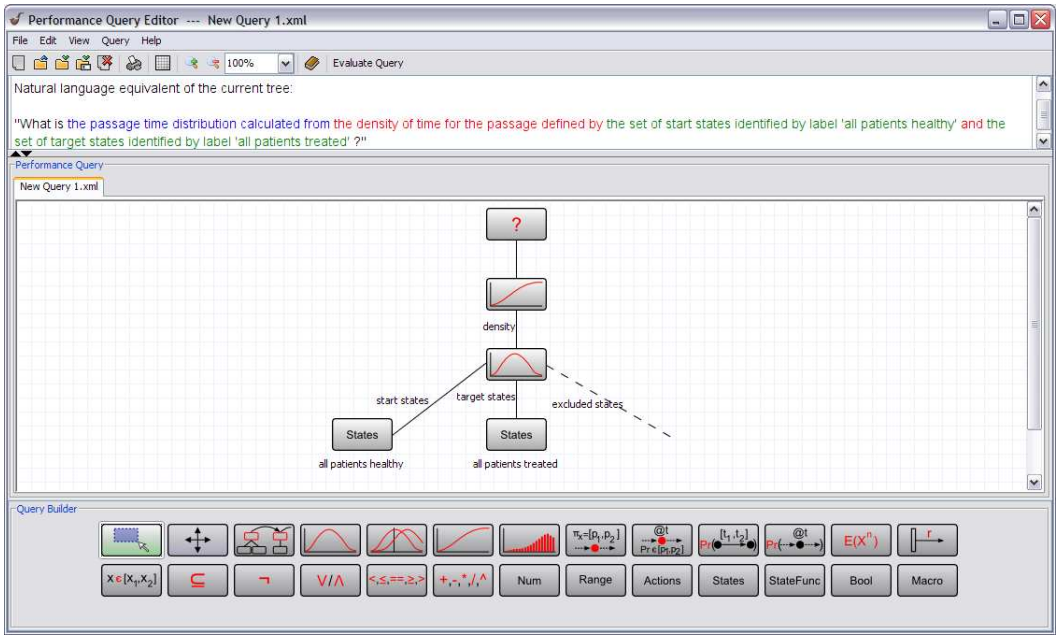


Fig. 4. Query tree with *PTD* and *Dist* operators.

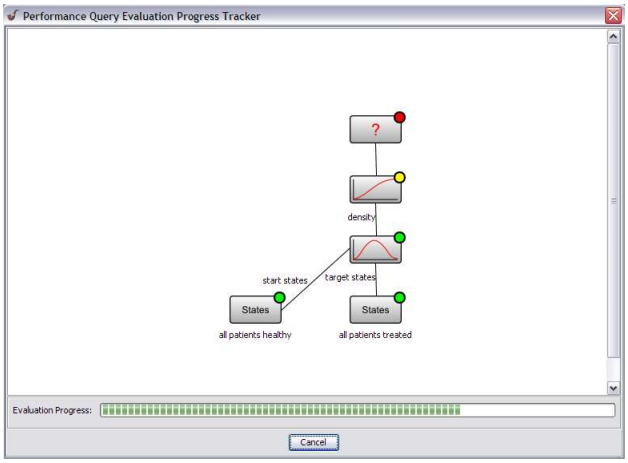
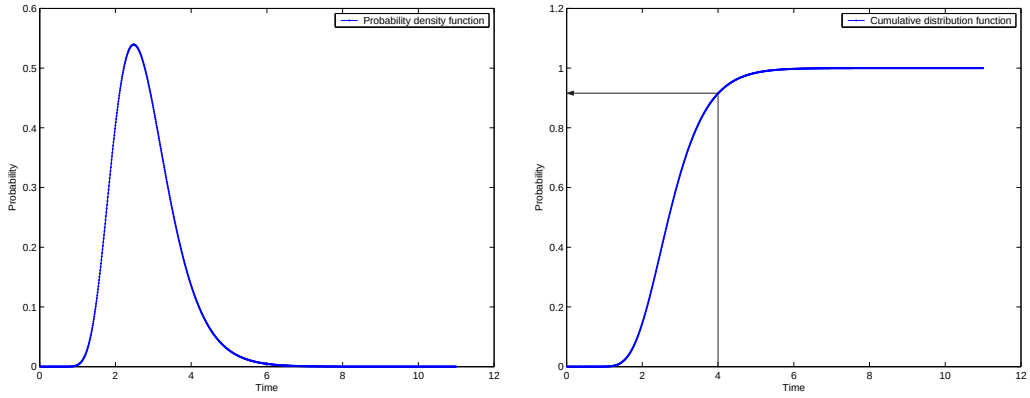


Fig. 5. Evaluation window for the query with *PTD* and *Dist* operators.

As mentioned in the previous section, the time range of interest for pdf and cdf plots is computed automatically. The result generated by the **PTD** operator for the hospital model in this case study is shown in Figure 6(a), while the result generated by the **Dist** operator is shown in Figure 6(b). In the traffic light-based status indicator system shown in Figure 5, a green status indicates that results have

been computed and are available for inspection (by clicking on the corresponding node).



(a) Result of evaluating **PTD** node

(b) Result of evaluting **Dist** node

Fig. 6. Results.

It can be deduced from the results in Figure 6(b) that all A&E patients are treated by time 4 with probability 0.9150 (rounded to 4 decimal places).

Example 2 : *What is the average rate of occurrence of surgeries and the steady-state probability distribution of the number of patients waiting for treatment?*

This example demonstrates three distinct features of Performance Trees: a steady-state probability calculation, a firing rate calculation and the ability to combine multiple independent queries into one. The Performance Tree for this query is given in Figure 7, which shows how independent sub-queries are combined using the **;** operator. The first part of the query is seeking the average rate of occurrence of an action, which is represented by the **FR** node, while the second part of the query addresses a steady-state probability distribution with the **SS:P** node.

The results of this query are as follows. Figure 8 gives the distribution of the number of patients awaiting treatment at steady-state. The average number of patients waiting to be treated at steady-state is 0.316 (to 3 decimal places), with a variance of 0.360 (also to 3 d.p.), and the average rate with which surgery is performed is 0.711 operations per hour.

5 Conclusions and Future Work

In this paper, we have introduced a distributed evaluation architecture for performance queries expressed as Performance Trees on Generalised Stochastic Petri Net models. We have detailed the query design and analysis workflow, and have shown in the context of a case study how results for passage time density, steady-state probability distribution and average firing rate calculations are obtained.

We are currently in the process of integrating support for all presently available Performance Tree operation nodes, as well as a mechanism for the specification of

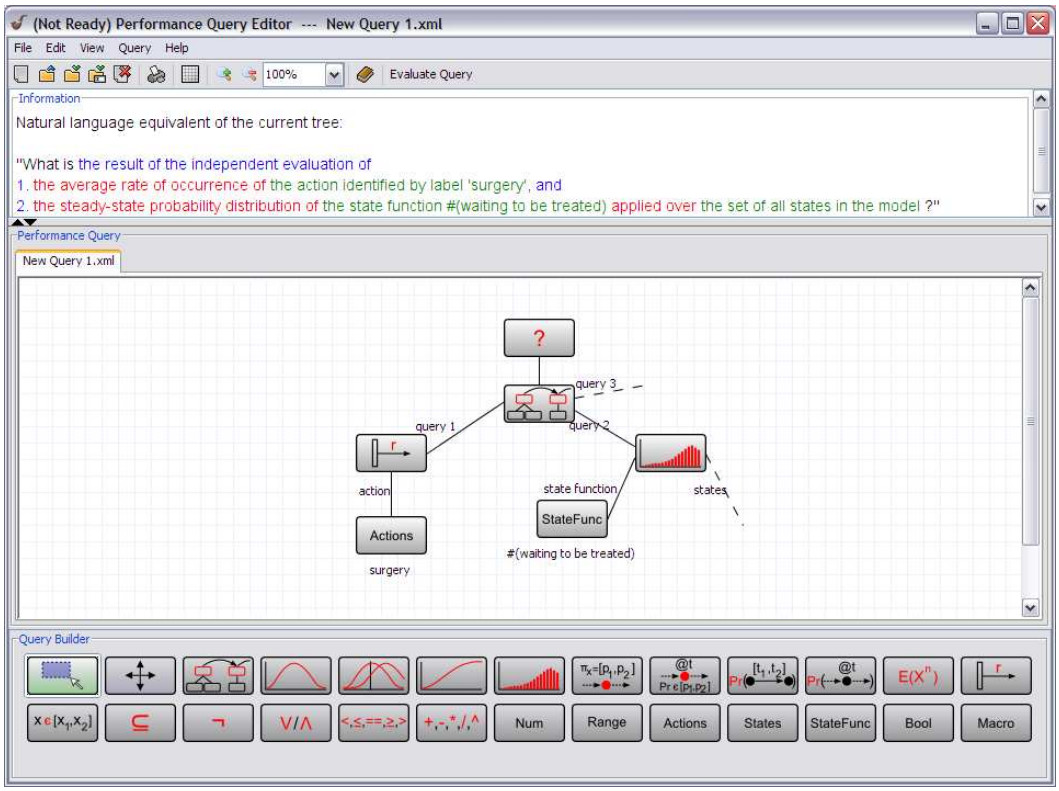


Fig. 7. Steady state and firing rate measure specification.

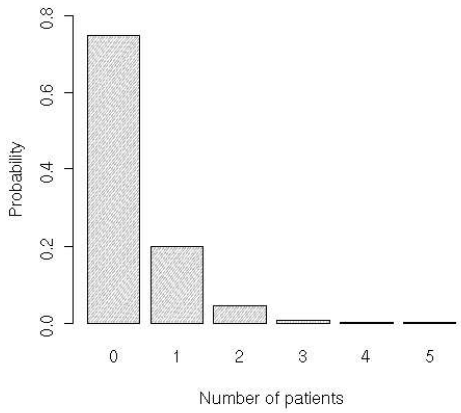


Fig. 8. Distribution of the number of patients waiting to be treated at steady-state.

tagged customers in a system, which will enable us to analyse customer-centric performance queries. Once the whole spectrum of Performance Tree-based queries can be evaluated, we will focus our efforts onto the optimisation and efficient scheduling of computations on the analysis cluster in order to achieve even better response time for our analysis pipeline.

References

[1] T. Suto, J. Bradley, and W. Knottenbelt, "Performance Trees: A New Approach to Quantitative Performance Specification," in *Proc. 14th IEEE/ACM Intl. Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS 2006)*, Monterey, CA, USA, September 2006, pp. 303–313.

[2] ——, "Performance Trees: Expressiveness and Quantitative Semantics," in *Proc. 4th Quantitative Evaluation of Systems Conference (QEST 2007)*, Edinburgh, UK, September 2007, pp. 41–50.

[3] F. Bause and P. Kritzinger, *Stochastic Petri Nets – An Introduction to the Theory*. Wiesbaden, Germany: Verlag Vieweg, 1995.

[4] M. Ajmone-Marsan, G. Conte, and G. Balbo, "A class of Generalised Stochastic Petri Nets for the performance evaluation of multiprocessor systems," *ACM Transactions on Computer Systems*, vol. 2, pp. 93–122, 1984.

[5] G. Ciardo, J. Muppula, and K. Trivedi, "On the solution of GSPN reward models," *Performance Evaluation*, vol. 12, no. 4, pp. 237–253, 1991.

[6] W. Knottenbelt, "Generalised Markovian analysis of timed transition systems," Master's thesis, University of Cape Town, Cape Town, South Africa, July 1996.

[7] P. Harrison and W. Knottenbelt, "Passage time distributions in large Markov chains," in *Proc. ACM SIGMETRICS 2002*, Marina Del Rey, California, June 2002, pp. 77–85.

[8] N. Dingle, P. Harrison, and W. Knottenbelt, "Response time densities in Generalised Stochastic Petri Net models," in *Proc. 3rd Int. Workshop on Software and Performance (WOSP 2002)*, 2002, pp. 46–54.

[9] J. Bradley, N. Dingle, W. Knottenbelt, and H. Wilson, "Hypergraph-based parallel computation of passage time densities in large semi-Markov models," *Linear Algebra and its Applications*, vol. 386, pp. 311–334, 2004.

[10] J. Abate, G. Choudhury, and W. Whitt, "On the Laguerre method for numerically inverting Laplace transforms," *INFORMS Journal on Computing*, vol. 8, no. 4, pp. 413–427, 1996.

[11] W. Weeks, "Numerical inversion of Laplace transforms using Laguerre functions," *Journal of the ACM*, vol. 13, no. 3, pp. 419–429, 1966.

[12] PIPE: Platform-Independent Petri net Editor – http://pipe2.sourceforge.net.

[13] N. Dingle, "Parallel computation of response time densities and quantiles in large Markov and semi-Markov models," Ph.D. dissertation, Imperial College, London, United Kingdom, 2004.

[14] H. Kulatunga and W. Knottenbelt, "Cumulative distribution function calculation using Laguerre transform inversion," *Electronics Letters*, To be submitted in January 2008.