Performance Trees
Implementation and Distributed Evaluation

Darren Brien, Nicholas Dingle, William Knottenbelt, Harini Kulatunga and Tamas Suto

Department of Computing
Imperial College London
United Kingdom

PDMC’08
29 March 2008, Budapest
Performance Modelling

- Performance modelling facilitates a detailed understanding of critical behavioural characteristics of systems.

- Hence, it is crucial to system designers to be able to extract performance measures from system models.

- This has sparked great demand for accessible ways of modelling systems and analysing their performance.
Query Specification With Performance Trees

- Need a way to specify performance queries on system models in a convenient manner
- Our approach: Performance Trees
- Why? Because they are
  - expressive
  - accessible
  - versatile
Tool support is critical to encouraging the adoption of any specification mechanism.

This talk’s focus is on an integrated performance analysis environment that provides tool support for Performance Tree-based query specification and evaluation.

Our analysis environment features double-layered distribution and parallelism:

- Concurrent evaluation of multiple query nodes, respecting dependencies.
- Parallel evaluation of individual computationally-intensive nodes.
Performance Trees

- A graphical formalism for performance property and measure specification that
  - incorporates quantitative analysis-oriented features as well as traditional property verification capabilities
  - can represent a number of high-level concepts, such as distributions, densities, convolutions, percentiles and moments
  - is extensible in a number of ways
  - can be used on multiple modelling formalisms
  - supports accessible graphical query manipulation
# Performance Tree Operators

<table>
<thead>
<tr>
<th>Graphical</th>
<th>Textual</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Mult" /></td>
<td>Mult</td>
<td>Evaluation of multiple independent performance queries</td>
</tr>
<tr>
<td><img src="image" alt="PTD" /></td>
<td>PTD</td>
<td>Passage time density</td>
</tr>
<tr>
<td><img src="image" alt="Dist" /></td>
<td>Dist</td>
<td>Passage time distribution</td>
</tr>
<tr>
<td><img src="image" alt="Conv" /></td>
<td>Conv</td>
<td>Convolution of two passage time densities</td>
</tr>
<tr>
<td><img src="image" alt="Perctl" /></td>
<td>Perctl</td>
<td>Percentile of a passage time distribution or density</td>
</tr>
</tbody>
</table>
## Performance Tree Operators

<table>
<thead>
<tr>
<th>Graphical</th>
<th>Textual</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Pr(t₁,t₂)" /></td>
<td>ProbInInterval</td>
<td>Probability of a passage occurring in a given amount of time</td>
</tr>
<tr>
<td><img src="image" alt="SS:P" /></td>
<td>SS:P</td>
<td>Steady-state probability distribution of a state function over a set of states</td>
</tr>
<tr>
<td><img src="image" alt="Pr(at)" /></td>
<td>ProbInStates</td>
<td>Transient probability of being in a set of states at a given time</td>
</tr>
<tr>
<td><img src="image" alt="E(X^n)" /></td>
<td>Moment</td>
<td>Raw moment of a passage time density or distribution</td>
</tr>
<tr>
<td><img src="image" alt="FR" /></td>
<td>FR</td>
<td>Average occurrence rate of an action</td>
</tr>
</tbody>
</table>
## Performance Tree Operators

<table>
<thead>
<tr>
<th>Graphical</th>
<th>Textual</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;, ≤, ==, ≥, &gt;</td>
<td>&lt;, ≤, ==, ≥, &gt;</td>
<td>Arithmetic comparison</td>
</tr>
<tr>
<td>+, −, *, /, ^</td>
<td>+, −, *, /, ^</td>
<td>Arithmetic operation</td>
</tr>
<tr>
<td>∨, ∧</td>
<td>∨, ∧</td>
<td>Boolean dis-/conjunction</td>
</tr>
<tr>
<td>¬</td>
<td>¬</td>
<td>Boolean negation</td>
</tr>
<tr>
<td>Macro</td>
<td>Macro</td>
<td>New concept defined by a combination of existing operators</td>
</tr>
</tbody>
</table>
Example Model: A Hospital’s A&E Department
Example Query
State Labels for Example Query

\[\text{start} \ := \ (\#(\text{healthy}) \ == \ 10) \land (\#(\text{nurses}) \ == \ 3) \land (\#(\text{doctors}) \ == \ 3)\]
target := (#(patient recovered) ≥ 1)
A Distributed Analysis Environment for PTs
Client Front-End
Client Front-End

- **PIPE2**, the Java-based open-source platform-independent Petri net editor
- Was originally developed to support the graphical design and animation of GSPNs
- Has been extended with a number of sophisticated analysis modules
- Most significant extension: Performance Query Designer module – implements support for the design and manipulation of Performance Tree queries
Client Front-End: Model Designer
Client Front-End: Query Designer

**Introduction**

**Performance Trees**

**Analysis Environment**

**Query Processing**

**Summary**

Client Front-End: Query Designer

Performance Query Editor — example-query.xml

Information

Natural language equivalent of the current tree:

"Is it true that the probability with which a passage defined by the passage time density of the passage defined by the set of start states identified by label "start" and the set of target states identified by label "target" takes place in the range 0 to 5 lies within the range 0.98 to 1?"
Client Front-End: Evaluation Progress Tracker
Client Front-End: Evaluation Progress Tracker
Client Front-End: Results (PTD Node)
Client Front-End: Results (ProbInInterval Node)
Client Front-End: Results (InInterval Node)
Analysis Server

- The coordinating entity of the evaluation back-end
- Accepts incoming requests from clients and delegates their processing to dedicated analysis threads
- Analysis threads are responsible for
  - performing dependency analysis on the query nodes
  - creating an execution schedule
  - coordinating helper threads that manage the evaluation of individual query subtrees
  - sending results to the client front-end for visualisation
Distributed Analysis Tools

- Performance Trees
- Analysis Environment
- Query Processing
- Summary

Diagram showing the integration of components for distributed analysis, including PIPE, Analysis Server, Sun Grid Engine, and various tools like DNAmaca, SMARTA, HYDRA, MOMA, PERC, and CONE.
Distributed Analysis Tools

The evaluation back-end is supported by an array of parallel and distributed analysis tools, such as

- **DNAmaca**, a steady-state solver for models with up to $O(10^7)$ states
- **SMARTA**, a semi-Markov response time analyser that uses hypergraph partitioning and Laplace transform inversion
- **HYDRA**, a Markovian passage time and transient analyser that uses hypergraph partitioning and uniformisation
- **MOMA**, an $n^{\text{th}}$ order raw moment calculator for GSPN models
- **PERC**, an analyser calculating percentiles of passage time distributions and densities
- **CONE**, an analyser computing convolutions of passage time densities
Hardware Infrastructure
The *Camelot* cluster forms the computational backbone of the analysis environment

- 16 dual-processor, dual-core nodes with 8GB RAM each
- Nodes are connected with both Gigabit Ethernet and Infiniband interfaces (2.6Gbit/s)
- Job management is handled by Sun Grid Engine, which exposes the cluster as a computational Grid resource
- Sequential and parallel MPI jobs are submitted to the cluster through a DRMAA interface
Query Validation

Need to ensure that only valid and complete Performance Tree queries are specified by users, as

- invalid node assignments might be attempted
- an insufficient number of arguments may have been provided to certain nodes

Performed by the client, both during query construction and before evaluation
A performance query often requires the calculation of many potentially dependent measures. This gives rise to an ordering that determines when and how results are to be evaluated. A dependency analysis is therefore necessary by decomposing a query into a set of subtrees using breadth-first postorder node sequencing and establishing an execution schedule for subtrees.
The evaluation of independent subtrees is performed in parallel – multiple subtrees may be submitted to analysis tools at the same time.

Dependent subtrees are scheduled for evaluation from a priority queue that takes dependencies into account.

Helper threads coordinate subtree evaluation.

Jobs are despatched to analysis tools once their dependents have completed execution.
Subtree Evaluation

- Depending on the type of subtree, different analysis tools are invoked on the analysis cluster.
- Helper threads communicate with SGE and manage the evaluation jobs.
- SGE uses built-in scheduling algorithms to determine the distribution of incoming jobs on available processors.
Results Processing

- The client front-end maintains an interface for the user to monitor the query evaluation progress.
- Status indicators obeying a traffic-light scheme signal the evaluation state of individual nodes.
- Analysis server communicates evaluation results to the client as soon as they become available.
- Users can visualise results of individual nodes that have been evaluated by clicking on them.
Conclusions

- We have presented a sophisticated distributed evaluation environment for Performance Tree-based queries which
  - provides portal-like access to a wide range of parallel and distributed modelling and analysis tools
  - enables user-friendly and automated specification and evaluation of complex performance queries
  - allows a wide range of performance metrics to be obtained from high-level system models
Work In Progress

- Evaluation support for a few remaining operators
- \textit{PEPA}-based model specification
- Performance Tree-based query specification on \textit{PEPA} models
- Optimisation strategies to increase evaluation efficiency
Thank you for your attention.

Any questions?