Response Time Analysis with Mapreduce

Distributed Response Time Analysis of GSPN Models with MapReduce

Department of Computing, Imperial College London

Oliver J Haggarty, William J Knottenbelt, Jeremy T Bradley
Outline

• Context
• Generalised Stochastic Petri Nets
• Response Time Analysis
• MapReduce
• PIPE2 Module
• Results
• Conclusions
Context

- Important to model systems to ensure they comply with SLA-specified QoS requirements
- Response time densities and quantiles give good indication of customer-perceived performance
- Here we build GSPN models of systems and extract response time measures from them
- This is a very computationally expensive procedure so we experiment with MapReduce paradigm to distribute computation
GSPNs

- Graphical formalism for describing concurrency / synchronisation

- Places
- Timed Transitions
- Immediate Transitions
- Tokens
GSPN Performance Analysis

Petri net and its reachability graph

Corresponding CTMC and Q matrix

Q matrix
Response Time Analysis

• Convolve state sojourn time densities for all paths from start state to target states in Laplace domain
• Exploits property that Laplace Transform of convolution of two functions is product of their Laplace transforms
• Solve sets of linear equations (of rank $n$) in $Ax=b$ form:

$$
\begin{pmatrix}
    s - q_{11} & -q_{12} & 0 & \cdots & -q_{1n} \\
    0 & 1 & 0 & \cdots & -p_{2n} \\
    0 & -q_{32} & s - q_{33} & \cdots & -q_{3n} \\
    0 & \vdots & 0 & \ddots & \vdots \\
    0 & -q_{n2} & 0 & \cdots & s - q_{nn}
\end{pmatrix} \begin{pmatrix}
    q_{13} \\
    p_{21} + p_{23} \\
    q_{31} \\
    \vdots \\
    q_{n1} + q_{n3}
\end{pmatrix} = L = \left[ L_{1j}(s), \ldots, L_{nj}(s) \right]
$$

where $L = \left[ L_{1j}(s), \ldots, L_{nj}(s) \right]$
Laplace Transform inversion

- Can use Euler inversion to find probability density function (PDF) or cumulative distribution function (CDF)
- Given a value of $t$, evaluates Laplace transform at various s-values from which it approximates the inverse $f(t)$ or $F(t)$
- Each s-value evaluation involves solving a complex linear system of rank $n$
- There are typically thousands of s-value evaluations required
MapReduce

- Devised by Dean and Ghemawat from Google
- Programming model to generate and process large data sets
- Inspired by primitive functions of Lisp

Map\((k_1,v_1)\) \(\rightarrow\) list\((k_2,v_2)\)
Reduce\((k_2,\text{list}(v_2))\) \(\rightarrow\) \((k_2,v_2)\)
Hadoop

- Open source Java implementation
- Consists of distributed filesystem HDFS and MapReduce framework
- Replication within HDFS used to reduce network bandwidth
- Built-in fault tolerance
PIPE2 Response Time Analysis Module

- Platform Independent Petri net Editor
  http://pipe2.sourceforge.net
- Developed by several groups of Imperial students and external collaborators
- Intuitive Petri net editor compatible with Petri Net Markup Language
- Pluggable analysis modules
PIPE2 Response Time Analysis Module

- PNML Data for GSPN graphical model
- Reachability Graph Generator
- Sparse Q Matrix Generator
- Laplace Transform Inverter
- Graph Display
- Steady State Solver
- User Input
- Dynamic Start/Target State Identifier
- Laplace Transform Matrices Generator
- s-Value Generator
- Gauss-Seidel Linear Eqn Solver
- Laplace Transform Inverter

MapReduce Framework

Serialise Matrices

Reconstruct Matrices
Hadoop helps with inversion

- Linear equation matrix copied into HDFS as binary file
- Groups of s-values assigned to TaskTrackers
- Map function solves a set of linear equations for each s-value using Gauss-Seidel
- Also calculates $L(s)/s$ to calculate CDF
Now for the easy bit …

- Reduce function collects all results into single file
- Results used by Euler algorithm to invert Laplace Transform
- This is now a trivial calculation as hard work of s-value evaluation is already done
Results

- Validation with Branching Erlang
Branching Erlang

- Trivial to calculate analytical results
- Proves capability to handle overlapping source and target states and bimodal density curves
- Excellent accuracy
Courier Protocol

- Larger model
- Changing window size allows for larger state spaces
- Test scalability with window size = 1
- State space = 29 010
- Examine pdf of time for acknowledgement to arrive after transmission is initiated by sender
## Scalability

<table>
<thead>
<tr>
<th>Cluster Size</th>
<th>No. Maps Per Node</th>
<th>Total Cores</th>
<th>Total Maps</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>3112.167</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>20</td>
<td>1596.322</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>40</td>
<td>809.653</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>8</td>
<td>80</td>
<td>433.173</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>16</td>
<td>80</td>
<td>256.694</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>32</td>
<td>80</td>
<td>192.982</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>15</td>
<td>80</td>
<td>252.515</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>30</td>
<td>80</td>
<td>165.561</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>60</td>
<td>100</td>
<td>131.754</td>
</tr>
</tbody>
</table>

**Hardware details:**

- Cluster of 15 Sun Fire x4100 machines
- Each with 2 dual-core, 64-bit Opteron 275 processors and 8GB RAM
- Connected by Infiniband with throughput of 2.5 Gbit/s
Performance with larger models

- Courier Protocol, window size = 3
- State space = 2,162,610
- Transitions between states = 5,469,150
- Start states = 439,320
- Target states = 273,260
- 50 t-points, 3250 s-value evaluations
- Laplace transform inversion = 8 hours 9 min (using 15*4 cores)
Conclusions

- Excellent scalability – approaching linear reduction in times for increasing cluster size
- Granularity of Map tasks has important effect on performance – particularly on heterogeneous clusters
- Good reliability from open source software
- Ease of use – framework provides most of functionality of distributing processing
The end.

Thanks … Any questions?
User Interface