REX: A Development Platform and Online Learning Approach for Runtime Emergent Software Systems

Barry Porter, Matthew Grieves, Roberto Filho, David Leslie
Introduction

- Designing, Analyzing and Maintaining – Millions of LOC: Is it sustainable?
- Software development: Models, Policies, and Processes
- Autonomous, Self-Adaptive, and Self Organized Software System
- Emergence of Software System - Autonomously from pool of available building blocks
- Responsive to actual runtime conditions.
- Can show rationale behind the choice
REX: Development Platform

- Implementation Platform - Dana (Dynamic Adaptive Nucleic Architecture)
- PAL Framework
  - Perceive – Internal + External conditions
  - Assemble and Re-assemble modules
  - Learn
- Online Learning – Statistical Linear Bandits, using Thompson Sampling.
Dana

- Component Based Software Paradigm
- All Components – Runtime Replaceable
- Multi-threaded imperative language (what and how)

Example: From Source code

component provides App requires io.Output out{
    int App:main(AppParam params[]){
        out.println("Hi! :-)");
        return 0
    }
}

interface File {
    transfer char path[]
    transfer int pos, mode
    File(char path[], int mode)
    byte[] read(int numBytes)
    int write(byte data[])
    bool eof()
    void close()
}

component provides App requires File {
    int App:main(AppParam args[]) {
        File ifd = new File(args[0].str, File.READ)
        File ofd = new File(args[1].str, File.WRITE)
        while (!ifd.eof()) ofd.write(ifd.read(128))
        ofd.close()
        ifd.close()
        return 0
    }
}

Figure 2 – Example interface to open, read and write files (top); and a component that uses this interface to copy a file (bottom).
Dana: Runtime Adaptation

Algorithm 1 Adaptation protocol

1: \textit{srcCom} $\triangleright$ Comp. to rewire a required interface of
2: \textit{sinkCom} $\triangleright$ Comp. with provided interface to wire to
3: \textit{intfName} $\triangleright$ Interface name being adapted
4: pause(\textit{srcCom.intfName})
5: \textit{r}_\textit{objs} = getObjects(\textit{srcCom.intfName})
6: rewire(\textit{srcCom.intfName}, \textit{sinkCom})
7: resume(\textit{srcCom.intfName})
8: for \textit{i} = 0 to \textit{r}_\textit{objs}.arrayLength - 1 do
9: \hspace{1em} if pauseObject(\textit{r}_\textit{objs}[\textit{i}]) then
10: \hspace{2em} \textit{a} = adaptConstruct(\textit{sinkCom.intfName}, \textit{r}_\textit{objs}[\textit{i}])
11: \hspace{2em} \textit{b} = rewireObject(\textit{r}_\textit{objs}[\textit{i}], \textit{a})
12: \hspace{2em} resumeObject(\textit{r}_\textit{objs}[\textit{i}])
13: \hspace{2em} waitForObject(\textit{b})
14: \hspace{2em} adaptDestroy(\textit{b})
15: \hspace{1em} end if
16: end for

Figure 4 – Adaptation sequence overview. A selected required interface \textit{r} is rewired, followed by each object in the set \textit{r}_\textit{objs}.
PAL Framework: Perception & Assembly

- **Perception**
  - Implemented using Recorder Interface
  - Data – Event and Metrics (Name, Value, Flag)

- **Assembly**
  - Starts with main component of target system
  - Read Required components (recursively)
  - Search interfaces in resources directory and their potential implementation
  - Example – Interface (io.File) → io (Implementation Directory)
  - Create a list of configurations
  - Use Adaptation protocol to reassemble
Sample Implementation: Webserver

- Number of components in system = 30
- File System, String Parsers,
- Number of configurations – $2 \times 3 \times (2+5) = 42$
- Request Handler – Avg response time
- HTTPHandler – Events for requested resource & their size
Exploration Vs Exploitation:

- Upper Confidence Bound Action Selection

\[ A_t = \arg \max_a \left[ Q_t(a) + c \sqrt{\frac{\ln t}{N_t(a)}} \right] \]

- Greedy – Exploit current knowledge to maximize immediate reward

- Posterior Sampling- Thompson Sampling

- Estimate posterior distribution using prior distribution
Multi-armed Bandit

- Arm – One configuration of webserver
- Action – choose one config and deploy

\[ \beta_0 + \beta_1 \Pi_{\text{RequestPT}} + \beta_2 \Pi_{\text{HttpCMP}} + \beta_3 \Pi_{\text{HttpCH}} + \beta_4 \Pi_{\text{HttpCHCMP}} + \beta_5 \Pi_{\text{Deflate}} + \beta_6 \Pi_{\text{CacheFS}} + \beta_7 \Pi_{\text{CacheLFU}} + \beta_8 \Pi_{\text{CacheMRU}} + \beta_9 \Pi_{\text{CacheLRU}} + \beta_{10} \Pi_{\text{CacheRR}} \]  

\( x_{\text{conf}} = (1, \Pi_{\text{RequestPT}}, \Pi_{\text{HttpCMP}}, \Pi_{\text{HttpCH}}, \Pi_{\text{HttpCHCMP}}, \Pi_{\text{Deflate}}, \Pi_{\text{CacheFS}}, \Pi_{\text{CacheLFU}}, \Pi_{\text{CacheMRU}}, \Pi_{\text{CacheLRU}}, \Pi_{\text{CacheRR}}) \)

\[ y = x_{\text{conf}} \beta + \epsilon, \]

Algorithm 2 Learning Algorithm

1: \( // \) matrix of all available \( x_{\text{conf}} \) vectors (configurations)
2: \( \text{actionMatrix} = \text{assembly.getConfigs}() \)
3: \( X = \text{new Matrix}() // \) list of observed \( x_{\text{conf}} \)’s to date
4: \( y = \text{new Vector}() // \) list of rewards seen for each \( X \)
5: \( n = 0 \)
6: \textbf{while running do}
7: \( // \) do linear regression & sample from posterior
8: \( \Lambda = X^T X + \Lambda_0 \)
9: \( \beta = \Lambda^{-1} (\Lambda_0 \beta + X^T y) \)
10: \( a = a_0 + (n/2) \)
11: \( b = b_0 + (y^T y + \beta^T \Lambda_0 \beta - \beta^T \Lambda \beta) \times 0.5 \)
12: \( \sigma^2 = \text{new InverseGamma}(a, b).sample() \)
13: \( \text{sample} = \text{new Normal}(\beta, \sigma^2 \Lambda^{-1}).sample() \)
14: \( // \) select the new configuration to use
15: \( i = \arg \max(\text{actionMatrix} * \text{sample}) \)
16: \( \text{assembly.setConfig}(i) \)
17: \( // \) wait for 10 seconds, then record observations
18: \( \text{result} = 1/\text{perception.getAverageMetric}() \)
19: \( \text{add row } i \text{ of } \text{actionMatrix} \text{ as new row of } X \)
20: \( \text{add } \text{result} \text{ as new element of } y \)
21: \( n++ \)
22: \textbf{end while}
Handling Environment Changes

- **Entropy**
  - High => Request for different resources
  - zero => Single resource requested repeatedly

- **Text Volume** – Highly Compressible
  - Example – HTML, CSS Files

- **High entropy interval** – more than 50% request of high entropy

- **7 Extra Regression Coefficients**

- **Total number of configurations** = 42*4 = 168

(1, \(\#\text{RequestPT}\), \(\#\text{HiEnt}\), \(\#\text{HiTxt}\), \(\#\text{HttpCMP}(\text{LowTxt})\), \(\#\text{HttpCMP}(\text{HiTxt})\), \(\#\text{HttpCH}(\text{LowEnt})\), \(\#\text{HttpCH}(\text{HiEnt})\), \(\#\text{HttpCHCMP}(\text{LowTxt,LowEnt})\), \(\#\text{HttpCHCMP}(\text{HiTxt,LowEnt})\), \(\#\text{HttpCHCMP}(\text{LowTxt,HiEnt})\), \(\#\text{HttpCHCMP}(\text{HiTxt,HiEnt})\), \(\#\text{Deflate}\), \(\#\text{CacheFS}\), \(\#\text{CacheLFU}\), \(\#\text{CacheMRU}\), \(\#\text{CacheLRU}\), \(\#\text{CacheRR}\)).
Results – Runtime Adaptation

- Webserver is actually paused
- `pauseObject` – busy waiting for new function call
- `pause` – prevent new objects from being instantiated

<table>
<thead>
<tr>
<th>Function</th>
<th>Average</th>
<th>Maximum</th>
<th>Minimum</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setConfig (idle)</code></td>
<td>599.60 ms</td>
<td>615.00 ms</td>
<td>397.00 ms</td>
</tr>
<tr>
<td><code>setConfig (busy)</code></td>
<td>1350.32 ms</td>
<td>5811.00 ms</td>
<td>510.00 ms</td>
</tr>
<tr>
<td><code>pause/resume (idle)</code></td>
<td>8.50 μs</td>
<td>9.94 μs</td>
<td>7.81 μs</td>
</tr>
<tr>
<td><code>pause/resume (busy)</code></td>
<td>13.22 μs</td>
<td>31.21 μs</td>
<td>8.51 μs</td>
</tr>
<tr>
<td><code>pauseObject/resumeObject (idle)</code></td>
<td>4.51 μs</td>
<td>5.34 μs</td>
<td>3.84 μs</td>
</tr>
<tr>
<td><code>pauseObject/resumeObject (busy)</code></td>
<td>28.54 μs</td>
<td>387.17 μs</td>
<td>4.35 μs</td>
</tr>
<tr>
<td>components adapted in <code>setConfig()</code></td>
<td>1.22</td>
<td>3.00</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Table 1 – Adaptation speed measured in different ways, from full configuration changes to individual component adaptations.
## Results: Divergent Systems

<table>
<thead>
<tr>
<th>Entropy</th>
<th>Text</th>
<th>Low</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td></td>
<td>Cache</td>
<td>Cache &amp; Compress</td>
</tr>
<tr>
<td>High</td>
<td></td>
<td>Default (Due to hash collision)</td>
<td>Cache &amp; compression</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Request pattern</th>
<th>File size (b) [GZ]</th>
<th>Default</th>
<th>Caching</th>
<th>Caching &amp; compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text low entropy</td>
<td>156,983 [12,757]</td>
<td>11.94 ms</td>
<td>9.56 ms</td>
<td><strong>0.70 ms</strong></td>
</tr>
<tr>
<td>Text high entropy</td>
<td>82,628 [11,949]</td>
<td>4.05 ms</td>
<td>0.60 ms</td>
<td>0.66 ms</td>
</tr>
<tr>
<td>Text low entropy</td>
<td>3,869 [1,930]</td>
<td>1.18 ms</td>
<td>0.59 ms</td>
<td>0.63 ms</td>
</tr>
<tr>
<td>Image low entropy</td>
<td>1,671,167 [1,667,464]</td>
<td>160.81 ms</td>
<td>150.72 ms</td>
<td>154.42 ms</td>
</tr>
<tr>
<td>Image low entropy</td>
<td>84,760 [66,914]</td>
<td>4.02 ms</td>
<td>0.66 ms</td>
<td>0.74 ms</td>
</tr>
<tr>
<td>Image low entropy</td>
<td>4,001 [3,895]</td>
<td>1.22 ms</td>
<td>0.55 ms</td>
<td>0.62 ms</td>
</tr>
<tr>
<td>Text high entropy</td>
<td>156,983 [12,757]</td>
<td>19.27 ms</td>
<td>19.66 ms</td>
<td><strong>3.04 ms</strong></td>
</tr>
<tr>
<td>Text high entropy</td>
<td>82,628 [11,949]</td>
<td>4.61 ms</td>
<td>3.27 ms</td>
<td>3.07 ms</td>
</tr>
<tr>
<td>Text high entropy</td>
<td>3,869 [1,930]</td>
<td>1.25 ms</td>
<td>2.93 ms</td>
<td>2.52 ms</td>
</tr>
<tr>
<td>Image high entropy</td>
<td>1,671,167 [1,667,464]</td>
<td>156.50 ms</td>
<td>156.64 ms</td>
<td>157.66 ms</td>
</tr>
<tr>
<td>Image high entropy</td>
<td>84,760 [66,914]</td>
<td>4.48 ms</td>
<td>3.19 ms</td>
<td>2.94 ms</td>
</tr>
<tr>
<td>Image high entropy</td>
<td>4,001 [3,895]</td>
<td>1.30 ms</td>
<td>2.90 ms</td>
<td>2.67 ms</td>
</tr>
</tbody>
</table>

*Table 2 – Results of different configurations under different request patterns, showing average response times. The standard deviation throughout these results is low, at around 0.2.*
Results:

- 1 test iteration = 10 second (1000 experiments)
- Large File => Less training samples

**Figure 7** – Learning using response times to large text files, with adjusted prior values for $\beta_0$ and $\theta_0$.

**Figure 9** – Learning using response times to a realistic (and highly varying) request pattern, using the NASA server trace [2].
Results: Alternating Request Pattern

- Left - Constantly forget and re-learn

Figure 8 – Learning without (left) and with (right) categorization on a request pattern that changes every ten iterations.
Thoughts

- Will the adaptation be computationally expensive as number of components and metrics increase? – Scalable?
- Impact on QoS during transition
- Ease of adding access patterns in model
- Overhead of providing various implementation for a component Vs Simple Knob Tuning
- Extra overhead of module loading for a large system
Questions ?