Enhancing Performance And Reliability of Rule Management Platforms

Mark Grechanik and B.M. Mainul Hossain

University of Illinois at Chicago
RuleE Management Platforms (REMPs) allow software engineers to represent programming logic as conditional sentences that relate statements of facts (i.e., rules) using high-level declarative languages.
Rule Structure And Example

Rules Format:

when <conditions>
then <actions>

Example:

rule simple_rule:
  When
    Student( points >= 80)
  then
    System.out.println("A+");
End
An Interesting Question

<table>
<thead>
<tr>
<th>Rule A:</th>
<th>Rule B:</th>
</tr>
</thead>
<tbody>
<tr>
<td>When Product ( Price &lt; 80) Then set( Price, 100);</td>
<td>When Product ( Price &gt; 50) Then set( Price, 200);</td>
</tr>
</tbody>
</table>

Price = 65
An Interesting Question

Rule A:
When Product ( Price < 80)
Then set( Price, 100);

Rule B:
When Product ( Price > 50)
Then set( Price, 200);

Price = 65
An Interesting Question

Rule A:
When Product ( Price < 80) Then set( Price, 100);

Rule B:
When Product ( Price > 50) Then set( Price, 200);

Price = 65
If-Then Decision Points

Alternative to rules is the imperative model where sequences of if-then statements with conditionals and loops are evaluated in a strictly defined order.
If-Then Decision Points

Alternative to rules is the imperative model where sequences of if-then statements with conditionals and loops are evaluated in a strictly defined order.

Hard to maintain and inefficient code that is not adaptable to frequent changes in business requirements.
REMPS Are Widely Used

According to market report from Forrester, the estimated revenue of business rules management systems (BRMS) increased from $265 million in 2008 to over $600 million in 2011. It is one of the fastest growing markets.

One leading vendor, IBM ILOG Optimization is used by over 50% of the world's largest companies, 1000’s of Universities, and 1000's of application providers.

CLIPS
JBOSS DROOLS
JESS
Oracle FUSION
BizTalk
How REMPs Work

![Diagram showing the execution of rules in REMPs](image)

- **Execution Platform**
  - Rule language compiler
  - Rule N
  - Rule 2
  - Rule 1

- **Rule language compiler**
  - DB
  - Rule N
  - ... Rule 2
  - ... Rule 1

- **Execution Platform**

- **PK**
- ... P3
- P2
- P1
How REMPs Work

Rule language compiler

Rule N
...  Rule 2  Rule 1

Execution Platform

DB

PK  ...  P3  P2  P1
How REMPs Work

Execution Platform

Rule language compiler

Rule N ... Rule 2 Rule 1

DB

PK ...

P3 P2 P1
How REMPWs Work

Execution Platform

Rule language compiler

Rule N  ...  Rule 2  Rule 1

DB

Execution Platform

PK  ...  P3  P2  P1
How REMPs Work

Rule language compiler

DB

Execution Platform

Rule 1

Rule 2

Rule N

Rule N

…

Rule 1

Rule 2

…

Rule N

PK

P3

P2

P1
How REMPs Work

Execution Platform

Rule language compiler

Rule N

Rule 2

Rule 1

DB

Rule N

Rule 2

Rule 1

Execution Platform

PK

P3

P2

P1

…
How REMPs Work

Execution Platform

Rule language compiler

Rule N  …  Rule 2  Rule 1

DB

No Access

PK  …  P3  P2  P1
Separation of Concerns

A key property of REMPs is that they encapsulate the control flow that includes fact inference and rule firing logics.

- It is a fundamental separation of concerns of the control flow and the rule business logic.

Software engineers concentrate on reasoning about higher-level business logic that they encode in rules without worrying about low-level details of rule invocations by effectively delegating this job to REMP engines.

- Rule-driven APplications (RAPs) are highly adaptable to changing requirements, since stakeholders simply add new rules as independent modules to RAPs.

www.PresentationPro.com
Benefits of REMPs And RAPs

- Easy to maintain & evolve
- Rules are easy to comprehend and highly modular
- RAPs are highly adaptable
Constraints of REMPs And RAPs

- RAPs may contain tens of thousands of rules
- Detection of conflicting rules is difficult
- Dependencies should not be introduced among rules
The PAR Model For REMPss

- Performance
- Reliability
- Adaptability

Big Data

Classic

REMPs
Fundamental Problem

Many REMP's execute RAPs sequentially.

Locks introduce complex dependencies among rules, thereby defeating the separation of concerns and eventually the adaptability of RAPs.

How to enhance the performance of RAPs without sacrificing their adaptability and reliability?
Motivating Example: Credit

rule "Rule-Credit" salience 10
when
   $cashflow : Cashflow( $account:account, $date : date, $amount : amount,
   type==Cashflow.CREDIT )
not Cashflow(account==$account,date<$date)
then
   //some code
   $account.setBalance($account.getBalance()+$amount);
   retract($cashflow);
end
Motivating Example: Debit

rule "Rule-Debit" salience 1
when
    $cashflow : Cashflow( $account : account,
    $date : date, $amount : amount,
    type==Cashflow.DEBIT )
    not Cashflow(account==$account,date<$date)
then
    //some code
    if($account.getBalance()>$amount) {
        $account.setBalance(
            $account.getBalance()-$amount); }
    else { new BlockedAccount($cashflow); }
    retract($cashflow);
end
Parallelism Interferes With Saliences

Let us assume that a REMP engine executes rules in parallel and lock objects are used to synchronize concurrent accesses.

Using a lock object effectively overrides the intention of the programmer to give the priority to the rule with a higher salience.

Given the large number of possible interleavings among tens of thousands of rules in a RAP, it is very difficult to reason about interactions between saliences and synchronization lock mechanisms.
Reliability Meets Performance

In fully parallelized REMPs, loss or reliability comes from two sources:

• different orders in which rules are executed by the REMP engines;
• races between parallelized executions of rules.

When the system produces different results consecutively for the same computational task using the same input data, it is a serious problem, since it reduces the confidence of the user in the RAP and it impacts negatively the perception of the user about the business value that the company or organization delivers.

• the execution order for different instructions can be affected by multiple factors beyond the control of stakeholders
• sometimes even slight changes in the non-functional parameters of the environment (e.g., paging on demand) for executing RAPs result in different orders of instruction interleavings that lead to different results, hence the loss of reliability
The Problem Statement

Enable REMPs to execute rules in RAPs in parallel

Do not violate the separation of concerns in REMPs by requiring programmers to use synchronization lock mechanisms for concurrent accesses to shared resources

Prevent races in parallelized RAPs without explicit using of locking mechanisms by programmers

Choose a better schedule for executing rules that share the same lock objects to improve the overall performance of RAPs
Core Ideas

1) Find all concurrent access to resources from rules where one of the accesses is write.
2) Define synchronizations around these accesses.
3) Impose a complete ordering among all rules that are fired in working memory.
Our Solution - PERLATO

**PErformance and Reliability for ruLe-driven ApplicaTiOns (PERLATO)** connects separate layers or REMPs in a way that enable us to solve the fundamental problem of REMP.

- we obtain a rule execution model from a RAP that approximate different execution scenarios by using the if-then structure of rules by analyzing their antecedents and consequents.
- the obtained rule execution model is used in PERLATO to detect races statically among these rules effectively and efficiently.
- the rule execution model and locking strategies for a given RAP are passed to the REMP engine, so that it can precompute an execution schedule for rules in a RAP to optimize the performance of the RAP.
- **We implemented PERLATO for JBoss Drools, an open-source enterprise-level REMP and we evaluated PERLATO on three RAPs.**
  The results suggest that PERLATO is effective and efficient, since we achieved up to 225% speedup on average without observing any races.
Map fo PERLATO

Massive Parallel Computing

Rule Execution Model

Rule Management Platform

- Knowledge Base
- Reasoning Engine
- Event Processing
- Enterprise Infrastructure

Key Concepts of Rule-based Engineering

- Implicit Rule Invocations
- Rule Independence And Modularity
The Architecture of PERLATO

```
RULES

1. Rules
2. Rule Analyzer
3. Model
4. Schedules
5. Concurrent Conflicts
6. Schedule Generator
7. Lockset Generator
8. Ruleset
9. Ruleset
10. Ruleset
11. Ruleset
12. Ruleset

RAP

REMP
```
Research Questions

- Is PERLATO effective in achieving higher speedups for subject RAPs?
- Is finer granularity locking strategy more effective in obtaining higher speedup for RAPs?
- Is symbiotic scheduling effective in obtaining higher speedup for RAPs?
## Result for Subject RAPs

<table>
<thead>
<tr>
<th>Application</th>
<th>Input Set</th>
<th>Facts</th>
<th>Rules Fired</th>
<th>Approach</th>
<th>Type</th>
<th>Avg</th>
<th>Med</th>
<th>Min</th>
<th>Max</th>
<th>$\sigma^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Seq</td>
<td>32.22</td>
<td>32.08</td>
<td>31.559</td>
<td>35.358</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Par</td>
<td>9.52</td>
<td>9.27</td>
<td>8.267</td>
<td>11.789</td>
<td>0.87</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DROOLS</td>
<td>Rule</td>
<td>13.44</td>
<td>13.6</td>
<td>10.927</td>
<td>14.911</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Atomic</td>
<td>10.85</td>
<td>10.82</td>
<td>10.21</td>
<td>11.867</td>
<td>0.14</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Variable</td>
<td>10.67</td>
<td>10.67</td>
<td>10.21</td>
<td>11.359</td>
<td>0.06</td>
</tr>
<tr>
<td>EEWS</td>
<td>1</td>
<td>246896</td>
<td>1088</td>
<td>PERLATO</td>
<td>Rule</td>
<td>15.86</td>
<td>19.02</td>
<td>15.794</td>
<td>21.549</td>
<td>2.07</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Atomic</td>
<td>15.62</td>
<td>15.58</td>
<td>15.016</td>
<td>16.476</td>
<td>0.11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Variable</td>
<td>15.08</td>
<td>15.02</td>
<td>14.24</td>
<td>16.689</td>
<td>0.21</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>334220</td>
<td>1454</td>
<td>DROOLS</td>
<td>Seq</td>
<td>19.51</td>
<td>19.44</td>
<td>18.906</td>
<td>21.115</td>
<td>0.24</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Par</td>
<td>6.05</td>
<td>6.19</td>
<td>3.947</td>
<td>8.242</td>
<td>1.32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PERLATO</td>
<td>Rule</td>
<td>8.33</td>
<td>8.22</td>
<td>7.025</td>
<td>9.442</td>
<td>0.42</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Atomic</td>
<td>7.24</td>
<td>7.13</td>
<td>6.837</td>
<td>8.259</td>
<td>0.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Variable</td>
<td>6.79</td>
<td>6.76</td>
<td>6.662</td>
<td>7.165</td>
<td>0.02</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>247113</td>
<td>572</td>
<td>DROOLS</td>
<td>Seq</td>
<td>20.4</td>
<td>20.32</td>
<td>20.023</td>
<td>20.97</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Par</td>
<td>19.52</td>
<td>19.52</td>
<td>19.115</td>
<td>20.072</td>
<td>0.07</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PERLATO</td>
<td>Rule</td>
<td>19.71</td>
<td>19.69</td>
<td>19.232</td>
<td>20.657</td>
<td>0.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Atomic</td>
<td>19.62</td>
<td>19.61</td>
<td>19.144</td>
<td>20.227</td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Variable</td>
<td>19.55</td>
<td>19.52</td>
<td>19.077</td>
<td>20.161</td>
<td>0.08</td>
</tr>
<tr>
<td>TAXC</td>
<td>1</td>
<td>16</td>
<td>95</td>
<td>DROOLS</td>
<td>Seq</td>
<td>42.48</td>
<td>42.48</td>
<td>41.553</td>
<td>43.63</td>
<td>0.32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Par</td>
<td>34.27</td>
<td>34.6</td>
<td>31.866</td>
<td>36.019</td>
<td>1.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PERLATO</td>
<td>Rule</td>
<td>36.87</td>
<td>36.83</td>
<td>36.016</td>
<td>37.812</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Atomic</td>
<td>36.86</td>
<td>36.77</td>
<td>36.007</td>
<td>37.775</td>
<td>0.21</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Variable</td>
<td>36.8</td>
<td>36.74</td>
<td>35.972</td>
<td>37.698</td>
<td>0.23</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>22</td>
<td>162</td>
<td>DROOLS</td>
<td>Seq</td>
<td>11.78</td>
<td>11.73</td>
<td>11.549</td>
<td>12.344</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Par</td>
<td>9.85</td>
<td>9.87</td>
<td>9.116</td>
<td>10.314</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PERLATO</td>
<td>Rule</td>
<td>9.93</td>
<td>9.9</td>
<td>9.641</td>
<td>10.431</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Atomic</td>
<td>9.91</td>
<td>9.92</td>
<td>9.145</td>
<td>10.342</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Variable</td>
<td>9.98</td>
<td>9.97</td>
<td>9.309</td>
<td>10.761</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>10</td>
<td>66</td>
<td>DROOLS</td>
<td>Seq</td>
<td>11.78</td>
<td>11.73</td>
<td>11.549</td>
<td>12.344</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Par</td>
<td>9.85</td>
<td>9.87</td>
<td>9.116</td>
<td>10.314</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>PERLATO</td>
<td>Rule</td>
<td>9.93</td>
<td>9.9</td>
<td>9.641</td>
<td>10.431</td>
<td>0.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Atomic</td>
<td>9.91</td>
<td>9.92</td>
<td>9.145</td>
<td>10.342</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Variable</td>
<td>9.98</td>
<td>9.97</td>
<td>9.309</td>
<td>10.761</td>
<td>0.06</td>
</tr>
</tbody>
</table>
Conclusions

We created a novel solution for enhancing performance and reliability of rule-driven applications.

The results suggest that PERLATO is effective, since we achieved over 225% speedup on average.
Email: drmark@uic.edu