Machine Learning applied to Process Scheduling

Benoit Zanotti

jicks@lse.epita.fr
http://www.lse.epita.fr

July 17, 2013
1 Introduction and definitions
  • Machine Learning
  • Process Scheduling
Plan

1. Introduction and definitions
   - Machine Learning
   - Process Scheduling
Definition of Machine Learning

Machine Learning is a field of Computer Science about the construction and study of systems that can learn from data.

Usual organizations of ML algorithms:
- Supervised learning (classification, ...)
- Unsupervised learning (clustering, ...)
- Semi-supervised learning
- ...

We won’t talk really about the theory. But:

- Pretreatment is very important.
- Usually, big tradeoff between speed and efficiency

In Process Scheduling, those factors will be limiting.
Plan

1. Introduction and definitions
   - Machine Learning
   - Process Scheduling

Machine Learning applied to Process Scheduling

Benoit Zanotti

Introduction and definitions

Our target: CFS

What can we do?

Results and analysis

Conclusion
What is Process Scheduling?

Definition

Process Scheduling is the method by which processes are given access to processor time. It is used to achieve multi-tasking.

There are many well-known scheduling algorithms. For example:

- First In, First Out
- Round-Robin (fixed time unit, processes in a circle)
Main concerns

A scheduler has mainly 3 metrics: throughput, latency and fairness. We can simplify them (in practice) by:

- Speed (how much time the scheduler itself uses, number of context-switching, ...)
- Fairness (giving equal CPU time to each process)
- Reactivity (are interactive processes given any advantages?)

A scheduler is complicated. Let’s optimize one using ML!
Plan

2 Our target: CFS
   - Inner workings
   - Advantages/Inconvenients
Plan

2 Our target: CFS
  - Inner workings
  - Advantages/Inconvenients

Introduction and definitions
Our target: CFS
Inner workings
Advantages/Inconvenients
What can we do?
Results and analysis
Conclusion
Inner workings of CFS

- Stands for **Completely Fair Scheduler**
- Scheduler of Linux since 2.6.23
- Just an RB-tree with elements indexed by the runtime of the process.
- Straightforward algorithm: just take the minimum of the tree.

CFS in Linux kernel is actually more complicated (handling Real-Time tasks, nice values, ...)

---

**Introduction and definitions**

Our target: CFS

**Inner workings**

Advantages/Inconveniences

What can we do?

Results and analysis

Conclusion
Why CFS?

- Quite simple and works really well
- Most familiar (I implemented one in mikro)
- Already efficient. I wanted to see what ML could do.
Plan

- Our target: CFS
  - Inner workings
  - Advantages/Inconveniences
Advantages/Inconvenients

- Very simple to understand
- Works really well in general cases
- No real corner cases
- A little light on the handling of interactive processes.
Plan

3. What can we do?
   - ML considerations
   - Applying ML to the CFS
What can we do?
- ML considerations
- Applying ML to the CFS
ML considerations

- Restricted to supervised learning (classification and regression mainly)
- Scheduler must be as fast as possible. Its ML components too.
- Avoiding complex code in the kernel is often a good idea.

→ precomputed model/profile for each processes
→ no complex methods, results will be mitigated
3. What can we do?

- ML considerations
- Applying ML to the CFS
Objective: reducing the number of context switches:
- A process time quantum should ideally not finish (process going to sleep)
- An estimation of the next quantum would help
- Based on the N lasts quantums
- Be careful not to be too unfair

Note: Many other objectives were possible...
Actual implementation

- Proof of Concept
- One using Taylor’s Theorem and one using a classifier
- Need to extract real runtime quantums and to create profiles
• The sequence of quantum s can be seen as a function of the time.
• Taylor’s theorem gives an approximation of a function on a point given its derivatives.
• Discrete derivation is only substraction.

\[ f(x + 1) = f(x) + f'(x - 1) + \frac{f''(x - 1)}{2} \]

This method is simple and fast, but not very precise.
Naive Bayes Classifier using the last 4 quantums:

- It is the best (found) compromise between speed and results
- Parameters and output are range of time, not the actual values
- Based on Bayes’ theorem. Outputs the labels with most probability
- Only 4 multiplications are needed for each label (there is 10 of them).
- Using bit manipulation, we can avoid any conditionals

→ it is fast, but clearly not the most accurate
Plan

Introduction and definitions
Our target: CFS
What can we do?
Results and analysis
Methodology and results
Analysis

Results and analysis
- perf and Linsched
- Methodology and results
- Analysis
4 Results and analysis
- perf and Linsched
- Methodology and results
- Analysis
perf

- Performance analysis tools for Linux
- Based on kernel-based performance counters
- Can be used to extract many scheduling stats
Linsched

- Linux Scheduler Simulator (in userland...)
- Easy to use (cycle of development, debugging, ...) and fast
- Can replay records from *perf*
- Hard to quantify how much time is used by the scheduler
Plan

4 Results and analysis
  • perf and Linsched
  • Methodology and results
  • Analysis
Methodology of the tests

- Use *perf* to extract records and datasets
- Use *WEKA* to compute profiles for each process
- Test using vanilla/modified linsched to see the gain
- Time the tests of vanilla/modified linsched to estimate how costly each method is
Results

Results of the simulation (without scheduler time)

<table>
<thead>
<tr>
<th>Method</th>
<th>Time Used (base=100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>100</td>
</tr>
<tr>
<td>Extrapolation</td>
<td>98</td>
</tr>
<tr>
<td>Classifier</td>
<td>95</td>
</tr>
</tbody>
</table>
Results

Results of the simulation (with scheduler time)

Time used (base=100)

<table>
<thead>
<tr>
<th>Method</th>
<th>Time Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla</td>
<td>100</td>
</tr>
<tr>
<td>Extrapolation</td>
<td>102</td>
</tr>
<tr>
<td>Classifier</td>
<td>98</td>
</tr>
</tbody>
</table>

Machine Learning applied to Process Scheduling

Benoît Zanotti

Introduction and definitions

Our target: CFS

What can we do?

Results and analysis

perf and Linsched

Methodology and results

Analysis

Conclusion
4 Results and analysis
- perf and Linsched
- Methodology and results
- Analysis
Analysis

- CFS is already quite good
- ML results are positive but very limited
- More complex pretreatment/ML techniques would yield better results... at which cost?
Conclusion
Conclusion

- It was only **one** idea on **one** objective.
- Using ML in scheduling is hard, because of the speed/results tradeoff
- Difficulties for a real kernel integration (passing the models, limiting abuses, ...)
- Basic rule in scheduling: "Simpler is Better"
- Another idea: run a (kernel ?) process every X hours to compute new profiles...
Questions?