Improving Backfilling by using Machine Learning to Predict Running Times in SLURM

David Glesser

2015-11-19
Improving Job Scheduling by using Machine Learning

Improving Backfilling by using Machine Learning to Predict Running Times

• By Eric Gaussier, David Glesser, Valentin Reis, Denis Trystram

• Presented this morning in the Resource Management session
Improving Job Scheduling by using Machine Learning

- Machine Learning algorithms can learn odd patterns
- SLURM use a backfilling algorithm
- the running time given by the user is used for scheduling, as the actual running time is not known
- The value used is very important

- better running time estimation => better performances

  ► Predict the running time to improve the scheduling
Improving Job Scheduling by using Machine Learning

We select a Machine Learning algorithm that:

- Uses classic job parameters as input parameters
- Works online (to adapt to new behaviors)
- Uses past knowledge of each user (as each user has its own behaviour)
- Robusts to noise (parameters are given by humans, jobs can segfault...)


Improving Job Scheduling by using Machine Learning

- We test 128 different algorithms on 6 logs (from the Feitelson Workload Archive) on the Pyss simulator

- A leave-one-out cross validation product give us the best algo that we called $E$-Loss:
  - Online linear regression model
  - Predict that a running time is more than the actual value cost more to the model
  - When we under estimate a running time, we add a fixed value (1min, 5min, 15 min, 30 min…) 
  - When we backfill jobs we sort them by shortest first
Improving Job Scheduling by using Machine Learning

**Table 4: Workload logs used in the simulations.**

<table>
<thead>
<tr>
<th>Name</th>
<th>Year</th>
<th># CPUs</th>
<th># Jobs</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>KTH-SP2</td>
<td>1996</td>
<td>100</td>
<td>28k</td>
<td>11 Months</td>
</tr>
<tr>
<td>CTC-SP2</td>
<td>1996</td>
<td>338</td>
<td>77k</td>
<td>11 Months</td>
</tr>
<tr>
<td>SDSC-SP2</td>
<td>2000</td>
<td>128</td>
<td>59k</td>
<td>24 Months</td>
</tr>
<tr>
<td>SDSC-BLUE</td>
<td>2003</td>
<td>1,152</td>
<td>243k</td>
<td>32 Months</td>
</tr>
<tr>
<td>Curie</td>
<td>2012</td>
<td>80,640</td>
<td>312k</td>
<td>3 Months</td>
</tr>
<tr>
<td>Metacentrum</td>
<td>2013</td>
<td>3,356</td>
<td>495k</td>
<td>6 Months</td>
</tr>
</tbody>
</table>
## Improving Job Scheduling by using Machine Learning

### Results on the average Stretch ($\frac{\text{real running time} \times \text{waiting time}}{\text{real running time}}$)

<table>
<thead>
<tr>
<th>Log</th>
<th>Our algorithm</th>
<th>Backfill</th>
<th>SoA</th>
</tr>
</thead>
<tbody>
<tr>
<td>KTH-SP2</td>
<td>51.4 (44%)</td>
<td>92.6</td>
<td>63.5 (31%)</td>
</tr>
<tr>
<td>CTC-SP2</td>
<td>20.5 (59%)</td>
<td>49.6</td>
<td>85.8 (-72%)</td>
</tr>
<tr>
<td>SDSC-SP2</td>
<td>75.0 (15%)</td>
<td>87.9</td>
<td>79.4 (10%)</td>
</tr>
<tr>
<td>SDSC-BLUE</td>
<td>34.7 (05%)</td>
<td>36.5</td>
<td>21.0 (42%)</td>
</tr>
<tr>
<td>Curie</td>
<td>27.9 (86%)</td>
<td>202.1</td>
<td>193.5 (04%)</td>
</tr>
<tr>
<td>Metacentrum</td>
<td>84.2 (14%)</td>
<td>97.6</td>
<td>87.2 (11%)</td>
</tr>
</tbody>
</table>

*Note: Values in parentheses indicate the improvement or degradation compared to the best known solution (SoA).*
Improving Job Scheduling by using Machine Learning

Figure 4: Experimental cumulative distribution functions of prediction errors obtained using the Curie log.

Our algorithm under-estimate more than over-estimate. This make the backfilling more aggressive (more jobs will be backfilled).
Improving Job Scheduling by using Machine Learning

Figure 4: Experimental cumulative distribution functions of prediction errors obtained using the Curie log.

Our algorithm gives the best scheduling performance, but it is not the best at predicting running times!
Conclusion

- Backfilling performance can be improved by changing the estimation of running times
- More precise estimations of running times does not mean better performances
- Scheduling performances can be increased using basic Machine Learning algorithms
Implementation in SLURM

- Computation time?
  - $O((\#\text{features})^2)$ for learning and prediction
  - $\#\text{features}=20$ in the paper

- No support for time reservations
  - Use of the user estimation for nodes that are reserved in the future

- No estimation of the starting time of the first job
  - Compute an estimation? Don't give it?

- Impossible to evaluate the implementation
  - Use a Slurm simulator
A Slurm simulator?
Yet an another SLURM simulator

• Previous works:

• Official Slurm simulator: code is changed, it has to be updated each time a new Slurm is out.

• Platform emulation: run *sleeps* instead of actual jobs, multiple slurmd per physical node (to emulate bigger cluster than you have access to)
Yet an another SLURM simulator

Virtual Machines
+ perfect behaviour
- heavy and slow
+ No modifications to SLURM

Classic simulators
- no guarantee on the behaviour
+ extra light
- Modifications of SLURM
Introducing Simunix, an UNIX simulator

- We implement the "UNIX" API: pthreads, pthread_mutex, gettimeofday, sleep, send, recv…

- Use Simgrid framework

  - We can run an unmodified slurm on a simulated cluster
Yet an another SLURM simulator

Virtual Machines
+ perfect behaviour
- heavy and slow
+ No modifications to SLURM

Simunix
+ close behaviour
+ light
+ No modifications to SLURM

Classic simulators
- no guarantee on the behaviour
+ extra light
- Modifications of SLURM
Yet an another SLURM simulator

How to force a binary to use our libraries?
• Change how linking is done!

• The Linux linker load from the system and LD_PRELOAD the needed shared libraries
• It fills the GOT (Global Object Table) with the address of each functions of each libraries
• The compiler compile
  
  ```
  sleep(10);
  ```

  to

  ```
  GOT[“sleep@libc“](10);
  ```

  (Of course, it's not exactly like this, if you have more question RTFM of the ELF format)
Yet an another SLURM simulator

How to force a binary to use our libraries?
• Change how linking is done!

• At runtime, simunix rewrite the GOT
  • Of the selected binary/libraries
  • Not on the simunix library nor the Simgrid library!
  • Addresses in the GOT are replace by our own functions:

    GOT["sleep@libc"] = &simunix_sleep;
    GOT["time@libc"] = &simunix_time;
    ...

Yet another SLURM simulator

Simgrid

- a framework to design simulators of distributed applications

- Supports:
  - advanced network models
  - energy consumption models
  - I/O models

- Actively developed

- Good practice: they (in)validate their simulator (they explicitly give the strengths and weaknesses of their models by testing them and compared them to real runs!)
Yet another SLURM simulator

How this work?

• Each intercepted calls communicate to an independent maestro process
Yet another SLURM simulator

Current works

• Optimize to simulate 1 year in a reasonable amount of time
• Support more Simgrid features:
  • run simulated apps not just a sleep (network contention...)
  • DVFS and energy
• Try out with other schedulers (every Linux software is compatible!)
• Publish!
Global conclusion

- We can improve the scheduling using machine learning
- Some more works need to be done to support this in Slurm
- Other learning algorithm should also be considered, like Learning2Rank's algorithms
Thanks