Understanding how system behave

- The behavior of computer systems
  - Oftentimes nontrivial & hard to understand

- When systems a real & operational
  - Even if they are (seemingly) very simple, they might lead to puzzling, nontrivial, hard to understand results
Understanding how system behave

- The fact that humans constructed the system
  - oftentimes doesn’t help

- What to do?
  - Apply the scientific method

1. Ask question
2. Construct hypothesis
3. Test with experiment
4. Hypothesis false?
Understanding how system behave

• Example
  - An operating system of a supercomputer

• Scheduling subsystem is simple enough
  - Not a whole lot more than FCFS

• Yet results are surprising & counterintuitive
  - We’ll apply the scientific method
Using Inaccurate Estimates Accurately

Dan Tsafrir, 17/5/2016 (3h), + (1h)
Based on keynote talk from JSSPP 2010
(Job Scheduling Strategies for Parallel Processing)
Typical supercomputer setting

- **Machine**
  - Comprised of 100s to 10,000s of nodes
  - Operational for 3-4-5 years
- **Workload**
  - 100s of users (physicist, chemists, ...)
  - Collectively submitting 10,000s to 100,000s of jobs
  - Jobs run seconds to 10s of hours
Supercomputer jobs

- Can be serial, but typically parallel
  - “Parallel” in the traditional sense
  - Crucial to job’s threads simultaneously
- Major job attributes:
  - User ID
  - Arrival time (= submission time)
  - Runtime
  - Size (num of processors required by job)
  - ...

Job $\leftrightarrow$ Rectangle

- Jobs can be “wide” or “narrow” (size)
- Jobs can be “short” or “long” (runtime)
Activity logs (or traces)

- Activity on machine is logged into a trace file, which may look like so:

<table>
<thead>
<tr>
<th>job ID</th>
<th>arrival time</th>
<th>size [CPU#]</th>
<th>runtime</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Aug 24, 12:00:01</td>
<td>2</td>
<td>00:15:37</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Aug 24, 12:05:37</td>
<td>128</td>
<td>01:50:01</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Aug 24, 13:25:20</td>
<td>49</td>
<td>18:00:00</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Parallel Workload Archive

[http://www.cs.huji.ac.il/labs/parallel/workload/logs.html]

<table>
<thead>
<tr>
<th>log</th>
<th>abbreviation</th>
<th>duration [months]</th>
<th>CPU #</th>
<th>jobs</th>
<th>utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cornell Theory Center</td>
<td>CTC</td>
<td>11</td>
<td>512</td>
<td>77,222</td>
<td>56%</td>
</tr>
<tr>
<td>Swedish Royal Institute</td>
<td>KTH</td>
<td>11</td>
<td>100</td>
<td>28,490</td>
<td>69%</td>
</tr>
<tr>
<td>San Diego SC (SP2)</td>
<td>SDSC</td>
<td>24</td>
<td>128</td>
<td>59,725</td>
<td>84%</td>
</tr>
<tr>
<td>San Diego SC (Blue Horizon)</td>
<td>BLUE</td>
<td>32</td>
<td>1152</td>
<td>243,314</td>
<td>76%</td>
</tr>
</tbody>
</table>
Context: Job scheduling

- Submit => wait => run

- FCFS
  - Causes fragmentation

- The “backfilling” optimization
  - Can jump over 1st queued job if not delaying it
Backfilling: pros

- **Simple and easy**
  - For users to understand ("it’s FCFS with...")
  - For developers to implement

- **Batch**
  - Scientific apps. often tailored to use entire memory
  - Insures co-scheduling

- **Significantly improves performance**
  - Utilization (from 40-60% to 70%), throughput, response time,...

- **Comparable to more sophisticated alternatives**
  - Involving preemption & migration
Pros' consequences

- Backfilling is very popular in production systems

<table>
<thead>
<tr>
<th>Vendor</th>
<th>product</th>
<th>free</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td>LoadLeveler</td>
<td></td>
</tr>
<tr>
<td>Cluster Resources</td>
<td>Moab</td>
<td>Maui</td>
</tr>
<tr>
<td>Platform</td>
<td>LSF</td>
<td></td>
</tr>
<tr>
<td>Altair</td>
<td>PBS-Pro</td>
<td>OpenPBS</td>
</tr>
<tr>
<td>Sun (Oracle)</td>
<td>GridEngine</td>
<td></td>
</tr>
</tbody>
</table>

- 60% of the "top-50" (in top500) use backfilling
- Due to popularity, the focus of many research efforts (dozens of variations, dozens of papers)
Backfilling: the price

- Need to know jobs’ runtimes a-priori
- Thus, users must provide estimates of how long their jobs will run
- (Jobs attempting to exceed their estimates are killed)

FCFS + Backfilling = “EASY”
Impact of user inaccuracy?

- Many papers address this question
  - ~15 years of research (earliest from 1996)
  - Latest I know won IPDPS’10 best paper
    "Analyzing & adjusting user estimates to improve job scheduling on Blue Gene/P" by Tang, Desai, Buettner, & Lan
- Said papers’ results almost always include misleading or erroneous parts
  - A major part of this talk
Methodology: how to check?

• Simulation
• Simulator’s input: activity log (can tweak)

<table>
<thead>
<tr>
<th>job ID</th>
<th>arrival time</th>
<th>size [CPU#]</th>
<th>runtime</th>
<th>estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Aug 24, 12:00:01</td>
<td>2</td>
<td>00:15:37</td>
<td>00:30:00</td>
</tr>
<tr>
<td>2</td>
<td>Aug 24, 12:05:37</td>
<td>128</td>
<td>01:50:01</td>
<td>02:00:00</td>
</tr>
<tr>
<td>3</td>
<td>Aug 24, 13:25:20</td>
<td>49</td>
<td>18:00:00</td>
<td>18:00:00</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

• Simulator’s output: performance (typically average wait time & slowdown)
Performance metrics

• Wait time
• Slowdown
  – (wait_time + runtime) / runtime
• Actually, bounded slowdown
  – Minimal runtime is bounded
  – Typically set to be, e.g., 10 seconds
    • runtime = max(runtime, 10sec)
• Average across all jobs
Methodology: how to artificially make estimates worse/better?

- **Choose a badness factor “F”**
  - Smaller $F$ => better accuracy
  - Bigger $F$ => worse accuracy

- **Badness model of the said IPDPS’10 paper**
  - $0 \leq F \leq 1$
  - $R = \text{runtime of job}$
  - $E = \text{user estimate for job}$
  - $\text{artificial\_estimate} = (1 - F) \cdot R + F \cdot E$
Methodology: how to artificially make estimates worse/better?

- Typically, papers use the “F-model”
  - $1 \leq F$
  - $R = \text{runtime of job}$
  - artificial estimate = uniform in $[R, R \cdot F]$

- Sometime, the “deterministic F-model”
  - artificial estimate = $R \cdot F$
Normalizing: $F = f+1$

- Typically, papers use the “f-model”
  - $1 \leq F \implies 0 \leq f$
  - $R = \text{runtime of job}$
  - artificial_estimate = uniform in $[R, R \cdot (f+1)]$

- Sometime, the “deterministic f-model”
  - artificial_estimate = $R \cdot F = R \cdot (f+1)$
Claim 1: Inaccuracy improves performance

<table>
<thead>
<tr>
<th>wait [minutes]</th>
<th>SDSC</th>
<th>CTC</th>
<th>KTH</th>
<th>BLUE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>320</td>
<td>23</td>
<td>115</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>20</td>
<td>100</td>
<td>115</td>
</tr>
<tr>
<td></td>
<td>280</td>
<td>17</td>
<td>85</td>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bounded slowdown</th>
<th>SDSC</th>
<th>CTC</th>
<th>KTH</th>
<th>BLUE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>80</td>
<td>5</td>
<td>75</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>4</td>
<td>70</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>70</td>
<td>3</td>
<td>65</td>
<td>25</td>
</tr>
</tbody>
</table>

accurate (F=1)
doubled (F=2, deterministic)
Explanation 1: The “holes argument”

S. Chiang, A. Arpaci-Dusseau, and M. Vernon [JSSPP, 2002]:

“...when multiplying estimates by two, jobs with long runtimes can have very large overestimation, which leaves larger holes for backfilling shorter jobs. As a result, average slowdown and wait may be lower.”
Claim 2: Performance is independent of $f$

**BLUE - wait**

- random (red)
- deterministic (green)

**BLUE - slowdown**

- random (red)
- deterministic (green)
Y. Zhang, H. Franke, J. Moreira, and A. Sivasubramaniam [IPDPS, 2000]:

"On average, overestimation impacts both the jobs that are running and the jobs that are waiting... the probability of finding a backfilling candidate effectively does not change with the overestimation."
The robustness claim

D. England, J. Weissman, and J. Sadagopan [HPDC, 2005]:

"...Our results support those of previous work and also indicate that backfilling is robust to inaccurate estimates in general.

It seem that, with respect to backfilling, what the scheduler doesn’t know won’t hurt it."
The insensitivity claim

W. Tang, N. Desai, D. Buettner, Z. Lan
[the IPDPS’2010 best paper]:

“Our analysis indicates that FCFS [with backfilling] is not sensitive to user runtime estimates.”
Intermediate summary

• Two observations regarding inaccuracy impact
  - Inaccuracy improves performance
  - Inaccuracy doesn’t affect performance

• Two contradictory explanations
  - The holes argument
  - The balance argument

• One mystery
  - Improved accuracy should result in better packing
  - How come the opposite is true?
Using mean & confidence exposes a clear trend

**BLUE:** V-shape (the norm)

**CTC:** L-shape

**f** (badness factor)

**wait [minutes]**

- **f=0:** deterministic
- random (mean)
- random (90% confidence)

27/54
The descending part

BLUE: V-shape

CTC: L-shape
Which is correct? Balance? Holes?

**certainly more backfilling activity**
“Heel & toe” dynamics

Assume $E = 2R$ (that is, $F=2$; deterministic)
The ascending part

BLUE: V-shape

CTC: L-shape
Bigger $f$ => more long jobs masquerade as short & vise versa

The probability the scheduler is erroneously told $j_1$ is longer than $j_2$, is monotonically increasing with $f$
The longer jobs enjoy the backfilling at the expense of the shorter jobs. (Results for BLUE)

Bigger $f \Rightarrow$ wider holes $\Rightarrow$ longer jobs enjoy backfilling

The diagrams show the runtime of backfilled and non-backfilled jobs, as well as the SJFness (% of jobs) for different values of $f$. The graphs illustrate the trend that as $f$ increases, the runtime of backfilled jobs decreases, while the runtime of non-backfilled jobs increases. The SJFness values also show a decrease with increasing $f$, indicating a higher scheduling efficiency.

- $f=0$: Random (mean) scheduling
- $f=0$: Random (90% confidence) scheduling
- $f=0$: Deterministic scheduling
Why is CTC different?

BLUE: V-shape

CTC: L-shape
The role of burstiness

Weekly temporal load

SDSC (84%)

CTC (56%)

KTH (69%)

BLUE (76%)

CTC336 (84%)

Manipulated time [months]
CTC-336

shrink size by half

original workload

shrink arrivals by half

Wait [minutes]

slowdown

f (badness factor)
Incidentally, as it turns out...

The log records wait times of the original jobs (real wait time, not simulated); when plotting utilization using this data, we get that...

with the exception of a few outliers at most 338 nodes are used in CTC, which means the “512” in the log is likely a bug.
Big picture: we uncovered a fairness/performance tradeoff

- “Unfairness” is the avg. delay of jobs beyond their “Fair Start Time”
- Jobs that start before that time contribute zero to the avg.
- **Multiplying by a factor simply means trading off fairness for performance**
Not just a theoretical result...

- The more accurate the value we multiply, the better the performance & fairness
- Increased accuracy actually improves performance
- F-model is crappy...
That’s all very interesting, but…

• **Question:**
  - Did what we learn help us figure out how to study the impact of **real** user inaccuracy?

• **Answer:**
  - No
  - The fact researchers consistently use some methodology does not make it right
Here's how real estimates look like

1. Modality => many “identical” jobs => bad scheduling info.
2. ‘Max’ is especially popular => such jobs never backfill !
   => Increased inaccuracy means increased modality
Max overlooked by prior art

W. Cirne & F. Berman [WWC, 2001]:

"it is reasonable to imagine the longer the execution time; the more likely it is that the job succeeds"
The alternative to runtime estimates
The alternative

- History-based system-generated predictions
- Many have shown predictions are more accurate
- Yet, predictions have never been used in production systems...
Failure due to

1. “Inaccuracy helps” misconception
   => refuted
2. Overhead & complexity of predictors (rough-set theory, instance-based learning, genetic algorithm, data mining)
   => we’ll use a trivial predictor
3. Technical difficulty: underprediction (jobs are killed when reaching their estimate)
   => we’ll overcome the difficulty
1st Attempt: Eliminate estimates’ dual role

- Each estimate has two rolls
  1) approximates the runtime
  2) bounds the runtime (= kill-time)
- New backfilling algorithm
  1) use estimates as kill-time only
  2) use predictions for everything else
  3) if a job outlives its prediction, let it be
- New prediction algorithm
  prediction = avg. runtime of last 2 jobs by user
Outcome: colossal failure

<table>
<thead>
<tr>
<th>log</th>
<th>change [%]</th>
<th>wait</th>
<th>slowdown</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDSC</td>
<td>+109</td>
<td>+136</td>
<td>+70</td>
<td></td>
</tr>
<tr>
<td>CTC</td>
<td>+38</td>
<td>+53</td>
<td>+44</td>
<td></td>
</tr>
<tr>
<td>KTH</td>
<td>+748</td>
<td>+729</td>
<td>+4</td>
<td></td>
</tr>
<tr>
<td>BLUE</td>
<td>+920</td>
<td>+1141</td>
<td>+78</td>
<td></td>
</tr>
<tr>
<td>avg.</td>
<td>+454</td>
<td>+515</td>
<td>+49</td>
<td></td>
</tr>
</tbody>
</table>

- Positive values are good for accuracy
- But, alas, bad for wait time and slowdown
Failure because...

scheduler’s optimistic view

what may happen

reservation based on first prediction

time

termination time
2\textsuperscript{nd} attempt: Add prediction correction

what may happen

reservation based on first prediction
termination time

reservation based on first prediction
corrected prediction

with corrected prediction

nodes

1

underprediction

2

3

4

3

4

1

2
Outcome: much, much better

<table>
<thead>
<tr>
<th>log</th>
<th>change [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>wait</td>
</tr>
<tr>
<td>SDSC</td>
<td>-10</td>
</tr>
<tr>
<td>CTC</td>
<td>-26</td>
</tr>
<tr>
<td>KTH</td>
<td>-16</td>
</tr>
<tr>
<td>BLUE</td>
<td>-21</td>
</tr>
<tr>
<td>avg.</td>
<td>-18</td>
</tr>
</tbody>
</table>
3rd attempt: The lesson from “heel & toe”

- Basic (reservation) order
  => FCFS, as usual

- Backfill order
  => shortest prediction (SJBF)

- More sensible than multiplying stunts
  1) directly obtains improvement, and
  2) reservation are more accurate
Outcome: better still

<table>
<thead>
<tr>
<th>log</th>
<th>change [%]</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>wait</td>
<td>slowdown</td>
<td>accuracy</td>
</tr>
<tr>
<td>SDSC</td>
<td>-10</td>
<td>-29</td>
<td>+87</td>
</tr>
<tr>
<td>CTC</td>
<td>-33</td>
<td>-37</td>
<td>+61</td>
</tr>
<tr>
<td>KTH</td>
<td>-17</td>
<td>-36</td>
<td>+28</td>
</tr>
<tr>
<td>BLUE</td>
<td>-33</td>
<td>-47</td>
<td>+102</td>
</tr>
<tr>
<td>avg.</td>
<td>-23</td>
<td>-37</td>
<td>+70</td>
</tr>
</tbody>
</table>
The virtues of recency (we've used only last jobs by user)
Conclusions

• **If you want to study inaccuracy**
  - Don’t use the real runtimes to generate your artificial estimate; that’s erroneous
  - Instead, make more jobs use the same estimate, especially the max
  - You will find that better accuracy always improves things (better performance, less unfairness)

• **If you want to improve estimates**
  - You should employ dynamic correction
  - Recency is more important than similarity
Conclusions

- Two type of “inaccuracy”:
  
<table>
<thead>
<tr>
<th>type</th>
<th>property of</th>
<th>nature</th>
<th>performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>real users</td>
<td>modal and favors 'max'</td>
<td>worsened</td>
<td></td>
</tr>
<tr>
<td>artificial (f)</td>
<td>schedulers</td>
<td>promotes heel &amp; toe</td>
<td>improves</td>
</tr>
</tbody>
</table>

- Multiplying = property of the scheduler
  - trades off fairness for performance

- “Inaccuracy helps” or “doesn’t matter” = error
  - f-model inadequate [IISWC, 2006]

- Need a realistic model
  - “Modeling user runtime estimates” [JSSPP, 2005]

- Should improve estimates
  - “Backfilling with system predictions” [TPDS, 2007]
The truncated f-model:

\[ \min((f+1) \cdot R, 'Max') \]

- SDSC - slowdown
- CTC - wait
- real estimates
- random (mean)
- random (90% confidence)
- deterministic
C. Bailey Lee & A. Snavely [IJHPCA, 2006]:

"The key point is that Mu’alem and Feitelson result only applies to the specific algorithms they studied and it is necessary to re-prove (or disprove) their results for each new algorithm individually."

- We contend the situation is not so bleak:
  - As long as reservation are allocated to promote fairness
  - Multiplying and heel & toe would push them away