Scheduling Workflows on Platforms where Energy Matters

Presented by
Rizos Sakellariou
but primarily thanks to the work of Ilia Pietri
(and many other people who helped over the years)
Work published in ICPP/PASA2014, UCC2014
Scheduling...

**Schedule**: “A plan for performing work or achieving an objective, specifying the order and allotted time for each part”
(http://www.thefreedictionary.com)

**Scheduling**: the constituent parts:

- Work
- Resources
- Objective(s)
In this talk...

• **Work**
  – Scientific workflows
    • DAG (nodes: work, edges: communication)

• **Resources**
  – Energy-constrained (DVFS enabled) resources

• **Objective**
  – Complete execution of the workflow by a certain deadline on a number of resources
  – Minimize overall energy consumption or the cost associated with it
DVFS-enabled resources

• “Dynamic frequency scaling (also known as CPU throttling) is a technique in computer architecture whereby the frequency of a microprocessor can be automatically adjusted on the fly…”

• Running at a lower frequency means we consume less power

• \textit{Energy} = \textit{Power} \times \textit{Time}
Energy is a significant cost

- Cloud providers want to minimize their running costs
- We found at least one provider offering different prices for a range of different CPU frequencies.
  - www.elastichosts.co.uk
- 391 different frequencies
  - VMs from 500MHz to 20000MHz
  - Price from £4.32 to £172.80
The question

How can we make use of the lower costs to use lower CPU frequencies and techniques such as CPU throttling when running workflows?
A DAG, a schedule, and an old idea
Characterize the Schedule

- **Spare time** indicates the maximum time that a node, $i$, may delay without affecting the start time of an immediate successor, $j$.
  
  - A node $i$ with an immediate successor $j$ on the DAG: $spare(i,j) = Start\_Time(j) - Data\_Arrival\_Time(i,j)$
  
  - A node $i$ with an immediate successor $j$ on the same machine: $spare(i,j) = Start\_Time(j) - Finish\_Time(i)$
  
  - The minimum of the above for all successors of task $i$ is the: **Spare time** of task $i$.

---

Example

DAT(4,7)=40.5, ST(7)=45.5; hence, spare(4,7) = 5
FT(3)=28, ST(5)=29.5; hence, spare(3,5) = 1.5

DAT: Data_Arrival_Time, ST: Start_Time, FT: Finish_Time
Characterize the schedule (cont.)

• **Slack** indicates the maximum time that a node, \(i\), may delay without affecting the overall makespan.
  
  \[ \text{Slack}(i) = \min(\text{slack}(j) + \text{spare}(i,j)) \], for all successor nodes \(j\) (both on the DAG and the machine)
The idea

- Given a schedule (mapping of tasks onto machines)
- Given that (according to the schedule) many tasks will always have some slack
  - Why don’t we try to lower the frequency of the tasks with a slack so that they run up to the slack (or they use as much as possible)?
    - Clearly, this will not affect overall makespan

What is the catch here?
Lowering frequency does not mean we save energy!

- Running at a lower frequency will require less power, but it will take longer!
- Remember: energy = power \times \text{time}
Thanks to Thomas Rauber
(presentation at the 9th Scheduling for large-scale systems workshop, Lyon, July 2014)
In addition...

- The workflow (DAG) is a collection of tasks
- We need to take into account the energy vs frequency behaviour of each task and overall (for the whole workflow)
- Different tasks will exhibit different behaviour
- If we try to apply frequency scaling for one task we have to pay some cost for switching frequency (small, but...)

The idea (2)

- Assuming that we need to meet a deadline and minimize energy:
  - 1. Start with a schedule running at highest frequency (can be easily obtained with HEFT, etc)
  - 2. Identify the most profitable in terms of energy reduction tasks (beyond some threshold)
  - 3. Lower to the next available frequency
  - 4. Assess the impact to the whole workflow (DAG)
  - 5. Go to 2 as long as there is overall energy reduction
  - 6. Cleanup and finish.

(Energy-aware stepwise frequency scaling – ESFS)
The intuition

- Reduce frequency by one step: (i) trying to make sure that what may be the local optimum for every task (in the U-curve) is not exceeded, and (ii) assessing the overall energy consumption for the workflow.
The models

• Power:

\[ P_f = P_{\text{base}} + P_{\text{dif}} (f - f_{\text{base}})^3 / f_{\text{base}} \]

(Pierson & Casanova, Euro-Par 2011)

• Task execution time:

\[ \text{Runtime} = (1 + \beta (f_{\text{max}} / f - 1)) \times \text{runtime}_{f_{\text{max}}} \]

(Etinski, Corbalan, Labarta, Valero, JPDC 2012)
Evaluation

• Baseline algorithms
  – EES (from CCGRID12)
  – HEFT

• Processor characteristics
  • $P_{\text{base}} = 152\text{W}$
  • $P_{\text{dif}} = 15.39\text{W}$
  • $P_{\text{idle}} = 60\% P_{\text{fmax}}$
  • Threshold: 0.01%

• Data from 3 workflows, 100 tasks each
  – LIGO
  – SIPHT
  – Montage

Full results in:
Discussion of results

- Different workflows exhibit a different behaviour
- The adaptive approach can produce energy savings without missing the deadline
- Simulation results need to be verified with real experiments
- The outcome is sensitive to the parameters used in the energy model. Some may be difficult to estimate / others change depending on the processor, etc.
Some useful comments

• It is quite easy now to measure the power consumption of real computers (Intel Node Manager, SmartWatt, etc.). Simulation alone provides weak evidence for saving energy since there are too many variables that are not accounted for in simulation.

• Focus only on processor power, which is typically less than 50% of the power in HPC servers. It is not clear if reducing processor power saves energy at the server or data center level. What about memory, fans, etc?

• Modern multi-core processors can achieve close to zero watts for a core when the core is not utilized and put in a low-power state. (as opposed to using 60% of peak for the chip idle power)

• Power consumption in a chip and in a server changes depending on the workload run, even at steady 100% utilization. Power consumption is therefore not constant for a particular clock frequency.

• The true value of beta may change for each workload
Follow-on work

• In the previous work we assumed that the objective was to minimize overall energy consumption. We assumed that we could change frequency for every task.

• In reality, we may need to choose one CPU frequency for every different machine/core we use (frequency scaling per resource not per task)

• Recall the pricing model from elastichosts.
The problem
(motivated by what elastichosts offer)

• 391 different frequencies
  – VMs from 500MHz to 20000MHz
  – Price from £4.32 to £172.80

• The connection between price and frequency is linear:
  – Cost for a certain frequency $f$ is
    $$A + B \times \left( \frac{f - f_{\text{min}}}{f_{\text{min}}} \right)$$
    where A and B are constants.

• Frequency determines runtime and as frequency relates to cost we can try to solve the problem of choosing the right frequencies to complete within a given runtime and budget
The algorithm

• Assuming that we need to meet a deadline and minimize cost:
  – 1. Start with a schedule running at highest frequency (can be easily obtained with HEFT, etc)
  – 2. Identify what resources give cost savings for the next available lower frequency
  – 3. Update the plan using the next available lower frequency for these resources
  – 4. Accept new plan if it gives lower cost and goto 2
  – 5. Finish
    (Cost-based stepwise frequency scaling – CSFS)
The Algorithm

Algorithm 1
Cost-based Stepwise Frequency Selection - CSFS.

Require: \( w \): workflow, \( curPlan \): HEFT plan, \( deadline \): user deadline

1: procedure SPEEDADJUSTMENT\((w, curPlan)\)
2: \[ \text{while } freq > f_{\text{min}} \text{ do} \] \( \triangleright \) Starting with \( freq = f_{\text{max}} \)
3: \[ \text{currentCost} : \text{cost of } curPlan \] \( \triangleright \) Eq. 2 for all resources and time
4: \[ freq = freqStep \] \( \triangleright \) next available lower frequency
5: \[ newPlan = curPlan \]
6: \[ \text{resourcesList} : \forall r \in newPlan \] \( \triangleright \) candidate resources
7: \[ \text{while } \text{resourcesList not empty do} \]
8: \[ \text{for } \forall r \in \text{newPlan do} \]
9: \[ \text{Compute } costSavings_r \] \( \triangleright 0 \) when deadline is exceeded
10: \[ \text{end for} \]
11: \[ \text{resourcesList} = \forall r \in \text{newPlan: costSavings}_r > 0 \]
12: \[ \text{Remove } r \in \text{resourcesList with largest costSavings}_r \]
13: Update task runtimes for each task \( t \in r \) using Eq. 1
14: Update start and finish times of all the tasks in the plan (\( \text{newPlan} \))
15: \[ \text{end while} \]
16: \[ \text{newCost} : \text{cost of } \text{newPlan} \]
17: \[ \text{if } \text{newCost} >= \text{currentCost then Reject } \text{newPlan} \text{ and break} \]
18: \[ \text{end if} \]
19: \[ \text{Accept plan } (curPlan = \text{newPlan}) \]
20: \[ \text{end while} \]
21: end procedure
Montage with 1000 tasks

Conclusion

- Energy-aware scheduling requires a good understanding of underlying energy-related aspects
- It appears that energy/cost savings may not be particularly high (unless we compromise performance)
- What happens in practice is not the same with simulation
- Different tasks have a different behaviour which is not well understood. In other words, frequency scaling does not affect all tasks the same.
- Per task frequency scaling is more complicated but doesn’t seem to result in significantly better results compared to per resource frequency scaling.
- Determining a good/effective/efficient combination of resources (running at different frequencies) is difficult!

www.sucreproject.eu