A run-time generic decision framework for power and performance management on mobile devices

Martin Peres
LaBRI, University of Bordeaux
Talence, France
Email: martin.peres@labri.fr

Mohamed Aymen Chalouf
IRISA, University of Rennes 1
Lannion, France
Email: mohamed-aymen.chalouf@irisa.fr

Francine Krief
LaBRI, University of Bordeaux
Talence, France
Email: francine.krief@labri.fr

***Abstract*—Nowadays, mobile user terminals are usually composed of multiple network interfaces and multiple processors. This means there are usually several independent ways of satisfying the immediate Quality of Service (QoS) expected by the user. Dynamic decision engines are used for selecting the most power-efficient interface and performance states in order to satisfy the QoS while increasing the battery life. Current decision-engines are not generic and need to be mostly re-written when a new processor or radio comes out. This is because there are no generic decision-making framework for real-time power and performance management that would allow creating re-usable bricks that could be shared by multiple decision engines. In this paper, we propose such a framework to ease the implementation of a run-time decision making with real-time requirements and a low memory/CPU footprint to increase the reliability of mobile devices.**

I. Introduction

Mobile end-user devices such as smartphones and tablets usually have multiple ways of accessing the Internet, may it be WiFi (b/g/n), EDGE, 3G, 4G or Bluetooth. All these ways of accessing the Internet are usually provided by two separate radios. The WiFi adapter supporting the WiFi modes b/g/n and Bluetooth while the EDGE, 3G and 4G is provided by another adapter. Depending on the current usage of the device, every way can become the most efficient one but, given the very dynamic nature of the user's usage of the device, it is impossible for a static configuration to always achieve the lowest power consumption while still meeting the user's expected Quality of Service (QoS).

Likewise, ARM's big.LITTLE [1] architecture provides a set of cores where half of them are optimised for power consumption while the other half of them are optimised for performance. Every core can be powered down (power-gated) or have its clock modulated to lower the power consumption or increase performance (DVFS [2]).

While migrating applications from one CPU to another can be done locally at a relatively low performance- and power-cost, it is not the case in networks. Before MultiPath TCP (MPTCP) [3], it was not possible to seamlessly support data connection handovers without a proxy. This means that switching network interfaces would close all the TCP connections. MPTCP is very promising for multihomed devices as it allows seamless handovers of TCP connections if both the client and the server support it. This will enable selecting the most power-efficient network interface for the current usage of the user without needing a proxy. The MPTCP protocol is however not final yet, but it is implemented nonetheless by Apple for Siri in iOS 7 [4] or in an experimental tree of Linux.

Power and performance decisions are not easy to make because of the network handover and processor migration costs. It may indeed be more power-efficient to stick to a less instantly-efficient mode that would avoid a ping-pong effect which would be both damaging to the user experience and to the battery life. Numerous articles are found in the literature about autonomic network selection. However, they all use an application- and hardware-specific decision making module that needs to be re-implemented when a new radio is being used because accurate models are heavily hardware-, driver- and protocol-dependent.

In this article, we propose a generic decision flowgraph for power and performance management. This flowgraph allows run-time decision making, even with real-time requirements and is highly reusable thanks to its modular nature. We also implemented a framework around this generic decision flowgraph to minimise the amount of work needed to implement a new power- and performance-management decision-engine. This is done by separating the decision process in different blocks that can be re-used later. This eases experimentation during the design phase and the well-defined interface allows dumping debugging textual and graphical debugging information which are used throughout the article to showcase our framework. Finally, by using the proposed framework, applications also separate the decision engine from the application which improves its re-usability.

Section II covers the state of the art related to power and performance management. The definition of the proposed decision flowgraph is presented in section III and its implementation is detailed in section IV. An evaluation of the framework is given in section V. Section VI concludes and presents the future work.

II. State of the art

Decision engines are usually used in the literature for network interface selection in multihomed mobile devices, consolidating workloads in data centres [5] or making dynamic power/performance trade-offs in CPUs/GPUs through dynamic voltage-frequency switching [2].

Most articles on network selection in a multihomed environment usually aim at selecting the interface that will best suit the QoS. Some decision models are based on user-defined rules [6] or policy-based priority queues [7]. Other articles on

the same topic incorporate more metrics in the decision process such as the available bandwidth or the radio signal strength [8] while some also include power considerations [9][10]. A generic mathematical model is also proposed [11] and one is based on genetic algorithms [12] instead of the usual rule-based or linear optimisation selection algorithms.

However, we are not aware about any previous work taking into account the real-time and low performance-impact requirements for performance and power management. The real-time requirement is important because decisions must be taken at a fast and constant rate to react to changes in the user's activity and its surrounding. The low performance impact requirement matters because the resources taken by the decision engine cannot be used to produce useful work for the end user and increase the power consumption, which decreases the potential power savings made by the decision engine.

Decision engines found in the literature are usually written in high-level languages such as Matlab or linear-optimisation frameworks that are hard to bound in execution time and RAM usage. These languages usually have a high performance impact on the CPU and RAM usage compared to non-managed languages such as C. Genetic algorithms have also been used and have, by nature, no execution time boundaries since they should never converge. The best solution can however be selected at any generation which makes it suitable for real-time requirements in a slow decision rate scenario but may not be acceptable for its CPU and memory impact.

PISA [13] proposed to separate the application-specific part of an optimisation engine from the non-application-specific part and defined interfaces for them to communicate. This modularisation allows re-using the optimisation algorithms and share them with other researchers. PISA is language-independent which makes it potentially suitable for real-time applications, depending on the algorithms used. It is however not a framework and thus lacks debugging helpers.

In this article, we propose a flowgraph for generic and run-time decision making that follows the modularisation approach proposed by PISA for multi-variable optimisation. This flowgraph has been implemented in a framework written in C which gets the best potential for performance and efficience. This framework focuses on allowing real-time decisions for power and performance management while remaining generic-enough to not constraint the decision-engine programmer. It also eases the development of a decision engine thanks to having different modules to chose from for the different stages and thanks to its advanced debug outputs.

III. Definition of an abstract decision flowgraph

In this section, we propose the definition of an abstract and generic decision flowgraph that is compatible with real-time requirements.

A decision flowgraph is needed when there are more than one independent way to achieve the same goal, yet, they do not impact the power consumption and performance in the same way. For instance, in the case of a multihomed mobile device, we could want to take a decision whether to use the WiFi network interface or the cellular one. Taking a decision requires a performance and power model (HW model) of both interfaces. With these models, it should be possible to determine the optimal configuration to balance the needed performance with power consumption at any given time.

The hardware (HW) models are supplied with the user's performance/QoS needs continuously so as it can self-optimise. Instead of feeding the immediate QoS constraints to the models, we propose feeding a prediction of what will be the evolution of the constraints and their expected variance.

Once a HW model outputs a decision along with its impact on performance and power consumption, they should be compared with each others. In order to do so in an abstract way, we propose that a score on every output metric of every HW model should be generated and then aggregated into one score per HW model. All these information are gathered in the same data structure and sent to the decision-making part that will decide which HW model should be selected.

Our abstract decision flowgraph is composed of 5 stages:

- Metrics: Collecting metrics / performance needs;
- Prediction: Predicting the evolution of the metrics;
- HW models: Proposing the best decisions possible;
- Scoring: Scoring the output of the local decisions;
- Decision: Selecting the best HW model;

This decision engine flowgraph is considered generic because it does not force using a single model with different parameters, unlike other decision engines in the literature. This decision making model improves interchangeability and real-time-worthiness of a decision engine An overview of the abstract decision flowgraph is presented in Figure 1. Each stage will be further discussed in the following subsections.

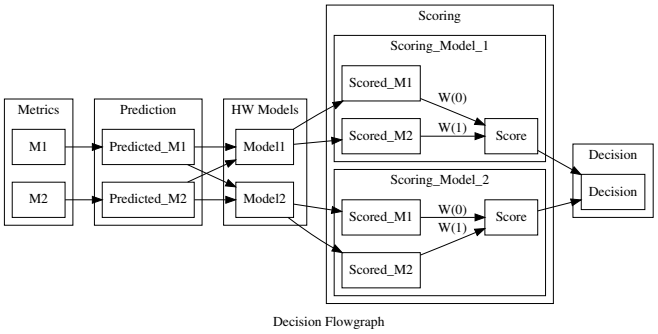

Figure 1. Overview of the abstract decision flowgraph

A. Decision flowgraph

A flowgraph represents a decision process. It includes the 5 stages discussed before and holds properties like the decision refresh rate. It can also run along-side other decision flowgraphs to, for example, allow an application to select the most efficient network interface while selecting the right performance level on the CPU.

B. Metrics

The role of the input metrics is to describe the current state of the system, including the performance requirements. They are not limited in numbers and should be updated periodically at a fast-enough rate to represent the current state and its variability. Poorly-chosen metrics will produce poor predictions at the next stage which will result in a poor decision quality. Examples of metrics could be the incoming and outgoing packets in the network stack or the CPU usage.

C. Prediction

The objective of the prediction stage is to supply the expected evolution of the system's state for a given time span. The needed time span depends on how often a decision is taken and how long it takes for a decision to be applied.

Prediction models may have multiple inputs of different types and generate multiple predictions of different types. For instance, a prediction model could take as an input the network packets received and predict the evolution of the reception throughput of the network interface.

Predictions act as constraints and can be metrics-based (run-time information on the state of the system) or static constraints. Static constraints can be used to express power or performance constraints such as latency that the HW model should take into account during its decision making.

The output of prediction models is composed of three graphs per prediction. The first graph represents the expected evolution of the predicted metric while the other two respectively represent the lowest and highest expected outputs for the metrics. These three graphs allow representing both the average and the expected variance at any point of the time span of the prediction.

Using graphs over mathematical formulas allows for a much more precise and real-time-friendly intermediate representation for predicted metrics. First of all, interpreting mathematical formulas at run time can be very CPU-intensive and linear optimisation has execution time that is hard to bound. Moreover, graphs are much easier to generate, read and score for a computer. They also represent the real expected evolution and, to some degree, the density of probability at any point of the prediction.

Some prediction models are proposed in the next section.

D. HW Models

HW models of a flowgraph should all describe an independent way to carry out the task that is needed by the user. They are user-provided, take the predicted metrics as an input and output the expected impact on the predicted metrics/constraints. Every output metrics of the models should be in the predicted metrics/constraints otherwise it is impossible to score it.

HW models have to be independent so that selecting one does not change how others calculate as this would require HW models to be aware of each others and this would break the modularity of the flowgraph.

In the case of network interface selection, there should be one HW model per network interface. All HW models would take the predicted network throughput along with the RF occupancy and latency constraints. HW models should then try to produce an optimal solution by selecting the right performance mode at the right time to have the lowest power consumption possible while still meeting the throughput requirement along with the latency and RF occupancy constraints. The HW models would then output graphs telling the evolution of the expected power consumption, RF occupancy and network latency in order to be scored.

The reason why there should be one HW model per interface and not per technology is because one network interface may support different technologies, such as the WiFi adapter also supporting Bluetooth, and they would not be independent.

E. Scoring

The scoring stage is meant to score the different HW models, before sending the result to the decision engine. The HW model's score is bound in $[0, 1]$ and is calculated based on the score of sub-metrics. Every sub-metric is assigned a weight that defines the importance of each metric ($W(0)$ and $W(1)$ in Figure 1). These weights need to be defined by the application. Scoring blocks can be user- or framework-provided.

F. Decision

The final stage of the decision process is selecting the most appropriate HW model and switching to it. The decision stage receives the scores of every HW model and metric, the HW models' output and the predictions. It is then responsible for evaluating the transition costs on performance and power consumption so as to avoid ping-pong effects that would be damaging to both the battery life and the QoS.

IV. Implementing the decision flowgraph in a framework

In section III, we proposed a flowgraph for a generic run-time decision engine. We have implemented this flowgraph into a framework in order to test it and make writing decision engines easier. We named this framework "Real-Time Generic Decision Engine" (RTGDE). Its implementation is available at https://github.com/mupuf/RTGDE.

This framework should not impede the implementation of a real-time decision process due to the framework's pipeline nature and the fact it barely does any processing at all except passing information from stage to stage. It should thus be possible to create a soft real-time decision engine as long as the predictions, HW models, scoring and decision algorithms used can have their execution time bound.

In this section, we detail the most important design constraints we had at the different stages of the decision pipeline.

A. Metrics

Run-time metrics collection and decision suggests multiple threads must be used to separate both processes. A cross-threads synchronisation is thus needed to allow the producer (metric collection) thread to send information (metrics values) to the consumer (prediction) thread. The use of a ring buffer, along with the usage of a mutex per metric, solves this communication problem. It is shown on Figure 2.

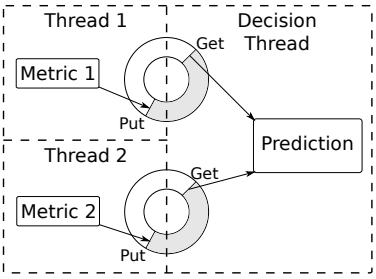

Figure 2. Using ring buffers to implement asynchronous communication between metrics collection and metrics prediction

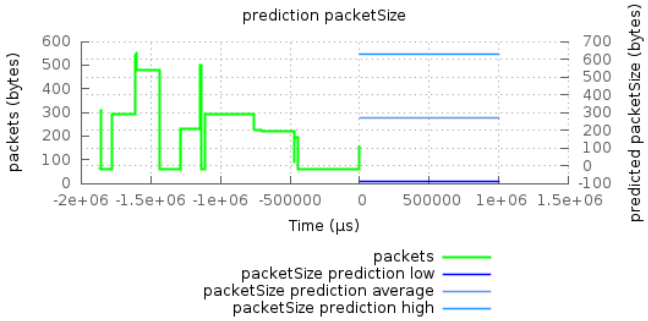

Figure 3. Example of the output of the "average" prediction method

B. Predictions

Predictions take attached metrics and output three graphs which are respectively the average, lowest and highest prediction over time. Graphs are used instead of formulas to ease development and make it easier to reach real-time guarantees.

The RTGDE framework aims at shipping multiple generic prediction engines. At the time of this paper, we have implemented the following ones:

- Simple: Equivalent to no prediction at all, it just outputs the latest value of the metrics;
- Average: Computes the average and the variance of the metric, outputs the average and the lowest/highest prediction based on a confidence level and the variance if the metrics follows a Gaussian.
- FSM: Considers the metric follows a Finite State Machine (FSM) with associated properties such as power consumption or throughput. The FSM prediction learns the density of probability to transit from one state of the FSM to an other and uses this knowledge to output the expected evolution of the metric. This prediction technique should work for periodic signals but requires a very high sample rate at the metrics. This prediction is not fully implemented yet.

Since every application has different needs, it may not be accurate to use generic prediction models. It was thus important to allow for user- along with framework-supplied prediction models. We decided to make it as easy as possible to do so. This means the internal interface should be exposed to the application in order to interface with it.

Constraints are considered as static predictions and define how to score the HW model's output metrics that are linked to it. For instance, the power consumption constraint of a model can be set so as the maximum score is achieved when the power consumption is 0 mW, the minimum score when drawing 3 W and the average score when consuming 1 W. Score is attributed linearly between these points.

Applications may require the prediction to be dumped along with the metrics data that was used to generate it. Predictions are then dumped as a Comma-Separated Values (CSV) file and a user-defined callback is called to allow the application to process this data. Figure 3 was generated using gnuplot, called from the user-defined callback of the pcap example located at '/tests/pcap/' in [14].

In this Figure, each point of "Packets" represents a received packet (size and time of arrival). The packetSize prediction low/average/high represent the output of the prediction stage concerning the packet size. In this example, it is not expected to change during the prediction period so the lines are flat.

C. Flowgraph

Prediction and HW models are attached to a flowgraph. Whenever the flowgraph is run, all the predictions algorithms generate the inputs for the HW models which are then called serially with a copy of the input metrics. They could however be run in parallel on multi-core processors to lower the execution time of the flowgraph.

D. Scoring

There is currently only one scoring algorithm implemented in the framework. The equation used to compute the HW model's score is given in Equ. 1 where $S_m(i)$ is the calculated score of the metric i while $W(i)$ is the weight given to the metric i and n is the number of metrics to be scored.

$$Score = \frac{\sum_{i=0}^{n} S_m(i) * W(i)}{\sum_{i=0}^{n} W(i)} \quad (1)$$

The function $S_m(i)$ is calculated by integrating the probability that the proposed result on the metric i is lower (or higher, depending on the metric) than its prediction at time t ($p_i(t)$), over time 0 to the prediction time-span (T).

$$S_m(i) = \frac{\int_{t=0}^{T} p_i(t).dt}{T} \quad (2)$$

Like for predictions, an application may require the scoring output to be dumped to a CSV file before calling a user-defined callback. Figure 4 is an example of the output of the default scoring method on the power consumption metric.

This figure is akin to Figure 3, except it is a power consumption constraint instead of a packet size prediction. We can also find an additional line (red) which represents the expected power consumption of the "radio-gsm" HW model. Finally, the decision impact, shown in light yellow, represents the difference between the ideal power consumption (power prediction low) and the expected power consumption.

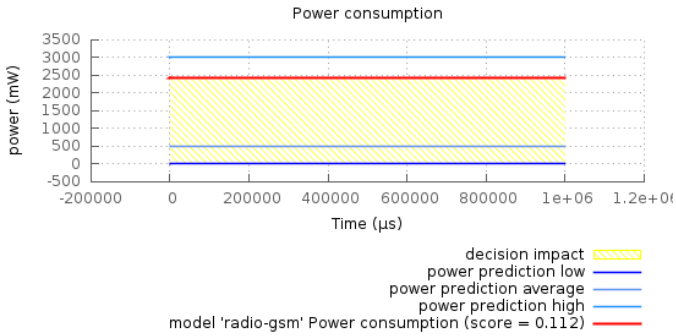

Figure 4. Example of the output of the default scoring method

E. Decision

We provide as much information as possible to the decision algorithm. This includes predictions and HW models outputs along with their scores.

Once a decision is taken, a user-defined callback is called with the result of the decision and all the information that was used to take it. As this call is currently made from the flowgraph thread, the application should not block.

V. Evaluation

In this section, we evaluate the genericity of the flowgraph and our framework by implementing two common use-cases for decision engines in green networking. We then evaluate the performance of the framework.

A. Network selection WiFi / GSM

As most of the decision engines in the literature are used for selecting network interfaces based on QoS and/or power efficiency, we decided to evaluate the framework by implementing a simple WiFi / GSM network interface selection.

The throughput requirement and expected packet count are predicted in real-time by sniffing the network traffic in the network stack, using libpcap [15] (the library used by tcpdump and Wireshark). In this experiment, the network activity was generated by browsing the web. The following constraints are set (Highest/Median/Lowest score):

- Power consumption: (0, 500, 3000 mW);
- RF spectrum occupancy: (0, 50, 100%);
- Latency: (0, 5, 10 μs).

For every decision, 2 predictions (Packet Size and Packet count) and 3 constraints (Power consumption, Radio Frequency (RF) occupancy and network latency) are generated. Both HW models then generate 3 outputs each for every decision. Given an update rate of 1 Hz, we get 11 CSV file every second. It is thus impossible to include all of them in this paper. Examples of outputs would be Figure 3 (output of the prediction for the "packetSize" metric) or Figure 4 (output of the scoring block for the "power consumption" constraint).

The WiFi and GSM radios are using the same HW model but with different parameters. The parameters are the name of the radio, its bitrate, the energy cost and delay to switch from reception (RX) to transmission (TX) and TX to RX and then, the power consumption when receiving and when transmitting. Those parameters are very simple and are not accurate as they do not take into account GSM and WiFi link quality. They are however sufficient to prove the genericity of the decision flowgraph and the framework as we are only demonstrating how the framework can be used and not evaluating the decisions taken by the engine.

The scoring technique is the one proposed earlier with the metric "Power consumption" (Figure 5) given a weight of 20, "Emission latency" (Figure 6) given a weight of 5 and "RF occupancy" (Figure 7) given a weight of 3. These weights should be defined according to the user's strategy and motivations. It is out of the scope of this paper.

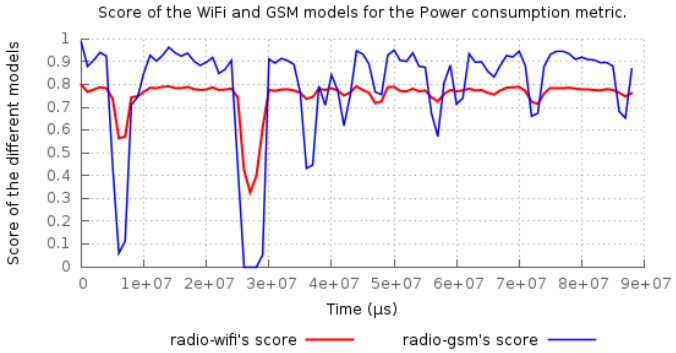

Figure 5. Evolution of the power consumption score of the two HW models

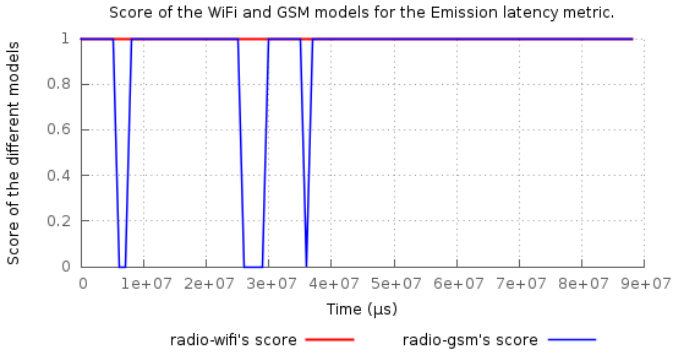

Figure 6. Evolution of the emission latency score of the two HW models

The decision to use the GSM model over the WiFi model is given if the following conditions are fulfilled:

- The average score of the GSM model in the last 30 seconds is higher than the WiFi model's score;
- The emission-latency average score of the GSM model in the last 30 seconds is higher than 0.9. This means the GSM's throughput is sufficient for the current load;
- The latest network interface switch happened more than 10 seconds ago.

As stated before, this will not yield a good decision but it is sufficient to prove the flexibility of the approach. The decision policy should be set according to the user's strategy.

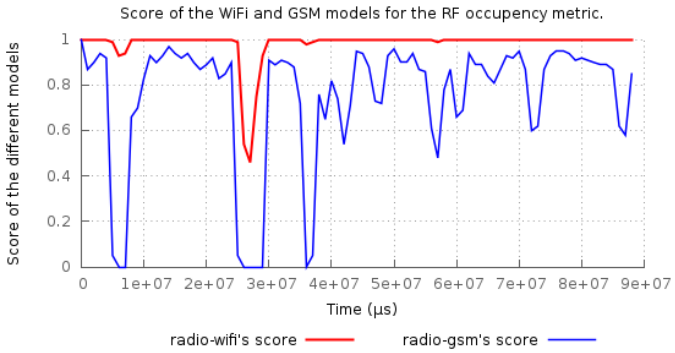

Figure 7. Evolution of the RF occupancy score of the two HW models

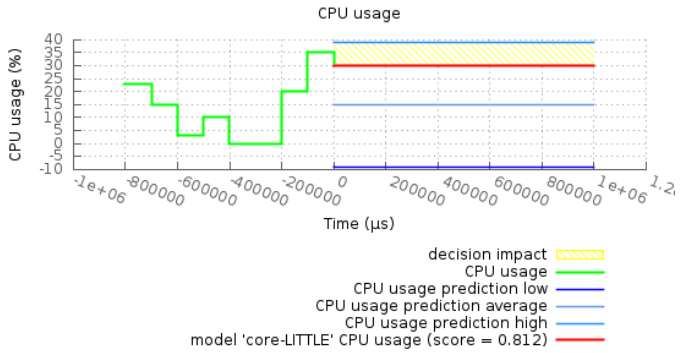

Figure 9. Evolution of the CPU usage metric and one prediction

The final score of both HW models along with the decision result is shown in Figure 8.

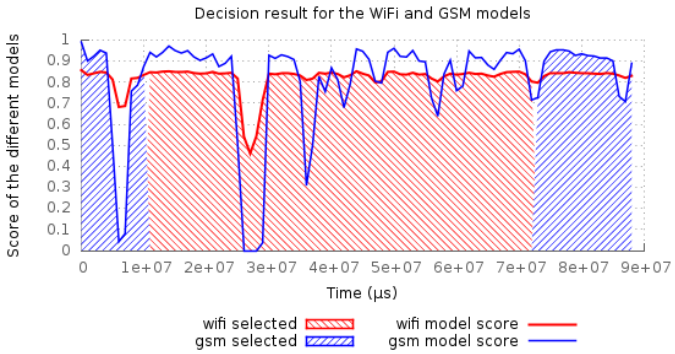

Figure 8. Evolution of the score of the two HW models along with the decision

All these graphs have been generated at run-time using the logging and callback capabilities of the framework.

In this evaluation, we proved it is possible to implement a network interface selection at run-time using our proposed generic flowgraph.

B. Selecting performance levels and processors

The second evaluation of the framework we performed was CPU performance level and core selection on a smartphone equipped with a big.LITTLE processor [1], where half the cores are optimised for speed (BIG) and the other half is optimised for power consumption (LITTLE). The decision engine's role is to select which set of core and performance level should be used for the applications currently running on the smartphone. It's input metric is the CPU usage which gets predicted using the "average" prediction system, as illustrated by Figure 9.

Each set of core (BIG or LITTLE) has 3 different performance levels. A performance level is defined by:

- Static power consumption: idle power consumption;
- Dynamic power consumption: power consumption added when 100% active;
- Performance: Maximum performance relative to the fastest performance level of the BIG core.

The LITTLE cores have the following performance levels:

- 0: Static = 10 mW, Dyn. = 200 mW, Perf. = 0.1;
- 1: Static = 15 mW, Dyn. = 300 mW, Perf. = 0.2;
- 2: Static = 25 mW, Dyn. = 500 mW, Perf. = 0.3.

The BIG cores have the following performance levels:

- 0: Static = 100 mW, Dyn. = 800 mW, Perf. = 0.25;
- 1: Static = 125 mW, Dyn. = 900 mW, Perf. = 0.63;
- 2: Static = 150 mW, Dyn. = 1000 mW, Perf. = 1.0.

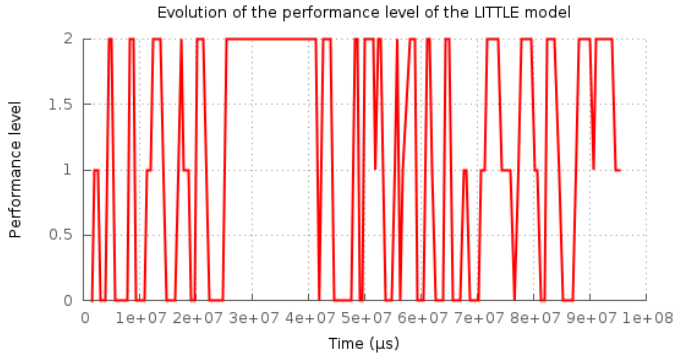

Figure 10. Evolution of the performance level used by the LITTLE model

We propose to use two HW models, one for the BIG set of cores and one for the LITTLE set of cores. BIG and LITTLE cores are handled using a single HW model with different parameters (name and performance levels). Performance levels are selected by both HW models while the set of core to be used is selected by the decision process. In Figure 10, we can see the selected performance levels for the LITTLE model across the experiment with 0 being the slowest performance level, 1 the median one and 2 the fastest one. Figures 11 and 12 respectively represent the evolution of the score for the performance impact / CPU usage and the power consumption score. The power consumption score comes from a static constraint set to 5/125/1000 mW (low, median, high).

The power consumption constraint is given a scoring weight of 10 while the performance impact score is given a weight of 13. The weights need to be adjusted according to the user's strategy.

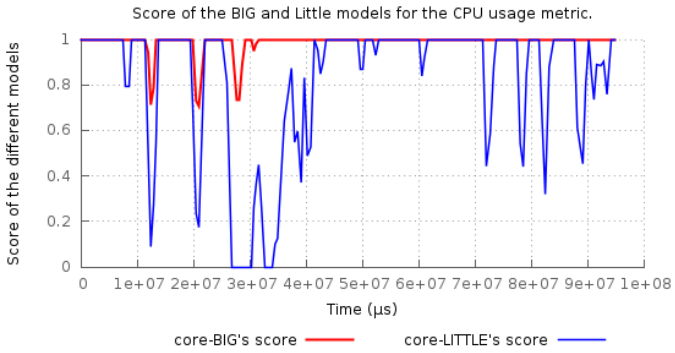

Figure 11. Evolution of the CPU usage score for the two HW models

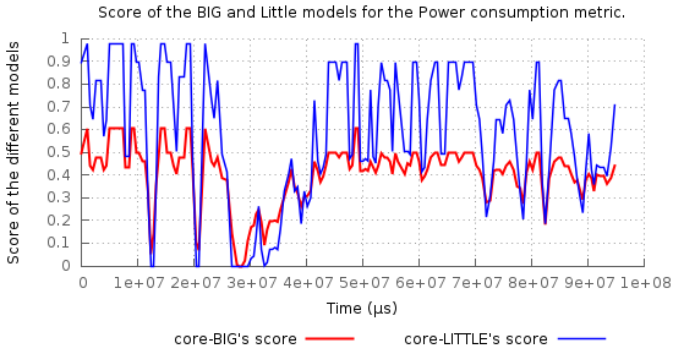

Figure 12. Evolution of the power consumption score for the two HW models

The decision policy is entirely based on the score of the two HW models. If the current model is LITTLE and if the score of the BIG model is 20% higher than LITTLE's score then the BIG model is selected. On the other hand, if the current model is BIG and the LITTLE's score has been 10% over the BIG's 5 times without the LITTLE's score dropping lower than 20% of the BIG's score, then the LITTLE model is selected. A decision is taken twice per second which is slow enough for performance- and power-cost of the migration to be negligible. Figure 13 presents the evolution of the score of the two HW models along with the proposed decision.

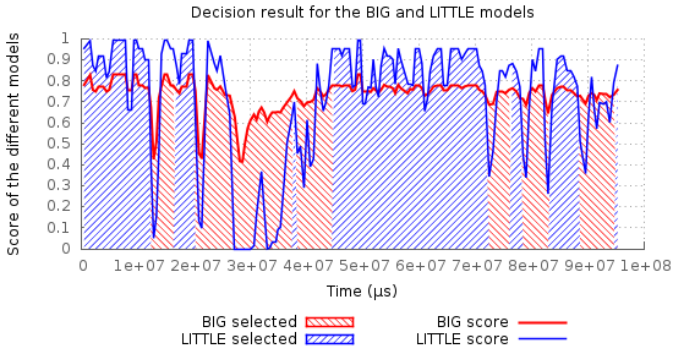

Figure 13. Evolution of the score of the two HW models along with the decision

C. Performance evaluation

The previous experiment, without CSV outputs and graphs generation, used approximately 296 kB of RAM on our system (Arch Linux x86_64) with a further 796 kB of libraries shared with other programs (libc, libglib, ld, libm, libpthread, libpcre, libXau). The actual memory footprint of RTGDE library is however just 36 kB and the volume of data created by the framework for this experiment is lower than 56 kB. Most of the remaining memory footprint of this experiment comes from the libgtop that was used to collect the CPU usage metrics, the libc and the stack allocation for the two threads of the process.

We also experimented with the CPU usage impact of the RTGDE framework in this evaluation and found that we had to increase the decision rate from 2 to 100 Hz to produce a visible impact on CPU performance of 2% on one core of our i7 860. One iteration of the decision process takes 43.95 μs in average with a standard deviation of 13.19 μs.

We decided to compare the memory usage and real-time worthiness of our proposition compared to MATLAB as it has often been used by researchers to implement decision engines [11][10]. We compared our proposition on Linux using GNU Octave which is an open source alternative compatible with MATLAB. To compare our proposition and an implementation made in Octave, we wrote the minimum application possible that executes code one thousand times per second in our proposition and in Octave, respectively found in [14] under the name "test _minimal.c" and "matlab_min.m".

To test the real-time worthiness of the two solutions, we compared the delay between the moment when the application requested to resume execution, called scheduling jitter, and the standard deviation of this jitter over 30 seconds when our testing Linux system was idle. The results highly depend on the CPU scheduler but also reflect any additional overhead the language is putting on real-time constraints. They are visible in Figure 14 where we can see that Octave's jitter is almost 50% higher than the one found in the C language used to implement our framework. Octave's standard deviation in the scheduling delay is more than 60% higher than C's.

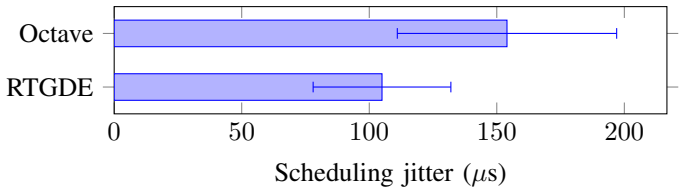

Figure 14. Jitter in the scheduling of decision process on Linux. Data collected during 30 seconds when the system is idle.

In the second test, we measure the memory usage of both solutions and split it in two categories. The "shared" memory usage comes from the different libraries used by both solutions, named this way because they can be shared in memory with other processes. On the contrary, the "private" memory usage is memory which is exclusively used by the program. The results of this test can be found in Figure 15 where we can see that Octave's private memory usage is 25.3MB along with 7.4MB worth of libraries that could be shared with other processes. On the other hand, our proposition has a private memory footprint

more than 150 times lower (168kB) than Octave's while its shared memory footprint is more than 15 times lower (1.4MB).

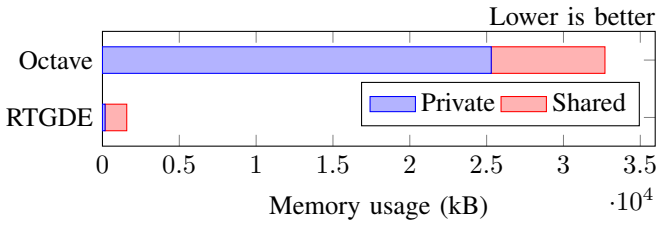

Figure 15. Minimum memory usage when using RTGDE or Octave to implement the decision engine

A comparative performance evaluation of MATLAB and C++ was carried out for plot generation and it was found that C++ was faster and more versatile [16]. C being a sub-set of C++, it should share the same characteristics.

Through this evaluation, we demonstrated that it is possible to implement a complex performance level selection at run-time on an heterogeneous CPU architecture by using our proposed flowgraph and framework. We also demonstrated that the framework's CPU and memory footprint can be negligible, even for mobile devices, contrarily to solutions found in the state of the art which are mostly based on MATLAB.

VI. Conclusion and Future Work

In this paper, we proposed a generic flowgraph for run-time decision making. We implemented it in a framework and evaluated it by successfully implementing run-time algorithms for network interface and CPU performance level selection.

On Linux, the RTGDE framework requires less than 58kB of storage memory, uses less than 100 kB of RAM for typical flowgraphs and has virtually no impact on CPU performance when not generating gnuplot graphs in real time. This is to be compared with other solutions such as MATLAB that can use up to 150 times as much as RAM as our proposition. This makes this decision flowgraph and framework very suitable for mobile devices. The implementation of this framework has been achieved in about 4000 lines of C code with only one dependency to pthread and is available at https://github.com/mupuf/RTGDE under the Open Source MIT license.

Future work include adding more generic prediction algorithms, evaluating the flowgraph on more scenarios and writing tools to help generating the code for setting up the prediction engine. Thanks to the modular nature of the framework, a website could also be written to allow researchers to share their prediction algorithms or HW models for radios, CPUs or GPUs to limit the amount of code that needs to be written to set-up the prediction engine. Currently, researchers have to implement HW models for every hardware used in their decision-making. Predictions could also be scored by RTGDE and be used to automatically select the most-accurate prediction system. The score could also help debugging poor prediction issues.

Acknowledgment

This work has been funded by the cluster of excellence CPU, University of Bordeaux. Part of this work was performed while Martin Peres was an invited researcher at the IMAGINE Research in the School of Electrical Engineering and Computer Science, University of Ottawa, Canada under the supervision of Professor Ahmed Karmouch.

References

[1] P. McKenney, "A big.LITTLE scheduler update," Jun. 2012. [Online]. Available: https://lwn.net/Articles/501501/

[2] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: the laws of diminishing returns," in *Proceedings of the 2010 international conference on Power aware computing and systems*, ser. HotPower'10. Berkeley, CA, USA: USENIX Association, 2010, p. 1–8. [Online]. Available: http://dl.acm.org/citation.cfm?id=1924920.1924921

[3] M. Handley, O. Bonaventure, C. Raiciu, and A. Ford, "TCP extensions for multipath operation with multiple addresses," Jan. 2013. [Online]. Available: https://tools.ietf.org/html/rfc6824

[4] Olivier Bonaventure, "Apple seems to also believe in multipath TCP," Sep. 2013. [Online]. Available: https://perso.uclouvain.be/olivier.bonaventure/blog/html/2013/09/18/mptcp.html

[5] J. Choi, S. Govindan, J. Jeong, B. Urgaonkar, and A. Sivasubramaniam, "Power consumption prediction and power-aware packing in consolidated environments," *IEEE Transactions on Computers*, vol. 59, no. 12, pp. 1640–1654, 2010. [Online]. Available: http://csl.cse.psu.edu/publications/power-tc10.pdf

[6] J. Ylitalo, T. Jokikyyny, A. J. Tuominen, and J. Laine, "Dynamic network interface selection in multihomed mobile hosts," *HAWAII INTL. CONF. ON SYSTEM SCIENCES (HICSS*, vol. 9, p. 315, 2003. [Online]. Available: http://citeseer.uark.edu:8080/citeseerx/viewdoc/summary?doi=10.1.1.99.9062

[7] L. Liu, Y. Wei, Y. Bi, and J. Song, "Cooperative transmission and data scheduling mechanism of multiple interfaces in multi-radio access environment," in *Global Mobile Congress 2009*, 2009, pp. 1–5.

[8] B. Hu, S. Chen, W. Ye, and X. Zhan, "An autonomic interface selection mechanism for multi-interface host," in *2012 8th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM)*, 2012, pp. 1–5.

[9] M. Chowdhury, Y. M. Jang, C. S. Ji, S. Choi, H. Jeon, J. Jee, and C. Park, "Interface selection for power management in UMTS/WLAN overlaying network," in *11th International Conference on Advanced Communication Technology, 2009. ICACT 2009*, vol. 01, 2009, pp. 795–799.

[10] P.-N. Tran and N. Boukhatem, "SiPiA: The shortest distance to positive ideal attribute for interface selection," in *Telecommunication Networks and Applications Conference, 2008. ATNAC 2008. Australasian*, 2008, pp. 175–179.

[11] M. Sowmia Devi and P. Agrawal, "Dynamic interface selection in portable multi-interface terminals," in *IEEE International Conference on Portable Information Devices, 2007. PORTABLE07*, 2007, pp. 1–5.

[12] G. Castignani, N. Montavont, and A. Arcia-Moret, "Multi-objective optimization model for network selection in multihomed devices," in *2013 10th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, Mar. 2013, pp. 113–115.

[13] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler, "PISA — a platform and programming language independent interface for search algorithms," in *Evolutionary Multi-Criterion Optimization*, ser. Lecture Notes in Computer Science, C. M. Fonseca, P. J. Fleming, E. Zitzler, L. Thiele, and K. Deb, Eds. Springer Berlin Heidelberg, Jan. 2003, no. 2632, pp. 494–508. [Online]. Available: http://link.springer.com/chapter/10.1007/3-540-36970-8_35

[14] Martin Peres, "mupuf/RTGDE." [Online]. Available: https://github.com/mupuf/RTGDE

[15] tcpdump, "Tcpdump/libpcap public repository," 1987. [Online]. Available: http://www.tcpdump.org/

[16] Tyler Andrews, "Computation time comparison between matlab and c++ using launch windows," Jun. 2012. [Online]. Available: http://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1080&context=aerosp