Run-Time Managed Mobile Application Execution

Michele Zanella, Giuseppe Massari and William Fornaciari
Dipartimento di Elettronica,
Informazione e Bioingegneria Politecnico di Milano
Milan, Italy
Email: {michele.zanella, giuseppe.massari,william.fornaciari}@polimi.it

***Abstract*—Achieving an optimal management of the energy budget of mobile devices, while matching the applications performance requirements is always a challenging task. In our research, we are exploring the possible benefits of driving run-time management strategies, from an application perspective, by integrating the programming model with the run-time system and exploiting suitable API for explicit application requirements specification.**

***Index Terms*—Mobile Computing, Power, Performance, Run-time Management, Resource Management, Mobile Programming**

I. Introduction and Related Works

While mobile computing devices offer always richer functionality and capabilities [1], [2], the energy budget management still represents the upper bound to the exploitation of further performance rooms. Moreover, new paradigms such as Fog and Edge computing [3]–[5], are starting integrating mobile devices as source/target nodes of the infrastructure, for both sensing and computational purposes [6]–[9]. This requires to carefully take into account power consumption and performance requirements of mobile applications, as well as to provide isolation, for separating the applications from the execution of external offloaded tasks.

From the energy efficiency standpoint, different solutions have been proposed, both hardware and software, to improve battery duration [10], [11]. At this regard, various solutions focus on improving an efficient usage of power-hungry components such as Wi-Fi, 3G and GPS. Other approaches operate on the backlight level of the display by reducing it without affecting the perceived gameplay quality [12]. Moreover, different power-saving applications, like SetCPU [13], have been developed. They apply different system-wide profiles on the basis of the actual device condition (e.g., battery level). However, such profiles are typically static, which means they do not capture the specific usage patterns.

On the hardware side, it is worth noticing that in the latest years, heterogeneous multi-cluster architectures, such as ARM big.LITTLE and DynamIQ, integrated low-power cores and high-performance ones on the same die, enabling the possibility of reducing power consumption or boosting performance, according to the current workload [14]. For instance, most of the OS kernels provides a set of CPU governors (i.e., interactive, performance, ondemand...) with a dynamic

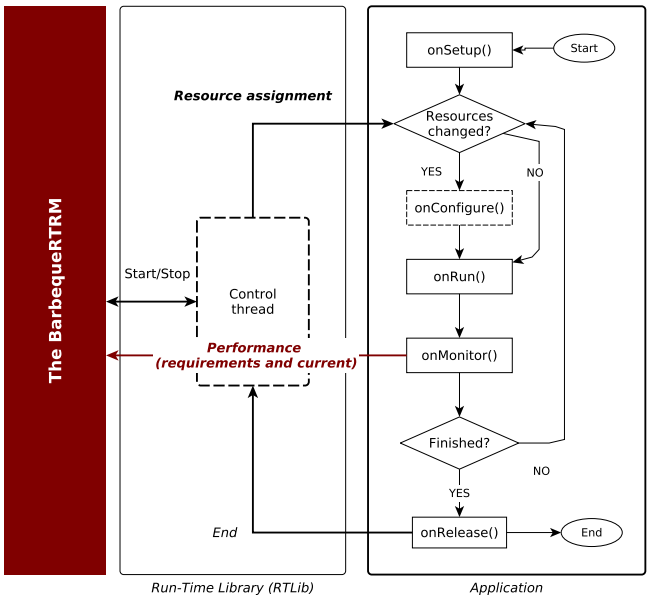

Fig. 1. Adaptive Execution Model

behaviour, although they do not take into account feedback and time varying requirements coming from applications.

Currently, the Android power management is based on a native *power driver* at kernel-level [15], to control the peripherals' states, and the *Application Framework*, through which application can acquire a "wake lock", when required, to prevent the system from switching off components [11]. Unfortunately, this approach does not allow us to integrate the power management actions into a fine-grained resource management strategy, based on the current user-level requirements of each single application.

In our research, we aim at exploring possible benefits coming from adapting the run-time management strategies (DVFS, task scheduling, resource partitioning) to the current application requirements. This is implemented by exposing a suitable API, through which the application can specify its performance (or power saving) goals to the operating or run-time system. This, will tune the amount of assigned resources (e.g, number of CPU cores), or a hardware configuration (e.g, CPU operating point) accordingly. Our approach follows an

978-1-7281-1796-6/19/$31.00 ©2019 IEEE

interesting work [16] where the presented framework can detect the state of a game application to infer the current performance requirement. Despite this work, however, we are able to provide a full integration with the application so that the resource manager can detect directly the run-time status of the application and negotiate the requirements, as well as having a finer control over device's resources.

The rest of the paper is organized as following: Section II discusses our approach and the design of the solution, while a preliminary experimental evaluation is presented in Section III. Finally, Section IV draws a roadmap for on-going and future works, concluding the paper.

II. Run-Time Adaptive Application Execution

In this work, we extend the Barbeque Run-Time Resource Manager (BarbequeRTRM) framework [17], to properly support Android based scenarios. We introduce an infrastructure on top of the native implementation of the resource manager, to effectively integrate it in the Android environment. In Figure 1, we sketch the concept behind the run-time managed application execution proposed by the framework. This allows us to implement mobile applications that:

1) Are aware of the assigned resources, such that they can properly configure themselves accordingly;
2) Can notify the run-time about dynamically changed performance requirements (e.g., explicit power-saving or boots mode).

This defines what we called *Adaptive Execution Model (AEM)*. A managed execution flow in which application and resource manager interacts, such that the former can adapt itself to the current status of the device (e.g., system-level power saving mode) or the latter can take into account the explicit application requirements. The resource manager will therefore assign resources based on the current system status (e.g. battery level, temperature, etc...) and the time varying applications and user requirements.

The Figure 1 shows the Run-Time Library (RTLib), between the application and the resource manager. A suitable wrapper has been implemented in the application library to make the run-time API available also to Android applications. The API included in the library is based on the idea of implementing an application-specific class (C++/Java) derived from the `BbqueEXC`. An instance of such a class spawns a control thread, in charge of synchronizing the application execution and the resource management actions, by calling the objects' methods (`on-` prefix) accordingly. Looking at the Figure 1, it should be immediate to notice how this follows the already consolidated programming approach to Android application development. In detail, the `onSetup` method will contain initialization code. In the `onConfigure` the application can check the amount of CPU, memory and network bandwidth assigned by the BarbequeRTRM and configure itself accordingly (as in Lines 29 and 31 of Listing 1), while the `onRun` includes the core of the processing. In particular, the implementation of the `onMonitor` method, can include (1) the setting of the application performance goal (i.e, the throughput), or (2)

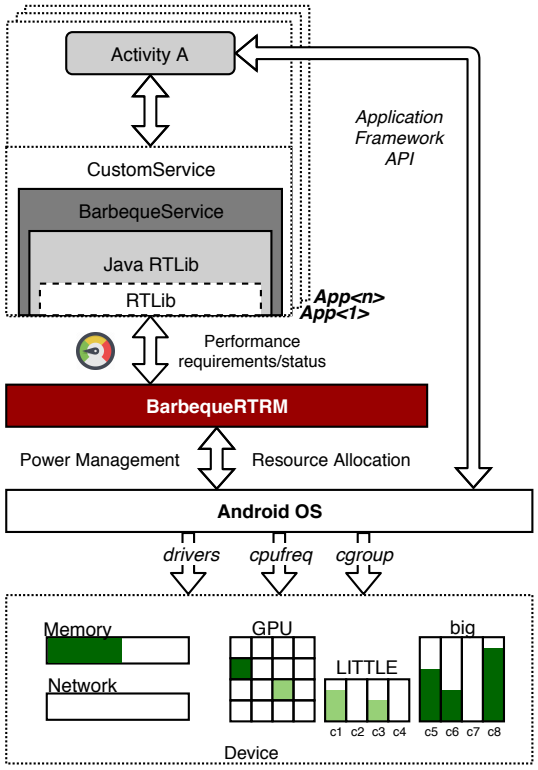

Fig. 2. Overview of the management framework and Android API

an explicit constraint on the resource allocation (as in Line 42 of Listing 1). In this regard, the application can come with a set of predefined resource assignment configurations, called *Application Working Modes (AWM)*. In general, the application's description consists of a `xml` file (called *recipe* in BarbequeRTRM jargon) containing information related to set of preferred resources (e.g., type of processors, memory and peripherals...), QoS requirements (e.g., completion time, frame per seconds, network bandwidth...) and power profiles (i.e., energy-saving, balanced, boost).

At run-time, the application can set an upper bound (e.g. "use at most 2 CPU cores" or "bound the network bandwidth utilization"), triggered by an explicit interaction with the user (application settings), to reduce the power consumption according to the actual user requirements. This application execution follows a control loop, in which it can re-enter the configuration step `onConfigure`, in case of resource assignment changes, an continue the execution until the suitable exit condition is not verified.

Figure 2, instead, provides an overview of the integration of the BarbequeRTRM in the Android system. For the `RTLib`-based interaction between application and resource manager, a Java service-based wrapper is provided [18]. In order to be run-time adaptive, the application must implement a `CustomService`, which extends the basic `BarbequeService`, as shown in Listing 1. This provides wrappers, binders and messengers interfaces towards the other activities of the application. Then, the `CustomService` has

```
1  import bbque.rtlib.RTLib;
2
3  public class CustomService
4      extends BarbequeService {
5
6  @Override
7  public void onCreate(){
8      ...
9      rtlib = RTLib.init(APP_NAME);
10      mEXC = new CustomEXC(
11     APP_EXC_NAME,APP_NAME,rtlib);
12     ...
13 }
14
15 protected class CustomEXC
16     extends AndroidBbqueEXC {
17
18     @Override
19     public void onRun()
20     throws RTLibException{
21         /* Workload */
22     }
23
24     @Override
25     protected void onConfigure(
26         int awm_id)
27     throws RTLibException {
28         ...
29         freq = getAssignedResources(
30             RTLibResourceType.FREQ);
31         cpu_quota=getAssignedResources(
32             PROC_ELEMENT);
33         ...
34     }
35
36     @Override
37     protected void onMonitor()
38     throws RTLibException {
39         ...
40         RTLibConstraint constraint =
41             new RTLibConstraint(...);
42         setAWMConstraints(constraint);
43     }
44 }
```

Listing 1. Example of BarbequeRTRM API in Android applications.

to implement the `AndroidBbqueEXC`'s callbacks through a `CustomEXC` (line 15), which is the Android equivalent of the `BbqueEXC` class previously introduced. On the other side, the native BarbequeRTRM daemon uses OS-level frameworks and third-party libraries to enforce resource management actions. For instance, Linux frameworks like `cpufreq` and `cgroup` are currently exploited to set CPU operating points, reserve CPU time quota or cores. In order to evaluate both application requests and device constraints for an effective resources management, the framework allows us to implement device-specific policies, as already shown in [19].

III. Experimental Evaluation

We performed a preliminary experimental evaluation, in order to have a proof-of-concept of the system prototype. For the experimental scenario, we executed the *Image Effect* benchmark, from the *MobileXPRT2015* [20] suite, on a *Hikey 960* development board, equipped with a 8-core big.LITTLE processor and Android OS 8.0. The "big" cluster is represented by a 2.4 GHz quad-core $A73$ CPU, while a 1.8 GHz quad-core $A53$ represents the low-power "LITTLE" cluster. In particular, the two CPU clusters have 5 DVFS operational points in the specific frequency range of each cluster, which is [903-2362] MHz for the "big" one and [533-1844] MHz for the "LITTLE".

AWM	Cluster	Freq(KHz)	Power(W)	Perf(fps)
1	big	2,362	3.09	2.79
2	big	2,112	2.78	2.50
3	big	1,805	2.58	2.13
4	big	1,421	2.53	1.74
5	big	903	2.25	1.13
6	LITTLE	1,844	2.58	1.37
7	LITTLE	1,709	2.52	1.21
8	LITTLE	1,402	2.07	0.86
9	LITTLE	999	2.23	0.65
10	LITTLE	533	2.21	0.31

TABLE I
THE SET OF PROFILED AWM. POWER AND PERFORMANCE COLUMNS ARE MEAN VALUES.

AWM	Cluster	Freq(KHz)	Power(W)	Perf(fps)
1	big	2,362	3.09	3.79
3	big	1,805	2.58	0.82
8	LITTLE	1,402	2.07	1.25
10	LITTLE	533	2.21	0.51

TABLE II
THE SET OF PROFILED AWM APPLIED DURING A DYNAMIC EXECUTION. POWER AND PERFORMANCE COLUMNS ARE MEAN VALUES.

During the evaluation we followed a two-steps approach. First, we profiled the entire execution of the benchmark in terms of (a) performance, expressed as the number of edited photos per second, and (b) power consumption of the board. This in order to define a set of AWMs with respect to the target device. For each AWM, we then reported the frequency setting, the average system power consumption and the performance estimation, as shown in Table I. The next step involved the profiling of the benchmark execution, where the application varied its requirements at run-time, simulating the user intervention. In this case, the Figure 3 shows the application varying the AWM setting and the related variation of performance and power consumption, as detailed in Table II. In this regard, we developed a sample policy that chooses different AWMs within a constrained range, set by the maximum throughput required by the user. In this sense, if in a specific moment we do not need to run the application at the topmost performance level, and we want to save some power, we can explicitly negotiate the AWM assignment with the resource manager. In fact, in the execution of our sample, we can observe how the initial AWM setting (AWM1) led the device to consume a lot of power, while the application performance were above the required level. After a new negotiation, the resource manager reassigned a less power consuming AWM10. Later, a

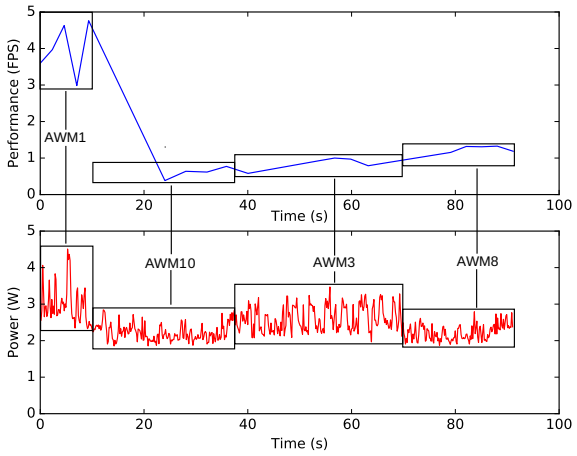

Fig. 3. Benchmark's performance and system's power consumption. The frames represent the execution profile of different AWM setting.

new change in the application requirements (boost), triggered the assignment of AWM3. Finally, a further scale down of requirements led to the last AWM reassignment (AWM8). In summary, thanks to this run-time managed execution model, the application can cooperatively reduce the device power consumption, allowing us to go beyond the usual system-wide low-power mode setting. Finally, as we can observe, the AWM are built through offline profiling. Future works will go in the direction of dynamically binding resource assignment and explicit application requirements at run-time.

IV. Conclusions and Future Works

In this work, we shown an alternative approach to power saving with respect to setting a system-wide low power consumption mode. This may be driven by the application themselves, in cooperation with a run-time resource manager (the BarbequeRTRM). To this aim, we introduced a programming model to develop adaptive and reconfigurable mobile applications. To prove the potentiality of our approach, we performed a preliminary evaluation on a real Android-based board using a benchmark application, suitably integrated with the proposed programming model.

The following steps of the research will move our focus in three directions: (a) building an experimental setup with different devices to better assess potential of the framework; (b) developing and validate novel QoS and energy aware management policy, that could adapt the resource assignment and power configuration, on the basis of both the application requirements and the current energy budget of the device; (c) considering a further exploitation of the OS platform framework for a improved management of the device.

V. Acknowledgments

This work has been partially funded by H2020-FETHPC projects: MANGO (n.671668) and RECIPE (n.801137).

References

[1] ARM, "big.little technology: The future of mobile," 2013.
[2] Y. Wang, I.-R. Chen, and D.-C. Wang, "A survey of mobile cloud computing applications: Perspectives and challenges," *Wireless Personal Communications*, vol. 80, no. 4, pp. 1607–1623, 2015. [Online]. Available: http://dx.doi.org/10.1007/s11277-014-2102-7
[3] F. B. et al., "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 13–16.
[4] C. Puliafito, E. Mingozzi, and G. Anastasi, "Fog computing for the internet of mobile things: Issues and challenges," in *2017 IEEE International Conference on Smart Computing (SMARTCOMP)*, May 2017, pp. 1–6.
[5] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct 2016.
[6] M. Zanella, G. Massari, A. Galimberti, and W. Fornaciari, "Back to the future: Resource management in post-cloud solutions," in *Proceedings of the Workshop on INTelligent Embedded Systems Architectures and Applications*, ser. INTESA '18. New York, NY, USA: ACM, 2018, pp. 33–38. [Online]. Available: http://doi.acm.org/10.1145/3285017.3285028
[7] C. Xian, Y. Lu, and Z. li, "Adaptive computation offloading for energy conservation on battery-powered systems," in *2007 International Conference on Parallel and Distributed Systems*, Dec 2007, pp. 1–8.
[8] D. Kovachev, T. Yu, and R. Klamma, "Adaptive computation offloading from mobile devices into the cloud," in *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, July 2012, pp. 784–791.
[9] J. I. Benedetto, A. Neyem, J. Navon, and G. Valenzuela, "Rethinking the mobile code offloading paradigm: From concept to practice," in *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, May 2017, pp. 63–67.
[10] Google, "Android power management," 2016, retrieved Feb 14, 2019 from https://source.android.com/devices/tech/power/mgmt.
[11] S. K. Datta, C. Bonnet, and N. Nikaein, "Android power management: Current and future trends," in *Enabling Technologies for Smartphone and Internet of Things (ETSIoT), 2012 First IEEE Workshop on*. IEEE, 2012, pp. 48–53.
[12] B. Anand, K. Thirugnanam, J. Sebastian, P. G. Kannan, A. L. Ananda, M. C. Chan, and R. K. Balan, "Adaptive display power management for mobile games," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '11. New York, NY, USA: ACM, 2011, pp. 57–70. [Online]. Available: http://doi.acm.org/10.1145/1999995.2000002
[13] M. Huang, "Setcpu," 2015, retrieved Apr 17, 2019 from https://www.setcpu.com/documentation.html.
[14] ARM, "Energy aware scheduler," 2016, retrieved Feb 14, 2019 from https://developer.arm.com/open-source/energy-aware-scheduling.
[15] S. K. Datta, "Android stack integration in embedded systems," in *International Conference on Emerging Trends in Computer & Information Technology, Coimbatore, India*, 2012.
[16] B. Dietrich and S. Chakraborty, "Lightweight graphics instrumentation for game state-specific power management in android," *Multimedia Systems*, vol. 20, no. 5, pp. 563–578, Oct 2014. [Online]. Available: https://doi.org/10.1007/s00530-014-0377-x
[17] P. Bellasi, G. Massari, and W. Fornaciari, "Effective runtime resource management using linux control groups with the barbequertrm framework," *ACM Trans. Embed. Comput. Syst.*, vol. 14, no. 2, pp. 39:1–39:17, Mar. 2015. [Online]. Available: http://doi.acm.org/10.1145/2658990
[18] BOSP, "The barbequertrm android api," 2019, retrieved Feb 14, 2019 from https://bosp.deib.polimi.it/doku.php?id=docs:rtlib:newandroidapp.
[19] M. Zanella, G. Massari, and W. Fornaciari, "Enabling run-time managed distributed mobile computing," in *Proceedings of the 9th Workshop and 7th Workshop on Parallel Programming and RunTime Management Techniques for Manycore Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*, ser. PARMA-DITAM '18. New York, NY, USA: ACM, 2018, pp. 39–44. [Online]. Available: http://doi.acm.org/10.1145/3183767.3183778
[20] A. Morgan, "Benchmark selection guide, vol.1," 2015, retrieved Feb 14, 2019 from https://bit.ly/2X2sPys.