ISSN:0975-9646

Performance-Energy Aggregate metric based Scheduler (PEAS) for Smartphones

Rashmi Devi, Preeti Sharma

Department of Computer Science and Engineering
Shri Ram College of Engineering & Management, Delhi-Mathura Road (NH-2)
Palwal, Haryana, India

***Abstract*— Since the introduction of Cyber Foraging, there has been a need for a scheme that partitions computation among client and server for optimal performance without compromising energy consumption. In previous schemes, scheduler was installed on server resulting in low adaptability to changing network and local computational resources. MARS, the first adaptive, online and lightweight scheduler, achieved better performance as well as energy savings over these schemes by enhancing adaptability due to its installation on mobile device. We introduce PEAS (Performance-Energy Aggregate metric based Scheduler), an improved version of MARS, which is even more lightweight hence more adaptable. PEAS operates on the principle: keep both server and the client fully utilized at all times such that energy constraints are inherently taken care of. A thorough evaluation of PEAS over wide range of applications and network conditions is presented. We propose modifications in our scheduler to tackle high energy consumption for poor networks. Converging trend in performance of different schemes with network improvement is also demonstrated.**

***Keywords*— Pervasive computing, Cyber foraging, Client-server, Adaptive, RPC, Scheduler**

I. INTRODUCTION

Mobile computing has become quite popular in last decade. Mobile computing has gone through leaps and bounds since its introduction in past two decades. However, technological advances had little impact; it is still speed and energy limited because of slow mobile processors and battery capacity. Wireless networks such as 3G, Wi-Fi introduces the possibility of using ample computing resources via migration of resource intensive computation tasks to external powerful server resources. Various methods since the birth of pervasive computing [1] had evolved and have been used for remote execution. Earlier works solely uses remote execution for performing given tasks irrespective of other parameters. However this strategy is not always beneficial. In poor network conditions, network power overhead consumes high amount of battery resources also high latency increases the total execution time. We also witness from above case that energy savings and run-time depends heavily on network conditions, computation task and server load. This calls for smart remote execution methods which consider local processor as a resource. The works of [2], [3], and [4] address the above problem and include local processor as a resource. These methods called for optimization of distribution of tasks to mobile and server resources the variable being network parameters. Several works for example [5] has been done either on energy or on performance. All these new methods were based on static client-server partition. The important network parameters such as network congestion, upload bandwidth, download bandwidth, latency etc. are highly variable in nature. These methods did not account for high variability of offloading parameters. However, improved methods such as [6], [7] delve into this problem up to some extent by introducing dynamic methods of pervasive computing. The work of MAUI [7] in the league of client server partitioning methods is quite remarkable in terms of energy saving and performance also to some extent. The above work is chosen as benchmark for static client server method in this work for comparison.

Our scheduler primarily deals with execution of remotely executable RPCs (Remote Procedural Calls) which are predefined by developer according to the application requirement. High degree of randomness in networking parameters calls for dynamicity in scheduling of remotely executable RPC's. As that would be much closer to real time scenario which is being encountered while offloading to server resources. In his regard, a novel method was introduced by Cidon and Tomer [8]. In their method, scheduler was installed on mobile device itself thus enabling scheduling decisions to be taken on mobile device itself instead of being pre-programmed by developer (static client server method) or by VM migration method. In VM migration method [9], [10] remote execution is conducted by migrating a VM from the mobile device to a remote server, or emulating the entire mobile operating system on the server. However, as discussed in related works section of MARS [8] due to adaptability issues VM migration is not preferred in our algorithm.

The introduction of scheduler on mobile fine grains the scheduling decisions and better results were obtained than prevalent state of the art methods. However, the weight of scheduler is of primary concern for installing it on mobile device. The algorithm used by [8] follows greedy approach to avoid complicated computations thus keeping scheduler to be light weight and adaptive. Our work is a continuation

in this direction. We aim towards optimizing the greedy approach being followed by MARS and we present a better algorithm which results in more adaptive, more light weighted scheduler, better speedup and better energy savings. The work of [8] is somewhat application specific which again requires a rigorous analysis for wide variety of applications over various networks to ascertain its utility. Thus, following above idea we have analyzed various application and presented them in form of abstraction for generalizing the results. Our focus is on wide applicability of our algorithm over different networks such as 3G, Wi-Fi etc. We have also analyzed for future networks conditions such as 4G and improved Wi-Fi.

II. THE PEAS ALGORITHM

The problem at hand is to minimize execution time and energy consumption of the given set of remotely executable RPCs. As mentioned previously, modern processors feature multiple threads and cores. For supporting such architectures in a rapidly varying wireless environment, dynamic remote execution systems need to be adaptive, online and lightweight. These criteria are inherently taken care of by the structure of PEAS. PEAS sorts the candidate RPCs for remote execution in a priority queue. A performance-energy metric, called EPOR (Energy-Performance Offload Rank) is used for sorting the queue. EPOR estimates energy savings and speedup gained by offloading the RPC. At any moment, we expect the RPC at the top of the queue to be the best candidate for offloading as it promises highest energy savings and speedup gain. Similarly, the RPC at the bottom of the queue with lowest EPOR is expected to be the best candidate for local execution. The EPOR metric arranges RPCs in ascending order of remote execution so whenever remote resource is available it offloads RPC present at the top of the queue and RPC at the bottom of the queue is locally executed. Before every scheduling decision, the queue is resorted by updating its EPOR metric to adapt to changing network and CPU resources. Since energy considerations are modelled into EPOR metric, this scheduling scheme should ensure energy savings along with performance gain.

A. RPC Model

A remote execution RPC is the smallest unit of remote execution. It is assumed that remote execution RPCs are parts of a program that are defined by the programmer to be considered as candidates for remote execution. These units require a significant and finite amount of time to execute locally (hundreds of milliseconds or more). Remote execution RPCs are assumed to be functionally idempotent and independent. They do not use any shared memory and synchronization mechanisms, and have no inter-thread dependencies. Each remote execution RPC is executed by the server in a separate thread i.e. the server can run large (ideally infinite) number of RPCs in parallel.

B. The EPOR Metric

EPOR metric is used by PEAS to sort RPCs in a priority queue. The idea behind this metric is to model both performance and energy criteria in a single parameter making PEAS extremely lightweight and easy to implement. EPOR metric is defined below:

$$\mathrm{EPOR} = \left(\frac{\mathrm{MobileExecutionTime}}{\mathrm{ServerExecutionTime} + \mathrm{CommunicationTime}}\right) \times \left(\frac{\mathrm{MobileExecutionTime} \times \mathrm{DevicePower}}{\mathrm{RemoteExecutionEnergy}}\right) \quad (1)$$

$$\mathrm{CommunicationTime} = \frac{\mathrm{InputSize}}{\mathrm{UploadBandwidth}} + \frac{\mathrm{OutputSize}}{\mathrm{DownloadBandwidth}} + \mathrm{RTT} \quad (2)$$

$$\mathrm{RemoteExecutionEnergy} = \left(\frac{\mathrm{InputSize}}{\mathrm{UploadBandwidth}} + \mathrm{RTT}\right) \times \mathrm{UploadPower} + \left(\frac{\mathrm{OutputSize}}{\mathrm{DownloadBandwidth}} + \mathrm{RTT}\right) \times \mathrm{DownloadPower} \quad (3)$$

The first term on RHS of equation (1) accounts for speedup on offloading i.e. local execution time over remote execution time while second term is the estimated mobile energy consumption of executing a RPC locally over its remote execution. It ensures that PEAS should remotely execute RPCs without compromising the device's energy consumption. This model of priority metric EPOR takes into account several dynamic factors; the network bandwidth, latency, the computation task and the local execution time on the mobile device. However, it does not take into account dependencies among RPCs e.g. whether there are any subsequent functions waiting for the RPC to complete.

In order to adapt to changing network conditions, PEAS periodically profiles the network and CPU. Before a scheduling decision, PEAS checks the new status of the network and mobile resources subsequently updates the EPOR of all the RPCs and resorts the queue according to the new EPOR values. When a new remote execution RPC enters the queue, its position in the queue is determined according to its EPOR.

C. PEAS Scheduler

PEAS algorithm operates on following principle:

"Given a set of remotely executable RPCs, keep both the network and the local CPU fully utilized at all times such that RPCs with high EPOR are executed remotely and RPCs with low EPOR are executed locally"

Every time a scheduling decision needs to be made, scheduler calls the PEAS algorithm given on next page. TopOfQueue and BottomOfQueue refer to the RPCs at the top and bottom of the queue respectively. saveRPCState() saves the RPC's data, so that if the network becomes disconnected, PEAS can resume the execution of the RPC locally. WaitForResource() is called when none of the resources is

PEAS Algorithm

```
1: if Network.isFree() AND Mobile.isBusy() then
2:     saveRPCState(TopOfQueue)
3:     offload(TopOfQueue)
4: else if Mobile.isFree() AND Network.isBusy() then
5:     runLocally(BottomOfQueue)
6: else if Network.isFree() AND Mobile.isFree() then
7:     saveRPCState(TopOfQueue)
8:     offload(TopOfQueue)
9: else
10:    WaitForResource
11: end if
```

available for executing the RPC then it waits in the queue until next resource is available. Network.isFree() refers to the availability of serial communication wireless network for migration of RPCs.

The above algorithm always keeps both local CPU (Mobile) and Network busy thus enabling full utilization of resources unlike MARS which requires an additional energy criterion to be satisfied before directing an RPC for remote or local execution. Thus performance of PEAS is bound to be better than that of MARS. Since EPOR modelling took into account energy considerations, the device energy consumption should never be unreasonably high compared to MARS or MAUI. Computational weight of the scheduler and its adaptability to changing network scenario go hand in hand. PEAS should therefore demonstrate better adaptability than MARS and hence other schedulers.

It is easy to apply our scheduler to multi-core processors. Each time a core becomes free, the scheduler pops a remote execution RPC from the bottom of the queue. We do not model multi-core features such as shared caches and thread migration among cores.

III. Evaluation

In this section, we evaluate PEAS's ability to improve the performance and energy savings of smartphone applications. Simulation methodology is described in section III A.

A. Simulation Methodology

We aim at carrying out a thorough evaluation of our scheduler that spans wide range of applications and network conditions. For this we used four types of applications based on associated communication and computation costs. To cover varied network scenarios, we used three different networks.

We categorized applications on the basis of their communication and computation costs. For an application the time consumed in transmitting data over network for remote execution, called communication cost, is considered as low and high. Similarly, the computation overhead or total execution time of an application, called computation cost, is considered as medium and high. This gives four different applications having:

1. Low communication cost and Medium computation cost (LM),
2. Low communication cost and High computation cost (LH),
3. High communication cost and Medium computation cost (HM) and
4. High communication cost and High computation cost (HH)

These abstractions cover most of the applications' space. Note that the comparison of computation and communication costs is with respect to the efficiency/capacity of the computing platform (both the mobile and the server) and of the wireless network protocols and transceivers. In our framework, applications that incur low computation cost are not considered as they can be executed on the mobile device itself.

First consider applications that incur high computation costs. The associated communication cost can be low or high. For low communication cost applications, the relatively small size of RPCs (up to about 30 KB) allows user to offload RPCs with minimal transfer time and low power overhead. However, their computation is complex and resource intensive thus favoring remote execution. Applications requiring complex mathematical calculations for e.g. finding prime numbers [11], encrypt-decrypt applications fall under this category. Second type of applications in high computation cost category are those with high communication cost e.g. video editing which consists of RPCs of the order of hundreds of kilobytes and also incurs high computation overhead. For such applications, network parameters: bandwidth and latency play critical role in scheduling decisions. Availability of good network conditions will promote offloading of RPCs.

Second set consists of moderate computation cost applications. Again, the associated communication cost can be low or high. Applications such as face recognition, augmented reality are considered to be high communication cost and medium computation cost applications having RPCs' size over about 30KB and small server execution time of the order of tens of milliseconds. Distributed voice recognition [12] and secure communications [13] are good examples of applications having low communication cost and moderate computation cost.

We considered three different networks: Poor, Average and Good. Corresponding data used in simulations is given in the following table:

TABLE I
NETWORK PROPERTIES

Parameters	**Poor Network**	**Average Network**	**Good Network**
Bandwidth Range	10-100 KBps	50-500 KBps	100-1000 KBps
Latency/RTT	100 ms / 200ms	25 ms / 50ms	10 ms / 20 ms
Network Idle Power Overhead	0.3 W	0.0005 W	~0 W
Power Upload Overhead	2 W	0.2 W	0.1 W
Power CPU Idle	1.5 W	1.5 W	1.5 W
Computation Overhead	0.1 W	0.1 W	0.1 W

TABLE II
RUNTIMES FOR GOOD NETWORK

Application	Runtime (in Seconds)				% Improvement in Runtime over	
	Local	MAUI	MARS	PEAS	MAUI	MARS
LM	2.52	0.3049	0.2903	0.2898	4.952443	0.172236
LH	100	2.4276	2.3036	2.3015	5.194431	0.091162
HM	5.04	1.363	1.107	1.05	22.96405	5.149051
HH	37.5	3.83563	3.49029	3.47826	9.317113	0.344671

TABLE III
RUNTIMES FOR AVERAGE NETWORK

Application	Runtime (in Seconds)				% Improvement in Runtime over	
	Local	MAUI	MARS	PEAS	MAUI	MARS
LM	2.52	0.6011	0.6159	0.514	14.4901	16.54489
LH	100	5.21	4.68	4.67	10.36468	0.213675
HM	5.04	2.261	1.966	1.773	21.58337	9.816887
HH	37.5	7.78747	6.44144	6.33168	18.694	1.703967

TABLE IV
RUNTIMES FOR BAD NETWORK

Application	Runtime (in Seconds)				% Improvement in Runtime over	
	Local	MAUI	MARS	PEAS	MAUI	MARS
LM	2.52	2.52	2.52	1.42	43.65079	43.65079
LH	100	18.11	24.67	16.61	8.282717	32.67126
HM	5.04	5.04	5.04	3.867	23.27381	23.27381
HH	37.5	35.9955	24.4451	18.185	49.4798	25.60881

TABLE V
ENERGIES FOR GOOD NETWORK

Application	Energy (in Seconds)				% Improvement in Energy over	
	Local	MAUI	MARS	PEAS	MAUI	MARS
LM	4.032	0.48355	0.48006	0.47918	0.903733	0.18331
LH	160	3.8602	3.8671	3.8636	-0.08808	0.090507
HM	8.064	2.203	1.858	1.767	19.79119	4.89774
HH	60	6.11904	5.80735	5.86837	4.096558	-1.05074

TABLE VI
ENERGIES FOR AVERAGE NETWORK

Application	Energy (in Seconds)				% Improvement in Energy over	
	Local	MAUI	MARS	PEAS	MAUI	MARS
LM	4.03326	1.003	1.051	0.889	11.3659	15.41389
LH	160.05	8.677	8.216	8.206	5.428143	0.121714
HM	8.066	3.824	3.411	3.124	18.30544	8.413955
HH	60.01875	13.1432	11.4265	11.2567	14.35343	1.486019

TABLE VII
ENERGIES FOR BAD NETWORK

Application	Energy (in Seconds)				% Improvement in Energy over	
	Local	MAUI	MARS	PEAS	MAUI	MARS
LM	4.788	4.788	4.788	4.375	8.625731	8.625731
LH	190	57.82	64.545	53.277	7.857143	17.45759
HM	9.576	9.576	9.576	13.527	-41.2594	-41.2594
HH	71.25	70.5795	60.1409	65.0841	7.786114	-8.21936

One can easily relate these networks to low bandwidth 3G, average Wi-Fi and improved Wi-Fi or 4G respectively.

We borrowed device, network and application data from [8]. We used this data and extended it to formulate above mentioned abstractions of devices, networks and applications keeping them realistic. An application consists of some number of independent tasks each having 6 sequentially dependent stages (linear multistage model [11]).

All the simulation results shown here are compared with best static partitioning MAUI and the previous best dynamic, adaptive mobile installed scheduler, MARS.

B. Results

Based on our simulation methodology, all four sets of applications were simulated over diverse networking conditions modelled as three networks namely good, average and poor. Comparisons are drawn with prevalent methods such as local execution, best static portioning (MAUI) and MARS for execution time and energy consumption. Consistency is maintained in simulations with respect to device's computational power and computational load for each algorithm and over different networks. In subsequent sections, network based simulation results shown in tables II-VII are discussed.

1) Good Network: The results obtained by simulating applications over good networks (tables II and V) exhibit interesting observations. The convergence in performance and energy savings of different scheduling algorithms is observed. The explanation lies in the fact that remote execution is facilitated by good networking conditions. Therefore, we could assert that most of the RPC's are scheduled for remote execution. However, in high communication cost and medium computation cost applications, increased use of local CPU provide PEAS 23% speedup gain over MAUI. The limited use of mobile CPU in other applications results in marginal speedup gain for PEAS.

2) Average Network: In average networks, due to low offload power overhead, energy consumption largely depends upon total runtime of the computational task. It results in making remote execution an energy efficient choice for the execution of RPCs. Therefore, both performance gain and energy savings favor offloading (tables III and VI). PEAS, being performance centric scheduler algorithm, also becomes energy efficient due to low offload power overhead. Speed gain up to 16% and 22% is observed for PEAS over MARS and MAUI respectively. Similar trend is followed in energy savings; varying up to 15% and 18% for PEAS over MARS and MAUI respectively.

3) Bad Network: PEAS is primarily designed for performance factor which is being exemplified by speedup results (tables IV and VII) shown by PEAS over MAUI (up to 50% speedup) and MARS (up to 44%). It is apparent from above results that PEAS prevails in poor networking conditions by fully utilizing available resources. These outcomes also indicate that PEAS is best suited for performance centric applications which require higher refresh rates such as video games. As discussed earlier, PEAS being primarily designed for speedup its energy consumption needs to be scrutinized due to presence of high upload network power over-head.

In high communication and medium computation cost applications, high energy consumption is observed. This is easier to understand as high communication cost would incur high energy consumption due to upload network power overhead. Also, medium computation cost would not allow PEAS to draw maximum benefits from remote execution resulting in extensive battery usage. However, except for high communication cost and medium computation cost case energy results are not drastically deteriorating in nature. Furthermore, in some applications energy savings up to 17% and 9% is observed for PEAS over MARS and MAUI respectively.

To satiate these needs, we devised battery usage modes namely high performance mode and power saver mode based on common prevalence of these modes in computers, laptops etc. Mode can be manually chosen by user or preconfigured in design. The user chooses the available battery mode catering to his/her need. High performance mode is PEAS algorithm only. Power saver mode requires high energy savings without catering to performance requirements. Selection of this mode activates energy aware offloading. This is easily implemented with insertion of just one more check in scheduling algorithm before every offloading decision. It should be noted that we performed all simulations assuming high performance mode.

IV. CONCLUSIONS

This work introduces PEAS, a single metric based, extremely light-weight and online scheduler for multi-threaded systems. The scheduler is easily extendable to multi-core clients. Our simulation results show that PEAS gives up to 50% better performance and energy savings over MAUI and MARS. The fact that PEAS maintains its superiority over wide range of applications and networks is also established from simulations. We also demonstrated converging trend in results of all schemes with improvement in network.

REFERENCES

[1] Satyanarayanan M., "Pervasive computing: vision and challenges, "Personal Communications, IEEE, vol.8, no.4, pp.10-17, Aug 2001 doi: 10.1109/98.943998

[2] A.Messer, I.Greenberg, P.Bernadat, D.Milojicic, D.Chen, T.J.Giuli ,and X.Gu. Towards a Distributed Platform for Resource-Constrained Devices. In Proceedings of the International Conference on Distributed Computing Systems (ICDCS), pp.43-5,2-5, Vienna, Austria, July 2002.

[3] M.Tatsubori, T.Sasaki, S.Chiba and K.Itano. A Bytecode Translator for Distributed Execution of ÓLegacyÓ Java Software. In Proceedings of the European Conference on Object- Oriented Programming (ECOOP) ,LN CS 2072, pp.236-255, Sprinter-Verla, 2001.

[4] E.Tilevich and Y.Smaragdakis, J-Orchestra: Automatic Java application partitioning. In Proceedings of the 16th European Conference on Object-Oriented Programming (ECOOP) , LN CS 237 4, pp.17 8-204, Sprinter-Verla, 2002.

[5] A. Rudenko, P. Reiher, G. Popek, and G. Kuenning, "Saving Portable Computer Battery Power through Remote Process Execution, "Mobile Computing and Comm. Rev., vol. 2, no. 1, pp. 19-26, 1998

[6] Hens, Raf; Boone, Bas; de Turck, Filip; Dhoedt, Bart; "Runtime Deployment Adaptation for Resource Constrained Devices,"

Pervasive Services, IEEE International Conference on , vol., no., pp.335-340, 15-20 July 2007 doi: 10.1109/PERSER.2007.4283936

[7] Cuervo, E., Balasubramanian, A., ki Cho, D.,Wolman, A., Saroiu, S., Chandra, R., and Bahl, P.Maui: making smartphones last longer with code offload. In MobiSys (2010), pp. 49–62.

[8] Asaf Cidon, Tomer M. London, Sachin Katti, Christos Kozyrakis, and Mendel Rosenblum. 2011. MARS: adaptive remote execution for multi-threaded mobile devices. In Proceedings of the 3rd ACM SOSP Workshop on Networking, Systems, and Applications on Mobile Handhelds (MobiHeld '11). ACM, New York, NY, USA.

[9] Kirsch, C. M., and Heiser, G., Eds. European Conference on Computer Systems, Proceedings of the Sixth European conference on Computer systems, EuroSys 2011, alzburg, Austria - April 10-13, 2011 (2011), ACM.

[10] Chun, B.-G., Ihm, S., Maniatis, P., and Naik, M. Clonecloud: Boosting mobile device applications through cloud clone execution. CoRR abs/1009.3088 (2010).

[11] Gitzenis, S.; Bambos, N.; , "Joint Task Migration and Power Management in Wireless Computing," Mobile Computing, IEEE Transactions on , vol.8, no.9, pp.1189-1204, Sept. 2009 doi: 10.1109/TMC.2009.34

[12] B. Delaney, T. Simunic, and N. Jayant, “Power Aware Distributed Speech Recognition for Wireless Mobile Devices,” IEEE Design & Test, vol. 22, no. 1, pp. 39-49, Jan. 2005.

[13] L. Yuan and G. Qu, “Design Space Exploration for Energy-Efficient Secure Sensor Network,” Proc. IEEE Int’l Conf. Application-Specific Systems, Architectures and Processors, pp. 88-97, 2002.