Power-Aware Middleware for Mobile Applications

Shivajit Mohapatra[2], M. Reza Rahimi[1], Nalini Venkatasubramanian[1]
[1]Dept. of Information and Computer Science,
University of California, Irvine, CA 92697-3425, USA.
[2] Motorola Research Labs, Schaumburg, Illinois (USA).
{mrrahimi, nalini}@ics.uci.edu,
{mopy@labs.mot.com}.

Abstract:

This chapter develops a middleware based approach to optimizing the power consumption of low-power mobile devices executing rich applications such as multimedia streaming and location based services. In mobile distributed environments, generic middleware services (e.g. caching, location management etc.) are widely used to satisfy application Quality-of-Service (QoS) needs in a cost effective manner. Such middleware services consume system resources such as storage, computation and communication and can be sources of significant power overheads when executed on low-power devices. In particular, we develop techniques for partitioning tasks to realize mobile applications into reconfigurable components; we also advocate a proxy based approach wherein components can be dynamically stopped or migrated away from a low-power device to an in-network proxy node, thereby extending the life time of the device. Specifically, 1)determine whether a reconfigurable component-based middleware framework can be utilized to achieve energy gains on low-power devices in distributed environments and 2) design and evaluate techniques for dynamically determining middleware component reconfigurations, and ascertaining the optimal frequency at which the restructuring should take place, for maximal energy and service time gains at the device. As the case study we will introduce a power-aware middleware framework (PARM) and identify some of the intrinsic requirements for the framework to be effective.

Contents

1 Introduction

The next generation of the internet and intranets will see a wide variety of small low power devices operating on board with high-end systems. With progressive improvements in technology and wide ranging user demands, these computers are now being exploited to perform diverse tasks in an increasingly distributed fashion. Mobile systems come in different flavors, including laptops, cellular phones, web-phones, PDAs, pocket computers, sensors to name a few. All such devices have wireless communication capabilities enabling them to operate in distributed environments. Due to their modest sizes and weights, these systems have inadequate resources - lower processing power, memory, I/O capabilities, storage and battery life as compared to desktop systems. These portable devices are mostly battery driven and often-times have to run on batteries for considerable time periods. Therefore, power management is a critical issue for these systems and tackling power dissipation and improving service times for these systems have become crucial research challenges.

The growing popularity of distributed middleware systems coupled with their scalability, flexibility and affinity to mobile and wireless architectures has made them a dominant methodology for supporting distributed applications. *We refer to "middleware" as an abstraction layer that sits between the operating system and the application and provides a level of abstraction for distributed applications*. The primary advantage of having middleware is that it runs on heterogeneous environments shielding applications from the individual underlying technologies. Today, wireless enabled end-to-end middleware frameworks find application in a variety of areas: gaming, chat systems and other similar peer-to-peer (P2P) applications are already available. Other applications like mobile banking, Business to Employee (B2E) systems like mobile workforce automation, Machine-to-Machine (M2M) systems like remote administration, manufacturing control etc. are quickly becoming a reality.

Applications in distributed environments require a multiplicity of services in order to accomplish their tasks. A distributed middleware system can provide important additional services like reliable messaging, distributed snapshots, clock sync services, location management services, CPU scheduling, persistent data management services, system security(encryption/decryption), directory management among others, the complex details of which remain hidden from applications. In a reconfigurable middleware framework, one or more of these component services can be independently started, stopped or moved by a meta-level entity. This plug and play approach obviates the need for all middleware components to be running on a low power device at all times. Additionally, new services may need to be installed /uninstalled to cope with dynamic requirements. Customizable middleware frameworks can be considered as important candidates for energy optimization as they can be pruned depending on the workload and residual power of the device. For example, a cache management service can be offloaded to a nearby proxy or cloud saving on both power and storage, while still providing appreciable performance benefits of caching [CM09, CBC10].

In this chapter, we present as a case study an adaptive, reconfigurable middleware framework (PARM) that can significantly improve energy uptake of low-power devices operating in distributed environments. It provides for runtime middleware customizations on a low-power device, while still maintaining the semantics of the middleware services. Furthermore, the adaptation decisions are shifted to an external entity with cross-application knowledge and overall system information. As the residual energy on the device diminishes, the PARM middleware dynamically reconfigures the component distribution on the low-power device to improve the time of service of the device. We develop a graph theoretic solution [AMO93, CGK97] for determining the optimal middleware component reconfigurations and empirically identify, when and how often, reconfigurations should occur for optimal energy gains. The reconfiguration decisions are driven by comprehensive profiling of the energy usage patterns of various applications and middleware components. Finally, we simulate and extensively test and analyze the performance of our approach using three different classes of applications that form a representative workload for all low-power devices.

The rest of the chapter is organized as follows: Section 2 discusses the current state of the art in support for energy efficiency for mobile applications. Section 3 argues for a proxy-based approach where an in-network node assumes some of the processing and storage capabilities required for mobile applications and introduces a power aware middleware framework (PARM) for this purpose. Section 3 characterizes the communication/computation tradeoffs as a PARM component reconfiguration problem, models it using a source parametric flow network and presents an optimal polynomial time solution to the component reconfiguration problem. In section 4, we present the performance evaluation and the prototype implementation of the PARM framework. We conclude by discussing future research direction in Section 5.

2 State of Art in Power Aware Mobile Computing

A tremendous amount of research has already been done for achieving power savings in low-power and embedded devices. The research efforts have ranged from defining measurement techniques [MZ95, MSS03], software strategies [LS96, IBM02], monitoring and modeling [FLS01, FS99, FPS02] for power management to more system level approaches like slackening or powering down components (e.g. CPU) when not in use. Modern mobile devices also present several opportunities to achieve power savings by providing capability to operate various hardware components in multiple low duty-cycle operating modes. Consequently, several interesting solutions for energy optimization have been proposed at various computational levels. At the architectural level, solutions include disk spin-down policies [DKB95, DKM94], turning of the displays during periods of inactivity, effective battery management [CR00], minimizing power by managing the mobile host's communication device [IGP95, KK98, SK97], system cache and external memory access optimization [FS96, HSA01, HKA01], dynamic power management of disks and network interfaces [C02 CV02], efficient compilers and application/middleware based adaptations [MV03, NSN97, FLS01].

Dynamic Voltage Scaling (DVS [BSM04, LS96, LS98, PS01, WWD94, YN02, YN04, YN03]) tries to address the tradeoff between performance and battery life, by adjusting CPU speed (and therefore power) depending on the system workload. It uses the fact that for most computers the peak computing rate needed is much higher than the average throughput that needs be sustained. DVS scales the operating voltage of the CPU along with the frequency to achieve power gains. Most voltage scaling initiatives are undertaken at the operating system (or lower) level and require OS support for adjusting CPU speeds. Hughes et. al. [HKA01] have analyzed architectural variability in the frame-level execution time for a number of multimedia applications on general-purpose architectures. [HSA01] proposes a technique for combining two hardware adaptations (architecture adaptation and dynamic voltage scaling) for reducing energy in multimedia workloads. The study concludes that DVS alone gives most of the energy benefits, while the more aggressive architectures are more energy efficient in the presence of DVS. In [SG01, ISG02], Gupta et al. present "online" power management strategies for systems with multiple idle power states. They compare the effectiveness of the online algorithms against the optimal algorithm (competitive analysis) and present a generalized analysis of dynamic power management strategies for systems with multiple sleep states. [JG02] extends the dynamic voltage scaling approach by calculating static slowdown factors for tasks that need to synchronize for access to shared resources.

There is also a rapidly growing body of work on HW–SW co-synthesis under energy constraints [SAE05, W04]. Typical work in design space exploration proposes ad-hoc scheduling policies to minimize the energy [SAE05] in the context of multimedia applications. However, such an approach adds the overhead of customized scheduling tables for each individual application. Prior work has also explored a co-synthesis methodology that guarantees real-time operation with low area/energy requirements [KBN08]; our techniques exploit the existing heterogeneity in MPSoCs by choosing from a library of well-known scheduling policies such as RM (rate monotonic), EDF(earliest deadline first), and permits extensive design space exploration multiple tradeoff points can be evaluated.

There have been several research efforts to dynamically modify application behavior dynamically to conserve energy, with some help from the operating system. Flinn and Satyanarayanan, demonstrated that a collaborative relationship between the operating system and applications can be used to meet user-specified goals for battery duration [FLS01, FS99]. Odyssey [NSN97] presents an applications aware adaptation scheme for mobile applications. In this approach the system monitors resource levels, enforces resource allocation and provides feedback to the applications. The applications then decide on the best possible adaptation strategy. The Milly Watt project [E99] explores the design of a power-based API that allows a partnership between applications and the system in setting energy use policy. In the context of this project, a currency model that unifies energy accounting over diverse hardware components and enables fair allocation of available energy among applications, and a prototype energy-centric operating system, ECOSystem, that implements explicit energy management techniques from the system point of view have been proposed [ZEL02]. Their goal is to extend battery lifetime by limiting the average discharge rate and to share this limited resource among competing tasks according to user preferences. PowerScope [FS99] is an interesting tool that maps energy consumption to program structure. It first profiles the power consumption and system activity of a computer and then generates an energy profile from this data.

At the middleware level, the focus has been on reducing the communication and computation required at the mobile device end while providing adequate QoS for the executing mobile applications. One of the primary directions in this context has been to optimize network interface power consumption [C02, CV02, FM01]. In [SAS01], an energy

aware EDF packet scheduling scheme for real-time traffic called modulation scaling is presented. In effect, this work describes a technique for integrating network communication into a dynamic power management scheme on a radio. [C02] compares the energy consumed by network interface cards for three different video streaming formats. They suggest the traffic shaping at a proxy for sending multimedia data, so that network cards can be transitioned to sleep states during periods of inactivity for saving power. An initial study of energy consumption characteristics of network interfaces was made in [SK97]. Radkov and Shenoy [SR03] present a methodology for estimating video decoding times and also provide two mechanisms for intelligent transmission of multimedia data. [FN01] makes a comprehensive evaluation of energy consumption of network interface cards and formulates formulas for relating energy consumption with data transfer sizes. Farkas [FFB00] et al., have characterized the energy consumption of a pocket computer and evaluate the energy and power tradeoffs of in the design of the JVM for the machine.

Given the high compute/communication overheads of rich multimedia traffic, much of the research in network level adaptations has focused on dealing with streaming multimedia content to the mobile device. A thorough analysis of power consumption of wireless network interfaces has been presented in [FM01]. Chandra et al. [CV02] have explored the wireless network energy consumption of streaming video formats like Windows Media, Real media and Apple Quick Time via techniques such as energy-aware traffic chaping close to the mobile device [CV02]. In [SR03], Shenoy suggests performing power friendly proxy based video transformations to reduce video quality in real-time for energy savings. Caching streams of multiple qualities for efficient performance has been suggested in [FPS02]. In [YN00], a resource aware admission control and adaptation is suggested for multimedia applications for optimal CPU gains. In [2] the authors propose a server controlled power management scheme much like ours where the server exploits the workload, traffic related information and feedback from the mobile device to minimize the energy consumption of the network interface card. Puppeteer [FLS01] presents a middleware framework that uses transcoding to achieve energy gains. Using the well-defined interface of applications, the framework presents a distilled version of the application to the user, in order to draw energy gains. In [ANF03, ANF04], Anand et.al. present a middleware that allows applications to provide “ghost hints" to OS level power managers for implementing power management strategies (for disk cache management). This is in effect similar to the approach of exposing and exchanging state information between system layers for coordinated adaptations.

Power optimization techniques developed at single level, incognizant of the other abstraction hierarchy levels, potentially misses opportunities for substantial improvements achievable through cross-level integration. One of the primary arguments for a cross layer approach was the realization that advances in battery and hardware technology cannot, by themselves, meet the energy demands of next generation mobile computers, i.e. higher levels of the system must also be involved [KWW02], [E99]. The cumulative power gains achievable by coordinating techniques at each stage can be potentially significant; but this requires a study of the trade-offs involved and the customizations required for unified operation. [E99] presents a case for higher level power management and states that applications/middleware can contribute significantly towards energy savings in smaller devices.

In this chapter, we argue for a middleware based approach whose goal is to control and direct the middleware component workload on a system to effect energy gains. For example, feedback from on-device middleware environment can complement the advantages of DVS resulting in higher energy gains.. [WWD94] compares a contrasted the characteristics of wide-area and embedded applications that influence the design of quality object middleware. [ABE00, ATS06, BC00, BSP03, GNM03, HMV05, PAL09] present a principled approach towards supporting adaptation by using reflection, in a component based middleware framework. Other techniques from compilers have been used for power-aware mobile applications [KHR03]. The GRACE project [ANJ02, ZEL03, YN04] proposed the use of cross-layer adaptations for dynamic CPU optimizations. They suggest both coarse grained and fine grained tuning of the CPU through global co-ordination and local adaptation of hardware, OS and application layers. In [YN00], a middleware framework for integrating soft real-time scheduling and DVS is presented. The idea is to apply DVS as far as possible, but while meeting resource reservation requirements of soft-real-time applications

Our approach is realized in DYNAMO [MDN07], a cross-layer middleware framework for evaluating power/performance/ QoS tradeoffs, we adopt the strategy of exchanging state information dynamically at all system levels of the system hierarchy (architecture, network, OS, middleware, applications) and driving simultaneous adaptations at each layer based on these exchanges for optimized performance and energy benefits. This work targets coordinated optimizations for all the three most power hungry components of a mobile system - the cpu, the LCD display and the wireless network interface card. The other key feature of the DYNAMO framework is that it utilizes intermediate servers in close proximity of the mobile device to perform end-to-end adaptations such as admission control, intelligent network transmission and dynamic video transcoding. The knowledge of these adaptations are then used to drive “on device” adaptations, which include CPU voltage scaling through OS based

soft real-time scheduling, LCD backlight intensity adaptation and network card power management. Our experimental results show that such joint adaptations can result in energy savings as high as 54% over the case where no optimization is used while substantially enhancing the user experience on hand-held systems.

The idea of task offloading onto a remote machine is not new [OH98, RRP99]; [RRP99] shows that the task offloading can deliver significant energy savings over noiseless wireless network channels, while the gains are offset over noisy communication channels. More recently, there have been efforts exploring the use of cloud computing to offload tasks [GRJ09, CM09, CBC10]; the use of proxies (or near clouds) to aid mobile devices is being explored. In Clone cloud [CM09] and [GRJ09] a dynamic task partition scheme is presented for offloading application subtasks onto a remote machine or virtual machine for achieving optimal performance according to the memory, and processing time. MAUI [CBC10] uses the same technique for achieving the optimal power management and extending the portable device battery life. In the following section, we discuss such a proxy-based approach to effect power efficiency for mobile applications and formulate a methodology for addressing computation and communication tradeoffs in such architecture.

3 A Proxy Based Approach to Power Aware Computing

A typical distributed system architecture using low power devices is envisioned in Figure 1. The model environment consists of several distributed servers (service providers), proxy servers, meta-data repositories (e.g. directory service) and mobile clients. The servers are high-end machines on the network that provide a variety of services (e.g. streaming video, web services). A set of proxy servers are available on the network and are used to perform an assortment of tasks. For example, a server can replicate data onto a proxy for load balancing and a client can offload expensive tasks onto a proxy in order to con-serve local resources. End clients are mobile devices that communicate with other entities via an access points (e.g base station) in their geographical proximity. The mobile clients route their requests to the servers through a proxy server associated with its base station. Depending on the system design, a device may either be bound to a proxy permanently, or it might disassociate and reconnect to another (possibly nearer) proxy. In the latter case, a proper hand of state information might be necessary between the proxies. For simplicity, we currently bind a device to a pre-determined proxy. The directory service stores the overall system state information and forms an important focal point for storing and retrieving data efficiently. The architecture is further bolstered by the presence of a high-end server called the “power broker”. The power broker can be looked upon as the crux of the system, making intelligent decisions and adaptations based on either the global system energy state or individual device power states. In a large scale distributed environment, there would be many such brokers distributed over the network. In PARM, the power broker determines the set of middleware components that can be offloaded from a device onto a proxy, to minimize the energy consumption at the device. For simplicity, we assume that the client is bound to a proxy. A more complicated scenario would include a mobility model for the clients and the state hand-o s involved therein. Additionally, we define policies which dictate the instants (when), as well as the frequency (how often) at which the broker reconfigures the devices. Based on the current system state, information from the device/proxy and one or more PARM policies, the broker runs the PARM algorithm to determine a new configuration (if any) for reassignment of middleware components between the device and proxy. All low-power devices update their current state information (e.g. running applications/middleware components, residual power, mobility information etc.) periodically to the power broker. An optimal algorithm at the broker would ensure the most beneficial configuration of components at the device and hence maximal energy savings. For the rest of the paper, we refer to any battery operated computer as a device and use the terms “components” and “services” interchangeable to represent middleware services.

The implicit role of middleware in dealing with system heterogeneity, scalability, resource management etc. makes it ideal for the above infrastructure, as it effectively shields applications from the intricacies of the underlying distribution. To make middleware technology more hard-hitting on low power devices, it has to be both expeditious as well as energy efficient. The next section presents a middleware framework that can contribute to significant power downsizes on low power devices. For the rest of the paper, we refer to any battery operated computer as a device and use the terms “components” and “services” interchangeably to represent middleware services.

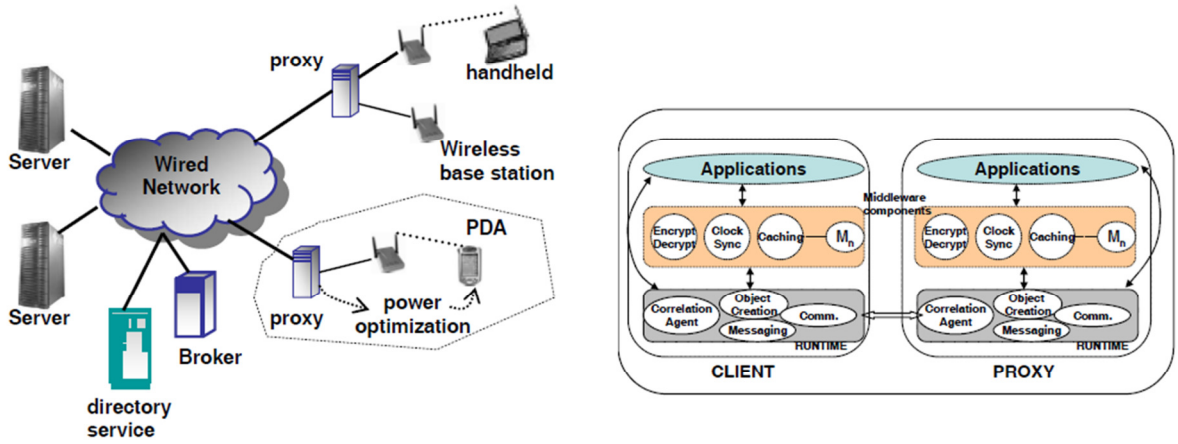

Figure 1 : System Architecture and PARM Framework

3.1 PARM: A Power Aware Reflective Middleware Framework

In this section, we present a flexible, reflective middleware framework that is suitable for low power devices operating in distributed environments. The goal is to realize a middleware solution that is implicitly capable of performing power optimizations. We purport that application based adaptation in conjunction with our middleware, can further substantiate the gains achieved implicitly through the middleware. Therefore, guidelines can be established for applications developed for the middleware, such that maximum power benefits can be achieved. Figure 1 gives a succinct depiction of the PARM middleware framework driving the applications on any computer in the system. PARM applications are developed using a set of programming interfaces exported by the middleware. Each application can either be stand alone or could be a collection of separate tasks that collectively constitute the application. The middleware framework comprises of a core runtime that is a part of every system within the environment. The core runtime services include the skeletal constructs for object creation, simple message passing and a communication framework for message routing. Any application can be developed by using these simple components. A "correlation agent" is included with the core runtime and performs various adaptations and housekeeping activities, while coordinating the various middleware components.

A set of "enrichment" middleware components are provided to impart a fuller abstraction level that enable applications to achieve more complex tasks through a simple interface. These components are independent and can be transparently plugged into the runtime by a system level entity. The middleware carries the burden of seamlessly adapting to the changing environment and providing an acceptable QoS to the application. For example, the core middleware runtime does not include a real-time scheduling service. A soft real-time application (e.g video streaming) requiring a certain level of QoS may not be able to meet its resource/timing demands using the skeletal services. On the other hand, a video application linked to the soft real time scheduler provided by the framework, would be guaranteed certain resources on the machine by the scheduler component and would be able to meet its deadlines. In the process, certain extra amount of work is done by the middleware and this might tally to a considerable cost in terms of energy consumed at the devices. Similarly, the other enrichment components carry out complex activities that would incur some energy cost on low power devices. We target these components as possible candidates for energy optimizations. Our approach would enable low power devices to make use of these services (whenever possible) while cutting down on the energy costs.

By "reconfigurable middleware framework", we advert to a middleware model in which the enrichment components (not the runtime) can be dynamically started and stopped/migrated by some meta-entity, without a effecting the execution of the other components. Middleware services can be discontinued on a device in a number of ways. Whenever it is not feasible to migrate a service we can decide to either stop the service locally or degrade the service, such that it does less work, albeit at a loss of performance but with power savings. Otherwise, we can move the middleware component to a proxy (with an abundance of resources) and simply use the results of the remote execution locally. In the process, a communication overhead is incurred that includes the cost of transferring state information and responses from the proxy. Additionally, offloading/stopping the individual components should be

transparent to the applications. In PARM this is achieved by leaving a component stub on the local machine as the component migrates to the proxy. The stub provides the transparency to the application and handles all the communication with the now remotely executing component. The light weight component stub is designed to have a very small computation overhead; it has an existence outside of the component and can be considered to be a first class object.

In most low-power devices, CPU operations and the network communication result in the highest energy costs. An optimization of the energy consumption costs of the CPU and communication would therefore optimize the overall energy consumption of the device. For simplicity, we model the energy costs for display (LCD, front-light) other than the CPU and the network adapter, as a constant fraction of the initial available energy of the device.

The basic premise of using a reconfigurable middleware framework is that auxiliary services can be added and stopped by a meta-level middleware entity, while maintaining transparency to the applications. A power efficient model could use this capability to its advantage by stopping redundant components and offloading expensive components to a proxy, which has both the power and the resources to execute the components. The problem then becomes that of identifying the components that can be migrated away from the device, such that the energy costs of computation and communication at the device are minimized. This section describes the mechanics of the PARM framework and presents an optimal algorithm for component distribution by modeling the PARM component reconfiguration problem as a source parametric flow network.

As the residual power on the device diminishes, executing the integrated framework of all the applications and components rapidly reduces the remaining time of service for the device. The PARM framework addresses this issue by dynamically tailoring itself in a manner such that the device remains functional for as long as possible. This is achieved by moving expensive middleware services transparently to a proxy server and using the residual power of the device as a principal factor in determining the component distribution. Furthermore, the PARM framework dynamically adapts to the decreasing residual power at the device, by trying to push more and more components to the proxy as the power on the device decreases. This is achieved by inflating the costs associated for executing the components on a device, causing the middleware to re-evaluate the distribution and making appropriate alterations (possibly choosing to assign the now expensive component to execute at the proxy). In the following sections we introduce the parameters used in characterizing the problem and present an optimal graph theoretic algorithm for ascertaining the component distribution.

4 Characterizing Computation vs. Communication in PARM

We use the following parameters to characterize the energy costs at the device for the various computation and communication operations. It is important to note that all energy costs are incurred at the device. The values are quantified by careful experimentation and profiling. Let BW_t and BW_r be the maximum bandwidth available to the device for transmitting and receiving data respectively. $P_{transmit}$ and P_{recv} are the average power consumption rates at the device while it transmits and receives data. $P_{runtime}$ represents the power consumption rate of the PARM runtime. Let T_i be the length of the i^{th} time interval (i.e. time interval between 2 consecutive executions of the PARM algorithm at the broker and let R_i represent the residual power on the device after the i^{th} time interval.$Size_t^k$ and $Size_r^k$ are the average are the average sizes of the messages transmitted and received respectively, by the k^{th}middleware component. We characterize the some of the other parameters as follows:

- PC_k: Average rate at which the k^{th} component consumes power due to computation.
- PS_k: Average rate at which the k^{th} component stub consumes power.
- NS_{di}^k: Average number of messages transmitted by the k^{th} component during the i^{th}time interval, when component is executing at the device.
- NR_{di}^k: Average number of messages received by the k^{th} component during the i^{th}time interval, when component is executing at the device.

- NS_{pi}^{k}: Average number of messages transmitted by the k^{th} component during the i^{th}time interval, when component is executing at the proxy.

- NR_{pi}^{k}: Average number of messages received by the k^{th}component during the i^{th}time interval, when component is executing at the proxy.

Using the above characterization, we derive the following energy costs for computation and communication.

1. Computation cost:

   - when middleware component "k" executes on the proxy during time interval "i"

$$EC_{proxy}^{k} = PS_k \times T_i$$

   - when middleware component "k" executes on the device during time interval "i"

$$EC_{device}^{k} = E_0 + \lambda . T_{const}$$

   - where,

$$E_0 = PC_k \times T_i$$

$$\lambda = \frac{1}{R_i}\ if\ R_{i-1} - R_i > 0$$

$$= \frac{1}{R_{max}} (otherwise)$$

$$T_{const} = some\ scaling\ factor\ determined\ from\ profiling\ information$$

2. Communication cost:

   - when middleware component "k" executes on the proxy during time interval "i"

$$CC_{proxy}^{k} = \frac{NS_{p_i}^{k} \times Size_t^k}{BW_r} \times P_{recv} + \frac{NR_{p_i}^{k} \times Size_r^k}{BW_t} \times P_{transmit}$$

   - when middleware component "k" executes on the device during time interval "i"

$$CC_{device}^{k} = \frac{NS_{d_i}^{k} \times Size_t^k}{BW_r} \times P_{transmit} + \frac{NR_{d_i}^{k} \times Size_r^k}{BW_t} \times P_{recv}$$

In order to optimize the energy in the system, we now have to find an allocation scheme that distributes the components between the device and proxy such that the overall energy cost at the device is minimized. Let U be the entire set of components that we consider. Let X be the set of components mapped to the device. Then $U - X$ is set of components mapped to the proxy. Our problem now becomes that of minimizing the computation and the communication costs at each interval. For each T we need to minimize the following term:

$$\sum_{k \in X} EC_{device}^{k} + \sum_{k \in X} CC_{device}^{k} + \sum_{k \in U-X} EC_{proxy}^{k} + \sum_{k \in U-X} CC_{proxy}^{k}$$

A Network Flow Representation of the PARM problem:

To achieve the optimal distribution of components between the device and the proxy, we cast our problem as a source parametric flow graph problem [AMO93]. The minimum cut of the flow graph then gives us an optimal mapping of components. We incorporate the energy costs of network communication and CPU computation into an energy flow network. To create the flow network (Figure 2), *we distinguish two special nodes in the network: a source node D and a sink node P.* Additionally, we define two conceptual nodes R_d and R_p, that represent the core runtime frameworks on device and the proxy respectively (Figure 2). We associate the source node (D) with the low-power device and the sink node (P) with a proxy. In Figure 2, B_d and A_p represent the energy costs of the parm runtime executing at the device and the proxy respectively. All the other nodes in the graph correspond to the PARM middleware componentsM_i . The arc capacities are assigned as follows: Each $A_i$denotes the energy costs of computation incurred at the device, when component M_i is executing at the proxy.

In the PARM framework it is the cost of execution of the component stub at the device. B_i denotes the energy cost of computation when component $M_i$executes at the device. Each B_i is defined as a non-decreasing linear function of the residual energy on the device. This makes the flow graph a source parametric graph, where the computation cost capacity of every source arc (B_i) increases with time. X_i represents the energy cost of network communication when component M_i is executing at the device and sending/receiving data to/from the proxy. Y_i represents the energy cost of network communication when component M_i is executing at the proxy and sending/receiving data to/from the device. The device runtime (R_d) is bound to the device (D) by giving assigning infinite energy cost to the arc from D to R_d. The proxy runtime is similarly bound to the proxy.

With this representation, there now exists a one to one correspondence between a minimum cut of the graph and the assignment of components to the source (device) and the sink (proxy). Let P_1 and P_2 be the assignment of components to the device and the proxy respectively. The minimum cut corresponding to this assignment is then $([\{D, R_d\} \cup P_1], \{P, R_p\} \cup P_2)$. The cut of the graph e ectively represents the minimum energy cost assignment of the components to the device and the proxy. Moreover, as the costs of the arcs B_i increase (residual energy at the device reduces), the minimum-cut would tend to exclude these arcs, thereby assigning more and more components to the proxy. In our framework, the energy costs of executing the components on the device (B_i) are represented as non-decreasing linear functions of the residual power on the device. As the power drains from the device, the cost of executing components on the device increases. The PARM algorithm now tends to push more and more components towards the proxy. We solve the above problem by using a modified FIFO pre-flow push algorithm, as described in the next section. Our algorithm has a worst case execution time of $O(n^3)$.

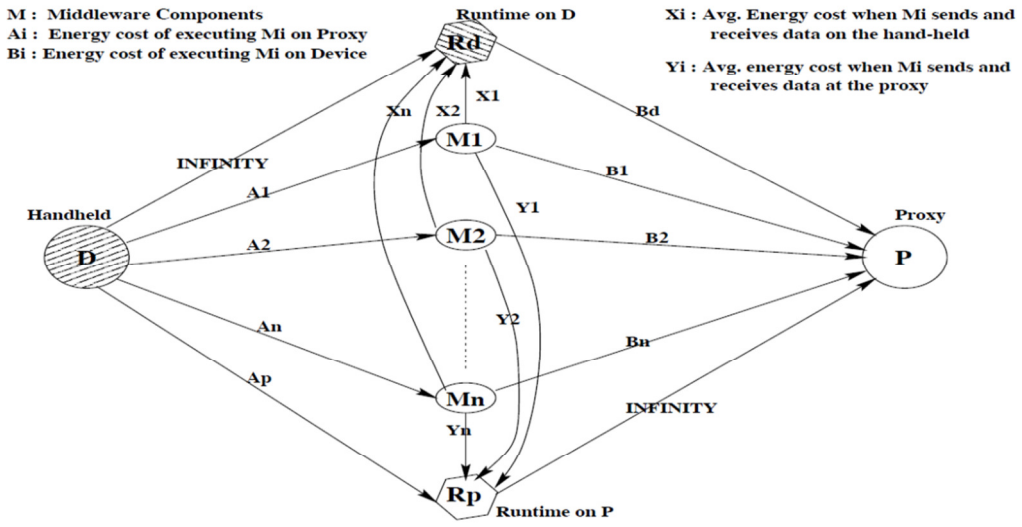

Figure 2 : Flow Network

4.1 The PARM Algorithm:

We now describe the PARAM reconfiguration algorithm (Figure 3), which uses a modified FIFO pre-flow push algorithm [AMO93], generally applied to solve maximum flow problems. The reconfiguration algorithm forms an integral part of the PARAM algorithm and determines the minimum-cut of the flow graph created in Figure 3, by first resolving the maximum flow of the graph. From the above flow graph, we construct a residual graph that tells us how much more flow (energy cost) can be sent along each arc (edge) of the graph. In effect it represents the residual energy flow capacities of each arc in the graph. For every unit of flow sent along an arc, a reverse arc is added to the residual graph with the same number of units. The algorithm is initially executed on the residual graph of the graph initially generated and subsequently, the algorithm works on the residual graph generated at the previous step. The main concept behind the algorithm is to initially push the maximum possible flow from the device (node *D*) to the proxy (node **P**, see PARM-INITIALIZE() in Figure 3). Depending on the capacities of the Intermediate $arcs(A_i, X_i, B_i, Y_i)$, the maximum allowable flow gets routed to the proxy (*P*). Once a saturation point is reached at the intermediate nodes M_i (i.e. the residual graph does not contain any forward arc to the node P), the surplus flow initially sent, flows back to the device (node *D*). At this point, the flow in the network (from *D* to *P*) is the maximum flow between the device and the proxy. However, our interest is restricted to finding the arcs that constitute a minimum cut of the graph. To achieve this we exploit the two phase nature of the pre-flow push algorithm. As stated above, the algorithm first tries to push all the excess energy flow towards the proxy (*P*), till either all the energy reaches it or the intermediate arc capacities are saturated. In the second phase, the algorithm sends the surplus flow (that could not reach node P) back to the source node D. An interesting observation here is that the algorithm typically establishes a maximum pre-flow (end of phase-1) long before it establishes a maximum flow (end of phase-2). In order to get a minimum cut of the graph we keep track of the intermediate nodes (M_i, R_d, R_p)that get saturated (i.e. cannot send any more flow to node P) during the first phase of the algorithm. At the end of the first phase we get a set of nodes that would eventually send their excess flows back to the source node (D). The minimum-cut would partition the graph such that these nodes are grouped with nodes$D$ & R_d. In effect, this is the set of nodes (componentsM_i) that would be assigned to the device. The other intermediate nodes (components) would be assigned to the proxy. Table 1 explains the notation that has been used in PARM algorithm.

n	the number of nodes in the graph
N'	A *SET* of all the components that send their excess flow back to D. The nodes in this set will be assigned to the low-power device.
$F(n1, n2)$	The energy flow from node "n1" to node "n2"
U(n1,n2)	The maximum energy capacity of the arc from node "n1" to node "n2"
$R_{i \to j}$	The residual energy flow available on arc from node "i" to node "j"
excess(n)	The excess flow at node "n". (= flow into "n" − flow out of "n")
Distance Label d(i)	distance of the node i from the sink node (node P)
Admissible arc(i,j)	The arc between node "i" and node "j" is considered an admissible arc if and only if $d(j) = d(i) - 1$.
Active nodes	The nodes in the graph that have a positive *excess*
LIST	A queue that maintains a list of the currently Active nodes

Table 1 : Terms used in the PARM algorithm

PARM Policies:

We investigate the performance of our strategy under a set of policies that dictate when and how often the PARM algorithm is executed for reconfiguring components on a device. The purpose is to determine which policy returns the best results in terms of energy savings at the device and ascertain the optimal times for executing the algorithm

for different classes of applications and components. Note that the PARM algorithm is executed at the power broker based on the device power state and the chosen policy. We categorize the policies based on the host (broker or device) that triggers the reconfiguration.

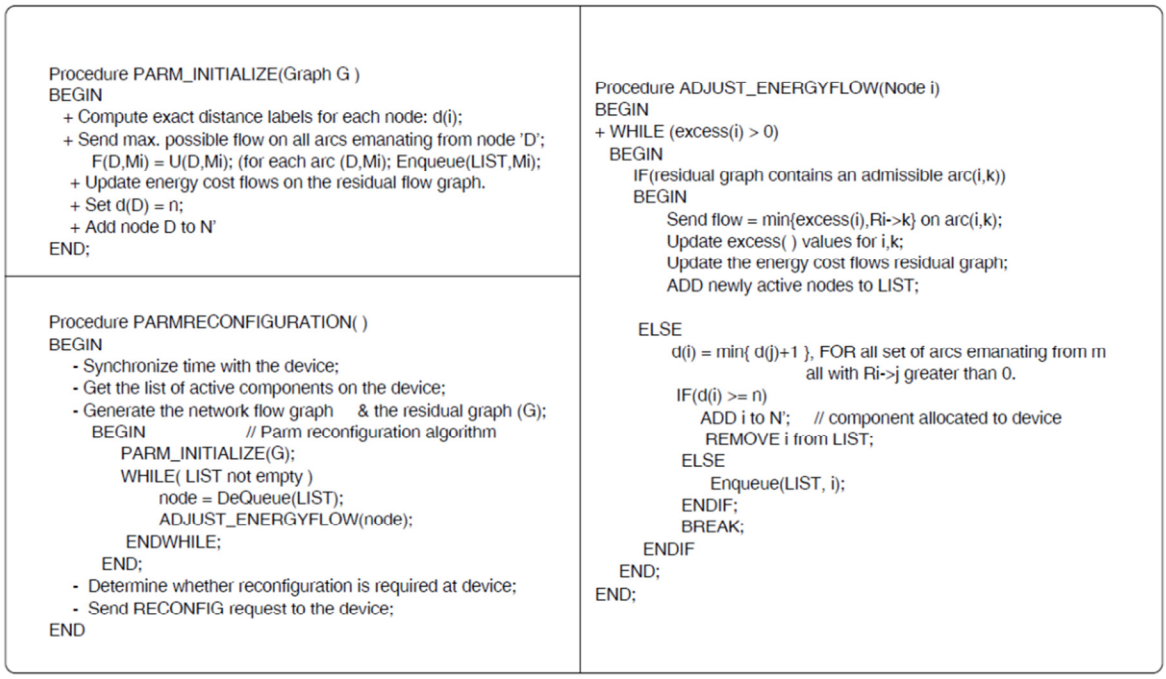

Figure 3 : The PARM Algorithm

- *Random*: The broker performs the component reconfigurations at random intervals. The importance of having a random trigger policy is that often randomly chosen policies do an excellent job in dealing with complex and contentious test cases, outperforming more intuitive policies. Moreover, the results from the random trigger policy provide an interesting case for comparing the performances of more intuitive and predictable policies.
- *Periodic*: The broker performs the component reconfigurations at periodic intervals determined by the system administrator. It solves the following purpose: given a group of applications and a set of associated middleware components, we use this policy to learn how periodic execution of the PARM algorithm at the broker impacts the power savings at the device. Moreover, it tells us how often the reconfiguration of components needs to happen in order to obtain optimal gains. These results are of a more predictable nature and can be cataloged for future reference, if the outcomes are favorable. The following two policies are device driven:
- *Application Triggered*: The PARM algorithm is triggered at the broker, whenever a new application/component is started up at the device. The device sends a signal to the broker, triggering the reconfiguration algorithm. This is a "reactive" policy, where the system responds to the change in load by affecting the reconfiguration algorithm to execute. Using this policy, we determine whether this causative behavior is more prolific (in terms of power savings) than the other "proactive" policies.
- *Threshold*: A reconfiguration is triggered by the device whenever the residual energy of the device drops below a certain threshold, determined as a percentage of the initial energy of the device. A set of threshold values are pre-determined to fix the number of reconfigurations. The idea here is more simple-minded. We choose to reconfigure the components only when we feel that the power level of the device has dropped below a certain boundary and needs attention. The value of the threshold determines the frequency of the reconfigurations.

4.2 Handling Component Migration

We now briefly discuss how component migration is handled within the PARM framework. The interplay between components and applications on the same machine are handled through local messaging and/or IPC mechanisms. The costs of these interactions implicitly get included in the energy costs for CPU operations. We now address the more intricate issue of communication between components/applications when components are moved to the proxy. Note that all services running on the device are replicated on the proxy. So migration costs do not involve transfer of the process image over the network. Consider a common scenario in our approach where a reconfiguration results in moving a component from the device to the proxy. The migrated service is started (if not already running) at the proxy and the state information is transmitted from the device to the proxy. This incurs energy overhead due to communication that needs to be accounted for in our optimization. The component stub is retained at the device and handles all messages received from the remotely executing component and routes them back to the application. Interaction between components on different machines is modeled as communication from one component to the runtime of the other machine. Note here that some middleware components cannot be migrated. In such a case the service is either stopped or degraded. We include the cost of maintaining the state information for the various components within the cost for executing our framework. We identify three different situations for which component migration needs to be addressed:

- *Case A: Application using a single middleware service:* This is the simplest of the three cases where an application on the device is using a single middleware service. As an example, think of a navigator application using a location management component where a GPS system is used to identify its current location. If exact precision is not a necessity for the application, the computationally intensive middle-ware component can be executed on the proxy and the results transmitted to the device, with possible energy savings.
- *Case B: Application using multiple middleware services:* Consider an application that requires to replicate its messages, order them and send them reliably to applications on other hosts. Furthermore, the application might require encrypting messages that are sent to some distant hosts (outside its domain) Each of the above functions are performed separately by distinct “enrichment” middleware components. The flow of the application message through the middleware components is as follows: create replicated messages, tag the messages for ordering, encrypt the messages using some algorithm implemented by the middleware component and then use the reliable messaging service to transmit the message. In such a scenario migrating one or more components to a proxy, might result in a condition where messages are sent back and forth between the device and the proxy. However, if the computation costs are much greater than the communication overhead, this arrangement of the components might still prove to be energy efficient.
- *Case C: Multiple applications using a middleware component:* A more complicated situation arises when there are multiple concurrent applications use a single middleware component. When the components are shared by the applications, moving the component to a remote machine will require individual application related state to be transferred to the remote machine. The component stub now performs more work to maintain and transfer the required data and now has to be modeled as a more power devouring unit. Additionally, if the computation overhead of such a component is not too high, it can be bound to the device and never migrated.

5 Performance Evaluation

In this section, we lay out the details of our simulation model and analyze the performance of our model under various system conditions. We present the results of our experiments and examine the effects of the different policies on power conservation. To simulate our system, we separately model the low-power device, the proxy server, the power Broker, the PARM middleware framework and the applications. The core middleware runtime framework executes on the device, the proxy server and the server running the power broker. Applications are developed using the APIs exported either by the PARM runtime or one or more of the “enrichment” components that are available with the middleware framework. Extensive profiling is used to record the power consumption pattern of individual components and applications. The system model is as follows: the low-power device is modeled after a Compaq iPaq 3650. The proxy and the broker are assumed to run on high-end ethered workstations with substantial resources. Upon startup, the device registers itself with the nearest proxy and the PARM runtime on the device records the residual power of the device, the current set of active applications and components and

updates this information with the directory service. A constant energy cost is incurred for communicating with the directory service, the proxy and the broker. We are currently building a prototype using the CompOSE|Q [VDM01] middleware framework to integrate the PARM reconfiguration algorithm for power optimizations in a distributed environment.

5.1 Experimental Setup

The type of application we choose to execute on the device has a significant bearing on the results of our experiments. We therefore opt for the types of applications that are currently regarded as suitable for hand-held computers and some applications can we think would be popular as the devices evolve. More-over, we target applications that consume appreciable amounts of power either through computation, communication or both. To generalize the experiment a bit further, we divide our set of representative applications into three classes: computation intensive (class-1, e.g Image processing applications, games, 3-D graphics applications), communication intensive (class-2, e.g. web browsing, real-time chat-ting, network monitoring) and both computation and communication intensive (class-3, e.g multimedia streaming, text to speech, GIS/navigation). [CY09] presents the energy consumption characteristics of MPEG video playback on hand-held computers. We borrow the some of the results presented in this paper to model a typical video playback application for our simulation. In [KWW02], power usage rates of some typical computation intensive applications are empirically determined. We draw on some of the results illustrated in these papers to model the power usage patterns of some of our applications. Table 2 illustrates the different power utilization values for a typical application from each of the above classes. The energy consumption patterns of some of the applications and components (e.g message ordering) are obtained from our profiling results as discussed earlier.

We choose typical applications in each category and separately profile the various computation and communication costs for the applications and the associated PARM components. An application typically communicates by either routing messages through a component or by sending independent messages via the runtime, both of which have power overheads. We regard the power consumed by the independent messages as the communication overhead for the application. The cost of messages through the components get accounted for, in the communication costs associated with the component. The costs of computation are recorded separately for each application. For each application, the average rate at which the application sends messages via components, its memory and CPU usage and other details that help in completely representing an application are also registered.

APPLICATION	CLASS 1	CLASS 2	CLASS 3
Components linked	Adaptive Scheduler Encryption-DES Decryption-DES	Reliable Messaging Clock Sync Message Ordering	Adap. Scheduler MM Message Serv. Clock Sync
Avg. Power(J/s)	0.65	0.23	0.38
Avg. Power(J/s)(comm)	0.21 (watts)	0.40	0.45
Avg. message size	64 (bytes)	128 (bytes)	1024 (bytes)
Avg. msgs via component(per/sec)	300,150,150	610,530,480	755,830,670

Table 2 : Application Model

The PARM Component Model:

To model the PARM components, we profile the energy pattern of each component while using it with a different number of applications from each class. In particular, we record the average power consumption rate of the component running on the device, and the power overhead of running the component stub on the device while the component is executing at the proxy. We also store the average number of control messages the component uses to

maintain state information when used in conjunction with different classes of applications. For example, a encryption mechanism that uses the current machine time might require the remote stub to regularly send the time of the remote machine for clock synchronization. To even out the variance introduced by single applications, an average message size per application class is used. We control the communication through a component by varying the size of the messages used as well as the number of messages routed through the component by applications from each class. Here are a few examples of the "enrichment" components that we use for our simulation : Replication services, Message ordering services, Encryption/Decryption services, Reliable messaging service, Scheduling, Clock synchronization, Location management service, Watermarking & User Authentication service.

The Execution Model:

The simulation is carried out in two phases. In the first phase the applications are executed on the device without the PARM algorithm. The energy consumed and the remaining time of service for the device are recorded. In the second phase the same set of applications are executed with the PARM algorithm executing at the broker. The broker implements the PARM algorithm and policies. It retrieves the individual state information for each device from the directory service and determines the optimal reconfiguration of components for the device. It then announces the new configuration to the device and the proxy. The device then initiates a routine that performs the offloading/downloading of the components to/from the proxy, based on the new configuration. The power consumed in the transfer of components is profiled and used as a part of the PARM algorithm. The energy consumed and the remaining time of service of the device are recorded. The proxy server in our simulation is a workstation that simply receives the middleware components from a device and executes them on behalf of the device. In our simulation we have a maximum of 10 applications each of which link to two or more "enrichment" PARM components. For each application class we assess the gains using a set of sporadic-start applications: applications that start and stop irregularly over time. Non-sporadic applications: applications that run continuously till the device runs out of power.

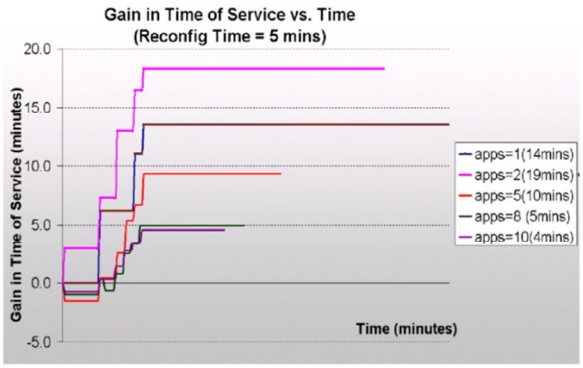

Figure 4 : Effect of applications on TOS (class-1, non-sporadic)

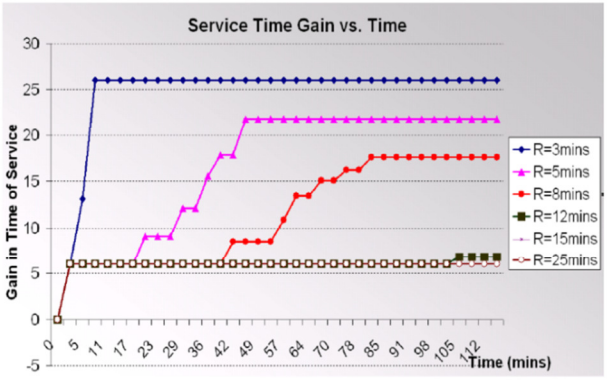

Figure 5 : Effect of Reconfig. time on TOS(class-1,non-sporadic)

5.2 Experimental Results

We analyze the performance of the PARM framework by evaluating its execution under different application loads and reconfiguration times. Our primary metric of evaluation is the gain achieved in the

- *Residual Energy (ER):* showing the unexpended energy left in the device.
- *Remaining Time of Service (TOS):* indicating the remaining time for which the device can be operational, under the current operational load.

By “gain”, we mean the TOS/ER saved by running the applications on the middleware framework with and without the PARM algorithm. When the algorithm is in use, the power broker determines an optimal set of components that need to run on the device. The device then moves the other components to the proxy. We study how the TOS/ER gains are impacted as the number of applications on the device is scaled up. Next we examine how often the broker should reconfigure the PARM components for a given set of applications to achieve optimal energy benefits. Finally we compare the energy savings for each class of applications and ascertain the optimal reconfiguration frequency for achieving the maximum energy profits. A default reconfiguration time of 5 minutes is specified for the simulations.

Performance of Class-1 Applications:

Figure 4 through Figure 9 depict the gains achieved over time for the class-1 applications. As seen in Figure 4, there is a clear improvement in the TOS at the reconfiguration points as the components are moved onto the proxy. The initial loss is due to the fact that the device has not been reconfigured and the overheads for updating the directory service and registering with the proxy and the broker have been incurred. Figure 5 shows the TOS saved for a set of applications as the broker reconfigures the components at different times. Frequent reconfigurations result in higher gains, and as the reconfiguration frequency reduces, the gains drop to almost zero (for reconfigurations at every 30 mins or more). Figure 8 and Figure 9 compare the energy gains achieved over time as the number of applications is scaled for different sets of sporadic and non-sporadic applications. As the reconfigurations occur, a steady gain in residual energy is noted. Moreover, it is noticed that the gains become more pronounced as the number of components used increases.

Figure 6 and Figure 7 display the TOS gains for sporadic-start class-1 applications with different number of applications and reconfiguration times respectively. The curves indicate irregular gains (as compared to the non-sporadic case) at different times, both when applications are scaled as well as when the reconfiguration times are different. The flat regions (const. gain) on the graph are the times between the configurations that resulted in a redistribution of components between the device and the proxy. A comparison between the two cases shows that there is a significant energy saving as we scale the number of applications.

Performance of Class-2 Applications:

Figure 10 and Figure 11 demonstrate the TOS and the energy gains respectively for non-sporadic class-2 applications as the number of applications scale from 1 to 10. The broker executes the PARM algorithm every 3 minutes. The graphs indicate that the maximum TOS gain is achieved when the number of applications is small. A comparison with the corresponding class-1 graphs indicate that most the gains achieved in this case is due to an initial reconfiguration of components (dependent on the nature of the components and applications started) after which the gains become steady. The explanation for this is that most of the energy overheads in this case is due to the communication. The increase in the cost of the components with time does not seem to result in frequent reconfigurations as the computation costs are small and the device runs out of power before the costs become high enough to cause a reconfiguration. The higher gain when compared to the class-1 case is due to the different energy model used for the applications and components in this application class. It is also observed that the there is a drop in the gains towards the end when the number of applications are increased. This is caused due to the overhead of reconfiguration when the device power is depleting and happens sooner when a greater number of applications are executing.

The next two graphs (Figure 12 and Figure 13) show the TOS gains for sporadic start applications. As in the case of the class-1 non-sporadic applications an irregularity in the gains is observed. However, the gains in this case are more than the class-1 sporadic applications. The dips in the curve in Figure 12 could be a result of a reconfiguration or an increase in the communication resulting in an additional overhead at the device.

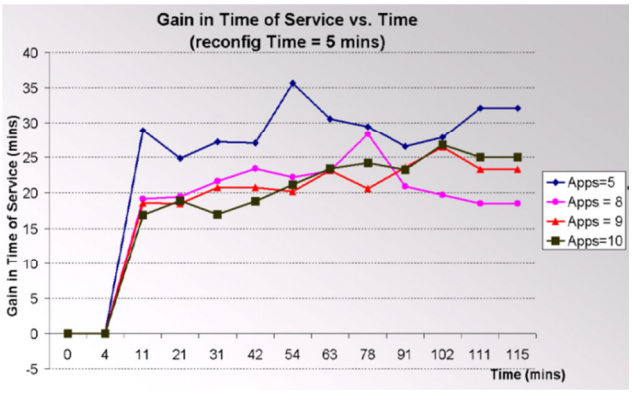

Figure 6 : Effect of applications on TOS(class-1,sporadic)

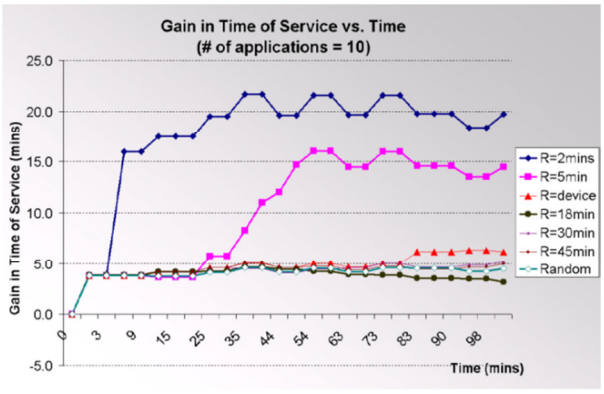

Figure 7 : Effect of Reconfig.time on TOS(class-1,sporadic)

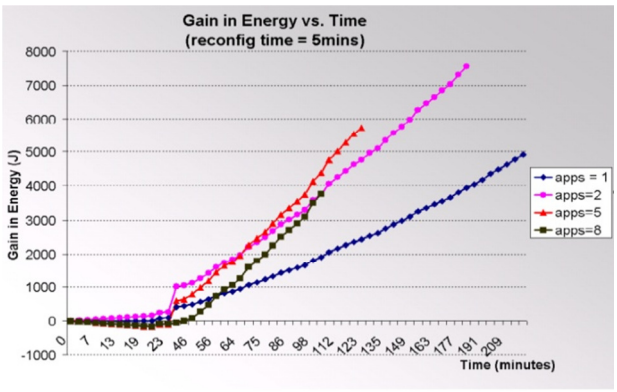

Figure 8 : Effect of applications on ER(class-1,nonsporadic)

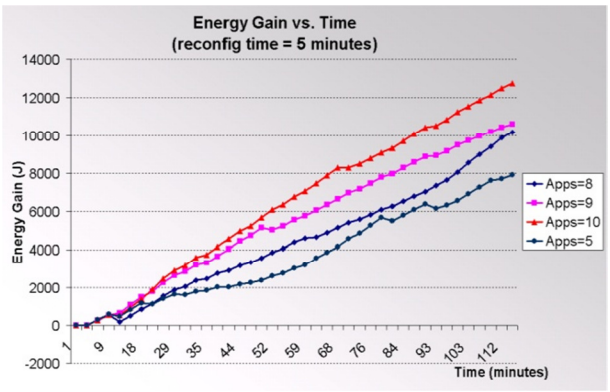

Figure 9 : Effect of applications on ER(class-1,sporadic)

Performance of Class-3 Applications:

Class-3 applications present a more realistic representation of the workload in hand-held devices. Most multimedia applications fall into this category. Owing to the resource scarcity in small devices, a number of applications that use moderate communication and computation would also fall into this category for smaller devices. The modeling of the components and applications for this application class is different from the models used in the previous cases. Figure 14 and Figure 15 show the graphs for TOS and ER gains as we scale the non-sporadic start applications for this class. As seen from the graphs, there is a reduction in the TOS and energy gain as we scale the number of applications. An interesting observation is that with a single application, gains stay low (Figure 14), showing that reconfigurations with a small number of components does not result in significant gains. Figure 15 indicates that the gains do not change significantly as the numbers of applications are increased. A comparison with the class-1 and class-2 applications reveals that the gains in this case are not as pronounced as in the previous cases. This is because both the communication and the computation costs now play a role in determining the component reconfiguration and it becomes hard to predict the outcome of a reconfiguration. One reason for the modest gains could be that the energy saved by offloading components is slightly o set by the cost of communication.

Figure 16 through Figure 19 show the TOS/ER gains achieved for the sporadic start class-3 applications. As seen in Figure 16, the TOS gain for these applications is jerky. The TOS gain decreases as the number of applications scale up. Figure 17 presents the energy gains as we scale the number of applications. The energy gains increase as we increase the number of applications, but become unpredictable when the number of applications are close. A comparison indicates that with class-3 applications, the changes are smoother and less frequent than the corresponding class-1 and class-2 applications. Moreover, the gains are also less as compared to the other two cases.

In this class, both middleware components and the applications consume substantial energy. Since, only the middleware components are considered for migration, the device drains out of power very quickly showing smaller gains. A possible inference is the with class-3 applications, application migration can significantly improve energy gains.

Figure 18 and Figure 19 represent the TOS and ER gains achieved over 6-sporadic start class-3 applications as the PARM algorithm is executed at different times. An interesting observation here is that there is a negative energy gain as we execute the algorithm at a frequency of over 8 minutes. A steady energy gain is observed when the algorithm is executed every 3 minutes. However, all the other cases seem to result in an overall loss in both TOS and saved energy. The reason for this is that the broker determines the distribution of components assuming that the applications (and the linked components) would be active over the next time interval (till the broker executes the reconfiguration algorithm again). However, in the sporadic-start case, the unpredictability of the timings of the application start/stop times seems two set the effectiveness of the PARM algorithm resulting in a loss in this particular case. This argument is supported by the fact that there is a gain when the reconfiguration frequency is high (3 mins), where the random nature of the application start/stop times has the least effect.

Comparative performance of the 3-Application classes:

We compared the impact of the periodic broker-triggered reconfiguration and the device-triggered thresh-old reconfiguration policies for the three classes of applications, for both sporadic start and non-sporadic start applications. Figure 20and Figure 21 compare the impact of different reconfiguration times (3mins, 10mins) for each class of applications. Both graphs show improvements in TOS gains when the reconfiguration occurs more frequently. Figure 22 and Figure 23illustrate that a simple-minded threshold triggered reconfiguration performs significantly worse than the periodic reconfiguration, especially for class-2 and class-3 applications. Class-3 applications show a negative gain for both the sporadic and non-sporadic start cases. For non-sporadic class-3 applications, the threshold triggered reconfiguration performs similar to case where reconfigurations occur every 10 mins (Figure 19) or more. We believe that the results will show marked improvements if the thresholds were set to trigger reconfigurations more frequently. Figure 24 and Figure 25 study the impact of reconfiguration time on a mixed workload of applications. It is observed that the reconfiguration time has little impact on the particular selection of applications. Only a reconfiguration every 3 minutes provides a noticeable gain.

We studied the impact of scaling the number of applications and varying the rate of execution of our proposed algorithm, on the gain in time of service and residual energy of a hand-held device, for three classes of applications. We observe that for class-1 applications, maximum service time gains of 15%-25% was achieved when the PARM algorithm was executed frequently. However, the gains were negligible when the algorithm executed once every 10 minutes or later. In case of class-2 applications the gains were more moderate than the class-1 applications. On an average there was a 7%-25% increase in the time of service of the device. In the case of the class-3 applications both the computation and communication costs played a significant role in determining the reconfiguration of components. The gains followed a pattern similar to the gains in class-2 applications, but were more moderate as the applications drained the device power faster than the other two classes. Interestingly, it was observed that the gains in the case of sporadic start applications in this class were negative when the reconfigurations happened less frequently (once every 8 minutes or more). A reconfiguration rate of around 3 minutes was found to give a significant gain in both energy and time of service. Our experiments illustrated that a simplistic device-triggered threshold driven policy performed inferior to the broker driven periodic reconfiguration, especially for class-2 and class-3 applications.

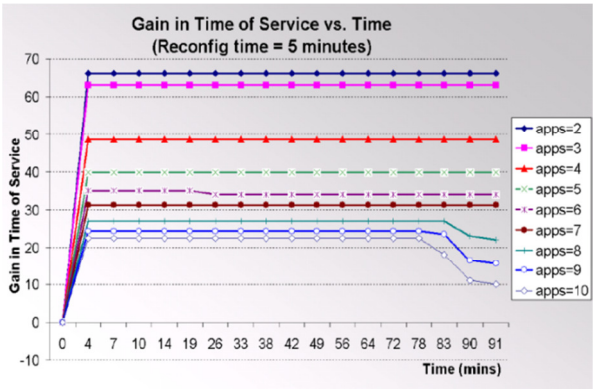

Figure 10 : Effect of applications on TOS(class-2,nonsporadic)

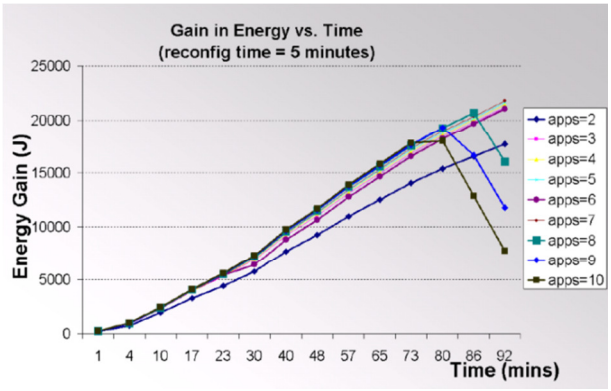

Figure 11 : Effect of applications on ER(class-2,nonsporadic)

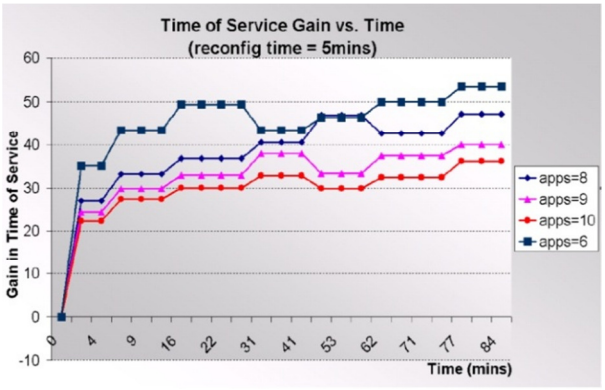

Figure 12 : Effect of applications on TOS(class-2,sporadic)

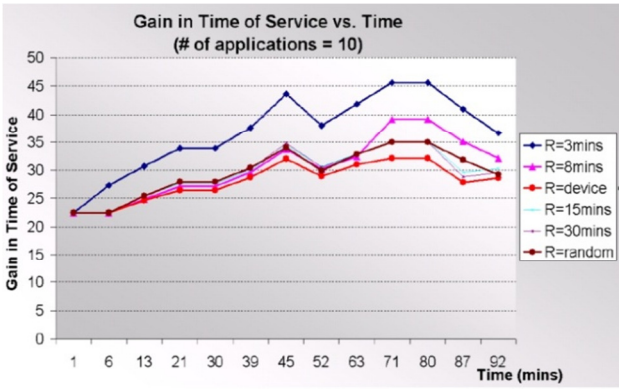

Figure 13 : Effect of Reconfig.time on TOS(class-2,sporadic)

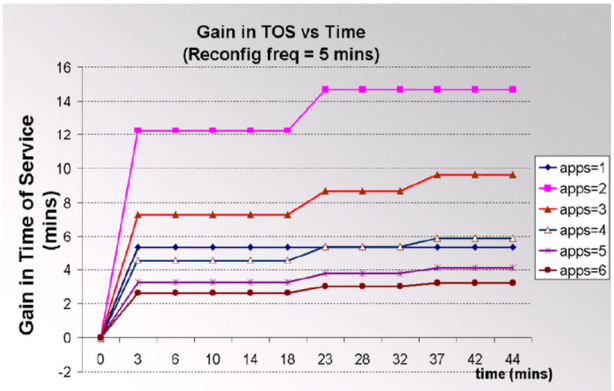

Figure 14 : Effect of applications on TOS(class-3,nonsporadic)

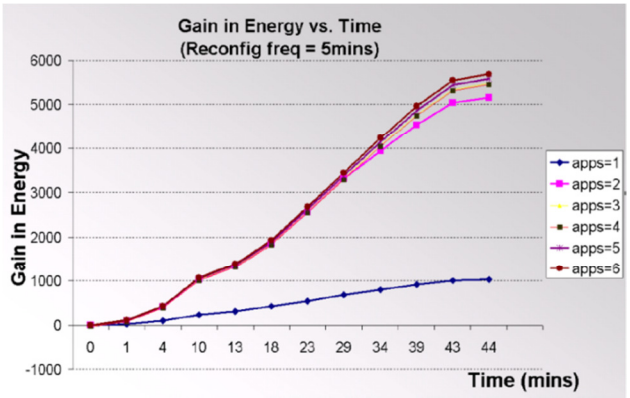

Figure 15 : Effect of applications on ER(class-3,nonsporadic)

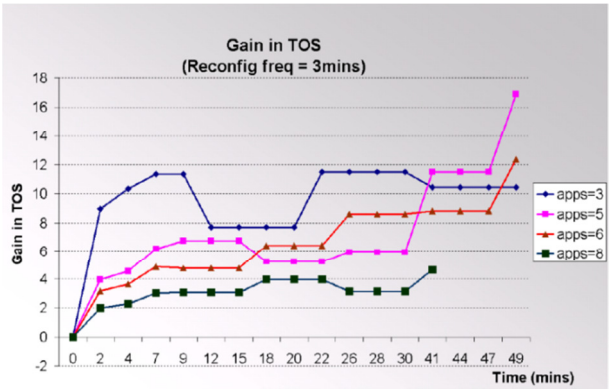

Figure 16 : Effect of applications on TOS(class-3,sporadic)

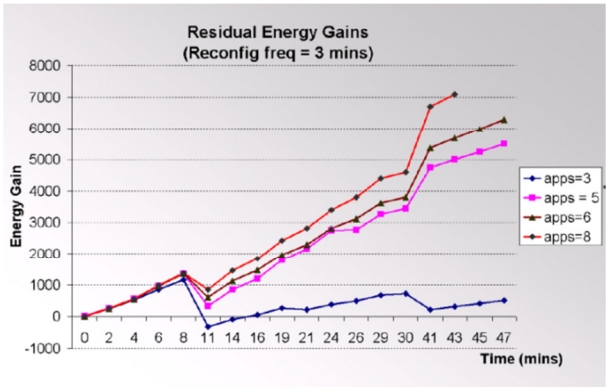

Figure 17 : Effect of applications on ER(class-3,sporadic)

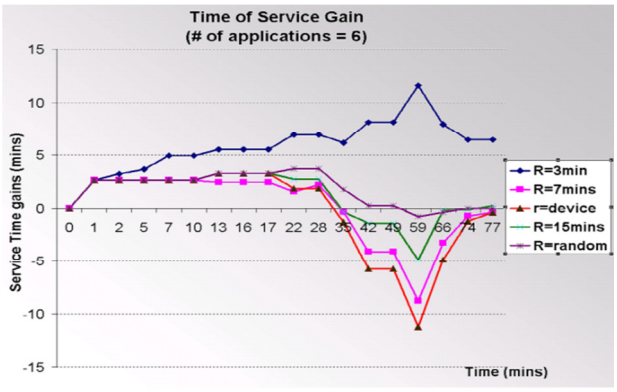

Figure 18 : Effect of Reconfig.Time on TOS(class-3,sporadic)

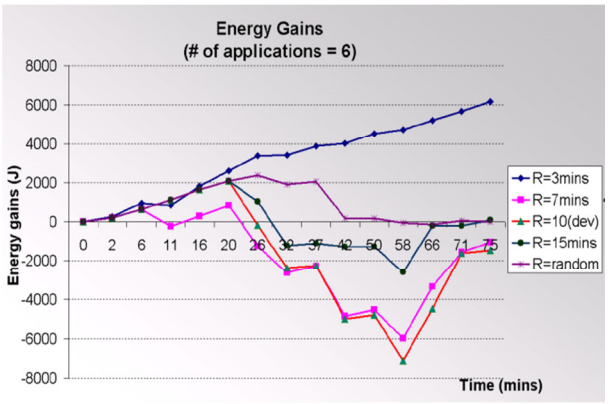

Figure 19 : Effect of Reconfig.Time on ER(class-3,sporadic)

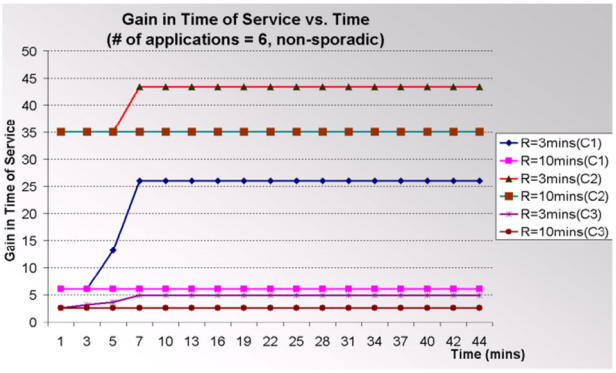

Figure 20 : Impact of Reconfig.Time on TOS

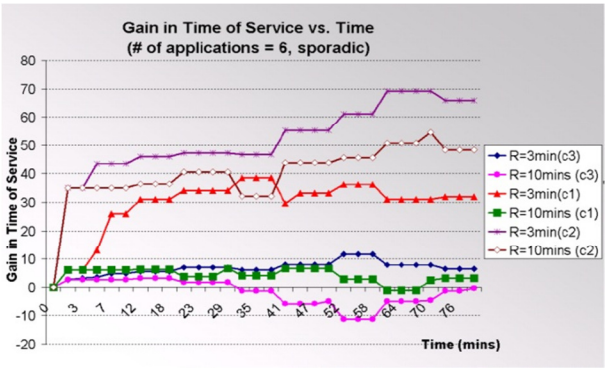

Figure 21 : Impact of Reconfig.Time on TOS

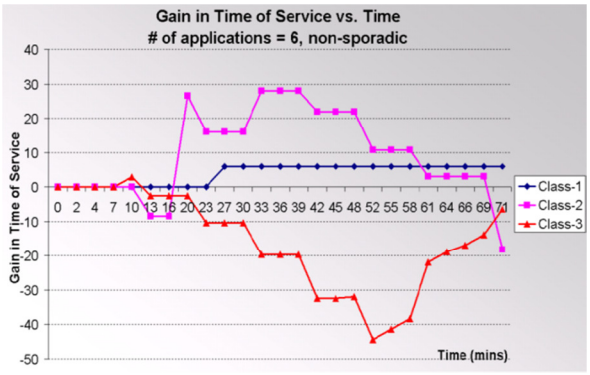

Figure 22 : Threshold triggered Reconfig. (non-sporadic)

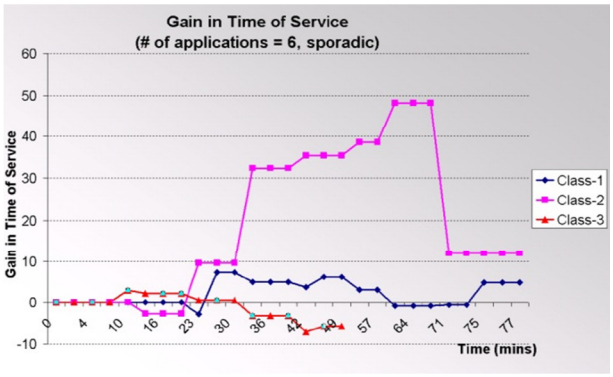

Figure 23 : Threshold triggered Reconfig. (sporadic)

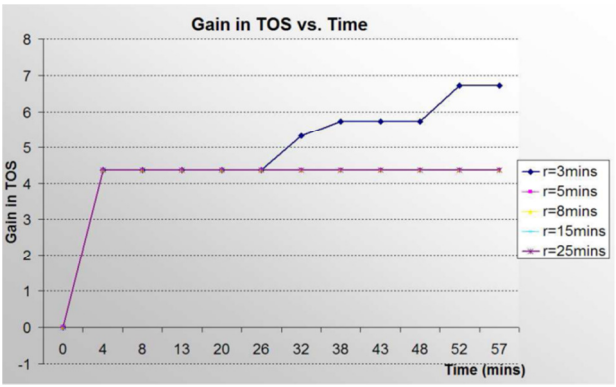

Figure 24 : Impact of Reconfig.Time on TOS(non-sporadic)

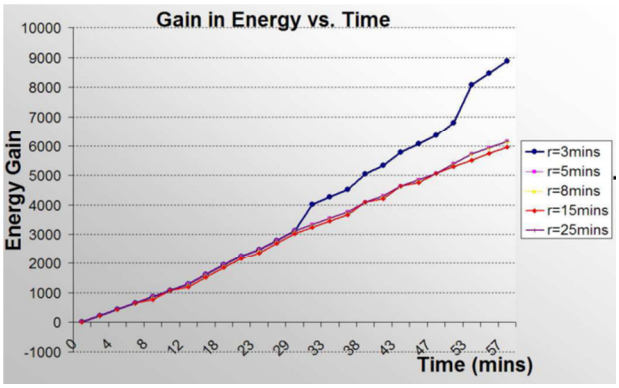

Figure 25 : Impact of Reconfig.Time on Residual Energy

5.3 Prototype Middleware Framework

We are currently implementing a prototype of the PARM model using the CompOSE|Q middleware framework developed by us. CompOSE|Q is a message-oriented reflective middleware framework that supports a concurrent active object model. It provides mechanisms to dynamically start and stop middleware components and has an application program interface that can be used to develop applications. The entire runtime is implemented using Java and the LDAP directory service interface. A detailed discussion of the design and implementation of the CompOSE|Q runtime and PARM model is outside the scope of this paper. So we simply present the outline of the details. Fig. 29 shows the integration of the PARM framework with the existing architecture. The core runtime consists of a node-manager that administrates the runtime components and performs functions like object creation, bootstrapping of the various meta-level runtime modules and communication with remote middleware components. The runtime has a skeletal communication framework that includes the Router, the Postman and the message receiver modules. The PARM specific runtime agents are the co-relation agent, the signaling agent, the registration agent and the reconfiguration agent. Each PARM middleware component contains a stub through which all communication between the component and the runtime is routed. The stub always executes locally and provides the transparency to the applications when components are migrated. The co-relation agent manages the individual component stubs on the device when the components are moved to the proxy. Incoming messages are also routed to the individual applications using the co-relation agent. The agent identifies the component stub for which the messages are intended and delivers the messages to the stub. The stub then submits the messages to the appropriate

applications. The runtime generates unique identifiers for each middleware component and the components are designed to keep track of applications that link to them. The signaling agent is used in PARM to signal either the broker or the proxy whenever required

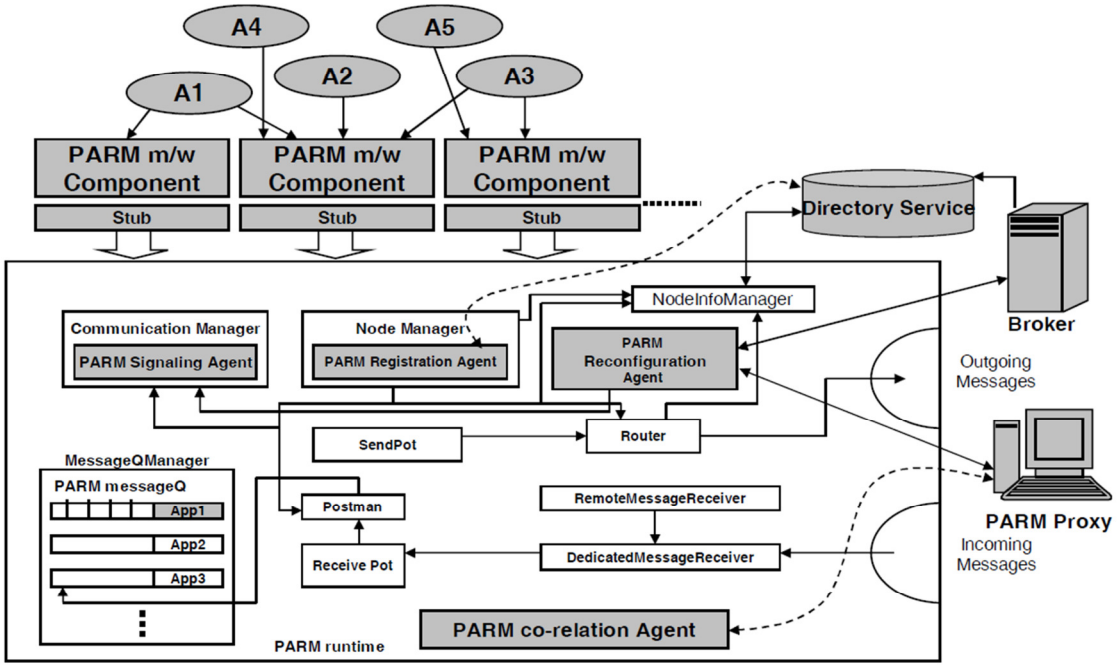

Figure 26 : PARM Prototype Implementation

For example, a signal can be sent to the broker to execute the PARM reconfiguration algorithm, every time new application is started at the device. A registration agent is used to register device state information with the directory service. At startup, the device registers itself with the nearest proxy using this agent. The registration agent can also be used to recover state information after abnormal terminations. The PARM reconfiguration agent forms the heart of the PARM runtime. It regularly receives updates from the broker and in conjunction with the co-relation agent manages the migration of PARM components (and possibly applications) to and from the proxy. Moreover, it maintains a list of current applications and components and actively monitors the residual energy of the device. The broker uses the profiled energy values for the components and applications to generate and execute the PARM reconfiguration algorithm, using a specific policy. Using the prototype we intend to measure the performance of the proposed framework in an actual distributed environment for different classes of applications. Of particular interest, is to study the impact of the reconfiguration on the performance of applications and the determination of a policy that generates the best power-performance trade of in a truly distributed environment.

6 Concluding Remarks

With new emerging technologies such as cloud computing and increasing popularity of low-power mobile devices, pervasive use of these devices within distributed frameworks is inevitable. In future, distributed environments have to be cognizant of the widespread presence of these devices and confront the new challenges introduced by these resource-deficient computers. Distributed middleware frameworks hide the heterogeneity of underlying distributed environments when there are many cloud providers. These middlewares could hide and optimize many different types of mobile applications complexities while different cloud providers provide different services with diverse QoS parameters. This will provide new challenges for computer scientist in mobile computing area.

In this chapter, we presented a distributed power-aware reflective middleware framework, state of art literature and explored the viability of applying such architecture to achieve significant power gains in low-power devices. The

explored framework PARM dynamically adapted to the diminishing power availability of the devices over time, by dynamically offloading expensive middleware components to a proxy, thereby increasing the service times of the device. An optimal polynomial time graph theoretic algorithm was formulated to determine the dynamic.

7 References

[AMO93] Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. "Network Flows: Theory, Algorithms, and Applications". Prentice-Hall, N.J., 1993.

[ABE00] Andersen, A., Blair, G. S., and Eliassen, F. "A reflective component-based middleware with quality of service management", PROMS (October 2000).

[ANF03] Anand. M, Nightingale. E. B, and Flinn. J "Self-tuning wireless network power management", *MOBICOM*, 2003.

[ANF04] Anand. M, Nightingale. E. B, and Flinn. J "Ghosts in the machine: Interfaces for better power management", *MOBISYS*, 2004.

[ANJ02] Adve. S. V, Nahrstedt. K, Jones. D, and et. al. "The Illinois grace project: Global resource adaptation through cooperation", *SHAMAN-02*.

[ATS06] Ashwini. H. S, Thawani. A, and Srikant. Y. N" Middleware for efficient power management in mobile devices." In Proceedings of the 3rd international conference on Mobile technology, applications (Mobility '06) (2006).

[ASD03] Acquaviva1. A, Simunic. T, Deolalikar. V, and Roy. S "Server Controlled Power Management for Wireless Portable Devices", *HP Labs, Technical Report*, 2003.

[B05] Buttazzo, G. C. "Rate monotonic vs. EDF: Judgment day". *Real-Time Syst. (2005).*

[BSM04] Biermann. D, Sirer E.J , Manohar. M "A Rate Matching-based Approach to Dynamic Voltage Scaling", Proceedings of the First Watson Conference on the Interaction between Architecture, Circuits, and Compilers, (October 2004).

[BC00] Blair, G. S., Coulson, G., and et. al, F. C. "A principled approach to supporting adaptation in distributed mobile environments", PDSE (2000).

[BSO03] Balan. R. K, Satyanarayanan. M, Park. S, Okoshi. T "Tactics-Based Remote Execution for Mobile Computing." In Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services (MobiSys 2003).

[CY02] Chakraborty, S., and Yau, D. K. Y. "Predicting energy consumption of mpeg video playback on hand helds". In In Proc. IEEE International Conference on Multimedia and Expo (August 2002).

[C02] Chandra, S. "Wireless Network Interface energy consumption implications of popular streaming formats". In MMCN (January 2002).

[CGK97] Chekuri, C., Goldberg, A. V., Karger, D. R., Levine, M. S., and Stein, C. "Experimental Study of Minimum Cut Algorithms". In Symposium on Discrete Algorithms (1997), pp. 324–333.

[CR00] Chiasserini, C. F., and Rao, R. "Energy efficient battery management". In IEEE Infocom (March 2000).

[CM09] Chun. B. G, Maniatis. P "Augmented Smartphone Applications through Clone Cloud Execution." In Proc. Of the 8th Workshop on Hot Topics in Operating Systems (HotOS), Monte Verita, Switzerland, (2009).

[CBC10] Cuervo. E, Balasubramanian. A, Cho. D. K, Wolman. A, Saroiu. S, Chandra. R, Bahl. P, "MAUI: Making Smartphones Last Longer with Code Offload", in ACM MobiSys 2010, Association for Computing Machinery, Inc., (2010).

[CV02] Chandra. S and Vahdat. A. “Application-specific Network Management for Energy-aware Streaming of Popular Multimedia Formats” In *Usenix Annual Technical Conference*, June 2002.

[DKB95] Douglis, F., Krishnan, P., and Bershad, B. “Adaptive disk spin down policies for mobile computers”. In 2nd USENIX Symposium on Mobile and Location-Independent Computing (April 1995).

[DKM94] Douglis, F., Krishnan, P., and Marsh, B. “Thwarting the power hungry disk”. In WINTER USENIX conference (January 1994)

[ED00] Efstratiou, Davies, C. K., and A, F. “Architectural requirements for the effective support of adaptive mobile applications”. In Middleware (2000).

[E99] Ellis, C. “The case for higher level power management”. In In Proceedings of HotOS (March 1999).

[FFB00] Farkas, K., Flinn, J., Back, G., Grunwald, D., and Anderson, J. “Quantifying the energy consumption of a pocket computer and a java virtual machine”. In ACM conference on Measurement and Modeling of Computer Systems (2000).

[FN01] Feeney, L., and Nilsson, M. “Investigating the energy consumption of a wireless network interface in an ad hoc networking environment”. In IEEE Infocom (April 2001).

[FLS01] Flinn, J., de Lara, E., Satyanarayanan, M., Wallach, D. S., and Zwaenepoel, W. “Reducing the energy usage of of cell applications”. In IFIP/ACM International Conference on Distributed Systems Platforms (2001).

[FM01] Feeney. L and Nilsson. M. “Investigating the Energy Consumption of a Wireless Network Interface in an ad hoc Networking Environment”, In *IEEE Infocom*, April 2001.

[FS96_a] Feng. W.C and Sechrest. S “Improving data caching for software mpeg video decompression.” In *IS&T/SPIE Digital Video Compresssion: Algorithms and Technologies*, 1996.

[FS99] Flinn, J., and Satyanarayanan, M. “Powerscope: a tool for profiling the energy usage of mobile applications”. In Proceedings of the Second IEEE Workshop on Mobile Computing Systems and Applications (1999).

[FPS02] Flinn . J, Park. S, Satyanarayanan. M “Balancing Performance, Energy, and Quality in Pervasive Computing.” In Proc. of the 22nd International Conference on Distributed Computing Systems (ICDCS), Vienna, Austria, (July 2002).

[GBW95] Golding. R, Bosch.P, Wilkes.J , “Idleness is not sloth,” Proc. of USENIX Winter Conf., New Orleans, L.A., 1995.

[GNM03] Gu. X, Nahrstedt. K, Messer. A, Greenberg. I, Milojicic. D “Adaptive Offloading Inference for Delivering Applications in Pervasive Computing Environments.” In Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (2003).

[GRJ09] Giurgiu. I, Riva. O, Juric. D, Krivulev. I, and Alonso “Calling the cloud: enabling mobile phones as interfaces to cloud applications ” In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware* (Middleware '09)

[HMV05] Huang. Y, Mohapatra S., Venkatasubramanian. N, "An Energy-Efficient Middleware for Supporting Multimedia Services in Mobile Grid Environments," International Conference on Information Technology (ITCC): Coding and Computing (2005).

[HKA01] Hughes. C. J , Kaul. P, Adve. S, Jain. R, Park. C, and Srinivasan..J ”variability in the execution of multimedia applications and implications for general-purpose architectures”. In *ISCA*, 2001.

[HSA01] Hughes C.J , Srinivasan. J, and Adve S.V “ Saving energy with architectural and frequency adaptations for multimedia applications”, In *MICRO*, 2001.

[IBM02] IBM and Monta Vista Software, “Dynamic Power Management for Embedded Systems", http://www.research.ibm.com/arl/projects/dpm.html, (November 2002).

[IGP95] Imielinski, T., Gupta, M., and Peyyeti, S. “Energy efficient data filtering and communications in mobile wireless computing”. In Proceedings of Usenix Symposium on Location Dependent Computing (April 1995).

[ISG02] Irani, S., Shukla, S., and Gupta, R. “Competitive analysis of dynamic power management strategies for systems with multiple power saving states”. In DATE (2002).

[JG02] Jejurikar, R., and Gupta, R. “Energy aware task scheduling with task synchronization for embedded real time systems”. In CASES (2002).

[KBD08] Kim. M, Banerjee. S, Dutt. N, and Venkatasubramanian. N " Energy-aware co-synthesis of real-time multimedia applications on MPSoCs using heterogeneous scheduling policies". ACM Trans. Embed. Computing. Syst. (2008).

[KK98] Kravets, R., and Krishnan, P. “Application-driven power management for mobile communication”. In Proceedings of MobiCom (1998).

[KHR03] Kremer. U, Hicks. J, Rehg. J. M “A Compilation Framework for Power and Energy Management on Mobile Computers “, In Springer Languages and Compilers for Parallel Computing, Lecture Notes in Computer Science,(2003).

[KWW02] Krintz, C., Wen, Y., and Wolski, R. “Application-level prediction of program power dissipation”. Tech. rep., University of California, San Diego, 2002.

[LWX02] Li, Z., Wang, C., and Xu, R. “Computation offloading to save energy on handheld devices: A partition scheme”. In Proc. of International Conference on Compilers, Architectures and Synthesis for Embedded Systems (November 2002).

[LS96] Lorch, J., and Smith, A. J. “Reducing processor power consumption by improving processor time management in a single-user operating system”. In Proc. of MOBICOM 96 (November 1996).

[LS98] Lorch, J., and Smith, A. J. “Software strategies for portable computer energy management”. In IEEE Personal Communications Magazine (June 1998).

[MZ95] Marsh, B., and Zenel, B. “Power measurements of typical notebook computers”. In Proc. of the USENIX Conf. (January 1995).

[MSS03] Martin. T. L, Siewiorek. D. P, Smailagic. A, Bosworth. M, Ettus. M, Warren. J” A case study of a system-level approach to power-aware computing” *ACM Trans. Embed. Computing Syst.* (2003).

[MV03] Mohapatra. S and Venkatasubramanian. N “PARM: Power-Aware Reconfigurable Middleware”. In *ICDCS-23*, 2003.

[MDN07] Mohapatra. S, Dutt. N, Nicolau. A, Venkatasubramanian. N, "DYNAMO: A Cross-Layer Framework for End-to-End QoS and Energy Optimization in Mobile Handheld Devices", IEEE Journal on Selected Areas in Communication, Vol. 25, No. 4, May 2007

[NSN97] Noble, B. D., Satyanarayanan, M., D.Narayanan, J.E.Tilton, and Flinn, J. “Agile application-aware adaptation for mobility”. In Proceedings of the 16th ACM Symposium on Operating Systems and Principles, Saint-Malo, France, (October 1997).

[OH98] Othman, M., and Hailes, S. “Power conservation strategy for mobile computers using load sharing”. In Mobile Computing and Communications Review (January 1998).

[PAL09] Paolo. B, Antonio. B, Luca. F “An IMS-Based Middleware Solution for Energy-Efficient and Cost-Effective Mobile Multimedia Services”, MobileWireless Middleware, Operating Systems, and Applications, Springer, (2009).

[PS01] Pillai, P., and Shin, K. G. “Real-time dynamic voltage scaling for low-power embedded operating systems”. In Proc. of the 18th ACM Symp. on Operating Systems Principles (2001).

[MZ95] Marsh, B., and Zenel, B. “Power measurements of typical notebook computers”. In Proc. of the USENIX Conf. (January 1995).

[RRP99] Rudenko, A., Reiher, P., Popek, G., and Kuenning, G. “Portable computer battery power saving using a remote processing framework”. In Mobile Computing Systems and Application Track of the ACM SAC (February 1999).

[SAE05]Schmitz, M. T., AL-hashemi, B. M., and Eles, P." Co-synthesis of energy efficient multimode embedded systems with consideration of mode-execution probabilities". *IEEE Trans. on CAD of Integrated Circuits and Systems (2005)*.

[SAS01] Schurgers, C., Aberthorne, O., and Srivastava, M. B. “Modulation scaling for energy aware communication systems”. In ISLPED (2001).

[SR03] Shenoy, P., and Radkov, P. “Proxy-Assisted Power-friendly streaming to mobile devices”. In MMCN (January 2003).

[SG01] Shukla, S., and Gupta, R. “A model checking approach to evaluating system level power management policies for embedded systems”. In Proceedings of IEEE Conference on High Level Design Validation and Test (2001).

[SK97] Stemm, M., and Katz, R. “Measuring and reducing energy consumption of network interfaces in hand-held devices”. In IEICE (Special Issue on Mobile Computing) (August 1997).

[VDM01] Venkatasubramanian, N., Deshpande, M., Mohapatra, S., and et. al. “Design and implementation of a composable reflective middleware framework”. In ICDCS-21 (2001).

[W04] Wolf, W." The future of multiprocessor systems-on-chips". In DAC ’04: Proceedings of the 41st Annual Conference on Design Automation (2004).

[WKP00] Wang, N., Kircher, M., Parameswaran, K., and Schmidt, D. C. “Applying reflective middleware techniques to optimize a Qos-enabled CORBA component model implementation”. In COMPSAC (2000).

[WWD94] Weiser, M., Welch, B., Demers, A., and Shenker, S. “Scheduling for Reduced CPU Energy” In Symposium on Operating Systems Design and Implementation (1994).

[YN00] Yuan, W., and Nahrstedt, K. “A middleware framework coordinating processor/power resource management for multimedia applications”. In IEEE Globecom (Nov 2000)

[YN02] Yuan. W , Nahrstedt. K, “Integration of Dynamic Voltage Scaling and Soft Real-Time Scheduling for Open Mobile Systems”, in Proc. of 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '02), (May 2002).

[YN03] Yuan.W , Nahrstedt. K “Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems”, in Proc. of 19th ACM Symposium on Operating Systems Principles (SOSP'03), Bolton Landing, NY, (October 2003).

[YN04] Yuan. W , Nahrstedt. K “Practical Voltage Scaling for Mobile Multimedia Devices”, in Proc. of ACM Multimedia 2004, New York, NY, (October 2004).

[ZEL02] Zeng. H, Ellis. C, Lebeck. A, and Vahdat. A "Ecosystem: Managing energy as a first class operating system resource". In *ASPLOS-02*.

[ZEL03] Zeng. H, Ellis. C, Lebeck. A, and Vahdat. A. "Currentcy: Unifying policies for resource management". In *Usenix Annual Technical Conference*, 2003.