PARM : Power Aware Reconfigurable Middleware

Shivajit Mohapatra & Nalini Venkatasubramanian
Dept. of Information & Computer Science
University of California, Irvine, CA 92697-3425
{mopy,nalini}@ics.uci.edu

Abstract

*In distributed environments, generic middleware services(e.g. caching, location management etc.) are widely used to satisfy application 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. We present a distributed middleware framework(*PARM*), that is inherently power-aware and reconfigures itself to adapt to diminishing power levels of low-power devices. In this paper, we i) determine whether a reconfigurable component-based middleware framework can be utilized to achieve energy gains in low-power devices, while preserving the semantics of the middleware services, ii) present and evaluate a graph theoretic approach for dynamically determining middleware component reconfigurations and ascertaining the optimal frequency at which the restructuring should occur, for maximal energy gains at the device. We use extensive profiling to chart the energy usage patterns of middleware components and applications, and use the profiled data to drive our reconfiguration decisions. Our simulation results demonstrate that our framework is able to save 5% to 35% of energy depending on the nature and class of applications and middleware components used.*

1 Motivation

The next generation of the internet and intra-nets will see a wide variety of small low power mobile devices operating on board with high-end systems, in large scale distributed environments. Due to their modest sizes and weights, these systems have inadequate resources - lower processing capabilities, memory, storage and power as compared to desktop systems. These portable computers are mostly battery driven and oftentimes have to run for considerable time periods; therefore power management becomes a critical issue for these systems. Tackling power dissipation and improving service times for these devices are crucial research challenges.

The growing popularity of distributed middleware systems coupled with their scalability, flexibility and suitability for mobile and wireless architectures has made them a dominant methodology for supporting distributed applications. Distributed applications require a multiplicity of services in order to accomplish their tasks. A distributed middleware system can provide important services like reliable messaging, distributed snapshots, clock synchronization, 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 system-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. 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 saving on both power and storage, while still providing the performance benefits due to caching.

Diverse efforts have been made to improve power management in mobile devices using hardware and software techniques. Dynamic Voltage scaling(DVS) [7] and dynamic power management techniques have been widely studied as methods for optimizing the power consumption. [4] describes a static task allocation scheme that splits connected tasks between a mobile device and a server for optimizing the energy consumption of the mobile device. Other middleware frameworks (e.g. Odyssey [6]) combine middleware notification and control with application adaptation to achieve performance improvements. Puppeteer [2] presents a middleware framework that uses transcoding to achieve energy gains. Our approach is to design an auto configurable middleware that customizes itself for optimal power benefits, independent of application and low-level adaptations. As future work, we plan to investigate the integration of application adaptations as well as low-level power management techniques with our framework.

In this paper, we present such an adaptive, reconfigurable middleware framework(PARM), that can signif-

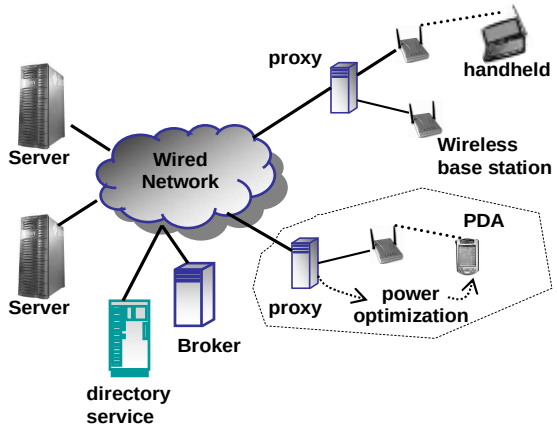

Fig. 1: **System Architecture**

icantly improve energy uptake of low-power devices operating in distributed environments. As the residual energy on the device diminishes, the PARM middleware dynamically reconfigures the component distribution on the low-power device. The middleware adaptation decisions are shifted to an external entity with cross-application knowledge and overall system information. PARM employs a graph theoretic solution for determining the optimal middleware component reconfigurations and identifying, 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.

The rest of the paper is organized as follows: Section 2 presents the overall system architecture and introduces the PARM framework; Section 3 characterizes the PARM component reconfiguration problem as a source parametric flow network and presents an optimal polynomial time solution to the component reconfiguration problem. In section 4, we evaluate the performance of PARM under different configuration criteria and conclude with future research directions in section 5.

2 System Architecture

A typical distributed system architecture using low power devices is envisioned in Fig. 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 conserve local resources. End clients are mobile devices that communicate with other entities via an access point(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 handoff 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-offs 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" interchangeably to represent middleware services.

The PARM Framework

In this section, we present a flexible, reflective message oriented middleware framework that is suitable for low power devices operating in distributed environments. Fig. 2 gives a succinct depiction of the PARM middleware framework that drives the applications on any computer in the system. PARM applications are developed using a set of application programming interfaces (APIs) exported by the middleware. The middleware framework

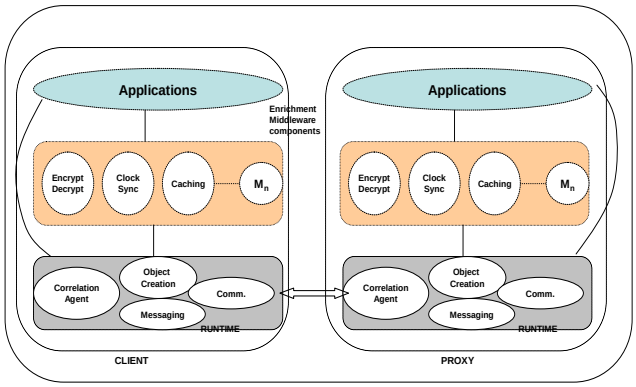

Fig. 2: **The parm Framework**

comprises of a core runtime that runs on all the systems within the environment. The core runtime services include the skeletal constructs for object creation, simple message passing and a communication framework for message routing. A "correlation agent" is included with the core runtime and performs 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. 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.

In PARM , middleware services can be dynamically started and stopped/migrated by the runtime without affecting 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 migrate 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.

3 The PARM Problem

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 at the device are minimized. Moreover, the cost of executing the components on the device are inflated as the residual power on the device drops. This adaptation causes the middleware to re-evaluate the component distribution and make appropriate alterations (possibly choosing to assign the now expensive component to execute at the proxy).

3.1 Characterization

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 $\boldsymbol{BW_t}$ and $\boldsymbol{BW_r}$ be the maximum bandwidth available to the device for transmitting and receiving data respectively. $\boldsymbol{P_{transmit}}$ and $\boldsymbol{P_{recv}}$ are the average power consumption rates at the device while it transmits and receives data. $\boldsymbol{P_{runtime}}$ represents the power consumption rate of the PARM runtime. Let $\boldsymbol{T_i}$ be the length of the $\boldsymbol{i^{th}}$ time interval (i.e. time interval between 2 consecutive executions of the PARM algorithm at the broker and let $\boldsymbol{R_i}$ represent the residual power on the device after the $\boldsymbol{i^{th}}$ time interval. $\boldsymbol{Size_t^k}$ and $\boldsymbol{Size_r^k}$ are the average sizes of the messages transmitted and received respectively, by the $\boldsymbol{k^{th}}$ middleware component. We characterize the some of the other parameters as follows:

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

- NS^k_{pi}: Average number of messages transmitted by the k^{th} component during the i^{th} time interval, when component is executing at the proxy.
- NR^k_{pi}: 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"
     $EC^k_{proxy} = PS_k \times T_i$
   - when middleware component "k" executes on the device during time interval "i"
     $EC^k_{device} = E_0 + \lambda.T_{const}$
     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 the profiling information
2. Communication cost
   - when middleware component "k" executes on the proxy during time interval "i"
     $CC^k_{proxy} = \frac{NS^k_{pi} \times Size^k_t}{BW_r} \times P_{recv} + \frac{NR^k_{pi} \times Size^k_r}{BW_t} \times P_{transmit}$
   - when middleware component "k" executes on the device during time interval "i"
     $CC^k_{device} = \frac{NS^k_{di} \times Size^k_t}{BW_t} \times P_{transmit} + \frac{NR^k_{di} \times Size^k_r}{BW_r} \times P_{recv}$

In order to optimize the energy in the system, we now have to find a 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_i we need to minimize the following term:

$$\sum_{k \in X} EC^k_{device} + \sum_{k \in X} CC^k_{device} + \sum_{k \in U-X} EC^k_{proxy} + \sum_{k \in U-X} CC^k_{proxy}$$

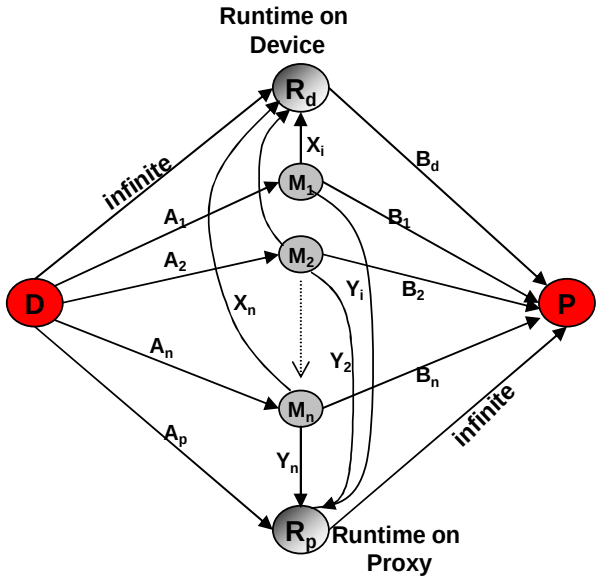

Fig. 3: **Flow Network**

Network Flow Representation

To achieve the optimal distribution of components between the device and the proxy, we cast our problem as a source parametric flow graph [1]. 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 a energy flow network. To create the flow network(Fig. 3), 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(see Fig. 3). We associate the source node(D) with the low-power device and the sink node(P) with a proxy. In Fig. 3, 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 components M_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 PARM 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

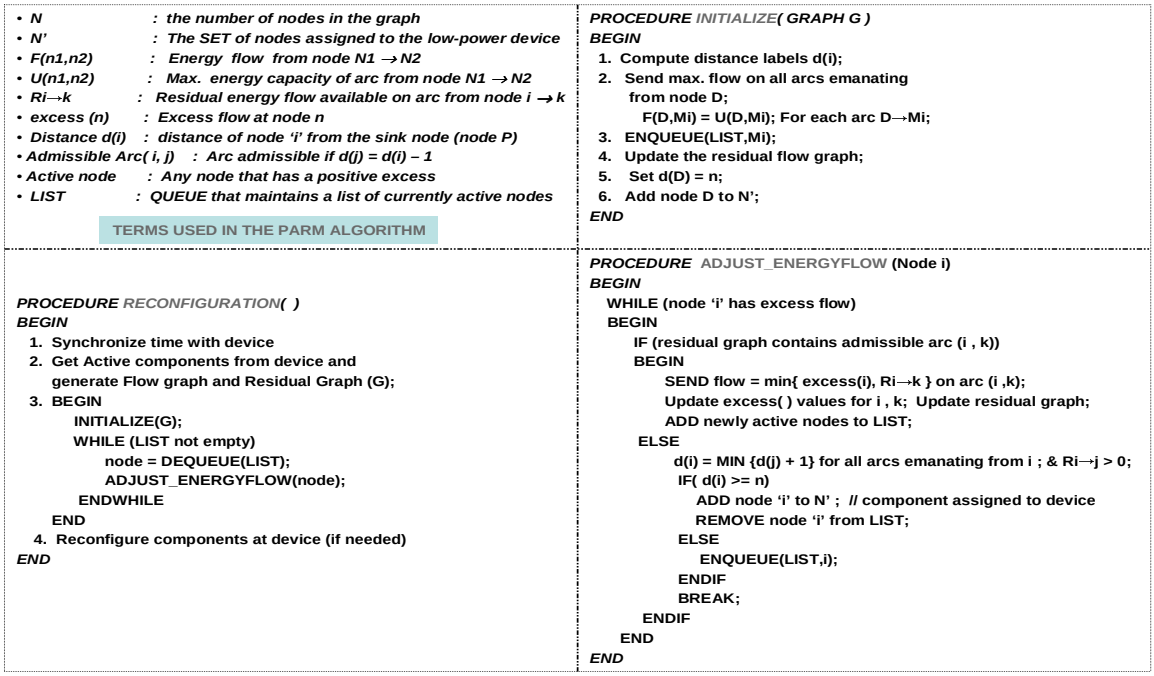

Fig. 4: **The PARM Algorithm**

and sending/receiving data to/from the device. The device runtime($\boldsymbol{R_d}$) is bound to the device(D) by giving assigning infinite energy cost to the arc from D to $\boldsymbol{R_d}$. 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 $\boldsymbol{P_1}$ and $\boldsymbol{P_2}$ be the assignment of components to the device and the proxy respectively. The minimum cut corresponding to this assignment is then ($\{D,\boldsymbol{R_d}\} \cup \boldsymbol{P_1}$], $\{P,\boldsymbol{R_p}\} \cup \boldsymbol{P_2}$). The cut of the graph effectively represents the minimum energy cost assignment of the components to the device and the proxy. Our algorithm has a worst case execution time of $O(\boldsymbol{n^3})$.

3.2 The PARM Algorithm

The PARM reconfiguration algorithm determines the minimum-cut of the flow graph created in Fig. 3, by first resolving the maximum flow of the graph. A *residual flow graph* is generated that 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 flow network in Fig. 3. Subsequently, the algorithm works on the residual graph generated at the previous step. The algorithm initially pushes the maximum possible flow from the device(node **D**) to the proxy(node **P**, Fig. 4). Depending on the capacities of the intermediate arcs($\boldsymbol{A_i}, \boldsymbol{X_i}, \boldsymbol{B_i}, \boldsymbol{Y_i}$), the maximum allowable flow gets routed to the proxy(**P**). Once a saturation point is reached at the intermediate nodes $\boldsymbol{M_i}$ (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. In order to get a minimum cut of the graph we keep track of the intermediate nodes($\boldsymbol{M_i}, \boldsymbol{R_d}, \boldsymbol{R_p}$) 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** & $\boldsymbol{R_d}$. In effect, this is the set of nodes(components $\boldsymbol{M_i}$) that would be assigned to the device. The other intermediate nodes(components) would be assigned to the proxy.

3.3 PARM Policies

We investigate the performance of our algorithm 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. We also ascertain the optimal times for executing the algorithm for different classes of applications and components. The power broker executes the PARM algorithm and initiates the required reconfiguration.

• *Random :* The broker performs the component reconfigurations at random intervals.
• *Periodic :* The broker performs the component reconfigurations at periodic intervals determined by the system administrator.
• *Application Triggered :* The PARM algorithm is triggered at the broker, whenever a new application/component is started up at the device.
• *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.

3.4 Profiling

Accurately profiling the energy consumption of the PARM runtime and the various middleware components is indispensable to the success of our approach. Without the knowledge of the energy overheads for communication and computation , the generation of the flow network is impossible. Given a set of "enrichment" middleware components and a set of applications that use them, we chart the energy usage (for both cpu operations and network communication) for all combinations of the components on the device and the proxy. Once the profiling is complete, the data can be made available to the power broker for generating the flow network. Appropriate hooks in the PARM runtime along with message tagging are used for measuring number of messages and individual message sizes. A more detailed description of the profiling is available in [5]. Fig. 5 depicts some of the energy values measured using our setup. Note that the profiled values are distinct for a particular device. However, as profiling is a one time effort, we assume energy usage can be profiled for different devices. In reality, the number of applications and components used in low-power mobile devices will also be limited.

Avg. Power (OS)	**4.2 J/s**
Avg. Power (OS + *PARM*)	**4.5 J/s**
Avg. *PARM* Runtime Overhead	**0.3 J/s**
Total transmitting bandwidth	**9 Mbps**
Total receiving bandwidth	**9 Mbps**
Avg. Power rate (transmit)	**3.65 J/s**
Avg. Power rate (receive)	**3.85 J/s**
Time to drain the battery of device	**270 mins**

Fig. 5: **Some profiled energy values**

4 Performance Evaluation

This section presents our simulation model and performance analysis. To simulate our system, we separately model the low-power device, the proxy server, the power broker, the PARM middleware framework and the applications. Applications are created using the APIs exported either by the PARM runtime or one or more of the "enrichment" components that are available with the middleware framework. The low-power device is modelled after a Compaq iPaq 3650 with a Lucent Orinoco PCMCIA network interface card. The proxy and the broker are assumed to be high-end wired workstations with substantial resources. Each device registers itself with a proxy and the PARM runtime on the device updates the device state at the directory service. The broker then runs the reconfiguration algorithm and announces the optimal configuration to the device and the proxy. A reconfiguration agent on the device then performs the necessary component migration [5]. As all components are replicated at the proxy, only a minimal amount of state(e.g queued messages) is communicated and this includes a small communication overhead. We are currently building a prototype using the CompOSE|Q [9, 5] middleware framework to integrate the PARM reconfiguration algorithm for power optimizations in a distributed environment.

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 applications that are currently regarded as suitable for hand-held computers and some applications can we think would be popular as the devices evolve. We further divide our set of representative applications into three classes: computation intensive (class-1, e.g *Image processing applications, interactive games*), communication intensive(class-2, e.g. *web browsing, network monitoring*) and both computation and communication intensive (class-3, e.g *multimedia streaming, GIS/navigation*).

To model the PARM components, we profile the energy pattern of each component while using it with a different 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. Table 1 illustrates the different power utilization values for a typical application and linked components for each application class. A detailed explanation of the simulation model can be found in [5]. Again, for each application class we assess the energy 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.

APPLICATION	CLASS 1	CLASS 2	CLASS 3
Middleware Components linked	Adaptive Scheduler Encryption-DES Decryption-DES	Reliable Messaging Clock Sync Message Ordering	Adaptive Sched. MM mesg service 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 1: **Application model**

4.1 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 load.

By "*gain*", we mean the TOS/ER saved by running the applications on the middleware framework with and without the PARM algorithm. We study how the TOS/ER gains are impacted as the number of applications scale; we also examine how often the broker should reconfigure the PARM components for a given set of applications to achieve optimal energy benefits.

Due to space constraints we only present a summary of the results here. A detailed performance report is available in [5]. We compared the impact of the periodic reconfiguration and the threshold reconfiguration policies for the three classes of applications, for both sporadic start and non-sporadic start applications. Fig. 6 and Fig. 7 compare the impact of different reconfiguration times for each class of applications. Both graphs show improvements in TOS gains when the reconfiguration occurs frequently. Fig. 8 and Fig. 9 depict the performance of the threshold policy for the three application classes. It was observed that class-3 applications showed negative gains for this policy and it performed significantly worse than the periodic policy on the same class of applications. However, adjusting the thresholds to trigger reconfigurations more frequently improved the gains appreciably. Fig. 10 and Fig. 11 study the impact of reconfiguration time on a mixed workload of applications. It was observed that the reconfiguration time had little impact on the selection of applications. A reconfiguration rate of around 3 minutes was found to give a significant gain in both residual energy and TOS. We also studied the impact of scaling the number of applications and varying the rate of execution the PARM reconfiguration algorithm, on the gain in TOS/ER, for three classes of applications [5]. We observed that for class-1 applications, service time gains of 15%-45% was achieved when the PARM reconfiguration algorithm was executed once every 5 minutes or less. The gains were found to be lesser, when the algorithm was executed less frequently. In case of class-2 applications, there was a 7%-30% increase in the time of service of the device. For 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). We infer that the unpredictability of the start/stop timings of the components seems to offset the effectiveness of the PARM algorithm in this case. In summary, the gains due to PARM were highest in the case of class-1 applications; gains were less moderate in the case of the other two classes of applications.

5 Related Work & Conclusion

A tremendous amount of research has already been done for achieving power savings in low-power and embedded devices. OS level efforts include techniques like dynamic voltage scaling [7], dynamic power management, effective battery management and optimization of communication devices [3]. Puppeteer [2] presents a middleware framework that uses transcoding to achieve energy gains. Odyssey [6] presents an applications aware adaptation scheme for mobile applications. The PARM framework uses a profiler to gather energy data for individual components and applications, but it does not require the application to perform any energy adaptations. In PARM the intent is to provide a framework that optimizes energy independent of applications. However, the PARM framework can reap significant additional savings by using adaptive applications. [8] shows that the task offload-

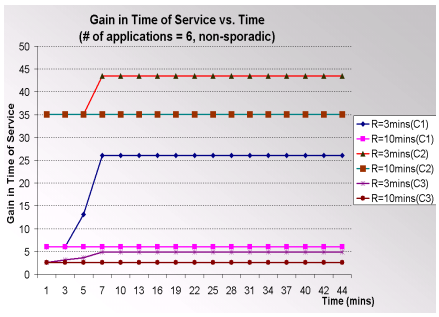

Fig. 6: **Recon.time vs TOS Gain**

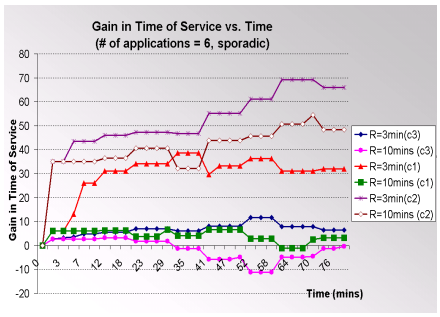

Fig. 7: **Recon.Time on TOS Gain**

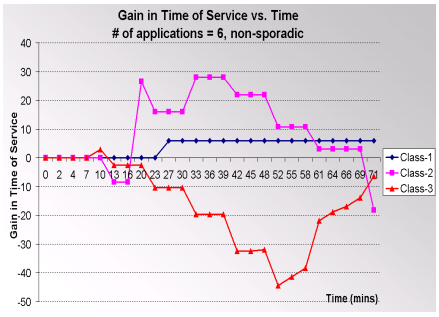

Fig. 8: **Threshold(non-sporadic)**

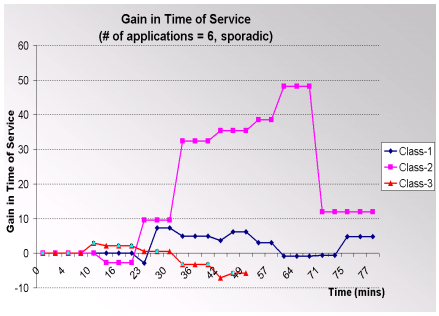

Fig. 9: **Threshold(sporadic)**

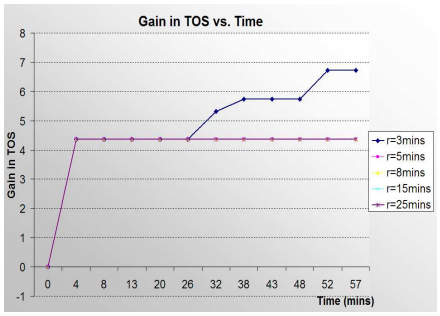

Fig. 10: **Recon.Time vs. TOS**

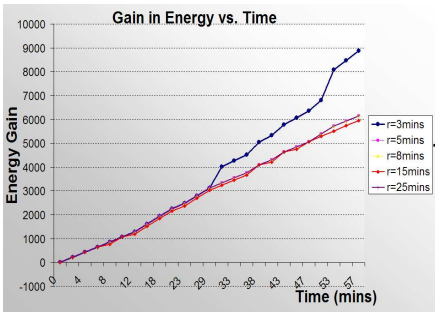

Fig. 11: **Recon.Time vs. Energy**

ing can deliver significant energy savings over a noiseless wireless network channels, while the gains are offset over noisy channels. In [4] a static task partition scheme is presented for offloading application subtasks onto a remote machine for energy savings. PARM on the other hand uses a dynamic component redistribution scheme that uses the residual power on the device as a parameter for determining the allocation of components. It is hard to compare PARM with traditional middleware frameworks such as CORBA, DCOM etc. as they are not designed for power optimization.

Concluding Remarks: In future, distributed environments have to be cognizant of the widespread presence of low-power devices and confront the new challenges introduced by these computers. In this paper, we presented a distributed power-aware middleware framework and explored the viability of applying such an architecture to achieve significant power gains in low-power devices. The framework dynamically adapted to the diminishing power availability of the devices over time, by dynamically offloading expensive middleware components to a proxy; a set of policies were designed to control the behavior of the system. The extensive simulation results of our experiments indicated that our framework can provide significant energy savings. Emerging middleware frameworks will be a dominant technology for employing wide-ranging mobile devices in future distributed systems. It is therefore imperative that middleware technology adapt itself for performing advantageously in such environments.

References

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

[2] Flinn, J., de Lara, E., Satyanarayanan, M., and et. al. "Reducing the energy usage of office applications". In *IFIP/ACM* (2001).

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

[4] Li, Z., Wang, C., and Xu, R. "Task allocation for distributed multimedia processing on wirelessly networked handheld devices". In *in Proc. of IPDPS* (April 2002).

[5] Mohapatra, S., and Venkatasubramanian, N. "Optimizing Power using a Reconfigurable Middleware". Tech. rep., UC, Irvine, 2003.

[6] Noble, B. D., Satyanarayanan, M., D.Narayanan, J.E.Tilton, and Flinn, J. "Agile application-aware adaptation for mobility". In *Proc. of 16th ACM Symposium on OS and Principles, France,* (October 1997).

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

[8] Rudenko, A., Reiher, P., and et.al. "Portable computer battery power saving using a remote processing framework". In *ACM SAC* (February 1999).

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