M-ECho: A Middleware for Morphable Data-Streaming in Pervasive Systems

Himanshu Raj Karsten Schwan
Ripal Nathuji
Center for Experimental Research in Computer Systems
Georgia Institute of Technology, Atlanta, GA 30332
{*rhim, schwan*}*@cc.gatech.edu, rnathuji@ece.gatech.edu*

Abstract

The end-to-end performance of pervasive mobile systems is commonly dictated by the availability of resources at the *weakest link*. However, a number of runtime adaptations or *morphing* steps can be performed to tune the system performance. In this paper, we present M-ECho, a middleware for system *morphing*. M-ECho is designed with focus on data streaming applications, specifically in the field of pervasive mobile systems. We consider an autonomous robotics application comprising of a set of cooperating mobile robots to demonstrate and evaluate M-ECho's system morphing capabilities. Optimizations are based upon metrics of average instantaneous power consumption at a single node (local) as well as the power consumed by all participants (global). Experimental results show that M-ECho is able to achieve improved end-to-end performance with its dynamic *code morphing* techniques.

1 Introduction

Pervasive mobile systems are often comprised of resource limited mobile nodes. These nodes can be characterized by decreased computing and communication capabilities, as well as limited battery power. Due to continuous environmental changes and changing system objectives, it is required that these systems adapt themselves in order to optimize their resource usage and performance.

In this paper, we present M-ECho, a middleware that provides support for continuous system adaptation and evolution, termed *system morphing*. In particular, M-ECho focuses on data streaming applications in pervasive domain. Specifically, the middleware (1) provides mechanisms for runtime behavioral changes and (re)deployment of program components and (2) makes middleware, and thereby applications, *aware* of current resources. Resource awareness involves runtime interactions between middleware and the underlying distributed platform. *Code morphing* is the runtime alteration of the implementations of specific program components, the goal being to dynamically create components with the properties that best match the system's resource usage directives. Code morphing is carried out by actions that include runtime code generation, code deployment or re-deployment, and by *change transactions* [8] that guarantee desired safety properties when distributed programs are changed at runtime. Morphing is triggered by changes in program objectives or behavior and/or in currently available resources.

Apart from traditional end-to-end performance criterion, such as increasing achievable bandwidth and reducing latency, efficient energy usage is an important objective for pervasive mobile systems since it directly relates to the longevity of the system. In this work, we focus on this criteria for evaluating the morphing capabilities of M-ECho for target systems.

Past research on energy efficiency in mobile systems has typically sought to prolong the battery lives of individual devices. Recent results extend to entire systems. An example is message routing in ad-hoc networks, where cooperative routing is performed to route messages to avoid using power-poor devices, thereby permitting the entire system to continue its operation [4]. Our work addresses the *system-wide power* consumed by the processing and communication actions of a distributed application. We focus on power and not on energy since (1) we assume that our tasks are long running and (2) available energy (battery power) is not a monotonically decreasing resource in the system. There might be opportunities for charging batteries in future or change in the total number of nodes comprising the system. The premise is that the application can function only as long as the *weakest* (the power poorest) device hosting its components. There is a variety of techniques to deal with predictable device failures due to power paucity, including migrating functionality par-

tially or completely onto different devices. Instead of requiring an application to explicitly implement such techniques, our research is creating middleware solutions that make it easier for developers to implement such techniques or even to automate their use.

The key advantage of code morphing over per device or per subsystem techniques for managing power is the ability to integrate application-level changes with system-level behavior changes. To evaluate and quantify this advantage, we have implemented code morphing in the M-ECho middleware. M-ECho implements a publish/subscribe paradigm of inter-machine communication, mapping the logical channels to link cooperating devices for inter-device communication. The data traversing these channels is described as events, where event providers, consumers, or intermediates can operate on events using well-defined event handlers. Event handlers are the software components for which we implement code morphing. Such handler morphing differs from prior work in application adaptation due to its ability to re-deploy handlers on demand and as currently needed by the application or its execution environment. The redeployment uses either dynamic compilation and code generation or static code repository. In a sense, handler morphing combines the abilities of compilers to generate the code most suitable for a platform (and its current resources) with parameterization or system-based techniques for dynamic program adaptation.

While M-ECho's implementation of code morphing targets event handlers, the idea of code morphing generalizes to other systems. Class and agent based migration has been used in Java and Corba frameworks [9, 7]. Tempo [14] uses code parameterization and compiler assisted code specialization for runtime optimization.

This paper makes two key contributions.

- It describes the code morphing mechanisms provided by M-ECho, where event handlers are dynamically modified in order to achieve desired changes in application behavior in response to system-wide changes in energy resources.
- Experimentation with a realistic, distributed application in the autonomous robotics domain both motivates this work and provides insights into the utility and limitations of middleware-level techniques like code morphing to improve performance. In this application, a team of autonomous robots must communicate environment information like images or laser data to each other, in order to attain some joint goal, such as foraging for resources. Attaining such a goal requires cooperation and joint actions, including tasks like motion and path planning.

An interesting insight derived from the use of M-ECho with the robotics application is a differentiation of *node local* from *system global* instantaneous power consumption as target performance metric. This is particularly relevant in mobile applications in which there are future opportunities for re-charging batteries (e.g., consider robots docking at a re-charging station). An interesting example in the robotics application is the use of compression techniques for image sensor data. JPEG compression, for instance, can reduce the instantaneous power consumption of the overall system by approximately $\sim 5\%$, even when we keep the full quality. If we reduce the quality to 75%, the overall system power consumption reduces to $\sim 8\%$. However, if the data provider (i.e., server) has the opportunity to re-charge its batteries in the near future, whereas the client does not, then it may be better to use simpler data reduction techniques, such as an 8 bpp (bits per pixel) grayscale conversion of a 24 bpp color image. This reduces client's power consumption up to $\sim 17\%$, at the cost of increasing server's power consumption by $\sim 10\%$. This tradeoff is entirely reasonable when server vs. client initial energy levels differ (and both should run out of power at approximately the same time) and/or when the server has future opportunities to re-charge whereas the client does not.

2 Issues with Power Consumption in Autonomous Robotics

Energy conservation has always been a goal for mobile devices due to their limited battery power. In distributed mobile systems, a range of techniques is used to conserve system-wide energy and ensure application longevity, ranging from node-level methods like dynamic voltage scaling [17], periodic sleeping or frequency scaling to reduce idle times, to compiler techniques such as generating energy-efficient code [11], to OS specific methods like power-aware scheduling [20], to application-level methods like agent migration [9] and changing attributes such as fidelity [6]. Amongst such techniques, online methods for power management typically address specific subsystems, beginning with early work on power-aware communication [10], to recent work on power-aware routing in wireless ad-hoc networks [18]. Such methods can take advantage of the fact that participating nodes tend to be freely interchangeable, so that one node can transparently replace another. In comparison, in the robotics applications addressed by our work, nodes have certain roles, which must be taken into account explicitly when applications are adapted.

Our target application domain is distributed autonomous robotics. Our specific application is one in which a team of robots cooperated to execute some common task, an example being disaster recovery and relief. Each rescue robot can perform multiple actions (i.e., play

different roles), constrained by the availability of certain peripherals. An example is a robot equipped with a laser but not with odometry capabilities, which means that the robot can perform environment sensing but not localization (i.e., help the team determine its robots' respective positions). Other constraints are due to current application or environment state. A robot with a good view of a disaster site, for instance, may have to maintain its critically important role as a sentry, despite its ability to also perform other tasks. Another interesting characteristic of this application is that energy need not be a uniformly decreasing resource. A robot may be able to increase its energy resources due to the availability of an external source of energy. The robot may be able to move towards an external power source and charge its batteries without causing a disruption of service in the system. Or, a robot may have solar batteries, which can be re-charged when the robot enters a sunny environment.

The specific prototype application being constructed by collaborating robotics researchers [1] will consist of up to 100 mobile robots equipped with diverse sensors such as laser, sonar and cameras, 802.11b-based wireless communication device(s) and Intel XScale based CPUs. This paper focuses on the robot team's IT infrastructure, in anticipation of future micro-robots and current micro-sensor-based systems where power consumption is dominated by the IT infrastructure's actions and behavior. The experiments conducted in this paper implement a sample collaborative task, using video streams to emulate sensor communications across different robots and power behavior of a localization algorithm.

The following list concisely articulates the energy-related goals of code morphing pursued for our mobile application. The objective is to extend the application's longevity.

1. Maintain suitable per device power health by modifying application behavior subject to current robot roles, device state, and application state. The goal is to always run application codes where most appropriate for the entire application, thereby optimizing application-wide metrics of utility/power. A sample method is to offload (i.e., delegate) tasks to the most appropriate robots, based on their current power health, capabilities, and application state.

2. Reducing application communication overheads, the method being to dynamically eliminate unnecessary data from event communications, thereby reducing power usage. This implies placing appropriate code (i.e., M-ECho event handlers and filters) onto data-providers (e.g., robots with sensors).

3. Create application-level overlays to help with task offloading, where codes ordinarily running on single nodes are morphed to processing pipelines mapped across multiple, network-connected machines. In comparison to prior work on network-level message routing, our work creates middleware overlays that route and process events, matching robots' power levels and processing capabilities. This is akin to assigning roles to sensor nodes in a wireless sensor network [16].

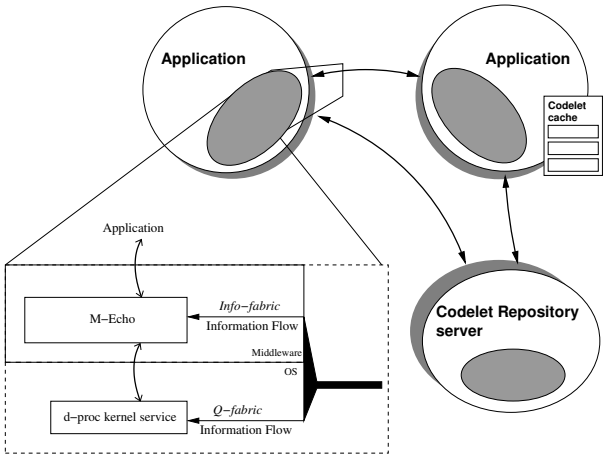

Figure 1: M-ECho Architecture

3 System Morphing with M-ECho

3.1 M-ECho Software Architecture

The M-ECho framework (refer to figure 1) consists of the following components:

1. *InfoFabric Middleware Platform*. The InfoFabric middleware platform is based on the ECho [5] publish/subscribe middleware which presents a distributed source/sink view for the whole system. Information is exchanged between source and sink nodes in form of *events*. *Event handlers* are deployed at the sink node to respond to specific events, while *filters* are deployed at either source or sink nodes to perform information filtering. These event handlers and filters are application specific. The middleware platform also interacts with kernel-level d-proc monitoring service which is part of Q-Fabric [15] infrastructure to receive information about system level events such as the remaining battery power at a node. Infofabric enables data aggregation at nodes, much in the same way provided by DFuse [12] and TAG [13]. The key difference is that the unit of aggregation are ECho events vs. raw sensor data.

   Since M-ECho utilizes the publish/subscribe paradigm of communication, a straightforward

representation of morphable components are the handlers and filters applied to the data events that traverse M-ECho-based applications. In fact, M-ECho's implementation of filters and handlers (also termed as code segments or codelets hereafter) simplifies morphing by limiting the generality of code and state compared to that of arbitrary program components. The codelets are either compiled dynamically using dynamic code generation or a suitable binary version is obtained from the *codelet repository service*, described below.

2. *Codelet Repository Service (CRS)*. The CRS functions as a repository for codelets. Codelets are stored along with multiple attributes. Examples of these attributes include the format such as binary vs. ECL source (ECL is a subset of C language [5], OS/platform and a cryptographic signature hash. The service provides a traditional FTP like get/put interface for obtaining/publishing codelets. By storing some pre-compiled versions of a codelet on the repository, we can save on the cost of dynamic code generation wherever possible.

3.2 Code Morphing Techniques

Our morphing work takes an overall system approach to dynamic adaptation. In other words, based upon system requirements (these requirements may be local or global), we adapt the application components (including middleware) and the environment components (such as OS and network). In the presented work, we focus on morphing in M-ECho via adaptation of application components.

Following are the specific morphing techniques that M-ECho currently implements:

- *Code parameterization*. We modify particular parameters to affect the outcome of a certain *codelet* such as changing the number of items that a aggregating function might be using, changing the quality parameter that dictates the computation being performed by a codelet and changing the rate of events to be generated.

- *Code substitution*. We replace one codelet by another. This is assisted via dynamic code generation for the specific platform or via CRS, as described above.

- *Code migration*. We use dynamic (re)partitioning and (re)deployment of application components based on environmental attributes provided by the Q-Fabric infrastructure such as available network bandwidth and remaining node power.

We collectively refer to the morphing techniques used by M-ECho as code morphing. The current version of M-ECho supports all the code morphing techniques described above. As we show later, with code morphing alone we are able to achieve goals 1 and 2 as stated in section 2.

4 Experimental Results

Our experimental scenarios are based upon a set of robots exchanging sensor information in order to perform localization and path planning. Power measurements focus on typical robotic computing subsystems. In particular, we use PXA255 XScale CPU based evaluation boards and StrongARM CPU based iPAQs to mimic the computing system of a robot. Power measurements are obtained by monitoring current with a measurement circuit based upon a current sensor IC, and simultaneously measuring voltage. These values are measured with the Picoscope 212/50 PC-oscilloscope [2]. In the applied mode, the oscilloscope returns an average of a set of samples obtained at 50 million samples per second every 1ms to measure single shot signals. We report the average of these power values as *average instantaneous power*.

We enhanced the commonly used Player/Stage robotics framework [3] to use M-ECho middleware. The current implementation of Player/Stage uses parameterization to perform limited morphing, adapting event rate(s) and event type. Using M-ECho, we are able to enhance the capabilities of the framework (1) to perform codelet substitution and (2) to perform codelet (task) migration across robots, all of which directly affect power usage. The goal is to improve energy usage for all devices participating in the application, that is, to extend the ability of all devices to continue to pursue their joint application-level tasks. In this context, we look at two energy-intensive tasks of this application, the acquisition and exchange of sensor data and the task of localization.

In our experimentation, former is implemented as the exchange and processing of video images captured by device-mounted cameras. Code morphing techniques such as code parameterization and code substitution are used to utilize diverse data encoding and compression techniques in order to control energy usage at information providers vs. consumers. For the latter task, we use a localization application which is part of the enhanced player/stage framework. The application uses the adaptive monte-carlo algorithm [19] and continuously tries to localize the robot based on the sensor data. We use a log of laser readings obtained from one random walk of the robot in a synthetic environment. The localization application reads sensor data from this log and computes the localization information. We use code migration to show that it is possible to achieve power savings along with

better QoS when localization is run locally on the robot vs. off-board.

4.1 Code Parameterization and Substitution

To simulate sensor date exchange between robots, we use a media application running on iPAQs, in which a source sends out raw 24-bit color 352X240 PPM camera images over a 802.11b wireless network and a client (or sink) displays them. The application utilizes several data reduction methods for system morphing. These methods include JPEG compression and image transcoding such as grayscale (GRAY) and black&white (BW) conversion and image cropping (CROP). System morphing techniques used are code parameterization, such as changing the JPEG compression level, and code substitution, such as changing data reduction methods (which requires changing the filters and handlers at source and sink respectively).

Table 1 shows the average instantaneous power consumed (in watts) by the a source and a sink node. *Idle* shows the base case power consumption for both nodes when they are idle. When the application is running, source sends data as fast as it can, assuming that it always has information to communicate. The table lists power values for different data reduction methods. Numeric value adjacent to JPEG denotes the quality parameter value used by the JPEG compression. The frame rate observed by the sink and the normalized power consumption for the desired rate of 1 frame per second (refer to the discussion below) are also shown.

We compare the different methods as following. Given a method can achieve more QoS (in our case, frame rate) than desired, we compute the normalized power as following:

$$Norm\ power(i) \quad = \quad F/T * 1/r_i * (P_i - I) + I$$

where r_i is the QoS (achieved frame rate) and P_i is the average instantaneous power for method i, I is the idle power and F/T is the desired frame rate. To obtain system power, we add respective power values for source and sink nodes. If the frame rate achievable is less than desired, the normalized power at a node is P_i (as is the case for BW and GRAY). However this implies a lower utility for the application (we assume that utility is a smooth monotonically decreasing function and not a unit step function).

These experiments demonstrate that the choice of morphing technique and corresponding method depends on the objective. For a desired QoS of 1 frame per second for our application, if the objective is to optimize client average power usage (regardless of the utility), morphing system can use code substitution to employ encoding methods such as BW or GRAY. If the objective is to optimize global average power usage, then code substitution can be used to employ JPEG compression. Furthermore, if we can tolerate some image quality loss, code parameterization can be employed to use 75% quality for optimal overall average system power usage.

CPU	Local		Remote	
Freq (MHz)	P (W)	T (s)	P (W)	T (s)
400	4.24	39	3.37	38
300	3.75	59	3.2	38
200	3.62	78	3.16	38

Table 2: Average instantaneous power usage and run time for a localization application on a node where localization algorithm is run locally vs. on a remote node

4.2 Code Migration

A common approach to reducing local power consumption at a node is to take advantage of CPU frequency/voltage scaling [17]. However, a reduced frequency often results in an impact to QoS. For example, for a localization application running on the evaluation board, table 2 shows that the local average instantaneous power consumption (Local P (W)) decreases with lower frequency (the processor automatically scales to the lowest voltage to support the desired frequency). However, the time (T) required to compute localization increases, thereby nullifying any gains obtained from frequency scaling. For our application, this effect is magnified due to the architectural characteristics of the PXA255 CPU. The localization algorithm requires floating point operations which are not supported in hardware. For all FP operations, the hardware traps to the OS and software libraries are used to perform the actual operations. This overhead can dominate performance for floating point intensive applications.

Assuming that the environment consists of some power and performance rich nodes that have the ability to periodically recharge themselves (i.e. their power consumption is not of concern in the performance of the system) and can process data faster than the local node, we demonstrate that the localization application can directly benefit from code migration. Table 2 shows the local node's average instantaneous power consumption when the localization algorithm is run remotely (Remote P (W)). In this case, local node not only uses less power with lower frequency, it can still finish the desired task in almost the same time as with higher frequencies. This also serves as an example of system morphing due to restricted node capabilities.

Method	Inst. Power (W)		Frame Rate	Normalized power (W)	
	Sink	Source	(Sink)	Sink	Source
Idle	1.66	1.49	0	1.66	1.49
Nocomp	2.16	1.96	1.9200	1.92	1.73
JPEG 100	2.3	2.12	3.9000	1.82	1.65
JPEG 5	2.32	1.91	6.6250	1.76	1.55
JPEG 75	2.33	1.97	6.1000	1.77	1.57
BW	1.59	2.01	0.3797	1.59	2.01
GRAY	1.63	1.94	0.3750	1.63	1.94
CROP	2.14	2.05	14.1375	1.69	1.53

Table 1: Source and sink power consumption for parameterization and substitution code morphing techniques

5 Conclusions and Future Work

In this paper, we present the design of M-ECho and its code morphing capabilities. Using M-ECho for an autonomous robotics application in a pervasive mobile domain, we demonstrate power performance benefits of code morphing techniques.

Our future efforts are focused on:

- Using compiler assisted techniques for the dynamic compilation of codelets for achieving power benefits, such as mixed code generation [11].
- Quantifying the energy cost of dynamic compilation of codelets vs. the use of CRS.
- Building morphable application level overlays based on node's capabilities to achieve goal 3 as stated in section 2.

6 Acknowledgments

We are thankful to Keith J. O'Hara and other robotics collaborators at the Borg Lab [1]. This research was supported in part by a NSF ITR award.

References

[1] The borg lab. http://www.cc.gatech.edu/∼borg.

[2] Picoscope. http://www.picoscope.com/.

[3] The Player/Stage Project. http://playerstage.sourceforge.net/.

[4] CHOKHAWALA, J., AND CHENG, A. M. K. Optimizing Power Aware Routing in Mobile Ad Hoc Networks. In *Proceedings of WIP Session, Real-Time and Embedded Technology and Applications Symposium* (Toronto, Canada, 2004).

[5] EISENHAUER, G., BUSTAMANTE, F., AND SCHWAN, K. Event services for high performance computing. In *Proceedings of High Performance Distributed Computing* (2000).

[6] FLINN, J., AND SATYANARAYANAN, M. Managing Battery Lifetime with Energy-Aware Adaptation. *ACM Transactions on Computer Systems* (May 2004).

[7] GU, X., NAHRSTEDT, K., MESSER, A., GREENBERG, I., AND MILOJICIC, D. Adaptive Offloading Inference for Delivering Applications in Pervasive Computing Environment. In *Proc. of IEEE International Conference on Pervasive Computing and Communications* (2003).

[8] ISERT, C., AND SCHWAN, K. ACDS: Adapting Computational Data Streams for High Performance. In *Proceedings of International Parallel and Distributed Processing Symposium* (2000).

[9] KON, F., YAMANE, T., HESS, C., CAMPBELL, R., AND MICKUNAS, M. D. Dynamic Resource Management and Automatic Configuration of Distributed Component Services. In *Proceedings of the 6th USENIX Conference on Object-Oriented Technologies and Systems* (2001).

[10] KRAVETS, R., AND KRISHNAN, P. Power Management Techniques for Mobile Communication. In *Proceedings of MOBICOM* (1998).

[11] KRISHNASWAMY, A., AND GUPTA, R. Profile Guided Selection of ARM and Thumb Instructions. In *Proceedings of ACM SIGPLAN Joint Conference on Languages Compilers and Tools for Embedded Systems* (2003).

[12] KUMAR, R., WOLENETZ, M., AGARWALLAA, B., SHIN, J., HUTTO, P., AND RAMACHANDRAN, U. DFuse: A Framework for Distributed Data Fusion. In *Proceedings of SenSys* (2003).

[13] MADDEN, S., FRANKLIN, M. J., HELLERSTEIN, J. M., AND HONG, W. TAG: A Tiny Aggregation Service for ad hoc Sensor Networks. In *Proceedings of OSDI* (2002).

[14] NOEL, F., HORNOF, L., CONSEL, C., AND LAWALL, J. L. Automatic, Template-Based Run-Time Specialization: Implementation and Experimental Study. In *Proceedings of the International Conference on Computer Languages* (1998), IEEE Computer Society, p. 132.

[15] POELLABAUER, C. *Q-Fabric: System Support for Continuous Online Quality Management.* PhD thesis, College of Computing, Georgia Institute of Technology, 2004.

[16] ROMER, K., FRANK, C., MARRON, P. M., AND BECKER, C. Generic Role Assignment for Wireless Sensor Networks. In *Proceedings of SIGOPS European Workshop* (2004).

[17] SIMUNIC, T., BENINI, L., ACQUAVIVA, A., GLYNN, P. W., AND MICHELI, G. D. Dynamic Voltage Scaling and Power Management for Portable Systems. In *Proceedings of Design Automation Conference* (2001).

[18] STOJMENOVIC, I., AND LIN, X. Power Aware Localized Routing in Wireless Networks. *IEEE Transactions on Parallel and Distributed Systems* (November 2001).

[19] THRUN, S., FOX, D., AND BURGARD, W. Monte Carlo Localization With Mixture Proposal Distribution. In *Proceedings of the National Conference on Artificial Intelligence* (2000).

[20] YUAN, W., AND NAHRSTEDT, K. Energy-Efficient Soft Real-Time CPU Scheduling for Mobile Multimedia Systems. In *Proceedings of SOSP* (2003).