Proceedings of the 8th Workshop on Hot Topics in Operating Systems, May 2001, Schloss Elmau, Germany

Self-Tuned Remote Execution for Pervasive Computing

Jason Flinn, Dushyanth Narayanan and M. Satyanarayanan
School of Computer Science
Carnegie Mellon University

Abstract

Pervasive computing creates environments saturated with computing and communication capability, yet gracefully integrated with human users. Remote execution has a natural role to play in such environments, since it lets applications simultaneously leverage the mobility of small devices and the greater resources of large devices. In this paper, we describe Spectra, a remote execution system designed for pervasive environments.

Spectra monitors resources such as battery energy and file cache state which are especially important for mobile clients. It also dynamically balances energy use and quality goals with traditional performance concerns to decide where to locate functionality. Finally, Spectra is self-tuning—it does not require applications to explicitly specify intended resource usage. Instead, it monitors application behavior, learns functions predicting their resource usage, and uses the information to anticipate future behavior.

1 Introduction

Remote execution is an old and venerable topic in systems research. Systems such as Condor [3] and Butler [15] have long provided the ability to exploit spare CPU cycles on other machines. Yet, the advent of pervasive computing has created new opportunities and challenges for remote execution. In this paper, we discuss these issues and how we have addressed them in the implementation of Spectra, a remote execution system for pervasive computing.

The need for mobility leads to smaller and smaller computing devices. The size limitations of these devices constrain their compute power, battery energy and storage capacity. Yet, many modern applications are resource-intensive, with demands that often outstrip device capacity. Remote execution using wireless networks to access compute servers thus fills a natural role in pervasive computing, allowing applications to leverage both the mobility of small devices and the greater resources of stationary devices.

Pervasive computing also creates new challenges [19]. When locating functionality, Spectra must balance the traditional goal of minimizing application latency with new goals such as maximizing battery lifetime. It must allow for wider variation in resources such as CPU and network bandwidth and monitor new resources such as energy use and cache state.

Pervasiveness causes additional complexity, and it is unreasonable to leave the burden of handling this complexity to applications. Spectra does not require applications to specify resource requirements for a variety of platforms and output qualities. Instead, it is *self-tuning*—it monitors application resource usage in order to predict future behavior.

2 Design considerations

The design of Spectra has been greatly influenced by the need to address the complexities of pervasive computing.

Spectra weighs several possibly divergent goals when deciding where to execute applications. Performance remains important in mobile environments, but is no longer the sole consideration. It is also vital to conserve energy so as to prolong battery lifetime. Quality is another factor—a resource-poor mobile device may only be able to provide a low fidelity version of a data object [16] or computation [20], while a stationary machine may be able to generate a better version.

Spectra monitors environmental conditions and adjusts the relative importance of each goal. For example, energy use is paramount when a device's battery is low. However, when the battery is charged, performance considerations may dominate. Monitoring battery state and expected time to recharge allows Spectra to adjust the relative importance of these goals.

Spectra monitors resources that are uniquely significant in pervasive environments. In addition to battery energy, file cache state is often critical. Consider a mobile client with limited storage running a distributed file system. When

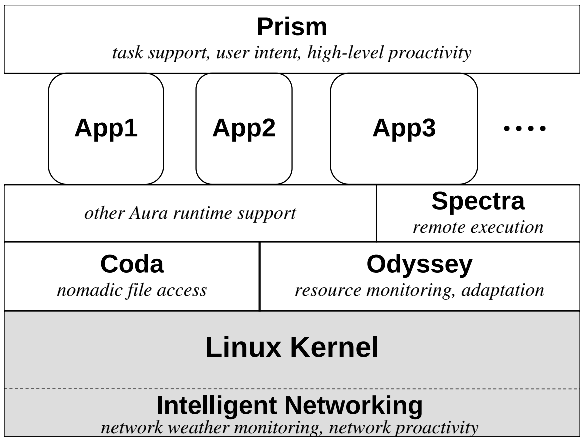

Figure 1. Aura architecture

there is a choice of remote execution sites, a server with a warmer file cache may often be preferable to one with a faster processor.

Finally, Spectra is self-tuning. Applications need not specify their expected usage of various resources. Providing estimates for even a single resource such as battery energy is very difficult since energy use depends upon the hardware platform and the degree of power management used. Spectra applications need only specify operations of interest and the input parameters to those operations. Spectra monitors and logs resource usage as applications execute. From logged data, it learns functions relating input parameters to resource usage, allowing it to predict future application resource use.

3 Implementation

3.1 Spectra overview

Spectra is the remote execution component of Aura, a new computing system being built at Carnegie Mellon University. Aura provides users with an invisible halo of computing and information services that persists regardless of location. As shown in Figure 1, an Aura client is composed of many parts. The Coda file system [10] allows mobile nodes to access shared data, even when weakly-connected or disconnected from the network. Odyssey [16] supports applications that vary their fidelity as resource availability changes. Fidelity is an application-specific metric of quality expressed in multiple discrete or continuous dimensions. For instance, dimensions of fidelity for speech recognition are vocabulary size and acoustic model complexity.

To provide a complete solution, Spectra must address several complex issues, including function placement, service discovery, execution mechanism and data consistency. Our initial prototype focuses on the first problem: deciding where and how operations should be executed. It uses existing technology to address the remaining issues. We hope to leverage service discovery protocols which allow attribute-value lookup [1, 23]. Similarly, while we currently use RPC-based remote execution, Spectra could be modified to use other mechanisms such as mobile code. Finally, Coda provides Spectra a single shared file system across multiple machines.

Spectra consists of three main elements:

- an application interface for describing operations.
- monitors that predict resource use and availability.
- a decision engine that selects the best execution option.

3.2 Application interface

Applications use the Odyssey multi-fidelity interface [14] to communicate with Spectra. The fundamental unit of discourse is the *operation:* a code component which may profit from remote execution. Spectra targets applications which perform operations of one second or more in duration—examples are speech recognition, rendering for augmented reality, and document processing.

Applications first register operations with Spectra. A registration lists possible fidelities and methods of dividing computation between local and remote machines. It also lists input parameters that affect operation complexity.

For example, we have modified the Janus speech recognizer [24] to use Spectra. The basic operation is utterance recognition. This operation has two fidelities: full and reduced. Reduced fidelity uses a smaller, more task-specific vocabulary than full fidelity. There are three modes of dividing computation: recognition may be performed on the client (local mode), on a server (remote mode) or on both (hybrid mode). In hybrid mode, the first phase is performed locally, yielding a greatly compressed data set which is shipped remotely for the completion of recognition. The single input parameter is the length of the utterance.

Prior to operation execution, an application invokes Spectra to determine how and where the operation will execute. The application passes in the value of the input parameters—for example, the size of an utterance to be recognized. Spectra chooses the best fidelity level and execution mode as described in Section 3.4 and returns these values to the application. For remote operations, Spectra also chooses the server on which the operation will be executed.

Applications execute operations by making remote procedure calls to the selected server. Direct procedure calls can be used in the local case to optimize performance. Applications inform Spectra when operations complete, at which time Spectra logs resource usage. The logged data allows Spectra to improve resource prediction over time.

3.3 Resource monitoring

Only part of the data needed by Spectra comes from applications—the remainder is supplied by resource monitors. Resource monitors are modular, resource-specific code components that predict resource availability and demand.

Prior to operation execution, each monitor predicts how much of a resource the operation will receive. Monitors make predictions for the local machine and for any remote servers on which the operation may execute. For instance, the network monitor predicts bandwidth and round-trip times between the client and each server. Spectra gathers the predictions in a *resource snapshot*, which provides a consistent view of resource availability for that operation.

Resource monitors observe application behavior to predict future resource demand. While an operation executes, each monitor measures its resource usage. Upon operation completion, these values are logged, along with the operation's input parameters, fidelity, and method of dividing computation. From this data, Spectra learns functions which predict operation resource usage. Thus, the more an operation is executed, the more accurately its resource usage is predicted.

We have built monitors for four resources: CPU, network, battery, and cache state. As CPU and network are well-understood resources, we describe these monitors only briefly here. The CPU monitor, described in [14], predicts availability using a smoothed estimate of recent CPU load, weighted by the maximum speed of the processor. During operation execution, the CPU monitor measures CPU cycles consumed on local and remote machines. The network monitor predicts available bandwidth and round-trip times to remote machines using the algorithm in [16]. For each operation, it measures bytes sent and received, as well as the number of RPCs.

3.3.1 The battery monitor

The battery monitor must provide accurate, detailed information without hindering user mobility. Previous energy measurement approaches are thus insufficient for the task. It is infeasible to use external measurement equipment [7, 21] since such equipment can only be used in a laboratory setting. Alternatively, one can calibrate the energy use of events such as network transmission, and then later approximate energy use by counting event occurrences [4, 13]. However, results will be inaccurate when the calibration does not anticipate the full set of possible events, or when events such as changes in screen brightness are invisible to the monitor.

Our battery monitor takes advantage of the advent of "smart" batteries: chips which report detailed information about battery levels and power drain. The monitor predicts availability by querying the amount of charge left in the battery. It measures operation energy use by periodically polling the chip to sample energy use.

The first platform on which we have implemented our battery monitor is Compaq's Itsy v2.2 [8], an advanced pocket computer with a DS2437 smart battery chip [5]. Since the DS2437 reports average current drawn over a 31.25 ms. period and voltage levels change little, we could measure power by sampling current at 32 Hz. Unfortunately, the DS2437's communication protocol makes the overhead of frequent sampling unacceptably high. The battery monitor balances overhead and accuracy by sampling at 6 Hz during operation execution. This rate accurately measures operation energy use with low (1.8%) CPU overhead. At other times, the monitor samples at 1 Hz—a rate sufficient to accurately measure battery charge and background power drain.

3.3.2 The cache state monitor

Data access can consume significant time and energy when items are unavailable locally. The cache state monitor estimates these costs by predicting which uncached objects will be accessed. It currently provides estimates for one important class of items: files in the Coda file system.

During operation execution, the monitor observes accesses of Coda files. When an operation completes, the monitor logs the name and size of each file accessed.

The cache state monitor currently uses a simple prediction scheme—it assumes the likelihood of a file being accessed during an operation is similar to the percentage of times it was accessed during recent operations of similar type and input parameters. The access likelihood is maintained as a weighted average, allowing the monitor to adjust to changes in application behavior over time. For each file that may be accessed, the monitor queries Coda to determine if the file is cached. If it is uncached, the expected number of bytes to fetch is equal to the file's size multiplied by its access likelihood. The monitor estimates the number of bytes that an operation will fetch by summing individual predictions for each file.

The monitor makes predictions for both local and remote machines. It also estimates the rate at which data will be fetched from Coda servers so that Spectra can calculate the expected time and energy cost of fetching uncached items.

3.4 Selecting the best option

Spectra's decision engine chooses a location and fidelity for each operation. Its inputs are the application's description of the operation and the monitors' snapshot of resource availability. It uses Odyssey's multi-fidelity solver [14] to search the space of possible fidelities, remote servers, and methods of dividing computation. Using gradient-descent heuristics, the solver attempts to find the best execution alternative.

Spectra evaluates alternatives by their impact on *user metrics*. User metrics measure performance or quality perceptible to the end-user—they are thus distinct from resources, which are not directly observable by the user (other than by their effect on metrics). For instance, while battery energy and CPU cycles are resources, execution latency and change in expected battery lifetime are user metrics.

To evaluate an alternative, Spectra first calculates a context-independent value for each metric. It then weights each value with an *importance function* that expresses the current desirability of the metric to the user. Finally, it calculates the product of the weighted metrics to compute a single value for evaluating the alternative. This calculation is a specific instance of the broader concept of "resource-goodness mappings" [17]. Spectra currently considers three user metrics in its evaluation: execution latency, battery lifetime, and application fidelity.

Spectra may use many resource predictions to calculate a metric's context-independent value. For example, execution latency is the sum of the predicted latencies of fetching uncached items, network transmissions, and processing on local and remote machines. Processing latencies are calculated by dividing the predicted cycles needed for execution by the predicted amount of cycles available per second. Network and cache latencies are calculated similarly.

Since importance functions express the current desirability of metrics to the user, they may change over time. For example, we use goal-directed adaptation [6] as the importance function for battery lifetime. The user specifies a duration that the battery should last, and the system attempts to ensure that the battery lasts for this duration. A feedback parameter, c, represents how critical energy use is at the present moment. Spectra adjusts this parameter using estimates of battery charge and recent power usage reported by the battery monitor. Given expected energy use, E, the battery importance function is $(1/E)^c$. As an example, when the computer operates on wall power, c is 0 and energy has no impact in evaluating alternatives.

For execution latency, we use an application-specific importance function that reflects perceptible deadlines for operation completion. For example, the speech recognizer's importance function for latency, L, is simply $1/L$. This function has the intuitive property that a recognition that takes twice as long is half as desirable to the user.

Fidelity is a multidimensional metric of application-specific quality. The importance of fidelity is user-dependent and is often expressed with utility functions that map each user's preferences to a single value. For the speech recognizer, the fidelity importance function gives reduced fidelity the value 0.5 and full fidelity the value 1.0.

4 Preliminary evaluation

Our evaluation measured how well Spectra adapts to changes in resource availability. As a sample application, we used the speech recognizer described in Section 3.2.

We limited execution to two machines. The client was an Itsy v2.2 pocket computer with a 206 MHz SA-1100 processor and 32 MB DRAM. The server was an IBM T20 laptop with a 700 MHz PIII processor and 256 MB DRAM. Since the Itsy lacks a PCMCIA slot (such as is available on the Compaq iPAQ), the two machines were connected with a serial link.

We first recognized 15 utterances so that Spectra could learn the application's resource requirements. We then created several scenarios with varying resource availability and measured how well Spectra adapted application behavior when a new utterance was recognized. Figure 2(a) shows measured execution latency and energy use for each possible combination of fidelity and location. For each scenario, the option that best satisfies the evaluation criteria for the speech application is highlighted. Figure 2(b) shows results when Spectra chooses the alternative to execute.

In the baseline scenario both computers are unloaded and connected to wall power. Spectra correctly chooses the hybrid mode and full vocabulary here. Using the reduced vocabulary in hybrid mode slightly reduces execution time, but not nearly enough to counter the reduction in fidelity.

Each remaining scenario differs from the baseline by varying the availability of a single resource. In the battery scenario, the client is battery-powered with an ambitious battery lifetime goal of 10 hours. Energy use is critical, so Spectra chooses the remote mode. As before, the small energy and latency benefits of using the reduced vocabulary do not outweigh the decrease in fidelity.

The network scenario halves the bandwidth between the client and server. Spectra correctly chooses hybrid execution and the full vocabulary in this scenario. The CPU scenario loads the client processor. Spectra chooses remote execution since the cost of doing the first recognition phase locally outweighs the benefit of reduced network usage.

In the cache scenario, the server is made unavailable and the 277 KB language model for the full vocabulary is flushed from the client's cache. Spectra uses the reduced vocabulary since the cache miss makes full fidelity recognition approximately 3 times slower than the reduced case.

Though preliminary, these results are encouraging, since Spectra chooses the best execution mode in each scenario. Further, the overhead of using Spectra to choose an alternative is within experimental error in all cases.

Scenario	Local/Reduced (Fidelity = 0.5) Time (s.)	Energy (J.)	Local/Full (Fidelity = 1.0) Time (s.)	Energy (J.)	Hybrid/Reduced (Fidelity = 0.5) Time (s.)	Energy (J.)	Hybrid/Full (Fidelity = 1.0) Time (s.)	Energy (J.)	Remote/Reduced (Fidelity = 0.5) Time (s.)	Energy (J.)	Remote/Full (Fidelity = 1.0) Time (s.)	Energy (J.)
baseline	37.4 (0.1)		69.2 (0.5)		7.8 (0.6)		8.7 (0.7)		9.3 (0.6)		10.3 (0.3)	
battery	37.4 (0.0)	22.6 (0.2)	69.2 (0.6)	43.5 (0.5)	7.3 (0.2)	3.5 (0.0)	8.6 (0.6)	3.6 (0.1)	9.2 (0.4)	2.4 (0.1)	10.2 (0.5)	2.5 (0.1)
network	37.4 (0.2)		69.8 (0.4)		9.2 (0.1)		10.5 (0.6)		22.2 (3.7)		21.4 (4.3)	N/A
CPU	75.2 (0.4)		137.6 (0.6)		12.4 (1.2)		12.7 (0.1)		10.8 (1.4)		12.0 (2.7)	
cache	36.6 (0.2)		105.4 (0.4)									

(a) Time and energy cost of each possible execution alternative

Scenario	Best Alternative	Chosen Alternative	Time (s.)	Energy (J.)	Fidelity
baseline	Hybrid/Full	Hybrid/Full	8.7 (0.8)		1.0
battery	Remote/Full	Remote/Full	10.6 (1.2)	2.7 (0.3)	1.0
network	Hybrid/Full	Hybrid/Full	10.7 (1.1)		1.0
CPU	Remote/Full	Remote/Full	12.0 (1.2)		1.0
cache	Local/Reduced	Local/Reduced	36.7 (0.2)		0.5

(b) Results of using Spectra to select an alternative

This figure shows how Spectra adapts the behavior of a speech recognizer in the resource availability scenarios described in Section 4. Part (a) shows the value of the three user metrics considered by Spectra (execution time, energy use, and fidelity) for each of the six possible execution alternatives. The highlighted alternative is the one that best satisfies the evaluation criteria for the speech application. Part (b) shows the results of using Spectra to select an alternative—it lists the best possible alternative, the alternative actually chosen by Spectra, and the values of the three metrics. Energy use is only measured in the battery scenario since the client operates on wall power in all other scenarios. Each result shown is the mean of five trials—standard deviations are shown in parentheses.

Figure 2. Spectra speech recognition results

5 Related work

Spectra's uniqueness derives from its focus on pervasive computing. It is the first remote execution system to monitor battery and cache state, support self-tuning operation, and balance performance goals with battery use and fidelity.

As the field of remote execution is enormous, we restrict our discussion of related work to the most closely related systems. Rudenko's RPF [18] considers both performance and battery life when deciding whether to execute processes remotely. Kunz's toolkit [12] uses similar considerations to locate mobile code. Although both monitor application execution time and RPF also monitors battery use, neither monitors individual resources such as network and cache state, limiting their ability to cope with resource variation.

Kremer et al. [11] propose using compiler techniques to select tasks that might be executed remotely to save energy. At present, this analysis is static, and thus can not adapt to changing resource conditions. Such compiler techniques are complementary to Spectra, in that they could be used to automatically select Spectra operations and insert Spectra calls in executables.

Vahdat [22] notes issues considered in the design of Spectra: the need for application-specific knowledge and the difficulty of monitoring remote resources.

Several systems designed for fixed environments share Spectra's self-tuning nature. Coign [9] statically partitions objects in a distributed system by logging and predicting communication and execution costs. Abacus [2] monitors network and CPU usage to migrate functionality in a storage-area network. Condor monitors goodput [3] to migrate processes in a computing cluster.

6 Conclusion

Remote execution lets pervasive applications leverage both the mobility of small devices and the greater resources of large devices. Our initial results with Spectra show that this benefit can be effectively realized if the system monitors pervasive resources, balances multiple goals in evaluation, and supports self-tuning operation.

Yet, much work remains to be done. Our early experience with Spectra suggests that predictions often involve tradeoffs between speed and accuracy. For example, when estimating remote CPU availability, Spectra might use a slightly stale cached value, or it might query the server to obtain more accurate information. If the difference between possible alternatives is slight, as for example with short-running operations, Spectra would do better to make a "quick and dirty" decision. However, when alternatives differ significantly, Spectra should invest more effort to choose the optimal alternative. This suggests to us that Spectra it-self should be adaptive—it should balance the amount of effort used to decide between alternatives against the possible benefit of choosing the best alternative.

Since resource logs can grow quite large for complex operations, we hope to develop methods for compressing log data without sacrificing significant semantic content. We also plan to investigate how the importance functions used in evaluation can be modified with simple user interfaces. Finally, we wish to evaluate Spectra using more dynamic resource scenarios.

Acknowledgements

Many people contributed to this paper. Keith Farkas, Lawrence Brakmo, Deborah Wallach, Bill Hamburgen, and the rest of the Itsy team at Compaq Western Research Lab provided a great deal of software and support that aided us in porting applications to the Itsy platform. Jan Harkes and Shafeeq Sinnamohideen assisted with the Coda file system. Rajesh Balan, Keith Farkas, David Petrou, and the anonymous reviewers gave us several helpful comments that improved the paper.

This research was supported by the National Science Foundation (NSF) under contracts CCR-9901696 and ANI-0081396, the Defense Advanced Projects Research Agency (DARPA) and the U.S. Navy (USN) under contract N660019928918, IBM Corporation, Nokia Corporation, Intel Corporation, and Compaq Corporation. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the NSF, DARPA, USN, IBM, Nokia, Intel, Compaq, or the U.S. government.

References

[1] Adjie-Winoto, W., Schartz, E., Balakrishnan, H., and Lilley, J. The Design and Implementation of an Intentional Naming System. *17th ACM Symp. on Op. Syst. and Princ.*, pages 202–16, Kiawah Island, SC, Dec. 1999.

[2] Amiri, K., Petrou, D., Ganger, G., and Gibson, G. Dynamic Function Placement for Data-Intensive Cluster Computing. *USENIX Annual Tech. Conf.*, San Diego, CA, June 2000.

[3] Basney, J. and Livny, M. Improving Goodput by Co-scheduling CPU and Network Capacity. *Intl. Journal of High Performance Computing Applications*, 13(3), Fall 1999.

[4] Bellosa, F. The Benefits of Event-Driven Energy Accounting in Power-Sensitive Systems. *9th ACM SIGOPS European Workshop*, Kolding, Denmark, Sept. 2000.

[5] Dallas Semiconductor Corp., 4401 South Beltwood Parkway, Dallas, TX. *DS2437 Smart Battery Monitor*, 1999.

[6] Flinn, J. and Satyanarayanan, M. Energy-Aware Adaptation for Mobile Applications. *17th ACM Symp. on Op. Syst. and Princ.*, pages 48–63, Kiawah Island, SC, Dec. 1999.

[7] Flinn, J. and Satyanarayanan, M. PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications. *2nd IEEE Workshop on Mobile Comp. Syst. and Apps.*, pages 2–10, New Orleans, LA, Feb. 1999.

[8] Hamburgen, W. R., Wallach, D. A., Viredaz, M. A., Brakmo, L. S., Waldspurger, C. A., Bartlett, J. F., Mann, T., and Farkas, K. I. Itsy: Stretching the Bounds of Mobile Computing. *IEEE Computer*, 13(3):28–35, Apr. 2001.

[9] Hunt, G. C. and Scott, M. L. The Coign Automatic Distributed Partitioning System. *3rd Symposium on Operating System Design and Implemetation*, New Orleans, LA, Feb. 1999.

[10] Kistler, J. and Satyanarayanan, M. Disconnected Operation in the Coda File System. *ACM Trans. Comput. Syst.*, 10(1), Feb. 1992.

[11] Kremer, U., Hicks, J., and Rehg, J. M. Compiler-Directed Remote Task Execution for Power Management. *Workshop on Compilers and Operating Systems for Low Power*, Philadelphia, PA, Oct. 2000.

[12] Kunz, T. and Omar, S. A Mobile Code Toolkit for Adaptive Mobile Applications. *3rd IEEE Workshop on Mobile Comp. Syst. and Apps.*, pages 51–9, Monterey, CA, Dec. 2000.

[13] Lorch, J. R. and Smith, A. J. Apple Macintosh's Energy Consumption. *IEEE Micro*, 18(6):54–63, Nov./Dec. 1998.

[14] Narayanan, D., Flinn, J., and Satyanarayanan, M. Using History to Improve Mobile Application Adaptation. *3rd IEEE Workshop on Mobile Comp. Syst. and Apps.*, pages 30–41, Monterey, CA, Aug. 2000.

[15] Nichols, D. Using Idle Workstations in a Shared Computing Environment. *11th ACM Symp. on Op. Syst. and Princ.*, pages 5–12, Austin, TX, Nov. 1987.

[16] Noble, B. D., Satyanarayanan, M., Narayanan, D., Tilton, J. E., Flinn, J., and Walker, K. R. Agile Application-Aware Adaptation for Mobility. *16th ACM Symp. on Op. Syst. and Princ.*, pages 276–87, Saint-Malo, France, Oct. 1997.

[17] Petrou, D., Narayanan, D., Ganger, G., Gibson, G., and Shriver, E. Hinting for Goodness' Sake. *8th Workshop on Hot Topics in OS*, May 2001.

[18] Rudenko, A., Reiher, P., Popek, G., and Kuenning, G. The Remote Processing Framework for Portable Computer Power Saving. *ACM Symp. Appl. Comp.*, San Antonio, TX, Feb. 1999.

[19] Satyanarayanan, M. Pervasive Computing: Vision and Challenges. *To appear in IEEE Personal Communications*.

[20] Satyanarayanan, M. and Narayanan, D. Multi-Fidelity Algorithms for Interactive Mobile Applications. *3rd Intl. Workshop on Discrete Alg. and Methods in Mobile Comp. and Comm.*, pages 1–6, Seattle, WA, Aug. 1999.

[21] Stemm, M. and Katz, R. H. Measuring and Reducing Energy Consumption of Network Interfaces in Hand-Held Devices. *IEICE Trans. Fundamentals of Electr., Comm. and Comp. Sci.*, 80(8):1125–31, Aug. 1997.

[22] Vahdat, A., Lebeck, A., and Ellis, C. Every Joule is Precious: The Case for Revisiting Operating System Design for Energy Efficiency. *9th ACM SIGOPS European Workshop*, Kolding, Denmark, Sept. 2000.

[23] Viezades, J., Guttman, E., Perkins, C., and Kaplan, S. *Service Location Protocol*. IETF RFC 2165, June 1997.

[24] Waibel, A. Interactive Translation of Conversational Speech. *IEEE Computer*, 29(7):41–8, July 1996.