THE REMOTE PROCESSING FRAMEWORK FOR PORTABLE COMPUTER POWER SAVING[1]

Alexey Rudenko Peter Reiher Gerald J. Popek[2] Geoffrey H. Kuenning[2]
University of California, Los Angeles
{arudenko, reiher@cs.ucla.edu, geoff@cs.hmc.edu, popek@platinum.com}

KEYWORDS: remote processing, power saving, client server, laptop battery, wireless communication

ABSTRACT

Recent research has demonstrated that portable computer users can save battery power by migrating tasks over wireless networks to server machines. Making this technique generally useful requires considerable automation. This paper describes a framework for automatically migrating tasks from a portable computer over a wireless network to a server and migrating the results back. The paper presents the framework's architecture, discusses key issues in creating the framework, and presents performance results that demonstrate that the framework is both useful and power-inexpensive.

I. INTRODUCTION

The continuing limitations on battery power make power management one of the most challenging problems in portable computing. Various power conservation techniques have been proposed, including turning off the portable computer's screen [2], optimizing hard drive I/O [7], slowing down the CPU [5], etc. Recently, process migration over wireless networks has been proposed to conserve power [11], [13]. Power-consumptive processes are migrated over wireless networks to a server that performs the actual computation, and the results are migrated back.

This technique has its own difficulties and drawbacks. The power consumption of radio communication is especially important, because presumably if a user does not have access to an AC power source, he also does not have access to a network connector. A wireless network must provide ubiquitous access to certain network services through radio. But radio communication devices also use power; [15] shows that one typical device is itself a heavy power consumer.

If the portable computer and server must each maintain copies of the data necessary to perform the task, maintaining consistency of persistent data is also a problem. Ideally, the server would permanently store replicas of the data. The portable computer would transport recently updated files to the server. The power costs of this data transport must be traded off against the power consumed for the task processing itself. Sometimes moving the processing to the remote server is cheaper, sometimes processing it locally is cheaper. This decision is potentially complex, but it can have a large impact on power consumption. System support for making this decision is desirable.

This paper presents the Remote Processing Framework, which provides system support for making such decisions and handling the mechanical issues of process migration. The paper compares the power consumption obtained from local and remote processing using the Remote Processing Framework to determine when migrating a process is worthwhile for one important class of tasks. These results show the importance of proper decision-making for such a framework.

The response time depends on the speed of the server. In some cases the response time can be slower than in the case of a local run, but the user will be willing to accept this in return for battery savings.

II. RELATED WORK

Two groups have recently and independently investigated this power saving technique. Rudenko et al., in [13] describe experiments using portable machines with WaveLAN radio devices, using experimental methods. This paper showed that significant power can be saved through remote processing for several realistic tasks. Othman [11] used simulation to show that battery life can be extended through process migration. The authors ran simulations of servers with different workload and bandwidth characteristics. The simulation showed that highly utilized mobile hosts are more likely to benefit from job migrations. If the CPU utilization is low, transferring jobs does not result in the CPU remaining idle long enough to lead to substantial savings. The paper offers some decision-making

[1] This work was partially supported by the Defense Advanced Research Agency Projects Agency under contract DABT63-94-C-0080.

[2] Gerald Popek is also affiliated with PLATINUM *technology, inc.* Geoff Kuenning is now affiliated with the Computer Science Department of Harvey Mudd College.

algorithms such as History and Adaptive Load Sharing (ALS), which learns and adapts its decision based on previous CPU time measurements for a particular process. The paper does not discuss how the algorithm adapts to changes in noisy communication channels, which can have a significant impact on power consumption.

A related but different approach to remote processing is taken by the InfoPad portable terminal system [10]. The InfoPad is a network I/O device with no computational power, relying on network servers to run major processes. Local computing is not possible, so this system does not utilize the techniques discussed here.

Jain [8] discusses optimizing distributed database operations that include portable computers, including process migration. One dimension for optimization that is considered is battery power. The authors use analytical methods to study this issue.

DCOM [4], Rover [9] and CORBA [16] offer programming tools that allow a designer to build distributed applications. These systems could be used to build the kind of framework discussed here, and could allow finer granularity of migration by dividing applications into power-consumptive and power-conservative parts. However, they have not yet been used for this purpose.

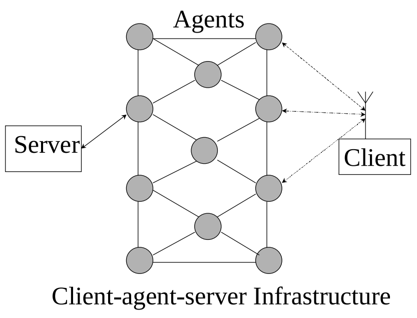

Figure 1: Client-agent-server Infrastructure for the Remote Processing Framework

III. DESIGN OF THE REMOTE PROCESSING FRAMEWORK

The design of the Remote Processing Framework (RPF) is based on a client-agent-server paradigm [1] (see Figure 1). On the client's side the framework automatically decides whether to run a process remotely or locally, moves necessary portions of the user's file system to the remote machine, runs the process locally or remotely, and handles the database of processes. On the server side the package performs the operations the user requested and reliably communicates the results back. RPF is written in C++ running on Linux and uses standard tools for data compression and transportation.

In an office-based RPF system, the server would be permanently assigned for the mobile client machines, and would permanently store replicas of files required for remote processing. For example, in the tests reported here processes were migrated to a Pentium server. The laptops were disconnected from both AC power and the wired network, but is equipped with WaveLAN cards. In more general cases, the mobile clients might move anywhere, and a ubiquitous agent infrastructure would be necessary to serve such travelers. This paper will only discuss the simpler office system, but the same principles could be used more generally with some extensions.

To run processes remotely the client needs to replicate its file system on the server. Relocating the whole file system from the client to the server is very power-costly, so moving only the modified data is preferable. The server must have access to the whole user file system, but if it has already been replicated when the client was tethered and power was not an issue, the server only needs to receive the updates since that time before it can run the process. When the process completes, the results must be sent to the client.

Remote execution is illustrated in Figure 2:

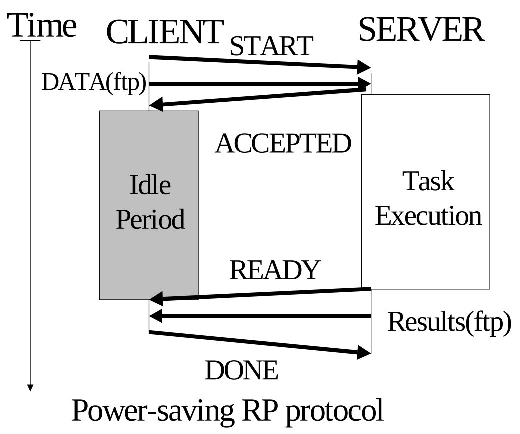

Figure 2: RPF protocol of the remote processing in client server implementation

1. The client and the server start with consistent copies of the data.
2. The client modifies the data and keeps track of its modifications.
3. The client decides to run the process remotely and sends its modifications and the process description to the server.
4. The server verifies whether it has enough resources and an appropriate environment to run the process. If so, the server sends an acknowledgment to the client.
5. The client, after receiving the acknowledgment, can run other jobs, remain idle, put its radio communication device into sleep mode, or suspend itself for power saving.

6. The server applies the user modifications to its local replica and executes the process. After the successful execution of the task, it sends a message to the client.

7. If the client is awake, it downloads the results from the server and acknowledges completion. If the client or its radio device is sleeping, the server waits. When the client awakens it sends the server a message to verify whether the task has completed. If the client awakens too early, the server informs the client, and the client can wait or sleep more.

If the client decides to execute the task locally in step 3, then steps 4-7 are not performed.

III.1 RPF Data and Control Protocol

An important component of RPF is the protocol used to transfer data and control messages between the client and server machines. We present an outline of the protocol here; consideration of error recovery is omitted to save space. For more information see [14].

The CLIENT runs on a portable computer equipped with a WaveLAN card. The SERVER runs on a server machine also equipped with a WaveLAN.

When the SERVER receives a START message with CLIENT information that contains the location of user modifications and instructions on running the process, it forks a child process to handle this CLIENT. The SERVER assigns a new communications port for the child, then returns to the waiting state. It can now work on another client request.

The child process verifies the consistency of resources between client and server machines and downloads the modified files to the server's replica. If the data was downloaded successfully, it sends an ACCEPTED message to the CLIENT; if not, NACCEPTED is sent. ACCEPTED contains a verification number assigned to this process by the SERVER. After getting the ACCEPTED message, the CLIENT can stay idle, put its WaveLAN into sleep mode, or enter suspend mode.

After sending the ACCEPTED message, the SERVER child forks a grandchild to handle process execution. The grandchild runs the process according to the instructions obtained from the CLIENT. When the processing is finished, the grandchild dies and the SERVER child sends a READY message to the CLIENT. If the CLIENT is awake, it downloads the results of the execution and the screen output of the executed process. This protocol assumes that no concurrent updates are generated

If the CLIENT is not awake, it cannot get the READY message, and the responsible SERVER child must wait until the CLIENT awakens. When the CLIENT awakens it sends an IS_READY message to the SERVER and the SERVER sends the READY message again. Then the CLIENT downloads the result. If the result was downloaded successfully, the CLIENT sends a DONE message to the SERVER. After getting DONE, the child responsible for communication dies.

III.2 The Process Database

All processes that ever use the RPF are recorded in a process database. Each process is identified by the command line that runs it and the path to the directory where the process runs. The database record contains the description of the environment, platform, and OS for the process. Each record of a process contains the list of files that should not be transferred from client to server and a list of files that should not be transferred from server to client even if they are modified. For example, if the process is a compilation, object files should not be transferred from server to client nor executables from client to server. The record of a process also contains temporary information obtained from the SERVER in the ACCEPTED message, like the number of the I/O port of the child, status of the process, etc. The record also keeps the statistics about power consumption by the process (mean and standard deviation) for the particular size of the user-modified information. The record is updated twice during the processing: after the ACCEPTED message is obtained, and after the processing is over. The mean and standard deviation are used for the calculation of confidence intervals used for the comparison of power costs.

III.3 Client/Server Resource and Environment Verification

To run the process remotely, RPF needs a consistent environment on both machines. While the RPF currently runs only on a limited variety of platforms and operating systems, in principle it could be ported to many others. But interoperation between different platforms or operating systems would be difficult, so the RPF must verify the consistency of:

1) Platforms: e.g., Sparc/x86, big-endian vs. little-endian

2) OS: Linux, UNIX, Windows95, etc.

3) Software packages: G++, libraries, tar, zip, etc.

4) Environment variables

5) Hard drive space, to ensure that enough is available for the process

6) CPU workload

Depending on characteristics of particular process, other environmental factors may prove important. Checks for these will be added as they are discovered.

III.4 The Rumlet Procedure

The client must be able to identify updated files and transfer them to the server. This process must be very cheap in terms of power consumption, or it will negate any gains from migration. Modifications that the user committed to his files can be recorded by trapping updates. Another possibility is to scan the directory and find all files whose *mtime* attribute is newer than one prerecorded in a special database.

Both approaches have advantages and disadvantages that can potentially have significant effects on power savings. For the current research we have chosen to implement scanning because it allows us to avoid kernel modifications, thus increasing the portability and reliability of the code.

Several file replication services use scanning [6], [12], but these systems do not consider power management issues. They tend to provide generality not required for RPF, at the cost of power-expensive operations. We designed the Rumlet program to scan a user's file system and record attributes of files to a database using little power. Modified files are recognized through comparison of their attributes to their recorded attributes in the database. The Rumlet procedure runs on the client site to find user-modified files and on the server site to find results, i.e. files newly produced by the process.

III.5 Off-the-Shelf Packages

Some processes were incorporated into RPF from standard packages. **ncftp** was used to transport user data and results between CLIENT and SERVER. **tar** and **gzip** are used for the conversion of multiple files that must be transported into one compressed file. The power costs of the extra processing for these conversions are compensated by the decreased power spent transporting smaller data packages. If a file is encrypted or zipped already, i.e. the further zipping will not be able to make the file smaller, this step can be skipped.

III.6 Decision-Making

The client must decide whether to run processes remotely. [13] showed that data size is the main factor in the power cost of the processing. But it is not the only cost. In noisy environments the power cost of remote processing grows dramatically. Also, changes in the process workload may affect the choice of where to run this process. But if the workload was changed accidentally, for example if the process fails because of a syntax error, then future decisions should not be influenced. Thus the decision-making procedure must be adaptive [11], but also must have sufficient conservatism not to be skewed by errors. We implemented a decision-making procedure for RPF according to the following principles:

1) Data gathering

RPF initially runs the same process alternatively, locally and remotely, encouraging the user to use battery power on his laptop in both cases and measuring the cost of both alternatives. This method obviously puts an extra burden on the laptop's battery; but sometimes it is impossible to predict which alternative gives the best results without trying both.

2) Primary decision-making

After RPF gets enough measurements to determine that one of the choices is better, it runs the process that way on most future occasions.

3) Validity checking

Occasionally RPF deliberately makes the "wrong" decision, and runs the process the other way. If, for example, it was decided to run the process locally, occasionally RPF runs the process remotely to be sure that the decision is still right. If the "wrong" way appears better now, RPF does not immediately switch. It continues to run processes in the

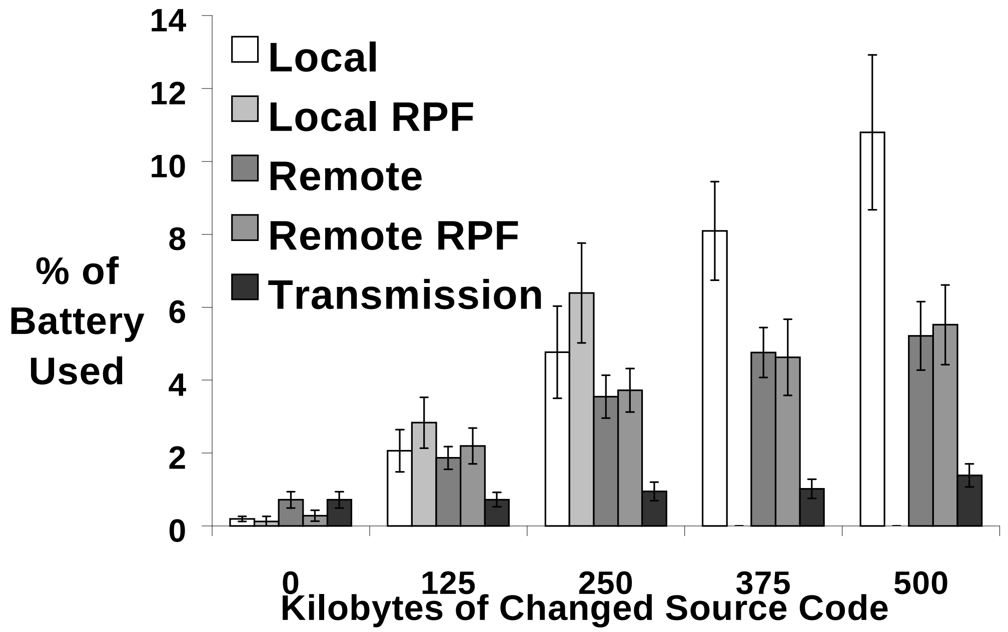

Figure 3: Comparisons of local and remote processing with and without RPF.

previously chosen way with shorter periods between "wrong" runs.

If the power costs of running the process in the previously chosen manner change dramatically, the alternative decision will be checked more often. Such fluctuation in a task normally run locally might be caused by the growth of the process workload, possibly making remote processing better.

If the power costs of running the process in the previously chosen manner demonstrate stability, the alternative decision will be checked less often.

4) Filtering

Sometimes changes in the power consumption of RPF tasks are the result of transients in network noise or statistical fluctuation, which should not be responded to rapidly. RPF continues to run the process in the initially chosen way until it is reasonably sure that the differences are due to continuing conditions. The formula used for the estimation of local and remote costs is the following simple digital filter:<Estimated power cost>= (1-α) *< Last N measured power costs / N > + α *< Last estimated power cost >, where $N > 0$ and α is the coefficient of conservatism of the system. If changes in the environment are rare, then the coefficient should be low and the result depends more on recently measured values. If changes happen often, and any particular change is not reason enough to change the decision, then the coefficient should be high and the current decision will depend more on the previous decision.

IV. EXPERIMENTS

We ran 70 experiments, consuming 200 hours of wall clock time, to measure the RPF. Each point plotted below typically represents four to eight hours of experiments. The processes

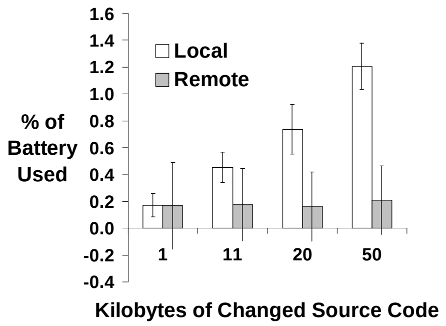

Figure 4: The power costs for local and remote compilation of the RPF package

tested here were compilations of real packages; one of them was RPF itself. The experiments were run using a Dell laptop with 486 100 mHz processors as the mobile computer, and a Pentium Pro PC running at 200 mHz as a server. All machines were equipped with WaveLAN radio cards. The server was also in regular use for other tasks. The queue waiting for processing on the server usually contained 0.7 to 0.8 tasks (not counting experimental tasks), which is a relatively small workload. The radio channel was mostly idle, but neighboring laboratories were using WaveLAN cards for their own research. In more realistic environments the server will serve more than one client through radio communication devices, and the radio channel will carry a significant amount of other traffic. The methodology of measurements is the same as the experiments described in [13]. All results are shown with 95% confidence intervals.

IV.1 RPF Power Cost

Figure 3 shows the costs of running the RPF itself. Local and remote compilations without RPF are compared with compilations using RPF. The X axis shows the size of the modified source code. The Y axis shows the power cost in percent of the battery life. For each amount of code changed, five bars are shown. The first shows the cost of the local compilation without using RPF and without running a WaveLAN card. The second shows local costs when RPF and a powered-up WaveLAN card are included. Running RPF without radio would be pointless, so this bar realistically shows the cost of having the option of migrating when the system chooses not to exercise it.

The third bar shows the costs of processing the compilation remotely, by hand and without RPF, as cheaply as possible. The fourth bar shows the costs of running remotely with RPF, rather than handcrafted methods. Finally, the fifth bar shows the power cost of merely transferring the altered files over the radio channel and the result files back. This bar suggests the lower bound on how little power would be consumed if the server were infinitely fast and there were no overhead costs.

Adding the WaveLAN card to the mobile machine increases the cost of processing proportionally to the duration of the process. In most cases, remote processing with RPF and without RPF is statistically indistinguishable, which means that the cost of RPF is negligibly small. At the zero point, the cost of Remote RPF shows the direct cost of a pure RPF protocol message exchange without useful processing and transporting, which is equal to 0.2 ± 0.1 percent of battery life.

The use of power-costly WaveLAN cards increases the cost of local processing that doesn't use them, but that is the price of having a communication service. The extra power cost of the WaveLANs grows with the duration of the local experimental runs, which depends on the amount of modified code. Because of this obvious trend in power costs, the measurements at the points 375- and 500-Kb were not necessary.

IV.2. The Necessity of Decision Making

If a given process should always be run locally (or remotely) regardless of how much of its data changed, the RPF would not need a decision capability. If the extrapolated lines of local and remote cost cross, the decision should be changed at the crossover point. If the crossover point is too close to 0, we do not need RPF's decision facility because we should always run the process remotely. If the crossover point is very far from 0, RPF's decision facility is again less necessary because users seldom change that much code, and can therefore afford the occasional power costs.

The first series of experiments compiled the RPF package itself. This package contains about 100K of source code, with executables of about 500K. The result is shown in Figure 4. For this package it nearly always pays to run the process remotely. The cost of remote processing grows very little because the speed of compilation on a fast server is so high that most of the cost is in transportation of source and executables.

Another series of tests used the code for the Rumor replicated file system [12]. This source code is about 500K, and executables are 7MB for a stripped and 28MB for unstripped executables. Usually the user is more interested in the unstripped version, because multiple recompilations occur during debugging, when the unstripped version is valuable. The results of these experiments are shown in Figures 5 and 6 for stripped and unstripped versions, respectively. The crossover for the stripped version is located between 11 and 20 KB of modified source code, corresponding to about 550 lines of C++. The crossover for the unstripped version is at 20 KB of modified source code, about 770 lines of source code.

Therefore, RPF's decision capability is useful for Rumor compilations because recompilation is common on both sides of the crossover point. The decision capability allows battery savings of around .5 percent per compilation for these realistic cases.

The remote bars demonstrate much slower growth with the growth of the modified user source code because the server is significantly faster than the portable computer, so the time required for remote execution grows slowly. For larger ranges of code changes, the growth of the remote power costs is noticeable. For example, consider the large changes in remote power costs in Figure 3 as the size of the modified source code

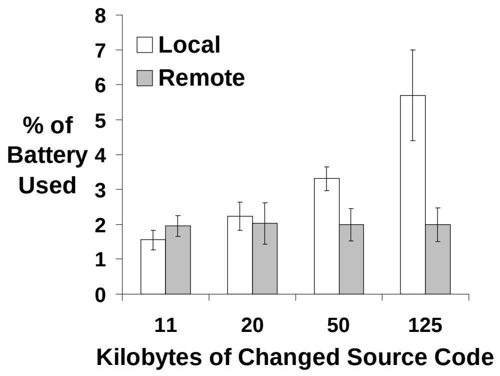

Figure 6: The power costs for local and remote compilation of Rumor for unstripped executables

grows from 0 to 500 Kb.

The confidence intervals of the remote bars on Figure 4 are relatively wide and cross the X-axis, due to the very low precision of the APM power measurement tool. The difference between two consecutive measurements can be up to 4 percent, and the second measurement can even show higher battery capacity than the first one.

[13] discusses advantages and disadvantages of this form of measurement method. This research did not use external electric power-measuring devices because of the difficulty of achieving a reliable experimental setup. Also, a typical running implementation of the RPF would not have such devices, and the system should use only the devices that are available to the average user on his laptop.

V. SUITABLE APPLICATIONS FOR RPF

The value of RPF depends on how many common applications have the characteristics demonstrated above for compilation. Some applications, such as MS Word, will not gain anything from it. [13] demonstrated that processes with high I/O and CPU activity (such as Gaussian elimination) can achieve very high percentages of power saving. Gaussian elimination's processing complexity is O(D**1.5), where D is the amount of data to be processed. Its transmission complexity is only O(D). The measurements in [13] showed that statistically noticeable power saving can be observed for matrices bigger than 500x500. Any process with complexity O(D**k) with k>1 will save power when run remotely for D>Do, where Do = (α) ** (1/(k-1)) and α is tangent to the transmission power cost.

Other tasks that seem suitable for RPF include:

1) Large simulation and numerical applications
2) Graphic applications
3) Genetic algorithms
4) On-line analytical processing
5) Statistical applications

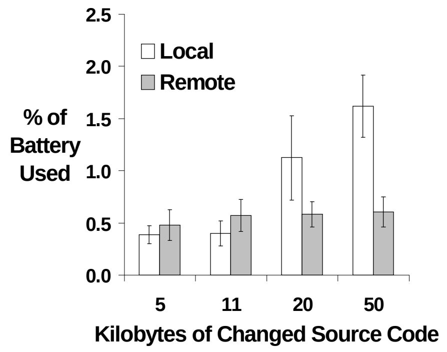

Figure 5: The power costs for local and remote compilation of Rumor for stripped executables

6) Other CPU-bound applications

All of these applications sometimes lead to large amounts of processing and/or disk I/O activity, but also sometimes are quite short. Therefore, RPF could be used to decide when to migrate such tasks in the expectation that particular runs will or will not consume large amounts of power.

The interactivity of a process also affects its suitability for RPF. If a user needs to communicate with his application running on the remote machine, some additional constraints appear. First of all, the user' s machine cannot be suspended during the idle period, but must stay alert to all remote calls.

Second, any intercommunication through a radio communication device is a very power-costly operation, so an application with a lot of interactivity will not win any power from the remote processing. Sometimes the amount of interactivity depends on unpredictable user behavior, in which case the framework's decision-making procedure will not be able to make consistent decisions.

VI. DISCUSSION AND FUTURE WORK

Our results show that remote execution of large tasks using RPF with a faster server can reduce power consumption by 3 to 6 times. However, remote execution when local execution is less power-costly can cause up to 0.5% loss of battery power per run. While this level of loss may seem insignificant, [13] showed that the power losses could be much greater in noisy environments. Therefore, the decision-making procedure must be intelligent enough to handle steady and bursty noise. Further studies of decision-making algorithms are needed to address this issue.

Redesigning RPF to permit decomposition of applications into separately migratable components using DCOM, Rover, or CORBA would increase the applicability of the methods. Measuring the power consumption of individual components would be challenging, however.

The existing RPF incorporates some, but not all of the features required to support the more general case of machines moving not just within an office environment, but anywhere. An additional module would be required to handle data transfer between a mobile machine and the user's home desktop or another server. This case also requires more care in maintaining data consistency among multiple replicas. Other improvements to RPF for this case would involve load management among server machines, search methods for finding servers suitable for handling particular remote executions, and migrating results to other servers to handle clients in motion during the process' execution. Security and privacy issues would also need to be addressed. All of these features would need to be added without increasing power consumption.

Further research is necessary to determine the best methods of making migration decisions for a wider variety of applications. Also, more effort is required to determine the best filtering techniques for handling process migration in environments with varying noise characteristics, since in such environments the costs of migration will also vary. Finally, wide deployment and field testing of the Remote Processing Framework would be desirable.

VII. CONCLUSION

Previous research demonstrated the possibility of improving the battery lifetime of untethered portable computers by using wireless networks to migrate large tasks to server machines. Migrating such tasks can save power from many sources on the portable computer, including the CPU, memory, disk, and screen. However, custom-crafted process migration required knowledge and care not practical for ordinary users.

We show here that a simple software framework can automate most of the difficult tasks of process migration for power management. By automating data consistency issues and handling the basic protocol synchronization to communicate between the portable client and its server, the Remote Processing Framework makes process migration for power management a feasible technique for the average user.

This research also answers the question of whether a useful Remote Processing Framework could be implemented without incurring large new overheads that overwhelmed the desired power savings. Earlier research using handcrafted process migration obtained benefits of up to 51 percent of the battery power expended [13]. This paper demonstrates that similar results are possible with an automated system requiring no user intervention. Given that users are already paying the cost of having a wireless network card in their portable computer, the extra cost of the Remote Processing Framework itself is around .2 percent of a typical portable computer's battery life for each invocation. Since migration can give benefits of 6 percent or more of the battery life for realistic tasks, this overhead is negligible. The key to lowering the costs of the framework was simplifying as many of the framework's details as possible, particularly its replication facilities.

This research further demonstrates the importance of having some degree of sophistication in the decision-making process that determines whether a given invocation of a task should be run locally or remotely. The same command can lead to very different power costs, depending on the state of the data involved. Incorrectly choosing to run a task either locally or remotely can be expensive in power. Fortunately, for important applications like compilation, the framework can automatically determine whether a particular invocation should be handled locally or remotely.

REFERENCES

[1] Ajay Bakre, B. R. Badrinath. M-RPC: A Remote Procedure Call Service for Mobile Clients, *Proceedings of the First International Conference on Mobile Computing and Networking, MobiCom'95*, Berkley, California, pp. 97-110, November, 1995

[2] James W. Davis. Power Benchmark Strategy for Systems Employing Power Management, *IEEE International Symposium on Electronics and Environment*, 1993.

[3] A.T. Galecki. NLMEM: a new SAS/IML macro for hierarchical nonlinear models, *Computer Methods and Programs in Biomedicine*, vol.55, no.3, Elsevier, March 1998, p.207-16.

[4] R. Grimes. *Professional DCOM Programming*, Olton, Birmingham, Canada: Wrox Press, 1997.

[5] David P. Helmbold, Darrell D. E. Long and Bruce Sherrod. A Dynamic Disk Spin-down Technique for Mobile Computing, *Proceedings of the Second Annual ACM International Conference on Mobile Computing and Networking*, Rye, NY, November 1996.

[6] John H. Howard. Using Reconciliation to Share Between Occasionally Connected Computers, *Fourth Workshop on*

Workstation Operating Systems, Napa, California, pp. 56-60, October 1993.

[7] Kinshuk Govil, Edwin Chan, Hal Wasserman. Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU, *Proceedings of the First International Conference on Mobile Computing and Networking, MobiCom'95*, Berkley, California, pp. 13-25, November, 1995.

[8] Ravi Jain and N. Krishnakumar. An asymmetric cost model for query processing in mobile computing environments, pp. 363-378, in *Wireless Information Networks*, ed. J. Holtzman, Kluwer Academic Publishers, 1996.

[9] Anthony D. Joseph, Alan F. deLespinasse, Joshua A. Tauber, David K. Gifford, and M. Frans Kaashoek, ROVER: A Toolkit for Mobile Information Access, *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pp. 156-171, Colorado, December, 1995

[10] Shankar Narayaswamy et al. Application and Network Support for InfoPad, *IEEE Personal Communications, Vol. 3, No. 2,* pp. 4-17, April 1996.

[11] Mazliza Othman, Stephen Hailes. Power Conservation Strategy for Mobile Computers Using Load Sharing, *Mobile Computing and Communications Review*, Vol. 2, No. 1, pp. 19-26, January 1998.

[12] Peter Reiher, Michial Gunter and Gerald Popek. A User-level File Replication Middleware Service, *Proceedings of the SIGCOMM Workshop on Middleware*, August 1995.

[13] Alexey Rudenko, Peter Reiher, Gerald J.Popek, Geoffrey H. Kuenning. Saving Portable Computer Battery Power through Remote Process Execution, *Mobile Computing and Communications Review*, Vol. 2, No. 1, pp. 19-26, January 1998.

[14] Alexey Rudenko Portable Computer Battery Power Saving Using a Remote Processing Framework, Master Thesis, Computer Science Department of UCLA, 1998.

[15] Mark Stemm, Randy H. Katz. Measuring and Reducing Energy Consumption of Network Interfaces in Hand-Held Devices, *Proceedings of 3rd International Workshop on Mobile Multimedia Communications (MoMuC-3*), Princeton, NJ, September 1996.

[16] Steve Vinoski. CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, *IEEE Communication Magazine*, Vol. 35, No. 2, February 1997.

Alexey Rudenko is a research assistant in the Computer Science Department at the University of California, Los Angeles. He received his M.S. degree from UCLA in 1998. His research interests focus on mobile computing, distributed file systems, and active networks.

Peter Reiher is an Adjunct Associate Professor of Computer Science at UCLA. Dr. Reiher received his Ph.D. from UCLA in 1987. He has worked on several distributed operating system projects at JPL and UCLA. Dr. Reiher's research interests include mobile computing, distributed operating systems, optimistic and predictive computing, and security for distributed systems.

Gerald J. Popek has been a Professor of Computer Science at UCLA since 1973. His academic background includes a doctorate in computer science from Harvard University. He co-authored "The LOCUS Distributed System Architecture," MIT Press, 1985, and has written more than 70 professional articles concerned with computer security, system software, and computer architecture. Dr. Popek was the founder of Locus Computing Corporation, and is currently also the Chief of Technology Officer for PLATINUM technology, inc.

Geoffrey H. Kuenning received his Ph.D. in computer science from UCLA. He is currently an assistant professor at Harvey Mudd College. He is currently performing research in mobile computing, distributed file systems, prediction, and clustering methods. Dr. Kuenning is a member of IEEE, CPSR, ACM, and a number of ACM SIGs.