Energy Optimizations for Mobile Terminals via Computation Offloading

Xiao Ma, Yong Cui, Lian Wang
Department of Computer Science
Tsinghua University
Beijing, P.R.China
Email: max11@mails.tsinghua.edu.cn
cuiyong@tsinghua.edu.cn
lianwang.cst@gmail.com

Ivan Stojmenovic
SEECS
University of Ottawa
Ottawa, Canada
TNList, School of Software
Tsinghua University
Beijing, P.R.China
Email: ivan@site.uottawa.ca

***Abstract*—With the rapid development of mobile networking and device capability, energy efficiency becomes an important design consideration due to the limited battery life of mobile terminals. Processing energy cost by CPU is one of the most significant power consuming components in mobile terminals. The emergence of mobile cloud computing (MCC) provides the opportunity to save processing energy through the way of offloading computation tasks to remote server(s). In recent years, considerable research has been devoted to computation offloading to achieve energy efficiency. This paper presents a comprehensive summary of recent work, investigates representative infrastructure of computation offloading, and analyzes key components and future design trends of energy efficient mobile applications.**

***Index Terms*—Mobile terminal, energy, computation offloading, cloud computing.**

I. Introduction

Mobile cloud provides a powerful and ubiquitous resource pool for mobile devices. In the meantime, CPU in mobile systems remains the most power-hungry component which consumes battery lifetime consistently and steadily. Alex Shye et al. [1] presented a power estimation model and provided insights about the power breakdown among hardware components. The experimental result demonstrates that although energy consumption widely varies drastically depending upon the workload (e.g. over 50% of the total system power is consumed by making phone calls frequently; conversely if only playing music, the DSP consumes significant power), the screen and the CPU are the two largest power consuming components.

Since wireless communication and distributed computing technology has improved dramatically in the recent years, using high-speed wireless connection and powerful computing and storage capability to handle computation tasks becomes possible. one popular technique is proposed to reduce the energy consumption of computation – remote execution. One can offload computational tasks of mobile devices to resourceful servers (e.g. high computing capacity laptop PCs or data center) in order to reduce the energy consumption of the clients. Specifically, the client sends the parameters of a task to the server(s); the server(s) executes the task and sends back the results. The energy of the device is saved only if the energy for sending the parameters and receiving the results is lower than the energy needed for executing the task locally on the client. Additionally, if the task's offloading time and execution time on the server are lower than execution time on the client, the performance of the mobile application will be improved.

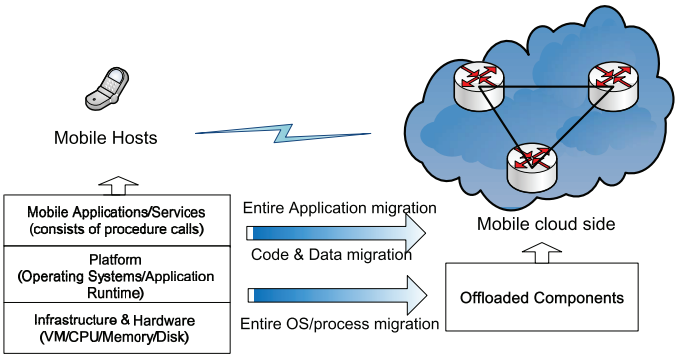

Fig. 1. general computation offloading schemes in mobile wireless environments

Fig. 1 shows general computation offloading schemes in mobile wireless environments. From the application layer perspective, the entire application or part of the code/data can be offloaded to the mobile cloud side to achieve better energy utility or better performance. The context and runtime of the application process is still maintained by operating system in mobile host. In the meantime, the whole execution platform (application runtime or infrastructure) can also be migrated to the cloud side, which indicates that the burden on programmers can be reduced.

The paper is organized as follows. First we give a brief overview of the concept of computation offloading, analyze the feasibility of saving energy through task migration. Then we present a survey of recent work in two aspects: fine-grained and coarse-grained offloading. Finally, to address future trends, we summarize and conclude the survey in the last section.

978-1-4673-2924-8/12/$31.00 ©2012 IEEE

II. Overview

The concept of offloading is not new. Since the 1970s, similar concepts, such as load balancing in distributed systems, have already been proposed. This concept still exists today in various forms. R. Balan et al. presented the concept of cyber foraging [2] for mobile devices and described it as a mechanism to augment the computational and storage capabilities of mobile devices through task distribution.

Can computation offloading really save energy? Plenty of theoretical analyses and practical experiment results ([3], [4]) provides a likely answer. Kumar et al. [3] provided a simple analysis for the decision on whether to offload computation to a server. The result shows that energy is saved if network bandwidth B is large enough while the ratio of (D/C) is low enough, where D represents the exchanged bytes of data between the server and mobile system and C represents the number of computation instructions. Portable1998 [4] evaluates the effectiveness of offloading techniques as determined through several experiments. The authors discovered that significant power savings are possible for certain common tasks of realistic size. The observed result indicates that up to 51% energy is saved when large tasks are offloaded, which means larger tasks (as determined by the number of computation instructions) are likely to benefit from offloading significantly. In the meantime, they also showed that more efficient ways of moving the data have the ability to provide greater savings.

For the last two decades, there have been many attempts to enable mobile devices to use remote execution for the purpose of improving energy efficiency and application performance. These approaches reduce application execution time on mobile devices, thus decreasing the energy consumption of both CPU and memory. These attempts could be classified into two approaches discussed in the following paragraph. The first approach involves fine-grained, energy-aware offloading of mobile code to the infrastructure. This approach relies on programmers to modify the program to handle partitioning, state migration, and adaptation to various changes in network conditions. Application can offload only part of the methods which benefits from remote execution to gain large energy savings. For instance, a media streaming application contains a decoder part and a video player part, with the former being a CPU-intensive part (and thus mainly consuming energy for the CPU and memory). As such, we could simply offload this CPU-intensive part without offloading any of the screen-intensive part. The second approach is coarse-grained task offloading scheme in which full process/program or full virtual machine is migrated to the infrastructure, and then programmers do not have to modify the application source code to take advantage of computation offloading. This approach reduces the burden placed on programmers. However, some parts of the program (e.g. those that are user interactive part) may not benefit from remote execution. Additionally, whole program migration may result in additional transmission overhead. Table I lists the related researches classified by: a) Type of server; b) Network environment; c) Partition Scheme; d) Partition granularity.

III. Fine-grained Offloading

As mentioned above, computation offloading can save energy only if the energy for transmission is lower than the local execution energy cost. Fine-grained offloading is intuitively proposed to transmit as little data as possible by partitioning program and offloading only the power-hungry parts of the application. As for transmission energy, network condition should also be considered. In order to handle partitioning, state migration, and changes in network conditions, one common approach for computation offloading is to rely on programmers to modify the existing program.

A. Static partitioning scheme

There are many early attempts on program offloading, most of which highly rely on programmers to statically split applications into two pieces. Protium [7] manually splits program code into viewers and services, one runs near the user, while the other runs in a service provider that is highly available, with persistent storage and abundant computation cycles. Viewers and services communicate via an application-specific protocol. Applications are built as if only a single viewer-service pair exists, with certain additional constrains. These constrains allow the Protium infrastructure to support connection, reconnection, multiple simultaneous viewers and session management. The authors implemented a prototype system of the infrastructure, and took five specific applications as example. If the program is large, complex, and has tangled state and display management, it has to be rewritten anyway; the burden on programmers is excessively large. In the last part of the paper, the authors also present a simple ”how-to” guide on partitioning. To partition an application, one must focus on the application-specific protocol. The protocol designer should decide which application state is viewer-specific and which is service-mediated. At the same time, control signaling and data should be separated, and changes of network condition should be considered; for example, if delay is ignored, what works acceptably on a LAN may be unusable with a 1-second round-trip time. Although many of the design concepts of this work sound like platitudes, they present a new possible direction of mobile application design – rebuilding all of our old applications to live in a new world.

Since the programmers might not be aware of the program cost of CPU and memory, and network conditions (e.g. bandwidth, RTT) or communication requirement may change dynamically, therefore, the above partition schemes (manually split program statically) cannot guarantee minimum energy consumption of computation. Many approaches are proposed to improve it, one of which is to estimate the energy consumption (communication energy and computation energy) before the program execution, then design an algorithm to make the optimal program partitioning based on the trade-off between computation and communication cost. Once the decision has been made, the offloading management will keep the partition and execute until the task completion. Works in

TABLE I
COMPARISON OF DIFFERENT WORKS ABOUT COMPUTATION OFFLOADING

References	Type of Server	Network environment	Partition scheme	Partition granularity
[5]	Crowd	Dynamic	Automatic	Fine-grained
[6]	Crowd	Dynamic	Automatic	Fine-grained
[3]	N/A	Static	N/A	N/A
[4]	Single server	Static	Manually	Coarse-grained
[7]	Data center	Static	Manually	Fine-grained
[8]	Crowd	Static	Automatic	Fine-grained
[9]	N/A	Static	Automatic	Fine-grained
[10]	Single server	Static	Automatic	Fine-grained
[11]	N/A	Dynamic	Automatic	Fine-grained
[12]	N/A	Dynamic	Automatic	Fine-grained
[13]	Data center	Dynamic	Automatic	Fine-grained
[14]	Data center	Dynamic	Automatic	Fine-grained
[15]	N/A	Static	Automatic	Coarse-grained
[16]	N/A	Static	Automatic	Coarse-grained
[17]	Data center	Dynamic	Automatic	Coarse-grained
[18]	Data center	Static	Automatic	Coarse-grained

[3] [8] [9] [10] all belong to this scheme. The communication cost depends on the size of the transmitted data and the network bandwidth, and the computation cost depends on the number of program instructions.

Yang K. et al. [8] proposed a novel offloading service that can seamlessly offload some of the tasks of a mobile application from an MH (mobile handset) to nearby, resource-rich PSs (called surrogates). The approach considers a combination of multiple resources including CPU, memory, and communication cost (e.g. bandwidth resource), provides mobility support and enables multiple surrogates to be utilized. The proposed client-server system architecture of the offloading service is divided into several parts – monitor, offloading engine, class instrumenting module, etc.. Resource monitor detect the memory use, CPU utilization and current wireless bandwidth of MH. Offloading engine divides the application into one local partition and several remote partitions. The class instrumenting module is a process that transforms classes to be offloaded into a form that is suitable for remote execution in surrogates. The partitioning scheme proposed in this work is described as follows: it transforms an application into a directed graph, the vertex set refers to java classes, and the edge set refers to the interactions (invocations and data access) among classes. The paper provides a close approximation to an optimal solution, and divides one application to (k+1) partitions, including one unoffloadable partition and k disjoint offloadable partitions.

Similar contributions on partitioning algorithm are made by Li Z. et al. [10]. The offloading units are defined at the level of procedure calls (tasks). For a given program, they profile the information about computation time and data sharing, then construct a cost graph and apply a task mapping algorithm to statically divide the program into server tasks and client tasks. After constructing the cost graph,they apply the branch-and-bound algorithm to the cost graph to minimize the total energy consumption of computation and the total data communication cost. The main idea of this algorithm is to prune the search space to obtain an approximated solution. Four kinds of messages are used in their model: task_start, task_end, data_send, data_request. The original program code is modified in which the offloading procedure call parts are replaced by these messages, while the remote execution management achieve the context state migration depending on these messages.

Previous works always assume that both the communication cost and the computation time can be obtained before the execution. However, due to the complex wireless network condition, the communication cost is more difficult to determine. Therefore, to estimate these costs, predicting the bandwidth between the local and remote systems is a significant problem. Bandwidth08 [9] proposed a framework to consider the classical bandwidth predictors (such as Last value, Mean filter, Network weather service forecaster (NWS), etc.) synthetically. The framework unifies such decision models by formulating the problem as a statistical decision problem that can either be treated ”classically” or using a Bayesian approach. We can see from the experimental result shown in the paper that Bayes strategy (Bayes and Bayes+CP) performs significantly better than the traditional predictors. This predicting model is more general and could be used for reference by other offloading systems.

B. Dynamic partitioning scheme

The two approaches mentioned above have one thing in common: they both maintain static program partition during task execution, and if connection status (e.g. disconnection during the program execution) changes, the former partition is not likely to take advantage of newly available communication resources. Many attempts ([11], [12], [13]) have been proposed to deal with this problem.

Ou S. et al. [11] mainly investigated the surrogate unreachability when mobile devices move following random way point (RWP) mobility scheme. They model the failure recovery time and total execution time of pervasive applications running under the control of the offloading system. Assuming the application is performed with the presence of offloading and failure recoveries, when any failure occurs, the code should be

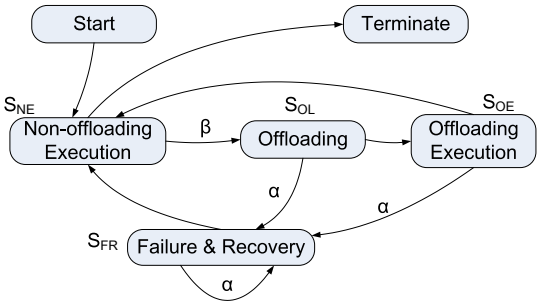

Fig. 2. Application execution state transition in offloading

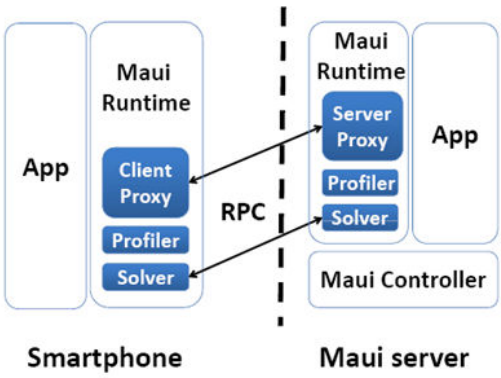

Fig. 3. High-level view of MAUI [13]'s architecture

re-offloaded. The state machine is shown in Fig. 2. This approach only re-offloads the failed subtasks, and thus reduce the execution time. However, the research has some deficiencies. First, the mobile wireless network environment is considered to be a wireless ad hoc local area network, which is too limited. Second, it only considers a disconnection event, with changes related to other types of network environment status (e.g. wireless bandwidth) not considered.

In order to cope with various mobile computing environment changes, a solution was proposed by [12] in 2006. This solution considers three common environmental changes (client side power level, connection status and bandwidth) and explains the suitable solutions for offloading in different environments. Client side power level obtains the highest priority because the battery change is non-reversible. The solutions involved in this paper are mainly general ones, and the detailed partitioning method is not involved.

MAUI [13] is a more intelligent solution. It enables developers to produce an initial partitioning of their applications with the minimal effort. Fig. III-B is the high-level view of MAUI's architecture. The proxy implements the decisions made by the MAUI solver, handling both control and data transfer based on the decision. The solver decides whether the method in program should be executed locally or remotely based on the input from the profiler. The profiler gathers the profiling information which is used to better predict whether future invocations should be offloaded or not.

The implementation details of each module is described as follows. Profiler: a) device profiling: power meter & linear regression model; b) program profiling: using past invocations of a method as a predictor of future invocations; c) network profiling: usage of PSM: more energy-efficient when RTT is long. It's unnecessary to use specialized tools to individually measure the round-trip time, bandwidth, and packet loss rate. They use a simple solution: sending 10KB to the server to obtain a fresh average throughput. Solver: taking profiler result as input, using 0-1 integer linear programming, 0 indicates executing locally and 1 indicates offloading.

This work simultaneously guarantees application's performance and energy consumption; furthermore, it reduces the burden on the programmer by automating many of the steps needed for program partitioning. However, the device profiling uses linear model, hence a lack of accuracy can be predicted. Additionally, the evaluation is based on a self-designed application, which may be too unique to be used generally.

Energy efficiency and performance optimization are not the only two consideration issues in MCC. Since the servers may have access to user's privacy, the security issues must also be considered. Chun B. G. et al. [14] take this issue into consideration. The authors pointed out that the modules containing sensitive data should be executed locally. The sensitive details are marked based on the programmer annotations during the process of code migration.

The key issue of fine-grained offloading is program partitioning. Partitioning brings extra overhead and the partitioning algorithm directly influences the efficiency of the offloading. As manual partitioning puts much burden on programmers and cannot adapt to changes of network conditions very well, it's not practical. Therefore, developing a partitioning algorithm to automatically calculate the approximated partitioning solution is more reasonable and has been widely explored. However, during the process of determining an appropriate partitioning, extra computing cost is involved. The performance of partitioning algorithm relies much on the dynamic estimation of communication energy consumption. To achieve better partitioning result, many efforts have been made to find effective methods of network condition prediction and estimation. The energy efficiency of fine-grained offloading would become better if program partitioning algorithms can get lower cost as well as higher accuracy.

IV. Coarse-grained Offloading

The fine-grained offloading scheme (in which only parts of the program are offloaded) is not always the optimal solution. Whether relying on the programmers to modify the application source code externally, or depending on the remote execution manager to calculate the approximated partitioning solution, both options lead to extra overhead that may consume additional CPU energy or burden on application programmers. Many coarse-grained energy efficient remote execution solutions ([15], [16], etc.) are presented to deal with these issues.

4

The viewpoint presented in [15] is different from those expressed in previous studies on computing offloading. This approach does not require estimating the computation time prior to the execution. Instead, the program is initially executed on the portable client with a timeout. If the computation is not completed after the timeout, it is offloaded to the server. The timeout is first set to be the minimum computation time that can benefit from offloading. This method, however, is proved to be 2-competitive. They further consider collecting online statistics of the computation time to find the statistically optimal timeout. The advantage of this method involves the instances with short computation time are executed at the client, and the instances with large computation time are offloaded to the server. It requires no burden on the application programmers and brings little calculation overhead. The limitation of this method involves the timeout wastes energy for the computation instances that are eventually offloaded, in which case the total application offloading may result in unnecessary transmission energy consumption.

Spectra [16] is a remote execution system designed for pervasive environments. It does not require applications to specify resource requirements for a variety of platforms and output qualities. Instead, applications need only to specify operations of interest and the input parameters for those operations, and then it deduces functions that make the input parameters relate to the resource usage to predict future application resource use. Different from previous works, Spectra introduces a concept called application fidelity, which 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. Reduced fidelity costs a smaller amount of energy compared with full fidelity. Spectra chooses the best fidelity level and execution mode and returns these values to the application. This approach indicates that sometimes executing heavy (full fidelity) PC-oriented applications on mobile devices is not necessary, and the methods to select the best execution fidelity present an extensible research question for future studies.

Apart from program offloading, another approach to remote execution involves providing operating system support for process or virtual machine migration. Since network latency exists and changes every time, how to synchronize the process between mobile terminals and remote servers becomes a difficult problem. ZAP [18] achieves the entire process migration by enabling OS support to handle checkpoint and restart issue. The work presented in [19] provides a feasible solution to make live OS migration a practical tool for servers running interactive workloads. In the meantime, several mobile cloud computing infrastructures emerged in recent five years (such as Clonecloud [17], Cloudlets [20], etc.) can better handle migration tasks in mobile device environments. moves an entire OS and all of its running applications to mobile cloud. Each of these approaches completely reduce the burden on application programmers. However, more challenges are brought to the development of MCC, for example, user privacy and security are even more difficult to guarantee while the entire application runtime is migrated to the cloud; unnecessary transmissions may be brought to the device to consume more battery energy. Therefore, we believe there are still tremendous opportunities for researchers to make ground-breaking contributions in this field.

V. Conclusion

The nature of power consumption comes from running applications. To better utilize the resource on the cloud, there is a trade-off between transmission energy cost and local execution energy cost. Though many works have made an effort to find the balance point, only a subset of those studies focuses on energy efficiency, while in many cases they merely focus on response time and other resource consumption. A large part of the researches uses modeling and simulation. Some of the models only offload parts of the program to the server, which reduce transmission cost, while at the same time bringing extra partition overhead. Other works try to transplant the entire OS or application runtime to the cloud, which ease the burden on external application programmers.

In this paper, we present a survey of energy-efficient computation offloading technologies within the environment of MCC. The work aims to convey to the research community the design principles of energy-efficient computation offloading in MCC and the challenging issues of this research direction.

Acknowledgment

This work was supported by the National Core-High-Base Major Project of China 2010ZX01045-001-005-4, NSFC Project 61120106008, 60911130511, partially supported by NSERC Canada Discovery grant. The work of the Ivan Stojmenovic was also supported by the Government of China for the Tsinghua 1000 Plan Distinguished Professor (2012-5) position.

References

[1] A. Shye, B. Scholbrock, and G. Memik, "Into the wild: studying real user activity patterns to guide power optimizations for mobile architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2009, pp. 168–178.

[2] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H. Yang, "The case for cyber foraging," in *Proceedings of the 10th workshop on ACM SIGOPS European workshop*. ACM, 2002, pp. 87–92.

[3] K. Kumar and Y. Lu, "Cloud computing for mobile users: can offloading computation save energy?" *Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[4] A. Rudenko, P. Reiher, G. Popek, and G. Kuenning, "Saving portable computer battery power through remote process execution," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 2, no. 1, pp. 19–26, 1998.

[5] D. Murray, E. Yoneki, J. Crowcroft, and S. Hand, "The case for crowd computing," in *Proceedings of the second ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds*. ACM, 2010, pp. 39–44.

[6] M. Demirbas, M. Bayir, C. Akcora, Y. Yilmaz, and H. Ferhatosmanoglu, "Crowd-sourced sensing and collaboration using twitter," in *World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010 IEEE International Symposium on a*. IEEE, 2010, pp. 1–9.

[7] C. Young, Y. Lakshman, T. Szymanski, J. Reppy, D. Presotto, R. Pike, G. Narlikar, S. Mullender, and E. Grosse, "Protium, an infrastructure for partitioned applications," in *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*. IEEE, 2001, pp. 47–52.

[8] K. Yang, S. Ou, and H. Chen, "On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications," *Communications Magazine, IEEE*, vol. 46, no. 1, pp. 56–63, 2008.

[9] R. Wolski, S. Gurun, C. Krintz, and D. Nurmi, "Using bandwidth data to make computation offloading decisions," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. Ieee, 2008, pp. 1–8.

[10] Z. Li, C. Wang, and R. Xu, "Computation offloading to save energy on handheld devices: a partition scheme," in *Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems*. ACM, 2001, pp. 238–246.

[11] S. Ou, K. Yang, A. Liotta, and L. Hu, "Performance analysis of offloading systems in mobile wireless environments," in *Communications, 2007. ICC'07. IEEE International Conference on*. IEEE, 2007, pp. 1821–1826.

[12] M. Tang and J. Cao, "A dynamic mechanism for handling mobile computing environmental changes," in *Proceedings of the 1st international conference on Scalable information systems*. ACM, 2006, p. 7.

[13] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 49–62.

[14] B. Chun and P. Maniatis, "Dynamically partitioning applications between weak devices and clouds," in *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*. ACM, 2010, p. 7.

[15] C. Xian, Y. Lu, and Z. Li, "Adaptive computation offloading for energy conservation on battery-powered systems," in *Parallel and Distributed Systems, 2007 International Conference on*, vol. 2. IEEE, 2007, pp. 1–8.

[16] J. Flinn, D. Narayanan, and M. Satyanarayanan, "Self-tuned remote execution for pervasive computing," in *Hot Topics in Operating Systems, 2001. Proceedings of the Eighth Workshop on*. IEEE, 2001, pp. 61–66.

[17] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*. ACM, 2011, pp. 301–314.

[18] S. Osman, D. Subhraveti, G. Su, and J. Nieh, "The design and implementation of zap: A system for migrating computing environments," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 361–376, 2002.

[19] C. Clark, K. Fraser, S. Hand, J. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, pp. 273–286.

[20] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *Pervasive Computing, IEEE*, vol. 8, no. 4, pp. 14–23, 2009.