Fundamental Challenges in Mobile Computing

M. Satyanarayanan
School of Computer Science
Carnegie Mellon University

Abstract

This paper is an answer to the question: "What is unique and conceptually different about mobile computing?" The paper begins by describing a set of constraints intrinsic to mobile computing, and examining the impact of these constraints on the design of distributed systems. Next, it summarizes the key results of the Coda and Odyssey systems. Finally, it describes the research opportunities in five important topics relevant to mobile computing: caching metrics, semantic callbacks and validators, resource revocation, analysis of adaptation, and global estimation from local observations.

1. Introduction

What is really different about mobile computing? The computers are smaller and bits travel by wireless rather than Ethernet. How can this possibly make any difference? Isn't a mobile system merely a special case of a distributed system? Are there any new and deep issues to be investigated, or is mobile computing just the latest fad?

This paper is my attempt to answer these questions. The paper is in three parts: a characterization of the essence of mobile computing; a brief summary of results obtained by my research group in the context of the Coda and Odyssey systems; and a guided tour of fertile research topics awaiting investigation. Think of this paper as a report from the front by an implementor of mobile information systems to more theoretically-inclined computers scientists.

1.1. Constraints of Mobility

Mobile computing is characterized by four constraints:

- *Mobile elements are resource-poor relative to static elements.*
  For a given cost and level of technology, considerations of weight, power, size and ergonomics will exact a penalty in computational resources such as processor speed, memory size, and disk capacity. While mobile elements will improve in absolute ability, they will always be resource-poor relative to static elements.

- *Mobility is inherently hazardous.*
  A Wall Street stockbroker is more likely to be mugged on the streets of Manhattan and have his laptop stolen than to have his workstation in a locked office be physically subverted. In addition to security concerns, portable computers are more vulnerable to loss or damage.

- *Mobile connectivity is highly variable in performance and reliability.*
  Some buildings may offer reliable, high-bandwidth wireless connectivity while others may only offer low-bandwidth connectivity. Outdoors, a mobile client may have to rely on a low-bandwidth wireless network with gaps in coverage.

- *Mobile elements rely on a finite energy source.*
  While battery technology will undoubtedly improve over time, the need to be sensitive to power consumption will not diminish. Concern for power consumption must span many levels of hardware and software to be fully effective.

These constraints are not artifacts of current technology, but are intrinsic to mobility. Together, they complicate the design of mobile information systems and require us to rethink traditional approaches to information access.

1.2. The Need for Adaptation

Mobility exacerbates the tension between autonomy and interdependence that is characteristic of all distributed systems. The relative resource poverty of mobile elements as well as their lower trust and robustness argues for reliance on static servers. But the need to cope with unreliable and low-performance networks, as well as the need to be sensitive to power consumption argues for self-reliance.

Any viable approach to mobile computing must strike a balance between these competing concerns. This balance cannot be a static one; as the circumstances of a mobile client change, it must react and dynamically reassign the responsibilities of client and server. In other words, mobile clients must be *adaptive.*

This research was supported by the Air Force Materiel Command (AFMC) and ARPA under contract number F196828-93-C-0193. Additional support was provided by the IBM Corp. and Intel Corp. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of AFMC, ARPA, IBM, Intel, CMU, or the U.S. Government.

1.3. Taxonomy of Adaptation Strategies

The range of strategies for adaptation is delimited by two extremes, as shown in Figure 1. At one extreme, adaptation is entirely the responsibility of individual applications. While this *laissez-faire* approach avoids the need for system support, it lacks a central arbitrator to resolve incompatible resource demands of different applications and to enforce limits on resource usage. It also makes applications more difficult to write, and fails to amortize the development cost of support for adaptation.

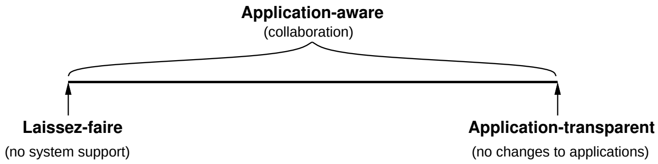

Figure 1: Range of Adaptation Strategies

The other extreme of *application-transparent adaptation* places entire responsibility for adaptation on the system. This approach is attractive because it is backward compatible with existing applications: they continue to work when mobile without any modifications. The system provides the focal point for resource arbitration and control. The drawback of this approach is that there may be situations where the adaptation performed by the system is inadequate or even counterproductive.

Between these two extremes lies a spectrum of possibilities that we collectively refer to as *application-aware adaptation.* By supporting a collaborative partnership between applications and the system, this approach permits applications to determine how best to adapt, but preserves the ability of the system to monitor resources and enforce allocation decisions.

1.4. The Extended Client-Server Model

Another way to characterize the impact of mobile computing constraints is to examine their effect on the classic client-server model. In this model, a small number of trusted server sites constitute the true home of data. Efficient and safe access to this data is possible from a much larger number of untrusted client sites. Techniques such as caching and read-ahead can be used to provide good performance, while end-to-end authentication and encrypted transmission can be used to preserve security.

This model has proved to be especially valuable for scalability [16]. In effect, the client-server model decomposes a large distributed system into a small nucleus that changes relatively slowly, and a much larger and less static periphery of clients. From the perspectives of security and system administration, the scale of the system appears to be that of the nucleus. But from the perspectives of performance and availability, a user at the periphery receives almost standalone service.

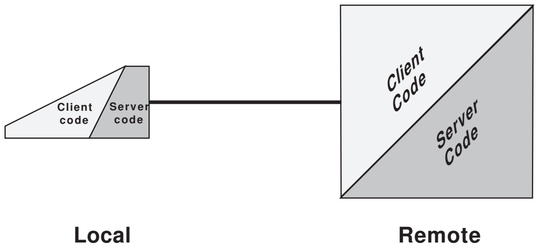

Figure 2: Temporary Blurring of Roles

Coping with the constraints of mobility requires us to rethink this model. The distinction between clients and servers may have to be temporarily blurred, resulting in the *extended client-server model* shown in Figure 2. The resource limitations of clients may require certain operations normally performed on clients to sometimes be performed on resource-rich servers. Conversely, the need to cope with uncertain connectivity requires clients to sometimes emulate the functions of a server. These are, of course, short-term deviations from the classic client-server model for purposes of performance and availability. From the longer-term perspective of system administration and security, the roles of servers and clients remain unchanged.

2. Summary of Coda and Odyssey Results

We have been exploring application-transparent adaptation since about 1990. Our research vehicle has been the *Coda File System*, a descendant of AFS [2]. Coda has been in active use for five years, and has proved to be a valuable testbed [13]. Coda clients are in regular use over a wide range of networks such as 10 Mb/s Ethernet, 2 Mb/s radio, and 9600 baud modems.

Since the research contributions of Coda have already been extensively documented in the literature, we only provide a high-level summary of the key results here:

Disconnected operation
: Coda has demonstrated that disconnected operation is feasible, effective, and usable in a distributed Unix file system [3, 4, 17]. The key mechanims for supporting disconnected operation include hoarding (user-assisted cache management), update logging with extensive optimizations while disconnected, and reintegration upon reconnection.

Optimistic replication
: Coda was one of the earliest systems to demonstrate that an optimistic replica control strategy can be used for serious and practical mobile computing [6]. It incorporates several novel mechanisms to render this approach viable. These include log-based directory resolution [5], application-specific file resolution [7], and mechanisms for conflict detection, containment and manual repair.

Support for weak connectivity
: Coda has shown that weak connectivity can be

exploited to alleviate the limitations of disconnected operation [12]. The mechanisms needed to accomplish this include adaptive transport protocols, a rapid cache validation mechanism, a trickle reintegration mechanism for propagating updates, and model-based cache miss handling for usability.

Isolation-only transactions

In the context of Coda, a new abstraction called isolation-only transaction has been developed to cope with the detection and handling of read-write conflicts during disconnected operation [9]. This abstraction selectively incorporates concepts from database transactions, while making minimal demands of resource-poor mobile clients and preserving upward compatibility with Unix applications.

Server replication

Coda has shown how server replication can be used to complement disconnected operation [15]. Although this is not particularly relevant to mobility, it is an important result in distributed systems because it clarifies the relationship between first-class (i.e., server) replicas and second-class replicas (i.e., client caches). It also represents one of the first demonstrations of optimistic replication applied to a distributed system with the client-server model.

More recently, we have begun exploration of application-aware adaptation in *Odyssey,* a platform for mobile computing. An preliminary prototype of Odyssey has been built [14, 18], and a more complete prototype is under development. The early evidence is promising, but it is far too early for definitive results.

3. Fertile Topics for Exploration

We now turn to the discussion of promising research topics in mobile computing. By its very nature, this section of the paper is highly speculative and will raise far more questions than it answers. Further, this is a selective list: it is certainly not intended to be complete. Rather, my goal is to give the reader a tantalizing glimpse of the rich problem space defined by mobile computing.

In choosing the five topics discussed below, I have followed two guidelines. First, these problems are more likely to be solved by rigor and analysis than by implementation and experience. Second, each of these problems is real, not contrived. Good solutions and insights to these problems will strongly impact the mobile computing systems of the future.

Each topic is presented in two parts: a brief discussion that lays out the problem space of the topic, followed by a sample of open questions pertaining to it. Again, my aim in posing these questions is not to be exhaustive but to offer food for thought.

3.1. Caching Metrics

Caching plays a key role in mobile computing because of its ability to alleviate the performance and availability limitations of weakly-connected and disconnected operation. But evaluating alternative caching strategies for mobile computing is problematic.

Today, the only metric of cache quality is the *miss ratio.* The underlying assumption of this metric is that all cache misses are equivalent (that is, all cache misses exact roughly the same penalty from the user). This assumption is valid when the cache and primary copies are strongly connected, because the performance penalty resulting from a cache miss is small and, to a first approximation, independent of file length. But the assumption is unlikely to be valid during disconnected or weakly-connected operation.

The miss ratio also fails to take into account the timing of misses. For example, a user may react differently to a cache miss occurring within the first few minutes of disconnection than to one occurring near the end of the disconnection. As another example, the periodic spin-down of disks to save power in mobile computers makes it cheaper to service a certain number of page faults if they are clustered together than if they are widely spaced.

To be useful, new caching metrics must satisfy two important criteria. First, they should be consistent with qualitative perceptions of performance and availability experienced by users in mobile computing. Second, they should be cheap and easy to monitor. The challenge is to develop such metrics and demonstrate their applicability to mobile computing. Initial work toward this end is being done by Ebling [1].

3.1.1. Some Open Questions

- What is an appropriate set of caching metrics for mobile computing?
- Under what circumstances does one use each metric?
- How does one efficiently monitor these metrics?
- What are the implications of these alternative metrics for caching algorithms?

3.2. Semantic Callbacks and Validators

Preserving cache coherence under conditions of weak connectivity can be expensive. Large communication latency increases the cost of validation of cached objects. Intermittent failures increase the frequency of validation, since it must be performed each time communication is restored. A lazy approach that only validates on demand could reduce validation frequency; but this approach would worsen consistency because it increases the likelihood of stale objects being accessed while disconnected. The cost of cache coherence is exacerbated in systems like Coda that use anticipatory caching for availability, because the number of objects cached (resident set size) is much larger than the number of objects in current use (working set size).

The Coda solution is to maintain cache coherence at multiple levels of granularity and to use callbacks [11]. Clients and servers maintain version information on individual objects as well as entire subtrees of them. Rapid cache validation is possible by comparing version stamps on the subtrees. Once established, validity can be maintained through callbacks. This approach to cache coherence trades precision of invalidation for speed of validation. It preserves correctness while dramatically reducing the cost of cache coherence under conditions of weak connectivity. Usage measurements from Coda confirm that these potential gains are indeed achievable in practice [12].

The idea of maintaining coherence at multiple granularities can be generalized to a variety of data types and applications in the following way:

- a client caches data satisfying some predicate *P* from a server.
- the server remembers a predicate *Q* that is much cheaper to compute, and possesses the property *Q* implies *P*. In other words, as long as *Q* is true, the cached data it corresponds to is guaranteed to be valid. But if *Q* is false, nothing can be inferred about that data.
- On each update, the server re-evaluates *Q*. If *Q* becomes false, the server notifies the client that its cached data might be stale.
- Prior to its next access, the client must contact the server and obtain fresh data satisfying *P*.

We refer to *Q* as a *semantic callback* for *P*, because the interpretation of *P* and *Q* depends on the specifics of the data and application. For example, *P* would be an SQL `select` statement if one is caching data from a relational database. Or it could be a piece of code that performs a pattern match for a particular individual's face from a database of images. *Q* must conform to *P*: a simpler `select` statement in the first case, and a piece of code that performs a much less accurate pattern match in the second case. In Coda, *P* corresponds to the version number of an object being equal to a specific value (*x*), while *Q* corresponds to the version number of the encapsulating volume being unchanged since the last time the version number of the object was confirmed to be *x*.

Semantic validation can be extended to domains beyond mobile computing. It will be especially valuable in geographically widespread distributed systems, where the timing difference between local and remote actions is too large to ignore even when communication occurs at the speed of light. The predicate *Q* in such cases serves as an inexpensive *validator* for cached data satisfying some complex criteria.

Consider the example of a transcontinental distributed system in the United States. Even at the speed of light, communication from one coast to the other takes about 16 milliseconds. A round trip RPC will take over 30 milliseconds. During this time, a client with a 100 MIP processor can execute over 3 million instructions! Since processor speed can be expected to increase over time, the lost computational opportunity represented by this scenario will only worsen.

Over time, the synchronous model implicit in the use of RPC will become increasingly untenable. Eventually, very wide-area distributed systems will have to be structured around an asynchronous model. At what scale and timeframe this shift will occur depends on two factors: the substantially simpler design, implementation, and debugging inherent in the synchronous model, and the considerably higher performance (and hence usability) of the asynchronous model.

One promising asynchronous model is obtained by combining the idea of cheap but conservative validation with the style of programming characterized by optimistic concurrency control [8]. The resulting approach bears some resemblance to the use of hints in distributed systems [19], and is best illustrated by an example.

Consider remote control of a robot explorer on the surface of Mars. Since light takes many minutes to travel from earth to Mars, and emergencies of various kinds may arise on Mars, the robot must be capable of reacting on its own. At the same time, the exploration is to be directed live by a human controller on earth — a classic command and control problem.

This example characterizes a distributed system in which communication latency is large enough that a synchronous design paradigm will not work. The knowledge of the robot's status will always be obsolete on earth. But, since emergencies are rare, this knowledge will usually differ from current reality in one of two benign ways. Either the differences are in attributes irrelevant to the task at hand, or the differences can be predicted with adequate accuracy by methods such as dead reckoning. Suppose the robot's state is *P*, as characterized in a transmission to earth. Based on some properties, *Q*, of this state, a command is issued to the robot. For this command to be meaningful when it reaches the robot, *Q* must still be true. This can be verified by transmitting *Q* along with the command, and having the robot validate *Q* upon receipt. For this approach to be feasible, both transmitting and evaluating *Q* must be cheap.

There are, of course, numerous detailed questions to be answered regarding this approach. But it does offer an intriguing way of combining correctness with performance in very wide-area distributed systems.

3.2.1. Some Open Questions

- Under what circumstances are semantic callbacks most useful? When are they not useful?
- What forms can P and Q take for data types and applications in common use? How does one estimate their relative costs in those cases?
- Can P and Q really be arbitrary code? Are there restrictions necessary for efficiency and practicality?
- How does one derive Q from P quickly? Are there restrictions on P that make this simpler?
- How does one trade off the relative cost and benefit of P and Q? Is the tradeoff space discrete or continuous? Can this tradeoff be made adaptive?

3.3. Algorithms for Resource Revocation

Application-aware adaptation complicates the problem of resource management. In principle, the system owns all resources. At any time, it may revoke resources that it has temporarily delegated to an applicaton. Alas, reality is never that simple. A variety of factors complicate the problem.

First, some applications are more important than others. Any acceptable revocation strategy must be sensitive to these differences. Second, the cost of revoking the same resource may be different to different applications. For example, reducing the bandwidth available to one application may result in its substantially increasing the amount of processing it does to compensate. A similar reduction in bandwidth for another application may result in a much smaller increase in processing. A good revocation strategy must take into account these differential impacts. Third, there may be dependencies between processes that should be taken into account during revocation. For example, two processes may have a producer-consumer relationship. Revoking resources from one process may cause the other to stall. More complex dependencies involving multiple processes are also possible. Unless revocation takes these dependencies into account, hazards such as deadlocks may occur.

Revocation of resources from applications is not common in current systems. Classical operating systems research has focused on resource allocation issues rather than resource revocation. As a result there is currently little codified knowledge about safe and efficient techniques for revocation. This deficiency will have to be remedied as application-aware adaptation becomes more widely used.

3.3.1. Some open questions

- How does one formulate the resource revocation problem?
- How does one characterize the differential impact of revocation on different applications?
- What strategies does one use if multiple resources must be simultaneously revoked?
- How does one distinguish between resources whose revocation is easy to recover from and those it is expensive or impossible to recover from?
- How does one handle deadlocks during revocation?

3.4. Analysis of Adaptation

How does one compare the adaptive capabilities of two mobile clients? The primary figure of merit is *agility*, or the ability of a client to promptly respond to perturbations. Since it is possible for a client to be more agile with respect to some variables (such as bandwidth) than others (such as battery power), agility should be viewed as a composite metric.

A system that is highly agile may suffer from *instability.* Such a system consumes almost all its resources reacting to minor perturbations, hence performing little useful computation. The ideal mobile client is obviously one that is highly agile but very stable with respect to all variables of interest.

Control theory is a domain that might have useful insights to offer in refining these ideas and quantifying them. Historically, control theory has focused on hardware systems. But there is no conceptual reason why it cannot be extended to software systems. Only careful investigation can tell, of course, whether the relevance is direct and useful or merely superficial.

3.4.1. Some open questions

- What are the right metrics of agility?
- Are there systematic techniques to improve the agility of a system?
- How does one decide when a mobile system is "agile enough"?
- What are the right metrics of system stability?
- Can one develop design guidelines to ensure stability?
- Can one analytically derive the agility and stability properties of an adaptive system without building it first?

3.5. Global Estimation from Local Observations

Adaptation requires a mobile client to sense changes in its environment, make inferences about the cause of these changes, and then react appropriately. These imply the ability to make global estimates based on local observations.

To detect changes, the client must rely on local observations. For example, it can measure quantities such as local signal strength, packet rate, average round-trip times, and dispersion in round-trip times. But interpreting these observations is nontrivial. A change in a given quantity can be due to a multiplicity of non-local phenomena. For example, packet rate will drop due to an overload on a distant server. But it will also drop when there is congestion on an intermediate network segment. If an incorrect cause is inferred from an observation, the adaptation performed by the client may be ineffective or counterproductive.

At present there is no systematic theory to guide global estimation from local observations. The problem is especially challenging in the absence of out-of-band communication, because the client cannot use an alternative channel to help narrow the diagnosis on the main communication channel.

3.5.1. Some Open Questions

- Are there systematic ways to do global estimation from local estimates?
- Can one bound the error in global estimates?
- What is the relationship of global estimation to agility of adaptation? Can one quantify this relationship?
- Can one provide system support to improve global estimation? For example, do closely-synchronized, low-drift clocks on clients and servers help?
- Can one quantify the benefits of out-of-band channels? For example, how much does the presence of a low-latency, low-bandwidth channel help with estimates on a parallel high-latency, high-bandwidth channel?

4. Conclusion

The tension between autonomy and interdependence is intrinsic to all distibuted systems. Mobility exacerbates this tension, making it necessary for mobile clients to tolerate a far broader range of external conditions than has been necessary hitherto.

Adaptation is the key to mobility. By using local resources to reduce communication and to cope with uncertainty, adaptation insulates users from the vagaries of mobile environments. Our research is exploring two different approaches to adaptation: application-transparent and application-aware. Our experience with Coda confirms that application-transparent adaptation is indeed viable and effective for a broad range of important applications. In circumstances where it is inadequate, our initial experience with Odyssey suggests that application-aware adaptation is the appropriate strategy.

In closing, it is worth speculating on the long-term impact of mobility on distributed systems. In his book *Mind Children*, my colleague Hans Moravec draws an analogy between the seminal role of mobility in the evolution of biological species, and its influence on the capabilities of computing systems [10]. Although Hans' comments are directed at robotic systems, I believe that his observation applies equally well to a much broader class of distributed computing systems involving mobile elements. Mobility will influence the evolution of distributed systems in ways that we can only dimly perceive at the present time. In this sense, mobile computing is truly a seminal influence on the design of distributed systems.

Acknowledgements

This paper is the result of many years of interaction and brainstorming with members of the Coda and Odyssey projects. These members include Jay Kistler, Puneet Kumar, David Steere, Lily Mummert, Maria Ebling, Hank Mashburn, Brian Noble, Masashi Kudo, Josh Raiff, Qi Lu, Morgan Price, Hiroshi Inamura, Tetsuro Muranaga, Bob Baron, Dushyanth Narayanan, and Eric Tilton.

I wish to thank Vassos Hadzilacos and Yoram Moses, the program chairmen of the Fourteenth and Fifteenth ACM Symposia on Principles of Distributed Computing respectively, for giving me the opportunity to present these thoughts as an invited speaker at PODC '95 and as an invited paper in the proceedings of PODC '96. I also wish to thank Matt Zekauskas and Brian Noble for their help in reviewing and improving the presentation of the paper.

References

[1] Ebling, M.R.
Evaluating and Improving the Effectiveness of Caching for Availability.
PhD thesis, Department of Computer Science, Carnegie Mellon University, 1997 (in preparation).

[2] Howard, J.H., Kazar, M.L., Menees, S.G., Nichols, D.A., Satyanarayanan, M., Sidebotham, R.N., West, M.J.
Scale and Performance in a Distributed File System.
ACM Transactions on Computer Systems 6(1), February, 1988.

[3] Kistler, J.J., Satyanarayanan, M.
Disconnected Operation in the Coda File System.
ACM Transactions on Computer Systems 10(1), February, 1992.

[4] Kistler, J.J.
Disconnected Operation in a Distributed File System.
PhD thesis, Department of Computer Science, Carnegie Mellon University, May, 1993.

[5] Kumar, P., Satyanarayanan, M.
Log-Based Directory Resolution in the Coda File System.
In *Proceedings of the Second International Conference on Parallel and Distributed Information Systems*. San Diego, CA, January, 1993.

[6] Kumar, P.
Mitigating the Effects of Optimistic Replication in a Distributed File System.
PhD thesis, School of Computer Science, Carnegie Mellon University, December, 1994.

[7] Kumar, P., Satyanarayanan, M.
Flexible and Safe Resolution of File Conflicts.
In *Procedings of the 1995 USENIX Technical Conference*. New Orleans, LA, January, 1995.

[8] Kung, H.T., Robinson, J.
On Optimistic Methods for Concurrency Control.
ACM Transaction on Database Systems 6(2), June, 1981.

[9] Lu, Q., Satyanarayanan, M.
Improving Data Consistency in Mobile Computing Using Isolation-Only Transactions.
In *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems*. Orcas Island, WA, May, 1995.

[10] Moravec, H.
Mind Children.
Harvard University Press, Cambridge, MA, 1988.

[11] Mummert, L.B., Satyanarayanan, M.
Large Granularity Cache Coherence for Intermittent Connectivity.
In *Proceedings of the 1994 Summer USENIX Conference*. Boston, MA, June, 1994.

[12] Mummert, L.B., Ebling, M.R., Satyanarayanan, M.
Exploiting Weak Connectivity for Mobile File Access.
In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*. Copper Mountain Resort, CO, December, 1995.

[13] Noble, B., Satyanarayanan, M.
An Empirical Study of a Highly-Available File System.
In *Proceedings of the 1994 ACM Sigmetrics Conference*. Nashville, TN, May, 1994.

[14] Noble, B., Price, M., Satyanarayanan, M.
A Programming Interface for Application-Aware Adaptation in Mobile Computing.
Computing Systems 8, Fall, 1995.

[15] Satyanarayanan, M., Kistler, J.J., Kumar, P., Okasaki, M.E., Siegel, E.H., Steere, D.C.
Coda: A Highly Available File System for a Distributed Workstation Environment.
IEEE Transactions on Computers 39(4), April, 1990.

[16] Satyanarayanan, M.
The Influence of Scale on Distributed File System Design.
IEEE Transactions on Software Engineering 18(1), January, 1992.

[17] Satyanarayanan, M., Kistler, J.J., Mummert, L.B., Ebling, M.R., Kumar, P., Lu, Q.
Experience with Disconnected Operation in a Mobile Computing Environment.
In *Proceedings of the 1993 USENIX Symposium on Mobile and Location-Independent Computing*. Cambridge, MA, August, 1993.

[18] Steere, D.C., Satyanarayanan, M.
Using Dynamic Sets to Overcome High I/O Latencies During Search.
In *Proceedings of the Fifth Workshop on Hot Topics in Operating Systems*. Orcas Island, WA, May, 1995.

[19] Terry, D.B.
Caching Hints in Distributed Systems.
IEEE Transactions in Software Engineering SE-13(1), January, 1987.