Scaling Down Distributed Infrastructure on Wimpy Machines for Personal Web Archiving

Jimmy Lin
The iSchool — College of Information Studies
University of Maryland, College Park
jimmylin@umd.edu

ABSTRACT

Warcbase is an open-source platform for storing, managing, and analyzing web archives using modern "big data" infrastructure on commodity clusters—specifically, HBase for storage and Hadoop for data analytics. This paper describes an effort to scale "down" Warcbase onto a Raspberry Pi, an inexpensive single-board computer about the size of a deck of playing cards. Apart from an interesting technology demonstration, such a design presents new opportunities for personal web archiving, in enabling a low-cost, low-power, portable device that is able to continuously capture a user's web browsing history—not only the URLs of the pages that a user has visited, but the contents of those pages—and allowing the user to revisit any previously-encountered page, as it appeared at that time. Experiments show that data ingestion throughput and temporal browsing latency are adequate with existing hardware, which means that such capabilities are already feasible today.

Categories and Subject Descriptors: H.3.4 [Information Storage and Retrieval]: Systems and Software—Distributed systems

Keywords: HBase; Hadoop; Raspberry Pi

1. INTRODUCTION

Most web archiving efforts to date have focused on issues related to scalability, for example, the infrastructure to support storage, access, and processing of large collections [16, 9, 5, 8, 15, 13]. In the case of the Internet Archive, the scale is on the order of hundreds of billions of pages and tens of petabytes. These technical efforts invariably involve distributing the computational and storage load across clusters of servers. This paper explores a different angle: instead of scaling "up" on more powerful individual servers or scaling "out" onto clusters of commodity servers, I describe an effort to scale a web archiving platform "down" onto "wimpy" machines [3]. Such devices are characterized by low single-core performance with limited memory and storage, but they consume very little power and are inexpensive. Wimpy machines stand in contrast to traditional "brawny" servers that reside in datacenters and pack ever more cores, memory, and storage into power-hungry enclosures. The Raspberry Pi, an inexpensive single-board computer about the size of a deck of playing cards, exemplifies a wimpy machine.

In this paper, I take Warcbase, an existing open-source web archiving platform designed to scale out on commodity clusters, and scale down the infrastructure onto a Raspberry Pi. The performance of this design is evaluated in terms of data ingestion throughput and temporal browsing latency. An immediate and obvious question is, of course, why? I provide two answers:

First, scaling down distributed infrastructure is an interesting technology demonstration [14]. The ARM-based processor inside a Raspberry Pi is similar to processors inside mobile phones, tablets, embedded devices, etc. and overall, such devices are far more prevalent than traditional servers and PCs. For example, the technology research firm Gartner forecasts that worldwide shipments of PCs in 2015 will total around 320 million units, compared to 2.3 billion mobile phones and tablets.[1] In the emerging "internet of things", wimpy devices (everything from smart thermostats and coffee makers to dashboard devices in the car) are becoming increasingly ubiquitous. Thus, it is worthwhile to explore how infrastructure designed for brawny servers might run in such hardware environments.

Second, a web archiving platform running on a Raspberry Pi enables potential applications in *personal* web archiving, which is related to the trend of lifelogging [10]. One can imagine building a cheap, lightweight device with a long battery life that continuously captures an individual's web browsing history—not only the URLs of the pages that an individual has visited, but the *contents* of those pages; this can be viewed as a logical extension of browser bookmarks. Furthermore, such a device would allow the user to view a facsimile of any page that he or she had previously encountered. Experimental results in this paper show that such a device is already feasible with current hardware and software—it is merely an issue of "packaging" and developing the most natural user experience.

The contribution of this paper is a demonstration that Warcbase, a web archiving platform designed to run on a cluster of commodity servers, can be successfully scaled down to run on a wimpy machine—specifically, a Raspberry Pi. Evaluations show that data ingestion throughput is sufficient to support content capture at human browsing speeds

WWW 2015 Companion, May 18–22, 2015, Florence, Italy.
ACM 978-1-4503-3473-0/15/05.
http://dx.doi.org/10.1145/2740908.2741695.

[1] `http://www.gartner.com/newsroom/id/2791017`

and that temporal browsing latency is no worse than browsing the web in general. I discuss the implications of these results for personal web archiving and how such capabilities might be deployed.

2. WARCBASE

Warcbase [13] aims to build an open-source platform for storing, managing, and analyzing web archives using modern "big data" infrastructure—specifically, HBase for storage and Hadoop for data analytics.[2] This section provides an overview of the HBase storage architecture and the Warcbase data model. Analytics capabilities are not discussed here since it does not make sense at this time to perform data analytics (e.g., text processing, graph mining, etc.) on a Raspberry Pi.

2.1 Storage Management

HBase is an open-source platform for managing large semi-structured datasets. It is best described as a sparse, persistent, multiple-dimensional sorted map, originally derived from Google's Bigtable [7]. An HBase table maintains a mapping from a 4-tuple to arbitrary values as follows:

(row key, column family, column qualifier,
timestamp) $\rightarrow$ value

Conceptually, values are identified by rows and columns. A row is identified by its row key. Each column is decomposed into "family" and "qualifier", where the family provides a mechanism for the developer to specify groupings of qualifiers that should be physically co-located. The timestamp allows storage of multi-versioned values. Rows are lexicographically sorted, and thus an important element in HBase schema design is to leverage this property for the application's benefit. HBase provides a number of basic operations on the client end, including gets, puts, and range scans, along with co-processors that allow processing to be pushed over to the server side. The consistency model guarantees single row transactions, but nothing more.

A table in HBase is divided into regions, which are ranges of rows. Each region is assigned to a RegionServer, which is responsible for handling operations on rows in the assigned regions. RegionServers coordinate region assignment via ZooKeeper [11], a highly-available replicated state machine and coordination service. HBase data are physically stored in HFiles on the Hadoop Distributed File System (HDFS). HDFS achieves durability and availability through replication, and HBase benefits from this design transparently. The entire stack was designed to be fault-tolerant and scalable, and in production has been demonstrated to scale to petabytes [2] across thousands of machines.

Since one of Bigtable's original use case was storing web crawls, schema design for Warcbase is relatively straightforward. In Warcbase, each collection is kept in a separate HBase table. Each resource (e.g., web page, image, PDF, etc.) is uniquely identified by its URL and a timestamp (the crawl date). The resource's domain-reversed URL serves as its row key, i.e., `www.umd.edu` becomes `edu.umd.www`. This design allows different pages from the same domain to be physically kept together, thus allowing client requests to benefit from reference locality. The raw content of a resource (HTML, PDF, image, etc.) is stored as the value in the column family "c" with the MIME type (e.g., "text/html") as the qualifier. Each version of the crawl has a different timestamp. The use of the MIME type as the column qualifier eliminates the need for a separate query when fetching content to support temporal browsing (since the response HTTP header requires a Content-Type field), compared to an alternative design in which the MIME type is stored separately. The tradeoff, however, is that the qualifier is unknown *a priori* when issuing a "get" to fetch a specific resource, thus requiring a (short) scan of the qualifiers. This, however, does not have a material impact on performance since in many cases the exact timestamp of a resource is also not known in advance, so a scan through different versions is needed to find the most appropriate one.

[2]`http://warcbase.org`

Web crawling is beyond the scope of the project, which assumes that content has already been captured in existing ARC or WARC containers. Warcbase includes an ingestion program that populates the appropriate HBase table according to the above data model. Once a web crawl has been ingested, HBase handles the complexities of storage management in a scalable, fault-tolerant, and distributed manner. This includes maintaining the mapping from rows to RegionServers responsible for serving the rows, splitting regions as more data are ingested, rebalancing region-to-RegionServer assignments, etc. When a RegionServer fails, HBase transparently handles failover to ensure availability. Underneath HBase, HDFS handles replication and other aspects of managing the raw data blocks. Because of this design, user-facing services can be implemented as (often, stateless) clients that encapsulate the application logic.

2.2 Temporal Browsing

The separation of storage management from other functionalities is a key aspect of Warcbase's design. One illustration of this architecture is the implementation of capabilities that allow users to access historical versions of captured web pages and to navigate links to contemporaneous pages—commonly known as temporal browsing. Warcbase includes adaptors that let OpenWayback,[3] a popular open-source software package that provides temporal browsing, to use Warcbase as the storage backend. This is possible via two abstractions that the OpenWayback provides [16]:

- **Resource Store**, responsible for retrieving archived content, while abstracting the storage medium and format.
- **Resource Index**, responsible for fielding queries against content in the resource store.

In a standard OpenWayback installation, the resource store provides access to ARC/WARC data stored on the file system (either local disk or network-attached storage). Alternatively, a resource store can access data via a remote service. Similarly, OpenWayback provides alternative implementations of the resource index: a local BerkeleyDB index, a local CDX index, or a remote resource index. Small-scale installations typically use the BerkeleyDB index.

Warcbase/OpenWayback integration is accomplished by providing custom implementations of the resource store and the resource index. These implementations connect to a REST API provided by Warcbase, which allows an OpenWayback instance to query for resources (e.g., lookup a particular version of a URL) and to fetch relevant content.

[3]`https://github.com/iipc/openwayback`

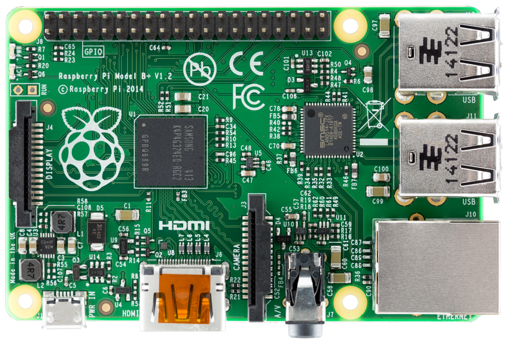

Figure 1: Raspberry Pi B+ Model (85.60mm × 56.5mm); peripherals connect via USB ports (right) and HDMI (bottom). (Image credit: Wikipedia)

In this way, investments that have already been made in the development of the OpenWayback system can be exploited for temporal browsing. Warcbase integration provides a seamless browsing experience that is identical to any other OpenWayback installation. However, the advantage is that, via Warcbase, storage management is offloaded to HBase. Collections can grow arbitrarily large in size without worrying about the scalability limitations of BerkeleyDB or CDX-based indexing.

As an alternative to the OpenWayback system, Warcbase can integrate with the Python WayBack system[4] in much the same way using the same REST API. The ability to switch different "frontends" highlights the flexibility of the loosely-coupled Warcbase design.

3. WARCBASE ON A RASPBERRY PI

The Raspberry Pi is an inexpensive, single-board computer about the size of a deck of playing cards. It was originally developed to promote computer science education, but has gained immense popularity in the "maker" community due to its ability to interact with the physical world through numerous expansion ports. The low cost of the machine has led to a variety of interesting applications in everything from controlling robots to retro arcade games.

The machine is based on the Broadcom BCM2835 "system on a chip" (SoC), which runs a 32-bit 700 MHz processor from the ARM11 family. The B+ model used in these experiments (Figure 1) has 512 MB RAM, and its performance is on par with a Pentium II processor from the late 1990s. The Raspberry Pi connects to an external display via an HDMI port and additional peripherals via four internal USB ports. Storage is provided via a microSD card.

Experiments described below used the following configuration: a Raspberry Pi B+ model overclocked to "turbo" at 1 GHz, a 64 GB microSD card, and an external USB dongle for wifi. The system runs the Raspbian flavor of Linux. The cost of the setup is as follows (as of Dec. 2014, in USD): the Raspberry Pi itself costs \$35 and the 64 GB microSD card costs \$25. The AC power supply costs \$9, or alternatively, one can purchase battery packs of varying capacities and prices. Note that these costs do not include the display and other peripherals.

[4]`https://github.com/ikreymer/pywb`

Oracle has released a version of the Java Virtual Machine (JVM) for the Raspberry Pi, so running HBase (implemented in Java) is reasonably straightforward. Since HBase is typically installed on a cluster, it relies on the Hadoop Distributed File System and a number of daemons (e.g., RegionServers) that run on the individual servers in the cluster—this configuration is typically referred to as fully-distributed mode. In pseudo-distributed mode, all the daemons run on a single machine, but still as separate processes (in different JVMs). In standalone mode, HBase bypasses HDFS to use the local filesystem directly and runs all daemons in the same JVM. Due to processor and memory limitations, the Raspberry Pi is unable to support the overhead of pseudo-distributed mode, but the standalone mode of HBase runs without any issues "out of the box".

3.1 Warcbase Data Ingestion

The first experiment evaluated the performance of ingesting web archive data into Warcbase. For this, I used sample data from Columbia University's Human Rights Web Archive[5] consisting of ARC data from web crawls performed in 2008. The sample collection contains 425 gzipped files totaling 43.4 GB.

The experimental procedure was as follows: I copied the ARC data onto the Raspberry Pi (via scp) in batches from a laptop. In this case, the Raspberry Pi was connected to the network via the external USB wifi dongle. Since the Raspberry Pi did not have sufficient space to hold all the ARC data and the HBase table concurrently, data ingestion proceeded in four separate batches: I copied a batch of ARC files over to the Raspberry Pi, ran the Warcbase ingestion program, and then removed the ARC files—repeating this procedure for all four batches. During ingestion, all records larger than 1 MB were discarded.

In total across the four batches, Warcbase ingested 1.68 million records in approximately 97000 seconds (about 27 hours), which translates into a sustained throughput of 17.3 records per second. During this experiment, the Raspberry Pi was connected to a wall outlet and power usage was measured using the consumer-grade "Kill A Watt" electricity usage monitor. The Raspberry Pi consumed around 2.6 Watts in steady state, with occasional peaks to around a maximum of 3.1 Watts. As a reference point, the device draws 2.0 Watts while idle.

3.2 Temporal Browsing Latency

The next experiment evaluated the temporal browsing performance of archived content stored on the Raspberry Pi. For this I used the Python WayBack system as the frontend to Warcbase, as it has the advantage of being less heavyweight than OpenWayback, which requires a Tomcat server. The Python WayBack provides a standalone web server that supports temporal browsing by connecting to the Warcbase REST API to fetch data stored in HBase. In this experiment, the Raspberry Pi was connected to the network via the USB wifi dongle and archived pages were accessed from a laptop using the Firefox browser.

To evaluate the end-to-end page load latencies associated with temporal browsing, I used the browser automation tool Selenium[6] to programmatically drive the Firefox browser. A total of 100 random HTML pages were sampled from

[5]`http://hrwa.cul.columbia.edu/`
[6]`http://www.seleniumhq.org/`

the archive and accessed in rapid succession, one after the other, using a simple driver program running on a laptop that connected to the Python WayBack instance running on the Raspberry Pi (over wifi). Once a page loaded, the driver program immediately proceeded to the next URL in the list. A page load was considered complete when the DOM property `Document.readyState` changed to "complete". Each page load completed in an average of 2.1 seconds (across three trials). The Raspberry Pi consumed around 2.4 Watts during this experiment.

Since Columbia University's Human Rights Web Archive was crawled using Internet Archive's Archive-It service, the content is also available in the Internet Archive's general web collection. As a reference, loading the same pages from the Internet Archive's Wayback machine averaged 2.9 seconds per page using the same Selenium-based driver program from the laptop (averaged over three trials). Using the same methodology, loading the Google homepage took 1.5 seconds on average and loading the Yahoo! homepage took 2.1 seconds on average, both across three trials.

These experimental results suggest that the experience of browsing archived pages on the Raspberry Pi using a combination of the Python WayBack system and Warcbase is no worse than the browsing experience of the web in general. Hands-on experience confirms these evaluation results.

4. IMPLICATIONS

Previous experiments show that Warcbase (with Python WayBack) can be scaled down onto a Raspberry Pi. This demonstration shows that it is possible to run the same web archiving software infrastructure on vastly different hardware configurations: from distributed clusters (the original target environment of Warcbase) to wimpy machines that are increasingly ubiquitous today. This is a nice result from a software engineering perspective because it means that investments in Warcbase can yield potential payoffs in a broad range of different applications and hardware environments. As Warcbase gains more features and wimpy machines grow more powerful, additional capabilities will become available in portable, low-power form factors "for free".

Beyond a technology demonstration, the ability to scale down Warcbase onto a Raspberry Pi opens up new opportunities for applications in personal web archiving. I explore some of these implications in detail below:

First, what is the scope of the personal web archiving problem? How much storage would be required to capture the contents of all the web pages that a user has visited or will visit? It is possible to develop some rough estimates based on existing large-scale web crawls: the December 2014 crawl by the Common Crawl Foundation[7] is 160 TB (compressed) and contains approximately 2 billion web pages, which is around 80 KB per page.[8] The largest microSD card for a Raspberry Pi available today is 128 GB; for simplicity, assume 100 GB can be devoted to storing data (and the remaining to the OS, swap, and other software). This would translate into 1.25 million web pages: at a rate of one page per second, such a setup would last 2.4 years of continuous web browsing, which is likely longer than the life of the Raspberry Pi itself—at which point, the user would probably just get a new device (with more storage). Although the sizes of web pages are growing, Moore's Law is increasing the capacity of solid-state storage at an exponential rate, so the replacement device will likely have a multiple of the capacity of the old device. Thus, the user can simply copy over all the old content and still have plenty of room to capture future activity. This upgrade-and-copy cycle could repeat indefinitely. Based on these assumptions, a (periodically-replaced) small, cheap, low-power wimpy device is likely able to hold the contents of all web pages a user will *ever visit*.[9]

Second, will a wimpy personal web archiving device be fast enough, both in terms of content ingestion and support for temporal browsing? The experiments in the previous section suggest so—the ingestion rate should be able to keep up with normal human browsing and the page load latency for displaying archived content does not seem markedly different from web browsing in general. Performance is already adequate with present day hardware, and will continue to improve over time.

A Raspberry Pi appears to have sufficient capability to serve as a personal web archiving device. How might such a device work in practice and what would the user experience be like? At a high level, such capabilities represent a logical extension of browser bookmarks [1, 12, 6]. Although the Raspberry Pi itself would fit into a user's pocket, current external battery packs are relatively bulky. Nevertheless, the entire package could easily fit into a backpack. The low power draw of the Raspberry Pi means that the user would be able to operate for relatively long periods without recharging. The user would carry the device everywhere, and it would continuously ingest web content that the user encounters. This could technically be accomplished either via proxying the user's browser sessions or via a lightweight browser plug-in that forwards data to the device: communication could occur over wifi or bluetooth. The user would be able to configure the device to blacklist certain sites (for example, don't capture the content of bank accounts), but otherwise, once configured, the personal web archiving device would just ingest data unobtrusively in the background. By default, only pages that the user has encountered would be captured, but more proactive approaches are possible—for example, automatically crawling "out" a small distance from the visited pages to gain broader coverage.

The temporal browsing experience would be similarly natural. Retrieving a previously-encountered page is as simple as pointing a browser from any device at the Raspberry Pi (and properly authenticating). A setup like the one described in the previous section would provide temporal browsing capabilities. Once again, connectivity can be provided either via wifi or bluetooth. The latter option makes it possible to browse archived content in the absence of internet connectivity.

This setup raises a number of potential objections and alternatives. I discuss two immediately obvious ones:

Why not offer personal web archiving as a cloud-based service? The simple answer is privacy. Users' web browsing

[7]`http://commoncrawl.org/`

[8]An analysis of a similar crawl from 2012 shows that 92% of the records are HTML pages, so this figure does not include images and other media for the most part.

[9]Of course, this calculation does not include media associated with the pages (images, videos, etc.). But even factoring in these data, the point remains that present technology is already adequate for capturing the web browsing history of typical users over moderately-long timeframes.

behavior potentially captures intimate aspects of their daily lives that they may not be comfortable sharing with a third party. Although one might argue that the logging of search queries and web beacons already represent significant intrusions into a user's privacy, the capture of actual content seems to raise even more concerns. In contrast, content ingested into a personal web archiving device is always under the control of the user and never communicated to third parties without user authorization.

Why not offer personal web archiving on non-wimpy devices? Certainly, the same capabilities described in this paper can be deployed on a sever connected to the network, or even directly on the user's laptop or desktop. What's compelling about using a Raspberry Pi? I believe that the answer is portability. Users today have multiple devices through which they access web content (e.g., laptop, tablet, mobile phone), often while away from the office or home. A personal web archiving device that the user carries everywhere can ingest content from and provide data access to multiple devices. This is similar to the "Personal Server" concept proposed over a decade ago [18].

Thus far, the focus has been on temporal browsing capabilities, which is limiting because users must already know the URL of the content they would like to access. Although it would be possible to build tools that allow users to navigate their histories by time and other metadata, full-text search capabilities would nevertheless be desirable. Previous studies have shown that a significant fraction of users' search behavior on the web consists of "refinding" [17], or searching for pages they had encountered before, which seems like a compelling use case for personal web archiving. Researchers have already explored running full-text search engines on mobile phones [4], so it should be possible to support full-text archive search on a Raspberry Pi.

5. CONCLUSION

The web archiving community has mostly been thinking "big", in conceiving of web archiving as primarily the activity of organizations—for example, national libraries and other cultural heritage institutions preserving the ephemeral web. This paper advocates thinking "small" as a complementary approach, where web archiving occurs at the personal level, as the byproduct of normal web usage. Ultimately, there is nothing to prevent individuals from contributing their personal web archives to a central repository on a periodic basis. The central repository would aggregate and anonymize the contributions, thus addressing potential privacy concerns, as well as perform de-duplication and other forms of data cleaning. One can think of this as crowdsourcing "collection development" to the masses, and this approach has the advantage in focusing preservation efforts on parts of the web that are actually useful, at least to someone.

6. ACKNOWLEDGMENTS

This work has been supported by the Andrew W. Mellon Foundation via Columbia University's Web Archiving Incentive Program and NSF award IIS-1218043. Any opinions, findings, conclusions, or recommendations expressed are mine and do not necessarily reflect those of the sponsors. I'd like to thank Columbia University Libraries for sharing web archive data and helpful comments from Ian Milligan and Ilya Kreymer.

7. REFERENCES

[1] D. Abrams, R. Baecker, and M. Chignell. Information archiving with bookmarks: Personal web space construction and organization. *CHI*, 1998.

[2] A. Aiyer, M. Bautin, G. Chen, P. Khemani, K. Muthukkaruppan, K. Spiegelberg, L. Tang, and M. Vaidya. Storage infrastructure behind Facebook Messages: Using HBase at scale. *IEEE Data Engineering Bulletin*, 35(2):4–13, 2012.

[3] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. FAWN: A Fast Array of Wimpy Nodes. *SOSP*, 2009.

[4] A. Balasubramanian, N. Balasubramanian, S. J. Huston, D. Metzler, and D. J. Wetherall. FindAll: A local search engine for mobile phones. *CoNEXT*, 2012.

[5] S. Barton. Mignify: A big data refinery built on HBase. *HBaseCon*, 2012.

[6] R. Boardman and M. A. Sasse. "Stuff goes into the computer and doesn't come out": A cross-tool study of personal information management. *CHI*, 2004.

[7] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber. Bigtable: A distributed storage system for structured data. *OSDI*, 2006.

[8] D. Gomes, M. Costa, D. Cruz, J. Miranda, and S. Fontes. Creating a billion-scale searchable web archive. *WWW Companion*, 2013.

[9] D. Gomes, J. Miranda, and M. Costa. A survey on web archiving initiatives. *TPDL*, 2011.

[10] C. Gurrin, A. F. Smeaton, and A. R. Doherty. Lifelogging: Personal big data. *Foundation and Trends in Information Retrieval*, 8(1):1–125, 2014.

[11] P. Hunt, M. Konar, F. Junqueira, and B. Reed. ZooKeeper: Wait-free coordination for Internet-scale systems. *USENIX*, 2010.

[12] W.-S. Li, Q. Vu, D. Agrawal, Y. Hara, and H. Takano. PowerBookmarks: A system for personalizable web information organization, sharing, and management. *Computer Networks*, 31(11–16):1375–1389, 1999.

[13] J. Lin, M. Gholami, and J. Rao. Infrastructure for supporting exploration and discovery in web archives. *WWW Companion*, 2014.

[14] T. Mühlbauer, W. Rödiger, R. Seilbeck, A. Reiser, A. Kemper, and T. Neumann. One DBMS for all: The brawny few and the wimpy crowd. *SIGMOD*, 2014.

[15] C. Neudecker and S. Schlarb. The elephant in the library: Integrating Hadoop. *Hadoop Summit Europe*, 2013.

[16] B. Tofel. 'Wayback' for accessing web archives. *International Web Archiving Workshop*, 2007.

[17] S. K. Tyler and J. Teevan. Large scale query log analysis of re-finding. *WSDM*, 2010.

[18] R. Want, T. Pering, G. Danneels, M. Kumar, M. Sundar, and J. Light. The Personal Server: Changing the way we think about ubiquitous computing. *UbiComp*, 2002.