COCO: A Web-Based Data Tracking Architecture for Challenged Network Environments

Saureen Shah
Digital Green
New Delhi, India
saureen@digitalgreen.org

Apurva Joshi
Digital Green
New Delhi, India
apurva@digitalgreen.org

ABSTRACT

Data tracking and analytics methods for organizations operating in rural locations are cumbersome – particularly for the many that continue to use traditional, non-electronic methods. To evaluate effectiveness and productivity, these organizations are being asked to adopt data and analytics methodologies that are technology oriented in nature. Today, network connectivity is increasing even in the most remote locations and electronic data input is gradually becoming less of an impediment to data collection. To be sure, uninterrupted bandwidth availability still eludes these locations, making it difficult for traditional web applications to function. In cognizance of intermittent to zero bandwidth availability in rural locations, we propose COCO (connect-online, connect-offline), a data input framework designed to seamlessly enable online as well as offline connection capabilities for applications. By building on the COCO framework, applications can continue operating by going offline. Preliminary results show that applications built on the COCO framework achieve low latency and high execution speeds by virtue of the framework's browser-client architecture. We also demonstrate the effectiveness of a COCO framework-enabled application over a traditional web application when run in a lab, and in rural locations.

General Terms

Design, Performance, Management, Reliability, Standardization.

Keywords

ICT4D, Content Management System, Model View Controller (MVC), NGO, Rural Development, Offline, Low bandwidth, Intermittent network.

1. INTRODUCTION

Tracking performance and productivity is integral to an organization in managing its various business functions, and reporting to its stakeholders. Over the last few decades, methods to track have evolved from rudimentary, non-electronic bookkeeping to sophisticated business intelligence and analytics systems built by enterprise software makers [1, 2]. While these systems are sophisticated, almost all come at an exorbitant price, and are often laden with features meant for large-enterprise customers. Small to mid-sized organizations typically do not require such complex systems, but still desire comparable tracking and analytics capabilities. Many among these organizations are non-profit, non-governmental organizations (NGOs).

Learning from the corporate space, NGOs and their donors are increasingly recognizing the importance of tracking and analytics capabilities. For NGOs with operations in rural locations, adopting new technologies to enable electronic data collection can be slow, and from a logistics and infrastructure perspective, laborious [3]. By and large, these organizations still rely on paper forms to input and collect information, typically done in remote field locations [4]. After collection, these forms are typically hand-delivered or couriered back to relatively better-connected regional offices for electronic data input or permanent storage. While these cases have some semblance to data collection, they both have their shortcomings. In the case of electronic data input, NGOs may benefit from some degree of data compilation and analysis, although frequently, such data is prone to loss, error, and subject to lag in delivery, mostly due to logistical challenges; and in the case of permanent storage, since this data is typically stored in paper format, any future possibility of electronic data input becomes invariably harder because of storage upkeep challenges and data loss over long maintenance durations.

Many of these issues have been mitigated with the increasing availability of network connectivity. But to be sure, network reception is most commonly accessed through commercial wireless data connections that come with limited service guarantees, especially in remote locations. In practice, most data connections provide bandwidth access commensurate with GPRS speeds, and remain prone to lost connections. In such an environment most web applications perform poorly. Latency of page loads is high, and as a result, page time-outs in browsers are common [5]. Also, most traditional web applications are built with respect to the client-server model. As the number of server round trips increase, overall user experience degrades. Applications developed with a client-side architecture, while resilient to common latency problems, still perform poorly when a data connection is completely lost. In an attempt to solve these problems in the context of the aforementioned environment, we propose a software system that leverages client-side resources and continues to operate during severed network connections. We call this system COCO – connect online, connect offline.

COCO is designed to support data-tracking activities for organizations with sizeable field operations and where organization staff may only have access to intermittent and/or

ACM DEV'10, December 17-18, 2010, London, United Kingdom.

poor-quality Internet connectivity. COCO is built as a standalone application in the Internet browser and requires no additional desktop software installation or maintenance. The system is designed in an open-source, customizable framework that can be deployed without the need of professional IT or engineering staff.

Some of the key features provided by the COCO framework include:

1. The Model View Controller (MVC) framework architecture: COCO is designed using the model-view-controller pattern [6]. Developers familiar with some of the most ubiquitous web development frameworks can easily build data input applications on COCO. The architecture allows COCO to be available as a framework, making many of its components and features reusable.

2. Offline mode: One of COCO key features is to allow an application to continue to operate without internet connectivity. In COCO, this mode is user induced. On detecting low bandwidth, or repeated severed connections, a user can click on an 'Offline/Online' UI element to toggle between connection modes. These UI elements are integrated as part of the framework.

3. Latency and execution speed: COCO achieves low latency and high execution speeds by virtue of its architecture. While offline, an application built on COCO runs entirely on the client-browser. This enables COCO to take advantage of the client's resources, and as a result, eliminates the need for any data connection requirements. In online mode, COCO achieves better to comparable latency compared to traditional web applications by making data requests only when absolutely required. Incurring client-server roundtrip costs are virtually eliminated by the nature of the client architecture, and an accompanying cache.

The COCO system's framework is developed using a variety of web technologies and reference to related work.

In the next section, we discuss related work that led to the creation of COCO. In Section 3, we describe the COCO architecture, key design decisions, and relevant implementation details. In the following section, we describe and evaluate preliminary results. And finally, in the last section we summarize the primary motivation for COCO, and discuss future work.

2. RELATED WORK

In the ideation phase of COCO, we explored related work in both the academic and commercial spaces. As conveyed earlier, COCO was inspired by an appreciation for similar and proven systems in the commercial space. Although an immense corpus of related work exists in the academic and commercial spaces, our goal has been to appropriate many of these ideas into a simple, scalable, and robust implementation.

A fair amount of work can be found in the area of local browsing. Rhea [11] proposes a technique called value-based caching by breaking contents of a page into small blocks and serving these blocks in remote areas through pre-calculated hashes. This architecture makes efficient use of bandwidth, but requires a proxy to store hashes for all blocks per client. The COCO system employs an approach of caching all requisite static content before using the application, making availability of this content guaranteed. A proposal by Chan et al [12] involves caching static content asynchronously when a browser makes a request and receives a response. This technique may be useful for areas with intermittent connectivity, but unviable in zero connectivity situations.

In the context of the developing world, the TEK system [22,23] aims to empower low-connectivity communities by providing a full Internet experience using email as the transport mechanism. The TEK client operates as a proxy on the user's machine, enabling users to browse downloaded pages using a standard Web browser. New searches are automatically encoded as emails and sent to the TEK Server, which queries the Web and returns the contents of resulting pages via email. While beneficial for web browsing, this architecture is less appropriate for data input applications where the input data needs to be validated and synchronized with the main server.

Mechanical backhaul networks [24, 25, 33] which use physical transportation systems have also been deployed in many rural regions. DakNet [24] provides email and Web connectivity by copying data to a USB drive or hard disk and then physically carrying the drive among locations that have no traditional network connectivity, sometimes via motorcycles. Mechanical backhaul networks are intermittent by nature. They have long delays and are operational only for a few times every day.

Some researchers have taken broader approaches to solve the intermittent connectivity issue in rural areas. For instance, The Delay tolerant network (DTN) [26, 27] architecture aims to provide interoperable communications between and among a wide range of networks that may have poor and variable performance characteristics. The design embraces the notions of message switching with in-network retransmission, late-binding of names, and routing tolerant of network partitioning to construct a system better suited to operations in challenged environments than most other existing network architectures, particularly today's TCP/IP based Internet. Another example called the WiRE network [10] aspires to build a comprehensive connected network in rural areas. These examples advocate a change to the basic service model and system interface most web applications have become accustomed to. In another example, RuralCafe [9] intends to support efficient web search over intermittent networks. The RuralCafe approach requires hardware such as local and remote proxies, and clients. Search results are pre-fetched and stored in proxies that serve as a cache. While beneficial for the purpose of search, this architecture is difficult for COCO to piggyback on because of its specialization. Solutions such as DTN, WiRE and RuralCafe offer alternatives in network architecture to mitigate bandwidth problems in remote regions, and while there is potential in its use for complex problems, simple data input applications at the level of COCO's complexity can continue to operate with low cost GPRS.

A reasonable corpus of work is also available on distributed file systems, designed to operate in low bandwidth environments. Coda [28] is a distributed file system with its origin in AFS2. It has several features like disconnected operations for mobile computing, high performance through client side persistent caching that are desirable for network file systems. Another distributed file system, TierStore [29], simplifies the development and deployment of applications in challenged network environments. For effective support of bandwidth-constrained and intermittent connectivity, it uses the Delay Tolerant Networking store-and-forward network overlay and a publish/subscribe-based

multicast replication protocol. Both file systems are limited in widespread practical use because of confinement to platform and network architecture. And even though these file systems are fitting for applications that require sharing of storage and files among peers, their setup becomes complex for simple tasks easily addressable by systems similar to COCO.

Some researchers have considered system architectures that address the challenges of poor, intermittent connectivity. The aAQUA portal [7] is a system that offers an offline extension to allow farmers to continue to use their forum application in rural areas. To make this application available on the client machine, aAQUA requires a few additional technologies. The most relevant of these technologies include a database to store a partition of the dataset, and a webserver to service browser requests [8]. The aAQUA portal's architecture emulates a browser experience by making the browser a light-weight client, offloading all business logic to a desktop application. The COCO system in contrast provides a framework to give traditional web data-input applications offline properties. Deployment of COCO is also simpler because of its browser-client architecture, requiring no additional hardware bound desktop software installations.

A number of commercial systems exist to track and analyze enterprise data online. These include systems by Salesforce Corporation [1], and a few of its competitors. Salesforce, for example, offers highly customizable products in the data and analytics space, but follows a license model, which makes it financially prohibitive to share analytics with a wider audience. The company also provides offline capabilities as a desktop application, although this approach makes it difficult to deploy in rural locations and cumbersome for much needed software updates. While the company recommends building a browser based system by employing similar techniques as COCO, it falls short of practicability, much less a framework implementation.

Other related works offer improvements for parts of the COCO system. Approaches like Numeric paper form for NGOs [4] and CAM [30] could help an organization using COCO digitalize paper forms. Other works like Open Data kit [31, 32] and Warana Unwired [13] could enable COCO's usage to become widespread for organizations interested in direct data collection via mobile phones.

The COCO system was created by using and analyzing commercial systems, and by synthesizing academic work most relevant to data tracking and analytics in poor, intermittent connectivity areas.

In the next section we describe the design and deployment of the COCO system in depth.

3. SYSTEM DESCRIPTION

The COCO system is designed as an application residing in the browser. The system is accessible as any other website by simply requiring the client to load a URL. Designed as a framework, the system follows from the model-view-controller (MVC) pattern [6], adopted by existing web frameworks. To support offline properties, the COCO system employs the use of a light-weight relational database and local cache, both stored in the browser. Data that is stored in the relational database during offline mode is populated and replicated back to the global repository through synchronization techniques. The system checkpoints at the database row-level to ensure data consistency and integrity, and to make synchronization tasks resumable. The next section delves into the technical details of the COCO system architecture.

The COCO system is architected as a Java web-framework, similar to and in proximate feature parity with popular frameworks such as Django [14] or Ruby on Rails [15]. The core of COCO's framework comprises of a variety of open source technologies. To maximize performance as a browser application, COCO is developed using the GWT 2.0 (Google Web Toolkit) Java software development framework [16]. GWT offers the ability to develop client-side JavaScript front-end applications in Java. COCO communicates with the server side framework by transferring data in the widely accepted JavaScript Object Notation (JSON) [17] format.

To enable offline mode, COCO employs the use of Google Gears [18], a browser plug-in that provides a persistent SQLite database to store data relationally and a local cache to access all requisite static content. In offline mode, a compiled Javascript file is generated and added to Google Gear's client cache for access by the browser. In online mode, COCO operates like any other web application, and is designed to reduce the number of requests to the server.

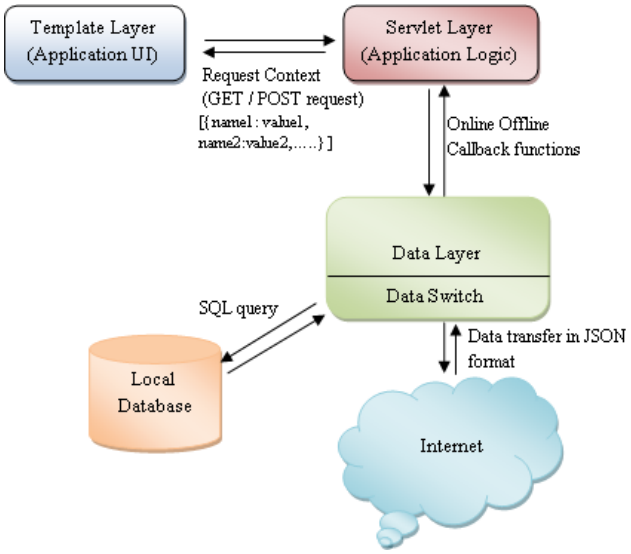

Figure 1: COCO Architecture

The COCO framework comprises three layers following the model-view-controller (MVC) pattern: template, servlet, and data (See figure1). The template layer defines the application UI. It makes use of HTML, CSS, images and some sparing use of native JavaScript to describe web forms. The template layer uses a 'RequestContext' object to exchange information with the servlet layer, attempting to emulate a real-world HTTP request. The 'RequestContext' serves as a data packet to transfer information from the template layer to the servlet layer. Each time a request is made by the template, the nature of the request is set similar to how the HTTP protocol sets a GET or POST. A query string containing name and value pairs is also passed down as a variable in the Request Context object.

The servlet layer is the business logic of the framework. It makes calls to the appropriate functions defined in the data layer based on the nature of the request from the template. It passes an 'OnlineOfflineCallback' object to the data layer. Functions defined in the 'OnlineOfflineCallback' object are called back when the data layer sends data back to the servlet, depending on

whether the request is served from the online web server, or the local offline database.

The data layer on receiving a request from the servlet determines the status of the application. If the application is online, the data request is served from the online repository, otherwise from the client's local database. Data transfer from the online repository takes place in the form of JSON objects while data transfer from the local database is in form of 'ResultSet' objects returned on running the SQL query. The data layer de-serializes the response returned from the online repository or local database into Java objects and passes these Java objects to the servlet layer using 'OnlineOfflineCallback' functions. The servlet layer then performs the application logic and passes back a response to the template layer. The template layer parses the response and renders changes to the browser.

3.1 Synchronization Details

One of the primary features of COCO is to operate in areas with intermittent connectivity. For an application to operate seamlessly, COCO caches static content and user data on the local client. COCO makes use of a technology called Google Gears, which has an inbuilt server that is capable of serving static content locally. Google Gears also provides a SQLite database which stores user data locally. Periodically, as and when network connectivity becomes available, user data is synchronized with the main repository to establish a consistent view between the local and global databases.

We have divided our synchronization process into two stages: (1) downloading data from the online repository to the local database and (2) uploading added user data from the local database to the online repository.

A user enables offline mode in COCO by downloading their data from repository while in online mode. The download initiates a HTTP GET request, sending along with it the user's login credentials. In response, the server replies with a unique id associated with the user. This unique id serves as a global primary key, shared across all tables, in the user's local database. This global primary key gets incremented on each SQL 'insert' operation into the local database.

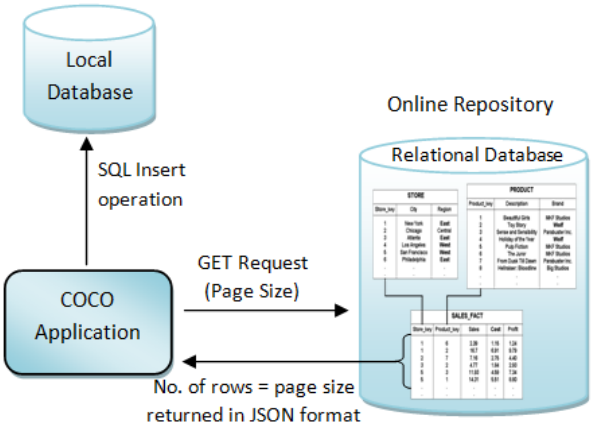

Figure 2: Download operation

After receiving the global primary key and saving it into the local database, the application issues another HTTP GET request to the server with an easily customizable application constant called 'Page size'. Page size is the number of table rows to be retrieved from the server with each GET request. The server, in turn, returns rows equal to page size as a JSON response object. On receiving the response, the application de-serializes the response into Java objects and stores them into the local database. The application then updates the global primary key in the local database. The application continues to make HTTP GET request to the server until it receives a marker to denote end of transfer.

The download process can resume from the last inserted row if a GET request is interrupted or the connection is lost during the download process.

Once the user's data is completely downloaded, he/she can operate the application in offline mode. In offline mode, all user operations are logged into a local table called 'FormQueue'. With network connectivity, a user can synchronize this data by uploading it back to the online repository.

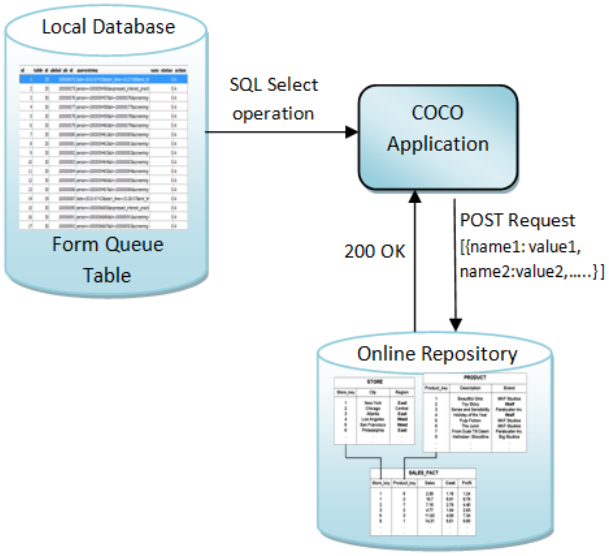

Figure 3: Upload operation

The upload process reads each row from the 'FormQueue' table and makes a POST request to the online repository. This module, like download, is also resumable.

3.2 Deployment

The COCO system can be easily installed without the need of IT/engineering staff. On logging into the COCO system, users see a dashboard as shown in figure 4:

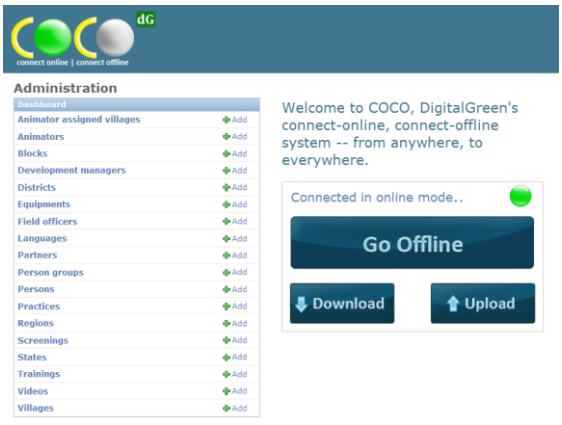

Figure 4: A screenshot of COCO application

By default the application is in online mode. Users can choose to go offline by downloading their data and then clicking on the 'Go Offline' link. When a user clicks on the download link, he/she will be prompted to download Google Gears if it is not already installed on the user's browser. Once Google Gears has been installed, downloading of data will start on clicking the download link. The application can be offline enabled by clicking on the 'Go Offline' link, once the downloading is complete.

Users can perform all operations in offline mode that can be performed in online mode. They can choose to go online when they have connectivity by clicking on 'Go Online' toggle button. Once they are online, they can upload their local modified data back to the main server by clicking on the 'Upload' button. When the data has been uploaded to the main server, locally added data becomes visible globally.

COCO is deployed at various rural locations in India that experience intermittent Internet connectivity. We describe preliminary results and feedback in the next section.

4. PRELIMINARY RESULTS

We installed the system for an India-based NGO – Digital Green [19] – and ran preliminary tests both in the lab and in the field with live data and real users. For the purpose of consistency, we collected our data by running the same tests for the same user in the lab and in the field, ensuring consistency in data sizes across test environments. Tests were conducted by simply using the application. Test results measuring latency were captured by web-profiling tools. In addition, we used networking desktop tools to collect information on basic statics such as bandwidth speeds and latency between packet hops. The following sections are an evaluation of our results by making a comparison between a traditional web application and the COCO system.

4.1 Methodology

We evaluate the benefits of COCO in two test environments, Digital Green lab with access to high-speed broadband, and a village, characteristic of a remote location with access to limited data connectivity. For both cases, we compare a simple data input system across three modes of operation: a traditional web application, COCO's online mode, and COCO's offline mode. Moreover, in both instances we hope to highlight COCO's three most salient benefits: continued offline operation for limited bandwidth situations, latency savings by reducing client-server roundtrips, and finally, execution speeds by virtue of COCO's browser-client design. In our evaluation, we use Digital Green's (NGO) data input application [20] as an example. The following are functional tests conducted across all modes of operation:

1. A test that compares retrieving and adding a simple web form. This test measures the time it takes to retrieve and save a small amount of data by issuing a HTTP GET and POST request, respectively. The test is conducted for a traditional web application and COCO's online mode. The equivalent test for COCO's offline mode is only relevant in the village setting given the broadband access in the lab setting.
2. A test that compares retrieving and adding a complex web form to gauge scalability across modes of operation. This test is similar to the first test in what it intends to measure.

We also conducted tests to measure overall task completion time for COCO's synchronization features. The two tests conducted in a lab and village setting while in COCO's online mode are:

1. Compare the time it takes the 'download' feature to complete for a small and large data set. This feature involves downloading a partition of the global repository to the local machine.
2. Compare the time it takes the 'upload' feature to complete for a small and large data set. This feature involves uploading offline changes back to the global repository.

4.2 Results

We describe and compare functional test results for retrieving content, using Digital Green's application as an example [20]. Built as a traditional web application, the data input application requesting a simple web form that requires 1 HTTP GET request for HTML content and 4 additional GET requests to load images and CSS, resulting in a total of 5 GET requests. In COCO's online mode, the number of requests remains the same unless the user has previously gone offline once, in which case, except for any dynamic data request, the local cache – separate from a browser cache -- serves all static content, resulting in a total of 1 GET request. In COCO's offline mode, 0 GET requests are made. Since requests are driven by the application, these results hold true for both lab and village test environments.

We now compare our latency measurements across both test environments. A simple test in our lab showed download bandwidth speed of 880kbps, and upload of 440kbps. Contrast these speeds with our Digital Green example village at Surshetikoppa, Karnataka, India, and we get a download speed of about 7kbps, and an upload speed of 3kbps. Running 'traceroute <server_location>' showed an added 350ms for a transatlantic hop from Delhi to New York, USA, and back. With these statistics in mind, we conducted our tests across the aforementioned modes of operation. We begin by measuring latency for loading a simple web form in a lab. For a traditional Digital Green data input application, the combination of 5 GET requests took < 1 second. In COCO's online mode, a page load took the same amount of time, but fared better after going offline once, which allowed the local cache to warm up by storing all requisite images and CSS. The resulting GET request to retrieve HTML content took 0.25ms. There were no measurements taken for offline COCO, since this mode does not issue any HTTP requests.

We now present our results for the example village. Assuming an uninterrupted bandwidth supply, loading a simple web form by the traditional web application and online COCO took about 20s. However, COCO in online mode after offline usage fared much better, with an average page load of about 5s. As described earlier, no latency was experienced in COCO offline mode. Although in an attempt to compare overall experience, we did measure absolute page load time, which were in the sub-second range.

Results for tests comparing synchronization tasks for a lab and village setting were also collected. As expected, the results are a function of bandwidth speeds, with an added assumption that uninterrupted bandwidth was available at the village.

Our tests show that application usage patterns amortized over time perform better on COCO compared to a traditional web application. In the case of online access, COCO's online mode

performs on par or better depending on prior usage patterns. COCO's offline capabilities allow for continued usage and execution speed, making any comparison to a traditional web application on these parameters moot.

5. CONCLUSION & FUTURE WORK

In this paper, our motivation was to build a system that offers a solution for data tracking and analytics to organizations, especially NGOs, operating in poor, intermittent connectivity areas. Increasingly, NGOs are being asked to prove their effectiveness through more sophisticated, technology driven methods. This paper seeks to introduce the COCO framework as an inexpensive, fast, and robust data collection and analytics system for organizations operating in remote locations. Our evaluation and results show the effectiveness of COCO in online mode as compared to a traditional web application, and its ability to continue to operate in intermittent to zero connectivity areas. At its core, the COCO system is a framework that seeks to contribute these features and benefits to organizations with similar desires.

Looking forward, we intend to make COCO extensible to interoperate with other platforms. COCO currently uses Google Gears as it provides a local server that caches and serves application resources locally and a SQLite database to store user's data. The latest version of the HTML standard, HTML 5 [21], comes bundled with an application cache to serve the application resources locally and a SQL database for local storage.

We plan to make the COCO application adapt well to the local bandwidth conditions by estimating bandwidth and setting application parameters dynamically. This will further enhance the performance of COCO on lower bandwidth connections.

Currently, users are required to manually toggle between "Online/ Offline" modes. A possible optimization could be to automate this process. This will require some automated detection of network access, which is challenging as network bandwidth has a continuous spread. The reason for the application to be offline could be any of these - the browser has the 'work offline' flag set, the network cable is unplugged, (Domain Name System) DNS is down, the server is unreachable or the server has a bug and cannot process your request etc.

We plan to integrate an analytics layer to enable users to analyze data as it is inputted to the application. Going forward, we are also looking at a possibility of integrating COCO with existing customer resource management (CRM) systems, like Salesforce. This will enable existing applications on different CRM platforms to operate on the browser and work offline with enhanced performance.

6. ACKNOWLEDGMENTS

We extend our gratitude to Rikin Gandhi and Kentaro Toyama for helping review this paper. We thank Digital Green for the inspiration. We are also grateful to our partner NGOs--Pradhan, Varrat, BAIF and SPS--for their support and cooperation in testing this system.

7. REFERENCES

[1] Salesforce.

http://www.salesforce.com.

[2] Microsoft .NET Customer solution case study. Rural Banking comes of age in India. http://srichidsirvice.com/downloads/ruralbanking_comes_age.pdf.

[3] S.M. Mishra, J. Hwang, D. Filippini, T. Du, R. Moazzami, and L. Subramanian. 2005. Economic Analysis of Networking Technologies for Rural Developing Regions. 1st Workshop on Internet and Network Economics.

[4] G. Singh, L. Findlater, K. Toyama, S. Helmer, R.Gandhi, R. Balakrishnan. 2009. Numeric Paper Forms for NGOs. Proceedings of the 3rd International Conference on Information and Communication Technologies and Development.

[5] J. Chen, L. Subramanian, and K. Toyama.2009. Web Browsing under Poor Connectivity. CHI Proceedings on Human Factors in Computing Systems.

[6] A. Leff and J.T. Rayfield. 2001. Web-application development using the model/view/controller design pattern. pp. 118–127.

[7] aAqua Portal.

http://www.aaqua.org.

[8] S. Sahni and K. Ramamritham. 2007. Delay Tolerant. Applications for Low Bandwidth and Intermittently Connected Users: the aAQUA Experience. WWW2007, Banff, Canada.

[9] J. Chan, L. Subramanian, J. Li. 2009. RuralCafe: Web Search in the Rural Developing World. WWW 2009 Madrid.

[10] L. Subramanian. A Low-Cost Efficient Wireless Architecture for Rural Network. http://www.cs.nyu.edu/~lakshmi/wire.pdf.

[11] S. C. Rhea, K. Liang, and E. Brewer. 2003. Value-Based Web Caching. Proceedings of 12th WWW Conference.

[12] H. Chang, C. Tait, N. Cohen, M. Shapiro, S. Mastrianni, R. Floyd, B. Housel,and D. Lindquist.1997. Web browsing in a wireless environment: disconnected and asynchronous operation in artour web express. MobiCom '97: Proceedings of the 3rd annual ACM/IEEE international conference on Mobile computing and networking, pages 260–269, New York, NY, USA.

[13] R. Veeraraghavan, N. Yasodhar, K. Toyama 2007. Warana Unwired: Replacing PCs with Mobile Phones in a Rural Sugarcane Cooperative. International Conference on Information & Communication Technologies for Development. Bangalore.

[14] Django.

http://www.djangoproject.com.

[15] Ruby on Rails.

http://rubyonrails.org/.

[16] Google Web Toolkit. http://code.google.com/webtoolkit/.

[17] JSON.

http://www.json.org.

[18] Google Gears.

http://gears.google.com/.

[19] Gandhi, R., Veeraraghavan, R., Toyama, K., &Ramprasad, V. 2007. Digital Green: Participatory video for agricultural

extension. Proceedings of the IEEE/ACM International Conference on Information and Communication Technologies and Development.Bangalore, India.

[20] Digital Green admin site.

http://www.digitalgreen.org/admin/.

[21] HTML 5, Latest published version. Editor:I. Hickson, 2010. http://www.w3.org/TR/html5/.

[22] L. Levison, W. Thies, and S. Amarasinghe. 2002. Providing Web search capability for low-connectivity communities. ISTAS, pages87–91.

[23] W. Thies et al. Searching the world wide web in low-connectivity communities. WWW, 2002.

[24] A. Pentland, R. Fletcher, and A. Hasson. DakNet:rethinking connectivity in developing nations. Computer, 37(1):78–83, 2004.

[25] United Villages. http://www.unitedvillages.com.

[26] K. Fall. A delay tolerant network architecture for challenged internets, 2003.

[27] S. Jain, K. Fall, and R. Patra. Routing in a Delay Tolerant Network. In Proc. ACM SIGCOMM, pages 145–158, August 2004.

[28] James J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. In Proc. of the 13th ACM Symposium on Operating Systems Principles (SOSP), 1991.

[29] M. Demmer , B. Du , E. Brewer. TierStore: a distributed filesystem for challenged networks in developing regions. Proceedings of the 6th USENIX Conference on File and Storage Technologies, p.1-14, February 26-29, 2008, San Jose, California.

[30] Tapan S. Parikh. Using Mobile Phones for Secure, Distributed Document Processing in the Developing World. IEEE Pervasive Computing, v.4 n.2, p.74-81, April 2005.

[31] Open Data Kit. http://opendatakit.org .

[32] Y. Anokwa, C. Hartung, W. Brunette, A. Lerer, G. Borriello. Open Source Data Collection in the Developing World. IEEE Computer Magazine.2009.

[33] R. Y. Wang, S. Sobti, N. Garg, E. Ziskind, J. Lai, and A. Krishnamurthy. Turning the postal system into a generic digital communication mechanism. ACM SIGCOMM, pages 159-166, Portland, OR, Aug. 2004