Data Compression, Storage, and Viewing in Classroom Learning Partner

by

Jessie L. Mueller

B.S., Massachusetts Institute of Technology (2011)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2012

Author .....
Department of Electrical Engineering and Computer Science
May 22, 2012

Certified by .....
Dr. Kimberle Koile
Research Scientist, MIT CECI
Thesis Supervisor

Accepted by .....
Prof. Dennis M. Freeman
Chairman, Masters of Engineering Thesis Committee

Data Compression, Storage, and Viewing in Classroom Learning Partner

by

Jessie L. Mueller

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 2012, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, we present the design and implementation of a data storage and viewing system for students' classroom work. Our system, which extends the classroom interaction system called Classroom Learning Partner, collects answers sent by students for in-class exercises and allows the teacher to browse through these answers, annotate them, and display them to the class on a public projector. To increase and improve data transmission, our system first intelligently compresses student work. These submissions can be manipulated by a teacher in real time and also are saved to a database for future viewing and study. This duel functionality allows for the analysis of student work from multiple lessons at the same time, as well as backup of student work in case of system failure. Teachers can compare the work from multiple students, as well as create portfolios of student work over time. The data storage and viewing system gives both teachers and researchers a view of both students' learning and how students interact with the software system.

Thesis Supervisor: Dr. Kimberle Koile
Title: Research Scientist, MIT CECI

Acknowledgments

I'd like to thank the people who helped and motivated me as I researched and wrote this thesis. I'd like to thank my supervisor, Dr. Kimberle Koile, for her vision, dedication and guidance. Her passion and commitment moves the entire team and made this thesis possible. I thank Andee Rubin from TERC for her insight and attention to detail. Her understanding of classroom dynamic greatly shaped the design of my work.

I am thankful for the rest of the CLP team. Thanks to Kelsey Von Tish and Eryn Maynard for their help in brainstorming and debugging and for being cheerful working mates in the office. I am very grateful for the collaboration with Claire DeRosa. Claire's work and support helped the incorporation of a new serialization method go smoothly. I am also especially grateful for Steve Chapman's work and dedication to CLP. His effort enabled the enhanced data management system to be included in CLP.

I am thankful for the hours Sherry Nichols, Renee Lockwood, and their students spent using CLP software. Their feedback was vital to this thesis's success. Additional thanks to Northeast Elementary for working with us. Thanks to Judy Storeygard and Lily Ko from TERC for their educational insights. Additional thanks to Lily for her help in the classroom doing data backups.

I am extremely grateful for the guidance of my friend and former colleague Martyna Jóźwiak, and for the motivation and support from my friends, especially Julia Boortz, Alex Dowling, Dina Betser, and Ruthi Hortsch. I'd like to thank my academic advisor Julie Greenberg for her support over the past four years. Finally I'd like to thank my parents, whose encouragement and wisdom is never ending. Even though I strayed from the family tradition of studying chemistry, your advice during my time at MIT has always been correct.

Contents

Chapter 1

Introduction

Researchers and educators have been interested in using new technologies in the classroom for many years. There are programs that give laptops to every student and create forums in which to continue discussions outside of the classroom. Recently, the focus of this research has turned to tablet computers and their uses and effects in the classroom. In high school and college case studies, tablet computers have demonstrated their usefulness by providing rapid feedback and increasing student satisfaction [2, 13]. The use of tablet computers can enable the type of clear, ongoing communication between students and teachers that is central to successful educational systems [25, 10, 21].

These laptop programs strive to improve student learning, but often have no mechanism for investigating how student learning occurs. This situation prevents educators from understanding how to best use these technologies and diminishes research returns. This thesis describes the work done to integrate a flexible data storage and viewing system into a classroom interaction system called Classroom Learning Partner (CLP), as well as to illuminate the broader technical challenges involved in the setup of classroom-scale networking systems. Our approach involved the use of a specialized serialization process and non-relational database to compress, store, and retrieve student work. The data storage and viewing system eases in-class use, allows teachers to easily track student work, and helps to illuminate CLP's effect in the classroom.

1.1 Overview

In this thesis, we created a data storage and viewing system for Classroom Learning Partner, a pen and touch classroom interaction system that runs on wirelessly connected tablet computers.To enable long term storage, we evaluated and improved networking and data compression practices and introduced the use of a non-relational database. Using CLP, students complete exercises in "electronic notebooks" on their own tablets and then submit their work to the teacher by taping a button when they are done. Our extension to CLP enables submissions to be saved to a database, with no extra steps required by students or teachers. Student work can be saved to a central database in real time, in addition to being cashed locally on student machines. The saved student work can be used to retrieve electronic notebooks for later classroom use or can be queried using our data viewing interface. The data viewing system allows educators and researchers to access electronic notebook pages from multiple students or from different lessons in one view, instead of needing to access multiple electronic notebooks.

We tested our system during visits to two fourth grade classrooms in Waltham, MA, and we observed the effects on system performance of saving all student work to a central database. We also logged and analyzed information on the size and time required to save various kinds of student work.

1.2 Previous Work

Early technology that enabled class participation in answering in-class problems included Personal Response Systems (PRS). A PRS consists of a wireless transmitter with buttons, often called a clicker, used to register a student's response to multiple choice or matching questions. The teacher can display the class's answers immediately in the form of a histogram, providing real-time feedback to students and helping the teacher understand if he or she should move on in the lesson or go back and review [6]. A PRS typically provides a method for saving and exporting questions and

student answers, but provides few means for search this data [22]. Furthermore, they can only be used with multiple choice and matching questions questions[1].

Classroom Presenter (CP), developed by Richard Anderson [24, 23], expanded upon the work of PRS by allowing teachers to ask all kinds of questions, though it did not aggregate student answers. CP consisted of a networked set of tablet PCs. Students used their pens to create "digital ink" inscriptions[2] to answer questions on their machines. They then wirelessly sent their work to the teacher, and the teacher could send back written feedback. Teachers also were able to lead class discussions based on student work displayed anonymously on a projector. CP improved classroom interaction by making it easier for teachers to collect and respond to student work in class. Later extensions to CP enabled the storing of student inking in a relational database [8]. In that work, a simple schema was developed for storing student strokes and experiments were conducted to verify that ink could be serialized into strings without data loss.

Our research group works on Classroom Learning Partner (CLP), a tablet-PC-based classroom interaction system that shares CP's goal of the submission and sharing of student work. An early version of CLP extended CP by adding ink interpretation, aggregation of student answers, and a permanent relational database [14, 17, 16, 15]. Currently, a new version of CLP is being developed for use in elementary school math and science education, funded by a four year NSF project called INK-12 [12, 11]. The new CLP expands upon the previous version by adding a diverse set of objects that give students additional tools for solving problems and creating explanations. It also now includes new interpretation methods that will help categorize submitted student work automatically [18].

In the previous version of CLP, student work was stored in a relational MySQL database [20]. This database emulated an object-oriented database by having a dif-

[1]PRS companies currently are working to expand the kinds of questions that work with their systems.

[2]By "inscription" we mean handwritten drawings, graphs, notes, etc.

ferent table for each relevant object. The schema used was functional but large and difficult to change. This kind of database was abandoned in the new CLP and replaced with a more flexible database called MongoDB, which will be described in detail in later sections.

1.3 Outline

Chapter 2 gives a high level overview of our objectives, approach and implementation. We discuss CLP's network design in Chapter 3 and our improved serialization mechanism in Chapter 4. Chapter 5 outlines the design and usage of a non-relational database for student work and Chapter 6 demonstrates a data viewing system capable of locating and displaying student work stored in the database. Results from classroom testing are presented in Chapter 7, and we discuss future work and contributions in Chapter 8.

Chapter 2

System Architecture

CLP runs on a set of wirelessly connected of tablet computers, which take on one of four distinct roles, depending on the mode in which the software is run. Students have tablets on which they create their work, save it, and submit it to the teacher. (See Figure 2-1.) The interaction metaphor is that of an electronic notebook: Students draw and write their work on notebook pages, which the teacher has created using the CLP authoring tool. The teacher's tablet receives and presents the submissions to the teacher. The teacher also can choose a subset of student submissions to send to the projector tablet. The projector tablet is connected to a projector and displays the student work anonymously to the class for discussion. (See Figure 2-2.) Submitted work is also saved to the server machine, where it is stored in a non-relational database for later use. Figure 2-3 diagrams the system of networked machines.

2.1 Data Hierarchy

As previously mentioned, the data metaphor for CLP is an electronic notebook. Students have different notebooks for different subjects and lessons. A notebook is comprised of multiple pages on which students can do their work. Pages contain ink strokes and pages objects. Strokes are the markings made with the pen on the screen. For storage purposes, strokes can be serialized into strings. Page objects are elements with specific properties. Some current page objects are text boxes, images,

Figure 2-1: **Student working on a tablet.** The CLP software enables student to submit written and drawn work and explanations.

Figure 2-2: **CLP use in a fourth grade classroom.** The teacher (on the right) receives work from students and controls the projected display from her tablet.

audio recordings, and an object that we call a stamp, which is used to create multiple identical images. Page objects and strokes that are authored by an instructor are immutable on student machines. Students can add strokes and create certain page objects on their own. Figure 2-4 shows this hierarchy, including example page objects and strokes.

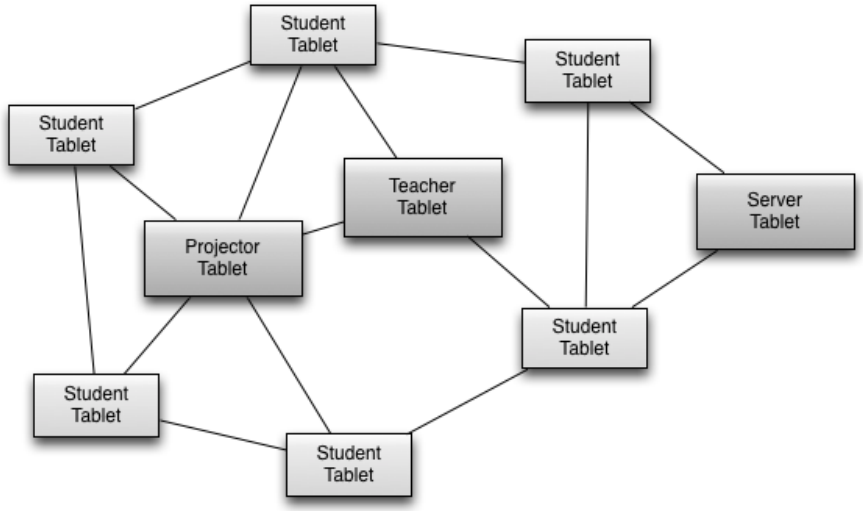

Figure 2-3: **An outline of the CLP wireless network.**

2.2 Page History

This year logging of student actions taken when interacting with pages was added to CLP. The page history, as it is called, logs student inking, moving and placing of page objects, saves and submits. Using the page history, teachers are able to play back a student's sequence of work, gaining insight into his or her thought processes [5]. Logging student actions enables smarter saving of student work (see Section 3.3.1), but also results in additional data that needs to be saved to a central location. The page histories and the extra data associated with them motivated the adoption of new data serialization strategies discussed in Chapter 4.

2.3 Objectives

CLP has been successfully used to increase classroom communication channels and maintain student engagement. The deployment of CLP in classrooms for testing and educational research has enabled students to create much relevant work for both teachers and educators. Unfortunately, the previous version of CLP did not support an easy way to organize this work in one location for viewing or retrieval. This version also was not robust enough to effectively send large multimedia or page histories over

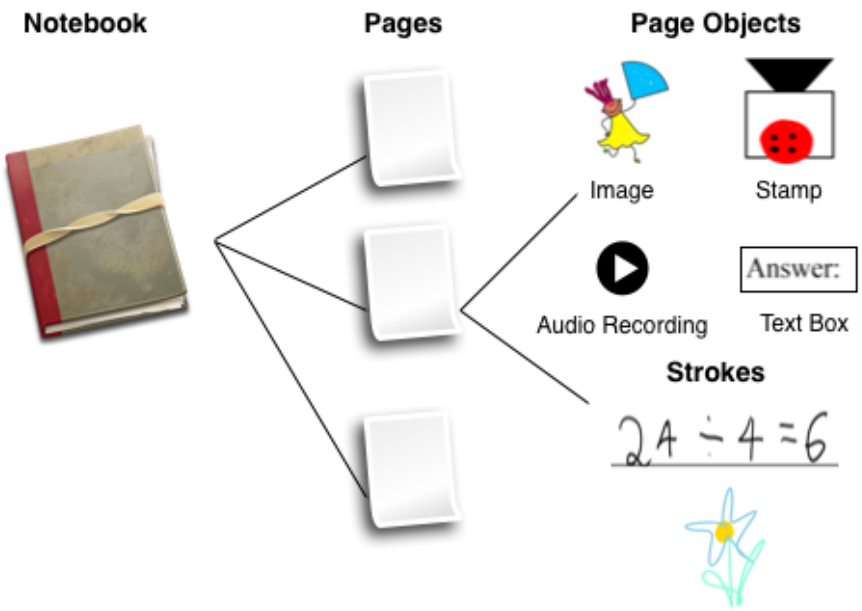

Figure 2-4: **The CLP data hierarchy.**

the network. The goal of this thesis was to create an easy method for saving all student work/data so that it can be queried and viewed at later times. We accomplish this goal by:

- Creating network protocols for sending all student work to one machine.
- Using new serialization techniques that drastically reduce the size of data sent over the network[1].
- Storing student work in a flexible and easily queryable database.
- Creating a simple viewing system to display results of database queries of student work.

[1]Serialization is the process of converting a data structure into a format that can be sent through a network or saved to disk.

Chapter 3

Saving Over a Network

Networking is at the core of the CLP system architecture. A wireless network between machines allows students to submit work to the teacher tablet, and allows the teacher to send work to the projector tablet to be displayed. The first goal of this thesis was to modify the network architecture to allow for the sending and saving of all student work to one machine. In a previous version of CLP, student notebooks were only saved locally on student machines, making the task of collecting all student work very time consuming. In this chapter we discuss the CLP network architecture and the implementation and results of the networking component of system that saves student work.

3.1 Network Architecture

CLP uses a mesh network with flooding to transmit messages between different tablet machines. A mesh network consists of a set of machines connected, in our case wirelessly, forming a connected graph [7]. In flooding mesh networks, messages are not routed to a specific destination. For example, a message sent from machine A intended for machine B, is propagated to every machine in a mesh networks, as shown in Figure 3-1. Each machine looks at the content of the message to decide if the message is intended for it. This setup allows the same message to be sent to many machines easily and does not require that machines have knowledge of network

topology.

In contrast, messages sent in routing mesh networks have a specific destination. A message sent from machine A to machine B would use a shorter path instead of propagating through the whole network. Routing mesh networks maintain more information about network topology. The setup and maintenance of this information can be difficult, especially in wireless networking situations. The information, however, does succeed in reducing the network bandwidth used when a message is sent to a specific target. Figure 3-1 shows in bold the shortest path between A and B, which would be used in a routing mesh network.

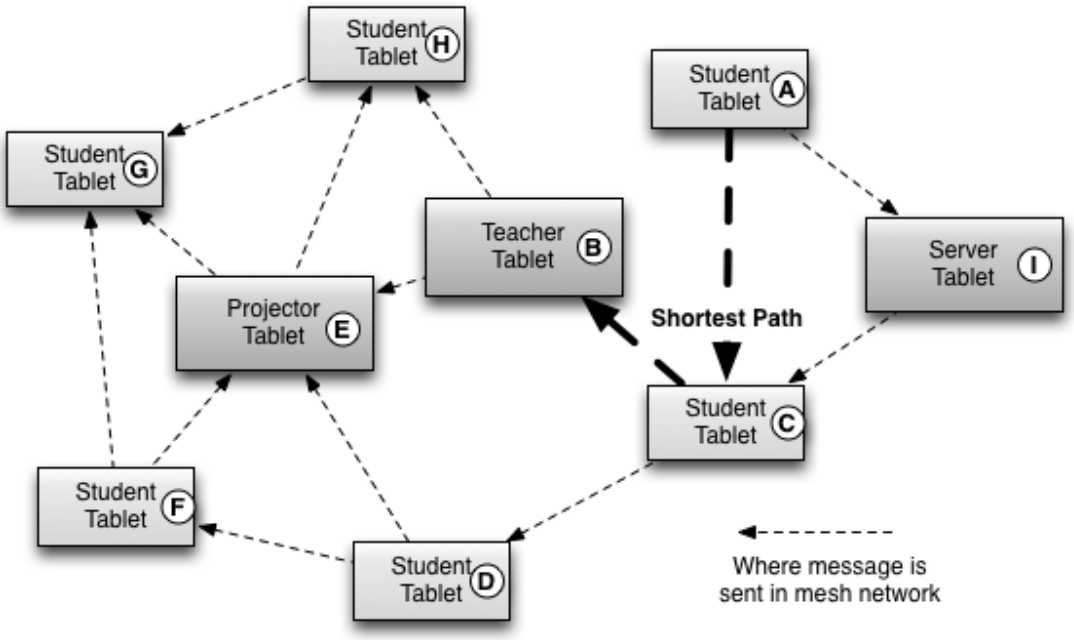

Figure 3-1: **Message sending in the CLP Mesh Network**

3.2 Motivation

As discussed in Section 2.3, a previous version of CLP made it difficult for teachers or researchers to look at or organize student work spanning multiple lessons. Thus, it is important to be able to send all student work to one location. The implementation of

networked saving of a page must take into consideration the organization and potential limitations of the network architecture. Researchers chose to use a mesh network in CLP for its simpler implementation and the ability to easily send the same message to multiple machines. This architecture was sufficient when only sending student work over the network, but it has become strained this year with inclusion of more data in the form of multimedia elements on pages and the logging of student interaction histories. (See [5] for more information about these additions). To reduce the use of network bandwidth, the page saving system must be smart about what gets sent, sending as little redundant data as possible. Saving via the network should occur in real time during classroom use be relatively fast so as to not disrupt classroom usage.

3.3 Implementation

We save student work on a page granularity. This design decision groups together page objects that appear on a page and helps to minimize network bandwidth usage. In a previous version of CLP, the entire notebook was saved locally to disk. Networked saving of an notebook would be the easiest to implement, given the previous work, but would negatively impact network performance. Students generally save after completing a few pages. If CLP were to send the whole notebook over the network each time, it would be saving many unchanged pages. Furthermore, saving only a whole notebook is inflexible if a teacher later wants to create a new notebook or view student work from pages originally in different notebooks. Using a page granularity to send student work balances these two goals.

3.3.1 Saving Pages

Pages are sent to the server machine to be saved only when they have been modified. The page history keeps track of all changes to a page. If the last item logged is a savePage item, the page has not changed since the last save, and is not sent to the server. If the item is of any other type, such as addInk or movePageObject, the page has modified data and is sent to the server to be saved. Page histories, because

they are so large, are not saved during class sending and replaced afterwords. The networked saving of page histories is discussed in the next Section 3.3.2.

To further reduce network bandwidth, submitting a page to the teacher also passes the page to the database. In the CLP mesh network, messages are propagated to all machines. In the previous version of CLP, every machine except for the teacher machine ignored page submission messages. In the current version, the server machine accepts these messages as well. The server saves the page to the database as a submission and also as a page save. The database has separate collections of submitted work, or the work that was purposely sent to the teacher, and saved work. While the submission collection may hold multiple copies of some student's page 1, because he or she submitted an updated version, the saved pages collection has only the most recent version, whether submitted to the teacher or not. See Chapter 5 for more information about CLP's new database usage. After sending a page submissions, a submitPage item and a savePage item are added to the history. The page will not be sent over the network to be saved again until it has been modified.

3.3.2 Saving Page Histories

Page histories tend to be larger than that the pages themselves and are saved separately to prevent network congestion. Pages with a large amount of student work rarely go above 200kB. The history associated with such a page is often between 1000kB and 2000kB. Currently in CLP, all page histories are sent to the server at the end of class. While histories could be saved along with pages, we chose to separate the two actions as we analyzed their effect on our network. This tradeoff allowed page histories to be saved for future use without negatively impacting in system performance during class.

Chapter 4

Data Serialization

Serialization is the process of converting a data structure into a format that can be stored and later extracted into its original form. A CLP notebook is serialized when it is saved to the database, and pages are serialized before being sent over the network. In this section we discuss a new serialization method for CLP that compresses messages, requiring much less network bandwidth.

4.1 Motivation

The inclusion of page histories and the networked saving of all student work led to more traffic on the CLP wireless network than in previous years. Early testing indicated that the smarter sending strategies discussed in Chapter 3 would not prevent long (on the order of minutes) delay during classroom use. As shown in Figure 4-1, the delay in sending across one machine increases gradually in messages smaller 2000kB. Above that threshold though, a large variance is seen is sending delay, with nearly half of the messages being received 10 seconds or longer after being sent. Student page histories are routinely this size or larger. This test was performed with two machines in a small network. For classroom use,involving 20 to 30 machines, the network saturation is reached even sooner. For this reason, we wanted to improve our serialization methods, making whatever data that did need to be sent over the network as small as possible. In addition to creating smaller serialized objects, any

new method needs to be fast, so as to not impede system performance during class. Furthermore, it must be flexible enough to handle the variety of page objects CLP uses.

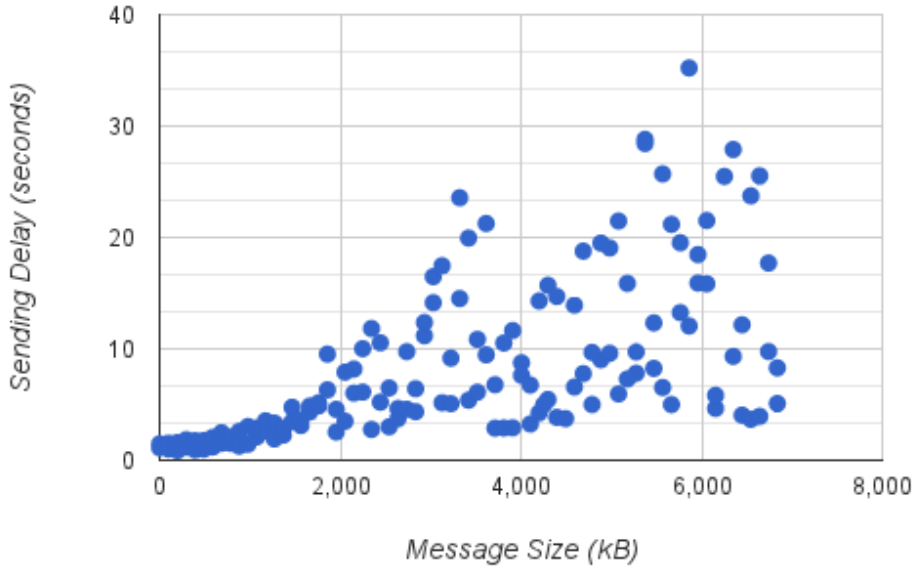

Figure 4-1: **Time to send message to teacher verses message size.** Above 2000kB, network performance becomes unreliable, with many messages being delayed. This test was done on a network 1/5 the size used in a normal classroom.

4.2 Data Serialization Formats

The previous version of CLP used the BinaryFormatter for all serialization tasks. BinaryFormatter is a standard .NET class for serializing complex object clusters. It records the structure and hierarchy of objects by including auxiliary data about each object in the serialized output. For example, the serialized output for a text box page includes the name of each of it parameters, such a content, position, size, etc.; their type and their value. The inclusion of auxiliary data means objects require minimal annotation to be used with BinaryFormatter. The auxiliary data is repeated for each instance of an object on a page, resulting in larger serialized outputs. Output size

is less of an issue when serializing whole notebooks to be saved locally to disk, but larger serialized outputs sent over the CLP network use more bandwidth.

This year we introduced a serialization method using Google's protocol buffers, a platform independent serialization mechanism [1]. Instead of including auxiliary data in the serialized output, protocol buffers use a separate file, the .proto file, that defines data types and hierarchy. This file is then compiled, providing methods for serialization and deserialization. Taking the data type and hierarchy information out of the message greatly reduced message size. Instead of including the names and types of parameters as BinaryFormatter-created serializations do, a protocol buffer output is primarily composed of the actual data. Figure 4-2 compares the output of the BinaryFormatter and Google's protocol buffers.

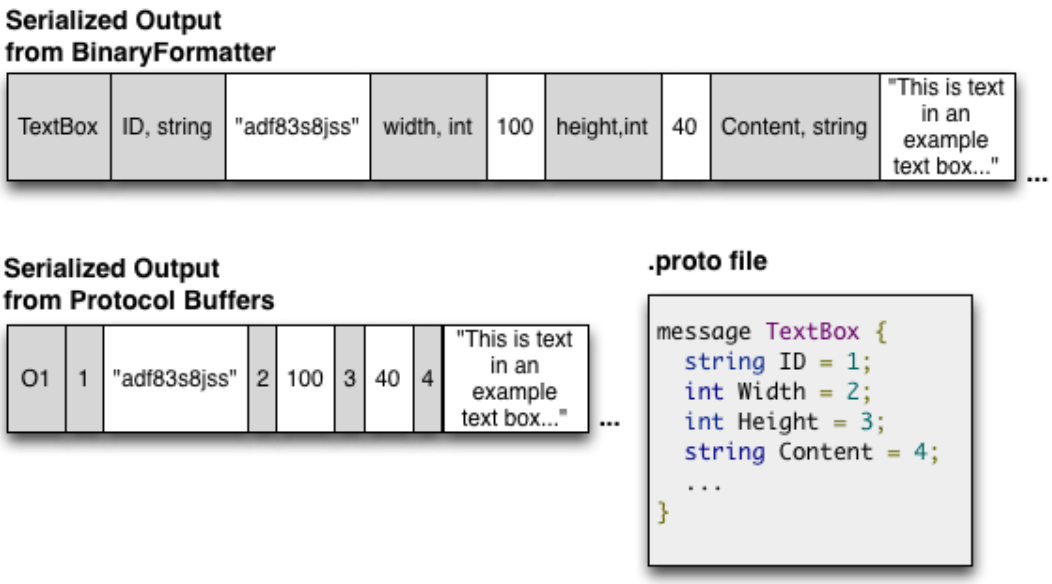

Figure 4-2: **A comparison of the output of BinaryFormatter and protocol buffer serialization routines.** Outputs from protocol buffers are smaller because type structure is defined once in an external file.

4.3 Implementation

A basic implementation of protocol buffers in CLP involves describing the data hierarchy so that less auxiliary data is needed in the output. This work allowed us to successfully serialize/deserialize notebook pages, but it produced only minor improvements over BinaryFormatter serialization. For some pages of student work, e.g. that shown in Figure 4-3, the protocol buffer serialized output was larger than the BinaryFormatter output.

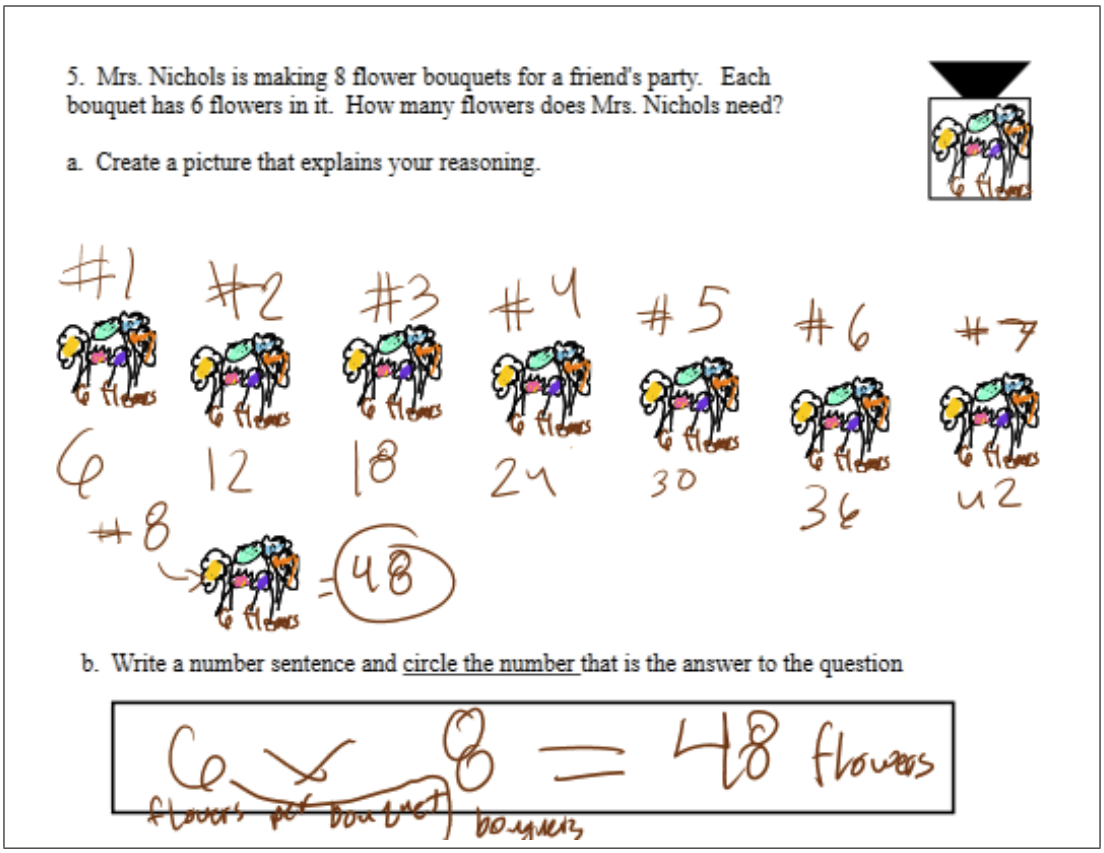

Figure 4-3: **Example of student work that can be better serialized with BinaryFormatter.** The basic implementation of protocol buffers created a serialized output that was 20 kB larger than that from BinaryFormatter.

The poor performance of protocol buffer serialization in the above example resulted from the way protocol buffer handles duplicated string: BinaryFormatter enables string interning by default, while protocol buffers does not. String interning is an optimization that stores just one instance of each unique string in the output.

Copies of this string are referenced back to the stored copy, preventing the string from being duplicated in the output. This capability allows for smaller outputs that use less network bandwidth when transmitted. Serialization converts the notebook page to a string. Therefore, student work with many repeated strokes or images, such as the one shown in Figure 4-3, contains many duplicate strings in the basic protocol buffers output.

While string interning is not included in the protocol buffer specification, it can be included in .NET implementation. With string interning enabled for the data in stamps and images, we saw much smaller outputs. See Section 7.2 for our results comparing the size of serialized outputs in our final protocol buffer based serializer implementation. See Appendix A for a more detailed description of this process and of the rest of the CLP protocol buffers implementation.

4.4 Use of Protocol Buffer Based Serialization

Currently, our new serialization method is used on all student page submissions, and in the networked saving of pages and page histories. During the development process, we have retained local saving with BinaryFormatter as a backup. For consistency with the locally saved versions, we decided to use BinaryFormatter serialization for database saves as well. Thus, pages sent to the server or teacher are serialized using protocol buffers, and pages saved locally or in the database are serialized with the BinaryFormatter. In the future, protocol buffers could conceivably be the only serialization method used. This consolidation would reduce the size of the database; however, the long term stability of protocol buffers in CLP would have to be evaluated first.

Chapter 5

Database

With methods in place for transmitting and effectively serializing student work (Chapters 3 and 4 respectively), we now look at storage. In this section we discuss the design and use of a non-relational database for the storage of student work.

5.1 Motivation

CLP notebook pages contain a diverse array of page objects, and associated data and new types are added as the project progresses. This year, for example, this year, handwriting and shape interpretation regions were added to CLP [18]. CLP needs a way to deal with this variability—it shouldn't require an extensive engineering effort to store and access pages containing new types of parameters or objects. The data storage system also must be easily searchable, helping teachers and researchers access and organize sets of student work. Finally, adding or accessing data must be relatively quick. It should take significantly less time to access data than to send it across the network.

5.2 Data Storage Options

In the Previous Work Section, we discussed the infeasibility of using a relational database to store student work [9]. In relational databases such as MySQL, it is diffi-

cult and costly to frequently change data schemas. Furthermore, relational databases don't store value-pair data particularly well [4]. For example, we would like to store page topic, descriptions such as Math,Fractions,Word Problem,Uses Stamps, etc. Each page may have a list of descriptors that is a variable length, which is hard to define with a relational schema.

For this thesis we also considered using a file system. Mechanisms for dealing with new datatype are already built into CLP for saving to a file system. Previously and now, notebooks have been saved locally on tablet machines. As discussed in Section 3.3 however, saving on a page granularity better facilitates teachers and researchers selecting various sets of student work. Furthermore, file system modification is required to attach arbitrary numbers and types of identifiers, such as the list of page topics, to items, and there is no special interface for complex queries based on these.

For its data storage needs, CLP now uses MongoDB, an open source non-relational database with an active development community. MongoDB is used in production on many sites and is also compatible with .NET [3]. MongoDB has replication management built in, to limit downtime in the event of failures and to prevent data loss. For future use, the system is also easily scaleable. Most importantly, MongoDB stores information in documents that are defined on the fly, in key-value pairs. Data of arbitrary length, such as ink strokes, can be stored, as well as images or other file types. Furthermore, MongoDB has built-in functions for quickly creating a query based on any combination of key value pairs.

5.3 Use of MongoDB in CLP

5.3.1 Background

In MongoDB, all data is stored in what are called BSON documents, which are basically collections of key-value pairs. An entry for a saved page is shown in Figure 5-1. Documents are organized into collections, which are somewhat analogous to tables

in relational databases like MySQL. Collections, however, do not impose a schema across all member documents. While each entry in a relational database table has the same number and type of fields, the keys in MongoDB can vary across documents. This flexibility allows us to add additional key-value pairs without invalidating older data. Currently, our database contains three collections, for the three types of work saved: `SubmittedPages`, `SavedPages`, and `PageHistories`.

Figure 5-1: **A page of student work stored in a BSON document.** Additional key-value pairs can be added on the fly, and the set of keys need not be the same for all documents.

5.3.2 Saving and Querying

Notebook pages and page histories are saved once they are received by the server machine. First the messages are deserialized using protocol buffers. Next, any information on which queries are run, such as page topics, is copied. Any field that will later be used in queries, such as `PageTopics`, must be added separately as a key-value pair. Then, the page is serialized using BinaryFormatters. Finally, for pages submitted to the teacher, a new BSON documents is populated and saved to the database. As previously mentioned, CLP networked saving operates on a page granularity, so one page or page history is saved in one BSON document. For page or history saves, an existing BSON document is updated with the new `PageContent` and `SaveDate` or

if this is the first save, a new document is created.

Queries can be performed on any set of keys stored in the database. Performing a query for a key only present in some BSON documents ignores documents lacking the key, making the structure flexible to change. Returned documents must be deserialized to be viewed in the CLP program. See Chapter 6 for more information about viewing student work queried from the database. See Appendix B for technical documentation regarding the use of MongoDB in .NET environments.

Chapter 6

Data Viewing System

To complete our system, we needed a way to access student work that has been sent to the server and saved in the database. In this section we discuss a basic data viewing user interface.

6.1 Motivation

Having all student work stored in one place is only useful if there is a simple means to query and access it. Entries in MongoDB can be queried via command line, but this is not a user friendly option for those not familiar with the program. Furthermore, the notebook pages are stored in serialized form in the database and can only be viewed after being deserialized in CLP. Additionally, teachers and researchers often want to look at specific collections of student work at once, without having to shuffle through many different files. To access student work and demonstrate the flexibility of our data storage system, we created a basic data viewing interface.

6.2 Implementation

Our data viewing system is a UI on the server machine, reachable by clicking on a tab at the top of the window. At the top of this interface are text boxes where the user can specify the query to be run in order to retrieve particular notebook pages

from the database. Users do not need to know how queries are set up in MongoDB; instead, they just list the students, dates and page topics that they are interested in seeing. The user interface provides tips of how to enter these parameters. See Figure 6-1 for a screen shot of the parameter specification ribbon.

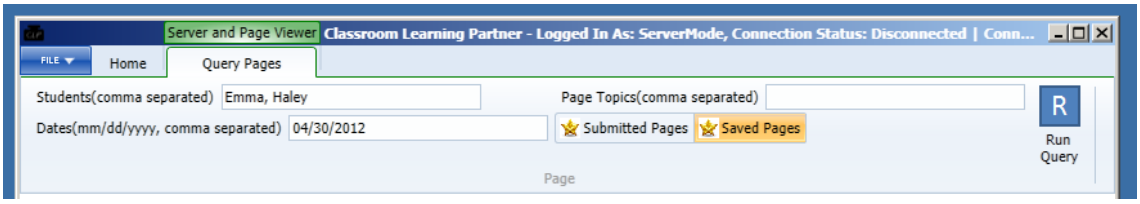

Figure 6-1: **Interface for specifying a query in the data viewer.** Users can list students, dates, and pages topics. They also have the option to query saved pages or submissions.

After clicking the Run Query button, the appropriate pages are selected from the database and deserialized. The pages of the first student specified show up in the left hand column of pages. In Figure 6-2, Emma's pages show up in the left hand column. This pages then can be selected and viewed.

Work from the other students listed in the query is viewable in a second vertical column, with each students' name on the top of the page. This interface is similar to the one already used for submissions on the teacher machine, making it easier for teachers and educators to understand and decreasing the learning curve in using it. Figure 6-3 shows additional student work in the second column.

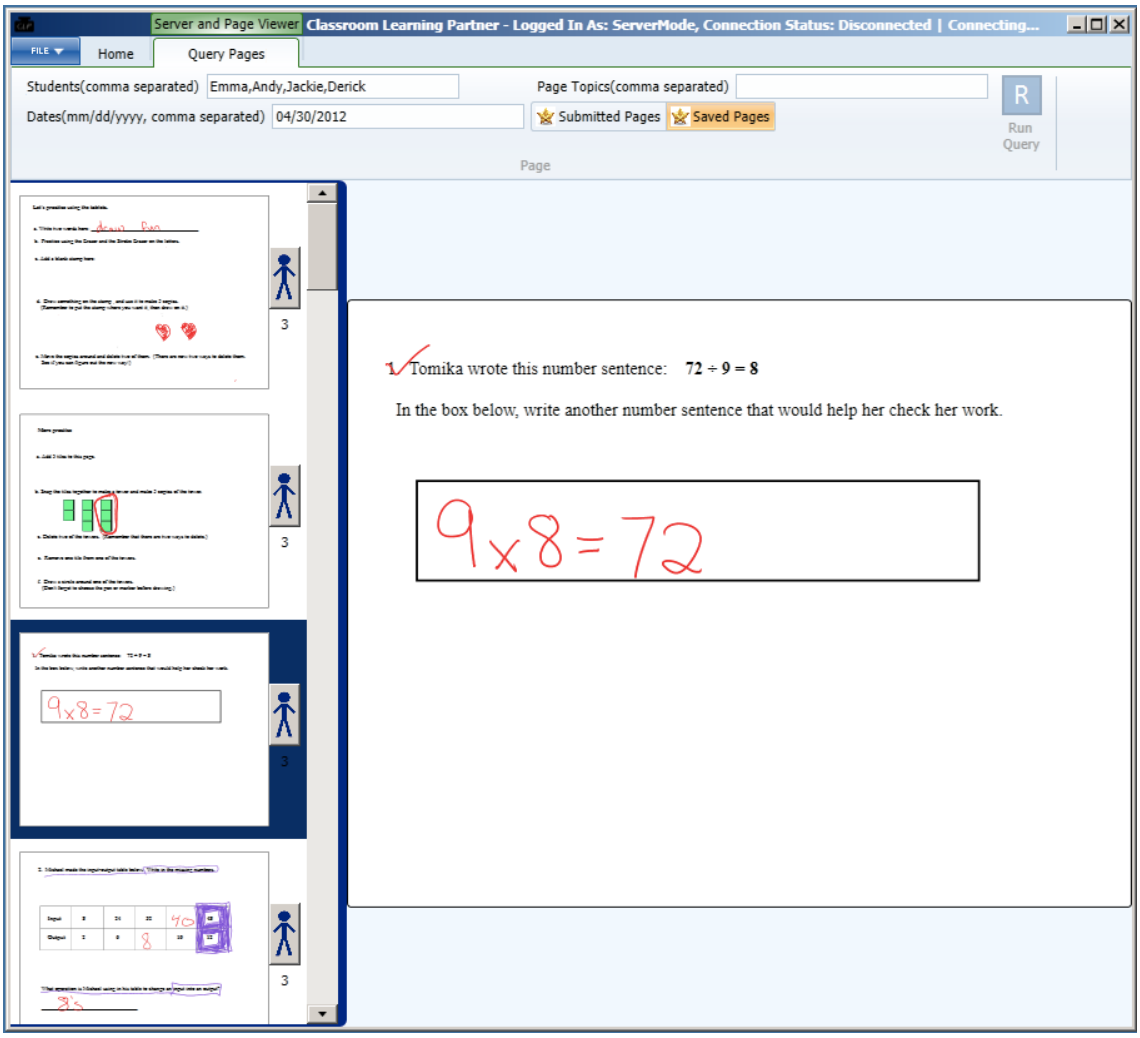

Figure 6-2: **Results of the query for pages.** Work from the first student listed, in this case Emma, are show in the left hand column. Pages can be clicked on to view at full size.

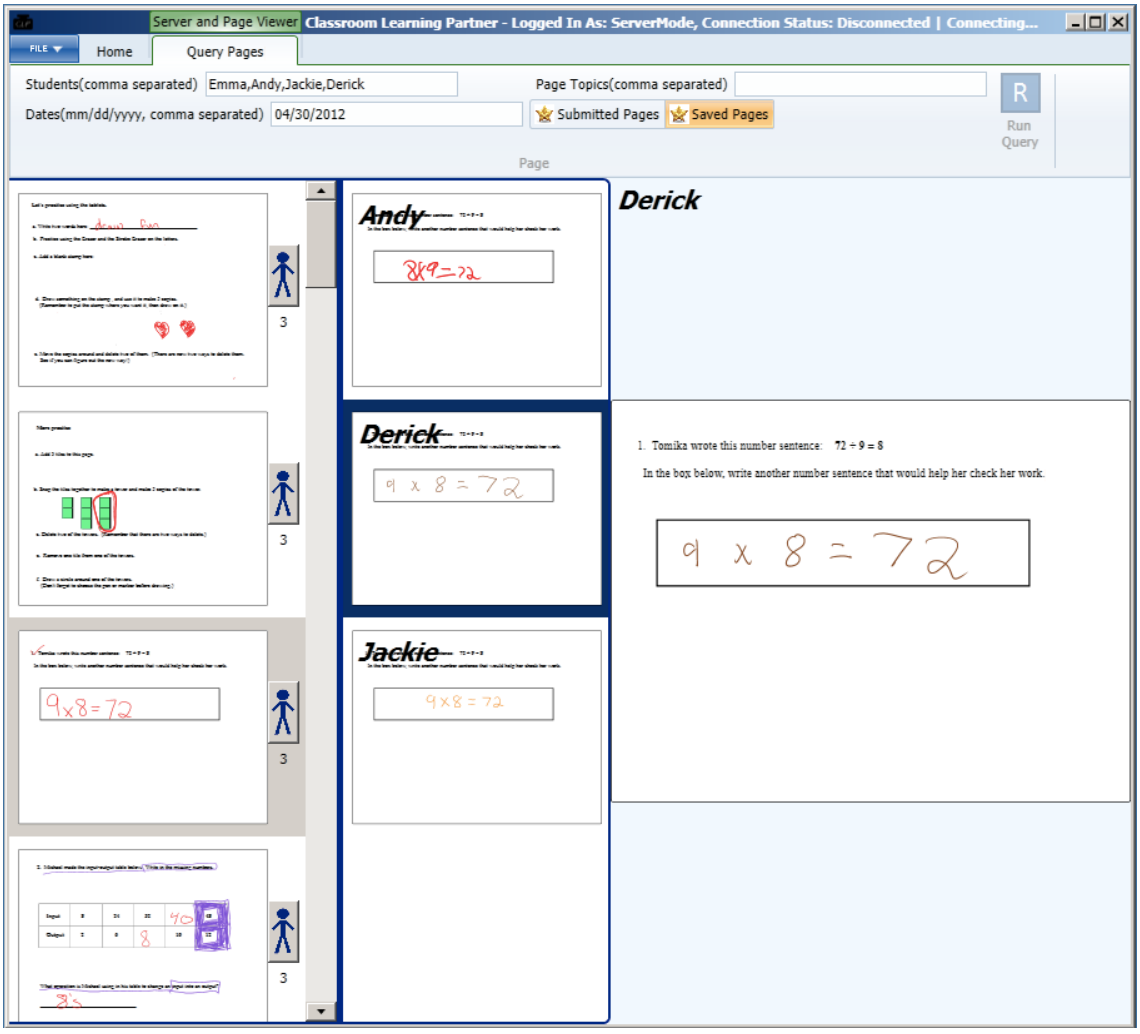

Figure 6-3: **Viewing additional student work.** After the first student, work from additional students is displayed like submissions, with the students' name at the top.

Chapter 7

Evaluation

We evaluated our system using both quantitative measurements and observation. We tested the entire CLP software system in elementary school classrooms various times over the past year. Final evaluations took place in one fourth grade classroom in Waltham, MA over the course of a week in early May. During this week, the teacher led students in one math lesson a day using the tablets.

7.1 Networked Saving

Network saving procedures were successful in the classroom. Researchers saw entries being added to the database as students saved or submitted work. Saving histories, done after class to prevent potential network slowdowns, was also successful, with histories being received in the database after being sent wirelessly to the server machine. To further investigate our network methods, we looked more closely at the time it takes for the student machine to prepare and send pages to the server. Currently, saving locally to the hard drive is a blocking task; students cannot view or edit their work until saving finishes. To be effective, network saving should take no more time than saving locally. Students frequently save their work and excess delay would be distracting in the classroom. Figure 7-1 compares the time it takes to save a full notebook locally with the time it takes to prepare seven pages and their histories for network saving.

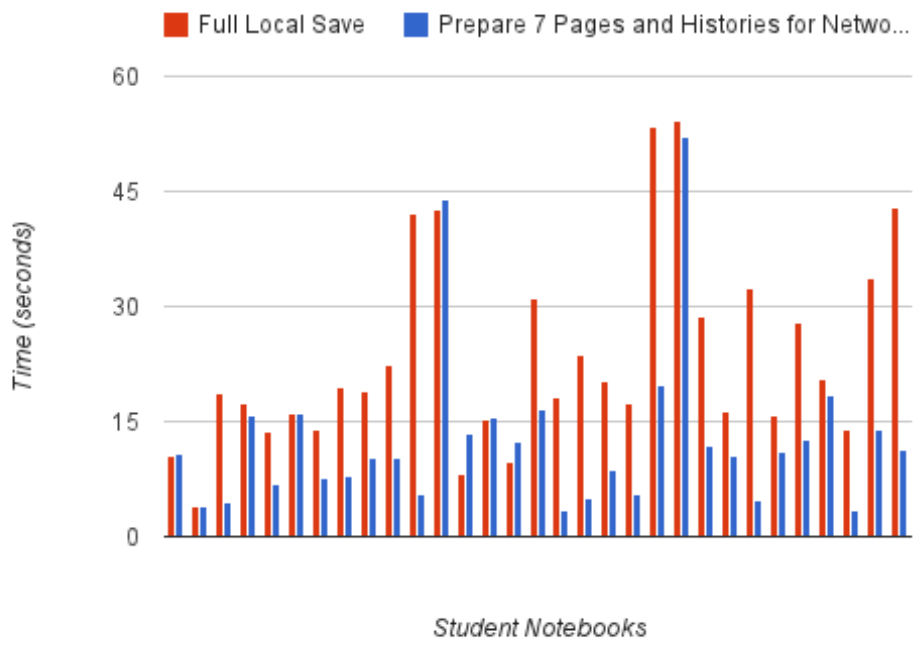

Figure 7-1: **A comparison of the time it takes to save the whole notebook locally with the time it takes to prepare seven pages and their histories for a networked save.** Experiment was run using April 30th and May 1st student notebooks. The preparation for network saving included using the improved protocol buffer based serialization technique on each page and history individually.

Figure 7-1 results show that for all but three notebooks used in the test local saving took longer than preparing for networked saves on seven pages. This result makes sense, given that all notebooks tested were longer that seven pages, but also suggests a great strength of our system. Networked saving only saves pages that have been changed, preventing the need to serialize every page in the notebook. In practice this will mean students will experience less of a delay when saving. This experiment shows that student machine preparation for networked saving tends to be comparable or quicker when dealing with seven pages. The results would be even stronger when only one page has changed, and students are often told to save before moving to another page to prevent any work from being lost.

7.2 Serialization with Protocol Buffers

To evaluate our new method of serialization, we look at the size of serialized outputs for both pages and histories as well as the speed of serialization. We found that serialization with protocol buffers greatly reduces the size of pages and page histories and can be performed as fast if not faster than BinaryFormatter serialization.

7.2.1 Serialization of Pages

To be considered successful, our new protocol buffer based method must produce smaller serialized outputs than the original BinaryFormatter method. We first compared the size of pages serialized by these two methods on March 21st. The results are shown in Figure 7-2. Each page showed a reduction in size when serialized with our new method, ranging from 72% of the BinaryFormatter size to only 2% of the size for one output. Over all pages, our serialization method showed a 50% reduction in size.

We observed sizes of serialized outputs again at the end of April to verify that other additions did not interfere with our previous work. Results from this test are show in Figure 7-3. In this experiment, we saw size gains ranging from 79% to 24%, and again saw an average decrease in size of 50% over all pages.

7.2.2 Serialization of Histories

We also must consider the size of page histories serialized with our improved method. Page histories are frequently very large, often over 1000kB per page, and were important in motivating implementation of a new serialization method. In addition to serialization, work this year also went into further compressing histories using the process of segmentation. Segmentation is the process of decreasing the number of observations needed to describe movement of object such as a stamp on the page. See [5] for more information about history segmentation. The output size of page histories serialized with the BinaryFormatter, our new protocol buffer based method, and the new method with history segmentation is shown in Figure 7-4. Again, serial-

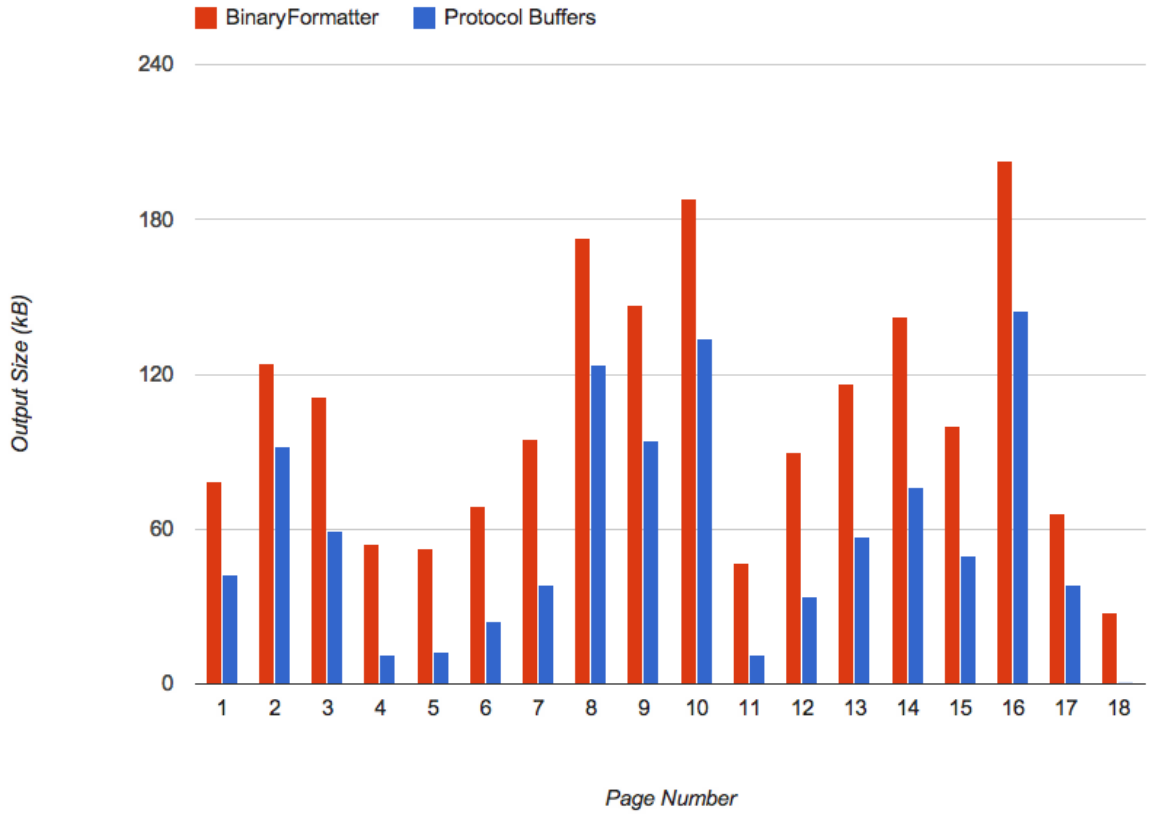

Figure 7-2: **A comparison of the output size of pages serialized by BinaryFormatter and Protocol Buffers** The results are from pages of students' March 21st notebooks. Each result is the average output size for that page over all students' work.

ization using the new protocol buffer based method made outputs drastically smaller. The new method led to serialized histories that are on average 30% the size of BinaryFormatter output. Including the history segmentation process shaved off another 10% of the size, for serialized output that average 20% of those serialized with the older BinaryFormatter method.

7.2.3 Serialization Speed

Our new serialization method has been shown to be effective in greatly reducing the size of serialized pages and page histories. It also, however, needs to be relatively fast to be successfully used in the classroom. Serialization is run every time a student saves

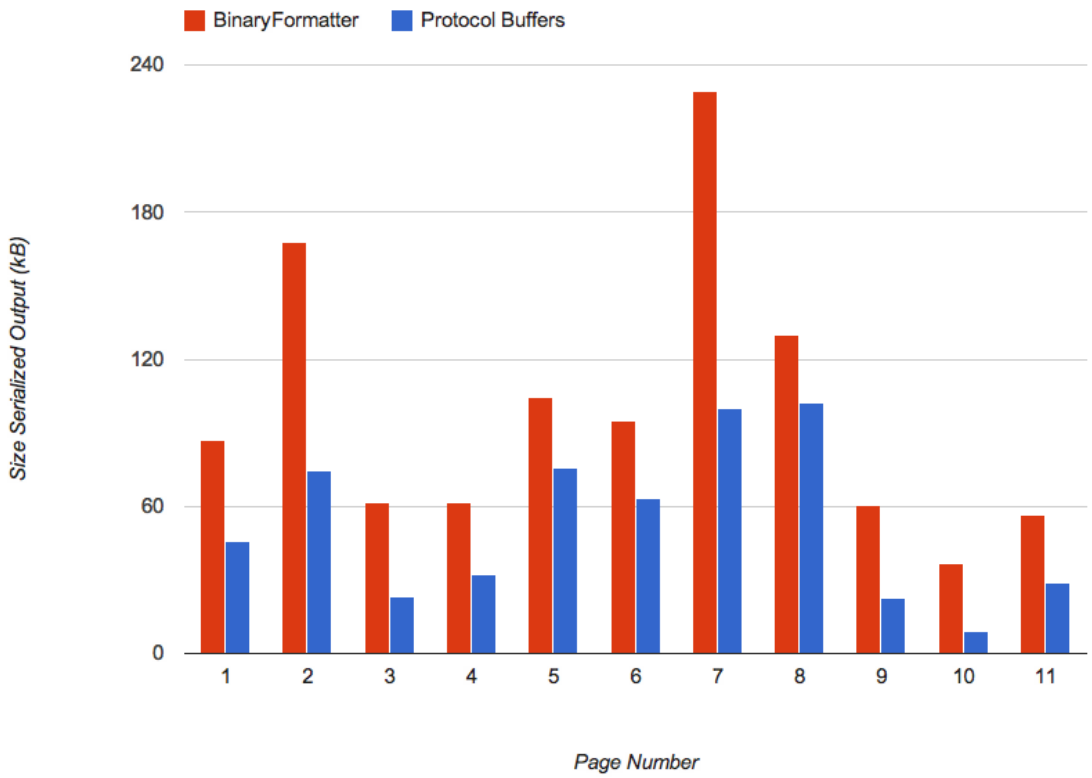

Figure 7-3: **A repeated comparison of the output size of pages serialized by BinaryFormatter and Protocol Buffers** The results are from pages of students' April 30th notebooks. Each result is the average output size for that page over all students' work. One page was omitted to prevent the graph from being skewed and unreadable. This page was serialized by BinaryFormatter to 1367kB and by Protocol Buffers to 358kB.

or submits work and therefore cannot delay his or her machine for long. We compared the time it takes to serialize pages using the older BinaryFormatter method and our new method and graphed them based on the size of the BinaryFormatter output. Results are shown in Figure 7-5. We graphed time vs size because in general it takes longer to serialize larger inputs. For all sized samples, our new serialization method took approximately the same amount of time or less. In the serialization of large inputs, our new method was at times a second faster than BinaryFormatters.

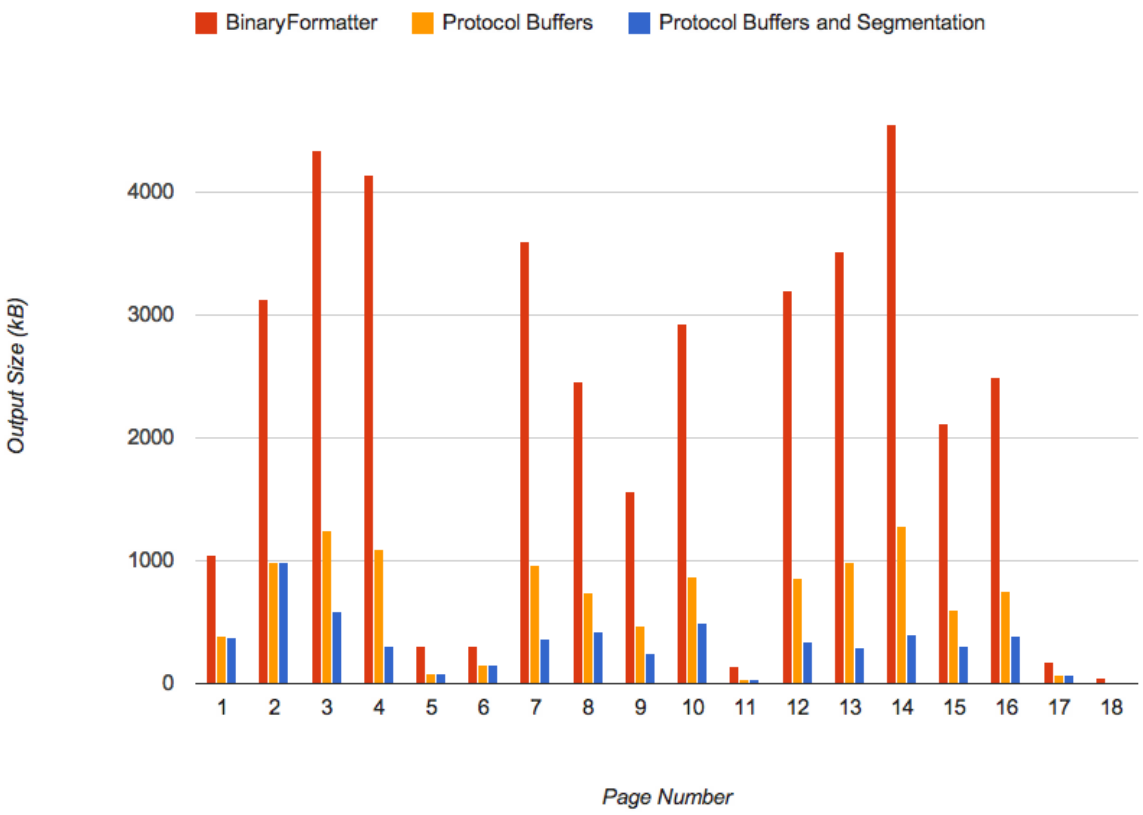

Figure 7-4: **A comparison of the output size of histories serialized by BinaryFormatter and Protocol Buffers** The results are from pages of students' March 21st notebooks. Each result is the average of output size for that page's history over all students.

7.3 Database and Data Viewer

To evaluate the database, we look at an actual use case. During testing in March, a student lost his work after the program exited without saving. This student had been submitting work to the teacher though. Using the database copies of his submitted work, researchers recreated his notebook, retrieving much of his lost work. The data viewing system provides an interface for completing tasks such as this. The system was used by researchers to organize and view student work from the May class trials.

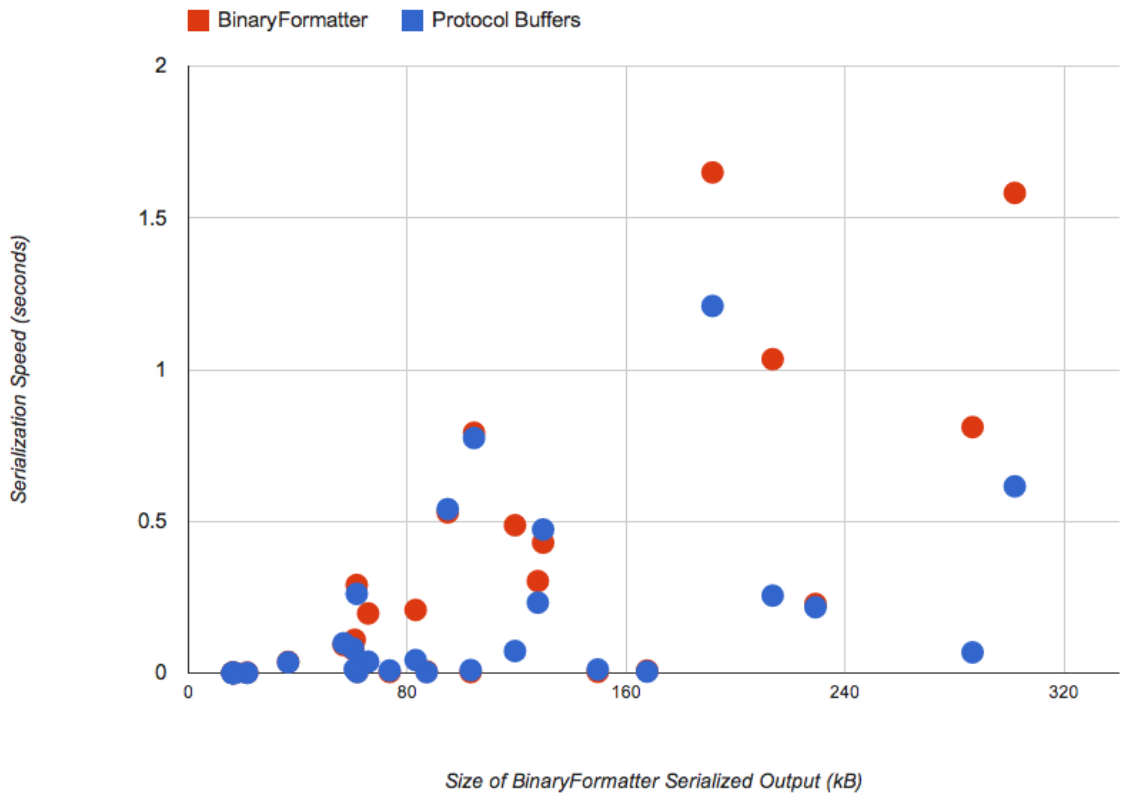

Figure 7-5: **A comparison of the speed of BinaryFormatter and Protocol Buffer serialization, based on output size** The results are from April 30th student work.

Chapter 8

Conclusion

8.1 Future Work

In terms of system architecture and classroom features, CLP has made great progress this past year. Our work has improved system performance in the classroom and expanded CLP's educational benefits. It also lays the foundation for further improvements. We end this thesis by suggesting the following future work on CLP data management.

8.1.1 Routing Mesh Network

Currently, CLP uses a flooding mesh network to wirelessly connect tablet machines in the classroom. Flooding mesh networks are easy to set up, but can needlessly waste network bandwidth. Most messages currently sent in CLP are intended for only one or two other machines. Propagating that message to all machines creates unnecessary network congestion. This thesis made promising gains on this issue by updating on a page granularity for database saves and by using an improved serialization method. Even greater benefits, however, will come from changing to a routing network architecture.

A routing mesh network requires machines to maintain knowledge of the network topology. That overhead, however, allows machines to route messages on effective

paths to the target, instead of flooding the whole network. This architecture will reduce overall network congestion, improving performance in saving and submitting pages and page histories. In addition such networks often include systems for acknowledging messages, creating a confirmation that a message made it to its intended destination. A routing network also will enable the creation of additional communication features in CLP. For instance, with more available bandwidth, the system could wirelessly distribute pages to students (see Section 8.1.2), access student submissions on demand (see Section 8.1.3) or enable students to send work to each other, without stressing the network. Also, extra network bandwidth supports the inclusion and sending of additional large page objects, such as student-recorded video. The adoption of a routing network will improve performance in network sending currently used in CLP and allow the implementation of new classroom communication methods.

8.1.2 Networked Page Distribution

This year we successfully implemented a system that saves all student work. Based on that progress, we should next work on distributing notebook pages to students wirelessly. Currently, new notebooks are distributed to each machine manually using USB flash drives before class starts. This process is tedious and requires the lesson to be set in advance. Additionally, there is no way for students to retrieve their old notebooks from the server machine. In order for students to open previous notebooks, their notebooks must all reside on the tablet they use in class. Methods that allow students to access their work from the database would greatly decrease setup time and improve lesson flexibility.

Creating this functionality will include both network procedures and UI additions. Teachers will need a way to select a certain set of pages to be distributed for a lesson. This step could be done ahead of time or in class. By default, the same set of pages should go to all students, but it would be useful for the teacher to specify additional pages for those who get through their work quickly. When students log in, their machines could automatically request teacher specified pages. Students, after

choosing their name from a list, should be able to choose from a list of previously authored notebooks (if allowed by the teacher) or the newly downloaded notebook. This architecture will allow teachers to distribute work on the fly and to lead class in a review of older work.

8.1.3 Access Submissions on Demand

Currently, pages submitted by the student in class are received by the teacher, projector and server machines. While the server machine stores the work in the database, the teacher and projector machines incorporate the submission into the currently open notebook. The data in open notebooks is stored in the tablets' memory. Large numbers of submissions use a lot of memory on the teacher and projector machines, making them slow or unresponsive. Because of this issue we chose to not include large page histories with submissions, but notebooks of just plain submitted pages are large enough to still cause lag. These notebooks also take a long time to open again later.

Teacher and projector machines need a method of accessing submitted pages on demand, instead of trying to store them all in a notebook. We present two possible ways to do this. First, small database instances could run on the teacher and projector machine acting as caches for submitted pages. When submissions are received, they are saved to the caching database, instead of incorporated into the notebook. The pages are queried for and loaded into the notebook only when the teacher clicks on the submissions button for that specific page. The submissions remain in memory for a few minutes after the teacher moves to a different page and are then flushed out. These submissions do remain in the database cache though, if the teacher views that page again. These caches are only assumed to be valid during class. If the teacher notebook is opened again later, the submissions to populate the caching database are queried from the main server database.

An alternative plan is to only store submissions in the server database. In this

case querying for submissions for a specific page is no longer a local action. The teacher machine sends the query over the network and waits for the server to send the message containing the page. Since this method supplies submissions over the network, it will take longer than the first method. It would also be a particularly poor choice to use with our current flooding network, since submissions would be resent to all machines in the network each time the teacher changes pages. Whether the time difference between this method and the caching method when used with a routing network will be noticeable will need to be determined experimentally. This method does have the advantage of using the same procedure whether used in class or outside of class.

8.1.4 Data Viewer Improvement

In this thesis we describe a basic interface for querying and viewing collections of student work. This simple system could be extended in many ways to further benefit teachers and researchers.

Viewing Page Histories

Currently, the data viewer can only query pages from the database. In the future, users should also be able to play back the histories associated with a set of pages. This functionality will require including the history player controls, currently only in the instructor instance, in the server instance so that page histories can be queried. Given a page, querying the database for the associated history is not complicated. See Appendix B for the code to perform such an operation. Since page histories are large, however, it is unwise to grab all page histories for a query at once. For larger collections of student work, this method will likely slow the program down greatly. Instead, similar to the technique discussed in Section 8.1.3, histories should be accessed on demand. The history for a specific page should only be queried and added to the notebook when that page is viewed. Furthermore, that history should not remain in the notebook indefinitely, but instead persist for some time period and

then be discarded. It can always be queried from the database again later.

Updating Pages

Researchers have expressed interesting in being able to update the records of pages stored in the database. For example, they might want to update the page topics associated with a specific page or set of pages in order to facilitate locating particular sets of student pages. This feature could be built into the data viewing system. After a researcher queries a set of pages, he or she could click a button attached to a page to update some field related to that page. After clicking submit, such a change would be recorded in the database. Additionally, the interface for specifying a query could be extended to accommodate batch changes. In this regard it would be similar to the find and replace window in text editors: The user would specify a query, and then selected fields would change in all resulting pages. Limitations must be set of which changes are allowed though. For instance, changing the student username on pages would be nonsensical and might lead to data lead to data having the same supposedly unique identifiers.

8.2 Contributions

We created a centralized data storage and viewing system for use in CLP. This system integrated into the previous CLP classroom workflow and enabled all student work to be saved. This work can later be explored in a new data viewing interface. Student work can be queried and grouped in diverse ways that were not available in earlier versions. We also introduced a new serialization method that aids both our work and the performance of the whole system. This method makes pages sent over the network on average 50% smaller and reduces histories to 30% of their raw size. Our system allows data in CLP to be easily stored and tracked, allowing researchers and teachers to more easily monitor student progress and assess educational outcomes.

Appendix A

Detailed Implementation of Protocol Buffers in CLP

We use protobuf-net, a .NET implementation of protocol buffers, in order to harness the power of protocol buffers in CLP [19]. Protobuf-net includes all the standard features of protocol buffers along with additional features needed for common .NET use cases. It integrates with existing code, preventing the need to maintain separate files. The use of protobuf-net helped us incorporate protocol buffers into CLP in a timely manner.

According to the documentation, using protobuf-net only requires the addition of annotations to classes that will be serialized. We found this process, described in Section A.1, slightly more time consuming than advertised. We discovered that some included structures in .NET cannot be serialized by default. Our solution to this is presented in Section A.2. In addition, serialization with basic protobuf-net annotations produced only minor improvements over BinaryFormatter serialization. This discrepancy was caused by the default way protobuf-net serializes duplicated objects. To fix this problem and improve performance, we wrote our own pre and post serialization routines, discussed in Section A.3.

A.1 Describing Classes and Hierarchy

Using protocol buffers requires specifying the types and relationships between objects to be serialized. In protobuf-net, this can be done in two ways. We initially placed annotations in each class involved in forming a notebook page. This includes one declaration at the beginning of the class and then assigning a unique integer to each parameter in the class that will be serialized. While this method makes sense for smaller programs, we found it to be too scattered for our needs. Describing hierarchy in individual classes means that parent classes must know what children inherit/implement their methods, which reduced the usefulness of our object oriented design. Furthermore, tweaking various setting to improve serialization performance required shuffling threw many files.

In our final implementation we use the second method, which is describing the entire class hierarchy in one file called a type model. This setup makes it easier to tweak parameters and maintain correct annotation as code changes. Class inheritance models are described in one central file, instead of making each individual class aware of potential children. Multiple type models can be defined, allowing for the serialization of different class hierarchies. This fact will give us flexibility in the future. Type models are not clearly described in protobuf-net documentation, but are the more robust method of specifying of the class hierarchy.

While type models prevented the need to put annotations in individual classes, protobuf-net did require other changes within classes. All types serialized/deserialized using protobuf-net are required to have a parameterless constructor. This constructor is used to create the object during deserialization. We added such constructors to classes that did not already have one, paying close attention to what procedures taken in the other constructors need to be duplicated. For example, it was not necessary to initialize a parameter that has been serialized, since those will be filled by protobuf-net after the object is created. Methods run at object creation, however, still need to occur.

Protobuf-net also requires public getters and setters defined for all serialized parameters across classes. This change was not difficult to make, but needed to be done with caution. Private getters and setters help enforce various conventions. For example, a private setter makes a parameter immutable by other objects. Without this extra layer of enforcement, we have had to be careful to maintain our specified object interaction model.

A.2 Working With Non-Supported Structures

Protobuf-net can serialize most standard .NET structures, such as int and DateTime; however, it lacks support for many structures outside the System namespace. The Point structure, used in `Position` to describe the location of page objects, is one such example. Protobuf-net does not support it by default, and the standard assembly cannot be easily annotated. To work around this, we created two additional parameters, `XPositoin` and `YPosition`, of type double. We wrote a method called before serialization that extracts the x and y coordinates of `Positon` and places them into the appropriate new parameters. `XPostion` and `YPosition` are serialized instead of `Positon`; in fact, `Positon` is not included in the type model. After deserialization, another method is called that initializes `Positon` with the newly deserialized values from `XPositoin` and `YPosition`. This setup can be used for other .NET objects and was the inspiration of our work to use object references the next Section.

A.3 Maintaining References

Unfortunately, the basic implementation of protocol buffer serialization did not provide effective for CLP. Most pages serialized with the basic method were only a few kB smaller than those serialized with protocol buffers, and some were significantly larger. As discussed in Section 4.3, this was due to the lack of string interning by default in protocol buffers. While BinaryFormatter automatically uses references for repeated strings of data, such as those arising from multiple copies of a stamp, protocol buffers

includes each copy, making a larger output.

Protobuf-net includes string interning that can optionally be turned on, but only for lists of strings. Unfortunately in the CLP data architecture, the repeated strings from stamps or images are stored in a hierarchy of objects. Furthermore, in experimenting with protobuf-net, we were unable to use string interning for more than one list at any point in the object hierarchy. We suspect this is a bug in protobuf-net, but created a method flexible to this and to the inclusion of other data in the future anyway.

To utilize string interning, we created a new list of strings, `serializeStringsList` for serialization in CLPPage. Pre-serialization, we look at the type of all objects on a page. For objects other than pressed or regular stamps and images, an empty string is added to `serializeStringsList`. For stamps and images, stroke/string data is copied out of the object and placed in the list instead. Stamps often consist of multiple strokes, which is serialized as multiple string. To keep this collection of strings together, we first add the number of strings to `serializeStringsList` and then each string. After copying, the properties that hold stroke and image data in the page object are set to an empty string. We still need to serialize the object, because it holds additional meta-data such as position; however, now the stroke/image data, if repeated, will only be in the serialized output once. This process is reversed after serialization and deserialization, returning the stamp and image data to the correct page object.

Appendix B

MongoDB in CLP

B.1 Starting MongoDB Instance

Starting MongoDB is simple. After downloading the files, insure that the files `\data\db` are present. Then open a terminal window in the `MongoDB\bin` folder and type

```
.\mongod
```

This starts and instance of the MongoDB database, using the `\data\db` to store data. The storage fold can be changed at start up using `-dbpath`. For example:

```
.\mongod -dbpath \ClassroomTrials\Waltham
```

will store data in the `\ClassroomTrials\Waltham` folder. After completing this step, the CLP server instance can connect to the database.

Another helpful command is `.\mongodump`, which backs up a database to a folder called `dump`. To restore a database from the `dump` folder, use `.\mongorestore`. `mongorestore` only inserts data. Entries with the same `_id` will not be rewritten.

B.2 Accessing MongoDB in CLP

To access the our MongoDB database in CLP we use the official .NET driver, written by 10gen, the makers of MongoDB. This driver provides methods and classes that integrate seamlessly in the CLP code base.

B.2.1 Accessing a Collection

First, create a connection to the database instance. This only needs to be done once in the program:

```
string ConnectionString = "mongodb://localhost/?connect=direct;slaveok=true";
DatabaseServer = MongoServer.Create(ConnectionString);
MongoDatabase nb = App.DatabaseServer.GetDatabase("Notebooks");
```

`MongoDatabase` can then be used whenever the program queries the database.

Then select the appropriate collection. This could be `Pages` (submitted pages), `SavedPages`, or `SavedHistories`. All queries are run at the collection level in MongoDB.

```
historyCollection = nb.GetCollection<BsonDocument>("SavedHistories");
```

B.2.2 Run a Query

There are many ways to specify a query using the .NET driver. Currently in CLP we use the Query Builder to specify a key-value pair that our query should find. For example, to find all work by Lisa we would write:

```
var query = Query.EQ("User", "Lisa");
```

Queries made with the query builder can be strung together to make more complex queries. For example to find all work by Lisa that was done in a certain set of dates, specified by the `List<DateTime> dateList`, use:

```
var query = Query.And(Query.EQ("User", "Lisa"),
Query.In("SaveDate", BsonArray.Create(dateList)));
```

B.2.3 Insert an Entry

After a query is created, it is run on the selected collection, returning a `MongoCursor`. We can iterate through the results using the `MongoCursor`. The order of objects returned by the query can also be specified. In the example below, we run the previously

specified query for Lisa's work in the page history collection, sorting by page number.

```
 MongoCursor cursor = historyCollection.Find(query)
.SetSortOrder(SortBy.Ascending("PageNumber"));
foreach (BsonDocument history in cursor){
// Do something with histories
}
```

Refer to the MongoDB website, for further documentation [3].

Bibliography

[1] Protocol Buffers-Google Developers. https://developers.google.com/protocol-buffers/.

[2] Human Factors Researchers Show That Tablet PCs Belong In Classrooms. http://www.sciencedaily.com/, March 2008.

[3] 10gen. MongoDB. http://www.mongodb.org/.

[4] Karwin B. *SQL Antipatterns: Avoiding the Pitfalls of Database Programming*. The Pragmatic Bookshelf, Raleigh, 2010.

[5] DeRosa C. Dynamic Multimedia in Classroom Learning Partner. Master's thesis, MIT, 2012.

[6] Duncan D. *Clickers in the Classroom*. Addison Wesley, San Francisco, 2005.

[7] Held G. *Wireless Mesh Networks*. Auerbach Publications, Boca Raton, 2005.

[8] Heines J. and Liang W. Combining, Storing, and Sharing Digital Ink. *Proceedings of SIGCSE*, 2007.

[9] Mueller J. Undergraduate Advanced Project: Data Storage and Viewing for Classroom Learning Partner. Technical report, MIT, 2011.

[10] Bransford J.D., Brown A.L., and Cocking R.R., editors. *How People Learn: Brain, Mind, Experience, and School*. National Academy Press, Washington, D.C, 2000.

[11] Koile K. and Rubin A. INK-12: Teaching and Learning Using Interactive Ink Inscriptions in K-12. NSF Discovery Research K-12, NSF 08-602, Sept. 2010 –Sept. 2014.

[12] Koile K., Reider D., and Rubin A. Ink-12: A pen-based wireless interaction system for k-12. In Berque D. Reed, R. and J. Gray, editors, *The Impact of Tablet PCs and Pen-based Technology on Education: Evidence and Outcomes.* Purdue University Press, West Lafayetee, IN, 2010.

[13] Koile K. and Singer D. Improving Learning in CS1 via Tablet-PC-Based In-Class Assessment. In *Proceedings of ICER 2006*, Canterbury, UK, 2006. University of Kent.

[14] Koile K. and Singer D. Assessing the impact of a tablet-pc-based classroom interaction system. In R. Reed and D. Berque, editors, *The Impact of Pen-based Technology on Education: Evidence and Outcomes.* Purdue University Press, West Lafayetee, IN, 2008.

[15] Koile K., Chevalier K., Low C., Pal S., Rogal A., Singer D., Sorensen J., Tay K.S., and Wu K. Supporting Pen-Based Classroom Interaction: New Findings and Functionality for Classroom Learning Partner. In *In Proceedings of First International Workshop on Pen-Based Learning Technologies*, May 24-26, 2007.

[16] Koile K., Chevalier K., Rbeiz M., Rogal A., Singer D., Sorensen J., Smith A., Tay K.S., and Wu K. Supporting Feedback and Assessment of Digital Ink Answers to In-Class Exercises. (2007a). In *In Proceedings of IAAI 2007*, July, 2007.

[17] Tay K. and Koile K. Improving digital ink interpretation through expected type prediction and dynamic dispatch. In *Proceedings of the Nineteenth International Conference on Pattern Recognition (ICPR 2008)*, 2008.

[18] Von Tish K. Interpretation and Clustering of Handwritten Student Responses. Master's thesis, MIT, 2012.

[19] Gravell M. protobuf-net: Fast, portable, binary serialization for .NET. http://code.google.com/p/protobuf-net/.

[20] Rbeiz M.A. Semantic Representation of Digital Ink the Classroom Learning Partner. Master's thesis, MIT, 2006.

[21] Black P. and Wiliam D. Assessment and classroom learning. *Assessment in Education*, 5(1):71–74, 1998.

[22] GTCO CalComp Peripherals. The InterWrite PRS User's Guide. http://www.interwritelearning.com/support/pdf/PRS_Users_Guide.pdf.

[23] Anderson R., Anderson R., Simon B., Wolfman S., VanDeGrift T., and Yasuhara K. Experiences with a tablet-pc-based lecture presentation system in computer science courses.. In *Proceedings of the 35th SIGSCE Technical Symposium on Computer Science Education*, pages 56–60, Norfolk, VA, 2004. SIGSCE.

[24] Anderson R., Anderson R., Vandegrift T., Wolfman S., and Yasuhara K. Promoting interaction in large classes with computer-mediated feedback. *Computer Support for Collaborative Learning*, pages 119–123, 2003.

[25] Yost S. Using a Tablet PC to Enhance Instruction and Productivity. Retrieved October 20 2011, from the Virginia Commonwealth University, Richmond, VA, Center for Teaching Excellence Website: http://www.vcu.edu/cte/programs/instructional_technology/tablet_PC_prog/TabletPC_EnhanceInstructionProductivity.pdf.