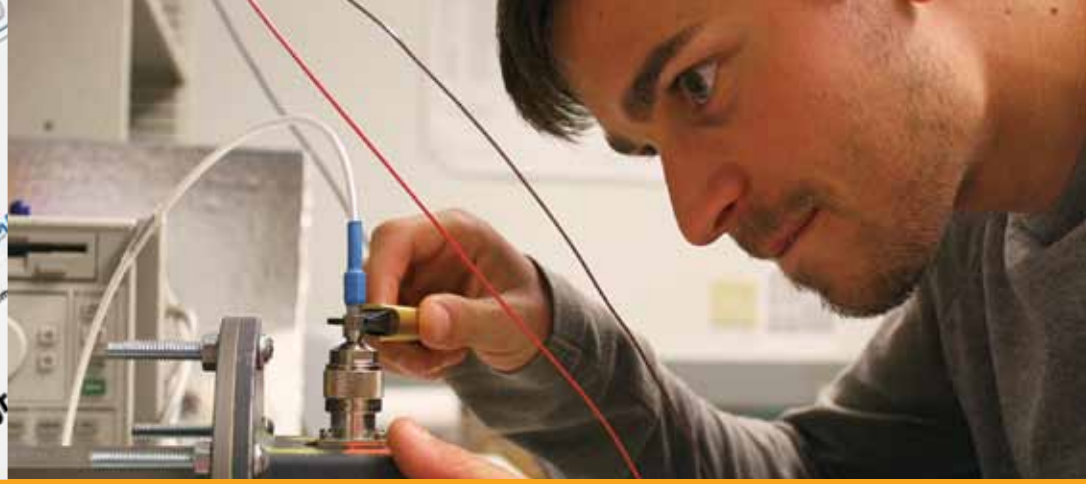
Baskin Engineering
UC SANTA CRUZ



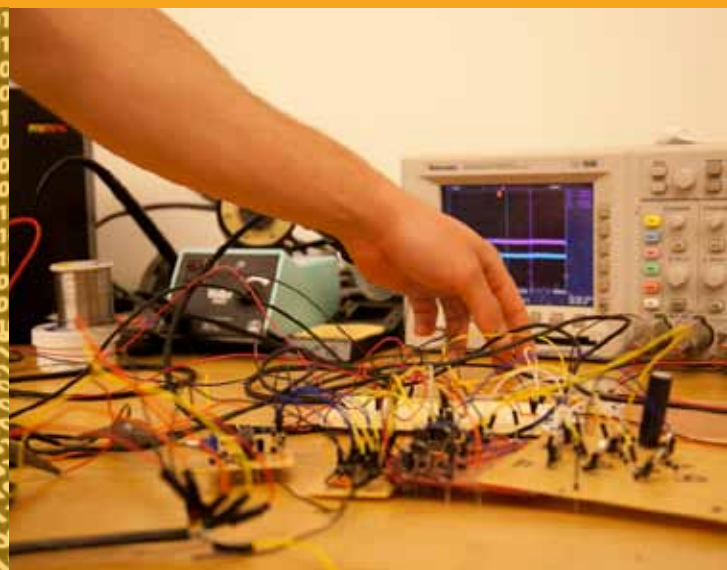
2015

CORPORATE SPONSORED

SENIOR PROJECTS

PROGRAM

PARTNERS DAY



INTRODUCTION

We are pleased to provide this booklet highlighting our fourth year of the Corporate Sponsored Senior Project Program. Also included in the booklet are a selected group of this year's Capstone projects from our student teams in Computer Engineering and Electrical Engineering working on faculty/student initiated projects. Our students have worked very hard during their time at UC Santa Cruz earning their degree and fulfilling this capstone design sequence.

Students who have participated in our Corporate Sponsored program have been provided with a unique opportunity to experience working on real-world projects that involve design, budgets, deadlines, teamwork and reviews with their corporate team mentor. They have come away with a sense of professionalism and pride in their work learned challenging skills, experienced entrepreneurship and been introduced to the many implications of intellectual property.

Throughout this academic year, the students have interacted with their teammates, some have made visits to their corporate sponsor's worksite and all have solved problems that arose along the way. The students take great pride in their completed projects and all they have accomplished during their time at UC Santa Cruz and the Baskin School of Engineering.

We also take great pride in what the students have accomplished. We are very grateful to our corporate sponsors for their willingness to sponsor this year-long program, mentor our students and provide them with challenging projects to work on.

Joseph P. Konopelski
Dean
Baskin School of Engineering

Baskin Engineering
UC SANTA CRUZ

ACKNOWLEDGEMENTS

We would like to acknowledge and thank the faculty, teaching assistants, and staff who have been so instrumental in the Corporate Sponsored Senior Project Program:

SENIOR DESIGN FACULTY

RICHARD JULLIG—Professor, Computer Science
soe.ucsc.edu/people/jullig

PAT MANTEY—Director, Corporate Sponsored Senior Project Program and Associate Dean for Industry Programs soe.ucsc.edu/people/mantey

DAVID MUNDAY—Lecturer, Computer Engineering

STEPHEN PETERSEN—Senior Lecturer, Electrical Engineering
soe.ucsc.edu/people/petersen

JOHN VESECKY—Professor, Electrical Engineering
soe.ucsc.edu/people/vesecky

TEACHING ASSISTANTS

JEFFREY BERTALOTTO—Teaching Assistant

DAVID HAHN—Teaching Assistant

GRACE LIN—Teaching Assistant

PAUL NAUD—Teaching Assistant

ETHAN PAPP—Teaching Assistant

CORPORATE SPONSORED SENIOR PROJECT PROGRAM STAFF

TIM BENSCH—Director of Corporate Development

LISA COSCARELLI—Special Agreements Officer

TIM GUSTAFSON—BSOE Technical Lead/BSOE Webmaster

ANGELINA GUTIERREZ—Program Coordinator/Executive Assistant, Dean's Office

LIV HASSETT—Associate Campus Counsel

FRANK HOWLEY—Senior Director of Corporate Development

JOSEPH KONOPELSKI—Dean, Baskin School of Engineering

MAUREEN MCLEAN—Director of Resource Planning and Management

CHRISTIAN MONNET—Development Engineer, Baskin Engineering Lab Support

SONYA NEWLYN—Special Projects Coordinator

LYNNE SHEEHAN—Network Administrator, Facilities/Machine Room

ANNA STUART—Senior Budget Analyst

BOB VITALE—Director of Laboratories and Facilities

SPONSORS

SPECIAL THANKS to our sponsors for your generous support of our Corporate Sponsored Senior Projects Program.
Your time, experience, and financial support were beneficial to our students and the success of their Senior Design Projects.

UC SANTA CRUZ
Baskin Engineering

Capstone Project

Akamai Page Performance Analysis

Gabriel de la Mora, Andrew Guttman, Adam Henry, Edward Nguyen
Raymond Colebaugh, Ian Gudger, Jason Heron, Kevin Stewart, Rita Valentine

Akamai

Abstract

Akamai Technologies is a content distribution network. They provide a network of distributed servers that deliver web pages and other content to users. Part of their offerings include a suite of features that clients can apply to their web pages that improve page load times.

In order to intelligently recommend features that would most benefit their clients, Akamai has been gathering **resource timing (RT) data**, information on how elements within a given web page load. Using this data, Our team set out to create a tool that models the different features to produce recommendations.

Approach

The project was structured into two parts: **model development** and **model implementation and verification**. For the model development, we represented the RT data as a **waterfall chart,** in order to visually convey object load times as sequential bars, similar to a Gantt chart. Using insight gathered by studying these waterfalls and statistical analysis of the data, we created models that accept RT data and look for patterns in the sequence of loading objects within a web page that suggest that the page would benefit from one or more Akamai features.

Waterfall Chart. A sample waterfall for a simple page. Bars show load times for page elements, with different colors signifying different key points in the loading process.

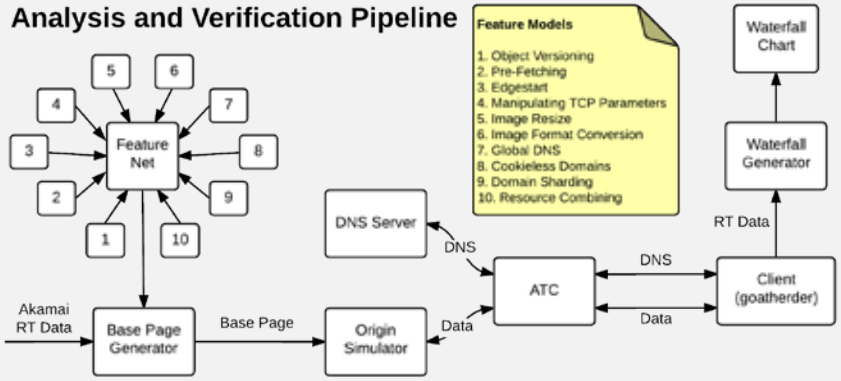
For the implementation and verification, we integrated a complete network testing framework allowing us to accurately simulate various network conditions to test the effects of applying the suggested Akamai features and estimate potential speed up.

Goat Herder is a tool that collects resource timing data from page loads in a controlled environment. The base page generator allows us to create base pages for GOATS with tweakable parameters. GOATS (Goat Occupied Akamai Team Simulator) acts as a webserver and simulates server latencies based on base page generator parameters.

Overview

A unified tool incorporates the individual feature models into a single analysis engine that produces a list of features to apply to a given page. The features each offer a unique strategy to reduce web page load time and may be suggested alone or in conjunction with other page performance features.

Object versioning - Allow frequently updated objects, normally uncacheable, to be cached by the web browser by versioning the object on the edge server.
Pre-fetching - Start fetching objects on the edge server before the the browser has requested them by parsing base page while delivering it.
Edgestart - Make use of unavoidable DNS latency by storing header on server and fetch objects in the header during wait.
Manipulating TCP Parameters - Some objects download slowly. Speed them up by increasing starting packet rate.
Domain Sharding - A web browser can only open up so many connections to one domain. Rewrite basepage to put objects on different domains so more objects can be downloaded at once.
Image Resize / Format Conversion - Resize/reformat images to avoid sending unnecessarily high quality images to small screens.
Global DNS - If the host does not have a global DNS presence but the customers are distributed globally, enabling Global DNS is recommended.
Cookieless Domains - Sending the root domain's cookies with every request can be detrimental to performance especially when combined with slow upload speed. This can be mitigated by moving as many of the requests as possible to subdomains without associated cookies.
Resource Combining - Combining several small resources into a single file can improve delivery by increasing the compression ratio and saving open connections, allowing more objects to be loaded simultaneously.



Acknowledgements

Mark Holland, Senior Manager, Akamai Technologies Inc.
Chuck Neerdaels, VP Engineering, Cloud Platforms, Akamai Technologies Inc.
Dr. Richard Jullig, UCSC - Grace Lin, UCSC

Analysis

For this project we created models (detailed in our project overview) for each feature that could potentially decrease the load time for a webpage. The team created these models based on the resource timing data provided by Akamai. Waterfall charts, similar to Gantt charts, provide a visual interface to the RT data, used for model development and to display the results of our performance analysis.

The resource timing data contains information such as time-to-load, download duration, size, etc. The data is parsed by each feature in a successive and modular fashion, identifying attributes of pages that could be optimized. After identifying these attributes, the system creates an after waterfall image, with the new optimizations applied to the objects on the page. This new waterfall image highlights the attributes of the page that are valid candidates for optimization and provides estimated performance speed ups for each of them.

Results

The end result of the project is a tool for analyzing resource timing data of Akamai's customers. The tool creates before and after waterfall images to provide visual comparisons of the resource timing data before and after using our tool to determine which of the Akamai features (see overview) would lead to a speedup for the customer.

Conclusion

The analysis framework we've built over the course of our project can offer targeted insight into which of the performance optimization features would best suit any given client.

Accomplishments:

- Created models for features
- Analyzed RT data using these models
- Identified loading patterns in the RT data that could be optimized by applying a feature
- Simulated new page loads and verified performance benefits

UC SANTA CRUZ
Baskin Engineering

Capstone Project

Smart Cart: Grocery Store IoT

Ruben Lopez, Jake McClenny, Nate Pickslay
Reese Robertson, Anthony Sacharny

ECHELON

Abstract

The goal of our project was to streamline the grocery store checkout process while minimizing the impact on both shopper and store owner, to create a more seamless shopping experience. The current method of removing and scanning each item from the shopping cart creates a bottleneck at the cash register. Our solution was to incorporate checkout into the shopping process, using a scan-as-you-go approach. The result is a shopping cart capable of accurately tracking its contents and communicating with a backend database, enabling inventory management and providing a foundation for building automation.

Approach

We explored aspects of current checkout methods in the industry to determine what problems still exist and how we could solve them. From this, we found that self-checkout is not a complete solution: it does not eliminate lines, there are numerous loopholes for theft, and produce is handled inefficiently. We researched technologies that we could use to tackle these three problems.

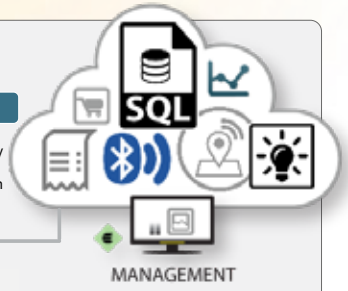
To reduce lines, the customer should be able to scan items as they shop instead of removing all the items from their cart once they reach the checkout area. In order to accomplish this, the cart needed a mobile scanner, built-in security measures, and an intuitive interface. Some of the solutions we considered for the scanner were QR codes to increase ease-of-use since they are omni-directional, traditional barcodes to use existing infrastructure, and RFID tags to make scanning completely automated. Security options depended on whether we opted for RFID tags, since item verification could also be handled by the same tags. Without RFID, accurate weight sensing was a proven solution that we could improve upon for use on the cart. The cart also needed a user interface for managing the items added to and removed from it. This interface needed to be both intuitive and unintrusive while ensuring the customer can access all the information currently printed on a receipt.

For theft protection, we decided on a few different possible directions. By using RFID tags, the cart could track its contents continuously by scanning the tags within its basket area and reporting any mismatches. Accurate weight sensing could also be used to differentiate items by searching for unexpected changes in weight or discrepancies between measured and expected weights. The final solution we came up with was using computer vision to check items entering and leaving the cart, ensuring items placed inside matched what was scanned.

Overview

Network / Back-end

- Wireless 802.11n connectivity
- SQL database for querying store inventory and tracking live items in the cart
- building automation based on cart location
- Webpage for monitoring cart contents in real-time



Weight Sensors

- Four single point load cell sensors and supporting circuitry used for accurate weight detection
- Item weights are cross checked against inventory database to ensure theft prevention

User Interface

- 5" capacitive touch-screen
- Real-time updates to price total
- Interactive list of items in the cart
- Status indication LEDs for user feedback

Item Identification

- Standard barcode scanner used to identify barcoded items
- Camera with computer vision software (OpenCV) to be able to read hue, saturation, and value measurements from pictures
- Able to create Haar Cascades to detect the shapes of objects
- The program for detection finds the color of the object as well as the shape to minimize the number of options for produce selection

Raspberry Pi / Power Supply

- All peripherals controlled by Raspberry Pi B+
- 11.1V 4400mAh Li-Ion Battery with 8 hour life-span
- 5V switching regulator

In-house Location Tracking

- Bluetooth Low Energy beacons at specific locations
- Uses the major and minor values advertised by beacons to determine location
- Bluetooth receiver on the cart reads the received signal strength indicator (RSSI) from the beacon to calculate approximate distance

Lights are in a dimmed, energy efficient mode when cart isn't nearby

Message sent over the network to brighten lights when cart is nearby

Building Automation with Echelon's IzoT Platform

- Interfaced LED lights to Echelon WiFi boards for energy efficient lighting control
- Lights illuminate brighter when cart is nearby, and then dim when not in proximity

Analysis

Upon reviewing an array of options for item scanning, security, and produce handling, we settled on the following solutions in order to minimize impact on the store and customers. While looking at QR codes, standard barcodes, and RFID, we had to take store infrastructure into consideration. In order to implement RFID or QR codes, every item would have to be redesigned with its own unique tag, which is beyond the scope of our project, so we moved forward with standard barcodes. While researching security options, we realized that computer vision technology is not yet at a level that would allow us to accurately recognize all items entering and exiting the cart without impacting the customer's routine or the cart's cost. By using accurate weight sensors, we were able to discretely and affordably track the cart's contents. Finally, we decided on OpenCV over Camfind because of its preferable failure mode. Camfind relies on constant internet connectivity as well as third party servers while OpenCV requires only a connection to our local database.

Results

Using the weight scale, the cart is able to accurately verify its contents against the items the user scans by comparing the expected weights to the actual weight measurements obtained from the load cells. This can ensure the proper items are added to the cart and the user isn't able to scan a lower-priced item and switch it out with something more expensive. The proximity detection using Bluetooth Low Energy is able to determine which of the beacons it is closest to by tracking the RSSI value and comparing it to the value of the other advertised beacons. Using OpenCV to create Haar cascades reduces the list of possible produce items to be selected on the screen. In the worst case, it can sort items by their color, and in the best case it can sort them by color and shape, only suggesting a limited selection of produce to choose from instead of the entire produce database. Due to the Lithium-Ion battery, the cart is able to run for up to 8 hours continuously. The SQL database allows the cart to offload some of its processing to a separate location, reducing the computational load on the microprocessor allowing it to seamlessly run without impacting the user experience.

Conclusion

One of the major limitations during the design process was the difficulty of the scale's mechanical design. By working with a mechanical design engineer, we could improve the accuracy and reliability of our weight scale system. Part of this also involves which cart we use as a base for the design. In the ideal case, we would design a completely new cart from the ground up to tailor it to our needs and avoid being constrained by existing hardware.

Acknowledgments

Special thanks to Shopper's Corner, Echelon Corporation, and Dave from the UCSC Machine Shop

UC SANTA CRUZ
Baskin Engineering

Capstone Project

OpenStack
Object Storage Metadata Search

Forrest Kerslager, Graham Greving, Jake VanAdrighem,
Nick Noto, Kevin Yeap

Abstract

The goal of this project was to add Object Storage Metadata Search(OSMS) functionality to the established capabilities of OpenStack Swift, an open source cloud storage platform. The OSMS API allows a user to query the metadata of accounts, containers, and objects stored in the Swift system. The output of the queries is filtered according to user-specified query statements and attribute lists. The output can also be sorted and be presented in multiple formats to a web-based interface. The searchable metadata is stored in an OrientDB or MariaDB database and populated at the proxy level. The RESTful API is implemented in middleware on the proxy server using Swift's implementation of the Python Paste Framework

Approach

Metadata Server Database

In order to improve upon the performance of the previous implementation of the OSMS system, we worked towards finding a more suitable option for housing our metadata server than the SQLite database formerly employed. Our research decisions hinged on several important factors and features that the database would need to provide. The specifications we sought included efficient database scaling, intuitive replication mechanisms, a compatible software license, and support for the query operations we required. No candidate fully satisfied all the requirements so we chose the two potential databases that had the most benefits. Our candidates were OrientDB, a distributed document database with SQL-like querying, and MariaDB Galera Cluster, a shared-nothing distributed database supporting master-master replication.

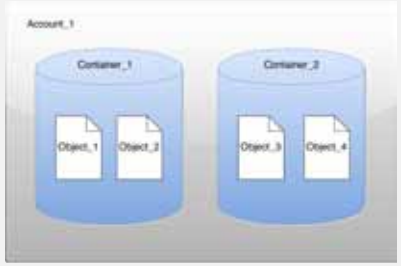
Replacing the Metadata Crawlers

To further increase the scaling capabilities of the OSMS system, we researched possible replacements for the server crawlers previously used for metadata collection. The research resulted in utilizing several of Swift's built-in classes along with our own custom metadata middleware located in the proxy pipeline to capture metadata upon storage at the proxy level. This was possible because OpenStack Swift's storage servers communicate with the proxy server where all incoming API requests are handled.

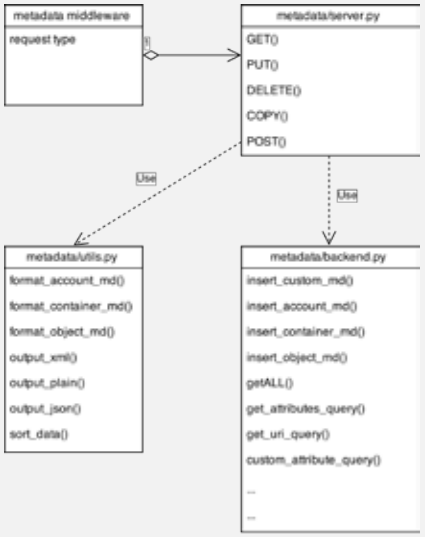
Overview

OpenStack Swift is an open source cloud storage platform used to store and retrieve data in large volumes with a RESTful API. The system is arranged into Account, Container, and Object servers along with a Proxy server for handling HTTP requests. Each Account, Container, and Object has its own list of associated metadata. The addition of Object Storage Metadata Search(OSMS) was first proposed by Lincoln Thomas from HP, with the goal of expanding OpenStack Swift to include general metadata search functionality. The previous implementation utilized a SQLite database to house the metadata server and server crawlers to periodically update the metadata database, consequently lacking scalability. Our goal was to provide a more practical, scalable implementation of the Object Storage Metadata Search API by replacing the database and introducing a new metadata collection mechanism.

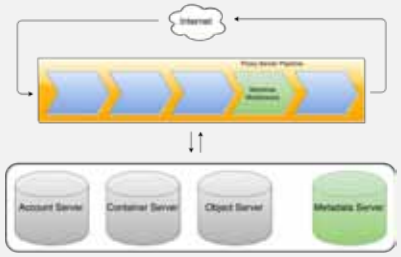
Data Model



Metadata UML Diagram



Swift Diagram



Acknowledgments

Dr. Sam Fineberg, Distinguished Technologist, Hewlett-Packard
TJ White, Software Engineer, Hewlett-Packard
Lincoln Thomas, Software Engineer, Hewlett-Packard
Dr. Richard Jullig, Faculty Advisor, UC Santa Cruz
David Haan, Teaching Assistant, UC Santa Cruz
Grace Lin, Teaching Assistant, UC Santa Cruz
Joe Arnold, SwiftStack

Performance Analysis & Testing

Our testing plan entailed preliminarily benchmarking the potential metadata databases we had chosen to ensure that they were capable of the scalability required. Additionally, we have completed white box testing on all components of our implementation, including exhaustive testing of the various supported HTTP requests and utility functions needed for the project. We have also completed black box unit tests on our proxy plugin and metadata server to ensure the output of all supported queries are correct and formatted as intended. Finally, tests designed to ensure scalability of our metadata collection and retrieval process as well as general overall functionality were completed on a multi-node Swift cluster using Amazon Web Services (AWS) EC2 service.

Results

Our testing demonstrated improved scalability along with the performance boost we had hoped we'd achieve. Both OrientDB and MariaDB scaled far better than the SQLite proof of concept developed previously. The custom metadata middleware we implemented for catching HTTP requests provides a more responsive and effective storage behavior to the metadata system. Our metadata collection technique also improved upon the scalability of the server crawlers, which ran increasingly slower with larger data sets. Tests involving general functionality also ran as intended.

Conclusion

This project was quite a learning experience for everyone involved. Despite several setbacks and challenges along the way, we were able to perservere and ultimately produce the results we had intended. The addition of the OSMS API offers users increased information regarding data they have stored in OpenStack Swift, along with providing an intuitive, powerful interface to do so. Changing the metadata server from a SQLite database to one created using OrientDB or MariaDB and capturing the metadata at the proxy level afforded drastically increased scalability and performance for the OSMS system.

UC SANTA CRUZ
Baskin Engineering

Capstone Project

Temporal:

Doctor-Patient Communication App

Kevin Bowden, Paul-Valentin Mini, Alejandro Aguilar, Max Hufft

imprivata

Implementation

The core functionality of the application is implemented in **Java** using **Android Studio**. Our app has a fragment-based hierarchy with two main activities.

The fragments are populated using **XML** layouts which enable the dynamic display of data. We used a third party to host our server using a **Python** frontend to interact with a **MySQL** Database backend which stores all of the application users' information.

Methodologies

- The security measures in use for Imprivata's Cortex app will be sufficient in securing data transactions on our app.
- We handle the issue of privacy by clearly distinguishing the parties involved in a conversation using names. We have also added a color code based on which profile is active, so there can be no confusion of who is speaking from the client's perspective.
- We have implemented a profile system where the base user can create dependent profiles to speak on behalf of their dependents.

Data Handling

All the data is synchronized with an external server hosted by OpenShift. It uses a Web Server Gateway Interface (WSGI) written in **Python** which serves as a frontend for a **MySQL** Database. Data is transferred from the server to the devices using URL calls which return **JSON**s.

Internally, the data is stored using a **SQLite** database which caches information required by the user who is currently using the application.

This gives users the ability to retrieve any data related to their account from any device as long as they are connected to the internet.

Abstract

Imprivata Cortext is a **secure mobile messaging platform for healthcare professionals**. Our objective was to take this concept and bring it to patients. We have created an **app that allows patients to securely communicate with healthcare professionals**. In addition to the communication, users are able to manage the rest of their medical needs. We also made it our goal to expand the scope of our app to include the management of a patient's dependents.

This translates to three areas of interest:

- Using Imprivata's existing security software to **securely communicate**.
- In terms of privacy, It is very important to understand **whom you are talking to and who is listening**. Our interface must make this clear.
- A patient will have the ability to manage all of their medical needs, as well as the medical needs and communication of their **dependents**.

Results

The main page of the app is a list of all the conversations that you've had with your doctors and other healthcare professionals. This allows users to easily navigate their **concerns** using the search bar and filter settings. Each concern is color coded to reflect whom the concern pertains to. Tapping on a concern sends you to a list a messages.

The user can manage all of their medical needs using the app. Users can see all of their healthcare professionals with favorites listed at the top of the **contacts page**. In the **appointments section**, the user can easily see, make, and change their current appointments. In the **prescriptions section**, the user can see the current status for all their prescriptions.

Any person with an account can make **profiles** for any dependents. Using a dependent's profile, a user can speak on behalf of a dependent and manage all medical needs using the other components in the app.

Technologies

- **Android** (Main Technology)
  - Java
  - XML
  - Android Studio
- **SQLite**(Internal Android DB)
- **Python + MySQL** (WSGI front end + database)

Software Architecture

Our software followed the model/view/controller pattern.

- The **Model** conceptualizes how the information will be handled and fed to the View. In our case we structured our information as follows: Concerns, Messages, Contacts, Auth, Profiles, Prescriptions, and Appointments.
- The **View** defines how the information given by the Model is to be displayed to the user. For this project, it was important to emphasize who the message is concerning and make it clear who can read the message. To address this, Profiles are color coded with a unique icon specified by the user's preferences and display the names of the people involved at the top of the Concern page.
- The **Controller** manages how the user interacts with the information presented by the View. The application uses custom designed advanced page transitions to give the user an intuitive way to interact with the data.

Acknowledgements

Ignacio Llamas
Software Engineer, Imprivata

Matt Caruano
Software Engineer, Imprivata

Pushkar Singh
Engineering Manager, Imprivata

Krutarth Shah
VP, Engineering Mobile Products, Imprivata

Richard Jullig
Faculty Advisor

Grace Lin
Faculty Advisor

Capstone Project

Lattice - FPGA vs MicroController

Rumer Calachan, Ma Lopez, Alexandra Marapao, Vendy Prum

Abstract

The project goal, as determined by Lattice Semiconductor, is to compare performance aspects of a Lattice FPGA and an M-cortex microcontroller as they run an implementation of a Digital Signal Processing (DSP) algorithm, such as speech recognition. We found the PocketSphinx library written in C from the open-source CMUSphinx speech recognition systems, originally developed at Carnegie Mellon University, that are primarily geared towards research in speech recognition as well as custom applications. Our high-level setup concentrates on one function of the PocketSphinx's Continuous program that is implemented one each board, C implementation for the microcontroller and Verilog implementation for the FPGA. Each implementation is timed and measured for performance comparison and analysis.

Approach

PocketSphinx, Speech Recognition Library

- PocketSphinx is a member of the CMUSphinx library, providing two programs for users to utilize for their speech recognition needs. We used the program called continuous that would take in live audio of a sentence or phrase, process calculations based on the probabilities and language model scores, and finally output the processed speech hypothesis onto the terminal. We chose PocketSphinx because it is an open source library written in C and is developed in a Linux environment.

CPU usage and runtimes

- To comprise the most ideal for a comparison of a Lattice FPGA towards an ARM M4 processor, analysis profiles were taken on a dual-core virtual machine with an affinity for the program set to one processor using Vtune Amplifier.
  - 30% of entire run time was devoted to calls to find_bg
  - For an audio file consisting of x words, the function find_bg() was called 5,178,484 times.

C-implementation

- Straight forward port of the program originally written in C.
- Used the FatFS file system in conjunction with a microSD card that stored the static bigram array as well as the other two sets of input data for processing.
- Used Em::Blocks development environment.

Verilog-implementation

- The implementation includes four main components: the state machine for find_bg, memory state machine for interacting with the EBR, and memory banks made up of EBR for bigram array and for n, w inputs.

Power Measurement

- Microcontroller
  - Current was measured over a certain amount of times and manually measured using a multimeter while being supplied 3V from the PC.

Overview

Accelerated Function: Find_bg()

- A bigram is a sequence of two letters, syllables or words. While utterances are being processed in PocketSphinx's Continuous algorithm, they are classified into certain n-grams by the language model where n signifies the number of elements in a sequence. When the algorithm detects a bigram, the bigram has a corresponding score that helps the program determine what words are better matched with a certain word IDs in the language model. These word IDs are stored within an array within the n-gram model data structure defined in PocketSphinx. The function determines the bigram scores by performing a binary search on segments of the bigram array, updating the indexes, until the word ID matches the one the current utterance word ID. If the binary search is unsuccessful, a linear search traverses through the array until the desired word ID is found. When this word ID is determined, it returns the index which tells the program at what index to find the particular bigram score.

Hardware:

LatticeECP3 Versa Development Kit

- The LatticeECP3 Versa is a versatile platform for developing embedded systems in a variety of applications and fields such as video transmission, image signal processing, data acquisition, and more. The LatticeECP3 is focused on making designing more efficient, while maximizing reliability, minimizing the cost, and power.
- The LatticeECP3-35 FPGA offers a lower power, low cost solution for a wide variety of applications such as wireless communication, video processing, surveillance, networking, and more.
  - The LatticeECP3 contains about 33,000 LUTs, 6.8Mbits of SRAM, 1Gbit DDR3, 64Mbit SPI Flash memory, offers support for PCI Express, Ethernet, and other various features.

Microcontroller: STM32F407VGT6 with ARM Cortex M4 processor

- ARM M4 Processor
  - The STM32F4 Microcontroller contains an ARM Cortex M4 processor, which is ARM's high performance embedded processor, which contains features pertaining to simplify the easy blend of control and signal processing capabilities.
  - The STM32F407VGT6 microcontroller is based on the high performance ARM Cortex M4 that operates with a frequency of 168 MHz. The microcontroller contains high - speed embedded memories of Flash memory up to 1Mbyte, 192 KBytes of SRAM, etc. Standard communication interfaces like UART, I2C, SPI, etc. are also available.
- STM32F4 Expansion Base Board
  - Accessory board containing microSD card slot along with other hardware components for STM32F4 family of boards.

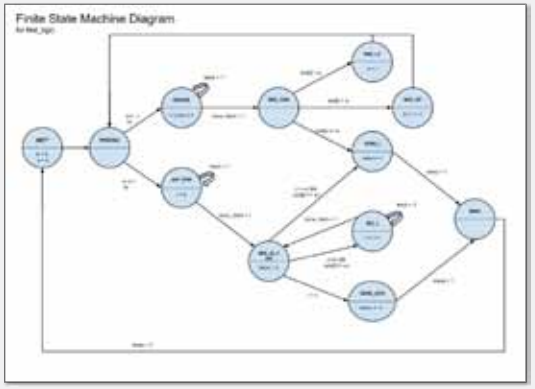
Software:

Lattice Diamond development software

- The Lattice Diamond is a GUI based FPGA design and verification environment for the ECP3 family and other devices, to create our Verilog modules.

Em:Blocks development Software

- Em::Blocks is a free C/C++ IDE for users working in the field of embedded software development.



Acknowledgments

Our team would like to acknowledge Lattice Semiconductor for sponsoring our senior design project and providing us with a research topic that we felt compelling and significant to explore. We would like to extend our appreciation to the Baskin School of Engineering for supporting this senior design program and providing a means for students to collaborate and complete a senior project of their own design and interest. We would also like to thank our senior design mentors, David Munday and Ethan Papp, for their insightful mentoring.

Analysis

Utilizing the Vtune Amplifier analysis tools, the total expected runtime of the Pocketsphinx Continuous program was 2.794 seconds. The top-down visual map of the Pocketsphinx Continuous program demonstrates that the program's functions are evenly distributed, where bottom-functions are relatively fast computations. The visual map of the program as well as a hotspot analysis confirms that find_bg() is where the program spends the majority of its time. In total, 4.1% of the CPU's usage was spent during calculations of find_bg(), resulting in a total contribution of runtime of .103 seconds. The audio file used for profiling consisted of 26 words, and overall the function find_bg() was called 5,178,484 times.

Microcontroller Measurements

Time Measurement

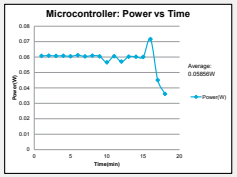
- Approximately 3000 runs were made in the 2 hour time frame.
- Average time taken per run: 16.2 milliseconds.
- Maximum time taken: 99.65 milliseconds

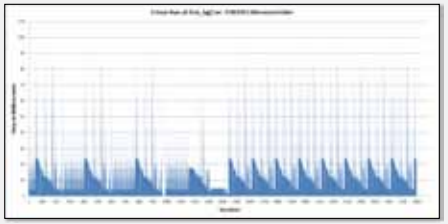
Power Measurement

- Approximately 1000 runs were made within the 18 minute measurement with an average power of 0.05856W/min.

Results

Microntroller: Power and time





Conclusion

At this time, we are still testing the FPGA portion of the project. We hope to find a significant contrast between the STM32F4 Microcontroller runtimes and the LatticeECP3 Versa runtimes along with the power consumption comparison.

Capstone Project

RISCV Soft Processor

Andy Yeh, Oscar Navarro, Keilah Reyes

Abstract

The goal of our project was to port the 5-stage Sodor processor core created by Christopher Celio at UC Berkeley from the Chisel and C++ built emulator to Verilog that could be put onto Lattice Semiconductor's FPGA. The Sodor processor uses the RISC-V ISA which was chosen for its suitability for hardware implementation as well as its being open source. With the Sodor core targeted to Lattice FPGAs, further comparisons to Lattice's existing soft processors can be made.

Approach

We started with the 1-stage core to understand how the unpipelined processor deals with instructions in simulation. This gave us a working baseline that we could compare to the more complex cores. One issue was to make sure that memory reads and writes were properly handled with Lattice's distributed (asynchronous) RAM.

While working on the 1-stage core, we needed to build the programs we would run on the Sodor processor. The test toolchain we created sets the Linux station environment variables such that we can implement the RISC-V toolchain compiler. Once the compiler is set, we have a script that compiles any C code and extracts the main portion of the program. Once that is done, another program runs through and gets rid of any unnecessary 0's. The finish product is a hex file that fits into 5 stage core EBR and can be loaded through Lattice's software tools.

Once we verified that the 1-stage core could run a program as expected, we moved onto the 3-stage core. This core natively supported synchronous RAM. We modified the core to use Lattice's dual-port RAM that implemented synchronous Embedded Block RAM (EBR). Using the same testbench and test programs, we verified that the 3-stage core produced the same results as the 1-stage core. This was a huge deal that allowed us to understand the differences between how a single cycle and pipelined core, as well as, asynchronous and synchronous ram operated.

We ended with the 5-stage core by taking everything we had learned about pipelined processors and the synchronous versus asynchronous RAM. The 5-stage core was natively asynchronous. However, to make the core more efficient and use fewer resources, we wanted it to use the synchronous EBR. The first issue, due to the core expecting an asynchronous memory, was found in the Instruction Fetch (IF) stage. We modeled the IF stage after the 3-stage core's IF stage so that instructions would be ready every cycle. The second issue came from the Memory (MEM) stage. To allow the core time for read or write data to take effect, we had the RAM access take place during the Execute (EXE) stage rather than the MEM stage. This allowed data to be ready by the MEM stage, and it could be passed to the Writeback stage (WB). We ran the same tests on the 5-stage core as we did the 1 and 3-stage cores. We expanded the 5-stage core to include memory mapped GPIO.

Once we verified the cores in simulation, we synthesized the 3-stage and 5-stage cores and created a bit file to put on the Lattice's FPGA, the ECP5. We used the Lattice Reveal Analyzer to view vital hardware signal traces to ensure the processor was running correctly on the FPGA board.

Figure 3: Sodor soft core processor (1-Stage and 3-Stage)

Overview

Synchronous vs Asynchronous Ram

The main difference between asynchronous and synchronous dual port RAM is how memory is accessed. Asynchronous read/write operations are triggered when signals are flagged, while synchronous read/write operations are triggered on the clock cycles. This means that synchronous RAM can pre-fetch data out of memory as all flags are determined; asynchronous ram is slower due to their architecture.

Unpipelined vs Pipelined Processor

Instruction pipelining is a technique to increase the number of instructions processed by the CPU in the given amount of time. Instead of processing each instruction sequentially (unpipelined), each instruction is split up into a few steps so different steps from multiple instructions can be processed at the same time.

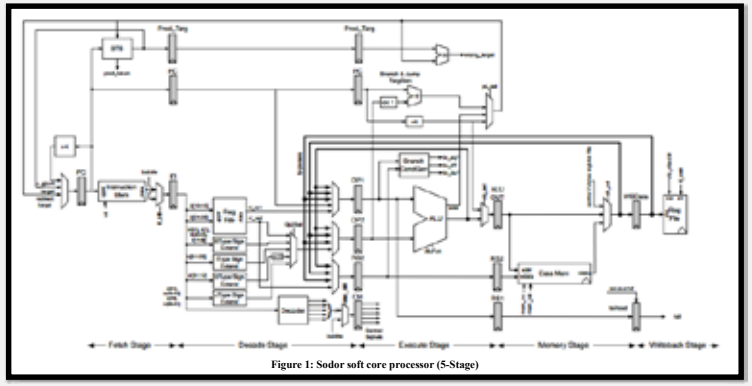


Figure 1: Sodor soft core processor (5-Stage)

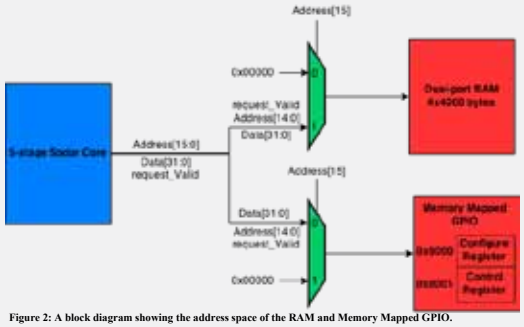


Figure 2: A block diagram showing the address space of the RAM and Memory Mapped GPIO.

Acknowledgments

We would like to thank Dave Dutt, Hua Tang, and Kam Chuen Mak at Lattice Semiconductors for their guidance throughout this project. Secondly, we would like to thank Christopher Celio helping us understand his work on the Sodor Processor. Lastly we would like to thank David Munday, Ethan Papp, and UCSC Baskin School of Engineering teaching staff for the guidance and leadership they provided.

Analysis

In the 1-stage, we replaced the original 3-port scratchpad with two dual-port RAMs for instruction and data memory. Each RAM is made using distributed memory and is 4x1024 bytes. This core requires an asynchronous RAM because it is single-cycle. Only distributed RAM, which is implemented in LUT logic, is capable of accessing memory and outputting the data in one cycle.

The 3-stage core uses a dual-port RAM that is 4x8192 bytes. The RAM implements Embedded Block RAM (EBR). The 3-stage is a pipelined, synchronous core. It has three stages: Instruction Fetch (IF), Execute (EXE), and Writeback (WB). With a synchronous RAM, reads and writes to RAM take two cycles to take effect. During the IF stage, the core predicts that the next instruction is located at the memory address PC+4. In the same clock cycle, the core sends a memory request to the RAM for address PC+4 while receiving the data from the previous IF stage and decodes it. If the core mispredicted, due to a branch or jump, a delay slot is added and the new memory address for the next instruction is sent to the RAM. If the core detects a hazard, a NOP is sent to the EXE stage and the PC is stalled. IF prediction and the delay slot ensure that instructions are ready for the next IF stage.

The 5-stage core uses a dual-port RAM implemented in EBR. The RAM is 4x4000 bytes. The 5-stage core is pipelined and synchronous. It has five stages: IF, Decode (DEC), EXE, Memory (MEM), and WB. The IF stage in this core works similarly to the 3-stage core IF stage. However, an extra delay slot must be added whenever a branch or jump occurs. The core must clear out the pipeline to ensure that no extra instructions are executed while waiting for the branch or jump to execute.

The RAM for the 5-stage is half the size of the 3-stage core because the 5-stage core has Memory Mapped GPIO. The RAM uses any address space above 0x7D00 while addresses below that are sent to the GPIO. The core uses the 16th bit of the address to determine whether the data should go to the RAM or GPIO. The GPIO uses a Configure register and a Control register. To write to the GPIO, a program must use a store instruction to write the desired output data to address 0x0000, the Configure register. Then the program must use another store instruction to write to address 0x0001, the Control register, with a 1 or a 0 to turn the GPIO on or off, respectively.

Results

We now have three types of RISC-V soft processors that are targeted for Lattice Semiconductor's ECP5. The 1-stage core is a single-cycle processor that uses asynchronous memory and is useful for analyzing single RISC-V instructions. The 3-stage core uses fewer LUTs than the 1-stage because it implements a synchronous memory in EBR. This core is a solid baseline for how a pipelined core should behave with SRAM. The 5-stage is also pipelined and uses SRAM which saves on LUT resources. It also has a memory mapped GPIO that can be used for outputting results to LEDs or reading and writing to a UART.

In addition to the processor cores, we have a compiler toolchain that allows us to write C programs and turn them into hex files that can be loaded directly into the RAM to be included with the synthesized bit file.

Conclusion

Now that we have three cores that are targeted for the Lattice ECP5, we can benchmark each processor and compare it to the Lattice Mico32 soft processor. The memory mapped GPIO paves the way for more IO capabilities in future iterations of the 5-stage core.

Capstone Project

eMMC Hardware Controller

John Dilger, Travis Johnson, Hunter Nichols, Sergy Pretetsky

Abstract

In business and consumer applications, bandwidth and latency to storage often bottlenecks the overall performance of the system. eMMC is a flash-based storage device designed to offer good performance while hiding the complexities of flash memory management. Interfacing with eMMC requires precise timing and clock management, requiring a controller to translate requests into the eMMC specification. Our project involved creating this controller in synthesizable verilog and targeting a Xilinx ZC706 evaluation board for testing.

Generally speaking, hardware should be minimized and simplified when possible to minimize power and die area usage. However, the eMMC device could be caught in many different error states that, if handled by hardware, would greatly increase the complexity of the controller. Furthermore, the initialization procedure would increase the complexity of the controller even more.

These factors led us to the philosophy that the eMMC controller should only perform the critical timing management and clock management that is required of it. All eMMC state management should be done by a software controller.

Approach

The eMMC controller is designed to interpret commands and responses as little as possible. It is the software driver's role, then, to control the state of the eMMC by sending appropriate eMMC commands and interpreting eMMC responses. The hardware controller is responsible for handling all timing, switching of eMMC speed modes, and switching between SDR and DDR modes (single data rate and double data rate). It is also responsible for reporting transmission errors to the software driver. Transmission errors include the hardware controller failing to detect any response when it is expected and a data block or response having mismatched CRC (cyclic redundancy check) values.

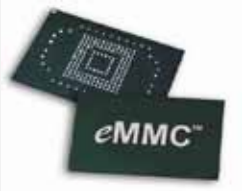


Figure 1: A sample eMMC chip http://www.xeltek.com/emmc-programming-using-xeltek-programmers/.

Overview

eMMC is a flash storage standard maintained by JEDEC. eMMC devices are typically packaged as BGA chips meant to be soldered and used in embedded applications. eMMC is commonly used in smart phones, tablets, and small laptops.

We used a Xilinx ZC706 evaluation board to realize our design. The ZC706 uses a ZYNQ 7000 SoC that includes two arm processors along with a programmable logic fabric. This allowed us to implement programmable logic and a software driver easily on the same platform.

The eMMC hardware controller is designed around several finite state machines (FSM). The primary FSM is responsible for handling the current operation being formed. Generally speaking, all eMMC transactions can be broken up into several operations:

1) Receive command from software driver and send it to eMMC.
2) Receive response from eMMC and send it to the software driver.
3) Send/receive user data to/from the eMMC to software.

The process of handling these operations is shown in the flowchart in figure 3.

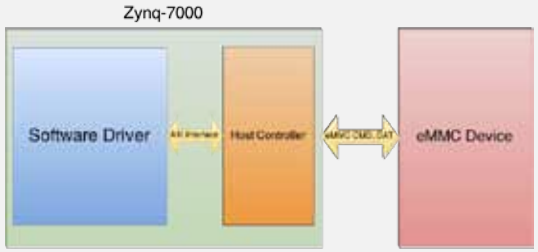


Figure 2: Diagram of Complete System

Analysis

Our first iteration of the eMMC controller featured an automatic initialization which involved the controller sending all the commands required to initialize the eMMC without needing input from software. Furthermore a strict input structure specific to the controller was required to issue the set of commands to perform reads and writes on the device. As we learned more about the specification and from early testing, it seemed preferable to change the focus of the controller to be more manual. This allows the connected software driver to specifically issue each eMMC command and have more freedom to access the device.

This design allows for several key features:

- Full control of the eMMC from software.
- Less complexity in the controller.
- Allows software to properly handle difficult errors.

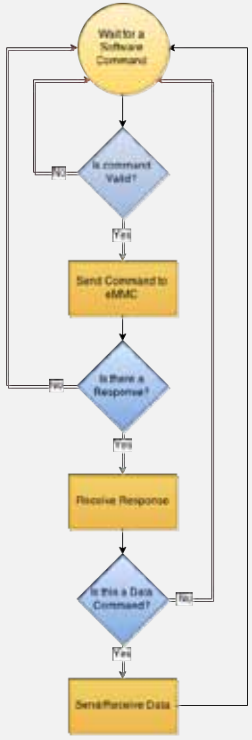


Figure 3: Flowchart of controller operation.

Results

With our supported set of eMMC commands, our controller is able to perform initialization, multiple/single block reads/writes, switch modes on the eMMC device, access the extended CSD, access the current status, and change the eMMC device state to inactive. Our controller also supports custom commands to change how the controller itself works. These controller commands toggle the clock between slow for initialization and fast for normal operation, set the data width and mode, and set the timeout length.

Our controller supports High Speed mode (up to 52 MHz) and Dual Data Rate operation, with groundwork in place for HS200 mode. Our controller also supports the option for a data bus width of one, four, or eight.

A software driver is able to interface with our controller through a series of FIFOs. They can send eMMC commands, receive responses, read/write data, and determine if there were any transmission errors.

Conclusion

A prototype eMMC controller was implemented on a Xilinx ZYNQ 7000 SoC. It supports eMMC commands that would allow the software driver to initialize the eMMC device, read/write data to the eMMC device in single blocks or multi-blocks, and change speed and bus modes.

The controller was tested by initializing the eMMC device, changing to high speed mode, changing to 8x bus width, and performing multiple single block and multi-block reads and writes.

Acknowledgments

We would like to thank the following people and companies:

Our mentors David Munday and Ethan Papp from UC Santa Cruz
Our advisors Jon Lexau, Hesam Moghadam, Vincent Lee from Oracle
Oracle for sponsoring this project
Xilinx for providing Vivado licenses

Capstone Project

Bare Metal Driver for eMMC Controller

John Dilger, James Giovaccini,
Kevin Jesse, Shania Asadipour

Abstract

Embedded MultiMediaCard (or eMMC) is flash storage in a form that allows it to be embedded onto a printed circuit board. Almost all mobile phones and tablets use this form of flash for main storage. The JEDEC standard defines a list of requirements to communicate with the eMMC module, all of which must be executed prior to sending and receiving data to the chip.

Our task is to design a solution on the Zynq-7000 SoC to communicate with the eMMC module.

Although other interface solutions exist, including purely software drivers (often written in C or Assembly Code), and purely hardware solutions (Integrated Circuits serving as a translation between one data protocol and another), our driver utilized both methods. The communication between hardware and software is done through a proprietary protocol known as AXI (Alternative Extensible Interface), which is developed by ARM Holdings.

The interface is available on Xilinx's Zynq-7000 SoC to allow the ARM Cortex A9 to send and receive data from programmable logic over AXI. Altogether, we are able to design the routing of the signals to the eMMC controller in programmable logic

Approach

Abstraction as Generic Memory

The software will enable the user to communicate with a set of eMMC devices using intuitive function calls to be called in a C program:

- emmc_initialize()
- emmc_get_devices()
- emmc_read(destination, device, address)
- emmc_write(source, device, address)

These functions require parameters that are not specific to eMMC, and could be required by a driver of a similar memory device type. This abstraction is accomplished by modeling the hardware and achieving data reliability.

Hardware modeling

In order to communicate effectively with the eMMC device, a driver must accurately and dynamically represent both the collection of chips that are embedded in hardware and the hardware that provides these chips with logics; clock speed, for example. This provides the software with the information needed to perform user operations. Errors causing the chip status to become unknown should be recoverable, and will cause the driver to repair the model or inform the user that the hardware can not be used correctly given the situation.

Data reliability

Write and read operations in eMMC are subject to noise on the bidirectional data lines and are protected by cyclical redundancy checks (CRC). When the software receives data feedback, or lack thereof, it will find that the reads and writes that span multiple blocks may have corrupted or missing blocks.

There is an opportunity for the driver to:

- Notify the user of which blocks had succeeded and which had failed.
- Perform a number of smaller reads or writes to patch the operation.

Automatic repeat requests (ARQ) are performed in the software to abstract away the CRC checking that occurs in eMMC transactions. These ARQs are block-selective to minimize overhead. Automated recovery provides a more reliable read or write request which assures the user that, if it can be contacted, all memory will be accessed.

Overview

This driver compliments a hardware controller created by the Oracle Hardware Team, and was concurrently designed. It generates a sequence of eMMC standard commands and places them in buffers to be removed by the eMMC controller. When the hardware indicates that the commands have been processed by issuing an interrupt, the driver will extract the result of the eMMC command.

Memory-Mapped FIFOs

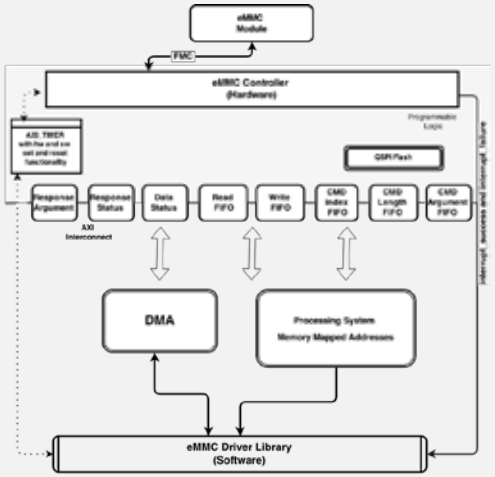
The primary method of communication across AXI involves populating FIFO's in Programmable Logic (see hardware) and accessing these FIFOs through memory-mapped locations in software.

Initialization

eMMC behavior is configurable, and initialization is required to adjust the eMMC devices and hardware controller to suit the user operations. Calling the initialization function will result in a search for all available devices located on a controller and have those chips prepared according to what the user requests. When no more chips can be found the initialization closes, clocking the controller to data operation speed.

Software Flow

System Block Diagram



Acknowledgments

We would like to thank Dr. David A. Munday and Ethan Papp for their supervision and support.

We would also like to thank the following individuals from Oracle for their support:

- Jon Lexau
- Vincent Lee
- Hesam Fathi Moghadam

Analysis

Performance Analytics

The performance of the software driver was measured in sectors (512 bytes) read and written from the eMMC module. Our design allowed for the user to write a sequence of sectors to the eMMC module, so we decided to measure the performance of our driver in regards to sectors written to the chip and sectors read from the chip as a function of length.

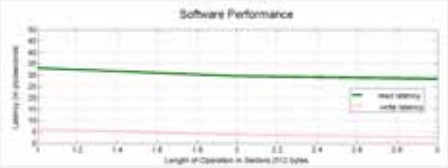
Software Scalability

As the project progressed, the hardware had switched from issuing eMMC commands automatically to issuing them manually. This change improved the scalability of the project by separating mutable application specifics from the immutable hardware. Issuing commands, rather than a read/write request, gives the software better control of error handling and allows for features to be added with less development effort. The non-user functions within the driver can be rearranged to create functions beyond read and write for other drivers if desired.

Results

Software Latency

To get an accurate measurement of our software latency in comparison with the overall latency of read/writes, care was taken during testing to select the locations where delay is measured. This measured the latency of the software from emmc_write() or emmc_read() being called to the completion of the call, minus any delay in which the processing system was waiting for an interrupt from programmable logic. In essence, this evaluates the driver in isolation to get a more accurate measurement of the driver's performance.



Conclusion

With the defined interaction between controller and bare metal driver, the development of a Linux driver can expand the user's capability with the eMMC. The PetaLinux kernel is small and yet powerful, coupled with an eMMC Linux driver can provide the same level of access in an operating system environment. The robustness of an operating system allows for thread management, such that during read and write operations the processor does not waste cycles before a data response. There is less accessibility to the processor in a kernel environment so development from bare-metal driver to a Linux driver can provide some difficulties. The software team explored both character and block driver models to represent the existing bare-metal driver in Linux.

The driver is an effective solution for communication with an eMMC module, however, further improvements can be made to the driver's data reliability and error handling.

UC SANTA CRUZ
Baskin Engineering

Capstone Project

Serena iPhone Application

Anthony D'Ambrosio, Evan Hughes, Neeraj Mallampet,
Rachelle Tanase, Vishakha Goel

SERENA

Abstract

Serena Software Inc is a provider of IT management products. One of their major products, Serena Deployment Automation (SDA), allows users to seamlessly automate their deployment process.

We were tasked with creating an iPhone interface for SDA. The goal was to allow users mobile access to SDA, providing them the ability to monitor and manage their deployment system anywhere.

Approach

MVC Architecture

- Model View Controller is a common software architectural pattern for developing user interfaces.
- It divides an application into three interconnected parts in order to abstract the internal representation of data from its presentation to the user.

Serena Server

- We were provided with Serena's Web Application and Server in order to learn how to interact with it and extract the necessary data.

Scrum Methodology

- Using the Scrum approach enabled well-coordinated execution of tasks and releases of the application.

Challenges

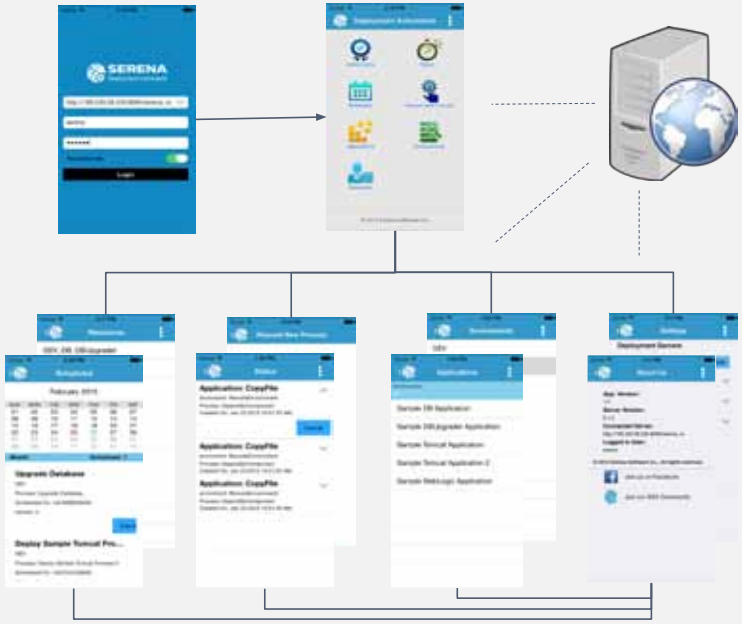
Throughout this project, our team hit a number of obstacles that we had to overcome. The most challenging problems we faced were:

- **Learning a new language**
  The team knowingly took on the challenge of learning a new language and technology, as we had very little prior experience with Objective-C and Xcode. This had a negative impact due to a steeper than expected learning curve, preventing us from utilizing the vast toolset efficiently for quite some time.
- **Poor time estimates in Sprint planning**
  The steep learning curve impacted our planning as well. We overestimated the amount of work we thought we could accomplish, only to be deterred by the time it took to learn before being able to implement various items.
- **Recreating the Android application**
  Our sponsors asked that we retain the look and feel of an Android application despite creating an iOS app. Some features common amongst Android apps, such as the overflow menu, are not provided in Apple's standard libraries. After much research we were able to achieve the creation of these features.

Overview

We created an iPhone application that mirrors most of the features of the Android application. It communicates with the SDA server to obtain the data for each screen. Our application allows users to approve or deny action items, view the status of running and scheduled processes, and request new processes.

The following screenshots detail the application's layout:



Acknowledgments

We would like to thank our sponsors Julian Fish, Ashish Soni, and Kyle Parent for their guidance in the development process as well as providing us this opportunity to further improve our software engineering skills. We would also like to acknowledge Professor Richard Jullig, our TA Grace Lin, and everyone else involved with CSSPP for getting us partnerships and doing all the behind the scenes work required to make this all possible.

What We Learned

We learned a lot about iOS programming and the scrum process over the course of this project. We learned how to use:

- **Objective-C and Xcode**
  - The first thing we had to learn was Objective-C and Xcode. Through practice we became more comfortable using the interface builder, Auto Layout, Objective-C's non-standard syntax, and making server calls to communicate with Serena's server.
  - All of our team members got hands on experience with Objective-C, whether it was making backend server calls or setting up the user interface for Serena's customers to interact with.
- **Agile and the Scrum Process**
  - Throughout the project we learned how to use Scrum outside of the classroom. This meant setting our own goals and deadlines.
  - With the weekly client meetings and planning sessions, we gained a deeper understanding of what industry best practices are, and how to best use them to achieve our goals.

Technologies Used

- **Xcode**
  We utilized Xcode and Objective-C to write our application. Xcode was the obvious choice of IDE since it is free and has all the tools you need for iOS development.
- **Source Tree / Git**
  Git is a distributed version control system that allows a team to collaborate separately with speed and efficiency.Source Tree is a powerful Git client with a very simple interface for developing projects. It helped us maintain our repository and integrating our codes as a team.
- **Trello.com**
  Trello is web based project management application which we used to maintain a scrum board to distribute and analyze tasks for the team.

Conclusion

Overall, building this application has been an excellent learning experience. Working with Objective-C and Xcode gave us experience and insight on how these skills are applicable in the "real world". Furthermore, we learned how to plan an entire project based off the documentation and verbal description provided by the customer. Despite the many challenges and steep learning curve, we were able to adapt, adjust and achieve usable functionality.

UC SANTA CRUZ
Baskin Engineering

Capstone Project

Scene Detection in Broadcast Television

Brittany Arthur, Hemant Ramachandran, James Marshall, Kevin Doyle

Abstract

TiVo offers a subscription-based, interactive television service that lets viewers program and control which television shows they watch, and when. This project consists of two parts.

- The first part is to develop a robust algorithm for detecting scenes. In this project, we chose to apply this algorithm to the detection of commercials. It is designed to work across a wide range of genres and types of shows. Predictions from three data sources are used together to arrive at a final prediction of where commercials reside. Weights indicating the accuracy of each algorithm are used to aid the process of deciding how to integrate predictions into a final result. These weights are calculated by measuring how many false positives each algorithm produced for a certain number of videos.
- The second aim of the project is to develop an App for TiVo. Through a simple, fun interface, users are able to find great shows and receive improved, personalized show recommendations from TiVo based on their taste.

Approach

After doing extensive research and making very few assumptions, we have developed a scene detection algorithm applied to commercials that returns a list of predictions of where commercials reside in a given video. TV scenes are made up of visual data, audio data, and closed captioning data. For each of these three sources, an algorithm is developed to predict where commercials are using that source alone, and each algorithm uses a pattern to arrive at a final prediction.

- The audio algorithm analyzes temporal fluctuations of silence.
- The black screen algorithm detects dense clusters of very dark video frames.
- The closed captioning algorithm utilizes a set of patterns to detect distinct formatting changes.

The findings are combined into a final prediction using a simple decision making algorithm that uses data about the historical accuracy of each algorithm, which has been calculated from a test set of data.

For the web app, our sponsors challenged us to come up with an exciting new feature for TiVo's customers. In response, we created an app that helps TiVo users find new shows that closely match up to their interests. We looked at popular apps to find out what made them so easy and fun for users to use.

TOOLS USED:

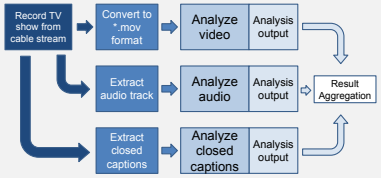
- Python for both the commercial detection project and the web app, which uses the Django framework
- OpenCV for video analysis
- CCExtractor for closed caption analysis
- Digital Ocean server for our production server
- HD Home Run to collect our data

Overview

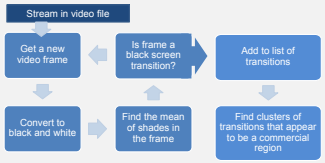
SCENE DETECTION ALGORITHM:

We have designed a general method for scene detection and applied it to commercials in various types of shows. The scene detection algorithm integrates predictions from algorithms using audio, black screen, and closed captioning analysis. The idea is to combine the three algorithms using weights resulting in a method that will accurately predict commercials.

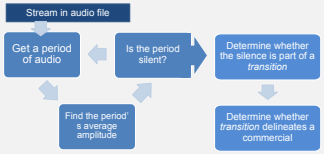
High Level Pipeline



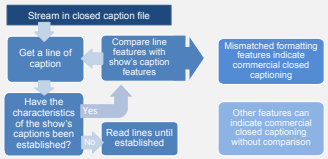
Video Analysis



Audio Analysis



Closed Caption Feature Analysis



SHOW PREFERENCE WEB APP:

In addition, we have built a web app that allows users to log in, see many TV shows, and evaluate them accordingly. Our web app gives TiVo users a way to gradually train their TiVo to their personal television viewing tastes. The user is presented with one show at a time and either selects the option to 'like' or 'dislike' the show. From this feedback the app builds profile information on a user's preferences. Ideally, this would be used to fill the TiVo database with information that they can use for that user and for that show, allowing for a more personalized experience for the user, and allows TiVo to have more information on each show.

Acknowledgments

Mark Berner and Matthieu Chamik from TiVo for being incredibly helpful sponsors throughout the project.
Dr. Richard K. Jullig, Grace Lin, and the CS Department at UCSC for giving us this opportunity to work on this exciting project.

Analysis

The graphs on the bottom show the location of commercials in a video over the length of the video. The actual location 'answers' are shown in yellow. The predictions of each algorithm are also shown along with the final prediction that is created by combining the three algorithms.

To determine the reliability of the detection method, our team used a diverse set of shows. For example, a sports show and a comedy may have distinctly different visual, audio, and closed captioning data than a game show. By testing the algorithms against a diverse set of data, our team was able to gain more confidence about how robust the algorithm truly was.

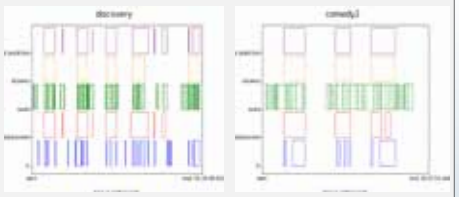


Figure 1,2: Predictions compared against actual locations of commercials

Results

Across the set of all video data, 78% of the entire commercial time was correctly predicted to be a commercial by this method. There was a diverse collection of videos not only in type, but in length and the average percentage of commercial time predicted when averaging each video result was 76.93% along with an 83.51% median and a 13% false positive rate.

Conclusion

We were able to achieve and learn the following:

- Using Python, OpenCV, the HD Home Run, the CCExtractor, and Digital Ocean Server, we combined black screen detection, audio, and closed captioning to make an algorithm that has a high accuracy of commercial detection and is able to work with all types of shows. It works particularly well under the assumption that shows do not start with commercials.
- Using Python, mySQL, and Django, we were able to make a web application for users to evaluate TV shows. The data they enter is gathered and stored in a database.
- We learned how to effectively work with Python as a programming language and use it in combination with Django.

UC SANTA CRUZ
Baskin Engineering

Capstone Project

Toshiba Storage Benchmark Workload Analysis

Peter Czupil, Craig Hu, Luis Nuñez, Hai Vo

TOSHIBA
Leading Innovation

Abstract

In a world where big data analysis is becoming increasingly more popular, storage device performance has become a focal point for optimization. Toshiba's goal is to understand the performance of HDDs (Hard Disk Drives) and SDDs (Solid State Drives) that are being used in a cloud environment through multiple Hadoop benchmark tests. To find interesting patterns in the data that potentially provide clues for design improvements Toshiba needed intuitive ways for interpreting the raw data.

Our project designed and implemented a benchmark data visualization system for the easy and rapid generation and viewing of a variety of analysis data in different visual forms. The system is designed to cope with industry standard benchmark tests which can result in millions of data points.

Approach

We were given disk I/O trace data resulting from HDFS (Hadoop Distributed File System) benchmarks (DFSIO, Terasort, Pi, etc...). This data consists of: address of request, seek time, request size and runtime of the benchmark, all formatted to CSV (Comma Separated Values) file format.

Disk I/O Analysis Research

To understand different ways of looking at disk traces, we reviewed research literature on disk I/O (Input/Output) data. We applied the different analyses that we identified to Toshiba's disk trace data.

Data Visualization

The size of the input data sets can be very large. This required algorithms that quickly computed data points and managed memory efficiently. However, our software needed to balance performance and readability; thus we used the Python programming language to create data visualization modules to incorporate the analyses from our previous research. We used different types of plots (heatmap, scatterplot etc.) to expose patterns in the data.

Data Management and Storage

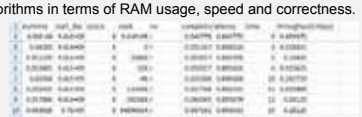
We stored the data in a PostgreSQL database to reduce the amount of computation we needed to do. We created a management system to handle data storage and retrieval.

End User Operability

We created a CLI(Command Line Interface) and GUI (Graphical User Interface) so that the end user has a pleasant experience invoking the visualizations.

Testing

The algorithms we developed were tested on sponsor provided data sets with different sizes that ranged from thousands of entries to hundreds of thousands entries to validate the viability of the algorithms in terms of RAM usage, speed and correctness.



This is an example of the disk trace from a benchmark run. Contained within are records detailing each disk request. It details the runtime, starting address, I/O size, seek distance, I/O type, latency, and average transaction speed of the I/O transactions handled by a disk.

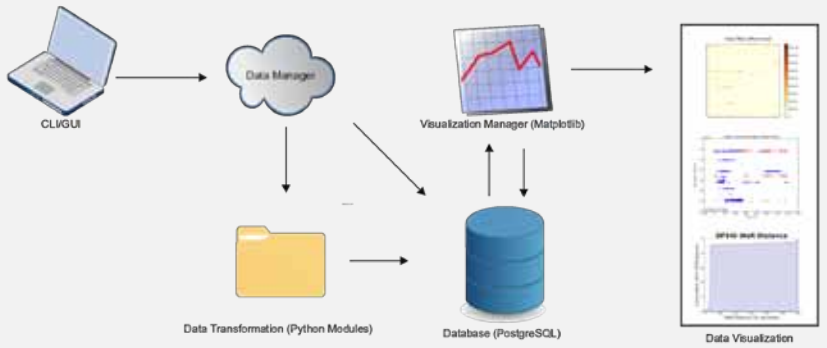
Overview

●We produced a visualization system for analyzing disk I/O traces generated by a HDFS benchmark run. Rapid and flexible exploration of this data may reveal interesting patterns that can provide clues to Toshiba for disk design improvements. Our research identified a range of I/O analysis and visualization methods. One unexpected design challenge was the large size of the datasets to be visualized. Given the large size of the dataset, our design goal was to analyze it with a moderate amount of RAM and time. RAM usage can be significantly shortened by storing computational data in a robust database. We then needed to optimize the data preparation time by introducing specialized algorithms pertaining to each type of data analysis. These algorithms were then coded into Python modules to transform the benchmark data into data points for our graphs.

●To analyze the benchmark data, we used multiple **types of data analysis techniques** such as:
- **Spatio-Temporal**: observes memory block access over a certain amount of time.
- **Read and Write Reuse Distance**: examines the reuse distance of each memory block.
- **Number of Reads to a Unit Block**: shows if I/O accesses tend to visit disk blocks that are near one another.
- **Write after Read**: determines temporal locality of disk requests i.e. about how long does it take (on average) for a write request to hit the same address as a previous read.

●To visualize the data, we created different **types of graphs** to represent the analysis, including:
- **Heat Map**: shows, e.g., which logical block address was accessed and at what frequency.
- **Line graph**: shows how features of the dataset may vary with respect to one another as a continuous relation.
- **Scatterplot**: shows how features of the dataset may vary with respect to one another as discrete points.

●A GUI/CLI frontend is used to choose which graph to display, how many graphs per plot and which data to use. The storage backend consists of a PostgreSQL database to store input data and its transformation data (data points). Once the frontend receives a graphing command, it sends the configuration to the data manager which interrogates the database to see if the graph has already been computed. If no graph exists for the given dataset, the data is sent to the Python modules for computation and stored in the database. Python plotting scripts are then executed to plot the data.



Acknowledgments

We would like to thank Ken Allen of Toshiba for the project idea, his patient guidance, and regular on-campus visits. Richard Jullig deserves a thanks for keeping our spirits high and believing in our ability to attain our goals. We also would like to thank Grace Lin for helping us create concrete user stories for our project. Another thanks goes to Ethan Miller for sharing his experience with storage devices and their benchmarks.

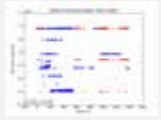
Analysis

A major part of the project involved researching effective ways of visualizing benchmark disk traces. Our data analysis was also focused towards comparing two (or more) different hard/solid state drives running with the same benchmark. We looked at the uniformity of the spatial and temporal locality of the disk requests. Specifically, we looked at measurements like:

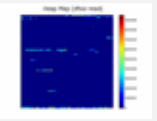
- read/write reuse distance
- number of reads to a logical unit
- Frequency of read/writes

In general, this analysis was used to show the spatial and temporal layout of the disk requests. Different ways to visualize this data (line graphs, heatmaps, etc.) were also essential to the analysis.

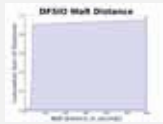
Results



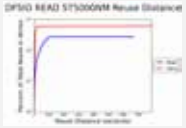
Spatio-temporal (scatterplot): shows the amount of hits to an address over the benchmark run.



Spatio-Temporal (heat map): demonstrates the frequency of hits to an address and the proximity they are in relation to one another.



Write after Read Distance CDF (line graph): The distance is time between the write after read (in seconds); this is when a location is read from and then written to with no requests to that address in between. .



Read Reuse Distance (line graph): Reuse Distance counts the I/O reads and writes for every unit of a transaction and turns it into a cumulative distribution function up to total percent reads and writes of the disk.

These are example graphs generated with DFSIO benchmark data.

Conclusion

Our project produced a benchmark data visualization system that allows Toshiba to explore flexibly, rapidly, and conveniently industrial-size benchmark data sets for data storage devices. Prior to this project the visualization was generated manually and resulted in crashes when the benchmark data got larger than the system's RAM. The system is also made to be simple and easy to use without programming experience. Lastly, all of the data that is benchmarked will be stored in the database so that future access to older data will be quicker and easier. Despite considerable challenges we met the goals put forth at the beginning of the project.

UC SANTA CRUZ
Baskin Engineering

Capstone Project

Distributed Performance Monitoring

Jesse Cheun, Jeffrey Petersen, Cristian Lewczyk, Wallace Luk
Project Alumni: Ephraim Chu, Lanjing Zhang, Steven Chou

xactly®
Inspire Performance™

Abstract

Xactly is a sales performance management solution that is offered in a multi-tenant Software-as-a-Service (SaaS). Specifically, Xactly services includes payroll, commission and incentive calculations for enterprise customers.

Xactly's clients need timely and accurate services, which requires any problems in Xactly's application server network to be quickly identified and rectified.

To make identifying problems and troubleshooting the server network easier, we helped collect and visualize a range of server and application metrics. Metrics are collected into a time series database and then displayed on interactive and comprehensive real time dashboards.

Tools

Open Source Tools
- Grafana - Visual Dashboards for Time Series Data
- OpenTSDB - Distributed, Scalable Time Series Database
- HBase - Non-Relational Database on HDFS (Hadoop Distributed File System)
- FlotCharts - Javascript Charting Library

Closed Source Tool
- Oracle Database

Approach

Setup
- A single node HBase cluster is deployed to store time series metrics.
- An instance of OpenTSDB is collocated within the HBase cluster to handle requests to the database.
- TCollectors are installed on servers to collect and push local system metrics to OpenTSDB.
- Grafana web server is installed on a server to host the Dashboard UI.

Oracle DB Collector
Xactly has application data stored in Oracle Databases. We have written a TCollector script to transform data from Oracle DB into time series metrics which are pushed to OpenTSDB. Being able to graph application metrics allows better monitoring of Xactly's applications and services.

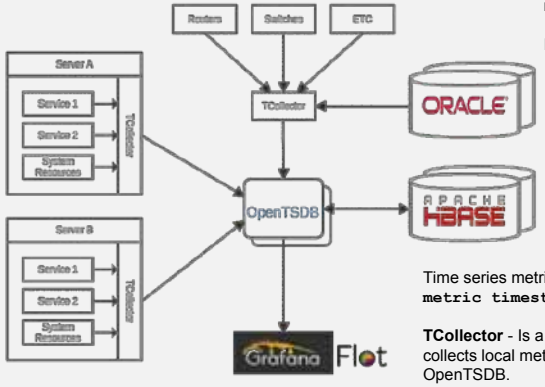
Grafana Gantt Chart
We implemented an option to display time series data as Gantt Charts, which wasn't a native feature in Grafana. Grafana uses the Flot Charts charting library to display graphs, which came with a handy plugin that made Gantt chart implementation easy. We added a few additional features such as displaying metadata from the data points, and the ability to filter by a threshold.

Overview

Distributed server networks have many points of failure and are difficult to effectively troubleshoot and debug; but they are necessary to provide the computational power and scalability required to provide reliable services to customers. **Xactly** has a *rapidly* expanding business that benefits hugely from distributed computation. It was our objective to collect various server metrics together into a powerful display to keep the pulse of their complex systems.

Types of metrics we collected:
- **Incentive Calculations**: Outputs from jvm tasks that calculates incentives for customers (Input Volume, Commission Count, Payment Count, Durations, server name, etc)
- **Analytics ETL**: Outputs from Extract, Transformation, Load tasks (Success/Failure Rate, Durations)
- **System Performance**: CPU Load, Memory, Network, Disk IO, Uptime

Architecture



OpenTSDB - is a scalable time series database that stores data points of metrics over time.

It has two interfaces:
- Writing - listens for incoming metric data from server via HTTP API or telnet and formats and stores the data into HBase.
- Reading - via CLI tools or HTTP API, users can retrieve time series data stored in HBase. The data can be aggregated or grouped while being retrieved from HBase.

Time series metrics come in the form of the expression:
`metric timestamp value tag [ . . . tagN]`

TCollector - Is a client side process written in Python that collects local metrics from the hosts and sends it to OpenTSDB.

HBase - OpenTSDB uses HBase as the database to store time series data in a single column. The row keys are encoded with the metric name, timestamp and tagk/v pairs as hex which allows quick reads and writes and takes advantages HBase region distribution.

```
00000150E22700000001000001
'------' '----------' '------' '------'
metric      time      tagk     tagv
```

Grafana - Retrieves time series data points within a time range from OpenTSDB and easily renders the data on a graph. Users can then create graphs and dashboards to help visualize large data and share with team members.

Acknowledgments

Thank you to everyone who worked to make this possible:
Anil Kona, Sr. Director of Engineering, Xactly Corp
Ron Rasmussen, CTO & SVP Engineering, Xactly Corp
Richard Jullig, Faculty Advisor, UC Santa Cruz
Grace Lin, Teaching Assistant, UC Santa Cruz

Analysis

The **Gantt Chart** above displays different jobs running as a series of runtimes. The x-axis represents the time, and the y-axis is the name of the task being run. This makes it easy to identify long running jobs which can be an indicator that a job has stalled.

The **Line Graphs** for memory and CPU usage are useful for identifying bottlenecks on the server. All graphs can be collected together into a single dashboard which makes it easy to draw correlations between different metrics.

You can also set an update interval in **Grafana**, which allows the graphs to be updated in real time so live server activity can be monitored.

Conclusion

Xactly now has a platform for collecting operational metrics for infrastructure components. These metrics are displayed in an interactive dashboard of graphs and charts based on time-series data. The added Gantt chart feature allows a comprehensive view of durational data.

Identifying and troubleshooting problems is simplified, as a user can view several metrics together in one dashboard. Future plans involve collecting more application metrics and pushing the platform into production.

We are very pleased to include posters for the Senior Design Projects that were done without industry sponsors. Some of these projects were instigated and/or sponsored by research at the Baskin School (e.g. CITRIS, CenSEPS), while others were created by students with the assistance of faculty mentors and TAs.

We have selected three of these projects for presentation in the program, and all were invited to display their posters that summarize their projects.

UC SANTA CRUZ
Baskin Engineering

Capstone Project

Smart Energy Analytic Disaggregation System

A.J. Ringer, Bryan Smith, Henry Crute, Pavlo Manovi

CITRIS

SEADS

Abstract

Currently, energy disaggregation monitoring research is limited by the usable bandwidth of sensors and their associated analog front-ends and digital acquisition systems. Additionally, a tight integration of an application processor, networking MAC/PHY, and analog front-end with an application in load disaggregation doesn't currently exist. The smart energy analytic disaggregation system, SEADS, has been developed to explore the costs of embedding disaggregation algorithms with various levels of spectrally rich energy data, and to create a source of data which can be integrated into a large energy event based database.

Approach

The main objective of the project is to create an Internet connected acquisition and analysis device capable of sampling high resolution data. This data is processed on the device and sent to a server or computer for further analysis and visualization. The SEADS project is currently being developed with two hardware paths which share a single core acquisition engine.

Our first hardware implementation aims to give us the hardware required to explore the networking and SAS scalability of energy data which would be produced by this system through known measures. These known measures include: RMS current, RMS voltage, power factor, apparent power, real power, reactive power, and voltage harmonics.

The second SEADS hardware path involves using the same modular components, but integrates them with a powerful embedded Linux device with the intention of testing how well known disaggregation techniques perform in an embedded setting. In contrast to the first hardware path, this system will produce data which is primarily event driven. In this event driven data system, bandwidth is saved as only information like, "oven on," or "oven off" is sent.

Figure 1 – SEAD Mockup with multi-core arm processor integrated onto a single PCB with wire hardness providing a rogowski coil and AC voltage.

Overview

Data Acquisition

A core component in both SEADS hardware designs is the analog front end. A digital acquisition unit was designed first for us to have data to bootstrap development of embeddable disaggregation algorithm, and to develop and test the analog front end which would be required in both SEAD and SEAD Light. A modular design was used such that the analog front could be swapped out in the event that it was found to be inadequate.

As four channels would be sampled at a rate of 125kHz simultaneously, with a resolution of 24-bits, a 20MHz SPI clock and a byte-wide CPU style USB FIFO buffer were used along with a PIC32 to handle sampling, buffering, CRC calculation, and transmission of data. A multi-threaded acquisition application on a GNU/Linux device running a real-time kernel was written with FTDI's D2XX libraries, as the standard kernel modules provided inadequate features and I/O latency. Below a picture of the stacked DAQ is shown.

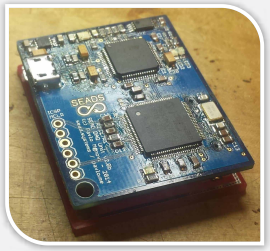


Figure 2 – SEAD digital acquisition system with microcontroller, USB FIFO, and mating analog front end.

A current transformer testing apparatus was built to test the effective bandwidth of current transformers when coupled with the digital acquisition unit. This testing apparatus created a bandwidth-limited pseudo square-wave alternating current through a resistive load to create a spectrally rich excitation current. A short-time Fourier series representation of a 60 Hz – 8 kHz sweep of the fundamental frequency of the square wave through the current transformer and the DAQ is shown below. Note that the system is designed for a bandwidth of 8-kHz in the test configuration.

SEAD

SEAD will integrate the same analog front-end as SEAD Light, and for testing purposes, the system will have a configurable signal bandwidth from DC to 32 kHz. This flexibility in the hardware will allow us explore how bandwidth and confidence in disaggregated loads are related, and to explore the limits of performance returns on disaggregated loads in relation to signal bandwidth and dynamic range. Beyond acquisition hardware flexibility, portability of code and a USB interface on our digital acquisition device allow us to test our algorithms on processors with varying processing power, e.g. Intel i7 vs ARM variants.

With this system we will integrate established disaggregation algorithms and test them against one another. If successful in creating an embedded disaggregation solution we will displays power consumption per device to user and present optional alerts when irregularities are detected.

SEAD Light

Unlike its more capable version, SEAD Light doesn't run a Linux kernel and requires a subsystem to handle internet/database interactions. To solve this lack of native connectivity, an ESP8266 802.11 module was integrated into our system with custom firmware written to handle network configuration, buffering capabilities, and HTTP interaction with our server.

Figure 3 – SEAD Light in one half of its enclosure. Board contains an HTTP stack, 802.11 physical layer, 32 bit microcontroller, AC/DC conversion and support for simultaneous sampling and transmission of 3 current and 1 voltage channel RMS measurements.

Acknowledgments

Thank you to the Center for Information Technology Research in the Interest of Society (CITRIS) for funding our project in its entirety, and to Paul Naud and Patrick Mantey for their continued support.

Analysis

SEAD light is designed to be easy to deploy en-masse, and acts as a source of data to test our web-based portable application. Designed with scalability in mind, the server stack was implemented on Django, a fast-paced web design framework. This allows for a highly documented and flexible REST API that is intended to change as the project develops. With most of the processing distributed among the connected devices, server resources can be better spent on user visualization.

Figure 4 – SEAD web application and visualization engine.

This experiment proved to us that with the high spurious free dynamic range of our sigma-delta ADC, our analog front end sufficient for our disaggregation application with most any iron core current transformer.

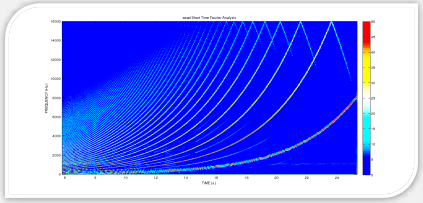


Figure 5 – Short Time Fourier Series representation of a square wave excitation of our sensor system from 1 Hz to 32 kHz.

Conclusion

Through tight integration of cutting-edge embedded technologies which allow high-speed digital-acquisition, signal-processing, and abstraction on a single device, the SEAD system can transmit events to the cloud via a low-bandwidth physical layer tolerant of frequency and prolonged outages. The SEAD devices are currently poised to improve the flow of energy information in a distribution system while maximizing scalability by limiting the required processing resources in the cloud.

UC SANTA CRUZ
Baskin Engineering

Capstone Project

Smart Wing Sail

Tyler Peterson, Eric Cao, John McCullough

SMART WING

Abstract

The purpose of this project was to prototype a modular autonomous solid wing sail unit to be used as a primary and/or assistive propulsion system for an autonomous seafaring vessel. The unit would act with fully autonomous response to its surrounding environment, continually generating for an autonomous craft the desired magnitude and direction of propulsion.

Approach

Autonomous ocean crafts require a precision propulsion system with sufficiently low power needs. By using wind energy to directly propel an ocean craft along with a physical system that can be precisely controlled, the Smart Wing Sail system is designed to meet these requirements.

The actuation of the wing sail requires much smaller actuators than a comparable cloth sail. Since conventional sails are controlled from the rear of a long boom with a center pressure combating the angle of the sail, a large force is necessary to maneuver the cloth sail. The wing-sail in comparison is aerodynamically balanced about its center and generates lift at the axis of rotation as seen in the top-down cross section of the wing in Figure 1 below.

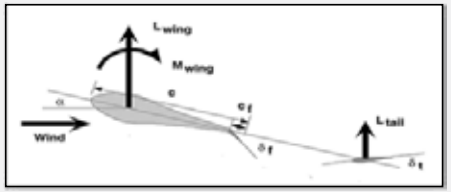


Figure1: Physics of the Wing Sail

Unlike a conventional sail, the wing sail is adjusted by actuating a much smaller tail wing. By controlling the angle of the tail wing, the angle of attack and the resulting direction and magnitude of the lifting force can be controlled through a simple balancing of moments. The tail wing requires relatively little power to actuate which saves both weight and cost in terms of the propulsion system, increasing the systems efficiency. Our goal was to develop a mechanical system using the wing sail principles and implement a control system that would allow us to precisely propel a craft.

Overview

Our Completed wing and system block diagram is shown in Figure 2. The cross section of the wing sail was designed to be symmetrical in order to provide lift from both sides, with three tail flaps and a tail wing acting as the control surfaces. The shape is optimized for a flap to chord ratio of 0.13 and for reaching optimal lift to drag ratio with a flap deflection of 45 degrees. This allows the wing to reach peak efficiency when the Lift Coefficient is greater than 1.8 . Figure 3 shows a rib cross section, including lightning and spar cap cut outs.

Figure3: Wing Rib

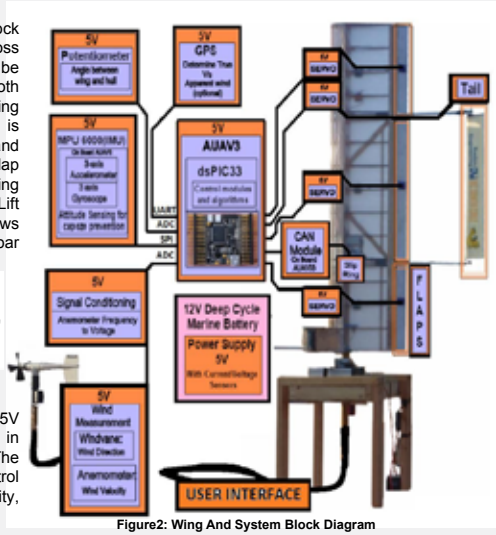
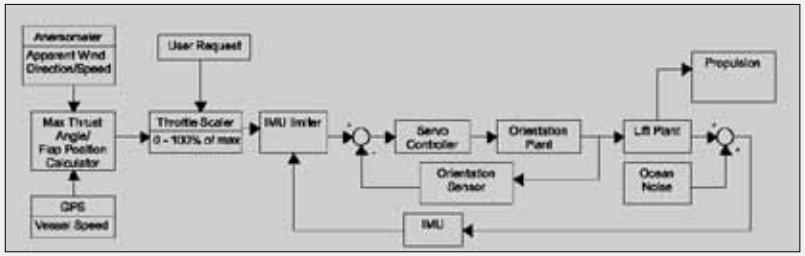


Figure2: Wing And System Block Diagram

The integrated electrical system supplies 5V regulated down from a 12V battery stored in the lower leading edge of the wing. The AUAV3 dev board was selected for the control system for its low cost, fast processing ability, and it pre includes the MPU6000 IMU,

The desired wing angle is determined through the sensor data and user inputs. The actual wing orientation is measured by the orientation encoder, which feeds back to the servo control, causing adjustments to correct for error. The combination of lift generated and noise in the form of ocean waves will affect the vessel attitude. This attitude is measured via the IMU and fed back to the servo control to prevent the vessel from capsizing if the lift generated is too extreme for the measured wind and ocean conditions.

Figure 4: Wing Sail control System



Acknowledgments

- We'd like to thank our senior design advisors, Patrick Mantey, and Jeff Bertalotto for their insight and guidance throughout this process. We would like to thank our advising professor Gabriel Elkaim for the chance to work on developing such an interesting system. Finally, we would like to thank UC Santa Cruz, Jack Baskin School of Engineering, and the BELS staff for providing the facilities and support necessary for this project.

Analysis

To verify the wing sail system would generate lift as intended, we devised a stationary test bench. By leveraging the stub mast over a 2d pivot point (simplified in Fig 4) we were able to measure the lift generated by using 2 orthogonal load cells(Fig 5) to measure the counter moments necessary to maintain static equilibrium. The wind heading required reference to the load cell axis in order to calculate total lift(Fig 6). A combination of statics, reference, and lift equation gives rise to expected load equations.(Eq1)

Fig 5:Static Moment Diagram- Idealized wind column allows for simple scaling factor.

Fig 6: Load Cells- Orthogonally brace stub mast to measure 2d leverage force.

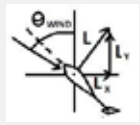


Fig 7: Wind Orientation- Lift force is combination of 2d components.

Eq1: X-axis Expected Load Equation:

$$Load_x = \cos(\theta_{wind}) * \frac{1}{2} {V_{wind}}^2 * \rho * SA_{wing} * C_{L_{wing}} * (\alpha + \alpha_d * \delta_{flap}) * \frac{D1}{D2}$$

Results

Fig 7 contrasts the expected and measured Loads for both X and Y axis sensors. The disparity between the expected and actual data is observable but the load calculation neglects the effect of drag and other factors. The analysis does suffice as a validation of expected behavior.

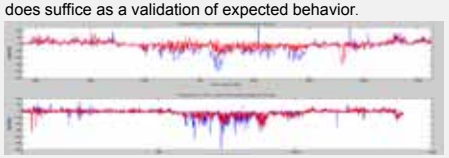


Figure 8: Expected and Actual Loads

Further analysis of our system centers around the optimization of our controller. The random nature of wind speed and wind direction requires additional design consideration and experimentation.

Conclusion

We have shown that a modular propulsion system with low power needs and precise control can be achieved with a relatively simple sensor system and mechanical design. The future applications of the wing sail are numerous and the growing need for such an autonomous vessel propulsion design can be seen in various applications which include the use of small winged vessels for unmanned oceanic research and exploration as well as the incorporation of large modular wing systems onto modern transport vessels to offset growing fuel costs.

Wing Sail on proposed shipping tanker

UC SANTA CRUZ
Baskin Engineering

Capstone Project

Low Cost Hyperspectral Imaging for UAVs

Edgar Valdez, Matthew Silva, Eric Paredes

Abstract

The goal of our senior design project is to develop a low cost Unmanned Aerial Vehicle (UAV) capable of producing hyperspectral maps designated areas. Presently, hyperspectral cameras on the market are too heavy for mounting on small UAVs and can be expensive. The project aims to use an ultra-lightweight, cost efficient Micro-electrical Mechanical System Fabry-Perot Interferometer (MEMS FPI) to replace the hyperspectral camera. The MEMS FPI can be used as a tunable filter to analyze a variety of light waves. By tuning the filter on the MEMS FPI we can capture a range of wavelengths that can replicate the results of a hyperspectral camera. The final product of the project will allow the UAV to map fields and collect humidity content, nutrient content, as well as other data, mainly, but not limited, to farmers

Figure 1: Prototype Quadcopter for preliminary testing

Figure 2: Octocopter that will be used in the final design

Approach

This project is intended to be done in two phases. The first phase is the prototype phase, where we experiment and test our equipment with a small quadcopter. This will reduce the expenses, in case of an accidental crash. Once we verify that the small autonomous quadcopter functions as desired, we advance to the next phase, which is transferring all the electronics from the small Hyperspectral Imaging UAV quadcopter to the larger octocopter. By taking two images, one with the MEMS FPI centered at 700nm and the other at 1000nm, we can generate a Normalized Difference Vegetation Index (NDVI) image. From this image, we will be able to indicate vegetation health.

Overview

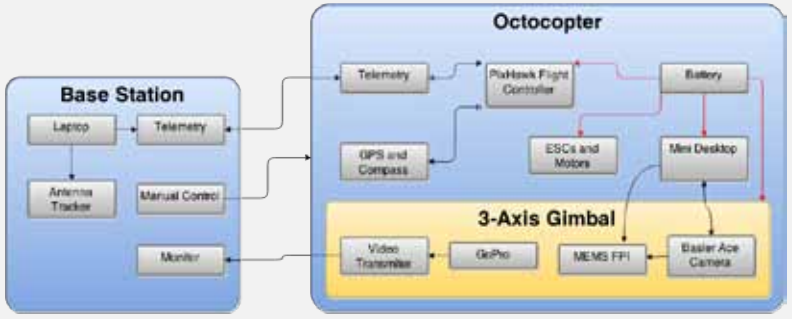


Figure 3: High Level System Block Diagram

Octocopter

- Piksi Real Time Kinetics (RTK) Gps for precision (433 MHz)
- 22.2 V 6s Lipo Battery with current monitor to monitor battery life
- 915 MHz telemetry between PixHawk and base Station Laptop
- 3DR compass for correct heading
- 8 electronic speed controllers and 8 brushless motors
- HP Stream Mini desktop

Figure 4: RTK GPS

Figure 5: MEMS FPI

3-Axis Gimbal

- 5.8 GHz 600mW Audio/Video Transmitter
- GoPro Hero4 Black
- Basler Ace acA1300-60gm-NIR GigE Camera
- Rikola's Microelectromechanical System Fabry Perot Interferometer (MEMS FPI)

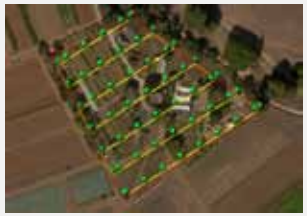


Figure 6: Survey Mission at UCSC Farm

Base Station

- Dell Inspiron Laptop with Mission Planner Installed
- 2 axis antenna tracker controlled by Pololu Maestro Servo Controller
- 915 MHz telemetry for Pixhawk communication
- 10.1" Lumenier LCD monitor for live feed
- Spektrum DX9 Remote Controller for manual control

Acknowledgments

On behalf of the UAV team, we would like to thank Professor Joel Kubby, Professor Stephen Petersen, Enes Mentese and Jeffery Bertalotto for their guidance in this project. We would also like to thank the UC Santa Cruz STEM DIVERSITY programs for providing resources and support. Lastly, we would like to thank NASA for funding the project.

Analysis

Our approach for determining a plant's health is by generating an NDVI image of photo data collected at 700nm and 1000nm. The rationale behind the NDVI derives from the ability of live green plants to absorb light in the visible (VIS) light range and reflect light in the near-infrared (NIR) range. With respect to the healthy plant, unhealthy plants absorb less light in the VIS light range and reflect less light in the NIR range.

$$NDVI = \frac{(NIR - VIS)}{(NIR + VIS)}$$

Figure 7: Illustration of the NDVI (top) and the mathematical representation of the NDVI (bottom) . Image courtesy of NASA.

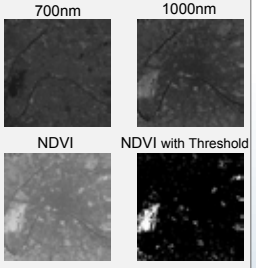


Figure 8: Satellite images of Paris with a 700nm filter (top left), 1000nm filter (top right) and the generated NDVI (bottom left) and an image with a preset threshold (bottom right) to exemplify the healthy areas.

Results

Presently we are capable of collecting images at different wavelengths using the MEMS FPI and the Basler ace camera. With this, we can generate NDVI images as shown below. We are also capable of collecting images while the UAV is in flight and collecting images using the GoPro camera. These images are image stitched with professional software, as shown below.

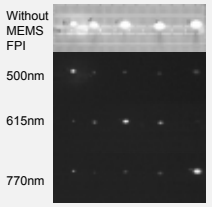


Figure 9: LEDs of different frequencies with corresponding MEMS FPI air gap value on the left.

Figure 10: Aerial images of the UC Santa Cruz Farm, stitched together

Conclusion

The MEMS Fabry Perot Interferometer is a novel device that is capable of replicating the data a hyperspectral camera can produce. The device is still in it's early stages, but the applications for the device are limitless.

soe.ucsc.edu

Email us:
fhowley@ucsc.edu

Join us on Facebook:
facebook.com/BaskinSchoolofEngineering

Patrick Mantey
Associate Dean, Industry Programs
CITRIS Campus Director
Director of ITI
Jack Baskin Endowed Professor, Computer Engineering
mantey@soe.ucsc.edu

Tim Bensch
Director of Corporate Development
tbensch@ucsc.edu

Frank Howley
Senior Director of Corporate Development
fhowley@ucsc.edu

UNIVERSITY OF CALIFORNIA SANTA CRUZ 1965–2015
THE ORIGINAL AUTHORITY ON QUESTIONING AUTHORITY

Baskin Engineering UC SANTA CRUZ