Team EcoData
CS Department
University of Idaho
May 10, 2012

Alex Fremier
Associate Professor
College of Natural Resources
University of Idaho

Letter of Transmittal

To: Alex Fremier and other interested parties

Subject: Hatch data management tool for the College of Natural Resources.

Purpose: This final report discusses the outcome, design, and development of the Hatch data management project. The goal of the Hatch project is to introduce new approaches to organizing data and making it more searchable in order to address the challenges of managing data in a scientific research environment. The Hatch project creates the foundation for a tool that allows users to upload, search, format, and visualize their data in ways they didn't think of before. This report highlights some of the problems with computation and data management in the current research environment, and makes suggestions for new approaches. It talks about specific algorithms for searching data, and standards for formatting and storing it.

Background: The amount of data being stored in scientific databases today is growing exponentially. The amount of available computational power, however, is no longer growing exponentially. The consequence is an gap opening between data and results. Researchers are trying to build and rebuild their computational infrastructure, to try to get computational growth back to the pace of data growth. But in this report, we suggest a complimentary approach to computing research data. The Hatch project comes from the idea that a significant amount of research data is redundant or not wanted. But it is also vast and currently unmanagable. Researchers end up wasting computation time on data points and sets that they do not care about, at the expense of ones they do care about.

Preliminary Work: With the experience of the authors of this report, there is more than 8 years of work in computer science, bioinformatics, and natural resources. There are over 3 years of work in computing clusters for the University of Idaho Initiative for Bioinformatics and Evolutionary STudies (IBEST), as well as 1.5 years of research in wildlife management.

By making use of these experiences, the EcoData team was able to create an extensible prototype for sophisticated scientific data management.

Team EcoData

"Hatch" Data Management Tool

Mike Solomon
Colby Blair
Computer Science Undergraduates
University of Idaho Computer Science Department
CS 481 Capstone Project

Spring 2012

Contents

1 Executive Summary **1**

2 Background **2**

3 Problem Definition **3**

4 Objectives **4**

5 Design Selection and Alternatives **6**
5.1 Database 6
5.1.1 Introduction 6
5.1.2 Relation Databases: per-document table creations 8
5.1.3 Relational Databases: tables for each datatype 8
5.1.4 CouchDB 9
5.1.5 Hatch Database 10
5.2 Search 12
5.2.1 Views 13
5.3 Visualization 14

6 Future Work **16**
6.1 New visualization types 16
6.2 New data manipulation methods 16
6.3 Performance enhancements 17
6.4 Cross-instance replication and sharing 17

7 Conclusion **18**

List of Figures

1 This proposal software's application domain in DE-CRRP 4
2 A database representing for PTAGIS data 6
3 A database representation for DNA data 6
4 The SQL syntax for creating the table in Figure 2 8
5 Document table as a lookup table 9
6 The Hatch relational - non-relational database hybrid 11
7 Typical representation of document data in CouchDB / JSON 12
8 The search interface 12
9 CoucbDB search through its internal binary tree. 14
10 A visualization example. 15

1 Executive Summary

Large amounts of data are often collected for modern scientific projects. Once this data is collected, it must be stored in an accessible location with easy access to tools for information management. Centralized databases often provide storage and access, but do not provide easy-to-use data management tools or basic data exploration such as graphing and visualization.

Our primary target is easing the management of large sets of ecological data from projects like PTAGIS, which collects information on tagged fish in the Columbia Basin. Large data sets such as these require large amounts of processing. There is a widening gap between the supply of data and our ability to process it [1].

Clearly, the inability to process scientific data weakens research projects. The Hatch project attempts to give researchers the means to visualize and explore their data before attempting comprehensive processing and testing. The ability to do this from a remote location via a web interface, combined with an inherently scalable design, means that a central processing location can be scaled transparently to allow more complex visualizations, data transformations, and forms of scientific analysis.

This tool is designed to work with general data sets, so any project that needs to filter, visualize, or transform data can make use of it. It has been built to support ecological data sets without specifically adding functionality that will not apply to other types of data.

2 Background

In the Columbia Basin today, millions of federal dollars are spent on PTAGIS and other systems to collect environmental data. This data includes fish location, ecological community composition, and abiotic data. Yet, the project results from most research are far from concrete. Despite the lack of understandable results, important decisions that affect the local environment and economy have to be made. Decisions like whether or not to conduct major habitat restoration projects are sometimes made without convincing data to suport them.

The Columbia Basin is just one small example. Bioinformatics is another data intensive field that generates more data that it can process. It is estimated that less than 10% of the data collected by bioinformatic researchers at the University of Idaho actually makes it through processing [2]. The rest has to be filtered as best as can be managed, and the low value data trimmed out. The problem with 10% data use is that not just the fat, but the meat and bone has to be cut away. Either significantly more data must be analyzed, or significantly less should be collected. At the very least, storing the data in an easily readable format can show where the gaps are. So far, our experience and research shows similar shortfalls in data analysis in the Columbia Basin.

There is a clear need for a tool that simplifies data management. Specifically, a tool is needed that allows users to work remotely with data, perform basic manipulations and filters, visualize data points, and explore data. We expect that such a tool would not only simplify the scientific process for researchers with large data sets, but allow the processing of a larger percentage of relevant data.

3 Problem Definition

One of the biggest problems researchers in the Columbia Basin have is moving their data somewhere meaningful. Central databases like PTAGIS offer a central storage and clients can push data there, but they don't offer useful tools for managing data. Once the data is pushed, it is hard to access and manipulate. Researchers are often in remote locations, and have low bandwidth connections, but would benefit greatly from a centralized location with built-in data management and exploration capabilities. Creating a robust tool will guarantee ease of use for the data, no matter the location.

Many researchers have significant data management tasks before even thinking about pushing data to the cloud. Once they are ready, they need seamless ways to synchronize data to and from the cloud. They also need to query their data, and filter it into small subsets. Most researchers don't have time to learn new programming languages or interfaces. They need a simple data management tool that has an intuitive user interface and fits their use cases. Once they have created a data subset, they will want to share it, save it, copy it, and compare it with other data sets. Getting the right data to the right place in the right amount of time is crucial.

In this interest, a data management tool needs to be written. It must meet the following requirements:

- Provide:
  - Reliable data storage
  - Basic data manipulations
  - Simple visualizations
- Have a gentle learning curve
- Allow remote access
- Allow local access
- Support many data sets (i.e., not just PTAGIS data)

4 Objectives

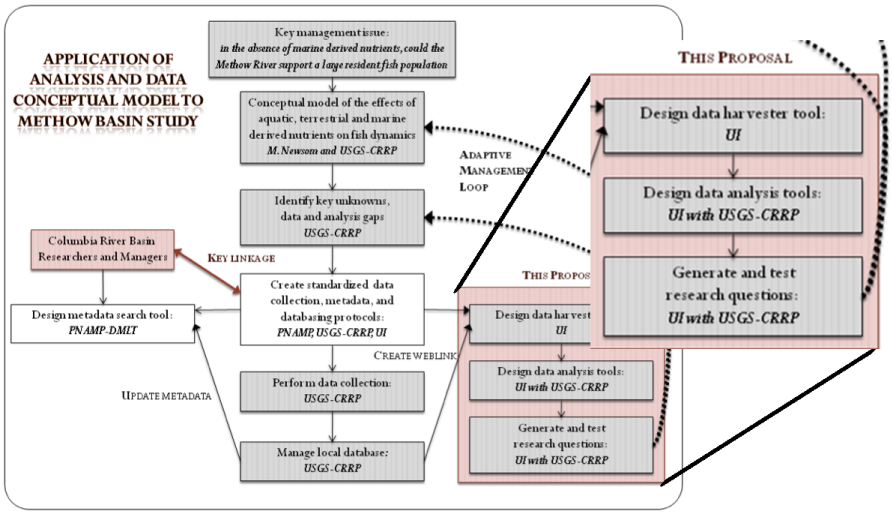

Figure 1: This proposal software's application domain in DE-CRRP

The proposed project will implement the UI data harvester portion of the grant funded USGS-CRRP project (Figure 1), directed by Alex Fremier of the UI College of Natural Resources. The data management tool will use a web-based GUI that can be installed locally on the clients' computers, but may also be accessed remotely. The tool will contain a expandable meta data server that can be connected to others to form a replicated data storage system. It will use an existing RESTful networking protocol that will allow for many different data formats to be synchronized across the cloud. The project will also have a graphical, web-based interface for simplified data management.

The replicated in-cloud model allows some significant advantages to researchers. They can see exactly what their data looks like before they submit it, due to the integrated visualizations. They can filter out bad data before it consumes bandwidth, and can retract undesirable data from the cloud, even after synchronizing it. It also frees them from central storage service management and fees, and encourages internode and inter-research commu-

nication. The tool can be set up on multiple hosts, and can be used for the benefits of the centralized cloud model. Each client can decide what topology suites them best.

This model uses a distributed database model. Distributed databases increase availability and reliability through automatic replication. They are more easily expandable, and can better protect from data loss from local disasters or malicious attacks. Moving data to where it is in highest demand also increases query performance. Offloading archive data to remote site with more resources preserves local resources. Replicated datasets can guarantee better availability. By staging data locally and filtering it before allowing it to be exchanged, the autonomy of the organization is better preserved, and the relevance of uploaded data can be improved.

The network protocol will allow incremental synchronization of data from host to host, even in less reliable environments. The tool will create an outreach from researchers in high availability areas to those in low availability, low bandwidth areas, and back. The network protocol will have support for major data formats such as CSV, and allow users to send incremental pieces of the data.

The intuitive interface will allow users to sort and manipulate their data, needing only basic knowledge of computers (and naturally, the data in question). Simple queries will create data subsets that users can bring to a workspace. The tool will be able to sort data however the user likes, on the fly. It will then be able to graph the ranges in the subset in most ways the user could want to sort them.

Once the user has manipulated their data subset to their satisfaction, they will be able to save instantly on the server and download it as a CSV file. The tool will also be extensible so that future analysis modules will allow them to run analysis on the local host, or remotely. These modules will be capable of running analysis jobs on other designated compute hosts, like workstations, clusters, or even Amazon's EC2. The tool will come only with basic

functionality for local analysis modules, but will be extensible for future, heavier compute options.

5 Design Selection and Alternatives

5.1 Database

5.1.1 Introduction

One of the biggest challenges with Hatch was how to organize data. Specifically, most organizations with scientific data have their own standard or format on how they store research data. Many of these organizations want to share data between each other, but they cannot decide how to merge the formats. Consider the following examples:

site	datetime	unique fish tag
TUC	02/16/06 19:08:15	3D9.1BF1E7919A
TUC	02/16/06 19:18:36	3D9.1BF1A998FA
TUC	02/17/06 18:21:03	3D9.1BF20E8FE2
...	...	...

Figure 2: A database representing for PTAGIS data

unique fish tag	DNA sequence
3D9.1BF1E7919A	ATGCTTAC...
3D9.1BF1A998FA	TTACGATC...
3D9.1BF20E8FE2	GTGGASCT...
...	...

Figure 3: A database representation for DNA data

In each of the examples above, the data is represented with **rows** and **columns**, much the same way someone would represent the data in a spreadsheet, such as Microsoft Excel.

These structures in a typical relational database (MySQL, etc). are called tables.

The above examples are simplifications of the rows and columns in actual research data, but they highlight one of the biggest issues with data storage using relational databases: they require you to know the column names ahead of time. Not only that, but they require that you know the data types of the values that go in those columns, and once a table is created expecting a certain format, it is hard to change.

The problem with needing to know the structure of research data before designing databases is that research data is semi-structured at best. Once it does represent some structure, it often changes. For example, once researchers finally decide what columns and data types should go in the table in Figure 2, another researcher may suggest more columns that should go in to the table.

This leads to endless edits to the database and program design by some software developer. The standard table format that everyone can agree on isn't useful to many researchers, because it usually leaves out many other needed columns and fields, or includes many irrelevant fields.

A better approach is needed. Researchers, not committees, should decide how to store data. Data should be mergable based on common values in different tables (like the unique fish tag column in Figures 2 and 3. The person who enters the data should decide how one particular dataset is stored in a database, and should be able to choose to store the same data in a different table format as they choose. There should be a simple tool that helps them do this.

The following sections describe different approaches to implementing a database design that enables data storage for dynamic or semi-structured data.

5.1.2 Relation Databases: per-document table creations

This approach is the simplest and follows the concept of table creation for data sets pretty closely. Basically, for each input document in the form of a spreadsheet, a new SQL table is created. The columns names and type are determined from the headers and data values in the spreadsheet.

```
CREATE TABLE ptagis_doc1
        (
                id int,
                site char(50),
                read_data_time date,
                tag char(50)
        );
```

Figure 4: The SQL syntax for creating the table in Figure 2

The biggest problem with this approach is that each document in the database is a table. When searching for a specific document, the database typically searches for the table name. This search is linear, and with hundreds, thousands, or hundreds of thousands of documents, frequently searching the database to look for values would be increasingly slow and therefore useless.

Another problem with this design is that building software to support this would be difficult and complicated, since it is not regarded as a good practice.

5.1.3 Relational Databases: tables for each datatype

Another approach is to create column tables for each data type, and let document tables just be collections of columns. Each of the document column values point to respective values in the column tables.

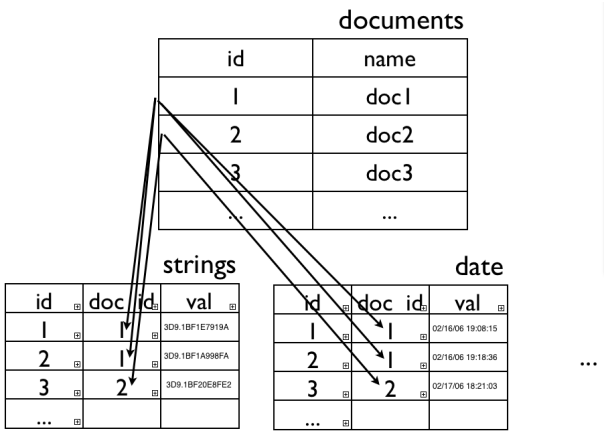

Figure 5: Document table as a lookup table

This allows for documents to have a dynamic number of columns with variable data types, but there are two problems with this approach. First, every value of a given type in every document in the database is put into one table (e.g. all values with a 'string' data type go into the 'string' table). With potential millions of data values, each table becomes an overflowing bucket and doesn't utilize the advantages of storing multiple columns and values in one table. Searches for data would require lots of filtering for just the data required from specific doucments and would therefore be inefficient.

The other issue is that every retrieval of data from a document would require many lookups. Data retrieval over significantly large data sets would quickly become very computationally intensive, and eventually impractical.

5.1.4 CouchDB

When one thinks about the fundamental issues with storing, searching, and merging research data, a core issue is identified: data is semi-structured. This is what makes trying to use relational databases so hard. They were made for datasets where you knew the structure up front, and seldom wanted to change their structures.

The one assumption that Hatch makes about the data is that **there are rows and columns**. This is the only assumption Hatch makes. This leaves the database and interface designs free from whatever changes are needed by the users.

This is done by storing data in JSON format. The data is this lightweight format, modeled exactly like Ruby on Rails 3 returns records from its Active Record. This allows users to input data however they like, define delimiters for columns using Hatch Input Filters, and Hatch does the rest. It finds the most specific data type the values can be stored in, skips non-matching entries, and populates all the forms on the web pages according to the data, all without dictating the structure of the data, or even knowing it before hand.

This leads to the technical implementation of the Hatch Database.

5.1.5 Hatch Database

Since Hatch uses Ruby on Rails, there are lots of tools and libraries for using the standard relational database, via Rails' Active Record. In many/most cases, Hatch actually wants to stick to the relational database. With Hatch's internal database structure, order is important. For example, a user will always have a name, email address, etc. A document will always have a name and an owner. However, the data in the document is the semi-structured data, and Hatch only wants to use CouchDB for that for the reasons described above.

The result is a relational–non-relational database hybrid. Hatch uses traditional relational database columns when the columns are fixed and known in advance (as is often the case for information such as user names and e-mails), and can add columns to a database entity (called a scaffold) on the fly using CouchDB. For example, we create a scaffold called Documents. Document always have a name, id, and collection/folder they belong to. But, documents may have a "data" section, or they may not. That data section may have arbitrary numbers of columns and rows of data that would match the flat file they came from

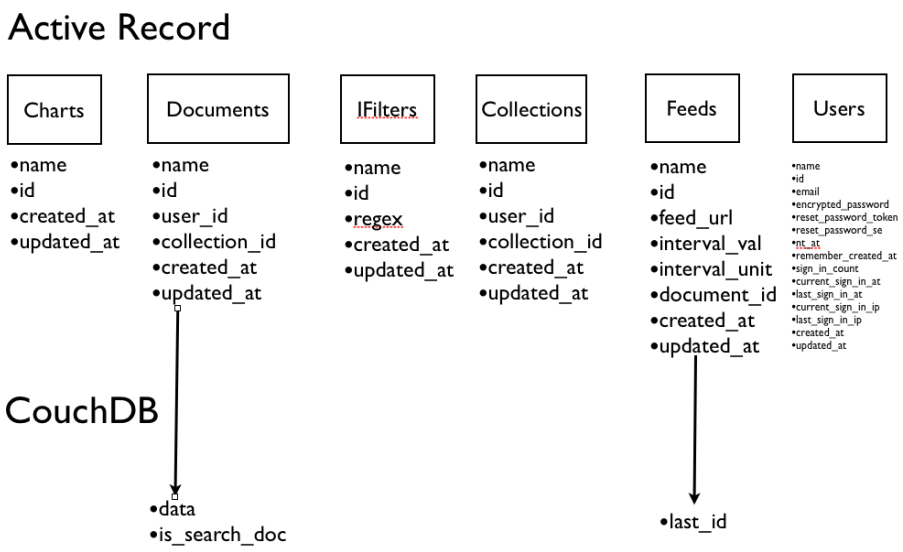

Figure 6: The Hatch relational - non-relational database hybrid

(like an Excel file). The document may instead have any other data that can be stored in JSON format. It is up to the user of the Hatch interface, not the database, to decide. By allowing Hatch to infer the structure and fields of data, Hatch is by extension allowing the user to decide how to format data.

Most Rails applications that use CouchDB completely replace Active Record with some library's version of it, like couchrest's Active Model. However, by replacing Active Record, you lose support for libraries that lots of Rails developers make, like pagination package `will_paginate`.

Hatch gets around this by using its database hybrid through a Ruby library called stuffing. Stuffing is a link that ties Rails Active Record records with CouchDB documents. It allows for rapid prototyping in development, and is a nice, modest alternative to replacing the internals of Active Record. Since the base model for database scaffolds is still Active Record, the huge amount of Rails Active Record based libraries still work.

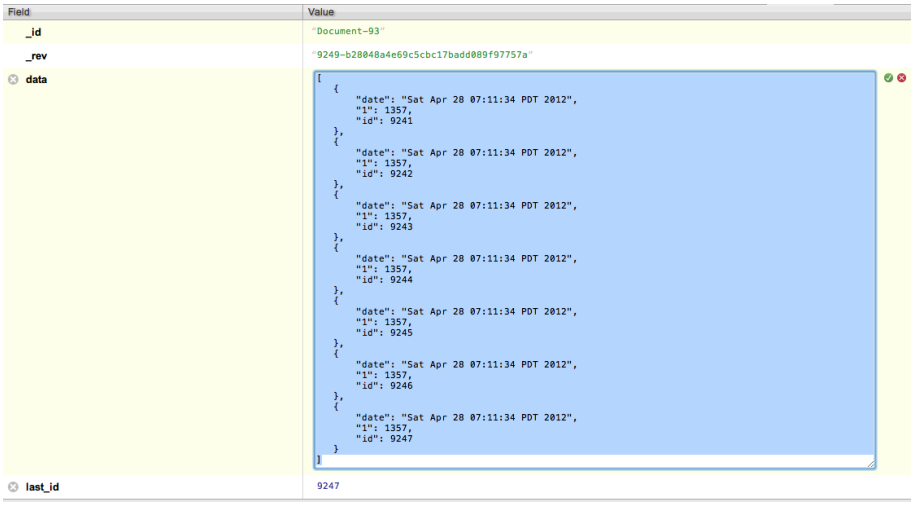

Figure 7: Typical representation of document data in CouchDB / JSON

5.2 Search

After the EcoData team addressed the issues with storing semi-structured data, the next big problem to address was how to efficiently search through the data. The interface that was desired was one much like Google's search; a simple search box, with two buttons. The interface needs to be extremely simple yet powerful for it to be effective for non-technical users.

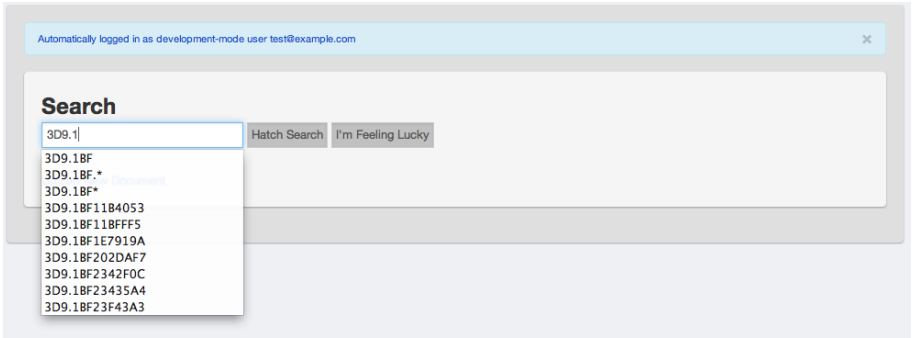

Figure 8: The search interface

For example, when a user entered in a search like '3D9.1', it would be assumed that any data starting with '3D9.1' should be returned. Because Hatch assumes that the user would want any data in the same row as the matching data, it returns the entire row.

For search, this could lead to a huge number of string comparisons in order to find all values that match. This would make search impractically slow. Luckily, CouchDB allows us to implement a practical solution to this problem.

5.2.1 Views

CouchDB uses precompiled queries called views. Views take developer-defined query templates and apply them to every document that is created or updated in the database when the document is saved. The results are precompiled lookup tables (actually heaps/binary trees), which make searches fast.

Hatch basically creates a view like the following pseudocode:

```
for each document
        for each row
                for each column value in row
                        emit(value, row)
```

`emit()` is a function that tells CouchDB how to create the search B-Tree. It takes two arguments; the key that the tree node will take, and the value that the node will return if the key matches the search. Hatch says 'every column value in a document is a key, and the return value is the data row it belongs to,' so if a search matches a document value, the search results return the entire data row.

CouchDB makes Hatch's job easier by having internal methods for string matching. For example, if a search for '3D9.1' is used with CouchDB startkey, CouchDB will return any string starting with '3D9.1'. Hatch doesn't have to invent a query language to tell the

database lots of parameters for matching values, which means users do not need to learn a new query language either.

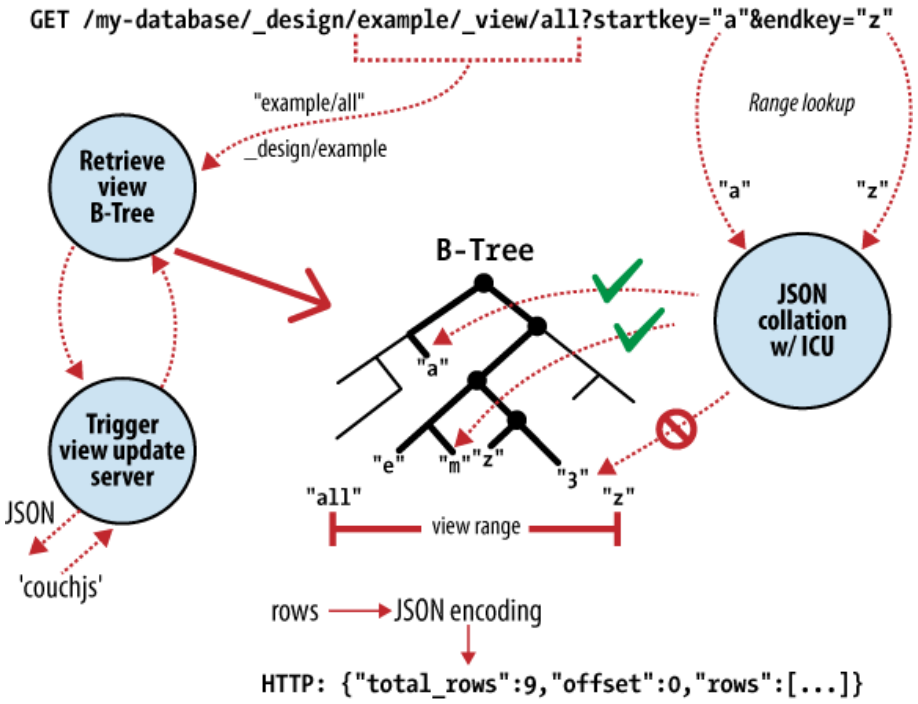

Figure 9: CoucbDB search through its internal binary tree.

The other advantage to CouchDB is that it stores values in the B-Tree structure. For string data types, this means that the database searches only within the '3D9.1' string range in the tree. When the search sees a child node starting with '3D9.0', it instead picks the child node staring with '3D9.1', and the entire '3D9.0' group is never searched through, which makes searches complete in a short time frame.

5.3 Visualization

When users are searching their data, it is important to get rapid feedback on the meaning, shape, and size of the data they have found. An intuitive method for this is through charts, graphs, and other visualizations.

Hatch's visualizations allow users to accomplish these tasks. They allow users to see unique data and graph values in comparison with other values. The Hatch Documents interface allow users to do things like categorize data and then use visualizations to graph the results.

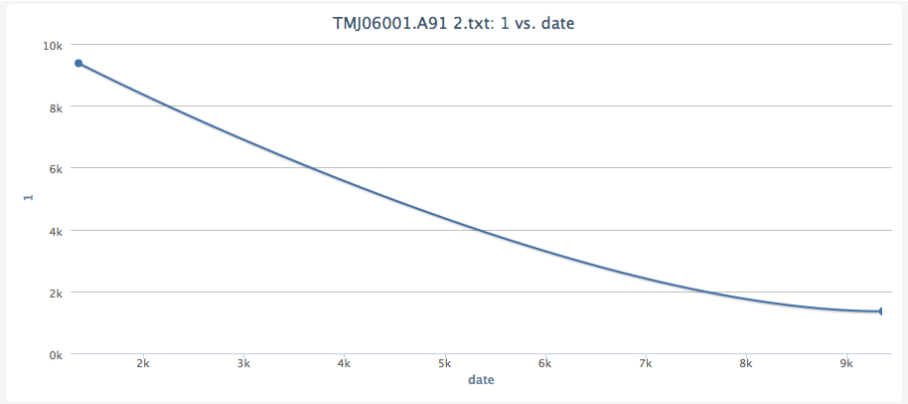

Figure 10: A visualization example.

Hatch uses HighCharts, a JavaScript library, to perform visualizations. It allows Hatch to incrementally stream new graph data to the client, instead of resending all the graph data. This is useful when users are working on low bandwidth connections, or with large datasets. By reducing unneeded data transfer, Hatch is able to work with larger datasets faster.

Hatch is able to check incrementally for new data. When data is graphed, it points towards the data in the document that needs visualization. If the data in the document changes, the graph adds the data points to the graph without refreshing the page.

This opens up the possibility of Feeds. Feeds are scheduled events in Hatch that pull JSON data from somewhere on the web. They push the data in to the document they point to. This data can simultaneously be used by a visualization in order to immediately graph the new data, as soon as a change is detected in the document. The result is Live Charts, and documents that sync with data on the web when new data is available. Instead of doing

a bulky request for data all at once, data can be taken in smaller requests, and Hatch can give users a real-time view their data as it comes into the system

6 Future Work

Hatch has been specifically designed to be extensible and modular. Because the needs of the customer are so wide-ranging and would require a larger effort than our small development team could reasonably complete in this time frame, Hatch has been constructed with future extension in mind since day one.

Several major areas exist which could readily be improved or extended. Included are a few of these areas along with brief descriptions and time estimates, based on a team of two working about 15 hours per week each. Many other parts of Hatch could be extended, but these are some of the most important areas for scientific research.

6.1 New visualization types

Currently, Hatch only fully supports single-series spline, line, and scatterplot charts. Most of the code exists to extend this to categorical types and the corresponding charts, such as pie and bar charts. In addition, supporting multiple-series data on a single chart is another obvious way to improve Hatch.

Time estimate: 1 week per chart type; 2 weeks for multiple-series data

6.2 New data manipulation methods

Hatch supports basic data manipulations such as filtering, sorting, and bringing together columns from different data sets. Obvious and useful extensions include single-step joins

between different data sets, log transforms, summary columns, and suggested data joins. These would form the basis for a more sophisticated data manipulation pipeline that could be separated into a distinct package, if desired.

Time estimate: 2 weeks per manipulation

6.3 Performance enhancements

Some areas of Hatch are not as responsive as could be. While performance is not crippling to the software, it can occasionally take a few seconds to complete an action such as a search. Some of this can be solved through more powerful hardware (most tests were run on aging laptops), but algorithmic improvements as well as general optimization could improve responsiveness greatly.

Time estimate: 1 month to double performance (more if other features are added)

6.4 Cross-instance replication and sharing

The original vision for Hatch included a system where each researcher could run a localized Hatch instance and use the fully functional software on their own machine. Later, this data could be shared with a centralized Hatch instance, or shared with other researcher's Hatch instances. This requires solutions for several difficult but solvable problems, and would require extensive testing.

Time estimate: 5 months

7 Conclusion

The Hatch project attempted to solve several difficult problems in an easy-to-use package. Due to time and resource constraints, Hatch must be seen as the beginning of a comprehensive data management system that Hatch is positioned to grow into over time through future efforts.

The developers followed accepted design patterns and consciously constructed the software to be easily extended in every area. Since it is likely this software will be developed further in the future, a considerable amount of effort was spent ensuring that each piece could be replaced or extended as necessary. Although the project does not meet every need the customer has, it accomplished the goal of either meeting crucial needs or putting most of the pieces in place to allow the straightforward extension of Hatch to meet them in the future.

Hatch defines a new model for aggregating, storing, and searching data. It embraces RESTful design and the Don't Repeat Yourself mantra. Hatch decouples design from data at all times; something that is not too common in scientific research. As a result, the features that are added to Hatch will continue to support any kind of data, as long as that data follows the core assumption Hatch makes: that data is in rows and columns. Hatch embraces JSON as much as possible, providing JSON views whenever possible. Feeds sync with JSON data on the web, and stores data in CouchDB's native JSON format. The result is embracing open data exchange through the web in one of the most ubiquitous and universal formats.

Although the certain features are not yet implemented, the essential core has been demonstrated to work as desired, and the future functionality of Hatch is only limited by developers and ideas.

References Cited

[1] Manek Dubash (2005-04-13). "Moore's Law is dead, says Gordon Moore". *Techworld*. Retrieved 2006-06-24

[2] Foster, James. *Visualizing Human Microbiome Ecosystems*. University of Idaho: Computer Science Colloquium, December 7th 2010. Seminar.