# Web Analytics with $N_8$stats

*Delivering desired content, better!*

By: Nathan J. Woods

## Abstract

The Web provides businesses the opportunity to present and market themselves as never before. Given the ability to present almost unlimited information, a key question is how should that information be organized. $N_8$stats is a web analytics system that I've designed to provide feedback on website structure and design in real-time. In particular, I explore the use of regression analysis to determine the correlation between time spent browsing a web page and page word count as a measure of whether users are reading content on a page. Markov chain analysis is also used to determine whether or not users are clicking through a particular page to reach content on a second or third page. This project walks through the development of the $N_8$stats tool and an analysis of the Carroll College web site using data collected during Fall 2012.

# Table of Contents

# Introduction

In an ideal world, all information would be presented in a simple manner that easily conveys the desired information. Attempts have been made to make this happen in nearly every aspect of our lives. Wikipedia seeks to convey the world's most complicated topics from only a few keystrokes away. Netflix wishes to give every customer a wide selection of movies for instant entertainment, and many more companies wish to offer their services to the public in the most efficient way possible.

But how can a business know if customers are using their services and information in the most efficient manner and whether the customers are satisfied with their experience? Most businesses use advanced statistical methods, surveys and occasionally an analyst or consultant to determine how well they are offering services. Some, typically larger, companies are able to fund such a group, but for the majority of small businesses, this desire slips away unnoticed or gets neglected due to a large initial cost or the complicated process of analysing gathered information. Some businesses choose to neglect the customer altogether or simply provide the information that they believe the customer wants.

While this problem is common among all business models, it appears more frequently in online business and web applications. Yet online business should have no problem gathering statistics, as most servers already gather traffic logs and other useful analytics data. Some companies go through the effort of using third party solutions to gather analytics for them. So gathering the data must not be the problem. The problem must be in the presentation and analyzing of the gathered data along with how to use the data to depict user's actions.

But what kind of reports do businesses want to see? Currently analytics systems show information like the number of visits, unique visits, average time on page, and where visitors come from. While these reports are informative, what is the true information users desire? Most likely answers to questions like, "How effective is my navigation structure?" or "Do I really need two pages for this content?" Unfortunately, these questions are much more difficult for an autonomous analytics system to answer. A greater, in-depth analysis along with specialized knowledge of a website is required to answer such questions. Put simply, the data is already there but nobody is asking the right kind of questions.

Introducing $N_8$stats: a web analytics system seeking answers to the harder questions, providing feedback on site structure and providing the information in a simple to understand manner. With this analytics system, webmasters should be able to deliver desired content the best way possible.

## Project Lifecycles

In any project, there exists a lifecycle that shows how the project moves from idea to production. To many, this lifecycle is linear, but the best projects go through a cyclic life cycle. Many models exist that explain how this process works: For example, a manager may see a project's lifecycle as: Design, Implementation, Testing, Evaluation and Analysis (Web).

Unfortunately, most managers choose to neglect evaluation and analysis because if customers do not complain, then obviously the customer is happy. These same managers also end up getting tired of projects and systems implemented and simply choose to redo a project because something has to change. Sometimes, that may be the right answer, but without the evaluation and analysis of a project, there is no way to tell if a simple change is all that was needed, or if the entire projects needs retrofitting.

This is where $N_8$stats fits in the project lifecycle. $N_8$stats attempts to evaluate how well a project is doing and suggest possible changes. These changes can be thought of as sub-projects and must go through a similar life cycle. By using this development ideal, project managers can stop wasting time recreating the wheel and spend more time improving what already exists based on what the users want.

## $N_8$stats' goal

The goal of $N_8$stats is to answer the harder questions in an autonomous manner. That is, the analytics system should require as little input from humans as possible. For example: instead of having the system ask if two pages are related, the system could assume they are if they have similar keyword signatures. This can be accomplished with algorithms that use crawled text and site statistics as inputs rather than human interaction.

Additionally, information should be presented in a timely manner, without requiring lengthy analysis periods. Some analytics systems today require that a large amount of historical data exist before summarizing any information about a website. Also, many analytics are re-generated at specific time intervals, usually daily to weekly, which seems to be quite slow. $N_8$stats seeks to push reports live as information is received. This also implies that the analytics page does not need to be refreshed to see new reports.

# Gathering data

As with any analysis, finding an application that gathers data and reports information never seems to be an issue until none of the available applications has the exact reporting capability that is desired. After extensive research in the subject three similar analytics systems were found that bring desirable traits to the table, but nothing seems to be a canned package of “That’s what I want!” Using these three systems as a guide, $N_8$stats was created with the intention of being a system that took the good from each of these other analytics engines and then takes them one step further.

## Related projects

Each of the following analytics engines operate at different times in relation to the user-server interaction, creating an interesting problem in re-combining them into a single usable system. This problem is discussed in greater depth in the process section.

### Aw-stats

By analysing server access logs, this analytics system is able to produce a static page that displays to-load-time live statistics. It does a good job of simplifying the data provided into a human readable form. Unfortunately, the only input data provided is minimal access log data. This data gives little information about a user’s interaction with a web server. Thus the results leave things to be desired. Written in Perl, Aw-stats parses flat log files to generate a statistics page each time the page loads. This is a very processor intensive routine to run each time a page is loaded, possibly affecting server performance.

### HummingBird Stats

This, less commonly known analytics system, is able to show live site action data. Written in Node.js, this application is able to open a continuous connection between the analyzer and the server that is updated with new data each time a user loads a page. Each page loads a tracking pixel that contains page specific data that is entered into a database and further analyzed. While this system offers live analytics, it lacks the larger reports and referral statistics.

### Google Analytics

Google Analytics has the most informative statistics among these three systems, with outstanding features such as in-page link percentages that show how many viewers have clicked a particular link. Google does a good job of showing throughput from page to page and breaking down other information in useful ways. Reports are computed on page load and, assuming this comes from a database, it is not terribly intense on the

server, yet it is only as up-to-date as the last refresh.

## This project

$N_8$stats' goal is to combine the best of the systems described above while correcting related problems. This system is designed to provide not only a continuous connection to the monitor, but also to the client so exact time spent on a page may be gathered along with popular mouse locations on a page. This system will use advanced database transversal techniques that allow fast inserts along with quick selects and report pulls, all built on top of an optimized minimalist storage system allowing for the most efficient database storage and interaction possible.

This system will also use a non-blocking programing language (node.js) to allow many simultaneous users to be both viewing and generating data for the system. The non-blocking nature of this language allows for system calls to execute and not stop (block) future system calls from firing until there is return data. Instead, this style of programming allows the system calls to be added to a call queue and be executed when the processor is in idle. In the end this process allows faster execution times with minimal required resources.

The continuous connection to both the monitor and user can allow live usage statistics to be transferred to both the monitor and other users (if desired). If the webmaster would like to put a page statistics module on any or all of the sites' pages, the pages would show how many other people are viewing the page and other analytics information could actively be displayed on users' pages. If authenticated, a more advanced version of this module will be automatically loaded on every page while browsing the site. And finally an overall monitoring page will be automatically updated as new users come to the site.

### Gathering user data

Gathering data is the initial part of any statistics application and is overlooked in many projects. Typically data is stored in a flat file system, allowing for extremely fast inserts but requiring more intense processing for generating reports. This flat file system raises storage concerns once the dataset becomes significantly large. By repeatedly storing the same data for multiple entries in the flat file, this system wastes space.

Instead, a relational database will be designed to minimize the total storage space required for the data. By using a relational database, insert times are increased but storage space is reduced with a result that reports can be produced faster. In order to increase insert speed, a combination of the non-blocking programming language, client stored cookies and database stored procedures will be used to minimize time spent on inserting.

### Data processing and presentation

Data processing will mainly occur on the database or within the analytics system. Simpler information can be transferred directly from a user connection to a monitor connection. More sophisticated, or moderate reports, require regularly updated requests from the database as new information is received. Advanced reports require fewer updates as these reports require complex linear algebra, regression and textual parsing. While incorporating many subsystems, all this should be executable from node.js.

Once a moderate or advanced report is created, it can be cached for future use. Using the cache, when the master analytics page loads, it can be fully populated with information and receive new reports as they are generated. This allows for live reports, semi-static reports and static reports to be viewed side by side, without over-taxing the analytics server.

## Technologies used

As with any computer based project, programs are built in layers. A complete computer system would be nearly impossible for a single person to assemble from scratch due to its complexity; so computers are assembled from modular components. Thus, to simplify the project, previously developed systems will be used to carry out various tasks for this project.

*EC2 (Elastic Cloud Compute)* is a web service offered by Amazon Web Services (AWS) that provides application developers a simple platform for scalable cloud computing applications. The basic idea is to have Amazon provide the application developer with a way to access an instance of a virtual machine that exists somewhere within their server farm. Billing is determined based on the resources used over a month. Traditional measures of usage, or metrics, would be millions of I/O requests, Gigabytes transferred out of an instance and number of instance-hours (hours in which the instance is active). With the correct configuration of an instance, an affordable and scalable application platform is within reach of anyone. (Amazon)

*Linux* is an operating system that is running on our EC2 instance. Linux has long been regarded for its high efficiency in process handling and low overhead of generic operating system processes, making it a prime choice for real time applications. Linux is licensed under the GNU General Public License and is freely available to all those who may desire such a stable operating system.

*Node.js (a.k.a. Node)* is a server side javascript programming language that is based on Google Chrome's javascript engine V8, compiled with additional system calls that allow for greater system interaction. Node provides a non-blocking language that can handle thousands of operations a second with extremely low memory impact.

*MySQL* is a database engine that is free and highly scalable for many projects; providing a cheap alternative to its close relative SQL. It is preferable to store data in a well-designed database, rather than a flat file to reduce the amount of storage space required. Indexes may also be created to further speed up processing on data.

## Process of data acquisition

Initially, when a user loads a page, a small script will be loaded from the $N_8$stats server that initiates the process. The script opens a websocket with the analytics server, which initiates a process to verify that the user is on a domain monitored by $N_8$stats. If the domain is being monitored by $N_8$stats, then the user's information can be added to the analytics system. Otherwise, the connection is dropped, reducing load on the analytics server.

Once the server recognises that the user is valid, it acknowledges the connection with the user by sending an "ack" message to the client. Upon receiving the ack message, the client's computer builds a list of useful pieces of information that is then relayed back to the server under the "ini" command, which is short for initiate data entry. This collection of information can contain anything from browser size to document title; pretty much anything accessible from JavaScript can be relayed back to the server. For version one of $N_8$stats, only the following were passed: document title, document referrer, and the values of "_browserHash" and "_sessionHash" cookies if available.

After the server receives this information, it must go through its own process to determine where data should begin to be stored. Typically, at this point, a flat file analytics system would compile all this data into a single string of text and write it to a file. As mentioned before, this structure consumes a lot of space and requires large amounts of parsing and processing to return useful reports. Thus, the database was designed to minimize the amount of redundant data. This means separating browser information from session information and separating session information from action information.

By separating the data into three different areas, we essentially create three different levels of data storage. At the top level, the most abstract level, we have the user's user-agent information. The user-agent contains data pertaining to what operating system and browser a user is using. At the middle level, we have the user's session information. Here, I chose to have the session store both location and incoming referrer information. Finally, at the lowest level, we have a user's action, which holds: a page's referrer, the page itself, and how long the user viewed a page. This stacking of information allows us to skip certain storage routines for each connection. Thus, we need to determine where to start storing information.

The problem is in attempting to minimize processing time while finding exactly where to start storing data, storing no more or less than necessary. The proposed solution, in

Figure 01, requires only two function calls in order to find where to start storing data. This design is based on a binary search pattern, proving that two function calls are the minimum to place an item.

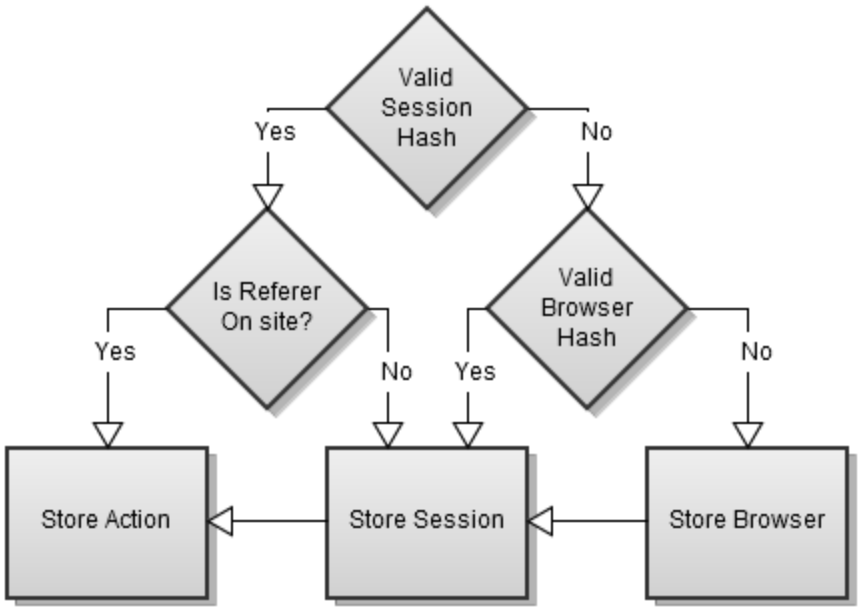


Figure 01: Proposed algorithm to find which 'level' to store a user's visiting information

When this process is complete a hashed value of the session identifier and browser identifier are sent back to the client to be stored. Then, as the user navigates through a website, the process will only store actions, instead of the whole browser-session-action combination of data.

Finally, as the user navigates away from a page, it closes the connection with the analytics server. This action causes the server to store the ending time of the page view, thus giving a time-on-page value.

# Data Analysis and Approach

For the purpose of this thesis, only the long term analyses are discussed. While other analytics are necessary for a full analytics system, many of the faster methods are trivial and can be handled by directly passing data from client to monitor. Other slightly-more complex analytics only require queries from a database after data has been gathered. The methods that follow require more analysis but provide the most useful information.

## Numerical Analysis

The purpose of numerical analysis in the context of web analytics is to find discernible patterns within data. By finding patterns we can see if users are acting as we expected

them to or if they are acting randomly. Numerical analysis techniques, such as linear regression, can be applied to almost all data generated by this analytics system as long as there is a source and effect variable (Chapra).

While any bivariate data can be analyzed, for the sake of this thesis, we will analyze the amount of time spent on a page vs. the number of words on a page (Cline). Using this analysis, we will create a metric to rate the adequacy of a site's navigation. In order to make such a metric the following two assumptions must be made.

1. Humans on average have a predictable rate at which they read words.
2. When users are "lost," they deviate from this predictable rate.

Using these two assumptions, we can tell if users are reading content or if they are not finding the content they are looking for. Thus, this metric should be able to show the effectiveness of a site's navigation by showing if users are reading content or looking for it.

## Retrieving Data

In order to obtain data to perform this analysis, two different pieces of information were required: the number of words on a page and the time spent on a particular page. These values come from two different data sources. The website's database was able to provide the number of words on a page while $N_8$stats provided average time on page. Making note that both data sets are large in size, this analysis may be a monthly exercise due to the strain on the database systems themselves.

### *Words per Page*

From my previous work study experience coding the Carroll College website backend, I have deep knowledge of the website and was given direct access to the site's database. This allowed me to transfer the necessary data to a private server and analyze it there, as not to disrupt the production database. During the data transfer, I chose to sanitize the data by removing all the HTML markup tags and special characters and ensured there was only a single space between all words. Once the data was properly sanitized, it was uploaded to the private server.

To count the words per page required some ingenuity since MySQL does not have the ability to count words natively, but does allow the replacement of characters. Character replacement gives us the ability to count the spaces in text and (by adding one) the words of a text. Taking the difference of content length and the content length without spaces gives the count of spaces in a particular text. This results in the following selection query.

```
LENGTH(content) – LENGTH(REPLACE(content, ' ', '')) + 1
```

In addition to calculating word counts, I was required to pull the uniform resource identifier (URI) of the pages; this provided the piece of information to tie both sources together.

*Time on Page*

None of the three analytics system described above had the ability to show the time a user spends on a particular page, so $N_8$stats was designed with the ability to gather this and other related information. This system was designed to maintain a persistent connection to any client's browser while the client looked at a page. As the user navigated to a page, a connection was made to a server, marking the start of a page view. As the user navigated away from a page, the connection was dropped from the server marking the end of a page view. These two timestamps were recorded along with the URI of the page visited.

Unfortunately, due to the nature of the web, substantial data sanitation was necessary to give unique page names. For instance, in the data collected from the Carroll site many duplicates such as /, /index.cc, and /INDEX.CC were prevalent. Fortunately, after sanitization, the structure of the analytics system proved extremely helpful in pulling the desired data. Pulling the URIs along with the average time spent on the unique page was a simple task despite the fact that there were over 800 pages to compute averages for and over ¼ million records to average.

*Putting it together*

Finally, it was time to combine the two data sources to get a single data set. Using the URI as the joining key, a query was designed to match the average time spent on a page and the number of words on that page. The scatter plots that follow seem to show that users are not finding what they are looking for, but further analysis is required to ensure this is the case.

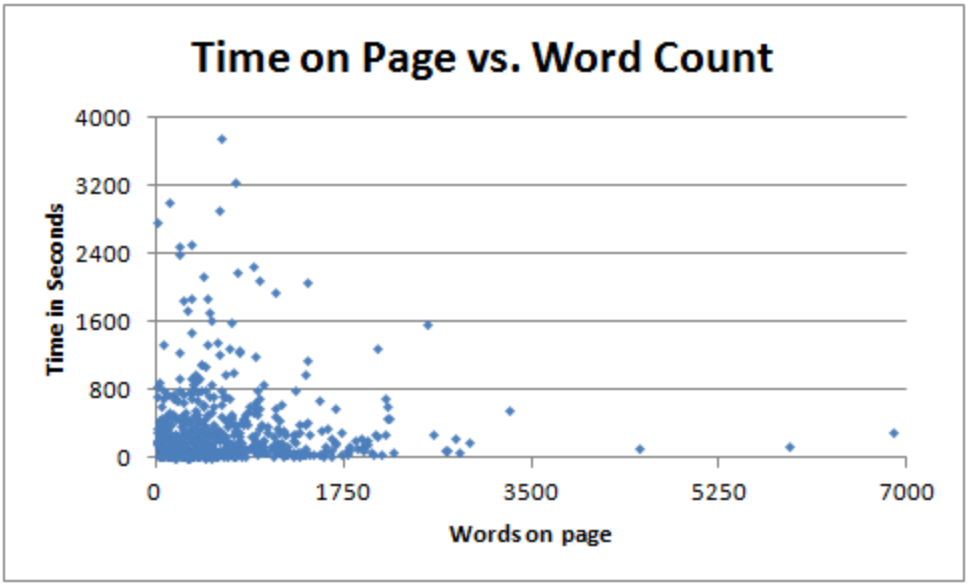

Figure 02: Overall plot of Time on Page vs. Word Count

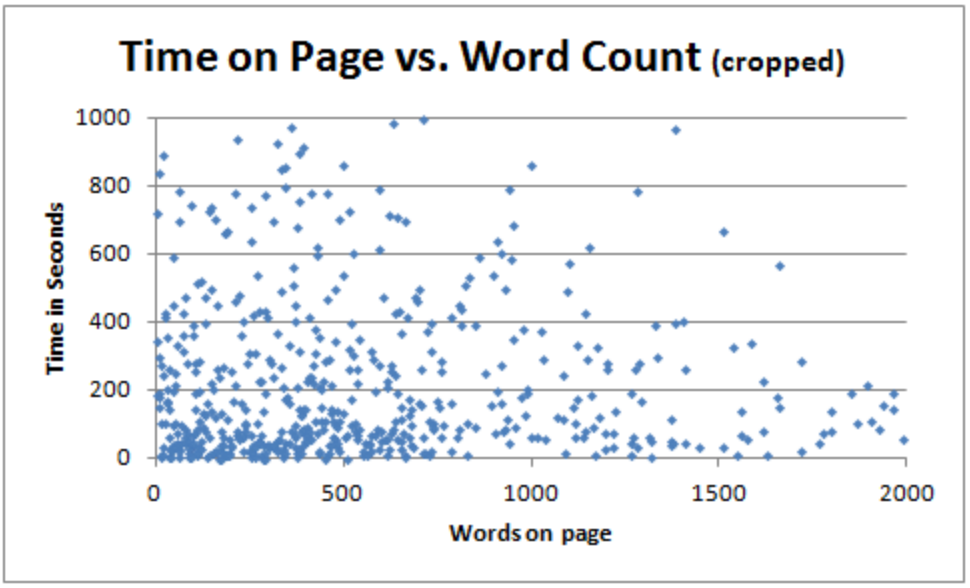


Figure 03: Cropped plot of Time on Page vs. Word Count

## Analysis

In this attempt to fit functions to data, we consider only five classes of functions. While it may not have been necessary to perform all four of the five possible base regressions for this data, it was interesting to see how the equations all point to a consistent conclusion. The final type of analysis, a trigonometric approximation, is not included here because it would make no sense in the context of the dataset. Trigonometric functions are used to represent data with periodic behavior and due to the nature of this data would be a worthless exercise.

$$\varepsilon = RSS = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

All these analyses use a method of regression to fit each function to the data. Regression aims to minimize the sum of the square of the residuals (RSS). Using a gradient search to find the best values for the parameters of $f(x)$ we can autonomously fit functions to data.

*Linear Analysis*

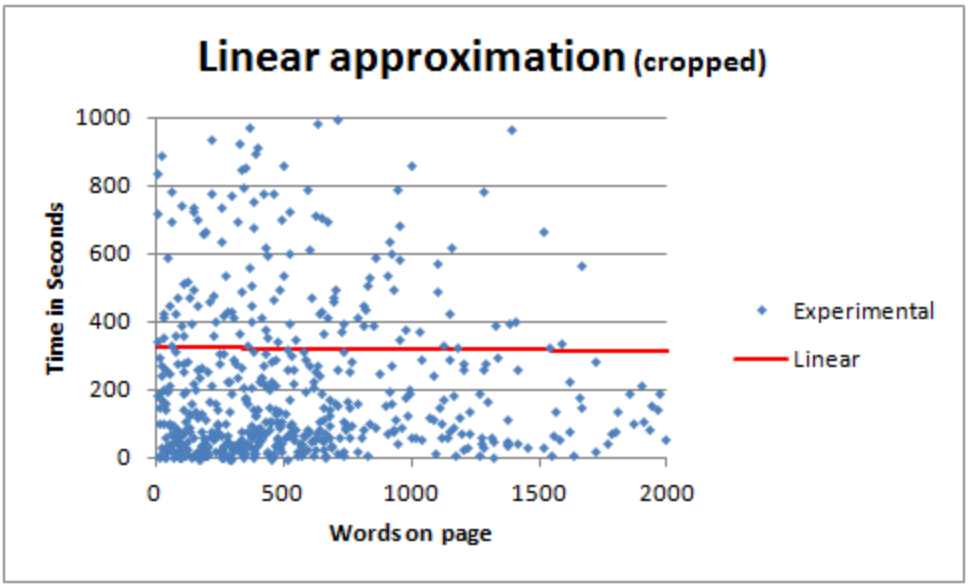


Figure 04: Cropped plot combined with the results of the linear approximation

Fitting a first order polynomial to the data will show if it contains any linear tendencies.

$$f(x) = A \cdot x + B \text{ where } A = -4.882 \cdot 10^{-3} \text{ and } B = 3.237 \cdot 10^{2} \text{ having } \varepsilon = 1.357 \cdot 10^{8}$$

This fitted equation shows that there are no linear tendencies among the data. In fact, it seems to show that the dataset is random, as it finds the average value (B) of the set within 1% error (Pierce).

$$\mu = 3.207 \cdot 10^{2} \text{ where } \%_{\varepsilon} = \frac{B-\mu}{\mu} \cdot 100 = 0.96\%$$

Higher power polynomials were excluded from this study because there is nothing to signify higher power polynomials as an accurate representation of the data.

*Power analysis*

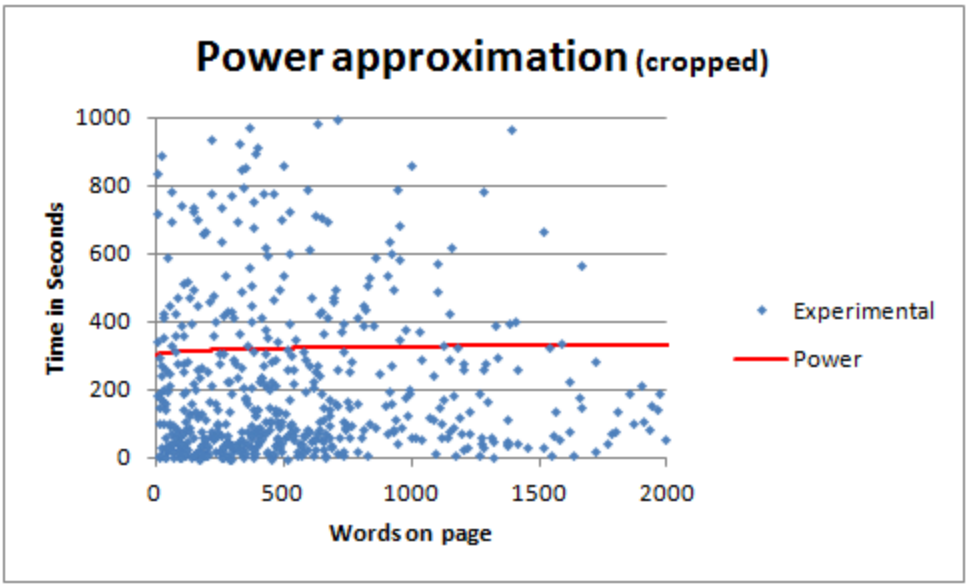


Figure 05: Cropped plot combined with the results of the power approximation

The power approximation will show if there are any power tendencies (positive or negative) among this data. While this curve appears to be similar to the line from Figure 4, it has a slightly positive slope.

$f(x) = A \cdot x^B + C$ where $A = 5.175,\ B = 2.640 \cdot 10^{-1},\ C = 2.948 \cdot 10^{2}$ having $\varepsilon = 1.357 \cdot 10^{8}$

A brief glance at the coefficients shows that C is no longer the system's $\mu$. Diving further into the $A \cdot x^B$ half of the proposed equation we can see that that portion alone ranges from 5.175 to 53.30, which when averaged again and C is added, gets us to 2% of the group's mean (324.1). This further affirms that this data is too scattered to be accurately represented by a function of this form.

*Exponential analysis*

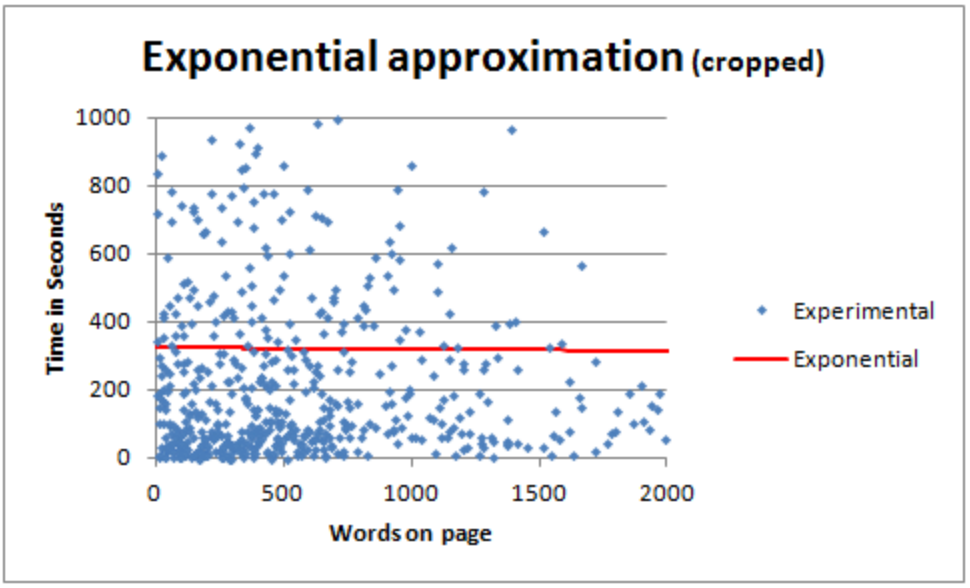


Figure 06: Cropped plot combined with the results of the exponential approximation

Fitting an exponential will show how well this data set aligns with an exponential function.

$f(x) = A \cdot e^{B \cdot x} + C$ where $A = 8.266 \cdot 10^{-2}$, $B = -5.731 \cdot 10^{-6}$, $C = -5.029 \cdot 10^{2}$ having $\varepsilon = 1.357 \cdot 10^{8}$

Similar to the power approximation this exponential approximation seems to show that there is little correlation between the number of words on a page and the time spent on a page. By looking at the output values for the lowest and highest word counts in the data, 323.7 and 291.8 respectively, we see that the data appears to have a slight decrease. This decrease of 32 seconds may look significant, but given the scale of our data, it is minimal at best. Essentially, this is saying that a page with 0 words takes 32 more seconds to read than a page with 7000 words. Note that again the average of the proposed function values is within 0.01% of the data's mean value.

*Logarithmic analysis*

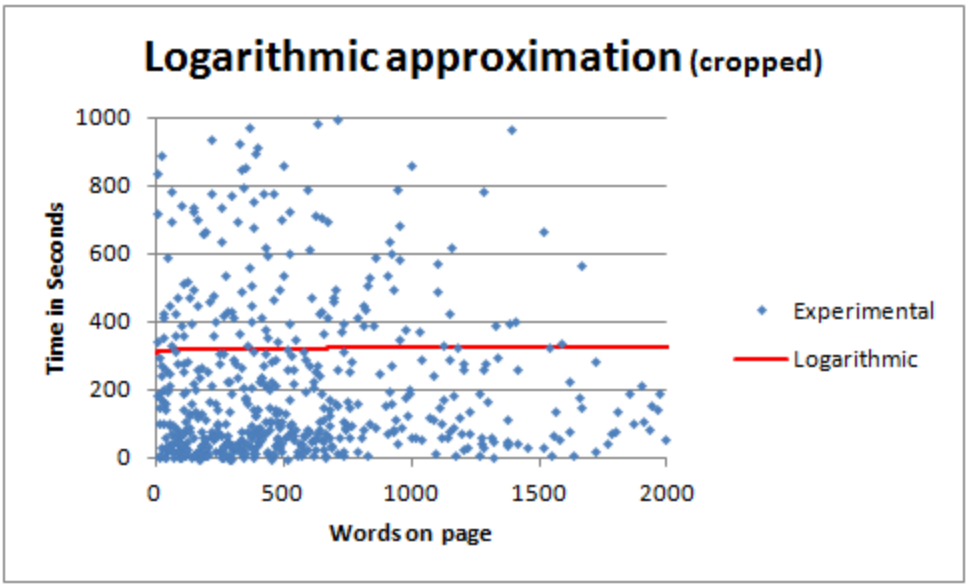


Figure 07: Cropped plot combined with the results of the logarithmic approximation

Finally, fitting a logarithmic function to the data will show if there are any logarithmic influences on the data.

$$f(x) = A \cdot ln(x) + B \text{ where } A = 2.373,\ B = 3.066 \cdot 10^2 \text{ having } \varepsilon = 1.357 \cdot 10^8$$

Again, it appears the data is not represented by any other function than that of a first order polynomial. The curve spans a similar set of values as previous functions and appears to be mostly horizontal. Thus, the resulting value is close to the average value of the data.

## Numerical Conclusions

If people were reading web content without any other distractions, we would expect to see about 180 to 200 words per minute or around 3 words per second on the page. If users were reading every word on a page we would expect the data to follow a linear trend with a slope of about 1/3. In actuality, most people don't read every word on a page, causing us to expect a more logarithmic function that levels off over time. Also, it may be useful to propose the fact that the fitting line must go through the origin since a page with 0 words should have 0 time spent on it.

By looking at all the regressions plotted together we see that they all converge to the average value of the function. This again shows that the data is too random to be fit by any specific function and requires further analysis.

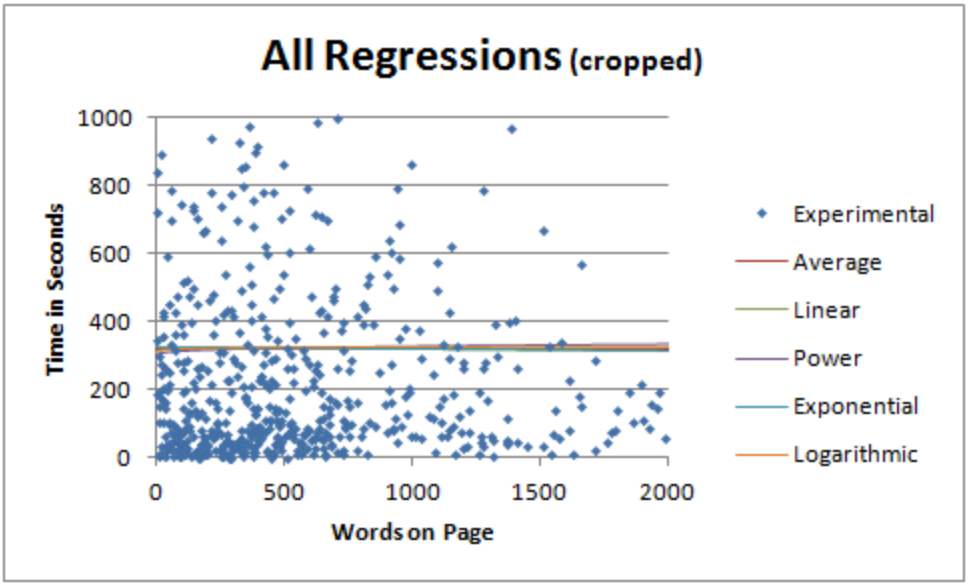


Figure 08: Cropped plot combined with all the regressions

Looking at a further cropped version of the Figure 08 (Figure 09), we easily see the differences between the regressions. While the plots seem to vary, we must first note the scales of the x vs the y axis. Noticing those two facts shows that this data is unable to be matched with a typical regression function and raises questions about the data gathering process.

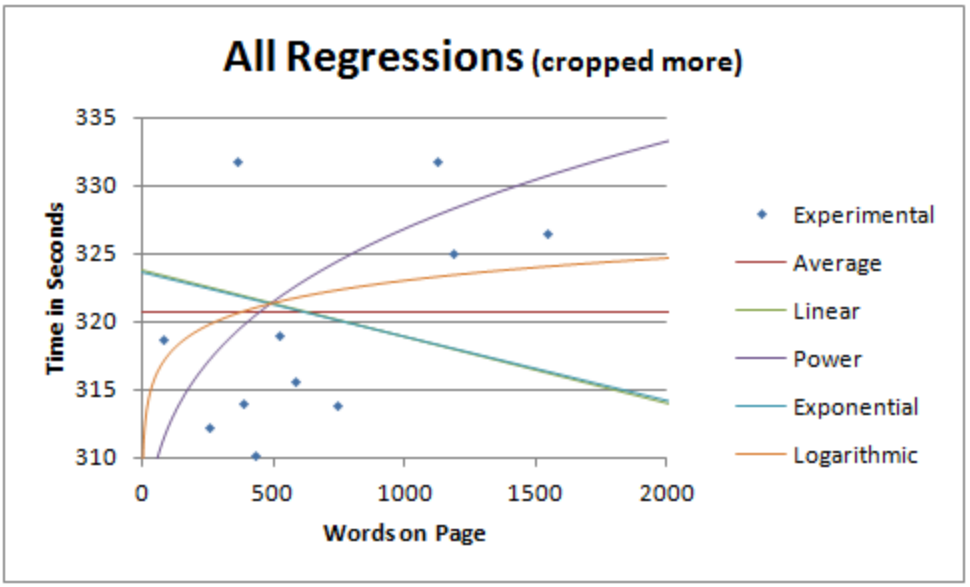


Figure 09: A y-axis cropped plot of Figure 08, showing the differences between the regressions

Unfortunately, this data was unable to be fit by any of the proposed functions, indicating one or more of the following happened.

1. The users of the Carroll College website are not reading content but are more interested in other facets of the web pages. This could be verified by going back to our two assumptions and seeing how far the users are deviating from the predictable reading rate of educated adults. These other facets may include images, videos or graphs.
2. The users of the Carroll College website are unable to find what they are looking for and thus spend time navigating through many pages in an attempt to locate the desired information.
3. The pages that contain web-applications and forms (long time-of-use pages) have very little textual content. This reduction in text content and increase in time on a page throws off word count analysis. To accommodate this situation, application pages should be excluded from the data set.
4. Finally, there is the possibility that the analytics engine did not account for special factors that may falsely influence the amount of time spent on a page. This could be demonstrated when a user switches tabs or windows while still keeping the Carroll site up, but not actively reading it. This behavior adds time spent on a page even though the page is not being actively viewed, which skews the data set. Modifications to the analytics system would be required if this is to be taken into account in the future.

Since $N_8$stats was custom written, a large weight is placed on the final possibility. While this analysis failed to find a suitable regression model, it pointed out an error in the web analytics system design. Without this analysis, the analytics system could have been pushed to a production state without ever knowing an error was present. Thus further development is required before version 2 of $N_8$stats is available to the public.

## Markov Chain analysis

Using Markov Chains, we can develop another metric for the effectiveness of a website's navigation and information distribution. Markov Chains are used to analyze Markov processes, where a system moves between a finite number of states (Markov). Modeling a website as a Markov process assigns each page as a state and adjusts weights between pages to model actual traffic patterns. Markov chain analysis gives us the ability to see where users will end up after n timesteps in the future and the ability to see the steady-state stability of a system. By analysing these DTMCs (discrete time Markov chains) we should be able to see if users cycle back to similar pages, indicating a failure to provide pertinent information to the user.

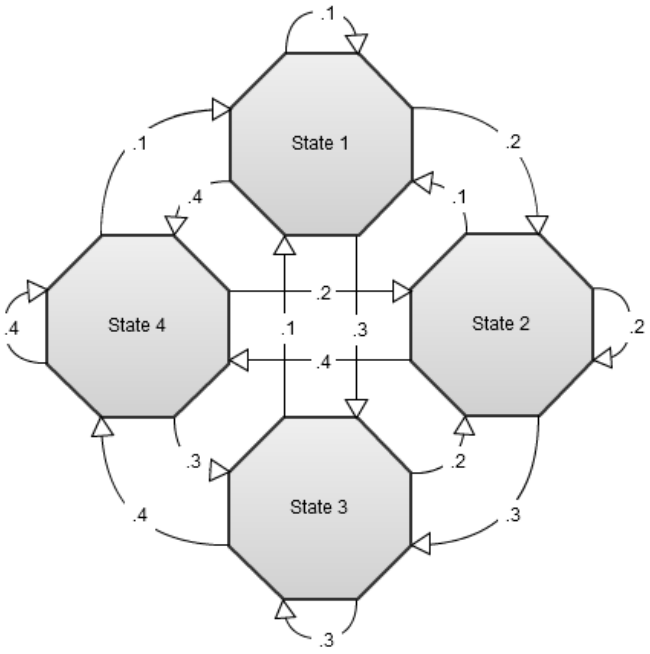


Figure 10: A visual representation of a Markov chain

A Markov chain can be represented graphically as a state transition diagram, as in Figure 10. We can also represent a Markov chain as an n by n matrix where n is the number of states in the Markov processes state-space (relation to graph theory). Each i,j entry in this matrix represents the probability of coming from state i and going to state j. Since we are dealing with probabilities, it may be worth noting that each row must sum to 1. Once such a matrix is generated we can use a variety of linear algebra techniques to find both the nth time step probability and the long term behavior of the system.

$$x = \begin{bmatrix} .1 & .2 & .3 & .4 \\ .1 & .2 & .3 & .4 \\ .1 & .2 & .3 & .4 \\ .1 & .2 & .3 & .4 \end{bmatrix}$$

Equation 1: The matrix representation of Figure 10's Markov chain

## Data

Fortunately, for this analysis, data gathering is a much simpler task thanks to the way the analytics system was designed. As previously mentioned in "Process of data acquisition", each time a user comes to the site an entry is added to the action table which contains the page the user is viewing and where the viewer is coming from. Using this structure, MySQL has the ability to use aggregate functions, such as sum or count, that can operate on a set of values. Additionally, by adding a group-by clause, we can

identify specific cases rather than finding the count of all items in a result query. This allows us to build a query, such as the following, that returns the number of times someone has traveled from one page to another (MySQL).

```
SELECT currentPage, referrer, COUNT(*) FROM `actions` GROUP BY currentPage, referrer
```

An example row resulting from such a query would tell us that 39 people have come from the homepage and went to the academics page of a sample website. Essentially this is returning a row for each non-zero entry $p_{ij}$ that exists in a Markov chain matrix. All that is left is a little data manipulation and filling in the zeros to get us a full Markov process modeled in a Markov matrix.

Noting that we now have a matrix showing the number of people that came from *i* and went to *j* we must normalize the matrix in order to perform our analysis.

## Analysis

In order to create a Markov matrix, we need to convert the number of people moving from page i to page j into a probability of moving from page i to page j. This is easily done by dividing each row by its sum. Since MATLAB is being used from a command line interface, the following function will perform this action. The diag method simply takes the input vector and converts it to a diagonal matrix as seen below. The sum(M, 2) method returns a vector containing the sum of the rows of matrix M.

```
M=diag(1./sum(M, 2))*M
```

$$\mathrm{diag}\left(\begin{bmatrix}1\\2\\3\end{bmatrix}\right) = \begin{bmatrix}1 & 0 & 0\\0 & 2 & 0\\0 & 0 & 3\end{bmatrix}$$

After pre-processing is complete, Markov analysis can begin. By raising M to the power n (where n=1,2,3,...) we can see the nth transition period probability matrix. By looking for larger values in the resulting matrices we can find where most users are and how they are moving through the site.

Furthermore, a long-term steady state analysis may be desired to find the overall landing page for all users on the website. This can be done with linear algebra and the use of eigenvalues and vectors.

## Results

Simply looking at the Markov matrix M before preprocessing we can find popular navigation tendencies of a population. For example, on the Carroll College website, 18,554 people went from the home page to the students page. This shows that instead of having a homepage Carroll may consider moving the students page to the homepage to reduce unnecessary traffic on the server. A few other interesting numbers show that

2061 users refreshed the home page and 1349 users refreshed the students page.

After squaring the matrix, the maximum value became a page that was inaccessible by any regular visitor of the site. After some discussion with my co-worker, it was discovered that he had been developing that page during my data acquisition. For his development, he would make slight modifications to some code and refresh the page, creating another link from that page to itself. His development actions created a 99% chance that someone who got to this page would simply stay there and refresh it. This is rather odd behavior for a regular site visitor, but this demonstrates that $N_8$stats could be used to show potential security threats to a website. This also shows that different users should be classified so that development analytics does not interfere with production analytics.

### Further

As mixed traffic types are a possibility, it may be desired to allow the user to filter out particular types of traffic in order to produce more meaningful results. This would weed out any development traffic that negatively affects the accuracy of the results.

Another source of error comes from the absence of emitting and absorbing states for when a user enters or leaves the website. Without these states, users can become stuck in a certain state and give faulty results. The addition of a source and sink state may be required to fix such problems.

## Keyword analysis

Generating a few new metrics for analyzing the navigation structure of a website is fine as long as there are people visiting a site. But what happens if the site is just starting or has lower visitor counts on particular pages? This is where an in-depth look into keywords works wonders for a new website.

Keywords are used by search engines to help them index a website. This index is designed to relate a user's search query to websites everywhere and find possible matches. Knowing this, it is easy to see that keywords play a very important role in bringing relevant and interested traffic to a website. Such traffic is called targeted traffic.

Before even starting the discussion about keywords, it may be useful to note the difference between traffic and targeted traffic. General site traffic is anyone going to a website. This is greatly different than targeted traffic, which is: users who are interested in a website's content actually going to that particular website. While these two ideas seem like traffic either way, both show clearly in web analytics. General site traffic increases bounce rates whereas targeted traffic takes time to read content and may even result in a future visit.

There are two ways a search engine can find keywords on a page for a website. The

first has each search engine blindly crawl a website's text and look for words and phrases that repeat. While this method works, it lacks the human interaction and can accidentally hit on irrelevant keywords that end up sending irrelevant traffic. The second way is to have the webmaster physically tell a search engine's spider what the keywords are through the use of meta tags. Typically, this is done by adding a line like the following within the head section of an HTML page which give a comma delimited list of keywords for a page.

```
<meta name="keywords" content="Analytics,Statistics,Website Management">
```

This method of communicating keywords to a search engine is preferred by search engines because it removes their text parsing process and it allows webmasters to choose targeted words to more effectively target their desired audience.

It may be worth noting that some search engines, such as Google, do both. This allows them to see if the webmaster provided keywords which actually appear in the body of a website. By checking if the keywords exist in both contexts, these systems can tell if webmasters are telling the truth about their content or if they are trying to target the wrong traffic. If the keywords in both sections don't match then these systems can arbitrarily choose from either set, resulting in some very interesting special cases. Or even worse, the search engine can rank pages poorly, giving them lower traffic flow.

This just shows how important it is to have relevant keywords that are actually used in the text. Sometimes it is necessary to reword portions of a page so the keywords may be emphasized; however, that requires the ability to write which, usually, requires little to no math. So instead of explaining how to rewrite text to represent keywords more accurately, we are going to use a search engine's own algorithm for finding keywords in text and allow webmasters to choose the most targeted keywords for a particular set of content.

### N-grams sequences

Traditionally, keywords are found in a body of text by finding repeated n-gram sequences. An n-gram is a contiguous sequence of n items from a given sequence of text. Since we are dealing with keywords and keyphrases we will be using word items for these n-grams. An example showing how n-grams are created is shown in the table below using "how now brown cow" as the source text.

| N | n-grams (comma delimited) |
|---|---|
| 1 | how, now, brown, cow |
| 2 | how now, now brown, brown cow |
| 3 | how now brown, now brown cow |
| 4 | how now brown cow |

Figure 11: Example of different length n-grams

Using some computational linguistics with n-grams we can re-create a method similar to the one search engines use to build their indexes but apply it to our page's text allowing us to target keywords and phrases that naturally arise from the text (n-gram). Here is a description of my algorithm to find significant n-grams within a page's text.

### Algorithm Description

*Step zero:* Gather and sanitize the input text by converting all alphabetic characters to lowercase and removing all characters that are not alphabetic or spaces. Also, ensure there is only a single space between words. The result is a sample of text that is all lowercase alphabetic characters with a single space separating words.

*Step one:* Generate an array of all possible word n-grams for a given text along with the number of times each n-gram occurs. First, break the sanitized text into an array of words. Since the sanitized text is space delimited, this can be easily accomplished with the text.split(' ') JavaScript command. Once this word array is built, loop over it once for each N chosen. Eight is a common number because keyphrases longer than eight words are not accepted by search engines. For each iteration over the array, grab sequences of N words and join them with a space. Using these joined words as the key in an associative array, it becomes possible to count the occurrences of specific N-grams. When step one is complete, we are left with a large associative array which has every possible N-gram found in the text as keys and the occurrence of that N-gram as its value.

*Step two:* Remove low count N-grams by removing all N-grams from the list with an occurrence count less than $C_{min}$, a parameter to the algorithm. This step just loops through each key in the associative array and deletes any entry with a value less than $C_{min}$. $C_{min}$ is a webmaster chosen parameter that affects the rejection of low count N-grams.

*Step three:* Remove all n-grams that start or end with common english words. Note this also removes n-grams that contain only common english words. This step uses a dictionary of common english words such as "a", "our" and "somewhere" and checks if

they are at the beginning or the end of the N-gram, removing the N-gram from the list if found.

*Step four:* find the longest, same count n-grams with same number of occurrences. This step stems from the fact that 2-grams and 3-grams can contain the same words. For example the grams “minute drive”, “drive from campus” and “minute drive from campus” all show up in a keyword analysis of a page, but it is easy to see that they are all part of the same keyphrase. Unfortunately, computers cannot make this distinction easily so this step was created to combine the keyphrases. This step used a helper function that returns all other N-grams that contain a particular supplied N-gram. Then the method checks to see if the occurrence counts are the same: If they are, then the shorter of the two N-grams is removed from the set. This function is called for each N-gram in the original set.

*Step five:* Present list of filtered n-grams to user for selection of targeted keywords and phrases.

At this point the webmaster has a list of all the keywords that a typical search engine’s keyword spider would catch. It is now up to the webmaster to chose entries from this list that are specifically targeted to the audience of a particular page. If the webmaster decides these keywords do not accurately portray the message of the text, then the text should be rewritten to emphasize more relevant keywords.

# Improvements

Initial improvements stem mostly from correcting errors within $N_8$stats itself. Once the analytics system is amended, more accurate results and further analysis may begin on the data itself. A majority of these errors came from small overlooked facts in the application design phase of the project.

## Design problems/errors

Throughout the design process, constant revisions were made to $N_8$stats as error conditions were found. Despite these efforts, $N_8$stats, like most tools, was imperfect in its first version. Whether it was during the month of gathering analytics or when retrieving data to analyze, numerous problems and mistakes were found. Many of these revolved around overlooking small parts of the database while others did not show until the system went active.

### Accounting for background time

This problem arose during the numerical analysis, when we were discussing how the data seemed to be random, with some page visits exceeding 2 days. This is caused by users switching tabs to browse the internet while keeping other websites up in the background. Since the pages are still open, the connection with the analytics system is never dropped, so the system still believes the page is being actively viewed. This large source of error can be resolved with a future revision of $N_8$stats by listening to the web browser's events and relaying them to the analytics system.

Within the JavaScript object model there exist a pair of events that fire based on a window's view state. These events are located on the window object and are defined by onFocus and onBlur events. By relaying these events to the analytics system, $N_8$stats could record start and stop times for the actual viewing of a page, thereby perhaps removing much of the noise found in the regression of time on page. Note: an alteration to the database would also be required to store multiple start and stop times for the opening of a single page.

Unfortunately, there is the possibility of having multiple windows up at a time in many of the operating systems of today. This means that even though a particular page is in focus, there is no way to guarantee that the user is actually reading it using this solution. That being said, this solution should greatly reduce the inaccuracies of making the assumption that if a page has a connection to $N_8$stats, it is actively being viewed.

### Traffic Sources

Stemming from the Markov analysis, it would seem that a separation of traffic types would be an improvement on the first revision of $N_8$stats. Being able to distinguish between types of traffic would be beneficial because it would allow a web developer's actions to be excluded from analysis. Having developmental traffic mixed with ordinary traffic leads to trends that do not accurately reflect the production traffic of a website. This problem could be resolved by setting a long term cookie when a developer logs into the analytics system. This flag could then be checked when initially connecting to $N_8$stats, allowing the analytics system to store the traffic as development traffic.

In addition to distinguishing between development and regular traffic, it may also be beneficial to distinguish between internal and external traffic. This can be best seen by looking at the bounce rate of Carroll's student page. While this number is abnormally high, it may be worth noting that every browser in campus computer labs has the student's page set as their default homepage. This is what causes such a high bounce rate and may be worth excluding from the data. This could be changed by providing the option to filter out specific traffic sources from particular analyses. Using either an exact ip match or an ip subnet match could both be possible implementations of a source filter

for internal and external sources.

## Database Structure

During the database design process there were a few design inefficiencies that were not visible before the live testing and data analysis began. Most of these errors stemmed from the database design. These mistakes are expected when designing a new system that has little written on the subject and few if any previously attempted implementations. These database alterations range from small oversights to larger, more problematic flaws that were initially not present.

### *Sanitize Data*

Limitations in MySQL forced the use of a value and hash type structure within tables that store variable length data. MySQL requires that any unique field have a defined length. This causes size problems when attempting to store URI strings that vary greatly in length. In order to overcome this limitation, a value and hash pair design was developed. This meant storing the value of an item alongside a hashed value of the exact same text. This technique permits the storage of human readable information while allowing the database to enforce unique entries.

| Value (variable length) | MD5 Hash (unique + static length) |
|---|---|
| /students/index.cc | 57d772985d1a485a87e1983871b863b8 |
| /STUDENTS/index.cc | 1c104381c15a5ee8334a644b4f8a601e |

Figure 12: Examples of the same URI's with different MD5 Hashes

This causes a problem with mixed case values, because while MySQL does not distinguish between mixed case values in a text field, changing the case of a single character can change the entire hash value of a piece of text. This results with duplicate entries for "/students/index.cc" and "/STUDENTS/index.cc" in the database. To fix such a problem, the application must convert all input strings into these value and hash pair tables to lowercase in order to ensure singular values for these data tables.

### *Break Apart URI*

In $N_8$stats version 1, entire URIs were stored in a single table. While this seems logical, when query strings are introduced, this causes a large problem. A great example of this can be seen in the search result page of the site, which ended up being stored more than four thousand different ways over the course of a month. Entries such as "/search/index.cc?q=students" and "/search/index.cc?q=alumni" littered the table with duplicated data. This could be improved by separately storing the query strings from the remaining part of the URI. This could improve past analyses and provide a gateway for new analysis, such as looking into what is most commonly searched for on a website.

*Table Structure*

Initial database designs were targeted at shrinking the data as much as possible and keeping a small table size. While appearing promising, it would be desirable to find more ways to shrink the data because by the end of the month of data gathering there were over ¼ million records in one of the tables, consuming more that 60 MB of space. This large number of records made retrieving data for analysis a time-consuming task.

A possible solution to this problem is to store a compiled report in a special table and removing all the entries that can be summarized in this report. This would allow for the system to remain unaltered.

Another, more mathematical solution would be to have running averages and counted fields to summarize data rather than storing every entry and averaging over it when rendering the analytics reports. This would constitute the storing of a cumulative sum of values and a number of times that the data had been updated, thus reducing the computation of averaging over multiple rows of a table to dividing the values within the same record. An example of the above compression scheme can be seen in the following scenario: storing GPA's for students in order to find the average GPA within particular classes.

<table>
<tr><th>Uncompressed</th><th>Compressed</th></tr>
<tr><td>
<table>
<tr><th>Class</th><th>GPA</th></tr>
<tr><td>Math</td><td>3.5</td></tr>
<tr><td>Math</td><td>3</td></tr>
<tr><td>Math</td><td>2.75</td></tr>
<tr><td>Physics</td><td>4</td></tr>
<tr><td>Physics</td><td>3</td></tr>
</table>
</td><td>
<table>
<tr><th>Class</th><th>Total</th><th>Count</th></tr>
<tr><td>Math</td><td>9.25</td><td>3</td></tr>
<tr><td>Physics</td><td>7</td><td>2</td></tr>
</table>
</td></tr>
<tr><td>Math: (3.5+3+2.75)/3 = 3.083<br>Physics: (4+3)/2</td><td>Math: 9.25/3 = 3.083<br>Physics: 7/2 = 3.5</td></tr>
</table>

FIgure 13: Size comparison of expanded and compressed average data storage

### Client key storage

Another issue that has a more unique solution is the way identifying information was stored in users' browsers for future use. As mentioned before, it became necessary to store a small bit of information on a user's computer in order to simplify the categorization process. As no database key values should ever be visible to the public as a matter of security, a MD5 hash of the keys was chosen to maintain this security. That means there were two of these MD5 hashes stored on a user's browser consisting of 32 alphanumeric characters each. While 64 characters doesn't seem like a lot of data, if every website a user went to stored 64 characters (not counting the storage structure and variable names) browsers would slow greatly. Also, such a long string of characters may become susceptible to corruption upon transmission.

Thus the idea of changing the base of the keys arose. The keys are stored in the database as base 10 non-negative numbers. By changing the base of these values to a higher base, more information can be contained within a single character. Upon initial research there appeared to be a strong leaning to base 62, which consisted of the numbers zero through nine and both upper and lower case letters (26*2+10=62). But upon further research, I found there there are 4 more characters that are considered "safe" web characters by the Internet Engineering Task Force: the hyphen, period, underscore, and tilde. By using these other characters, base 66 could be able to represent values. This proves a great improvement over the MD5 hash since the hash always has 32 characters where this base 66 solution could represent $2.5 \times 10^{56}$ numbers before it became 32 characters long (IETF).

On the other hand, once key values became very large ($> 2.5 \times 10^{56}$) it would be better to use the MD5 hash because it holds constant at 32 characters long. Thus a proposed solution of using base 66 until the base method exceeds 31 characters in length, at which point the MD5 hash would be used to ensure that the transferred string does not exceed 32 characters. As for decrypting the values, checking if the transferred string is 32 characters long would immediately show if the value should be analyzed using the MD5 method or the base change method.

## Further analysis and reports

Over the course of this paper, two methods were described that improved the navigational structure of a website as well as a method to improve search engines' indexing of a website. While this system currently neglects the basic reports which can easily be added in the future, further analysis of the information is possible.

### Live Reports

With the use of Node.js, the power to run live reports is possible. Showing current active viewers on the site, along with plotting points on a map where users are coming from can be done live thanks to this websocket technology. These reports simply pass traffic hits from the user directly to the monitor, without even waiting for the data to be inserted into the database.

### Semi-Static Reports

While it would be nice if all reports were live, due to the enormity of the data, some information must be stored on the server and compiled into a semi-live report. These reports contain more long term pieces of information that require database reports and minimal mathematical processing. These reports can be set to occur at specific intervals on the analytics server and transfer updated reports to the monitor as they are compiled. These reports include browser source information that explain what web browsers clients use most.

### Semi-Live Reports

While it may appear that a semi-static report and a semi-live report mean the same thing, they are two completely different entities. This semi-live report would require initial data to be provided by a database report, but with minimal processing, and can use data from live reports to update on the fly. These reports include average time on page, using a storage method that stores the count along with the sum. This would require minimal processing for a monitor. These semi-live reports do not appear in any of the others researched and would be very interesting to see in action.

# Conclusion

Even though two metrics for monitoring the effectiveness of a website's navigation and an algorithm to suggest keywords for a webpage to improve indexing levels were proposed, $N_8$stats was plagued with design flaws. To further research in any of the above processes these problems will need to be fixed. Regardless of the faults, these metrics answer questions that have not been asked of any analytics system before. By using these reports along with past reports web designers and developers can create the best websites possible based on user input and not based on uninformed decisions. This can save time and money of businesses everywhere and hopefully, make the world a little better informed.

# References

"Amazon Elastic Compute Cloud (Amazon EC2), Cloud Computing Servers ." Amazon Web Services, Cloud Computing: Compute, Storage, Database. N.p., n.d. Web. 7 Mar. 2013. <http://aws.amazon.com/ec2/>.

Cline, Kelly. Personal interview. 26 Feb. 2013.

Chapra, Steven C., and Raymond P. Canale. Numerical methods for engineers. 6th ed. Boston: McGraw-Hill Higher Education, 2010. Print.

"IETF." Internet Engineering Task Force (IETF). N.p., n.d. Web. 7 Mar. 2013. <http://www.ietf.org/rfc/rfc3986.txt>.

"Markov chain." Wikipedia, the free encyclopedia. N.p., n.d. Web. 7 Mar. 2013. <http://en.wikipedia.org/wiki/Markov_chain>.

"MySQL :: MySQL 5.0 Reference."MySQL :: Developer Zone. N.p., n.d. Web. 7 Mar. 2013. <http://dev.mysql.com/doc/refman/5.0/en/group-by-functions.html>.

"n-gram." Wikipedia, the free encyclopedia. N.p., n.d. Web. 7 Mar. 2013. <http://en.wikipedia.org/wiki/N-gram>.

Pierce, Rod. "Percentage Error." Math is Fun - Maths Resources. N.p., n.d. Web. 27 Feb. 2013. <http://bit.ly/ZHuIQW>.

"Web Application Testing: Back to the Drawing Board?." Information Technology for Students and Professionals. N.p., n.d. Web. 7 Mar. 2013. <http://bit.ly/XU6Wyl>

# Definitions

These words are used throughout the text.  In order to avoid confusion their definitions follow.

Analytics Server: The server on which the analytics are executed. See $N_8$stats.

Bounce rate: The percentage of all viewers to a website who view only a single page and then leave..

Client: A client is any viewer of a company's website.

Monitor: Anyone who can view the web analytics of a website.

$N_8$stats: Custom written analytics system described in “Gathering Data”.

Server: A company’s web server, serving pages to clients.

Time on page: The time spent viewing a particular page on a website.

to-load-time live: Content is only as updated as the last time the browser is refreshed. Not using websockets.

URI: Uniform Resource Indicator. Example: /students/index.cc

User: See Client

Web-Application: A web page that interacts with the user without changing the URL

Webmasters: Managers of a website, have full access to modification of a website.

Websocket: A persistent connection between a browser and a server that allows the instant passing of data without the need of a page refresh.

# SIGNATURE PAGE

This thesis for honors recognition has been approved for the

Department of Mathematics, Engineering, and Computer Science .

Dr. Mark Parker, Director — 6/3/2013 Date

Dr. Anthony Szpilka, Reader — 9/10/13 Date

Phil Rose, Reader — 6/13/13 Date