Dynamic Aggregation to Support Pattern Discovery: A case study with web logs

Lida Tang and Ben Shneiderman

Department of Computer Science
University of Maryland
College Park, MD 20720
{ltang, ben}@cs.umd.edu

Abstract. Rapid growth of digital data collections is overwhelming the capabilities of humans to comprehend them without aid. The extraction of useful data from large raw data sets is something that humans do poorly because of the overwhelming amount of information. Aggregation is a technique that extracts important aspect from groups of data thus reducing the amount that the user has to deal with at one time, thereby enabling them to discover patterns, outliers, gaps, and clusters. Previous mechanisms for interactive exploration with aggregated data was either too complex to use or too limited in scope. This paper proposes a new technique for dynamic aggregation that can combine with dynamic queries to support most of the tasks involved in data manipulation.

1 Introduction

Current technologies have enabled massive collections of data. Unfortunately, disk storage grows at a faster rate than Moore's law. Newer and faster algorithms for data analysis are always in demand to harness the flood of data. If the amount of data can be reduced to a manageable size, then humans can find patterns that automated algorithms may have missed. Dynamic Queries (DQ) is an interactive technique for data exploration.[1] Users manipulate sliders to filter out data. Each slider corresponds to an attribute of the data. A requirement of dynamic queries is that the visualization must keep up with the user's manipulation within 100 milliseconds. Since a large portion of the computer's computation is spent on visualization, when the data sets grow, the time to complete drawing grows proportionately. Thus DQ isn't suitable for dealing with large amounts of data. Aggregation is an effective way of managing large data sets. It summarizes groups of similar data elements and can greatly reduce the number of glyphs that are shown on the screen. Because users can specify how to aggregate the data, the important aspects of the data will be preserved while the data set size is reduced. Patterns that are hidden within millions of data points can emerge dramatically when aggregation reduces these into thousands of points.

1.1 Motivation

Large datasets poses two problems to interactive exploration. One is how to represent the elements on the screen fast enough. Second is if you draw it on the screen, can the user even understand it. Visual occlusion is a problem in general for visualization. If the user can't see the data point, then the time spent drawing the item was wasted. This problem can be solved for small numbers of items. The commercial data analysis package, SpotFire (www.spotfire.com), randomly jitters the data points continuously, so that clusters that occupy the same point can be seen. This technique can't be scaled up easily [2]. With larger data sets, the occlusion problem grows more and more pressing. The average size of current displays, in terms of pixels, is 1024x768, with the highest resolution on commercial screens being 1600x1200. The maximum number of data points that can appear on screen with no occlusion is then around 2 million; with each data item one pixel in size. Due to the non-uniform nature of most data sets, many data points will still occlude others. Because occlusion hides many data points, the visual representation can deceive users by not showing clusters that exist in the data.

Figure 1 shows 380016 data points using a scatter plot of clients to a website versus the time a request came. Are there any patterns in that data? Size coding of data points further decreases the maximum number of data points shown and increases occlusion. The number of data points that can be effectively understood with size coding enabled is probably orders of magnitude less than the theoretical maximum. Thus, aggregation is needed to reduce the total number of data points down to a manageable size. This also will reduce the rendering time of the display and increases the interactively of the whole system. Figure 2 shows only 24869 data points with the same axes, but now the data points represent the number of requests that came from a client in an hour. Now horizontal black stripes are easily seen representing repeated client visits. The dark black vertical areas in early July indicate a period of intense activity while the blank vertical indicated a server outage. These patterns are hard to predict yet they are readily visible and invite further investigation

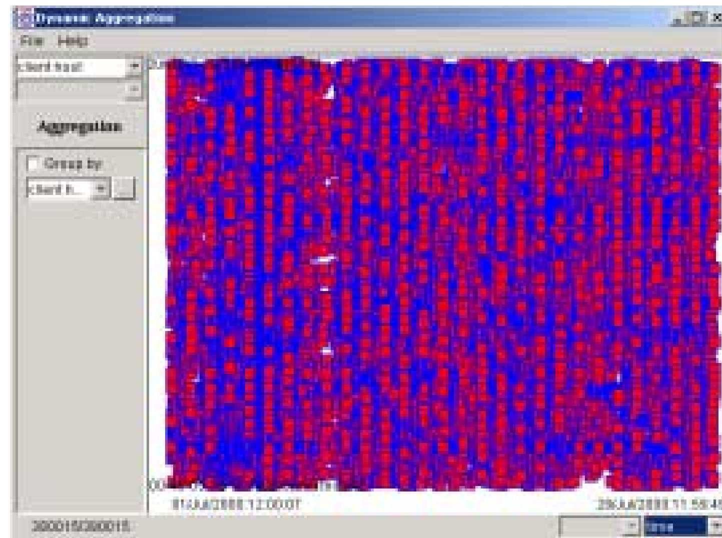

Fig. 1. Web log data when viewed without aggregation makes pattern discovery impossible

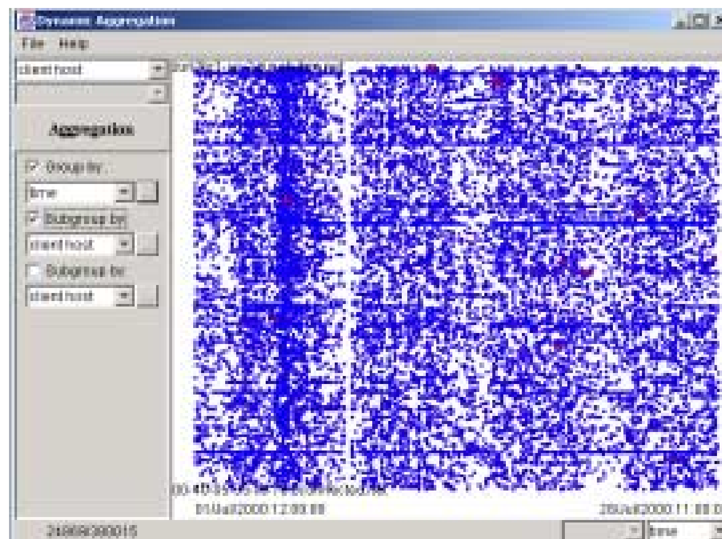

Fig. 2. Showing client requests aggregated by hour reveal several interesting activities and a server outage

1.2 Related Works

Using aggregation and Dynamic Queries (DQ) together is not a new idea. Goldstein et al [3] proposed it in 1994. An interface mechanism called Aggregate Manager (AM) was combined with DQ, which produced a

powerful combination. (see Figure 3 and 4) DQ is used to select a subset of the data set; this can be transferred over to AM as an aggregate group. AM can then do aggregation on different aggregate groups, then pass the data back to DQ for display. One of the lacking area of DQ is providing conjunct of disjunct groups, which can be performed by AM. Users can create multiple groups, and then pass them to the AM. The AM can then create an aggregation based on those groups and pass them back to the visualization. Using AM along with DQ provides many possible combinations for data manipulation, which is powerful but can be hard for users to understand and fully control. One of the success of DQ is that it is simple to understand and operate, and yet is sufficiently powerful for most tasks. The coupling of AM with DQ disrupts the simplicity.

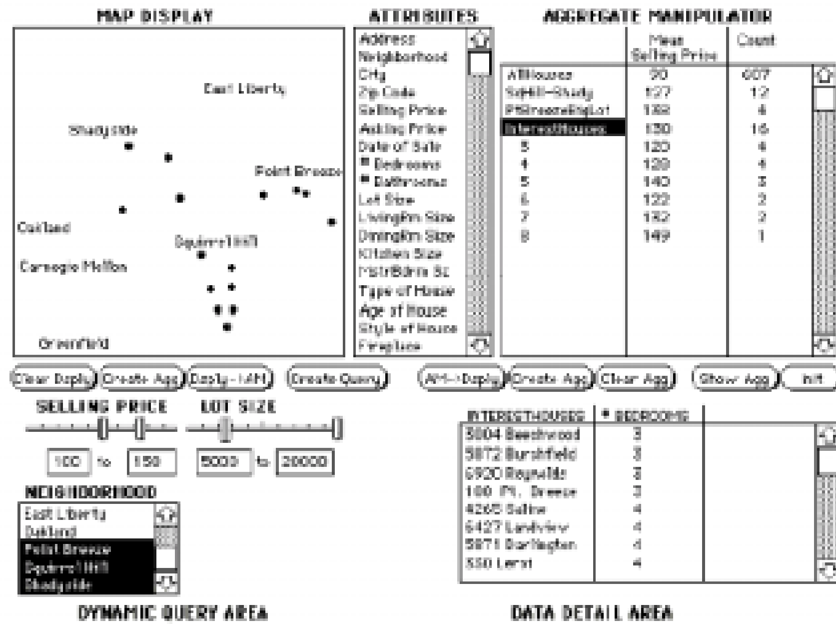

Fig. 3. The workspaces of AM with DQ

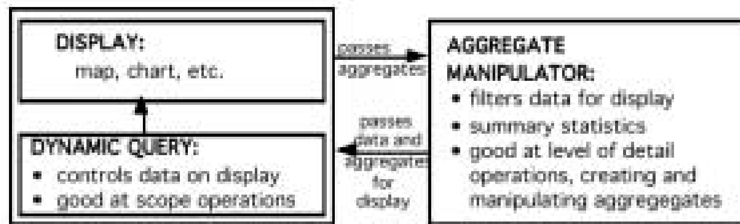

Fig. 5. The flow of data with AM with DQ

An alternative approach to user-controlled aggregation is automatic aggregation. Chuah [4] used automatic aggregation in SolarPlot, a circular histogram. Elements are mapped to a pixel on the circumference of a circle, the height of a spike that emanates from the pixel represents the number of data values that fall with in that pixel. This aggregation is intuitive and simple, the scale of the aggregation depends on the diameter of the circle, and the aggregated value is easily understood. There are several limitations to using SolarPlot. The use of a circle as the primary visualization leaves a large portion of the display unused, while part of the motivation for aggregation is to

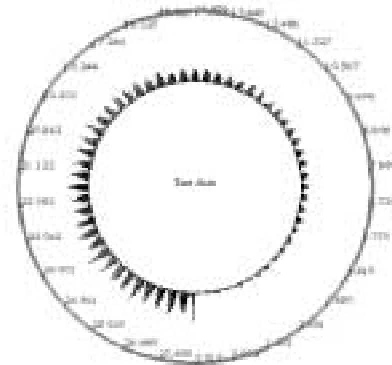

Fig. 4. SolarPlot showing a histogram

use screen space more efficiently. SolarPlot only encode one dimension of data in the visualization, any correlations between fields are harder to find as a result.

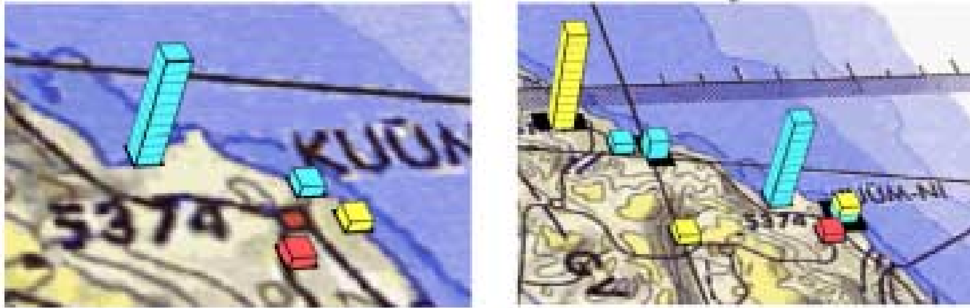

Fig. 6. Close up and zoomed out view of Aggregate

TowersRayson's [5] aggregate towers provide another automatic aggregation algorithm. The data points are displayed as cubes on a 3d plane. As the user zoom in and out, data points are clustered based on their geospatial location. The aggregate groups are represented by stacks pointing out of the plane. The cubes still retain their original color-coding. This automatic technique alleviates 2D occlusion problem by forcing it in to 3D. These stacks of data towers will occlude each other in 3D, but is easily remedied by allowing the user to freely rotate the view.

2. A Simple Manual Aggregation Interface

Automatic aggregation is useful as a way to reduce occlusion. However, having no user control makes automatic aggregation of limited use. Goldstein's AM is complex and hard to learn. A simple interface for manual aggregation is a nice compromise.

What are the basic steps involved in aggregation? First, a group of data item must be selected. In AM, this is done manual by the user; SolarPlot and Aggregate Tower use a spatial criterion. Secondly, the fields of the group must be summaries in a meaningful manner. In AM, the user selects a summary statistic from a pop-up menu, then the user selects an attributes to be shown with that statistic from another pop-up menu; the other interfaces provide a fixed histogram like representation. Lastly, the group should be able to be divided into subgroups.

Because DQ and aggregation are a powerful combination, the current interface is designed to be integrated with DQ easily. Fredrikson et al. [6] explored using aggregated data in conjunction with Spotfire, and showed the uses of different kinds of aggregation. Thus Spotfire's interface was used as the starting point of our system. The data are plotted as a scatter plot based on two attributes of the data. Combo boxes at the edges of the screen select the fields being plotted. A panel on the side displays DQ controls and detail on demand. The entire interface is in front of the user. Our system has similar characteristics as Spotfire. The aggregation controls are located on the left side so that DQ can be placed on the right side as is. The primary aggregation control is a combo box that can be enabled or disabled. Specifying a group of data is easy to achieve using DQ. However, creating many groups manually can be time consuming and should be automated. The user only needs to select a field to group on, and have the program figure out which data point belong in which group. The default grouping algorithm used is equivalence grouping. This is an easily understood algorithm and is fairly useful. Should the user require a different grouping criterion, clicking on the "..." button to the right of the combo box will bring up an options dialog. Here, the user can choose which algorithm to use and to configure the algorithm to their liking.

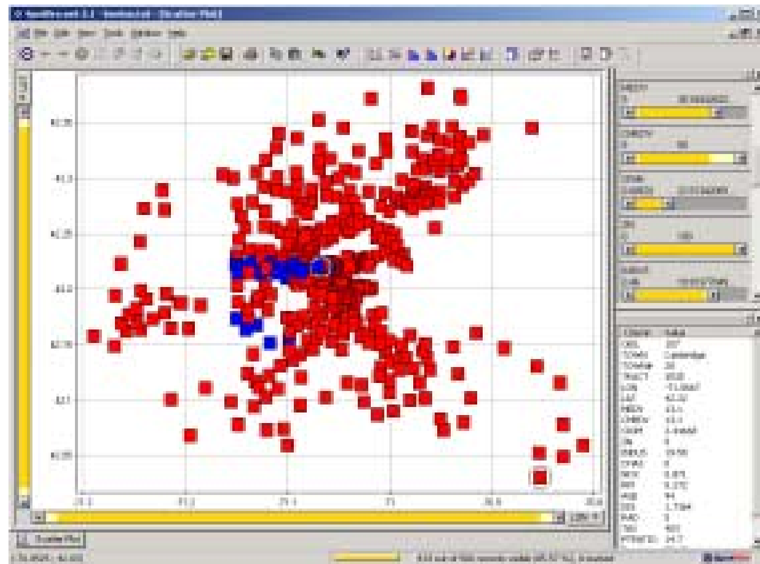

Fig. 7. Spotfire interface layout

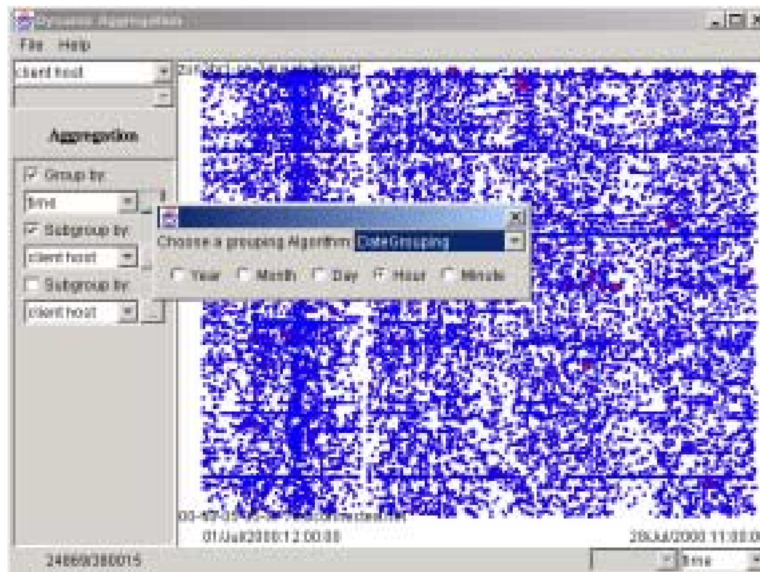

Fig. 8. Choosing to group by Date and selecting the granularity of grouping

Goldstein et al. defined four main types of decomposition, or grouping algorithms:

- Natural groupings - e.g. day, month, year, holiday vs. non-holiday
- Element frequency divisions - e.g. partition into groups with same number of elements
- Set interval divisions - e.g. each group have the same length interval
- Statistical methods

Aggregate tower's occlusion avoidance grouping would fall into the statistical methods, and SolarPlot's scheme is an interval division algorithm. Once the grouping computation is finished, the results are shown on the screen with each dot now representing a particular group, the size of the dot is currently coded to show the number of elements in that group. The secondary aggregation controls are the aggregate method combo boxes. Those are located below the vertical axes field selector, and to the left of the horizontal axes field selector. The user can select different aggregation algorithms for each axis independently. Creation of subgroups is exactly the same procedure as creating a group, select a field and a grouping algorithm.

3. System demonstration

Our current implementation is in Java. The main obstacle in using Java as a testing platform was not its speed, but rather, its appetite for memory. This is a problem in dealing with large data sets in general, however, Java's large primitive types and other inefficiencies are ill suited to hold large amounts of data in memory. The following graph is a plot of CPU time consumed to load the data versus the number of row in the data file. The data file is in a plain text format read in from the hard drive. CPU usage time is used instead of physical time because of the low level of memory in the testing machine resulted in sever thrashing which affects performance greatly and doesn't accurately reflect the program's performance.

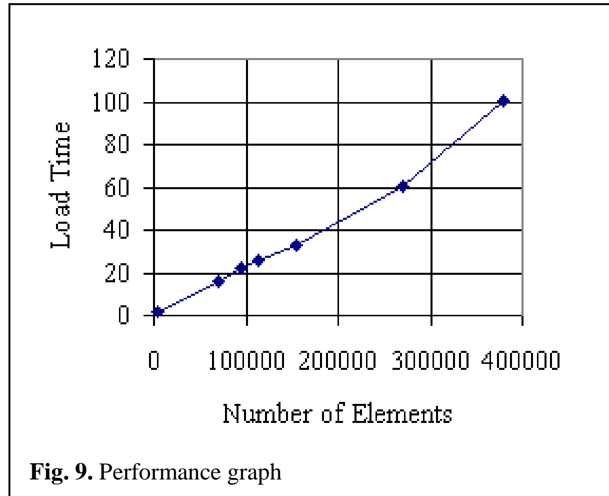

Fig. 9. Performance graph

The dataset used is extracted from web logs. The data is taken from University of Maryland's Computer Science web server. Only the requests that belonged to the HCIL section of the website (www.cs.umd.edu/hcil) was extracted. Hochheiser et. al. [7] explored the same dataset using SpotFire. However, some of their visualizations used preprocessed data that could be created just by using our system. The data have the following five fields:

- Client host
- time: timestamp of request
- url: the URL requested
- return code: the server response code to request
- bandwidth used: number of bytes transmitted for that request

The rough correspondence between number of elements and file size is around 10k data points per Meg. The data is first read into memory. Nominal data is sorted and given an ordinal value so they can be represented in screen coordinates. Then all data is wrapped up in different classes based on the type of data it is. It was surprising to see that the load time is roughly linear, since sorting of nominal data is order n(log n). Investigation into this paradox found that even though as dataset grows, the number of unique nominal data is growing at a sub-linear rate. This offsets the supra-linear rate of sorting thus producing a linear loading time.

Web log data is very large and has only a few fields. Most traditional web analysis packages create tables of statistics and static graphs. The user merely feed the data to the program, and it is the program that decides what to report back to the user. Hochheister et al. argued that interactive star field visualization, like Spotfire, is a valid way of analyzing web log data. However, in order to find some of the interesting features involved preprocessing and aggregation. Thus, using the same web data will be a good test of the flexibility and power of our simple aggregation interface. The data file used is 35 Meg tab delimited plain text file. The data covers four weeks of requests from July 1, 2000 to July 28, 2000. Since the web data consists of individual client requests, one logical grouping would be to group by user. Just by viewing the size of the groups (count of rows when group by client host), one can detect abnormally large numbers of requests from a particular user. (Figure 10)

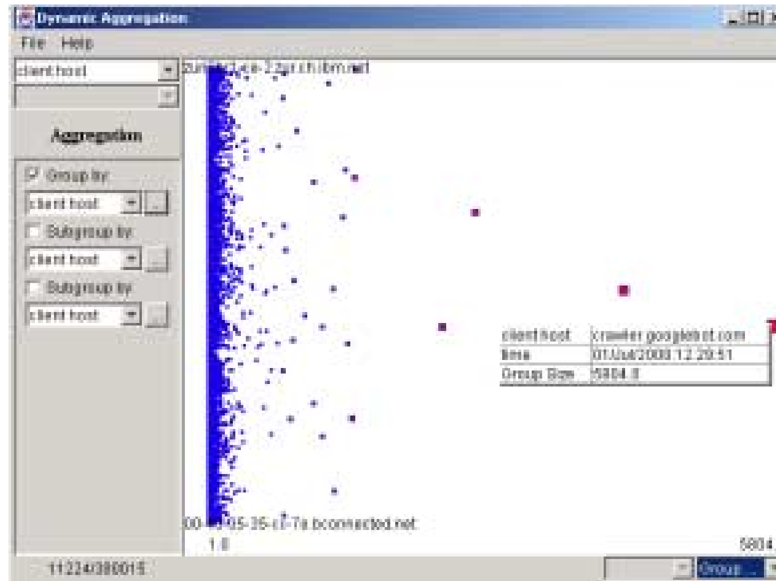

Fig. 10. Finding the most frequent visitor by aggregating by client host

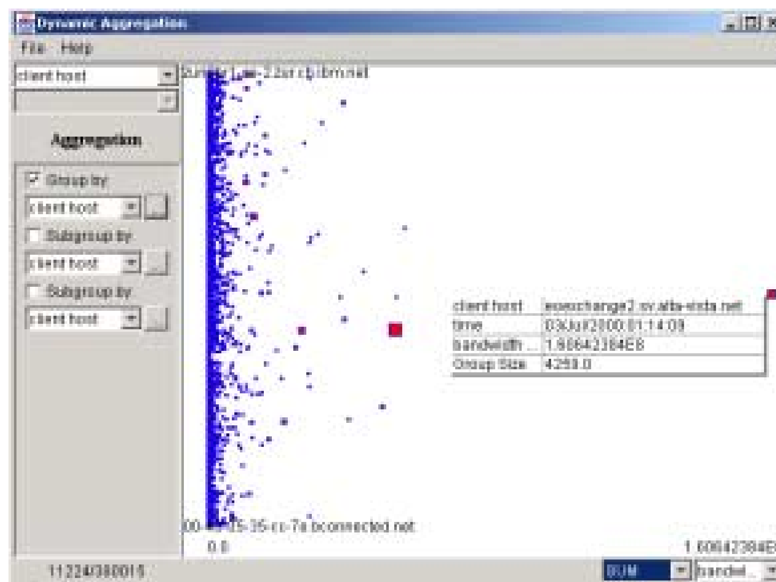

Fig. 11. Finding high bandwidth usages

We find that the Google spider the most frequent visitor of HCIL. To check out how much bandwidth did Google consume, we just need to change the field we are viewing to “Bandwidth used” and set the aggregator function to sum the field. From this graph, we find that it isn’t Google, but another crawler, EoExchange that is taking the most bandwidth. (Figure 11) We can also see that EoExchange consumes almost three times more bandwidth and has one quarter less requests.

Grouping by URL can similarly reveal which URL is the most popular, and how much each byte each URL used. A more interesting visualization is looking at the first time an URL is accessed (group by URL and use MIN on the time field); this might show related URL accesses. Since URLs that are close are logically related in someway, thus lines and clusters represent interesting information. The two vertical lines represent concurrent access of a set of related files. They turn out to be Java classes used in an applet. Seeing this, a web master might decide to make a jar out of those Java classes to save bandwidth. The horizontal line is a series of slides that was crawled by Google. This is interesting to see because the

crawling happened over three days, suggesting that Google tries to spread out crawling over time to avoid affecting the website performance.

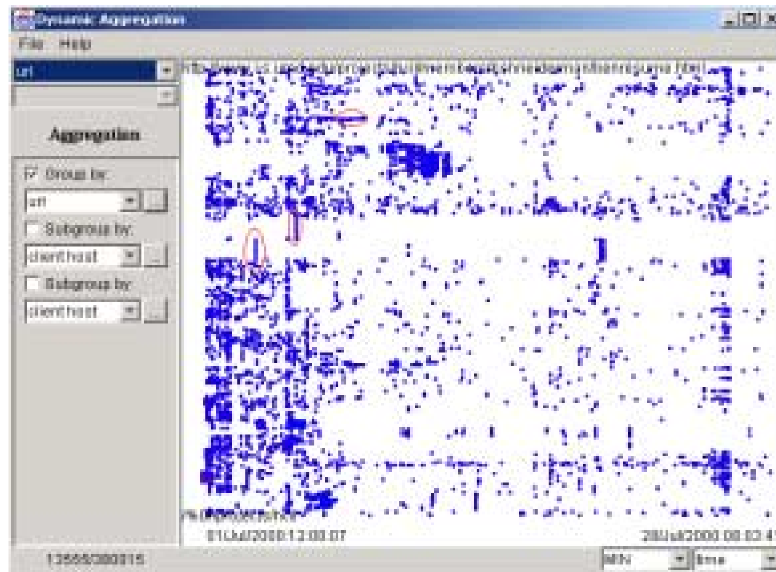

Fig. 12. Interesting access patterns

To find repeat visitors to a particular page, one can group first by user, then by URL. Thus the size of the dot is based on how many times a user used the web page. (Figure 13) This is an important metric for websites, because it is a measure of the site's "stickiness."

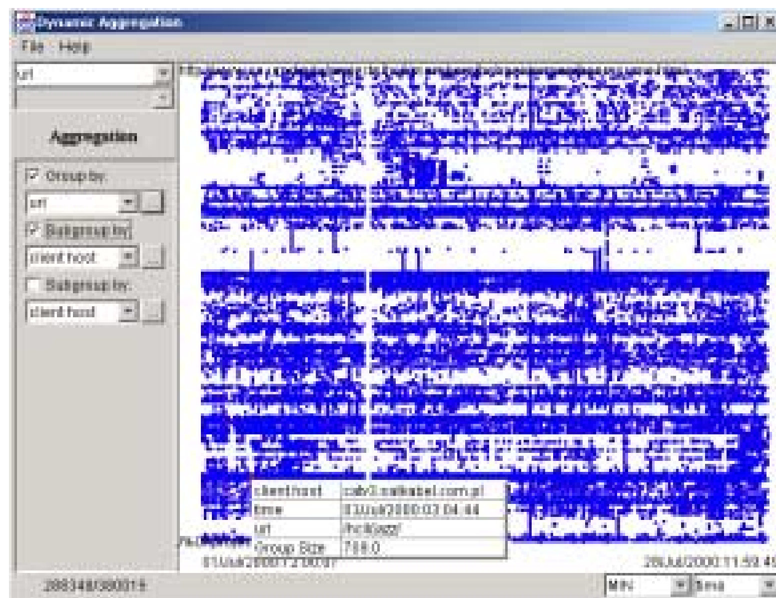

Fig. 13. URL stickiness

Viewing the user visits across the time domain will show usage patterns of users, and will show repeat users and one time visitors. Just grouping by time and clients using equivalence grouping gives a cluttered view.

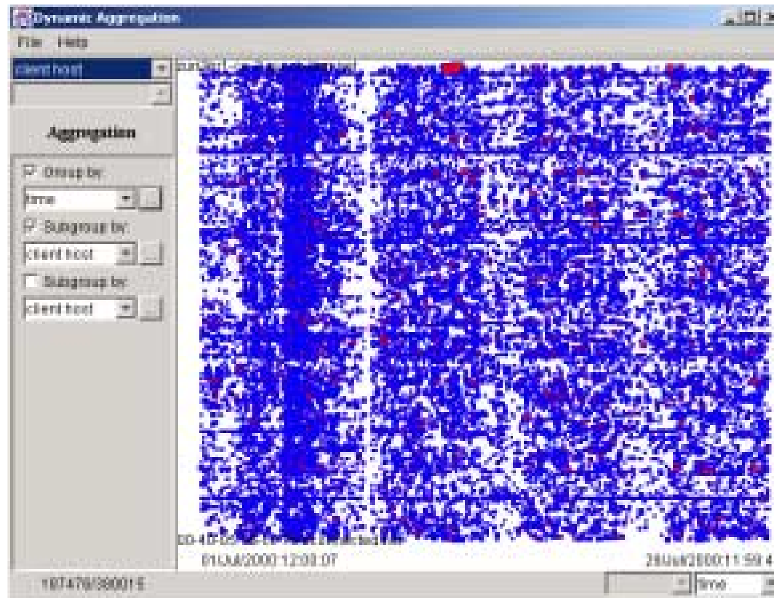

Fig. 14. User activity over time using equivalence grouping

Grouping by day shows us that most users are repeat users, indicated by the solid lines across time. The bandwidth hog EoExchange shows up in this graph as well. Google's accesses are well hidden and spread out across days, which validate the previous observation. (Figure 15)

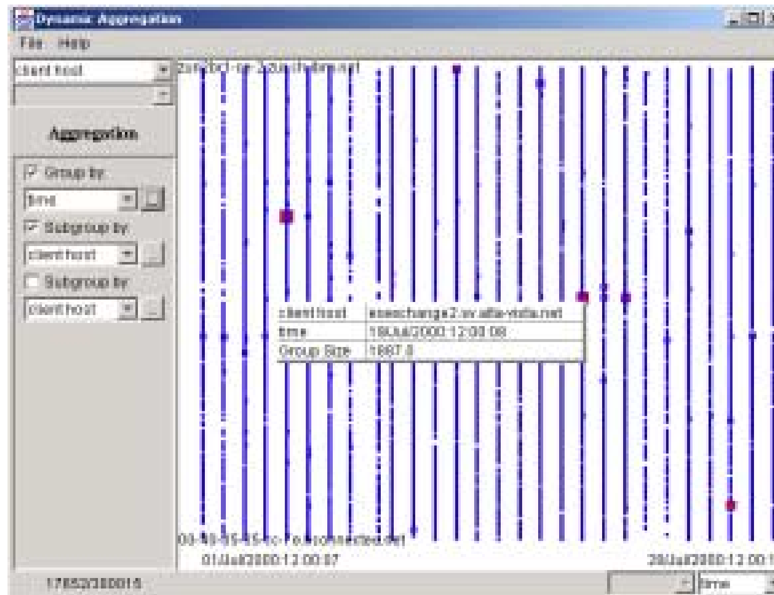

Fig. 15. User activity over days

4. Implementation and Design

The logical view of the system is a linear flow of data. Data flows from a data source and passes through many processors and the final products are values that can be used directly by the visualization.

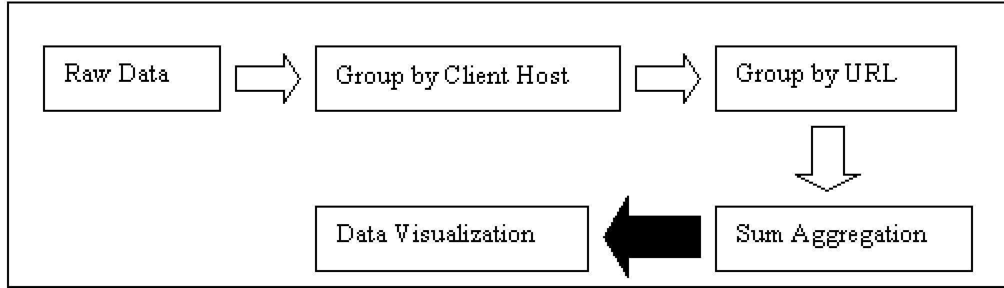

Fig. 16. The flow of converting raw data into visualization data

This is an abstraction that is used commonly in computer graphics (called rendering pipeline), and in audio/video processing (see Java Media Frameworks), and should be familiar to Unix users as well (pipes). The client of the data, the visualization, requests data from the end of the pipeline (a.k.a. pulling), and each processor request data from upstream. If the pipeline is empty, then the data flows directly from the data source and is processed along the way. Each processor should also implement caching, so that subsequent requests for data are fulfilled quickly. The actual implementation isn't nearly as simple as the design because of optimization issues. A *TextDataSource* is responsible for reading the data in from disk and is at the beginning of the pipeline. The last processor in the pipeline is the *Aggregator*, this class is responsible for the summing of information once the groups are created. Because this always applied at the end, it doesn't need to be integrated into the pipeline. AggregateDataSource is the class that does all of the work. It calls the *GroupingAlgorithm* class to break the data into groups, then uses the Aggregator to produce the output.

Table 1. Example of web log data

URL	Client host	Group by URL	Group by host	Group by both
/hcil/jazz	L4p26.dav.mother.com	0	0	0
/hcil/elastic-windows	J200.inktomi.com	1	1	4
/hcil/jazz	Dialin68.computeron.net	0	2	2
/hcil/jazz/learn/	L4p26.dav.mother.com	2	0	6

The most basic grouping algorithm is equivalence grouping. All data types support this grouping and is the default algorithm used in the system. Table 1 shows some sample data with the results of grouping. The first two fields are fields in the actual data, the next two fields show the results of grouping on one of those fields. The grouping algorithm will number the groups consecutively, starting from 0. Identical entries in the field that we are grouping by produce the same group number. We could use a hashing function to produce group numbers, but sequential groups makes creation of subgroups easier. Once grouping calculations are done for a particular field, it can be cached for later use. Subgroups can easily be created using the group ids of the fields used in subgrouping. Column 5 of Table 1 shows the group number when grouping by URL and Client host. The following simple formula can be used to generate subgroup ids:

$$subgroupID = Group1ID * NumberOfGroup2 + Group2ID \quad (1)$$

A problem with using the above equation is when dealing with large group ids, multiplying two large numbers might result in wrap around. Hashing based on the group ids can solve this problem. Using this technique is faster than direct calculations of the subgroups with no caching, since the grouping algorithms may take a long time to complete. The only draw back to this technique is that the memory requirements of caching can be quite large. A LRU or priority queue algorithm can be used to manage the cache space if memory is scarce. AggregationDataSource also caches the results of aggregation. It uses Java's hashtable to associate a field name with an array of values returned by the Aggregator object. The amount of data needed to store these data is much smaller due to the fact that the aggregated dataset is much smaller.

5. Future Work

Dynamic Queries provides a visual and interactive means of filtering of data. Filters are nothing more than another kind of data processor, and as such can be integrated with our system very easily. Integrating DQ into our current system should require little change to our design. Since DQ requires binning of values so each pixel on the visual control correspond to a group of data that can be filtered out, we can use the interval grouping or element frequency grouping algorithm that is already developed for dynamic aggregation. There are two alternatives to the position that DQ will occupy in the pipeline, one before aggregation and one after. Putting DQ before aggregation would probably be more powerful but since this changes the data upstream, all data downstream would have to be flushed and recomputed. This can be prohibitively slow for interactive requirements of DQ. Putting DQ after aggregation can also be useful and is much more computational tractable due to the smaller number of data items. The integration of DQ to our system would produce a system that can cover most of the data manipulation techniques.

Another area of importance is how to compute with much larger datasets. Currently, our system loads everything into memory before doing any computation. This is clearly unfeasible for datasets with gigabytes and terabytes of information. If aggregation means having to go through the entire database before an answer can be returned, the user would suffer a long delay before any real exploration of data can occur. A promising technique to integrate with would be combining the dynamic aggregation interface with "Online Aggregation" developed by Haas and Hellerstein [8]. Online Aggregation returns partial answers to database queries along with a confidence measure. This confidence measure is updated incrementally, meaning that partial results can be returned and viewed. In this way, query on a large database merely takes more time to get a good enough confidence measure, while the user can still explore the data returned.

6. Conclusion

We have developed a linear data flow design that produces groups to be used in aggregation, unlike the cyclical flow used in AM. Due to the simplicity of the system, we believe that users can understand the state of the system and will be able to use the tool effectively. However, due to the inherent complicity of the aggregation concept, users should have in mind a specific aggregation that they require. Unlike DQ, in which users can explore and experiment with data, aggregation should be thought of as creation of a new dataset. This new dataset can then be explored by DQ. A usability test should be conducted to test how readily users understand using the interface and which grouping algorithm and aggregation algorithm are needed to have a rich set of tools so the user can find answers to more complex questions than what was considered in the paper.

References

1. Shneiderman, Ben. (1994). "Dynamic Queries for Visual Information Seeking." *IEEE Software. 11(6), 70-77.*
2. Manson, Jeremy (1999) "Occlusion in Two-Dimensional Displays: Visualization of Meta-Data" http://www.cs.umd.edu/class/fall99/cmsc838s/Project/jmanson
3. Goldstein, Jade and Roth, Steven F. (1994) "Using Aggregation and Dynamic Queries for Exploring Large Data Sets" *Proceedings of ACM CHI'94 Conference on Human Factors in Computing Systems 1994 v.2 p.200*
4. Mei C. Chuah and Roth, Steven F. (1998) "Dynamic Aggregation with Circular Visual Designs" *Proceedings of Information Visualization, IEEE, North Carolina, October 1998.*
5. Rayson, James K. (1999) "Aggregate Towers: Scale Sensitive Visualization Decluttering of Geospatial Data" http://www.mitre.org/support/papers/tech_papers99_00/rayson_aggregate/index.shtml
6. Fredrikson, A., North, C., Plaisant, C. and Shneiderman, B. (1999) "Temporal, Geographical and Categorical Aggregations Viewed through Coordinated Displays: A Case Study with Highway Incident Data" *Proceedings of the Workshop on New Paradigms in Information Visualization and Manipulation, Kansas City, Missouri, November 6, 1999 (in conjunction with ACM CIKM'99), ACM New York, 26-34.*

7. Hochheiser, H., and Shneiderman, B. (2001) "Using Interactive Visualizations of WWW Log Data to Characterize Access Patterns and Inform Site Design" *Journal of the American Society for Information Systems*, 52(4), February, 2001.
8. Hellerstein, J.M. et al. (1997) "Online Aggregation" *Proceedings of the ACM-SIGMOD International Conference on Management of Data, 1997.*