Supporting Rapid Processing and Interactive Map-Based Exploration of Streaming News*

Michael D. Lieberman Hanan Samet
Center for Automation Research, Institute for Advanced Computer Studies,
Department of Computer Science, University of Maryland
College Park, MD 20742
{codepoet, hjs}@cs.umd.edu

ABSTRACT

The database architecture and system design of NewsStand, a database system that analyzes and displays streaming news using a map user interface, is described. Special emphasis is given to NewsStand's pipe server, which coordinates individual, independent analysis modules in a processing pipeline, and NewsStand's relational database schema, designed to accommodate responsive spatial querying and retrieval via NewsStand's user interface. Examples of these spatial queries, which are variants of top-k window queries, are also presented. Experiments on the live NewsStand database system demonstrate its capability for rapidly processing large amounts of streaming news as well as the interactivity of its map user interface as measured by database querying.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexin g; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retri eval

General Terms

Algorithms, Design, Performance

Keywords

Streaming news, Database design, NewsStand, Information extraction, Semi-structured data, Text mining

1. INTRODUCTION

In this paper, we describe the database architecture and system organization of the NewsStand database system [17, 20, 23] (found at `http://newsstand.umiacs.umd.edu`) that enables the rapid processing and map-based retrieval (see also the related QUILT system [19, 22] and the SAND Browser [18]) of streaming news. NewsStand constantly polls thousands of RSS feeds from news sources, downloads new articles, and performs a variety of processing on them, storing results in its PostgreSQL database to make these articles easily and quickly retrievable. Note that retrieving data by location is relatively easy in the case of explicitly spatial data (e.g., points, lines, polygons), while it is considerably more difficult for spatial data encoded as text (e.g., "Paris"), due to ambiguities in natural language. In particular, many names of places are also names of other entities (e.g., "Paris, France", a place, versus "Paris Hilton", a person) and also, many places share the same name (e.g., "Paris, France" versus "Paris, Texas" and over 60 other places named "Paris"). These ambiguities are resolved by NewsStand's *geotagger* [10–13], which associates news articles with the locations mentioned in them, thereby making the articles retrievable via spatial queries. Also, NewsStand's *clusterer* [24] groups together articles about the same topic, so that clusters can be retrieved by location as well.

Note that commercial aggregator systems such as Google News, Bing News, and Yahoo! News mainly focus on delivering articles from local news sources, based on the IP address of the visitor, rather than based on the content of articles. Also, they present such articles linearly, rather than through a map interface, which limits their querying capability. In addition, recent research has addressed and explored continuous queries over streaming data (e.g., [6, 7]), and many research systems explore streaming news as well as other media such as images, audio and video (e.g., PersoNews [3], Newsjunkie [4], Europe Media Monitor [8]).

This paper differs from our earlier work [10,12,13,16,23], which describe elements present in NewsStand's backend processing, by instead focusing on the database architecture and system organization that enables the smooth and rapid operation of NewsStand's processing and retrieval capabilities. NewsStand's design incorporates several features that allow it to handle large amounts of streaming news data (about 50,000 articles per day) as well as a large underlying database (over 300GB of data). The architecture including its processing modules and a description of data flow through the system is described in Section 2. Central to the NewsStand database operation is its *pipe server*, described in Section 3, which acts to coordinate NewsStand's many backend processing modules by assigning batches of processing work via a communication protocol, and also monitors the system's health by verifying that documents are being processed smoothly. Where the pipe server directs modules to perform work, NewsStand's *SQL database*, based on PostgreSQL and described in Section 4, stores information about documents present in the system, as well as the results of their processing. This database design evolved from that used in STEWARD [14], a spatio-textual search engine with similar processing and querying capabilities. After processing the input news documents, NewsStand's user interface retrieves location-associated news clusters to display in its map user interface by executing variants of what are known as *top-k window queries*. These queries, described in Section 5, retrieve the k highest scoring news clusters in NewsStand's database that are located in the map's current view-

*This work was supported in part by the National Science Foundation under Grants IIS-07-13501, IIS-08-12377, CCF-08-30618, IIS-09-48548, IIS-10-18475, and IIS-12-19023.

ACM SIGSPATIAL GIS '12, November 6-9, 2012. Redondo Beach, CA, USA

ing window. Experiments in Section 6 show the voluminous nature of streaming news processed by the NewsStand database system, and demonstrate its ability to rapidly process streaming news.

2. ARCHITECTURE

NewsStand's backend processing is organized as a pipeline, with individual *slave modules* running on multiple computers. Documents stream into the pipeline and flow through various stages of processing, performed by slave modules connected to the pipeline. Each module performs a different type of processing on documents entered to the system, with later modules in the pipeline often depending on results of earlier modules. The main challenge in designing NewsStand's architecture was coordinating these modules so that they can execute simultaneously without excessive idle time.

To address this challenge, NewsStand features two centralized sources of control and synchronization:

1. A **pipe server** tracks documents as they flow through the processing pipeline and assigns documents to slave modules.
2. An **SQL database** stores information about documents and results of each processing stage.

Both the pipe server and database have specific communication protocols to which slave modules must adhere. Also, notice that this arrangement decouples NewsStand's control channel (pipe server) from its data channel (database).

Figure 1 shows a graphical overview of NewsStand's architecture. Slave modules are shown as rounded rectangles in blue, with arrows indicating data flow. In the center are NewsStand's pipe server and SQL database. NewsStand's Web interface accesses the central database to retrieve data for display. Each retrieval action is posed in terms of a corresponding SQL query within NewsStand's database. These queries are detailed in Section 5. Below, we provide brief descriptions of each slave module type, which will aid the understanding of NewsStand's database schema (described in Section 4). NewsStand's slave modules include:

1. **RSS Grabber**: Polls RSS feeds and retrieves URLs to news articles.
2. **Downloader**: Downloads HTML news articles from URLs.
3. **Cleaner**: Extracts article content from source HTML.
4. **Clusterer**: Groups together articles about the same story.
5. **Topic Classifier**: Assigns general topic types to articles (e.g., "Business", "Sports").
6. **Geotagger**: Finds textual mentions of geographic locations and assigns lat/long values to each.
7. **People/Disease Finder**: Finds textual mentions of people, diseases, and other entities.
8. **Media Extractor**: Extracts images and videos, and captions associated with them.

Document processing proceeds as follows. When a slave module instance starts, it connects to both the pipe server and database. The pipe server then sends document identifiers to the slave instance. For each document, the slave instance retrieves information about that document from the database, performs its processing on the document, and stores the results in the database. When all documents have been processed, the slave reports back to the pipe server that the batch is finished, and receives another batch of documents to work on. Document identifiers are added to the pipe server by the first module in the pipeline, namely the RSS grabber. Note that individual instances of modules are directed by the pipe server to work on independent batches of documents. In this way, processing bottlenecks are avoided by starting additional instances of slave modules that require more processing time, allowing greater scalability. The pipe server and its protocol are described in more detail in Section 3, while the database layout is presented in Section 4.

3. PIPE SERVER

The pipe server serves as the control system for NewsStand's slave modules. It maintains a collection of work queues called *pipes*, with one pipe per slave type. Each pipe contains a number of document identifiers, referred to as *docid*s, which correspond to documents moving through stages of processing. Each slave module connects to the pipe server to receive work batches of *docid*s that are intended for an instance of that slave type. In addition to passing work batches to slaves, the pipe server is well-positioned to track statistics related to document processing, such as how many have been processed by each module and how quickly. Furthermore, the pipe server knows which slave modules are currently running, and can be used to track and restart misbehaving or erroneous slaves, either manually or with an automated script. Note that the *docid*s traveling through these pipes correspond to documents in NewsStand's database. However, it is interesting that from the perspective of the pipe server, these *docid*s are simply numbers to be tracked, because the pipe server does not connect to the database directly for reliability reasons. That these numbers correspond to database documents is incidental to its operation.

Several factors influenced the pipe server's construction. First and foremost, being the central controlling software in NewsStand, it must be highly reliable, and never go down or crash. It should have reasonable memory usage and not have significant external dependencies, such as requiring a full-fledged database to function properly. Furthermore, it should be resilient to unreliable slave modules, which could disconnect at any time, either explicitly, or due to software bugs or networking problems. All of these cases must be handled gracefully. In addition to reliability, speed of processing is a key factor in the design. Because new articles are constantly streaming in to NewsStand, the pipe server must not be a bottleneck in articles' processing time.

The pipe server and its communication protocol have several features that address these goals. First, the pipes and the *docid*s contained in them are stored in a disk-based hash, which does not depend on NewsStand's database. As a result the *docid*s move very quickly through the pipes so that they can keep up with the very rapidly incoming new data. Also, when communicating with slave modules, rather than waiting for immediate responses from each slave, which could slow processing, the pipe server employs non-blocking input/output and buffering. Each connected slave is tracked individually with regard to the work batch sent to it, and this work does not move to the next pipe until the slave sends back a valid work complete response. Note that this design assumes that slaves are not malicious (e.g., reporting that work was finished when it was not). The main drawback of this design is that the pipe server amounts to a single point of failure. If the pipe server does halt, NewsStand's processing will also cease. However, over months of measurement, we found that the pipe server's stopping was due to rebooting the server on which it runs, rather than reliability issues introduced by its design.

4. DATABASE DESIGN

There are two main goals behind this design:

1. Managing the large amount of data streaming through NewsStand and derived from its processing.
2. Serving NewsStand's map query interface quickly to facilitate interactive browsing and exploration of news.

To address the first goal, NewsStand employs a central PostgreSQL relational database that holds all data downloaded and generated from processing. Due to our domain of streaming news, data is constantly being added to and deleted from this database, unlike typical SQL databases where the data is mostly static. This data churn can wreak havoc on performance and must be managed care-

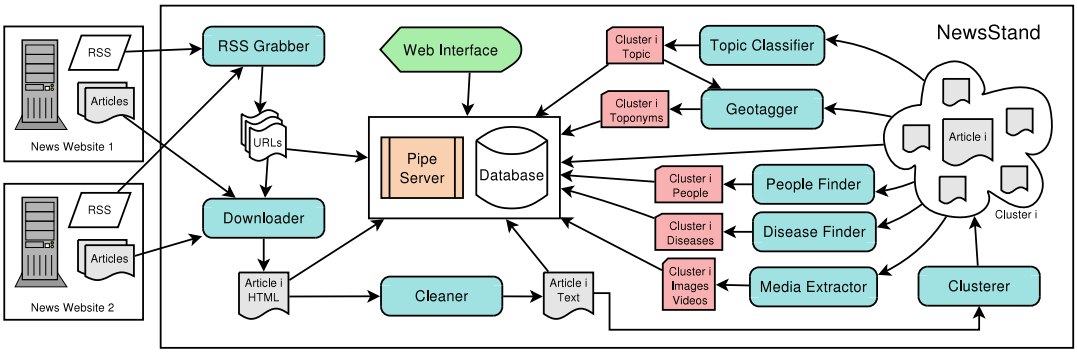


Figure 1: NewsStand's architecture, including slave modules (blue) and intermediate output (red). Arrows indicate data flow.

fully. For example, database query planners track statistics about data value distributions within these tables to determine efficient query execution plans given a particular query. Heavy data churn means that these statistics will quickly go out of date and must be updated frequently. In PostgreSQL, this operation is known as *vacuuming*, and we perform vacuuming regularly in the database. To address the second goal of serving NewsStand's map query interface quickly, we maintain a separate *cache database* containing only the most recent data, which improves querying performance (see Section 5 for a description of such queries).

Figure 2 provides an overview of NewsStand's database schema, showing the tables used for NewsStand's core data processing. Tables are color coded by purpose: Red relates to individual documents (`feeds`, `docs`, `doc_text`, `media`), blue for clusters of documents (`clusters`, `cluster_locs`), and because locations play such an important role in all of NewsStand's querying, we give special attention to geotagging tables, shown in green (`entities`, `doc_locs`). Also, indexed attributes are indicated by filled circles next to the attribute names. Each table's tuples have one identifier as primary key (shown as filled diamonds next to attribute names), and these identifiers are used in numerous foreign key constraints, shown as arrows, which enforce data integrity. Note that Figure 2 is a simplified depiction of NewsStand's database, which currently contains over 100 tables, but these few serve as a useful illustration of the core data and functionality present in NewsStand, as well as the general design strategy.

In subsequent sections, we describe the tables for documents (Section 4.1), geotagging (Section 4.2), and clusters (Section 4.3), as well as how they are populated by NewsStand's processing modules (see Section 2 for a brief description of these modules). Also, we describe NewsStand's cache database in Section 4.4.

4.1 Documents

Tables related to document processing are shown in red in Figure 2. Document processing begins with the `feeds` table, which holds information about the RSS feeds that NewsStand's **RSS grabber** polls to inject new documents into the system. Along with a feed `name` and `url`, a timestamp `last_retrieved` and time interval `last_interval` are stored with each feed. The latter two attributes are used by the RSS grabber to adjust the polling intervals of RSS feeds. If the RSS grabber polls a given feed and finds new data, then the polling interval is decreased; otherwise, it is increased. In this way, a feed's polling interval gradually moves toward its data generation rate. Finally, an additional `parent_id` field stores an optional pointer to a "parent" RSS feed. This field is filled for multiple RSS feeds that belong to the same news source (e.g., a single newspaper with multiple feeds corresponding to different newspaper sections), and allows quick retrieval of all feeds from a single news source.

While polling RSS feeds, the RSS grabber populates the `docs` table, which holds information about individual documents. For each document, the `feed_id` value is set to that of the feed from which it came, and the document's publication date, source URL, title, and snippet of text, all of which come from the RSS feed, are stored in the `pub_date`, `url`, `title`, and `snippet` attributes, respectively. The `cluster_id` and `topic` fields are populated by NewsStand's clusterer and topic classifier, to be described shortly.

The next phases of processing involve the `doc_text` table, which stores text versions of the document. After the `docs` entry for a given document d has been created, d is downloaded by NewsStand's **downloader** module, and this HTML version of d is added to the `html` field for d. Next, NewsStand's **cleaner** module takes the downloaded HTML version, and produces a cleaned version, storing the result in the `clean` field. Finally, the document's clean text is mapped to its vector space representation and stored in the `vector` field, which facilitates full text indexing and keyword searching. Queries involving a keyword component which leverage this vector are further described in Section 5.

After retrieving the document's text, images [21], videos, and other media which are found by NewsStand's **media extractor**, they are stored in the `media` table. Several relevant attributes are stored along with each media entry, including the media's `type` (e.g., "image", "video"), `url`, embedding `html`, `caption` as determined from the embedding structure or other methods (e.g., image `alt` tags), and `width` and `height`.

4.2 Geotagging

NewsStand's **geotagger** module involves two steps: First, finding all textual references to geographic locations (called *toponyms*) in a document, and then for each toponym, assigning the proper lat/long values for the toponym. These steps are also referred to as *toponym recognition* and *toponym resolution*. Refer to Lieberman et al. [10–13] for fuller descriptions of NewsStand's geotagger.

Several tables in NewsStand's database are populated by the **geotagger** module. The most important of these are the `entities` and `doc_locs` tables, shown in green in Figure 2, which contain entities and locations found during toponym recognition and resolution, respectively. `entities` is populated with entities found in the toponym recognition process, such as person names, organizations, and locations. Each entity is associated with the `doc_id` of the document from whence it came, as well as the `start` and `end` offsets within the document, the `type` of entity, the text `phrase` of the entity, and the entity's `score` which is determined during the entity recognition process. This `score` can reflect, for example, the confidence that the given entity is correct. Of course, toponym recognition is nominally only concerned with location entities; however, in the course of toponym recognition, knowledge that a given term or pattern often signals an entity of some other type can aid in distinguishing locations from non-locations (e.g., knowing that "Mr." often precedes a person's name), and these en-

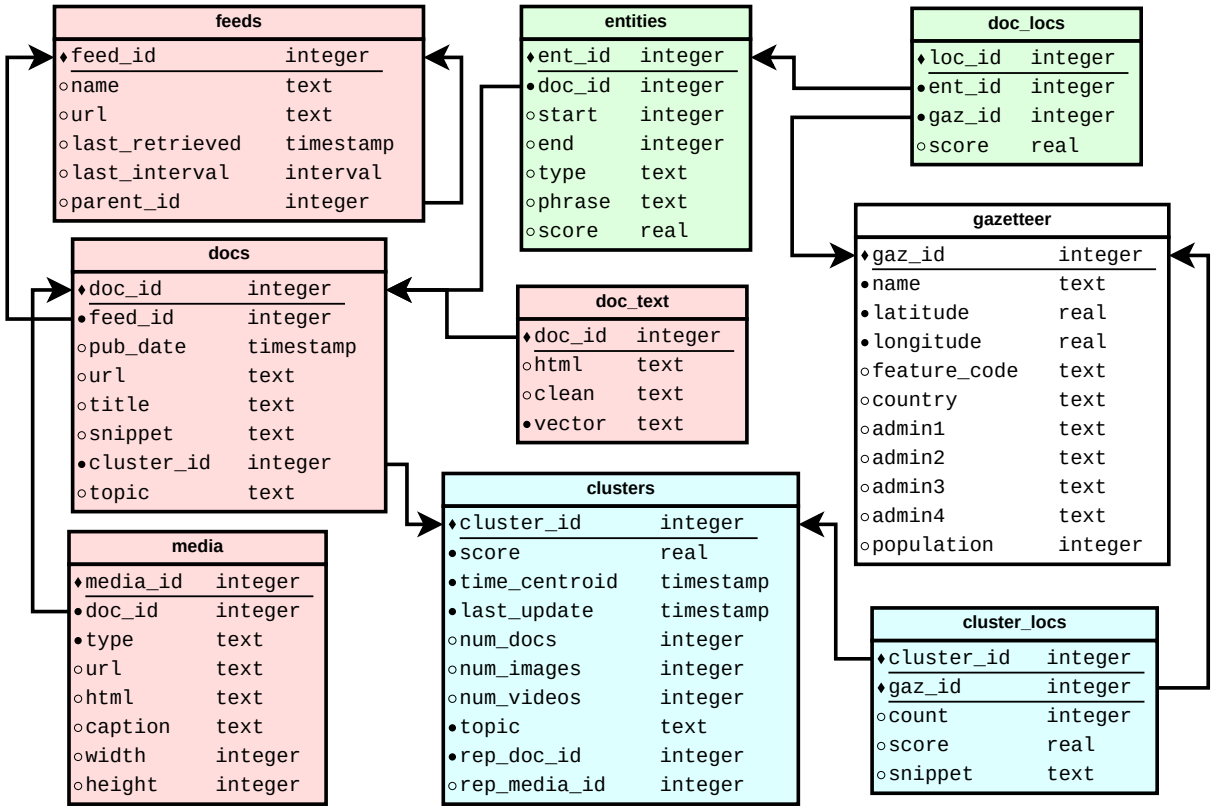


Figure 2: NewsStand's database schema, featuring tables used in core processing and querying. Tables related to individual documents (red), geotagging (green), and clustering (blue) are shown. Arrows indicate foreign key constraints, and indexed attributes are shown by filled shapes adjacent to attribute names (filled diamonds for primary keys, filled circles for other attributes).

tities are also stored. Other modules, such as the **people finder** and **disease finder**, also store entities in the `entities` table.

After finding entities in toponym recognition and populating the `entities` table, the geotagger proceeds with toponym resolution to resolve any geo/geo ambiguities and assign final lat/long values to each location entity in `entities` (i.e., the document's *toponyms*). Lat/long values are obtained from a *gazetteer*, a list of locations and metadata, which is stored in the `gazetteer` table. Our gazetteer is based on GeoNames, a crowdsourced gazetteer containing over 7.5 million locations. Output from the toponym resolution process is stored in the `doc_locs` table, which serves to tie together a toponym entity (`ent_id`) and a gazetteer entry (`gaz_id`). Also, a toponym `score` reflects the toponym's importance within the document, based on frequency and distribution, and is used to find the document's geographic focus (i.e., central location associated with it). Note that the `doc_locs` table does not store location information explicitly (e.g., lat/long values), which is instead stored in the `gazetteer` table. This design centralizes location information to enable easy updates, which occurs nightly when the `gazetteer` table is synchronized with GeoNames.

4.3 Clusters

NewsStand's **clusterer** plays a central role in processing streaming news. Tables involved with clusters of documents are shown in blue in Figure 2. The `clusters` table aggregates information about clusters. It contains the cluster's importance `score`, computed as a combination of several factors such as freshness of the documents in the cluster, cluster size, rates of growth such as velocity and acceleration, and diversity of news sources within the cluster. The `score` is used extensively when retrieving clusters to display in NewsStand's user interface (Section 5). Additional information stored with each cluster includes the time centroid of documents in the cluster (`time_centroid`), time of last cluster update (`last_update`), and number of documents, images, and videos in the cluster (`num_docs`, `num_images`, `num_videos`). Also, the cluster's `topic` is computed based on the topics of documents contained within it, and a representative document and image are chosen to display along with the cluster (`rep_doc_id`, `rep_media_id`).

After geotagging the documents in the cluster, the cluster itself is associated with locations found in the documents. These locations are stored in the `cluster_locs` table, which, analogously to `doc_locs`, ties together a cluster (`cluster_id`) and a location (`gaz_id`). Also stored are the number of times the location appears in documents in the cluster (`count`), how relevant the location is to the cluster as a whole, determined by the scores of instances of that location in the cluster (`score`), and a text `snippet` from the cluster's representative document that contains the location. This snippet is shown in NewsStand's interface, to give users an understanding of the location's relevance to the story.

4.4 Cache Database

NewsStand's main database serves as a data repository, where the main goals are consistency, reliability, and enough space to hold the large amount of news flowing through NewsStand. NewsStand downloads about 50,000 new documents each day, and stores the most recent four months' of news, including the documents' actual text rather than simply a URL, at any given time. While we plan to eventually store more than four months of news, to enable more interesting analytic applications, we currently do not due to a lack of database disk space. Also, document processing generates much more data to be stored in the database. NewsStand's process-

ing modules constantly communicate with the database, resulting in heavy query traffic with many tuple changes (i.e., inserts, updates, and deletes). These changes could cause `SELECT` statements to block while waiting for transactions to complete. As a result, this database is not suitable for serving NewsStand's user interface, where the goal is interactive query speed.

Instead, for serving the user interface, we maintain a separate, smaller *cache database* containing only the most recent news data (several days' worth). The cache database has mostly the same schema as the main database, except with less data. Smaller tables result in more table rows residing in the database's cache buffers, rather than having to go to disk to respond to queries. In addition, since NewsStand's user interface does not modify the main database, queries will not block on waiting for modifications. Of course, the cache database needs to be updated from the main database in a timely manner so that the latest news is always being served. A special cache updating module continually polls the main database for clusters whose `last_update` value is newer than the previous update time, and copies the cluster and its associated information to the cache. In this way, updates to the cache database are minimized and more database query processing resources are used for serving the interface.

5. DATABASE QUERIES

Here, we describe some of the database queries used to retrieve data for NewsStand's map user interface. With NewsStand's database schema (described in Section 4), each interface action is easy to cast in terms of an SQL query. Broadly, there are two modes of using NewsStand, corresponding to the two basic queries that it supports:

1. **Top stories mode**: Where is topic *X* happening?
2. **Map mode**: What is happening at location *Y*?

Note that NewsStand's database contains far more information than is feasible to send over a network connection and display in a client's user interface. Instead, we are only interested in the top few results for each query. To this end, as we will see in the following sections, the queries used within these two modes are all variants of what is known in database parlance as *top-k queries*, i.e., queries that return the first k ranked results according to some ranking function, which varies depending on the particular query. In addition, many more queries are used throughout NewsStand's interface than are presented here, due to lack of space. Instead, we only present the queries associated with story and location retrieval, though these queries serve to illustrate the principle of top-k retrieval that applies to all of NewsStand's interface queries.

Additionally, the queries introduced below feature *joins* among multiple tables in the database, which often hinder query performance. However, these joins are presented for conceptual understanding rather than reflecting the actual implementation of these queries. As noted in Section 4.4, in order to improve the speed of querying and hence improve interactivity, the user interface queries are executed in the cache database rather than in NewsStand's main database. When executing these queries, we use what are called *materialized views* of the query results, which amount to precomputation and storage of the query results prior to runtime. For example, in the query "`SELECT * FROM a, b WHERE a.id = b.id AND a.val1 = 'X' AND b.val2 = 'Y'`", creating a materialized view of this query would involve executing the query's join portion, namely "`SELECT * FROM a, b WHERE a.id = b.id`", and storing the result. When executing the original query, the filter conditions ("`a.val1 = 'X' AND b.val2 = 'Y'`") would only need to be applied to the materialized view result, without any joins at runtime. This optimization further improves cache query performance.

Below, we continue with descriptions of NewsStand's interfaces and queries that return clusters in top stories mode (Section 5.1) and map mode (Section 5.2), as well as queries involving a single cluster used in both modes (Section 5.3).

5.1 Top Stories Mode

Answering queries of the form "Where is topic *X* happening?" (i.e., feature-based queries [2]) is the purview of NewsStand's first mode, referred to as "top stories mode" or "text mode". Figure 3 shows NewsStand's interface in top stories mode. Assuming a landscape display, the left pane shows the top-k story clusters, ranked in importance from top to bottom of the visible part of the display screen. This pane is populated by one of several queries presented in Figure 4, depending on filtering parameters. Figure 4a is the basic query to populate the pane, which retrieves clusters in order of cluster score. Additionally, several links at the top left allow for filtering of the top-k clusters according to their topic types (e.g., "Business", "Sports"), while at top right, a menu allows selecting clusters with articles from particular news feeds, and a keyword search allows the selection of clusters relevant to particular keywords. These functionalities are implemented using the SQL presented in Figures 4b, 4c, and 4d, respectively, which are all variants of the basic top-k query of Figure 4a except with additional constraints. For the keyword query of Figure 4d, the cluster rankings are modified to include a keyword relevance measure in addition to the cluster's score.

As the mouse is hovered over the clusters in the left pane, the most relevant locations to the selected cluster are displayed on the map in the right pane of the display using what we term "markers" which are icons corresponding to the most dominant topic type of the elements of the cluster. This action corresponds to the feature-based query of "Where is topic *X* happening?", in that its input is one of the news clusters, and its output is the set of locations that are relevant to the cluster. Also, note the presence of a slider at the top of the right pane, whose movement to the right (left) allows the maximum number of locations for which icons are displayed for the highlighted cluster to be increased (decreased). The identity of the locations for which icons are present depends on the number of times the location is mentioned in the articles that make up the cluster, with priority given to those that are mentioned most frequently. The presence of this slider is precisely the novel aspect of NewsStand that enables it to answer top-k variant of the feature-based query of "Where is topic *X* happening?" The algorithm that performs the display ensures that all of the desired locations can be seen, and thus the area displayed is the minimum bounding box of the locations. In database parlance, what we have here is an instance of a top-k query where k corresponds to the number of visible locations for a particular cluster or a cluster that contains a particular keyword. That is, we have a "top-k locations" query. This query is implemented in SQL as shown in Figure 7a, to be described shortly.

5.2 Map Mode

NewsStand's second mode, referred to as "map mode" or "full screen mode" and shown in Figure 5, is used for answering queries of the form "What is happening at location *Y*?" (i.e., location-based queries [2]). In this case, NewsStand provides the capability of reading over 10,000 newspapers (RSS feeds) by using a map. As mentioned earlier, the result of processing the RSS feeds is a set of clusters of articles by topic. Initially, the map contains topic icons at locations corresponding to those in the k most representative clusters, where "representative" takes into account factors such as importance measured by currency, size and rate of growth of the cluster in terms of velocity and acceleration, as well as a desire to have a good spatial distribution in the area being displayed. A variant of the feature-based query can also be executed in map mode. This is done by entering a keyword in the "search" box at upper right. The result is a set of k clusters with respect to the designated keyword, displayed using topic icons at the clusters' locations. Once a search has been activated, all subsequent searches will be restricted to the keyword. However, searches are restricted to the displayed part of

Figure 3: NewsStand's top stories mode.

```
SELECT c.cluster_id FROM clusters c
ORDER BY c.score LIMIT k
```

(a) Get top clusters

```
SELECT c.cluster_id FROM clusters c
WHERE c.topic = t
ORDER BY c.score LIMIT k
```

(b) Get top clusters of a given topic t

```
SELECT DISTINCT c.cluster_id
FROM clusters c, docs d
WHERE c.cluster_id = d.cluster_id
   AND d.feed_id = fid
ORDER BY c.score LIMIT k
```

(c) Get top clusters with articles from feed fid

```
SELECT DISTINCT c.cluster_id
FROM clusters c, docs d, doc_text dt
WHERE c.cluster_id = d.cluster_id
   AND d.doc_id = dt.doc_id
   AND match(dt.vector, kw)
ORDER BY kwrank(c.score, dt.vector, kw) LIMIT k
```

(d) Get top clusters with keyword kw

Figure 4: Queries used to populate the left pane of top stories mode.

the map (i.e., they are spatially restricted and are analogous to a spatial join operation).

Again, while in map mode, a slider is present at the upper right corner of the map whose movement to the right (left) effectively allows the maximum number of different clusters for which topic icons are displayed at their representative locations to increase (decrease). Although this feature seems very similar to its analog in top stories mode, its semantics are actually very different. In particular, each additional location corresponds to potentially increasing the number of viewable clusters although this is achieved by increasing the number of locations for which icons are displayed. The presence of this slider represents a second novel aspect of NewsStand as it enables it to answer the location-based query of the form "What is happening at location *Y*?" where, contrary to conventional assumptions, *Y* is not a location, but is actually a region corresponding to the part of the world that is viewable. Thus, moving the slider to the right increases the number of clusters that could be viewed, as these clusters are associated with the various locations, although the clusters associated with the additional location could be the same as the clusters associated with the existing viewable locations. Thus, we see that the number of viewable clusters resulting from moving the slider to the right is non-decreasing. In database parlance, what we have here is an instance of a top-k query where k corresponds to the number of viewable clusters. In other words, we have a "top-k clusters" query. That is, the resolution (also referred to as the zoom level, but measured here in terms of the number of clusters that are visible, in contrast to the conventional definition which is in terms of visible area) also increases.

Another database analog of the top-k clusters query is a ranked spatial range query or a ranked spatial join query. The challenge in implementing this query lies in deciding on the order in which the loations corresponding to clusters are delivered to the user, which is a function of their importance. This need not necessarily be the number of times they are mentioned. For example, it could be based on the number of clusters in which they appear at least once. Another factor could be their currency in terms of the time at which they arrive, and the velocity and acceleration of the cluster's rate of growth. Again, these issues arise because our data is dynamic on account of our streaming environment. Given the above analogies of top stories mode and map mode with the top-k query, it is also natural to let k vary, where k denotes the number of markers (topic type icons), which is achieved by using the slider.

Figure 6 presents examples of top-k cluster queries used in map mode. Figure 6a illustrates the default mode, where a set of clusters are used to populate the visible map window. The `clusters` table is joined with `cluster_locs` to retrieve the cluster locations, as well as with the `gazetteer` table to retrieve lat/long information, which in turn is used to filter the clusters to only include those that lie within the query window qw. Results are ordered by the cluster's score. Notice that this query is related to that of Figure 4a, except for an additional query window constraint on the cluster's locations. Also, as before, we use variants of this query to add more query constraints involving topics, source feeds, and keywords, which are shown in Figures 6b, 6c and 6d.

5.3 Single-Cluster Queries

In this section, we present queries that retrieve information about a single cluster. These queries are used in both top stories mode and map mode, since both modes involve the retrieval and display of news clusters and information associated with them. As with previous user interface operations, these actions are easy to cast in terms of SQL using NewsStand's database schema. These queries are presented in Figure 7, and are described in detail below.

Recall from Section 5.1 that the primary form of queries in top

Figure 5: NewsStand's map mode.

```
SELECT DISTINCT c.cluster_id
FROM clusters c, cluster_locs cl, gazetteer g
WHERE c.cluster_id = cl.cluster_id
   AND cl.gaz_id = g.gaz_id
   AND contains(qw, g.latitude, g.longitude)
ORDER BY c.score LIMIT k
```

(a) Get top clusters in query window qw

```
SELECT DISTINCT c.cluster_id
FROM clusters c, cluster_locs cl, gazetteer g
WHERE c.cluster_id = cl.cluster_id
   AND cl.gaz_id = g.gaz_id
   AND contains(qw, g.latitude, g.longitude)
   AND c.topic = t
ORDER BY c.score LIMIT k
```

(b) Get top clusters in query window qw with topic t

```
SELECT DISTINCT c.cluster_id
FROM clusters c, cluster_locs cl, gazetteer g
WHERE c.cluster_id = cl.cluster_id
   AND cl.gaz_id = g.gaz_id
   AND c.cluster_id IN (
      SELECT d.cluster_id FROM docs d
      WHERE d.feed_id = fid
   )
   AND contains(qw, g.latitude, g.longitude)
ORDER BY c.score LIMIT k
```

(c) Get top clusters in query window qw with articles from feed fid

```
SELECT DISTINCT c.cluster_id
FROM clusters c, cluster_locs cl,
   doc_text dt, gazetteer g
WHERE c.cluster_id = cl.cluster_id
   AND cl.gaz_id = g.gaz_id
   AND c.rep_doc_id = dt.doc_id
   AND contains(qw, g.latitude, g.longitude)
   AND match(dt.vector, kw)
ORDER BY kwrank(c.score, dt.vector, kw) LIMIT k
```

(d) Get top clusters in query window qw with keyword kw

Figure 6: Queries used in map mode.

```
SELECT * FROM clusters c, cluster_locs cl, gazetteer g
WHERE c.cluster_id = cl.cluster_id
   AND cl.gaz_id = g.gaz_id
   AND c.cluster_id = cid
ORDER BY cl.score LIMIT k
```

(a) Get top locations for cluster cid

```
SELECT * FROM clusters c, docs d,
   feeds f, media m
WHERE c.rep_doc_id = d.doc_id
   AND d.feed_id = f.feed_id
   AND c.rep_media_id = m.media_id
   AND c.cluster_id = cid
```

(b) Get cluster summary for cluster cid

```
SELECT * FROM docs d
WHERE d.cluster_id = cid
ORDER BY d.pub_date LIMIT k
```

(c) Get all articles in cluster cid

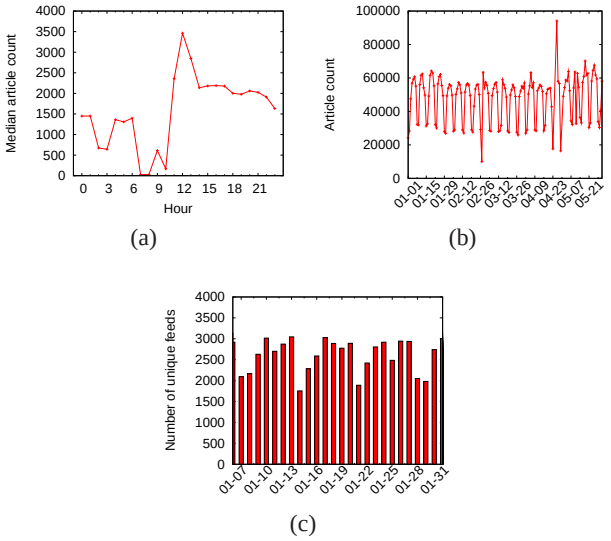
```
SELECT * FROM docs d, media m
WHERE d.media_id = m.media_id
   AND d.cluster_id = cid
   AND m.type = t
ORDER BY d.pub_date LIMIT k
```

(d) Get media of type t for cluster cid

Figure 7: Queries to retrieve single-cluster information.

stories mode is "Where is topic X happening?", where X corresponds to a cluster of news. This query corresponds to Figure 7a, which retrieves the top-k locations associated with a given cluster with identifier cid. The locations are retrieved in order of their relevance score to cluster cid.

In addition, several queries are used to render cluster information in NewsStand's interface. In both top stories mode and map mode, summary information about each cluster is displayed, in different ways. In top stories mode, each cluster is shown with its representative article's title, along with a text snippet from the article, its last update, total number of documents in the cluster, and so on. These data are retrieved with the query in Figure 7b. Additional links in the left pane allow users to retrieve all documents, images, or videos associated with a single cluster. Documents are retrieved using the query in Figure 7c which takes data from the `docs` table, while the retrieval of images and videos is accomplished with the query of Figure 7d, which retrieves media of the appropriate type from the `media` table.



(a) (b) (c)

Figure 8: Number of articles retrieved by NewsStand (a) per hour and (b) per day, as well as (c) the number of unique feeds supplying these articles.

6. EXPERIMENTS

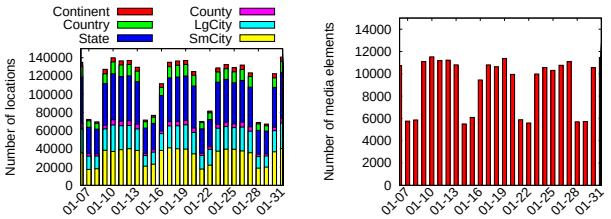
This section describes the results of several experiments intended to characterize the amount of streaming news collected by the NewsStand database, as well as its processing and querying time. Unlike typical system evaluations on news data, which use small, static corpora of news from a single generally prominent source (e.g., Reuters, New York Times), these experiments were conducted on the live NewsStand database system over several months' worth of streaming news data. As a result, they better characterize NewsStand's long term performance on streaming news.

6.1 Data Collection

First, we measured how frequently articles are collected and entered into the NewsStand database through the many RSS feeds that it polls. Figures 8a and 8b present statistics about the number of articles retrieved by NewsStand from RSS feeds per hour (measured using Eastern Standard Time) and per day, respectively, as measured over a five month period from January to May, 2011. Both figures show the large volume of news processed by the NewsStand database every day, which dwarfs typical corpus sizes (i.e., hundreds of documents) used in information retrieval and geotagging research (e.g., [1, 5, 9, 15]). They also show the generally cyclical publishing rate of sources polled by NewsStand, with most articles being published during daytime hours and during the week, rather than on weekends. The clustering of article publishing times between the hours 11–13 in Figure 8a is also a byproduct of NewsStand's current focus on US-based news sources. Figure 8b shows that the NewsStand database ingests on the order of 50,000–60,000 articles on weekdays, and about 30,000 articles on weekends. Also, Figure 8b demonstrates dips in article counts followed by peaks near 26 Feb and 25 Apr, which are due to system downtime at those times, and subsequent catching up of the system. We also measured the number of unique feeds that supply these articles over time, and these feed counts are shown in Figure 8c. Over all measured days, NewsStand processed and displayed articles of around 2,000–3,000 distinct news sources, indicating the breadth and variety of news sources and data available in NewsStand.

6.2 Data Content

While a primary goal of NewsStand is to capture as much news as possible to its users, the actual content of such news is important as well. Here, we present measurements of the extracted content found by NewsStand in the articles that it retrieves from news sources. For this experiment, we measure content in terms of the number and types of locations found by NewsStand's geotagger, as well as the amount of multimedia found by NewsStand's media extractor. These measurements were made over a month's time, and are shown in Figures 9a and 9b. The number of locations and media follow the characteristic week-versus-weekend pattern seen earlier, with around 130,000 extracted locations and 11,000 extracted media during the week. With about 40,000 articles downloaded daily, this corresponds to 3–4 locations per article on average, which indicates the usefulness of NewsStand's map-based interface, since it will be filled with a large number locations and thus there is much data to query. Further, examining the breakdown of location types, smaller places including states and small cities (under 100,000 population) dominate the location type counts, followed by larger places including large cities (over 100,000 population) and countries. These type counts show NewsStand's focus on highly local streaming news which is more difficult to geotag automatically, but results in a much richer, local news experience. On the other hand, the relatively low number of images and videos, about 1 image per 3–4 articles, shows the difficulty of extracting relevant images and image captions while also filtering for advertising and other spurious media.



(a) (b)

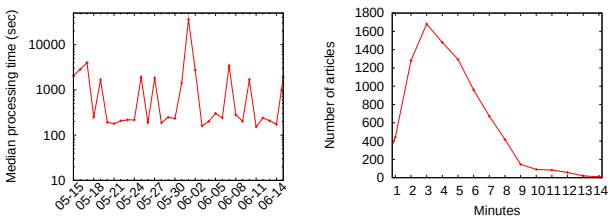
Figure 9: Different types of content found by NewsStand in articles, including (a) number and types of locations, and (b) amount of multimedia (images and videos).

Table 1: Database table sizes.

	Rows	Data	D + I
cluster_locs	4.0M	8GB	15GB
clusters	2.8M	1.9GB	4.5GB
doc_locs	25.8M	5GB	12GB
docs	6.3M	20GB	81GB
doc_text	5.5M	83GB	87GB
entities	207M	25GB	38GB
feeds	10k	200MB	350MB
gazetteer	8.0M	1.5GB	4.2GB
media	1.4M	4GB	9GB

6.3 Data Size

Next, we examined the sizes of tables and indexes present in NewsStand's database. The sizes of the tables shown in Figure 2 are listed in Table 1. For each table, we include the number of rows, the total disk space consumed by the data in the rows, and the total disk space of data plus the indexes present in the table ("D + I"). We can see that the `doc_text` table is by far the largest at 83GB, which is not surprising given that it contains the full text of articles in the database. However, the `docs` table rivals `doc_text` in size when accounting for the index space as well, since the `docs` table has many more columns and indexes on these columns, including a full-text index on the `snippet` attribute for keyword searches. Also, the `doc_locs` and `entities` tables have many rows, reflecting the many locations and other entities found by NewsStand's geotagger and other processing modules.



(a) (b)

Figure 10: Backend processing time in terms of (a) median processing time per document and (b) time for articles to appear in NewsStand's Web interface.

Table 2: Amount of time required by each module to process single documents, measured in seconds.

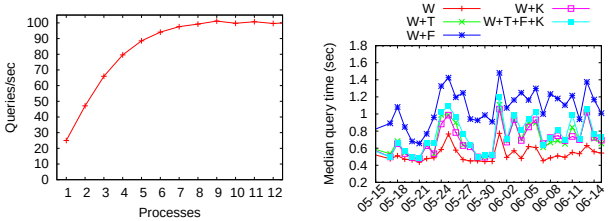
Downloader	1.024	Geotagger	2.961
Cleaner	0.098	People finder	0.535
Clusterer	1.648	Disease finder	0.125
Topic classifier	0.047	Media extractor	0.166
	Total	6.604	

6.4 Processing Time

Next, we examined the time required for a new article ingested into the NewsStand database to be completely processed by all its modules and to become available via the Web interface. Figure 10a presents these processing times as measured over one months' worth of news. On most days, articles were fully processed within a few minutes, demonstrating that NewsStand's database architecture is well adapted for processing streaming news. However, several spikes in the processing time appear as well, with a large spike on 1 Jun. These spikes are likely due to downtime and maintenance for the machines on which the NewsStand database modules execute, which due to being a research system are somewhat unavoidable.

We also directly measured the amount of time it took for articles to appear in NewsStand's Web interface. Over a day's time, we executed a query through the Web interface every 5 minutes to retrieve the 30 most recent articles, and measured how long it took for those articles to be processed. We took these measurements during a day where NewsStand was processing documents under typical daily loads—i.e., not during processing spikes due to downtime or other factors. Figure 10b shows the results, with most retrieved articles taking between 3–5 minutes to appear on the Web interface. This result corroborates our previous finding in indicating NewsStand's suitability for processing streaming news quickly.

We further broke down article processing times in terms of how much time is spent in each of NewsStand's processing modules. These times are listed in Table 2. Processing times per document are rather low, with the total processing time for a single document on average being just over 6 seconds. As work is batched in groups of at most 100, a given work batch would travel through the system and be fully processed by NewsStand's modules in at most 10 minutes, at full load. Of course, if the system is not saturated with work, then articles can be fully processed in a shorter time, and as demonstrated by Figure 10b, we typically observe articles in NewsStand that are only a few minutes old. Examining individual processing times in detail, the geotagger, clusterer, and downloader modules are bottlenecks in processing time, accounting for fully 85% of the total processing time for a document. Reasons for the slowness of these modules vary, but can generally be accounted for by the number of database queries that they make. The geotagger and clusterer modules execute a large number of database queries to draw in additional evidence for use in geotagging and clustering. On the other hand, the downloader module's primary bottleneck is the time required to download articles from websites. Note that despite their slow processing times, we avoid slowing the system as a whole by starting more instances of these modules, thus increasing the system's processing throughput. We also start more instances when a large amount of work is waiting in pipes, due to modules having been stopped for some time.



(a) (b)

Figure 11: Query performance as measured by (a) the sustained number of queries executed by NewsStand's database, and (b) median query times for top-k window queries with various additional constraints.

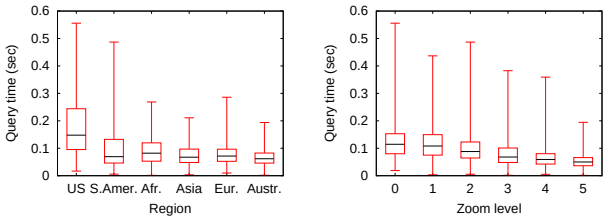
6.5 Query Performance

Our final set of experiments were designed to test the interactivity of the NewsStand database Web interface, as measured by the time required to execute queries generated by the interface, especially in NewsStand's map mode. As outlined in Section 5.2, these queries are all variants of top-k window queries. Recall that Figure 6 shows examples of this type of query, which all contain a query window qw expressed in lat/long values, and return the k clusters with highest scores that fall within qw. We generated queries by randomly creating query windows over land masses with a variety of sizes. Each window consists of lower left lat/long values, width, and height. We executed these queries in NewsStand's cache database (described in Section 4.4), and measured query performance over time. In this way, we tracked live performance fluctuations that arise from the NewsStand database operating on streaming news.

Our first set of query performance experiments tested the number of top-k window queries per second that could be sustained by NewsStand's cache database, which should be roughly proportional to the number of users that could be browsing in NewsStand's interface simultaneously. For each experiment, we created several test processes, each of which would connect to the database and execute queries as quickly as possible. By increasing the number of processes and hence database connections, we saturated the database's query capacity to determine an upper limit on the number of queries per second that can be handled simultaneously. For this experiment, k was fixed at 200, matching the value used in NewsStand's user interface. Though this value may seem small, especially compared to the database's size, it is actually reasonable given limited bandwidth and screen space.

Figure 11a presents the queries per second (qps) rate delivered by NewsStand's cache database, as measured over one minute of processing. As the number of test processes increased, the qps rate likewise increased, to a ceiling of about 100, as shown by the flattening of the qps curve at around 8–9 test processes. If we consider that a user of NewsStand could generate at most two queries per second, via continuous actions such as scrolling, panning, and zooming, the system as-is can support up to 50 users at a time. These performance results are respectable, especially given that they were executed on a live system, constantly being updated with additional streaming news. To gain additional performance when scaling the system up to support larger numbers of users, further options such as database replication and round-robin scheduling could be considered.

Our next query time experiment combined the top-k window queries described above with additional constraints that are added



(a) Region (b) Zoom

Figure 12: Region and zoom level versus query time.

by certain queries available in NewsStand's interface. We tested the following additional constraints:

1. **Topic**: Retrieve stories relevant to a given topic (e.g., "Business", "Sports").
2. **Feeds**: Retrieve stories with articles from a set of news feeds.
3. **Keyword**: Retrieve stories relevant to a keyword.

To select topics, feeds, and keywords to use in these queries, we sampled query values present in NewsStand's query log files, and selected among them randomly. Furthermore, in keeping with our goal of benchmarking streaming news, we executed a set of these queries every five minutes over a one month period, to track performance of the live NewsStand system over time. Figure 11b contains performance results for these queries, shown as the median query time per day for each query type, with constraints specified as letters: "W" for window, "T" for topic, "F" for feeds, and "K" for keywords, and combinations such as "W+F" referring to a window query with a feed constraint. Examining the results, we see that query times fluctuate from day to day, but tend to fall under 1 sec. The plain top-k window query was faster than the other query types, with additional constraints generally slowing the query time. Further, the feeds query was consistently slower than the other query types, most likely due to the requirement that query results be from a set of feeds, rather than a single value. Nonetheless, query performance was relatively consistent across all dates and times tested, reflecting NewsStand's stability as a live system.

For our final query time experiments, we examined query times for queries placed in a particular region or zoom level. Figure 12a presents query times for random window sizes and positions categorized by the location of the query region, as a box and whisker plot. For all regions, query times were again respectable, with all medians being under 0.2 sec. Furthermore, notice that the query times tended to have skewed distributions, with most query times being low, but with high outliers. We also observed that query performance for query windows in the US region is significantly higher than in other areas. This is likely due to NewsStand's source bias for newspapers in the US rather than other areas. Figure 12b explores query performace differences by varying the zoom level where again both query window size and positioning were randomized and no distinction was made for different regions of the world. As might be expected, as the zoom level increases (thus decreasing the query window size and hence the number of markers present in the window), query times decreased. As before, query times are generally low, with some larger outliers.

7. CONCLUSION

NewsStand's database architecture makes it suitable for processing documents using *cloud computing*, an area for future research, since multiple instances of slave modules can execute simultaneously. Rather than having a single cache database, several such databases on separate computers could allow greater scalability and a larger eventual user base. With a larger user base, additional analytics can be used to improve querying performance. For example, the geographic location of users (as determined by IP addresses), or their most frequent window queries, can be used to inform caching strategies. As the prevalence of streaming data on the Web increases, database systems such as NewsStand that are capable of quickly processing this streaming data will be ever more important.

8. REFERENCES

[1] E. Amitay, N. Har'El, R. Sivan, and A. Soffer. Web-a-Where: Geotagging web content. In *SIGIR'04*, pages 273–280, Sheffield, UK, July 2004.

[2] W. G. Aref and H. Samet. Efficient processing of window queries in the pyramid data structure. In *PODS'90*, pages 265–272, Nashville, TN, Apr. 1990.

[3] E. Banos, I. Katakis, N. Bassiliades, G. Tsoumakas, and I. Vlahavas. PersoNews: A personalized news reader enhanced by machine learning and semantic filtering. In *ODBASE'06*, pages 975–982, Montpellier, France, Oct. 2006.

[4] E. Gabrilovich, S. Dumais, and E. Horvitz. Newsjunkie: Providing personalized newsfeeds via analysis of information novelty. In *WWW'04*, pages 482–490, New York, May 2004.

[5] E. Garbin and I. Mani. Disambiguating toponyms in news. In *HLT/EMNLP'05*, pages 363–370, Vancouver, Canada, Oct. 2005.

[6] M. Henzinger, B.-W. Chang, B. Milch, and S. Brin. Query-free news search. *World Wide Web*, 8(2):101–126, June 2005.

[7] F. T. Johnsen, T. Hafsøe, C. Griwodz, and P. Halvorsen. Workload characterization for news-on-demand streaming services. In *IPCCC'07*, pages 314–323, New Orleans, LA, Apr. 2007.

[8] M. Krstajic, F. Mansmann, A. Stoffel, M. Atkinson, and D. A. Keim. Processing online news streams for large-scale semantic analysis. In *ICDEW'10*, pages 215–220, Long Beach, CA, Mar. 2010.

[9] J. L. Leidner. An evaluation dataset for the toponym resolution task. *CEUS: Computers, Environment, and Urban Systems*, 30(4):400–417, July 2006.

[10] M. D. Lieberman and H. Samet. Multifaceted toponym recognition for streaming news. In *SIGIR'11*, pages 843–852, Beijing, China, July 2011.

[11] M. D. Lieberman and H. Samet. Adaptive context features for toponym resolution in streaming news. In *SIGIR'12*, pages 731–740, Portland, OR, Aug. 2012.

[12] M. D. Lieberman, H. Samet, and J. Sankaranarayanan. Geotagging: Using proximity, sibling, and prominence clues to understand comma groups. In *GIR'10*, Zurich, Switzerland, Feb. 2010.

[13] M. D. Lieberman, H. Samet, and J. Sankaranarayanan. Geotagging with local lexicons to build indexes for textually-specified spatial data. In *ICDE'10*, pages 201–212, Long Beach, CA, Mar. 2010.

[14] M. D. Lieberman, H. Samet, J. Sankaranarayanan, and J. Sperling. STEWARD: Architecture of a spatio-textual search engine. In *GIS'07*, pages 186–193, Seattle, WA, Nov. 2007.

[15] B. Martins, I. Anastácio, and P. Calado. A machine learning approach for resolving place references in text. In *AGILE'10*, pages 221–236, Guimarães, Portugal, May 2010.

[16] G. Quercini, H. Samet, J. Sankaranarayanan, and M. D. Lieberman. Determining the spatial reader scopes of news sources using local lexicons. In *GIS'10*, pages 43–52, San Jose, CA, Nov. 2010.

[17] H. Samet, M. D. Adelfio, B. C. Fruin, M. D. Lieberman, and B. E. Teitler. Porting a web-based mapping application to a smartphone app. In *GIS'11*, pages 525–528, Chicago, Nov. 2011.

[18] H. Samet, H. Alborzi, F. Brabec, C. Esperança, G. R. Hjaltason, F. Morgan, and E. Tanin. Use of the SAND spatial browser for digital government applications. *CACM*, 46(1):63–66, Jan. 2003.

[19] H. Samet, A. Rosenfeld, C. A. Shaffer, and R. E. Webber. A geographic information system using quadtrees. *Pattern Recognition*, 17(6):647–656, November/December 1984.

[20] H. Samet, B. E. Teitler, M. D. Adelfio, and M. D. Lieberman. Adapting a map query interface for a gesturing touch screen interface. In *WWW'11*, pages 257–260, Hyderabad, India, Mar. 2011.

[21] J. Sankaranarayanan and H. Samet. Images in news. In *ICPR'10*, pages 3240–3243, Istanbul, Turkey, Aug. 2010.

[22] C. A. Shaffer, H. Samet, and R. C. Nelson. QUILT: a geographic information system based on quadtrees. *IJGIS*, 4(2):103–131, April–June 1990. Also University of Maryland Computer Science Technical Report TR–1885.1, July 1987.

[23] B. E. Teitler, M. D. Lieberman, D. Panozzo, J. Sankaranarayanan, H. Samet, and J. Sperling. NewsStand: A new view on news. In *GIS'08*, pages 144–153, Irvine, CA, Nov. 2008.

[24] B. E. Teitler, J. Sankaranarayanan, and H. Samet. Online document clustering using the GPU. Technical Report CS-TR-4970, University of Maryland, College Park, MD, Aug. 2010.