Resource-Adaptive Real-Time New Event Detection

Gang Luo Chunqiang Tang Philip S. Yu
IBM T.J. Watson Research Center
{luog, ctang, psyu}@us.ibm.com

ABSTRACT

In a document streaming environment, online detection of the first documents that mention previously unseen events is an open challenge. For this online new event detection (ONED) task, existing studies usually assume that enough resources are always available and focus entirely on detection accuracy without considering efficiency. Moreover, none of the existing work addresses the issue of providing an effective and friendly user interface. As a result, there is a significant gap between the existing systems and a system that can be used in practice. In this paper, we propose an ONED framework with the following prominent features. First, a combination of indexing and compression methods is used to improve the document processing rate by orders of magnitude without sacrificing much detection accuracy. Second, when resources are tight, a resource-adaptive computation method is used to maximize the benefit that can be gained from the limited resources. Third, when the new event arrival rate is beyond the processing capability of the consumer of the ONED system, new events are further filtered and prioritized before they are presented to the consumer. Fourth, implicit citation relationships are created among all the documents and used to compute the importance of document sources. This importance information can guide the selection of document sources. We implemented a prototype of our framework on top of IBM's Stream Processing Core middleware. We also evaluated the effectiveness of our techniques on the standard TDT5 benchmark. To the best of our knowledge, this is the first implementation of a real application in a large-scale stream processing system.

Categories and Subject Descriptors

H.3.3 [Information Search and Retrieval]: information filtering

General Terms: Algorithms, Experimentation

Keywords: online new event detection, document streaming

1. INTRODUCTION

In a document streaming environment, documents come from one or more sources. *New event detection* (NED) is the task of capturing the first documents that mention previously unseen events. This task has practical applications in several domains, where useful information is buried in a large amount of data that grows rapidly with time. Such domains include intelligence gathering, financial market analyses, and news analyses. Applications in those domains are often time-critical and the use of an *online new event detection* (ONED) system is highly desired. For instance, the US government is building a massive computer system that can monitor news, blogs, and emails for anti-terrorism purposes [13, 22], and ONED is an essential component of this system.

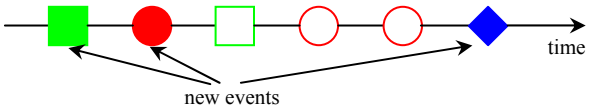

Figure 1. Events in a document stream. Different shapes correspond to different events. Filled shapes represent the documents that need to be captured.

Recently, ONED has attracted much attention [2, 3, 7, 11, 19, 21, 23, 29, 37, 38]. In order to provide a standard benchmark for comparing different algorithms, National Institute of Standards and Technology (NIST) has organized a Topic Detection and Tracking (TDT) program [34], where ONED is one of the main tasks. Despite all the efforts, there is still a significant gap between the state-of-the-art ONED systems and a system that can be used in practice.

Most of the existing ONED systems compare a new document D to all the old documents that arrived in the past. If the similarity values between D and the old documents are all below a certain threshold, D is predicted to mention a new event. This method has quadratic time complexity with respect to the number of documents and is rather inefficient. For example, in the latest TDT5 competition [34], many systems spent several days on processing just 280,000 news articles, whose total size is less than 600MB. This processing speed is orders of magnitude slower than a typical document arrival rate.

In practice, an ONED system can monitor a large number of document sources. For example, Google news has 4,500 sources [16] and Yahoo! news [36] has more than 5,000 sources. In the intelligence gathering system that is being developed by the US government [13, 22], document sources cover an even wider spectrum, including emails, instant messages, web bulletin boards, and blogs. Therefore, a practical ONED system needs to handle a high document arrival rate without resorting to an excessive amount of hardware resources. Moreover, due to the bursty nature of document streams, an ONED system should be able to operate gracefully even if it runs out of resources. These performance issues, however, have not been addressed in previous studies.

Figure 2 shows the architecture of a traditional ONED system, where the output documents are put into a queue, waiting to be consumed. The consumer can be either a person or a computer program that does further deep analysis (e.g., machine translation of foreign documents). The processing speed of the consumer can be much slower than the peak output rate of the ONED system. For

SIGMOD'07, June 12–14, 2007, Beijing, China.

example, the state-of-the-art machine translation speed is measured by the number of words per second [24].

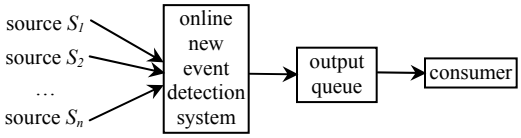

Figure 2. A traditional online new event detection system.

None of the existing ONED systems has considered the following user interface issues: (1) When the consumer is overloaded and cannot keep pace with the output rate of the ONED system, less important documents need to be dropped from the queue (or moved to a low-priority queue) so that the consumer can focus on important documents. (2) Depending on the concrete requirement of the consumer, documents can be sorted in the queue according to different criteria (e.g., importance or arrival time) so that desired documents are processed by the consumer first.

In this paper, we propose a comprehensive framework for ONED that covers a large design space. Within this framework, we propose a system that improves upon existing systems from four perspectives to address the above-mentioned problems. First, various indexing and compression methods are used to increase the document processing rate by orders of magnitude without sacrificing much detection accuracy. Second, when resources are tight, our system focuses on the important documents and attempts to maximize the benefit that can be gained from the limited resources. Third, when the new event arrival rate is beyond the processing capability of the consumer of the ONED system, our system avoids overwhelming the user by further filtering and prioritizing new events before presenting them to the consumer. Fourth, the importance of document sources is computed, which can be used to guide the selection of document sources.

The main challenge in improving efficiency and effectively using the limited resources is to minimize the amount of saved information without losing much information that is critical for the detection accuracy. Regarding to providing a friendly user interface, the main challenge is to decide the relative importance of different documents. For this purpose, we use the intermediate computation results of ONED to determine which documents' contents are repeated by the other documents that arrive later, and automatically create *implicit* citation relationships among all the documents. Those documents with a large number of citations are considered important. At the same time, citations among documents are merged together to obtain linking relationships among document sources, which are used to compute the importance of document sources.

We implemented a prototype of our framework on top of IBM's Stream Processing Core (SPC). As described in Jain et al. [17] in detail, SPC is a stream processing middleware that provides an application execution environment for processing elements (or applications) developed by users to filter and analyze data streams. To the best of our knowledge, this is the first implementation of a real application in a large-scale stream processing system. Our evaluation on the TDT5 benchmark shows that our techniques can (1) improve the document processing rate by two orders of magnitude (reducing the processing time for the entire document set from more than three days to less than 12 minutes), (2) maximize the benefit that can be gained from the limited resources, (3) provide an effective user interface, and (4) produce an importance ranking of document sources that matches with our real world experience.

The rest of the paper is organized as follows. Section 2 introduces a baseline system. Section 3 presents our general framework for ONED. Section 4 describes the techniques for improving efficiency. Section 5 shows the resource-adaptive computation methods. Section 6 discusses the user interface issues. Section 7 describes the document source ranking method. Section 8 investigates the performance of our techniques. We discuss related work in Section 9 and conclude in Section 10.

2. A BASELINE SYSTEM

To set the stage for the discussion of our techniques, we first describe a baseline ONED system in this section. This baseline system is similar to the ONED system reported in Braun and Kaneshiro [8], which achieved the best detection accuracy in the latest TDT5 competition. Our improvements presented in the next several sections are based on this baseline system.

Following the convention of information retrieval literature [30], we define *vocabulary* as the set of all the distinct words. A *term* is a word. A *first-story document* is a document that describes a previously unseen event.

The baseline system uses a variant of the state-of-the-art Okapi formula [28, 30] to compute both term weights and the similarity values of document pairs. We first give a brief summary of Okapi. In Okapi, both documents and queries are represented as vectors. Each element of a vector is the weight of a term in the vocabulary. Terms that are important to a document are assigned large weights. Terms that do not appear in the document have weights zero. The relevance between a document D and a query Q is computed as the inner product of D's vector and Q's vector. The intuition behind Okapi is that the more times a term t appears in a document D and the fewer times t appears in other documents (i.e., the less popular t is in other documents), the more important t is for D. Also, the effect that longer documents have more words needs to be compensated by normalizing for document lengths.

Consider a document set S. For each term t in the vocabulary and a document $D \in S$, Okapi uses the following formulas:

(f1) term frequency (tf) weight

$$w_{tf} = \frac{(k_1+1)tf}{k_1[(1-b)+b\times dl/avdl]+tf},$$

(f2) inverse document frequency (idf) weight

$$w_{idf} = \ln\frac{N-df+0.5}{df+0.5}.$$

Here tf is t's frequency (i.e., number of occurrences) in D, N is the total number of documents in S, df is the number of documents in S that contain t, dl is the length of D in bytes, and $avdl$ is the average length (in bytes) of all the documents in S. b and k_1 are two predetermined constants. Typically, as suggested in Singhal [30], $b=0.75$ and $k_1 = 1.2$.

Consider a query Q. For each document $D \in S$, Okapi defines its score (i.e., the degree of relevance for answering Q) as the sum of term weights of all the terms that appear in both D and Q. Each term weight is computed using the tf weight for D, the tf weight for Q, and the idf weight. In the case of ONED, we need to compute the similarity value between two documents $D_1 \in S$ and $D_2 \in S$. Hence, we change Okapi slightly to fit our purpose: the similarity value between D_1 and D_2 is computed as the inner product of D_1's vector and D_2's vector. More specifically, for either document D_i (i=1, 2),

a tf weight $w_{tf,i}$ is computed. The term weight is defined according to (f3). The similarity value is computed according to (f4), where the sum is over all the terms that appear in both D_1 and D_2.

(f3) term weight $w_t = w_{tf,1} \times w_{tf,2} \times w_{idf}$,

(f4) $similarity_{D_1, D_2} = \sum_{t \in D_1, D_2} w_t$.

In the above computation, the similarity value is not normalized to [0, 1], as Okapi has already made normalization for document lengths.

Next, we present the details of the baseline system. As standard pre-processing operations in information retrieval, for each document, (1) stemming is performed using the standard Porter stemmer [26], and (2) stopwords are removed by using the standard SMART stopword list [32]. In a document streaming environment, the document set S keeps changing as new documents continue to arrive. As mentioned in Braun and Kaneshiro [8], the computation of the tf and idf weights can be based on a static document set S' (such as the TDT4 document set) that has characteristics similar to S. For a term that does not exist in S', its df is treated as one. Compared to the method that incrementally updates the statistics N, $avdl$, and df, this static method has lower overhead while the detection accuracy remains roughly the same [8].

When a new document D arrives, D is first pre-processed and its information is saved in memory. Then D is compared to all the old documents that arrived in the past. If all the similarity values between D and the old documents are below a threshold T, D is predicted to mention a new event. In this case, D is put into the output queue, waiting to be consumed. Otherwise if the similarity value between D and an old document D_{old} is above T, D is predicted to mention the same event as D_{old} and thus not considered as a first-story document.

3. A GENERAL FRAMEWORK

In this section, we propose a comprehensive framework for ONED, as shown in Figure 3. This framework defines a fairly large design space and is much more general than the traditional ONED system shown in Figure 2. It contains several components. An ONED system can be obtained by instantiating each component with a concrete policy. All the techniques described in the following sections are policy examples.

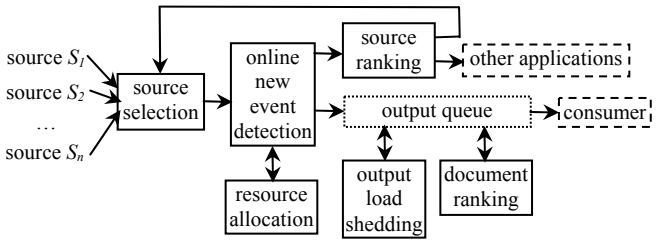

Figure 3. A general framework for online new event detection.

Next, we describe the components in this framework. The source selection component determines the document sources from which documents are received. Documents from these selected sources are fed to the ONED component, where first-story documents are identified. The identified first-story documents are sent to an output queue Q_o, waiting to be processed by the consumer of the ONED system.

When resources are tight, the resource allocation component determines how to maximize the benefit that can be gained from the limited resources. When the consumer is overloaded and cannot keep pace with the output rate of the ONED system, the output load shedding component determines which documents in Q_o should be dropped or moved to a low-priority queue (waiting there until the consumer becomes free). The document ranking component determines the order in which documents in Q_o are presented to the consumer.

The source ranking component takes the information generated by the ONED component as input to compute the relative importance of document sources. This importance information is sent back to the source selection component to guide the selection of document sources. Other applications can also use this importance information for their own purposes, e.g., online advertisement. In the following sections, we elaborate on the individual components in this framework.

4. TECHNIQUES FOR IMPROVING EFFICIENCY

In this section, we describe our techniques for improving the efficiency of the ONED component. The baseline system described in Section 2 has two shortcomings regarding to efficiency. First, as new documents continue to arrive, the number of previously arrived documents keeps increasing, and eventually the memory will not be able to hold the information for all the old documents. However, due to the real-time nature of ONED, generally all the data structures that are used should be kept in memory to avoid expensive I/Os. Second, it is expensive to compare a new document with all the old ones. To reduce both storage and computation overhead, we limit both the number of saved documents and the number of terms kept for each saved document without sacrificing much detection accuracy. Here *saved documents* refer to the ones whose information is saved in memory.

4.1 Reducing the Number of Saved Documents

Typically, the discussion of an event lasts for a finite amount of time in news articles, and a new document is unlikely to mention the same event as a document that is fairly old. Hence, documents that are too old are not very useful and we only keep in memory the information of those old documents that are within a sliding window of the last W days. Here W is a predetermined constant. Once an old document expires from this sliding window, its information is thrown away immediately. Our experiments in Section 8.1 show that a good value for W is usually between 24 and 32 days.

Typically, an event is mentioned by a large number of documents. Only one of these documents is the first-story document. For example, in the TDT5 document set, for the 250 specified events, on average each event is mentioned by 40 documents [34]. All the documents that mention the same event tend to be similar to each other. Therefore, it is an overkill to compare a new document with all the old documents that mention the same event. Instead, we only keep the information of the first-story documents. When a new document D arrives, D is compared with the old first-story documents. If D is predicted to be a first-story document that mentions a new event, D's information is saved in memory. Otherwise D is discarded.

4.2 Reducing the Number of Saved Terms

All the terms in a document D can be sorted in descending order of their tf×idf values. In general, those terms with large tf×idf values are important to D. As has been observed in Allan et al. [3], in computing the similarity value of two documents, we only need to use those important terms of the two documents, as those terms contribute to most of the similarity value. A similar effect has been observed in a general information retrieval environment [27, 31, 33]. Hence, for each saved document, we only keep the top-K terms with the largest tf×idf values rather than all the terms. Here K is a predetermined constant. Only the top-K terms are used to compute the similarity values of document pairs. Our experiments in Section 8.1 show that K can be as small as 100 without severely degrading the detection accuracy.

4.3 Pre-Filtering

To reduce the overhead of computing similarity values, a pre-filtering technique is used. Our idea is to use a low-overhead method to quickly filter out most of the documents that mention different events from the new document. In this way, we can substantially reduce the number of similarity values that need to be computed. Consider two documents D_1 and D_2. If D_1 and D_2 mention the same event E, their top terms tend to have some overlap. That is, some term(s) describing E is likely to appear in the top terms of both D_1 and D_2. Thus, top terms can be used to quickly filter out unnecessary computations. More specifically, we have a predetermined constant M ($M \leq K$). Before computing the similarity value of D_1 and D_2, we first check whether the top-M terms of D_1 and D_2 intersect. If so, we continue to compute the similarity value of D_1 and D_2. Otherwise, we predict that D_1 and D_2 mention different events and do not compute their similarity value. Our experiments in Section 8.1 show that $M=10$ is usually sufficient to achieve a good pre-filtering ratio without losing much detection accuracy.

4.4 Building Indices

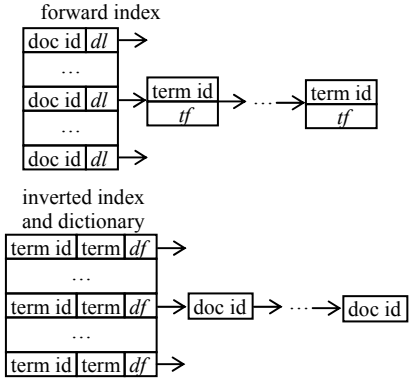

Figure 4. Index data structures.

We build indices to avoid unnecessary processing of the documents that have been pre-filtered out. Each term in the vocabulary has a term id. Each document has a doc id corresponding to its arrival time. As shown in Figure 4, two indices are kept for all the saved documents: a forward index and an inverted index. The forward index has an entry for each saved document. These entries are sorted in descending order of documents' arrival time. This allows us to quickly identify and drop the information of those documents that have expired from the sliding window of the last W days (see Section 4.1). For each saved document, the corresponding entry keeps the document length dl and the top-K terms associated with their term frequencies tf (see Section 4.2). These terms are sorted in ascending order of their term ids. Consequently, the similarity value of two documents can be computed through an efficient "merge" of their term lists.

For each saved document, only its top-M terms are tracked by the inverted index. The inverted index has an entry for each term in the vocabulary. The entry for term t is a posting (linked) list of the doc ids of all the documents whose top-M terms contain t. These doc ids are sorted in descending order so that merging posting lists can be done efficiently. Since typically $M<<K$, the document-term information in the inverted index is only a subset of that in the forward index. When a new document D arrives, we only scan the M posting lists that correspond to D's top-M terms. These M posting lists are merged together to find the doc ids of the candidate documents that may mention the same event as D. This is the pre-filtering technique described in Section 4.3. Then for each such candidate document D_c, the forward index is used to compute the similarity value of D and D_c. The similarity value computation is performed at the same time that candidate doc ids are generated. In this way, if the similarity value of D and an old document is greater than the threshold T, D is predicted to be a non-first-story document and the processing for D stops immediately. Otherwise if D is predicted to be a first-story document, D's information can be easily added into the inverted index, as D's doc id is larger than the doc ids of the saved documents.

4.5 Parallel Processing

The above discussion assumes the use of a single computer. Our framework can be naturally extended to use a cluster (say, C) of computers to process incoming documents at a higher rate. The concrete method is as follows. All the saved documents are partitioned into C sets (e.g., using round-robin partitioning [15]). When a new document D arrives, it is parsed on one computer to obtain its term frequency list. Then this information is sent to all the computers to compare D with the saved documents. If any computer predicts that D is not a first-story document, the whole ONED system considers D as a non-first-story document and throws D away. Otherwise D is considered as a first-story document and its information is saved on a computer according to the document partitioning schema.

5. RESOURCE-ADAPTIVE COMPUTATION

In this section, we describe the resource-adaptive computation methods. If the arrival rate of new documents is high (e.g., due to the bursty nature of document streams), the ONED system may become overloaded in two ways: (1) the CPU cannot process all the incoming new documents, or (2) the memory cannot hold the information for all the identified first-story documents within the last W days. In the first case, we need to adjust the parameters (e.g., W, T, and M) of our techniques to increase the throughput of the ONED system. In the second case, the information of some saved documents must be removed from memory. In both cases, the goal of the resource utilization component in Figure 3 is to minimize the loss in detection accuracy.

5.1 CPU-Bound Case

We first consider the case that the ONED system is not fast enough to handle all the incoming new documents. As will be shown in Section 8.1, decreasing the parameter values (W, T, and M) of our techniques can increase the throughput of the ONED system. For example, the smaller the W, the fewer old documents are included for comparison with the incoming new documents. The smaller the M, the fewer incoming new documents can pass the pre-filter and get compared with the old documents. Hence, we can intelligently shed the input load by adaptively varying the amount of processing applied to incoming new documents based on the load.

Our ONED system has a FIFO queue Q_i (the subscript i stands for input) that can hold at most V documents. When a new document D arrives, if the ONED system is busy, D is first put into Q_i, waiting to be processed by the ONED system. When Q_i becomes close to full, the ONED is overloaded. In this case, we have multiple methods for improving the system throughput, all at the expense of sacrificing the detection accuracy of the ONED system:

(1) **Method 1 (dropping new documents)**: The incoming new document is simply dropped.
(2) **Method 2 (adjusting *W*)**: Let W_d represent the default initial value of the sliding window size W. Whenever Q_i becomes close to full, W is decreased by 5%. When the number of documents in Q_i drops below $V/2$, W is increased by 5% if $W<W_d$.
(3) **Method 3 (adjusting *T*)**: The same as method 2 except that we adjust T, the threshold for the similarity value.
(4) **Method 4 (adjusting *M*)**: The same as method 2 except that we adjust M, the number of top terms used for pre-filtering.
(5) **Method 5 (adjusting *W*, *T*, and *M*)**: Whenever Q_i becomes close to full, one of W, T, and M is decreased by 5% in a round-robin fashion, e.g., first W, then T, then M, and so on. When the number of documents in Q_i drops below $V/2$, one of W, T, and M is increased by 5% in a round-robin fashion if the value of this parameter is smaller than its default initial value.

We compare the performance of these methods in Section 8.2.

5.2 Memory-Bound Case

Next, we consider the case that the memory is used up and the information of some saved documents must be removed from memory. We introduce a definition that will be frequently used in the rest of the paper:

Implicit citation (or simply citation): When a non-first-story document D_{nf} arrives, if D_{nf} mentions the same event as an existing first-story document D, we say that D is *cited* by D_{nf} once.

Intuitively, to minimize the loss in detection accuracy, we need to keep in memory the information of those documents that will be cited by a large number of documents in the future. If we treat memory as a cache and citations as cache hits, this becomes a cache management problem. Hence, we can use a traditional cache management algorithm such as LRU to manage all the saved documents in memory. We will compare the LRU policy with the following policies in Section 8.2:

Random policy: Randomly remove the information of saved documents from memory.

Time policy: Always remove the information of the oldest documents from memory.

6. USER INTERFACE ISSUES

In this section, we discuss the user interface issues.

6.1 Output Load Shedding

In practice, the processing rate of the consumer can be slower than the output rate of the ONED system, particularly when a burst of first-story documents arrive. In this case, some documents need to be dropped from the output queue Q_o so that the consumer will not become overloaded. The output load shedding component strives to minimize this impact by dropping less important documents from Q_o.

Intuitively, the importance of a document D is measured by the importance of the event E mentioned by D, and the importance of E is related to the number of documents mentioning E. We use the following method to judge the importance of a first-story document D. The total number of citations that D have received so far and will receive in the future is called the final citation number of D, which is denoted as $C_{final}(D)$ and reflects the importance of D. As a companion concept, the number of citations that D have received so far is called the current citation number of D, which is denoted as $C_{current}(D)$.

The main idea of our output load shedding method is as follows. To avoid overwhelming the consumer, the size of the output queue Q_o is fixed. Documents are removed from Q_o when they are consumed by the consumer. When Q_o becomes full, some document must be dropped from Q_o before a new document can be inserted into Q_o. Intuitively, for the documents in Q_o, their current citation numbers partially reflect their importance. Hence, we keep track of the current citation numbers of the documents in Q_o. One naive policy is to drop from Q_o those documents with small current citation numbers. This policy, however, is unfair. Newly arrived documents tend to have small current citation numbers but they can be important if they will receive a large number of citations in the future. Thus, it is not desirable to always drop newly arrived documents in favor of those documents that arrived a long time ago. To address this problem, Q_o is split into two parts: the new part Q_{o_new} and the old part Q_{o_old}. A newly arrived document D first stays in Q_{o_new} to accumulate citations. When D moves from Q_{o_new} to Q_{o_old}, its current citation number has become close to its final citation number and can roughly reflect its importance. Documents in Q_{o_old} with small current citation numbers are considered as less important and thus the candidates to be dropped from Q_o.

The concrete output load shedding method works as follows. For each document in the output queue Q_o, we use a counter to keep track of its current citation number. When a document D is first inserted into Q_o, D's counter is initialized to zero. As described in Section 2, when a new document D_{new} arrives at the ONED system, D_{new} is compared with the saved documents in memory. If the similarity value between D_{new} and a saved document D_{old} is above the threshold T, D_{new} is predicted to mention the same event as D_{old}. That is, D_{old} is cited by D_{new} once. In this case, if D_{old} still exists in Q_o, D_{old}'s counter is incremented by one.

The resource utilization method described in Section 5.2 is revised slightly. The documents in the output queue Q_o is a subset of the saved documents in memory. When memory overflows, the information about the documents in Q_o is never removed from memory, as this information is needed to keep track of the current citation numbers of the documents in Q_o.

The output queue Q_o can hold at most N documents, where N is a constant specified by the consumer of the ONED system. Q_o contains two parts: the new part Q_{o_new} and the old part Q_{o_old}. Q_{o_new}

is a FIFO queue and can hold at most $p \times N$ documents, where p is a predetermined constant ($0 \le p \le 1$). Q_{o_old} can hold at most $(1-p) \times N$ documents. All the documents in Q_{o_old} are sorted in ascending order of their current citation numbers. The optimal value of p depends on both N and the document set. It can be determined using a training document set that has similar characteristics as the actual document set. Each time a first-story document D is identified, D is inserted into Q_{o_new}. If Q_{o_new} is full, the oldest document in Q_{o_new} is moved to Q_{o_old}. If Q_{o_old} becomes full, the document in Q_{o_old} that has the smallest current citation number is dropped.

Note that it is not desirable to use the LRU algorithm to manage Q_{o_old}, because our optimization criterion is the citation number rather than the cache hit ratio. LRU can incorrectly drop the documents with large citations numbers if their last citations happened a long time ago. Our key observation is that a good policy should consider both document arrival time and current citation number. Our algorithm is one of the policies that consider these two factors. We leave it as a subject for future work to investigate other alternatives.

6.2 Document Ranking

When presenting results to the consumer, the document ranking component can sort the documents in the output queue according to a criterion different from that used in the output load shedding method. This allows the consumer to process the desired documents first.

For this purpose, we keep a pointer queue Q_r (the subscript r stands for rearrangement) that contains N pointers. Each pointer links to a different document in the output queue Q_o. These pointers are sorted according to the policy that is specified by the document ranking component. Documents in Q_o are presented to the consumer in the order that their pointers are sorted in Q_r.

The document ranking policy depends on the concrete requirement of the consumer. One policy is to sort all the pointers in Q_r in ascending order of the corresponding documents' arrival time. Consequently, the consumer always processes the oldest document first.

A second policy is to sort all the pointers in Q_r in descending order of the corresponding documents' importance (i.e., current citation numbers) so that the consumer can see the currently-most-important document first. This policy may introduce starvation, as documents that arrive later and quickly accumulate a large number of citations can always jump ahead of a document that arrived earlier but does not receive citations any more.

One solution to address this problem is to break Q_r into two queues: the new queue Q_{r_new} and the old queue Q_{r_old}, as shown in Figure 5. All the pointers in Q_{r_new} are sorted in descending order of the current citation numbers of the corresponding documents. All the pointers in Q_{r_old} are sorted in ascending order of the arrival time of the corresponding documents. When a document D is first inserted into the output queue Q_o, the pointer to D is in Q_{r_new}. After D has stayed in Q_o for a certain amount of time T_c, where T_c is a constant specified by the consumer, the pointer to D is moved to Q_{r_old}. Both the currently-most-important document (with the largest current citation number) whose pointer is in Q_{r_new} and the oldest document whose pointer is in Q_{r_old} are presented to the consumer simultaneously. The consumer determines which of these two documents to process first. This gives the oldest documents in Q_o a chance of being seen by the consumer rather than getting starved.

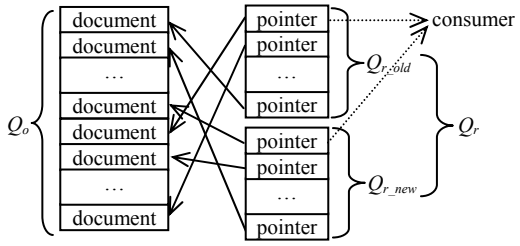

Figure 5. One arrangement of the output queue Q_o.

7. RANKING DOCUMENT SOURCES

For many applications, it is desirable to know the importance of document sources [12]. For example, due to its limited processing power, a system may only want to process documents from those importance sources rather than all the available sources. In this section, we describe a document source ranking algorithm. The source ranking component uses this algorithm and the information generated by the ONED component to compute the importance of document sources.

Intuitively, a document source is important if it is often the first source to report important events. An important event is mentioned by a large number of documents. Hence, a document source is important if it emits a large number of first-story documents, and many of these first-story documents are frequently cited by the other documents. Our key observation is that the citations among documents create implicit "links" among document sources. In other words, the citations among documents can be merged together to obtain linking relationships among document sources. Then a PageRank-style algorithm [25] can be used to compute the importance of document sources. Note that PageRank and other similar algorithms [20, 25] use explicit links among web pages to compute the importance of web pages, whereas our algorithm uses automatically created, implicit links to compute document source importance.

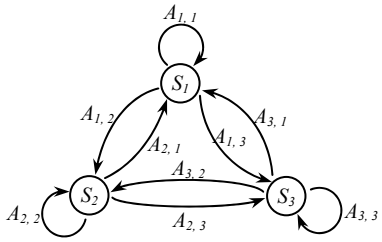

Figure 6. Three document sources citing each other.

The concrete document source ranking algorithm works as follows. Suppose there are n document sources: S_1, S_2, …, and S_n. We keep a matrix $A_{n \times n}$. Initially, $\forall i, j$ ($1 \le i \le n$, $1 \le j \le n$): $A_{i,j} = 0$. Each time the ONED system discovers that a document from source S_i ($1 \le i \le n$) cites a document from source S_j ($1 \le j \le n$), $A_{i,j}$ is incremented by one. That is, $A_{i,j}$ is the number of times that S_i cites S_j, as shown in Figure 6. Matrix $B_{n \times n}$ is a normalized version of $A_{n \times n}$ in the sense that each row of B sums to one. That is,

$$B_{i,j} = A_{i,j} \Big/ \sum_{k=1}^{n} A_{i,k} .$$

$B_{i,j}$ represents the fraction of S_i's citations that go to S_j.

Let R_n be the importance column vector of all the n document sources. That is, R_i $(1 \le i \le n)$ represents the importance of source S_i. Intuitively, if a source S_i $(1 \le i \le n)$ is important, the source S_j $(1 \le j \le n)$ that S_i frequently cites is also important. Also, the importance of a source is influenced by the importance of other sources according to the citation frequencies. If we regard $B_{i,j}$ as the proportion of S_i's importance that contributes to the importance of S_j, we have

$$R_i = \sum_{j=1}^{n} R_j \times B_{j,i} .$$

In matrix form, this is

$$R = B^T \times R .$$

Hence, R is the dominant eigenvector of B^T that corresponds to eigenvalue one.

In general, to ensure that matrix B is ergodic, we can use a method similar to the random surfer model in the PageRank algorithm [25] so that $\forall i, j$ $(1 \le i \le n, 1 \le j \le n)$: $B_{i,j} \neq 0$. Then R is guaranteed to be computable using a power method [25]. The computation of R only needs to be performed periodically whereas $A_{i,j}$'s need to be updated continuously. This allows us to keep track of the changes in source importance without incurring much computation overhead.

8. PERFORMANCE EVALUATION

We implemented a prototype of our framework on top of IBM's SPC stream processing middleware [17]. Our implementation uses two processing elements (PEs) that consume and produce streams of data through input and output ports, respectively. One PE produces the document stream and sends it to another PE that implements ONED. To the best of our knowledge, this is the first implementation of a real application in a large-scale stream processing system.

The latest TDT5 benchmark [34] was used to evaluate the performance of our techniques. This standard benchmark for NED contains 278,109 pieces of news that came from seven news agencies between April and September 2003. These news agencies include Associated Press, Agence France Presse, Xinhua, New York Times, Ummah, LA Times/Washington Post, and CNN. Each news source is treated as a news stream. Our measurements were performed on two computers, each with one 1.6GHz processor, 1GB main memory, one 75GB disk, and running Linux.

For an ONED system, the miss probability P_{miss} is the probability that a first-story document is incorrectly predicted as a non-first-story document. The false alarm probability P_{FA} is the probability that a non-first-story document is incorrectly predicted as a first-story document. In TDT [34], the performance of an ONED system is measured in terms of the normalized detection cost C_{Det} that is defined as follows:

$$C_{Det} = \frac{C_{Miss} \times P_{Miss} \times P_{target} + C_{FA} \times P_{FA} \times (1 - P_{target})}{\min\{C_{Miss} \times P_{target}, C_{FA} \times (1 - P_{target})\}},$$

where $C_{Miss}=1$ and $C_{FA}=0.1$ are preset costs. $P_{target}=0.02$ is the a priori probability of a target (i.e., first-story document). The smaller the normalized detection cost, the better the quality of the predictions made by an ONED system.

8.1 Techniques for Improving Efficiency

In this section, we evaluate the performance of our techniques for improving efficiency. Our techniques use a few parameters. The default parameter values are as follows: $T=100$ (the threshold for similarity value), $W=29$ (the sliding window size in days), $K=250$ (the number of top terms kept in each saved document), and $M=10$ (the number of top terms used for pre-filtering purpose). We perform a sensitivity analysis, using a set of experiments to evaluate the impact of parameter values on the quality of the predictions made by the ONED system and the document processing speed. In each experiment, we varied the value of one parameter while keeping the other parameters unchanged. In all these experiments, the memory was always large enough to hold all the identified first-story documents in the last W days.

W (Sliding Window Size)

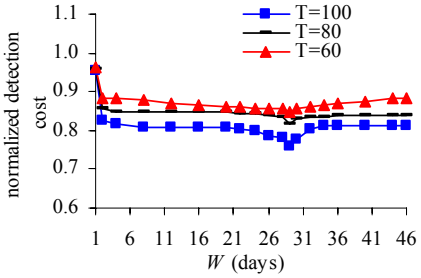

Figure 7. Normalized detection cost vs. *W*.

The first experiment concerns W, the size of the sliding window. Only old documents within the last W days are saved. The default value of W is 29. We varied W from 1 to 46. Figure 7 shows the impact of W on the normalized detection cost. (Note: to make figures in Section 8 more readable, the y-axis does not always start from zero.) If W is too large, many old, useless documents are saved. This makes it difficult to correctly identify the first-story documents. If W is too small, not enough old documents are saved to conserve useful information. Hence, many non-first-story documents are incorrectly predicted as first-story documents. Especially, there is a big jump of the normalized detection cost when W changes from 2 to 1, as a large number of events are reported within two consecutive days but not in a single day. The normalized detection cost reaches its smallest value when $W=29$ and becomes larger as W deviates from 29. The safe range for W is between 24 and 32. When W is within this range, our method can make good predictions.

Figure 8 shows the impact of W on the throughput of our ONED system. This throughput is measured by the number of documents that can be processed per second, where each piece of news is a document. The larger the W, the more documents are saved and the more old documents are included for comparison with the incoming new documents. Hence, the system throughput decreases as W increases.

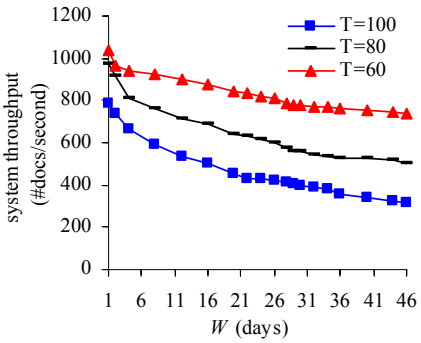

Figure 8. System throughput vs. *W*.

T (Threshold for Similarity Value)

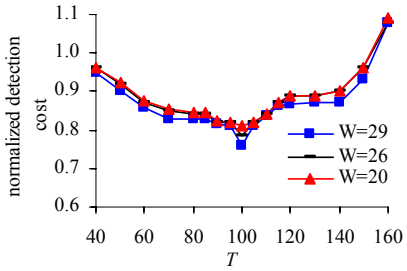

Figure 9. Normalized detection cost vs. *T*.

The second experiment concerns the threshold *T* for the similarity value. The default value of *T* is 100. We varied *T* from 40 to 160. Figure 9 shows the impact of *T* on the normalized detection cost. If *T* is too small, many first-story documents are incorrectly predicted as non-first-story documents and thus the false alarm probability is large. If *T* is too large, many non-first-story documents are incorrectly predicted as first-story documents and thus the miss probability is large. The normalized detection cost reaches its smallest value when *T=100* and becomes larger as *T* deviates from 100.

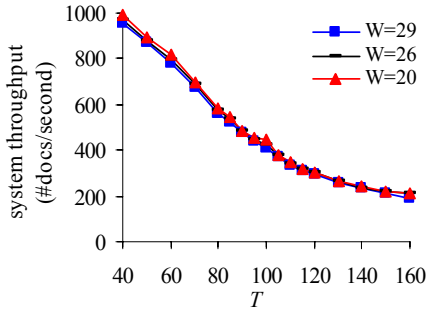

Figure 10. System throughput vs. *T*.

Figure 10 shows the impact of *T* on the throughput of our ONED system. The larger the *T*, the more documents are predicted and saved as first-story documents, and the more old documents are included for comparison with the incoming new documents. Hence, the document processing speed decreases as *T* increases.

K (Number of Top Terms Kept in Each Saved Document)

The third experiment concerns *K*, the number of top terms kept in each saved document. The default value of *K* is 250. We varied *K* from 50 to 400. Figure 11 shows the impact of *K* on the normalized detection cost. If *K* is too small, not enough information is captured in the computed similarity values and the detection accuracy will degrade. Hence, the normalized detection cost increases as *K* decreases. After *K* becomes larger than 100, the detection accuracy is not very sensitive to the value of *K*. When *K≥250*, the computed similarity values have captured enough information and increasing *K* more does not help much in improving the normalized detection cost.

M (Number of Top Terms Used for Pre-filtering)

The fourth experiment concerns *M*, the number of top terms used for pre-filtering. The default value of *M* is 10. We varied *M* from 3 to 25. Figure 12 shows the impact of *M* on the normalized detection cost. If *M* is too small, many old, relevant documents are not included for comparison with the incoming new documents. Hence, many first-story documents are incorrectly filtered out as non-first-story documents. If *M* is too large, many old, useless documents are compared with the incoming new documents. This makes it difficult to correctly identify the first-story documents. The normalized detection cost reaches its smallest value when *M=10* and becomes larger as *M* deviates from 10.

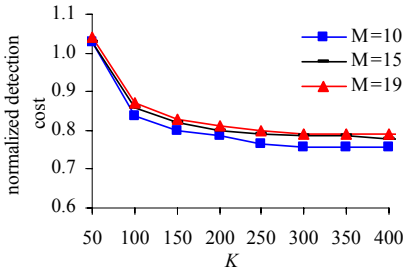

Figure 11. Normalized detection cost vs. *K*.

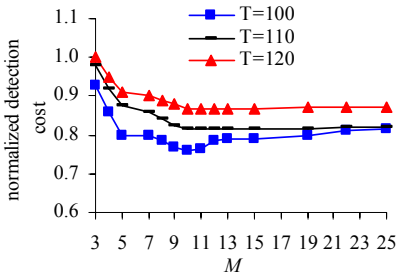

Figure 12. Normalized detection cost vs. *M*.

Figure 13 shows the impact of *M* on the throughput of our ONED system. The larger the *M*, the more old documents are included for comparison with the incoming new documents. Hence, the document processing speed decreases as *M* increases.

Using our techniques for improving efficiency, it takes 684 seconds (less than 12 minutes) to process all the documents in the TDT5 data set. In contrast, the baseline system described in Section 2 uses 285,908 seconds (more than three days) to process all the documents in the TDT5 data set. Compared to the baseline system, our techniques improve the efficiency by two orders of magnitude (418 times).

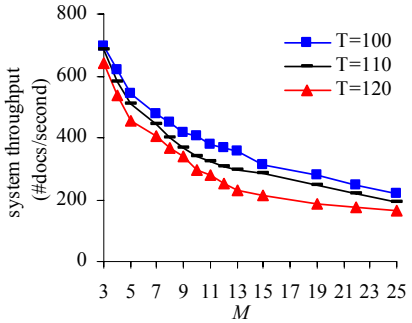

Figure 13. System throughput vs. *M*.

In the TDT5 competition, among all the participants, Stottler Henke Associates Inc. achieved the best normalized detection cost 0.7155 [8]. Using the default parameter values, our prototype achieves a normalized detection cost of 0.758. This number is only slightly (0.758/0.7155-1=6%) worse than the best result in the TDT5 competition.

In summary, with a minor degradation in detection accuracy, our method can significantly increase the document processing rate. Each parameter has a not-very-small safe range, within which our method can make good predictions. That is, the quality of the predictions is insensitive to parameter changes. However, when the parameter value is outside of this safe range, the quality of the predictions will degrade.

8.2 Resource-Adaptive Computation

In this section, we evaluate the performance of our resource-adaptive computation algorithms.

CPU-Bound Case

Let $R_{arrival}$ denote the document arrival rate, and T_{max} denote the maximum throughput of the ONED system when the default parameter values of our techniques are used. The load factor is defined as $u=R_{arrival}/T_{max}$. The ONED system is overloaded when $u>1$. In our ONED system, the input queue Q_i can hold at most $V=100$ documents. (The results for other values of V are similar and hence omitted.)

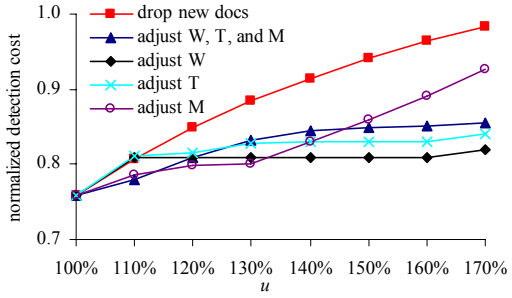

Figure 14. Normalized detection cost vs. u.

We varied the load factor u from 100% to 170%. Figure 14 shows the impact of u on the normalized detection cost for the five methods that improve throughput (see Section 5.1). Among these five methods, dropping new documents performs the worst. In most cases, adjusting W performs the best or close to the best. This is because among the three parameters W, T, and M, varying W tends to have the least impact on the normalized detection cost (see Figures 7, 9, and 12).

Memory-Bound Case

Let M_{actual} denote the number of identified first-story documents in the last W days that can be held in memory. That is, M_{actual} represents the memory size. In the TDT5 document set, at any time, our ONED system always identifies no more than $M_{full}=7{,}500$ first-story documents in the last $W=29$ days. (The TDT5 document set has only seven document sources. In a real world ONED system that monitors a large number of document sources, we would expect M_{full} to be much larger than 7,500 and hence memory overflow is much more likely to occur.) $q=M_{actual}/M_{full}$ roughly reflects the portion of identified first-story documents in the last W days that can be held in memory.

As mentioned in Section 5.2, when memory overflows, we can use the LRU algorithm, the random policy, or the time policy, to manage all the saved documents in memory. We varied q from 7% to 100%. Figure 15 shows the impact of q on the normalized detection cost. The smaller the memory, the less useful information can be held in memory and the worse the detection accuracy. Hence, the normalized detection cost increases as q decreases. It is natural that when $q<100\%$, the LRU policy always works better than the other two policies.

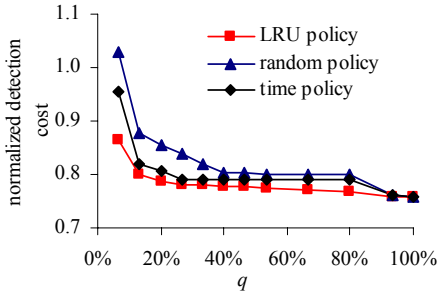

Figure 15. Normalized detection cost vs. q.

Compared to random old documents, those documents that are very old are much less likely to mention the same event as the new document and thus removing their information from memory will have a smaller impact on the detection accuracy. Therefore, when $q<100\%$, the time policy always works better than the random policy. When q is close to 100%, the performance difference among the three policies is minor. However, once q becomes less than 90%, the LRU policy exhibits significant performance advantages over the other two policies.

8.3 Output Load Shedding

In this section, we evaluate the performance of our output load shedding algorithm. Our algorithm uses two parameters. The default parameter values are as follows: $N=1000$ (the size of the output queue) and $p=0.9$ (the portion of Q_o that is Q_{o_new}). For the consumer of the ONED system, let r denote the ratio of its consuming rate to the output rate of the ONED system. In our tests, the default value of r is 0.5.

Recall that $C_{final}(D)$ denotes the final citation number of a first-story document D. We define the importance of D as $f(C_{final}(D))$, where $f(x)$ is a real non-decreasing function of x. We use the average importance value of the first-story documents that are processed by the consumer as the performance metric of an output load shedding policy. The larger the average importance value, the better the output load shedding policy.

The method described in Section 6.1 uses two queues Q_{o_new} and Q_{o_old} and hence is called the **two-queue policy**. When the output queue Q_o becomes full, we compare the two-queue policy with the following two policies:

Random policy: Drop a random document from Q_o.

Time policy: Drop the oldest document from Q_o.

Two functions are used: $f_1(x)=x$ and $f_2(x)=ln(x+1)$. We perform a sensitivity analysis, using a set of experiments to evaluate the impact of parameter values on the average importance value. In each experiment, we varied the value of one parameter while keeping the other parameters unchanged. In all these experiments, the consumer always consumes the oldest document in the output queue first. (We also tested other consuming policies and the results are similar.)

***N* (Size of the Output Queue)**

The first experiment concerns N, the size of the output queue. We varied N from 50 to 550. Figures 16 and 17 show the impact of N on the average importance value when functions $f_1(x)$ and $f_2(x)$ are used, respectively.

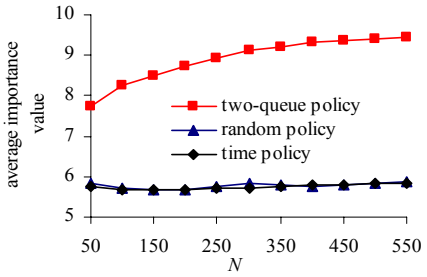

Figure 16. Average importance value vs. N ($f_1(x)=x$).

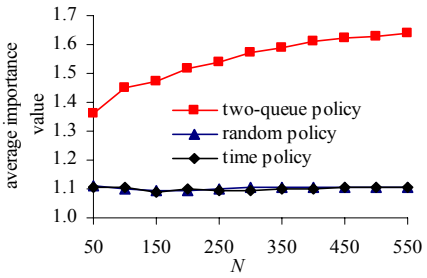

Figure 17. Average importance value vs. N ($f_2(x)=ln(x+1)$).

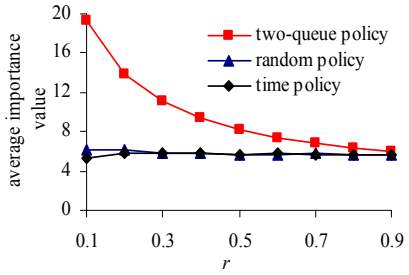

Figure 18. Average importance value vs. r ($f_1(x)=x$).

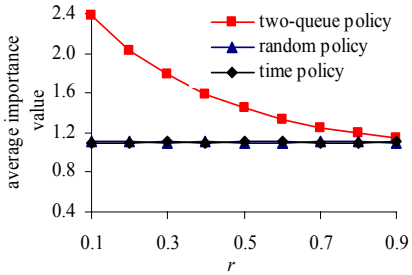

Figure 19. Average importance value vs. r ($f_2(x)=ln(x+1)$).

When dropping documents from the output queue, neither the random policy nor the time policy considers the importance values of the documents and hence the dropped documents have random importance values. Thus, the random policy and the time policy have roughly the same performance, which is not influenced by N much. In contrast, the two-queue policy considers the importance values of the documents and performs much better than the other two policies. The larger the N, the more documents that the two-queue policy can consider in making output load shedding decisions and hence the better the decisions. Thus, the average importance value of the two-queue policy increases with N. The above observations apply to both $f_1(x)$ and $f_2(x)$.

r (Ratio of the Consuming Rate to the Output Rate)

The second experiment concerns r, the ratio of the consumer's consuming rate to the output rate of the ONED system. We varied r from 0.1 to 0.9. Figures 18 and 19 show the impact of r on the average importance value when functions $f_1(x)$ and $f_2(x)$ are used, respectively. Due to the same reason discussed above, the random policy and the time policy have roughly the same performance, which is not influenced by r much. The two-queue policy performs much better than the other two policies. The smaller the r, the fewer documents are consumed by the consumer of the ONED system and hence the more selective the two-queue policy can be in choosing these documents. Thus, the average importance value of the two-queue policy increases as r decreases.

p (Portion of Q_o that is Q_{o_new})

The third experiment concerns p, the portion of Q_o that is Q_{o_new}. In the two-queue policy, we varied the parameter p from 2% to 98%. Figures 20 and 21 show the impact of p on the average importance value when functions $f_1(x)$ and $f_2(x)$ are used, respectively. Due to the same reason discussed above, the random policy and the time policy have roughly the same performance, which is not influenced by p. The two-queue policy performs much better than the other two policies. When p is very small, Q_{o_new} is small.

Hence, a document does not have much chance to accumulate citations before it is moved to Q_{o_old}. Then in Q_{o_old}, young documents tend to have fewer citations than old documents and get dropped from the output queue first when the consumer is overloaded. This deteriorates the average importance value of the two-queue policy, as such young documents can be important ones if they will receive a lot of citations in the future. When p is very large, few documents are stored in Q_{o_old}. Then the two-queue policy does not have many candidates to choose when making output load shedding decisions. This also deteriorates the quality of the decisions made by the two-queue policy. The average importance value of the two-queue policy reaches its maximum value when p=90% and decreases as p deviates from 90%.

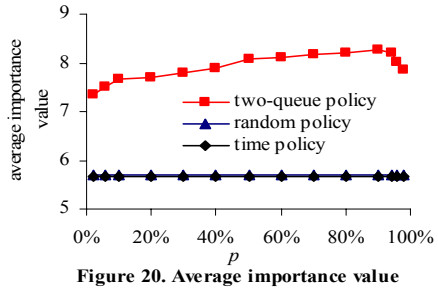

Figure 20. Average importance value vs. p ($f_1(x)=x$).

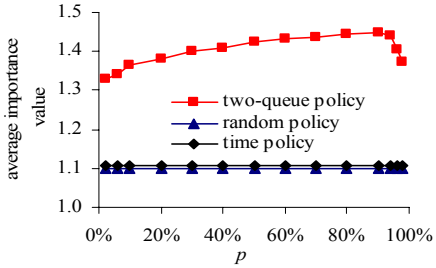

Figure 21. Average importance value vs. p ($f_2(x)=ln(x+1)$).

8.4 Ranking Document Sources

Table 1. Document source ranking.

source	rank	eigenvalue
Associated Press	1	1
Agence France Presse	2	0.9912
Xinhua	3	0.5568
New York Times	4	0.1981
Ummah	5	0.0586
LA Times/Washington Post	6	0.0373
CNN	7	0.0156

In this section, we evaluate the performance of our document source ranking algorithm. The TDT5 document set has seven document sources. Table 1 shows the computed ranking of all these sources after the entire document set has been processed. All the sources are sorted in decreasing order of their eigenvalues. We acknowledge that ultimately the judgment of the importance of document sources is a subjective issue, just like the evaluation of Web page ranks provided by Google's PageRank algorithm. Moreover, unlike some well-understood (but yet controversial) rankings such as school ranking, there is no authoritative ranking of international news agencies. Despite these difficulties in evaluation, we consider the computed source order reasonable and consistent with our real-world experience. Associated Press (AP) is the largest news agency in the world and has the highest rank. Agence France Presse is the third largest news agency in the world and ranked the second in the sorted list. Xinhua is the most authoritative news agency in P.R. China. It is often the first one in reporting important news related to China. As the world is gaining more and more interest in China, those news related to China are widely cited by the other news agencies. Hence, Xinhua is ranked the third among all the seven sources. Companies such as CNN are major players in the public media market but not the most important news agencies, because they employ few news reporters and produce only a low volume of news. Most news appearing on their media are purchased from other news agencies and not included in the TDT5 document set. Therefore, it is not surprising that these companies are ranked low in the sorted list.

We performed a second experiment to show the effect that our computed document source ranking considers not only the number of first-story documents but also the number of citations received by a first-story document. In Table 1, AP is the most important document source. We split the AP documents into s equal-sized AP partitions and treat each partition as a separate document source. We compute a new ranking after replacing the original AP source with the s AP partitions. We varied s from 1 to 5. The results show that for a fixed s, all the AP partitions have similar (consecutive) ranks. Figure 22 shows the highest rank of all the AP partitions. The larger the s, the fewer first-story documents are emitted from an AP partition while the average number of citations received by a first-story document remains the same. Hence, the highest rank of all the AP partitions drops gradually as s increases, which matches with our expectation.

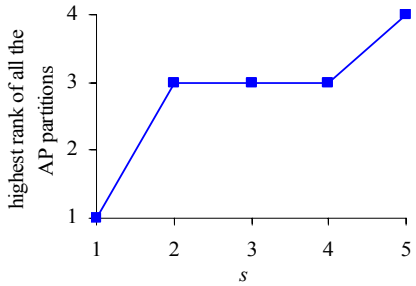

Figure 22. Highest rank of all the AP partitions vs. s.

9. RELATED WORK

ONED has been studied before [2, 3, 7, 11, 19, 21, 23, 29, 37, 38]. However, there is a gap between the existing ONED systems and a system that can be used in practice. This paper attempts to close this gap.

Sentence-level novelty detection has been studied in the TREC Novelty Track [35] and [1, 4]. Zhang et al. [39] proposes performing NED in an information filtering environment. Extending our techniques to these two environments is an interesting area for future work.

Bharat et al. [6, 9, 10, 14] consider the problem of finding near-duplicate documents on the Web. In our case, we focus on finding documents that mention the same event and these documents are usually not near-duplicates of each other.

Based on the real-time resource availability information, Arnt et al. [5] proposes dynamically composing the information retrieval techniques that are used to answer queries. In our case, when either memory overflows or the consumer of the ONED system is overloaded, those documents that are regarded as less important for detection or presentation purposes are thrown away.

The output queue is a buffer between the ONED system and the consumer. Breaking a buffer into two parts has been proposed in the 2Q buffer management algorithm [18]. 2Q focuses on improving buffer hit ratio while our output load shedding algorithm focuses on dropping less important documents.

Tu et al. [40] use a control theory based approach to perform load shedding in data stream management systems. They treat all data tuples equally and the goal is to keep the average tuple processing delay below a threshold. In contrast, we explicitly differentiate the importance of documents and the goal is to minimize the loss in detection accuracy.

Ranking news sources has been considered in Corso et al. [12]. The ranking method in Corso et al. [12] does not consider the arrival time of news. If two news sources S_1 and S_2 report the same set of news while S_1 always reports before S_2, the method in Corso et al. [12] will give the same rank to S_1 and S_2. In contrast, our document source ranking algorithm considers the timeliness that events are reported by document sources and will rank S_1 higher than S_2.

10. CONCLUSION

This paper proposes a comprehensive framework for online new event detection and improves an ONED system from four perspectives: efficiency, resource-adaptive computation, user interface, and document source ranking. We implemented a prototype of our framework on top of a stream processing middleware. Our experiments with the standard TDT5 benchmark show that the proposed techniques can improve the document processing rate by two orders of magnitude without sacrificing much detection accuracy. When resources are tight, our techniques can maximize the benefit that can be gained from the limited resources. When the consumer of the ONED system is overloaded,

our techniques automatically drop less important documents and only present the most important ones to the consumer. Moreover, the computed importance ranking of document sources matches with our real world experience.

11. ACKNOWLEDGEMENTS

We would like to thank Jiuxing Liu and Xiaoqiang Luo for helpful discussions.

12. REFERENCES

[1] J. Allan, R. Gupta, and V. Khandelwal. Temporal Summaries of News Topics. SIGIR 2001: 10-18.

[2] J. Allan, V. Lavrenko, and H. Jin. First Story Detection in TDT is Hard. CIKM 2000: 374-381.

[3] J. Allan, R. Papka, and V. Lavrenko. On-Line New Event Detection and Tracking. SIGIR 1998: 37-45.

[4] J. Allan, C. Wade, and A. Bolivar. Retrieval and Novelty Detection at the Sentence Level. SIGIR 2003: 314-321.

[5] A. Arnt, S. Zilberstein, and J. Allan et al. Dynamic Composition of Information Retrieval Techniques. J. Intell. Inf. Syst. 23(1): 67-97, 2004.

[6] K. Bharat, A.Z. Broder, and J. Dean et al. A Comparison of Techniques to Find Mirrored Hosts on the WWW. IEEE Data Eng. Bull. 23(4): 21-26, 2000.

[7] T. Brants, F. Chen. A System for New Event Detection. SIGIR 2003: 330-337.

[8] R. Braun, R. Kaneshiro. Exploiting Topic Pragmatics for New Event Detection in TDT-2004. TDT-2004 Workshop.

[9] A. Z. Broder. Identifying and Filtering Near-Duplicate Documents. CPM 2000: 1-10.

[10] Y. Bernstein, J. Zobel. Redundant Documents and Search Effectiveness. CIKM 2005: 736-743.

[11] F. Chen, A. Farahat, and T. Brants. Story Link Detection and New Event Detection are Asymmetric. HLT-NAACL 2003.

[12] G.M. Corso, A. Gulli, and F. Romani. Ranking a Stream of News. WWW 2005: 97-106.

[13] M. Clayton. US Plans Massive Data Sweep. The Christian Science Monitor, February 09, 2006. http://www.csmonitor.com/2006/0209/p01s02-uspo.html, 2006.

[14] J. Cho, N. Shivakumar, and H. Garcia-Molina. Finding Replicated Web Collections. SIGMOD Conf. 2000: 355-366.

[15] D.J. DeWitt, J. Gray. Parallel Database Systems: The Future of High Performance Database Systems. CACM 35(6): 85-98, 1992.

[16] Google News Homepage. http://news.google.com, 2006.

[17] N. Jain, L. Amini, and H. Andrade et al. Design, Implementation, and Evaluation of the Linear Road Benchmark on the Stream Processing Core. SIGMOD Conf. 2006: 431-442.

[18] T. Johnson, D. Shasha. 2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm. VLDB 1994: 439-450.

[19] G. Kumaran, J. Allan. Text Classification and Named Entities for New Event Detection. SIGIR 2004: 297-304.

[20] J.M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. JACM 46(5): 604-632, 1999.

[21] X. Li, B.W. Croft. Novelty Detection Based on Sentence Level Patterns. CIKM 2005: 744-751.

[22] E. Lipton. Software to Monitor Overseas Opinions of U.S. The New York Times, October 4, 2006. http://news.zdnet.com/2100-9588_22-6122641.html, 2006.

[23] Z. Li, B. Wang, and M. Li et al. A Probabilistic Model for Retrospective News Event Detection. SIGIR 2005: 106-113.

[24] C.D. Manning, H. Schütze. Foundations of Statistical Natural Language Processing. MIT Press, 1999.

[25] L. Page, S. Brin, and R. Motwani et al. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.

[26] M.F. Porter. An Algorithm for Suffix Stripping. Program 14(3): 130-137, 1980.

[27] M. Persin, J. Zobel, and R. Sacks-Davis. Filtered Document Retrieval with Frequency-Sorted Indexes. JASIS 47(10): 749-764, 1996.

[28] S.E. Robertson, S. Walker, and M. Hancock-Beaulieu. Okapi at TREC-7: Automatic Ad Hoc, Filtering, VLC and Interactive. TREC 1998: 199-210.

[29] N. Stokes, J. Carthy. Combining Semantic and Syntactic Document Classifiers to Improve First Story Detection. SIGIR 2001: 424-425.

[30] A. Singhal. Modern Information Retrieval: A Brief Overview. IEEE Data Eng. Bull. 24(4): 35-43, 2001.

[31] T. Sakai, K.S. Jones. Generic Summaries for Indexing in Information Retrieval. SIGIR 2001: 190-198.

[32] SMART Stopword List. http://www.lextek.com/manuals/onix/stopwords2.html, 2005.

[33] C. Tang, S. Dwarkadas. Hybrid Global-Local Indexing for Efficient Peer-to-Peer Information Retrieval. NSDI 2004: 211-224.

[34] TDT Homepage. http://www.nist.gov/speech/tests/tdt.

[35] TREC Novelty Track. http://trec.nist.gov/tracks.html, 2004.

[36] Yahoo! News Homepage. http://news.yahoo.com, 2006.

[37] Y. Yang, T. Pierce, J.G. Carbonell. A Study of Retrospective and On-Line Event Detection. SIGIR 1998: 28-36.

[38] Y. Yang, J. Zhang, J.G. Carbonell et al. Topic-conditioned Novelty Detection. KDD 2002: 688-693.

[39] Y. Zhang, J.P. Callan, T.P. Minka. Novelty and Redundancy Detection in Adaptive Filtering. SIGIR 2002: 81-88.

[40] Y. Tu, S. Liu, S. Prabhakar et al. Load Shedding in Stream Databases: A Control-Based Approach. VLDB 2006: 787-798.