Real-Time Summarization of Big Data Streams

Echtzeit Zusammenfassung von Big Data Streams

Master-Thesis von Andreas Rücklé aus Darmstadt

Dezember 2015

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Ubiquitous Knowledge Processing
& Distributed Systems Programming

Real-Time Summarization of Big Data Streams
Echtzeit Zusammenfassung von Big Data Streams

Vorgelegte Master-Thesis von Andreas Rücklé aus Darmstadt

1. Gutachten: Prof. Dr. Iryna Gurevych
2. Gutachten: Prof. Dr. Patrick Eugster

Tag der Einreichung:

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 14. Dezember 2015

(A. Rücklé)

Abstract

Events like natural disasters, riots or protests trigger an increased information need for many people, because of regional closeness, social relations or general interest. Due to a high amount of news-articles that are created by different publishers during such events, it is nearly impossible for individual persons to process all information with the goal of staying up-to-date. Real-time summarization systems can help in such cases by providing persons with updates on the event while the situation still is developing, without requiring the individual person to manually analyze a large amount of news-articles. In this master thesis, a framework for real-time summarization is presented and multiple summarization systems based on this framework are introduced. Besides achieving a good summarization quality, another focus of this work was to retain real-time properties in terms of summarization and in terms of computational performance. Based on a simple approach defined as Baseline, different improvements were made with the goal to create an advanced system which achieves a performance similar to other state-of-the-art temporal summarization systems. The best resulting system of this work is an adaptive approach which is able to change configurations and algorithms during run-time to automatically select the best method to summarize each target-event. The adaptive selection is performed by detecting the importance of an event, based on its news-coverage. The system also makes use of an approach that requires all information to be reported by multiple sources before it can be included in an update. The adaptive summarization system showed superior results in terms of summarization quality compared to the Baseline system. Furthermore, a comparison to a state-of-the-art temporal summarization system also showed better results of the adaptive approach. At the same time, all real-time goals were achieved.

Zusammenfassung

Verschiedene Events wie etwa Naturkatastrophen oder Protestbewegungen rufen ein erhöhtes Informationsbedürfnis bei unterschiedlichen Personengruppen hervor. Diese sind entweder direkt betroffen, kennen betroffene Personen oder haben ein spezielles Interesse an dem Thema. Während eines solchen Events veröffentlichen zahlreiche Zeitungen und Online-Redaktionen eine große Menge an Nachrichten die sich auf das Event beziehen. Das macht es nahezu unmöglich für einzelne Personen alle Informationen zu verarbeiten um vollumfänglich informiert zu bleiben. Systeme für Echtzeit Zusammenfassung (”Real-Time Summarization”) können in solchen Fällen helfen indem sie interessierten Personen Zusammenfassungen und Updates zu Events anbieten während die Vorgänge noch laufen, ohne dass die Personen selbst eine große Menge an Nachrichtentexten verarbeiten müssen. In dieser Master Thesis wird ein Framework für Real-Time Summarization vorgestellt und verschiedene konkrete Umsetzungen von Systemen auf Basis dieses Frameworks gezeigt. Neben dem Erreichen einer guten Qualität für die Zusammenfassungen wurde eine weiterer Schwerpunkt auf das Erreichen bestimmter Echtzeit-Anforderungen gesetzt, einerseits im Zusammenhang mit dem Erstellen von Zusammenfassungen und andererseits im Zusammenhang mit dem Rechenaufwand. Basierend auf einem einfachen Ansatz, der als Baseline definiert wurde, sind verschiedene Verbesserungen und Optimierungen entwickelt worden mit dem Ziel eine Qualität für die Zusammenfassungen zu erreichen, die vergleichbar mit anderen state-of-the-art Systemen ist. Das beste System dieser Arbeit ist ein adaptiver Ansatz, der Konfigurationsparameter und Algorithmen während der Laufzeit wechseln kann um die beste Methode zur Zusammenfassung zu wählen. Dies wird erreicht indem die Wichtigkeit des Events erfasst wird, basierend auf dem Umfang der Berichterstattung. Das adaptive System nutzt außerdem einen Ansatz der sicherstellt, dass eine Information von mehreren Quellen berichtet wurde bevor sie ausgegeben wird. Für dieses adaptive System konnte eine bessere Qualität der Zusammenfassungen erzielt werden als für das Baseline-System. Auch im Vergleich zu einem state-of-the-art System konnte eine bessere Performance des adaptiven Systems festgestellt werden. Ebenso wurden alle gestellten Echtzeit-Anforderungen von diesem System erfüllt.

Contents

List of Figures

List of Tables

1 Introduction

Important and ongoing events such as natural disasters, plane crashes, protests and riots require affected persons to stay informed at any point in time to reduce the chance of being caught in a dangerous situation. Third parties may also be interested in most recent information and event-updates when friends or relatives are involved in the ongoing situation or because of geographical attachment [33] and general interest in the related topics. News media during such events usually publish a large amount of news-stories, which are highly redundant and may also contain mistakes and wrong information. For individuals it is often not possible to review all the stories in detail, because of the broad news-coverage and the high number of information sources. Traditional approaches for news-consumption and aggregation fail in this scenario. Google News for example is a capable tool of showing important headlines and news-articles, even for ongoing events. But it is not capable of summarizing the overall content of the underlying news-articles. As another example, Twitter currently also lacks the ability to create summaries for events, even though it is likely to contain the desired information on its platform. Through a hash-tag search on this platform, information on events can be found in real-time. However, this information is not properly filtered which overloads the user with content and does not provide a good solution for the problem described earlier. One approach that can help are live news-tickers that traditional publishers and news-websites often provide for specific high-impact events such as natural disasters. These tickers are updated whenever a new sub-event is detected and therefore, the approach is similar to a real-time event summary. The major downside of such a ticker-based approach is the involvement of human editors which perform sub-event detection and do the writing of the updates. Updates are therefore expected to rely only on few data sources that are real-time, such as the sources which major news agencies provide, to not overload the editors themselves with information. Since editors may have different opinions on events, the resulting ticker entries may potentially be biased towards certain opinions. Another downside that comes with human editors is a distortion of the real-time aspect, because the editors may not be available during all times of the day and the creation of new update texts for humans is time consuming.

An approach using automatic summarization to create summaries and providing updates on events in real-time as they happen has a significant potential to improve the current situation. By using a high number of information sources, such as all major and all local news stations in an area, a differentiated and more neutral summary of the event could be created. Such an approach would therefore rely on Big Data streams of news-articles. In comparison to current news-tickers, the automated approach would not bind any human resources, allowing to summarize a large number of events and even non-high-impact events. Many other improvements could be made possible with such an automated system. For example, the time until a new update is detected could be reduced to a minimum, because news-articles would be analyzed in near real-time. Any delays imposed by the effect of relying on human editors would not be present for such an approach. Furthermore, such a system can base its decisions on many more news sources than an editor-based approach, which can increase the probability of detecting false and wrong information that should not be included in a summary update. Different approaches for similar systems exist [1, 23, 24, 35], however most of them are not capable of processing news-articles in real-time, as they are published. Furthermore state-of-the-art systems leave significant room for improvement, especially for systems that emphasize on a real-life scenario where the summary size has to be sufficiently small to be suitable for the usage during an actual event without overloading the user with too much information.

In this work, an automated summarization framework is created, which is capable of summarizing multiple events with a special focus on real-time aspects, streaming data and concurrency. The framework implementation abstracts from all the technical details that are required to run such a real-time summarization system, which includes database access, component structures and parallelism features. Based on this framework, different summarization systems are introduced. First, a baseline approach is shown, which is based on simple algorithms that were optimized for real-time summarization. Different improvements based on this baseline approach are presented, such as a component that requires multiple sources to report an information before it allows the information to be included in an update. The primary achievement of this work is an adaptive approach that is capable of automatically changing system configurations for target-events with the goal to choose the best summarization approach for each event, based on its current importance which is measured by the overall related news-coverage. Important attributes and goals for all summarization systems of this work were the quality of the resulting summaries, but also the real-time aspects. Real-time in the broader summarization context means that the system constantly analyzes the input stream and updates the summary whenever a new sub-event is detected. An approach that partitions the input data into several segments which are analyzed separately would not be considered as real-time, since it introduces a major delay. Real-time in terms of a computational aspect on the other side means that there is an upper bound for the time that the summarization system requires to fully process an item of the input stream. Both aspects are covered in this work.

This work is structured as follows: In section 2, related work and related approaches are presented. In section 3, the overall technical foundation of the summarization systems is described and the underlying framework implementation is motivated. In the main section 4, different summarization systems are introduced, based on the framework created earlier. This includes a baseline system and different improvements on baseline. The creation of the individual summarization systems is followed by an evaluation and a comparison of these systems in section 5, where the summary as well as the computational details of the systems are analyzed. The last section 6 provides an overview over the achievements of this work and motivates future work.

2 Related Work

2.1 Single- and Multi-Document Summarization

Automatic summarization is a research area studied since the middle of the 20th century, which was first publicly discussed by Luhn [21]. Many different approaches were created until today. One major differentiation between the different summarization approaches is the input, which can either be a single document or multiple documents. Single-document summarization is the process of generating an abstract or a short summary which describes the content of a single document. In contrast, multi-document summarization relies on multiple input documents on the same topic to create one summary for all of them. Documents for summarization can be different kinds of texts, for example news-articles, scientific papers, e-mail messages or even conversations. One other key differentiation of approaches is how the the summary is created. This can either be a concatenation of extracted sentences or an artificially generated text, based on the sentences and information included in the input documents.

2.1.1 Extractive Approaches

The goal of extractive summarization approaches is to build a summary by extracting sentences from a single or multiple text sources. The extracted sentences are then concatenated to form the output summary. The primary challenge for such approaches is the identification of relevant sentences which describe the topic and the content of the input sufficiently well.

Basic approaches are based on algorithms that only use term frequencies and term probabilities. The term probability is computed by $p(w) = \frac{n}{N}$ where n is the term frequency (number of occurrences of w in the input) and N is the number of word tokens in the input document. The SumBasic summarization approach [30] for example relies on the assumption, that on-topic sentences are expected to have a higher average term probability value compared to sentences that are off-topic or less relevant within the topic. Sentences with the highest average probability are extracted in a greedy fashion to form the summary (probabilities are altered after each step). Improvements can be made by including additional information from a background corpus in the summarization process. Such a corpus provides data about common distributions of terms within documents of the same type. With this information, the frequency of each term (or a normalized value) in the input can be compared against the data of the background corpus. For example, TF*IDF[1] weights [27] can be computed for all terms in the input. Since topic-related terms are expected to occur more often in the input and less often in the background corpus, a high TF*IDF weight represents important and descriptive terms. TF*IDF weights are easy and fast to compute, therefore many systems and algorithms in extractive single- and multi-document summarization are using them to some extend [9,16,24]. To extract sentences using TF*IDF weights, greedy approaches rank sentences according to the average TF*IDF scores of all terms and choose the highest ranked sentences. An alternative to this is the detection of topic-signatures [20], which is based on the Log Likelihood Ratio. Topic-signatures are terms that are expected to be highly descriptive or highly relevant for the topic of the input. Because terms can either be topic-signatures or not, this is a binary measure. The basic idea behind identifying terms as topic-signatures is to check if a term occurs significantly more often in the input than it would occur by chance, measured by statistics on the background corpus. In the context of summarization, a sentence is important if it

[1] Term Frequency * Inverse Document Frequency

contains many topic signatures. For extractive summarization, an algorithm can for example calculate the (normalized) number of topic signatures for each sentence in the input and extract sentences with the highest values. Approaches using topic-signatures often produce better results than standard frequency-based methods [12].

More sophisticated ways to create single- or multi-document summaries include supervised machine learning methods as well as clustering- and graph-based approaches. To identify sentences that should be included in the summary, supervised machine learning approaches can be used to classify sentences as important (included in the summary) or not important (not included in the summary). Many different features are possible, for example the existence of specific vocabulary, the sentence position in the document, sentence length, existence of named entities or the number of topic-signatures in the sentence. For extractive summarization, different classifiers can be used, for example Support Vector Machines (SVM) [11], Naive Bayes or Hidden Markov Models (HMM) [6]. An advantage of using a HMM-based approach is that it only has few assumptions of independence. This is an advantage in summarization, because the HMM does not assume that a sentence probability is independent from the previously selected sentences which can be used to reduce redundancies. Clustering approaches are especially used for multi-document summarization [13] where more input data is available than in single-document summarization. Clusters in such approaches contain sentences with similar content, therefore the concatenation of representative sentences from each cluster can form an extractive summary. Graph-based approaches are based on a similar idea, with the advantage that a sentence is not required to be part of only one cluster. One popular graph-based summarization method is LexRank [9] which is based on the popular PageRank algorithm.

Besides taking one or more documents as an input, there are approaches that also make use of a target-query for the summarization process. Such query-focussed summarization approaches are relevant for different real-life scenarios, for example to summarize events based on a large amount of news-articles. Different systems that are motivated by such real-life scenarios incorporate query-data to some extend [2, 16, 23, 24]. One simple and popular query-focussed algorithm that is capable of creating a summary for single or multiple documents is MMR [4]. It produces a summary by greedily selecting sentences that have a maximum similarity to the target-query and at the same time have a minimum similarity to any previously selected sentence. Therefore, this approach also performs redundancy-removal.

2.1.2 Generative Approaches

In comparison to extractive approaches for summarization, where the summary is created from sentences that are extracted from the input documents, generative summarization includes sentences that are artificially created. These can be modified versions of sentences from the input documents or new sentences created based on extracted information. Generative summarization is motivated by the observation that a simple concatenation of different sentences from the original texts, even if they are highly descriptive for the original document, sometimes form summaries that are not fluent to read. Furthermore, extracted sentences may consist of multiple parts with only one part being relevant for the summary.

Different approaches for generative summarization exist, for example sentence compression, where parts of sentences are tried to be removed to create a compressed sentence that is more concise and

compact [15, 29, 32]. Sentence compression can be used to improve summaries or to meet certain restrictions on sentence length, for example in headline-generation tasks. Another approach in generative summarization is sentence fusion, which tries to merge different sentences into a new sentence that contains all major information from the original input sentences [3]. Sentence fusion can for example be used to create compact sentences that contain different relevant information pieces. Generative approaches in comparison to extractive approaches add an additional amount of complexity to the overall process, because the performance of a sentence generation is much more complex to assess than in simple sentence extraction.

2.2 Update Summarization

Multi-document summarization receives documents of the same topic as an input and creates a single summary for them. This is a retrospective approach, because in multi-document summarization the summary is only generated once, without being capable of summarizing events over time or updating the summary when new information emerges. Update summarization tries to solve this issue by providing updates to previous summaries, which only contain new or changed information. Update summarization as defined in TAC 2008 [8] is the task of creating an update summary based on another, previous summary that was created beforehand, and a set of new documents that contain new information. The goal of update summarization is to present the user, who is expected to know all details of the previous summary, with new and relevant updates only. Redundancies with the previous summary should be avoided. The intention behind the classical update summarization approach was to summarize only a small amount of data. This is reflected in the update summarization tasks of TAC 2008 and TAC 2009, which were based on a small data set that only contained 20 documents per topic.

Systems in the area of update summarization often choose a two-step approach. The first step is the generation of a multi-document summary, based on the set of new documents. The second step is a redundancy-removal step to remove any sentences which contain information that is redundant with information of the previous summary [7, 10].

2.3 Temporal Summarization

When using update summarization over several subsequent time windows of an event stream, a temporal summary can be created that is updated in regular time intervals. Incremental update summarization (IUS) is an approach based on this consideration, trying to automatically summarize long-running events over time [24]. The first layer of the IUS approach is an update summarization system which creates update summaries over time. These summaries are then used as an input for the second layer that performs the incremental update summarization. This layer decides whether to include individual sentences of the update summary in the resulting temporal summary or not. The improvement against a basic update summarization system is a more dynamic nature of the IUS system, because it is able to decide how much update sentences should be added to the temporal summary. This decision can be based on different measures such as the overall novelty of the information or the quality of each individual sentence. The overall intention is similar to the TREC Temporal Summarization (TREC-TS) challenge [2]. The motivation of this challenge is to simulate a system that takes a stream

of documents as an input and generates temporal summaries, based on these documents and a list of target-queries, which describe events that should be summarized. Whenever a new sub-event related to the event of a target-query is detected, an update for the summary of this event should be emitted by the system. Several participating teams developed systems for this challenge which relied on different approaches incorporating techniques ranging from query expansion and text clustering [35] to pipelined architectures with classifier-based sentence extraction [23].

Even though this is a highly active research area, temporal summarization is not new. In 2001 Allan et al. already defined a temporal summarization approach and developed a system which was capable of creating a temporal summary by detecting event-updates in hourly time-windows [1].

2.4 Real-Time Summarization

Many systems in the area of temporal summarization use incremental approaches which process documents over fixed time windows. The resulting systems therefore can not react to new data in real-time, since they have to wait until the current time window is closed. One example of the few real-time capable systems for news-articles is the system developed by McCreadie et al. for TREC-TS 2014 [23]. They developed an architecture that can make decisions to emit a new update for the temporal summary at any time, for each incoming document. Most other systems of the same challenge relied on approaches using fixed time windows.

An area where the real-time aspect is much more present is the summarization of social media message streams, especially for Twitter. Real-time capable systems in this area are highly attractive, because they can be used in real-life scenarios by using the Twitter streaming APIs. Different systems were developed to summarize events in real-time by detecting important sub-events on Twitter message streams [34, 36]. Other similar areas where a real-time analysis based on Twitter was created is the area of real-time sentiment analysis [31] and real-time event detection [26]. Summaries in the area of Twitter usually are shrinked versions of the underlying hashtag-filtered message stream. Compared to temporal summarization, events on Twitter are often short-term. A popular event-type for summarization on Twitter is sports-games. For such short-term topics, sub-events are usually simple actions like goals (soccer) or touchdowns (football). Even though real-time in this context means that the summarization system is able to extend or update the summary at any point in time, computational real-time properties are usually not part of related work.

3 Framework and Technical Foundation

On the internet, there are many different news-sources which publish news-articles on recent events every day. During high-impact events, different users may want to stay up-to-date as the event develops, because they or their relatives and friends could be affected by it. In such a situation it is almost impossible for a single person to scan all news-articles and evaluate the individual information-nuggets for themselves. An automated system with real-time access to a large number of news-sources can help by providing a summary on the event, which is updated whenever a significant sub-event is detected. Updates on the event would help an interested person to stay informed during the ongoing situation. In this work, different approaches for real-time summarization systems are introduced. To build such systems in the most developer efficient way possible, a unified framework and an abstraction from all technical details is required. In this section, the framework that was used throughout the work for all summarization systems is introduced and the implementation of the framework is outlined.

The overall summarization process in this work starts with an input stream of news articles which are extracted from the web. These articles are processed by a specific summarization system, which has access to a list of target-queries that describe interesting events in a few words (e.g. ”Costa Concordia” or ”Queensland Floods”). The output of such a system is a stream of updates for each target-query. This overall setting is visualized in figure 1. The framework introduced in this section serves the purpose as a foundation for the different summarization systems, abstracting from technical details and at the same time providing consistent interfaces for specific parts of the summarization system. This leads to good interchangeability properties.

In the following subsections, requirements and the design and implementation of the framework is described in detail. Additionally, the datasource and data preprocessing steps are shown, forming the input corpus for all summarization systems that were created as part of this work.

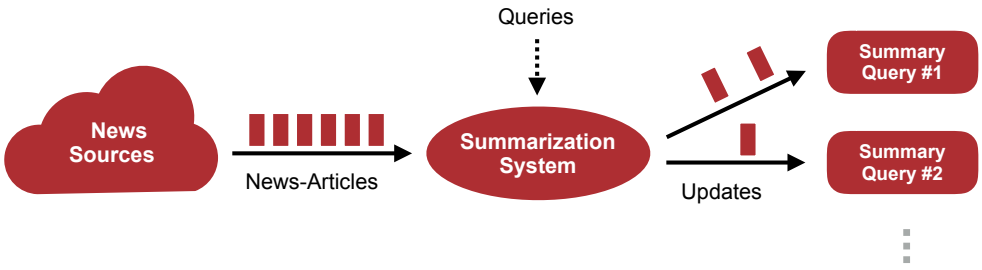

Figure 1: A visualization of the general idea and motivation of the summarization setting. On the internet there are a large number of news-sources that publish news-articles throughout the day. The stream of news-articles is used by the summarization system to create update-streams for individual summaries.

3.1 Framework Requirements

For the overall framework, several different requirements were identified. Besides some essential requirements that correspond to the capability of summarizing news-articles over time, the following important and high-level technical requirements especially stand out:

- **Stream processing**: All data should be processed as streaming data. The input of the framework is a stream of news articles which arrive as soon as they are published (or a simulation of such). The output of the framework is a stream of query/update pairs. The framework furthermore has access to a list of queries that define which events should be included in the summarization process.
- **Interchangeability**: The framework should provide the general structure of the summarization system by defining interfaces for all critical parts of the system. This allows a quick replacement of any part of the system. The requirement is also intended to enable quick prototyping of different approaches to implement a summarization system.
- **Parallelism**: The framework should provide support to parallelize the data processing and summarization process. This yields faster results on simulated input streams or, in practice, could enable the handling of sudden high amounts of incoming data (spikes).
- **Technical Abstractions**: An actual summarization system that is based on the framework implementation should not contain any code related to technical requirements.
- **Multiple Query Support**: The framework should be able to support multiple queries at once without sacrificing any of the other requirements (e.g. parallelism).

The framework design which is introduced in the following section 3.2 is based upon these requirements. The implementation of this design as presented in section 3.3 enforces these requirements for all summarization system implementations that are based on this framework.

3.2 Framework Design

The overall framework is designed to support stream processing throughout the whole architecture and to allow parallel computation at different stages. Its basic design is similar to the framework used by McCreadie et al. for their TREC-TS 2014 submission [23]. However, there are several important differences which affect the design point of view as well as the technical side. The framework of this work for example is more generalized, allowing more sophisticated implementations at any component inside the framework. This later on is especially useful in the last step of the framework to enable a more advanced approach which relies on the idea of aggregated past observations.

The requirements which were defined in the previous section 3.1 were achieved by defining consistent, fixed interfaces between the components as well as forcing the summarization system implementation to be split into different well-defined component implementations. The basic architecture of the framework and its components is shown in figure 2. By using this design, the framework is able to launch multiple instances of each component at the same time, allowing for parallelism. The individual component responsibilities are the following:

- **Filter**: The input for this component is an instance of HtmlDocumentRaw. This data structure, besides some metadata about the document only contains the plain HTML-markup of a news-article webpage. The main responsibility of the Filter is to check the input document against multiple queries. Query/document pairs that are related to each other, which means that the document is about the topic of the query, are emitted and handed over to the next component. The input data structure of the next component, the Document Summarizer, has to be an Arti-

cleDocumentRich. Other than the HTMLDocumentRaw, the ArticleDocumentRich contains only the article text, which is split into sentences containing tokens (words). Therefore, document conversion has to be performed in the Filter as well, which is the secondary goal of the Filter. It is expected that the conversion is required by the filtering process itself, therefore no additional computational effort is required.

- **Document Summarizer**: This component receives a Query/ArticleDocumentRich pair, where the document is preprocessed and considered as relevant for the related query-text. The purpose and responsibility of this component is to extract sentences (DocumentSentence) from the input document which contains important information and summarize the content of the input document. Document Summarizer therefore performs single document summarization with an arbitrary number of output sentences. Each Query/DocumentSentence pair is then passed to the next component. The Document Summarizer is not required to pass any sentences if it detects that there are no sentences that describe the overall topic of the document sufficiently well.

- **Updater**: Receives a Query/DocumentSentence pair as an input, where the sentence is extracted from a document which is considered as relevant to the target-query. The sentence itself therefore contains relevant information that is on-topic. The responsibility of the Updater is to decide whether to output a new update for the summary of the event which is related to the query. This update can either be the input sentence or any other previous sentence that the updater did not emit yet. The most basic case would be to check whether the input sentence contains any new information in comparison to the previous updates of the same query. If so, the sentence is emitted as a new update, otherwise it is discarded. It is important to note that it is intentional to allow the Updater to emit previously seen sentences, because this behavior enables a much broader range of Updater implementations.

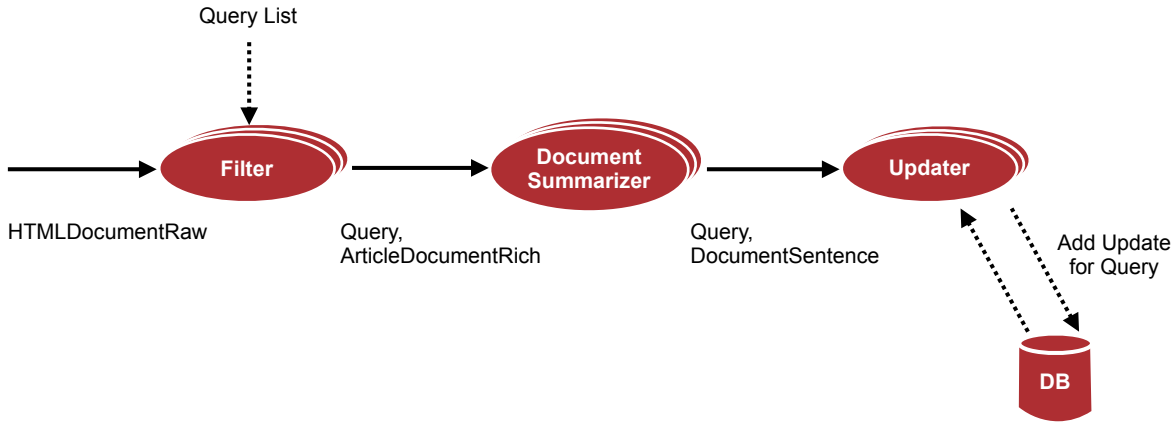

Figure 2: The overall framework-design with all components. Input and output interfaces of each step are well-defined. Multiple instances of each component can be launched which allows for parallel data processing. The database to store updates is not formally part of the framework design to not limit the scope of the summarizer implementations.

As seen before, the framework is designed to be suitable for a wide variety of summarizer implementations. Only interfaces between the individual components are fixed, the implementation itself

and the algorithm choice for filtering, summarizing and updating is completely independent from any framework restrictions.

It is important to note that this framework design has one potential bottleneck, which is the number of queries. In the design showed earlier, each filter action checks a document against all queries. Therefore, the performance is decreasing with an increasing number of queries. A solution for this issue exists and is outlined in appendix B. The architecture described there was not implemented due to an increased complexity, which would not bring any additional benefit for this work other than resolving this bottleneck. The number of queries throughout this work stays on a low level, therefore there are no direct effects that result from this bottleneck.

3.3 Framework Implementation

The framework as described in section 3.2 was implemented in a separate project called ”Summarizer Core” (or ”Core”). Core fully implements the pipeline as shown in figure 2. It transparently provides methods to start multiple instances of each component which can be used to parallelize the overall summarization system. Core furthermore transparently enables stream processing throughout the application without requiring the specific summarizers to implement additional logic for it. This is achieved by using Apache Storm [28] as a basis for Core and abstracting from it, so that the summarization systems based on Core can be implemented without any direct relation to Storm itself. Since Storm is a Java-based project, Core had to be implemented in a JVM-compatible language as well to make full use of it. For this reason, Core was implemented in Scala, which further has the benefit of allowing to write cleaner and less error prone programs compared to Java. Through the usage of Storm, summarization systems based on Core are also resilient and easy to distribute across multiple machines.

The framework implementation provides fixed interfaces for all components of the architecture which have to be used by the individual component implementations of the summarization systems that are based on Core. Through the fixed input and output interfaces, systems automatically have the advantage that individual component implementations can be easily replaced and components of multiple summarization systems are compatible by default. At the same time, the freedom of the component implementations is not restricted by the data structures, because each component can add arbitrary metadata to all output instances which can be accessed in all subsequent components for further analysis. This, together with the abstraction from Storm, also enables to quickly build multiple summarization strategies in different projects that are based on the same structural architecture. Therefore the framework implementation fulfills all requirements listed in section 3.1. Besides the properties mentioned before, it contains the following additional important fragments:

- **Performance Measurement**: When a component processes an item, the execution time is automatically measured within Core. Also, when a new item is emitted from the component, the measurement result is automatically added to the metadata of the output item, which allows to analyze performance data of individual items later on. Measurements are also added to the system datastore which enables the analysis of the overall long-term component performance.

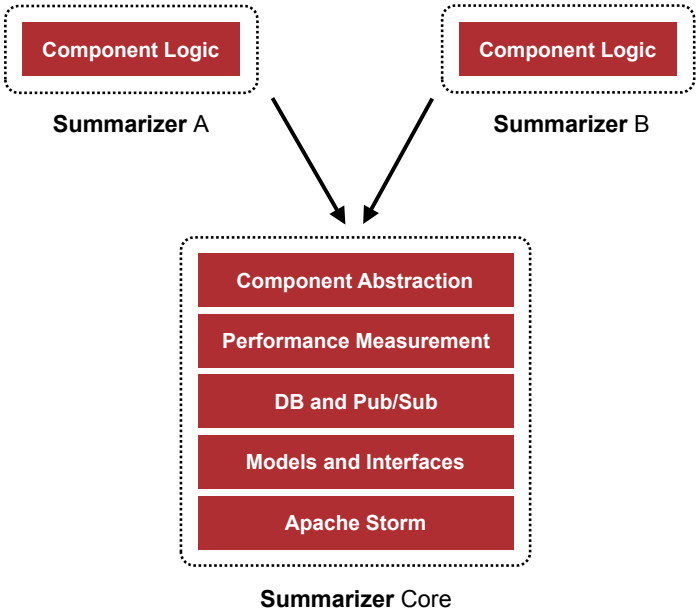

Figure 3: A visualization of different summarizer implementations that are using Core. Core abstracts from the underlying technological details, therefore individual summarizer implementations include implementations for the component logic only.

- **Data Storage and Pub/Sub**: Interfaces for data storage components were defined to enable easy replacement of the database as well as a pub/sub system. Core also brings implementations of all these interfaces for the redis database and message broker [2].
- **Caching Utilities**: In-memory caching is an effective method of reducing IO-operations when frequent access to certain data items is required. Core provides several caching utilities to tune performance and prevent component instances from blocking due to slow IO-operations.

As an extension to the overall framework implementation, a summarization frontend was created, allowing users and developers to inspect outputs of the summarization system based on Core in real-time while the summarization process is still running. The frontend shows the current system status, recent performance measures and a list of all target queries with the related summary updates. Screenshots of the user interface are included in appendix A. The frontend also includes metadata inspection which allows to view all metadata entries that were added to an update throughout the summarization process. This feature is especially useful for debugging since every component can add arbitrary metadata to each item. The summarization frontend therefore provides an efficient and time-saving way to quickly assess system outputs. It is implemented in a separate project which is compatible to all summarization systems based on Core. A more detailed description of the summarization frontend can be found in appendix A.

As a conclusion, the framework implementation provides a solid base for potential summarization systems and contains powerful abstractions from technical details that enable valuable properties such as stream processing and parallelism. In figure 3, two summarization systems based on core are visualized. The required implementation logic for the summarization systems is minimized through

[2] In-memory key-value store; http://redis.io/

the usage of Core. They do not contain any unnecessary boilerplate code which results in improved productivity and faster prototyping.

3.4 Document Corpus

The overall goal of the framework architecture and summarization systems is the process of summarizing a large amount of news-articles in real-time. Unfortunately there is no large real-time data source containing news-articles freely available on the web. Therefore, such a source is simulated throughout this work by traversing a big corpus of news-articles in temporal order. Two different alternative corpora were considered and investigated:

- **Common Crawl** [3]: This corpus by the Common Crawl Foundation contains an extremely large amount of web page crawls (volume: petabytes). It is a general-purpose corpus which contains content of different languages and different types (html, pdf, xml, ...). Crawls are not sorted by timestamp in a fine-grained way and the analysis of some historic random samples of news-articles in the web revealed that timestamps are often different from the article publish date. Another downside is that these crawls are not categorized by web-page type (social, blog, news, ...).
- **TREC-TS 2014 corpus**: The TREC-TS 2014 challenge [2] provided a corpus for its participants containing a large amount of documents (~50 million; 550 GB). Documents were crawled from multiple sources (news, social, blog) with nearly all documents being in English. All documents which were published in the same hourly time-window are grouped together, therefore a partial sort order is available. Document categorization by source type is available. The corpus furthermore contains rich NLP tagging information for all documents.

Different advantages of the TREC-TS 2014 corpus were the reason to choose this corpus as data source for all simulations and tests in this work. The crucial benefit was the categorization and temporal sort order of this corpus which does not exist in the same extend on Common Crawl. This choice has further advantages, because the TREC-TS 2014 challenge focusses on similar goals compared to this work. By using the TREC-TS 2014 corpus all the additional resources from the challenge could be used, for example test-events and evaluation metrics. To allow the simulation of a stream of incoming news-articles based on the chosen corpus, different preprocessing steps were required. The result is a new corpus that can be processed from start to end without requiring any further filtering or sorting. The following actions were performed to create the new simulation corpus:

- **Removal of non-news-article documents**: Through this removal-step, all documents which were not crawled from news-websites were removed. The resulting corpus therefore does not contain any documents from either social media or forums, which are usually radically different from news-articles.
- **Removal of non-required information**. Documents in the TREC-TS 2014 corpus were preprocessed by the corpus creators, which means that besides the HTML-markup, the extracted webpage text is available with sentence splitting and tokenization already performed. Furthermore, named entity recognition and part of speech tagging information also are available in the TREC-TS 2014 corpus. Since the summarization systems based on the framework presented

[3] http://commoncrawl.org/

in section 3.2 only use the HTML-markup as input, all other information was removed[4]. For evaluation purposes, the original sentences splitting positions and token positions were kept.

- **Establishment of a complete sort order**. All documents inside an hour grouping were sorted by timestamp. The resulting corpus therefore contains a complete sort order. This allows to process all documents of the corpus in the order in which they were published.

The resulting corpus contains 6,488,989 documents between 12/2011 and 04/2013. There are 15 test-events which partially overlap in their timeframe. For dates without an active event, the TREC-TS 2014 corpus does not contain any documents, therefore the resulting corpus for this work only contains documents at dates where one of the 15 events was still active. With this corpus, it makes sense to only use the events of TREC-TS 2014.

[4] Using the preprocessed information about sentence splittings, named entity recognition etc. would result in non-realistic performance measures, because in a real-world environment data would also be raw.

4 Summarizer Implementations

In this section, the implementation of different summarization systems is described, which is the main part of this work. With the architecture design and Core implementation as described before in section 3, the technical foundation is the same for all summarization systems of this work. Technical details in this section are therefore limited to the specific behavior of the particular component implementation. The first system which was developed as part of this work is a simple approach which relies on basic algorithms only. This implementation was chosen to be the baseline throughout this work. The goal for subsequent implementations and system changes was to improve upon this approach.

During development, four different events from the set of the 15 test-events of TREC-TS 2014 were used for validation and optimization purposes. The events were: Boston Marathon Bombing, 2012 Afghanistan Quran burning protests, 2013 Eastern Australia floods (or "Queensland Floods") and Costa Concordia disaster and recovery. Because of the usage during development, evaluation results for these events are listed separately in the evaluation section and do not contribute to the final scores.

This section is structured as follows: First, the baseline implementation is described and all component implementations and algorithms are presented. In the subsequent (sub)sections, improved systems and changes are shown. The last (sub)section contains an overview of all developed systems which are then tested in the evaluation.

4.1 Baseline

The first summarization system that was created in this work only relies on simple techniques which do not require much time to compute and are fully parallelizable. This approach was then declared as the baseline-system ("Baseline") for this work on which improvements were made upon. Even though Baseline is a basic and efficient system, it also contains some advanced concepts. In this section, the specific implementations of the individual components of Baseline are described and chosen algorithms and configurations are shown.

4.1.1 Filter

The Filter is the first component in the framework architecture. Its inputs are streaming HtmlDocumentRaw instances, which contain the HTML-markup of a news-article webpage. Its outputs are streaming Query/ArticleDocumentRich instances, which contain the article text in a tokenized and sentence-split form. The purpose of this component is to discard articles which are irrelevant for all target queries and to pass Query/ArticleDocumentRich pairs to the next component for all documents that are relevant for a target query. The secondary goal is to create the processed document representation ArticleDocumentRich from the HTML-markup of the HtmlDocumentRaw instance.

To successfully check that some document is relevant for a query, the article text has to be extracted from the document first. In this case, the input document consists of HTML-markup only. To extract the article text from the HTML-markup, the naive way would be to simply strip away any HTML-tags and to use the resulting text as the article text. However, this approach has different obvious flaws.

Usually there is a lot of boilerplate content on news-websites, for example a comments section, the navigation bar and often boxes linking to related articles. Using the naive approach, the extracted article text would also contain text snippets from these parts, which may be completely unrelated to the article topic and may be of poor quality (comments section). As a consequence, the first sentences of the so extracted article text would not necessarily be topic-related. For this reason, another approach was used for the component implementation which only extracts the actual article text and discards any boilerplate content. This has the advantage that from beginning to the end of the extracted article text, no low-quality boilerplate sentences are included and the start of the extracted article corresponds to the start of the actual article content. To extract the article text from the HTML-markup the Filter implementation relies on the boilerpipe library, which is based on shallow text features to detect boilerplate content [18]. This library performed reasonably fast in some preliminary tests.

After boilerplate removal and article-text extraction, the next step is to tokenize the article-text and to perform sentence splitting. Filter for these tasks relies on the 3rd party library Stanford CoreNLP Toolkit [22]. After sentence splitting, additional information is added to the sentence, such as a sentence ID, information about the source document (timestamp, source ID), the original sentence text, and a list of tokens with stop-words removed. Each token contains the original text string and a token value, which is a lowercase stemmed (Porter) version of the token-text. By using the list of sentences, the final ArticleDocumentRich is constructed.

With the ArticleDocumentRich instance, the actual filtering can be performed. The simplest way would be to check if all query tokens are included in the article (stemmed). This approach quickly was discarded after initial testing, because too much topic-unrelated documents were considered as relevant for the query. The observation was that a lot of articles contain cross-references to other articles in the middle or at the end of the document. Therefore a much more restrictive filtering approach was necessary. An article is discarded without further analysis if it contains less than 10 sentences, indicating that this is not a complete news article but rather an extract. If it contains enough sentences, the article is considered as relevant to a query if it contains all query terms (stemmed) in the first 5 sentences and at least twice in the whole document. The motivation behind this approach is the assumption that the most important facts are often described at the beginning of an article (headline, subline, abstract), and also are mentioned in the rest of the article (the actual content). Through this double-checking, articles which are off-topic but contain a reference to an on-topic article are not considered as relevant to the target-query. Compared to the simple approach, the number of relevant articles for the event Boston Marathon Bombing significantly decreased from 30,307 to 8,423. Through this reduction, the other component implementations could be implemented without any removal techniques to filter irrelevant content.

A note on computational performance: No computational complex algorithms or IO operations are used in the Filter. Article extraction using boilerpipe requires less than 10ms on an average computer and an average HTML document. The same applies to sentence splitting and tokenization using Stanford CoreNLP Toolkit, which is also faster than 10ms on an average computer and an average document. To provide an upper-bound for computational performance, only the first 3 million characters of the HTML-markup are considered for further processing. The upper-bound only affects individual outliers, because it is unlikely for HTML-markup of mainstream news-articles to contain more than 3 million characters.

4.1.2 Document Summarizer

As the second component of the architecture, the Document Summarizer receives the output of the Filter as an input. This is a Query / ArticleDocumentRich pair. The task of this component is to perform single document summarization on the article in regard to the target query. Since this is not the traditional single document summarization but rather a part of a bigger summarization system, other objectives apply. It is not required to create a comprehensive and complete summary, however relevant and topic-descriptive sentences should be emitted. For the implementation of Baseline a twofold process was chosen. First, a basic heuristic is applied which removes sentences that do not fulfill the following requirements:

- **Length restrictions:** Number of tokens without stop words must be between 7 and 30. This ensures that sentences which are likely to be meaningless out of context (less than 7 tokens) or sentences that contain too much information (more than 30 token) are not included in the summary.
- **Named entity heuristic:** The sentence must contain at least one token that starts with a capital letter with the first token and all query tokens being excluded. This is a basic heuristic for a named-entity in the English language, since they usually start with a capital letter.

Similar heuristics were also used in [23].

The actual single document summarization is then performed using the remaining sentences as an input. A greedy sentence selection method was chosen for summarization. Maximal marginal relevance (MMR) [4] selects a sentence with the goal to maximize the similarity to the query and to minimize the similarity to all previously chosen sentences of the same document:

$$MMR = argmax_{S_i \in S_{all} \setminus S_{chosen}} \left(\lambda \, sim(S_i, q) \; - \; (1 - \lambda) max_{S_j \in S_{chosen}} sim(S_i, S_j) \right) \tag{1}$$

In the implementation of Baseline, MMR chooses only two sentences with a λ value of 0.5, which does not privilege one of the two factors. Similarity is computed using a vector space model and cosine similarity with the sentence being represented as a bag-of-words. The weight for all words is computed using a score similar to TF*IDF. Since all input is streaming data, it would be expensive to compute inverse document frequencies for all words whenever a new document arrives. Therefore a static background corpus, the unigram version of web1t, was chosen instead. Web1t was created by Google in 2006 and contains term counts from a huge amount of crawled web pages. Klein et al. showed, that there is a strong correlation between document frequencies and term counts [17]. This means that the usage of term counts from a source like web1t is similar to using the actual document frequencies. The weight for a word therefore is computed with TF*ITC with ITC being the inverse term count which approximates the inverse document frequency, based on the static background corpus web1t. The actual similarity measure operates on lowercase stemmed values with stop words excluded. Since term counts from web1t do not represent lowercase stemmed values, the term count TC of a stemmed value has to be inferred from its original text. In the target sentences, different original texts for the same stemmed words may be included. To resolve this issue, the term count for a stemmed lowercase word is retrieved by averaging all term counts of the original texts for words that have the same stem in the target sentences. The term count of a lowercase stemmed value s is computed as follows:

$$TC(s) = \frac{1}{|T_s|} \sum_{t \in T_s} TC_{web1t}(t) \tag{2}$$

With T_s being a set of words (original texts) that occur in the target sentence which are stemmed to s. Other possibilities of retrieving term counts for lowercase stemmed values are possible, for example the minimum or maximum value of the term counts from the original values.

After MMR extracted the target sentences that represent the single-document summary (two sentences), the Document Summarizer removes sentences with a MMR score less or equal to zero. Such sentences either have no query similarity or the similarity to the other selected sentence is higher than the query similarity. The number of output sentences therefore is not fixed, it possible for Document Summarizer to output no sentences. All sentences that remain after this filtering step are passed to the next component for summary updating.

A note on computational performance: Usually MMR is considered as computationally expensive, since it requires one similarity comparison for each sentence to the query as well as a similarity comparison to all chosen sentences. The more sentences there are and the more sentences to choose, the more similarity comparisons are required. In this implementation, the number of input sentences was capped after 50 sentences and the number of sentences to select is low as well (two sentences). The maximum number of similarity calculations therefore is limited. On the other hand, the similarity computation using the cosine similarity is computationally cheap. The top 20,000 words from web1t were pre-loaded and cached. Throughout the lifetime of the component instance, the cache automatically detects the 20,000 most recently used words and holds them in memory. Therefore, IO-operations are minimized which leads to a fast processing time for all items.

4.1.3 Updater

The last component of the framework is the Updater. It receives DocumentSentence/Query pairs as an input, where the sentence is a part of the single document summary of a document related to the target-query. The responsibility of the Updater is to decide whether a new update for the summary of the query should be emitted, based on the current input. For Baseline, the most simple approach was chosen. Every input sentence is emitted as an update of the summary related to the target-query if the sentence does not duplicate content of recent summary updates. Duplicate detection is done by calculating the similarity between the input sentence to the most recent summary updates, using the same similarity measure as in Document Summarizer. If a similarity value exceeds a certain threshold value, no update is emitted. If all similarity values are below the threshold, the input sentence is added as a new update for the summary of the input query. During development, the threshold was determined experimentally based on outputs from the events used during development.

This approach is effective and does not create summaries which are too long, because the implementation of the Filter component of Baseline is very restrictive and does not pass a lot of documents. With other, less restrictive filtering methods, more sophisticated Updater implementations may be necessary.

A note on computational performance: Even though the similarity measure is computationally cheap and, as in the Document Summarizer, uses the same caching mechanism for ITC scores, the number of similarity comparisons had to be limited to avoid a decreasing performance over time when more updates are included in the summary. For this reason, the similarity is only computed on the 500 most recent updates of the target-query summary. To prevent unnecessary database operations, each Updater instance caches these updates. Whenever a new update is emitted, a message is published to the pub/sub system, telling other instances of the updater to add the new update to its cache. In the

current implementation, each updater stores updates for all queries. This means that the total number of queries is a potential bottleneck. However, a simple solution would be to route DocumentSentence/Query pairs of the same target-query to the same Updater instance. Therefore, each Updater instance only has to cache the updates for the events or queries it is responsible for.

4.1.4 Confidence Scores

Confidence scores for updates describe the confidence of the summarization system that an update is beneficial for its summary. Such values could be used in real-life systems to visually highlight important updates and to display less important updates in an unobtrusive way. For evaluation purposes, these can be important values as well, especially for an evaluation that relies on manual annotations. Since it would be infeasible to manually annotate a large summary consisting of multiple hundrets of updates, confidence scores can help to efficiently reduce the summary size by selecting only the top-updates for the evaluation. In the TREC-TS 2014 evaluation for example, confidence scores were used to extract the top-60 updates of each summary which were then processed by human annotators. Because in this work the evaluation metrics of TREC-TS 2014 are used for evaluation, a confidence score had to be calculated in all summarization systems.

For Baseline, several potential values were investigated to be used as a basis for confidence scores, for example the similarity against the most recent updates or the score calculated during single document summarization. However, none of these scores could successfully establish an order of updates with the most beneficial updates having the highest confidence scores. Therefore, a simple but intuitive method was used to calculate confidence scores for updates. It is based on the assumption that news-articles which are published at the beginning of an event contain more relevant information than news-articles that were created after the event started, because opinion-related content is expected to grow over time. The function to calculate confidence values therefore is monotonically decreasing with the number of updates that were already emitted:

$$confidence(u) := \frac{1}{n} \tag{3}$$

With u being the update that should be emitted and n being the size of the summary (number of updates).

4.2 Multiple Sources Updater

By running the implemented Baseline summarizer against the development events, interesting results were obtained. For high-impact events with a broad news-coverage such as *Boston Marathon Bombing* and *Costa Concordia*, Baseline emits too much updates. The overall number of updates for these events is very high (*Boston Marathon Bombing*: 854) and may exceed the amount of updates which a real-life user may want to read. It is assumed that these events can fully be summarized in 100 sentences or less with all major information being included, because the related Wikipedia articles are of a similar size. The high number of updates for Baseline is not a result of a wrong threshold in the updater which prevents it from discarding similar sentences. The reason is that after a short period of time for such high-impact events, the number of opinion- and gossip-related news-articles as well as portraits increases significantly with each telling a different story and containing different information. Updates

based on such individual opinionated stories are not expected to benefit the overall summary. An improvement over Baseline therefore could be to prevent the summarizer from emitting such updates and thereby improve the overall summary quality. This is the goal of the Multiple Sources Updater.

The main assumption behind the Multiple Sources Updater is that important and urgent information is likely to be reported by multiple sources or mentioned in multiple articles with a low temporal distance. On the other hand, irrelevant and wrong information or opinion-related contents are expected to be reported only by a single source. Furthermore it is expected that the reverse assumption is true as well, meaning that some information that is reported by multiple sources is likely to be relevant. An approach implicitly utilizing this would be a graph-based algorithm applied on a large amount of news articles over a certain time-window, with the goal to find centroid sentences based on sentence-similarity. However such an approach is not real-time in respect to the requirements of this work, since it only decides once in its time window (e.g. one hour) on which sentences to emit. The Multiple Sources Updater therefore uses another, different approach that satisfies the requirements imposed on the overall framework introduced in section 3.1.

The most important concept of the Multiple Sources Updater is the candidates list. It contains all recent sentences that were not emitted as an update, because they contained information that was not reported by multiple sources. Sentences of this list are used to determine if there are recent sentences with similar content, which means that multiple sources report that information. When a new DocumentSentence/Query pair arrives at the Updater component, the following steps are executed.

1. **Classifier**: Discard all irrelevant sentences. Although the Filter component in the Baseline implementation tries to extract the article text without boilerplate content, some sentences arriving at the Updater component are clearly irrelevant. Such sentences can be lists of the news-article authors alongside the article date, sentences containing boilerplate content that was not removed (comments section) or broken sentences which were split at a wrong position. A naive bayes classifier was trained to classify sentences as clearly irrelevant/not irrelevant based on multiple features like the frequency of uppercase letters of the sentence, the frequency of non alphabetic characters or the absence of crisis vocabulary. The classifier was tuned to minimize false positives with the goal reduce the amount of wrongly discarded sentences. Training data was obtained through manual labeling of sentence outputs from summarization systems that participated in the older TREC-TS 2013 challenge.

2. **Similarity**: Check that the target sentence contains new information. This is equal to the approach of the Updater in Baseline where any input sentences are checked against the most recent updates of the related summary. If any similarity value between the target sentence and an update exceeds a certain threshold, the target sentence is discarded. Similarity is computed as in Baseline.

3. **Multiple Sources**: Check if the target sentence contains information that was reported multiple times (~by multiple sources). If there are N sentences in the candidates list which are similar to the target sentence, one of those sentences is emitted as an update and the sentences are removed from the candidates list. If there are less than N similar sentences in the candidates list, the target sentence is added to the candidates list. Initial test showed that only values N=2 and N=3 produce good results, even for the development events with a broad news-coverage, such as Boston Marathon Bombing or Costa Concordia. The similarity measure used in this step

is equal to the similarity measure used in the previous step. Thresholds for determining if two updates are similar were found experimentally.

The overall structure of the approach is visualized in figure 4.

After a target sentence passed all stages of the Updater described before, one of the N similar candidates is chosen to be emitted as an update. The basic idea is to include any information in the update which is reported in the majority of the candidates. This is the basic motivation of sentence fusion [3], which is an approach from generative summarization. Due to comparison reasons, this work can only rely on extractive approaches. Therefore one of the candidate sentences has to be selected as having the biggest information overlap to all other candidates. This candidate is then emitted as an update for the summary. In this implementation, the overlap is measured by sentence similarity. The similarity measure is the same cosine similarity used in the Document Summarizer. The candidate which has the highest average similarity s_{avg} to all other candidates is chosen to be the update. For $N \leq 2$ all candidates have the same value for s_{avg}, therefore in this case a simple heuristic is applied which chooses the sentence that contains the most digits. The motivation behin this approach is the assumption that for similar sentences, the sentence that contains most digits is also more precise [5].

When the Updater of the Baseline system is replaced with the Multiple Sources Updater, the number of output sentences of the MMR-based Document Summarizer can be increased, because the Multiple Sources Updater is more restrictive in emitting updates. The multiple sources approach can also benefit from receiving more input sentences, because the candidates list in this case is filled with more sentences more quickly. First tests with the Multiple Sources Updater showed that the best effect is achieved with four to five output sentences for the Document Summarizer. Tests also showed that the amount of updates with the Multiple Sources Updater and the overall summarization quality is heavily influenced by the choice of N and the overall news-coverage of the target-event. This observation also was validated in the overall (secondary) evaluation which is presented in section 5. The need to optimize the configuration of the Multiple Sources Updater for the target-events was the motivation for the adaptive approach.

4.2.1 Confidence Scores

With the Multiple Sources Updater and its candidate selection there are multiple values that can be used as a base for a confidence value. Other than for the Baseline approach, several values were identified which are likely to correlate with the importance of an update. These are:

- **Candidate similarity**. A high similarity means that the information was reported multiple times with no significant deviation, whereas a low similarity means that the information was reported multiple times with lower information overlap. A higher similarity therefore should lead to a higher confidence score.
- **Temporal closeness**. A small difference in the timestamps of the similar candidates means that the information was reported by multiple sources at nearly the same time. This may indicate breaking-news or new important information. Bigger differences in the timestamps on the other hand indicate less urgent information. A smaller difference in timestamps between similar candidates therefore should lead to a higher confidence score.

[5] There are often similar sentences like ”Many people were on still on board.” and ”23 people were still on board.”. The last sentence is preferred for the summary, because it is more precise since it contains the exact number of people.

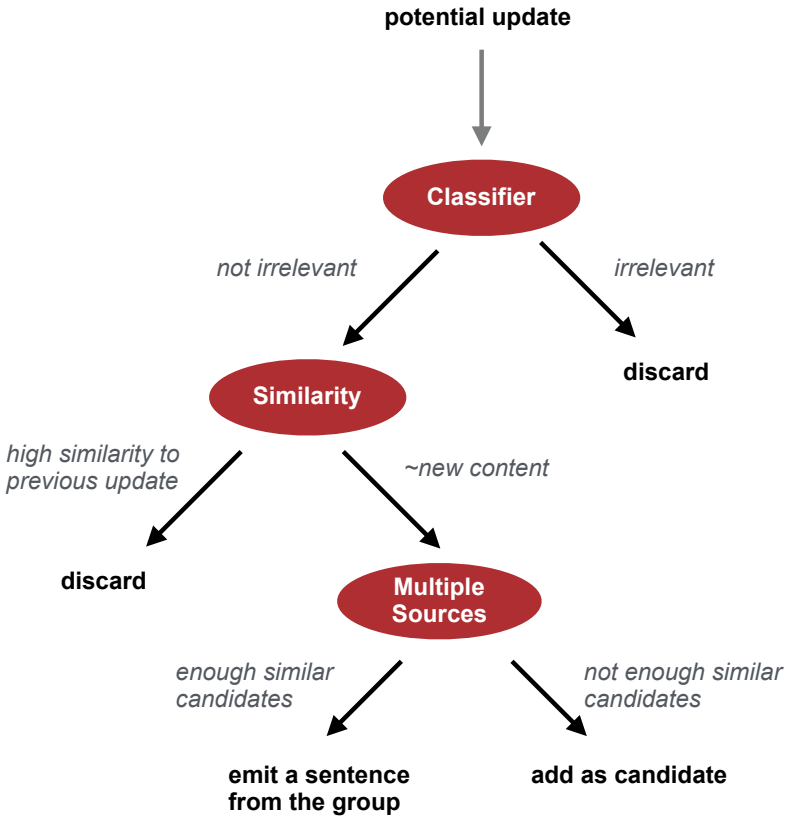

Figure 4: An overview of the individual steps implemented in the Multiple Sources Updater. A classifier discards all input sentences that are clearly irrelevant. The similarity check against the most recent updates of the same summary discards all sentences which do not contain novel information. If an input sentence passes these components, the actual multiple sources check is performed.

- **Number of candidates**. A higher number of required candidates means that an information was reported by more sources, whereas a lower number of required candidates means that an information was reported by less sources. A higher number of required candidates therefore should lead to a higher confidence score.

The final confidence score therefore consists of three components:

$$confidence := similarityComponent * timelinessComponent * requiredCandidatesComponent \quad (4)$$

Where the similarity component is computed by averaging the similarity values from the selected candidate u to all other candidates of C:

$$similarityComponent := \frac{1}{|C \setminus u|} \sum_{c \in C \setminus u} Similarity(i,c) \quad (5)$$

The timeliness component describes how close the timestamps T of the candidates C lie together in respect to some reference interval MaxRange (~24 hours). If the range of the timestamps exceeds MaxRange, the score will be zero and therefore the overall confidence score will be zero as well.

Since confidence scores of zero indicate that that there is no certainty that the corresponding update is beneficial for the summary, these updates are discarded.

$$timelinessComponent := \frac{max\left(0, MaxRange - max(T) + min(T)\right)}{MaxRange} \tag{6}$$

The required candidates component finally adds an additional bonus for every required candidate:

$$requiredCandidatesComponent := 1 + \frac{N}{5} \tag{7}$$

With this relatively sophisticated approach of calculating confidence scores, it is possible to incorporate different aspects that are unique to the Multiple Sources Updater. Compared to the approach of Baseline, which uses a monotonically decreasing function to assign lower confidence values to updates when the summary increases, the approach of the Multiple Sources Updater is more intuitive. It further has the advantage that updates which are included in the summary long after the event started can be scored with a high confidence value. This may be required for events which end with an important sub-event, such as the capturing of the primary suspect in a criminal investigation.

4.3 Adaptive Algorithm

During the development phase, the Multiple Sources Updater was tested against the development events and results were analyzed to find possible weaknesses and to identify potential further improvements. Output summaries were tested against the same evaluation metrics that were used in the overall evaluation which is presented in section 5. Results of these development tests are presented in textual form only, to show the motivation behind certain changes. Observations and results during the development phase especially showed that it is critical to choose the right group size for the Multiple Sources Updater for each event to achieve a good result. Since events with a broad news-coverage have many news-sources reporting about it, a larger group size leads to better results. For events with a thinner news-coverage, a large group size may prevent the summarization system from emitting a reasonably high amount of updates. In such cases a smaller group size produces better overall results. An overview of results from multiple group sizes are listed in table 1. It shows that for the event Boston Marathon Bombing a group size of three candidates produces best results, whereas for an event with a less broad news coverage in the English-speaking area, the smaller group size of two candidates is the better choice.

Even if the summary for the event Afghanistan Quran Burnings with the smaller group size was better than with the bigger group size, it was clear that other improvements were required to produce a great summary. At the same time, the expectation was that many unseen events could be similar in news-coverage to the Afghanistan Quran Burnings, because any large large geographical distance from the USA potentially leads to a thinner news-coverage in U.S.-media [5]. This especially motivated the further changes. The biggest problem with the Afghanistan Quran Burnings results were the low amount of updates and the low coverage of the event topic, which was acceptable but not good. Since there was no increase of non-important updates with the smaller group size, it was expected that further changes that lead to a lager amount of updates could be made without a major drawback. Besides the Updater itself, one other key component which is responsible for the amount of updates

Table 1: A comparison of the results for the Multiple Sources Updater with different group sizes. Cells with a green background color indicate the better choice of the group size for the target-event.

Event	Group Size N=3	Group Size N=2
Boston Marathon Bombing	• Reasonable amount of updates (72) • Good coverage of event topic • Few non-important updates	• A lot of updates (237) • Good coverage of event topic • Many non-important updates
Afghanistan Quran Burnings	• Extremely few updates (6) • Bad coverage of event topic • No non-important updates	• Few amount of updates (13) • Acceptable coverage of event topic • No non-important updates

is the Filter, which was implemented to filter documents in a restrictive fashion (compare section 4.1.1). For the Afghanistan Quran Burnings event, it considers 279 documents from a total of 481,278 documents, that are in the corpus between the event-start and the event-end, as relevant for the query. This underlines that a change in the Filter component is required for events with a thin news-coverage. In comparison, the restrictive Filter considers 8,423 of 330,204 documents as relevant for the target-query of Boston Marathon Bombing.

To increase the number of documents which are passed to the next component for events with a thin news-coverage, a less restrictive Filter implementation was created. It is based on the Baseline Filter, with the following changes:

- Discard the article if it contains less than 5 sentences (instead of 10).
- The article is relevant for a query if it contains all query terms in the first 20 sentences (instead of 5) and the document contains at least one query token twice (instead of all tokens).

To further support the Multiple Sources Updater and to emit more updates for the less restrictive approach, the Document Summarizer outputs five (instead of four) sentences and the thresholds on the Multiple Sources Updater are slightly lowered. These changes were validated experimentally and showed the desired results. A comparison of the less restrictive approach against the standard approach is shown in table 2. Results show that for both events, the Boston Marathon Bombing and the Afghanistan Quran Burnings, the amount of updates emitted is increased significantly with the less restrictive approach. Whereas this has negative effects for the results of Boston Marathon Bombing, the summary of Afghanistan Quran Burnings benefits from the higher amount of updates. For the other development events, Costa Concordia and Queensland Floods, results showed that there was no significant benefit for both when using the less restrictive approach, because the number of updates using the standard approach already were acceptable. At the start of Queensland Floods however, since there was only small news coverage at the beginning of the event, results could potentially benefit from using the less restrictive approach at the beginning and the normal approach at the end.

The next logical step was to combine the different individual configurations into a combined system that is able to utilize the advantages from all configurations. The idea was to automatically select the configuration which is best suited for an event to create the best possible summary. One approach would be to choose the configuration for an event upfront on event-start based on the query string. This would be easy to implement, but the query string is not expected to reveal lot of information about the news-coverage or the importance of the event. Therefore a dynamic approach was chosen in this

Table 2: A comparison of the less restrictive approach and the standard approach of the Multiple Sources Updater with the same group size of N=2. Cells with a green background color indicate a good choice for the target-event.

Event	Group Size N=2	Less Restrictive Approach, Group Size N=2
Boston Marathon Bombing	• A lot of updates (237) • Good coverage of event topic • Many non-important updates	• Too much updates (1735) • Good coverage of event topic • Most updates not important
Afghanistan Quran Burnings	• Few amount of updates (13) • Acceptable coverage of event topic • No non-important updates	• Reasonable amount of updates (66) • Good coverage of event topic • Few non-important updates

work which allows to select and change configurations on runtime, based on the current importance of the event. Importance is measured by the news-coverage which is indicated by the amount of relevant documents that pass the Filter for the event-related query. This approach requires more implementation effort, because the streaming data has to be analyzed and the changes have to fit into the framework with its concurrency requirements. However, the advantages were expected to be significant, because this also allows to change configurations during the event itself when important changes are detected.

4.3.1 Configuration and Algorithm Choice

In this section, the adaptive approach is described in detail together with the overall configuration selection method and different event measurements over time, which show the challenges of how to process the event news-coverage. The first important step towards the adaptive approach is to define all possible configurations. These are listed in the following table 3.

Table 3: The different configurations which are used in the adaptive approach.

Name	Filter	Document Summarizer	Updater
B	Baseline Filter	MMR-based summarizer which outputs max. 4 sentences	Multiple Sources Updater with N=3
A	Baseline Filter	MMR-based summarizer which outputs max. 4 sentences	Multiple Sources Updater with N=2
A+	Less restrictive Filter	MMR-based summarizer which outputs max. 5 sentences	Multiple Sources Updater with N=2 and lower thresholds

To select the best configuration choice, the current popularity of an event is analyzed based on the detected news coverage of that event. The detected news-coverage is measured in documents per hour. This is the amount of documents the Filter component judges as relevant for the event-query. Since in the simulation of the news-stream one hour of the simulation corpus is processed in a couple of seconds

by the summarization system, documents per hour refers to one hour in the simulation corpus. To be consistent, the Filter used to detect the news-coverage always is the default Filter from the Baseline implementation. This prevents oscillating configuration switches, because the configuration choice itself can not influence the news-coverage detection process.

Since the news-coverage in hourly blocks is changing a lot, moving averages over several past hours were used for any further analysis. This smoothens the overall curve which is required for the analysis of the news-coverage in order prevent frequent configuration switches. In news-coverage detection, two moving averages are used, a short-term moving average (MA6) and a longer-term moving average (MA24) over 6 and 24 hours. A comparison of the raw (hourly) relevant document counts and the MA6 and MA24 is visualized in figure 5.

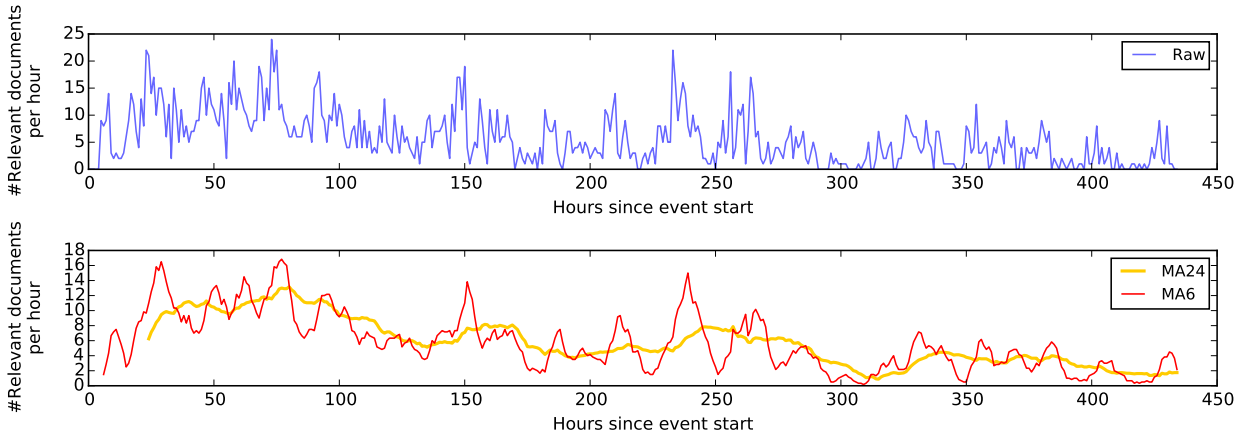

Figure 5: A comparison of the relevant documents of the event Costa Concordia in raw/hourly form and the moving averages MA6 and MA24.

The graph clearly shows that there are many situations where the number of relevant documents in two subsequent hourly frames differ a lot. Such rapidly changing data can not be reliably used for a direct analysis. This is one of the reasons why the two different moving averages are used in the analysis. The general requirements for the adaptive approach were defined as the following:

- Up-spikes in news-coverage should be detected as fast as possible. Otherwise the less restrictive approach would be active during times where a broad news-coverage is present which would lead to a large amount of updates that are potentially non-relevant or originate from opinion-related content.
- Sudden and temporary drops in news-coverage should be ignored if the previous level is recovered soon. At night for example, even during times with broad news-coverage, the amount of published news-articles decreases significantly. The amount of relevant content also decreases at such times, therefore no action has to be taken to increase the amount of updates or to choose a less restrictive summarization approach.
- Frequent configuration switching should be avoided. The number of configuration switches over the whole event should be small to enable a meaningful analysis of the results.

The requirements imply that it should be more easy to switch from a less restrictive approach to a more restrictive approach where the opposite should be harder. This can easily be done using MA6 and MA24 with MA6 being used to detect up-spikes in news-coverage and MA6 together with MA24

being used to detect a lowering of the news-coverage. The overall system is threshold-based, meaning a switch is performed when certain values exceed or fall below fixed thresholds. Thresholds were defined experimentally, based on the following specific requirements that were obtained through the observations of the individual configurations for the development events:

- For Boston Marathon Bombing the adaptive approach should select the configuration B as fast as possible and perform no additional change afterwards. This is motivated by earlier observations where the Multiple Sources Updater with N=3 achieved by far the best results for this event.
- For Afghanistan Quran Burnings the adaptive approach should stay with configuration A+ as long as possible. As before, this is motivated by the observations that showed the superiority of the less restrictive approach for this event.
- For Costa Concordia the adaptive approach should choose configuration B for the beginning of the event and configurations A or B for the rest. This is motivated by the observation that the beginning and end of the event differ in the news-coverage (compare figure 6).

These requirements are fully satisfied by the implementation of the adaptive approach with its specific ruleset. The first important rule is the configuration that an event starts with. The best configuration to start with, considering the requirements mentioned earlier is A+, because this is the least restrictive configuration which means that a change to all other configurations can be done in the least amount of time. Furthermore at the beginning of an event it is expected that most information is relevant (compare section 4.1.4), the choice to start with the least restrictive configuration therefore should have minimal negative effects. The configuration switches are then performed by applying the rules that are listed in table 4. It is important to mention that the rules for the direction from A+ to B are very different to the rules for the direction from B to A+. This prevents the system from frequent configuration switching.

Table 4: Rules for the adaptive approach to switch configurations based on the values of the moving averages MA6 and MA24.

Old Conf.	Condition	New Conf.
A+	$MA6 > 6$	A
A	$MA6 > 14$	B
B	$MA24 < 6$ and $MA6 < 6$	A
A	$MA24 < 1$ and $MA6 < 1$	A+

In figures 6 and 7, the configuration switches of the most interesting development events in terms of the adaptive configuration selection are shown, which are Costa Concordia and Boston Marathon Bombing. In the event Costa Concordia the adaptive approach switches configurations most often which is in line with the requirements, since at the beginning B is chosen and for the rest A or B is chosen and no extreme amount of switching takes place. For Boston Marathon Bombing the adaptive approach quickly switches to B with no further changes throughout the event. Visualizations of algorithm switches for the other development events, Quran Burning Protests and Queensland Floods, are included in appendix C. In the Quran Burning Protests no configuration switches are performed, which means that the adaptive approach uses A+ throughout the whole event. For the event Queensland Floods the situation is similar, with the difference that at the end there is a large increase of news-articles and the adaptive approach switches from A+ to A and then to B.

Since all the requirements listed before are satisfied with the implemented rule set, the adaptive approach should be able to combine the advantages of the individual configurations into a single system. Initial testing showed that the adaptive approach can in fact achieve even better result scores than

individual configurations. This is done by choosing different configurations for different parts of the event to achieve the optimal overall summarization result. Actual result scores of the final evaluation are listed in section 5.

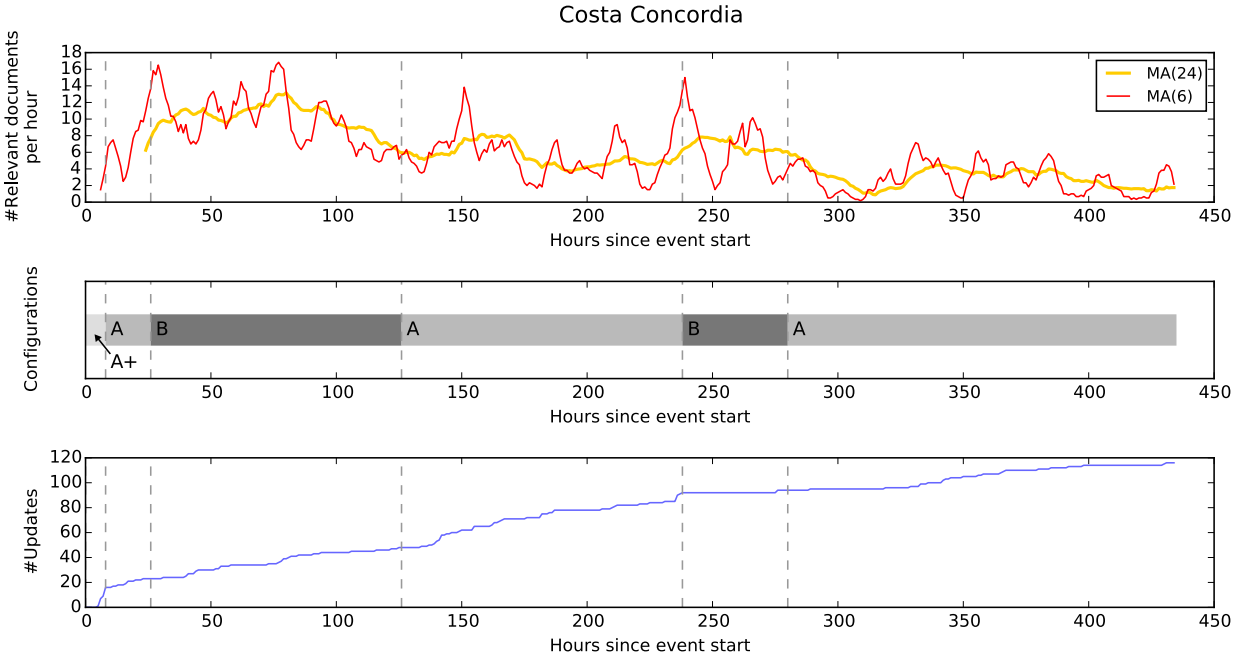

Figure 6: Configuration switches of the adaptive approach with the event Costa Concordia. After the event-start, the configuration is quickly switched to B because an increased popularity of the event is detected (capsize of the ship). After a couple of days, the popularity drops slowly and the adaptive approach switches back to configuration A. About five days later, a temporary up-spike in news-articles is detected resulting in a configuration switch to B (new information about the Costa Concordia and its captain emerged). The number of relevant articles then quickly drops and configuration A is chosen for the rest of the event.

4.3.2 Implementation Details

The actual implementation of the adaptive configuration switching process is non-trivial since there is potential concurrency in the components. Furthermore, each component instance has to be notified of any configuration switch. In this section, the implementation of the adaptive approach inside the framework is described.

Because the Filter component already receives all documents as an input and checks whether a document is relevant for a query, this component was chosen to drive the configuration switching process. A disadvantage of the Filter is the expected concurrency. The Filter processes the largest number of instances, because it receives the most input data[6]. The major challenge therefore was to ensure that algorithm switches are only broadcasted once and that there is no major communication overhead between all the individual component instances, which could potentially be hosted on different machines. To deal with any concurrency issues, the implementation of the adaptive approach offloads the majority of the computational work to the database. This is done through a script that is loaded once

[6] Most news-articles are not relevant for any target-query and therefore are discarded by the Filter. These do not go through the Document Summarizer or the Updater.

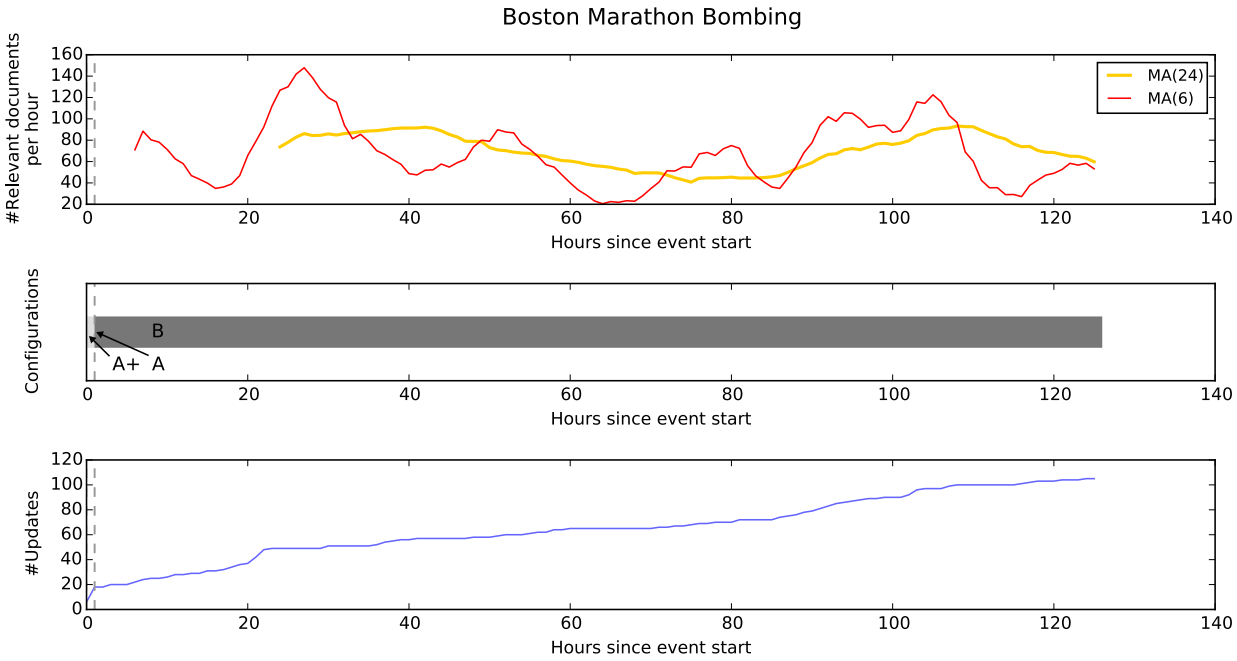

Figure 7: Configuration switches of the adaptive approach with the event Boston Marathon Bombing. Throughout the duration of the event, there is a borad news-coverage and high interest in the event. Therefore, the configuration is switched from A+ to A and from A to B as fast as possible. The adaptive approach then keeps the configuration B for the rest of the event.

to the database and can be executed from all component instances. Whenever a Filter instance detects that an input document is relevant for a query (using the restrictive approach), the Filter instance starts the script on the database with the query and the timestamp of the document as an input. The script then adds the timestamp to a list of recent timestamps for the query and calculates the moving averages MA6 and MA24. For the query, the script retrieves the current configuration and checks if the configuration should be changed, based on the rules of table 4. If the configuration has to be changed, the new configuration value is written to the database and a message is posted on the pub/sub system to inform all interested parties. With the adaptive summarization system, instances of all components subscribe to this event and change their behavior for the target-query whenever a configuration change event for that query is received. A visualization of this process is shown in figure 8.

This has the major advantage that the database takes care about the parallelization issues. No synchronization-related code had to be implemented inside the components of the summarization system and the amount of messages that is transmitted between the component instances and the database is reduced to a minimum. At the same time, a good performance is achieved, because various database queries are directly executed within the script that runs on the database. This implementation does not introduce additional restrictions in terms of concurrency and a distributed execution of the summarization system. Observations on development-events showed that this approach does not affect the computational performance in a significant way.

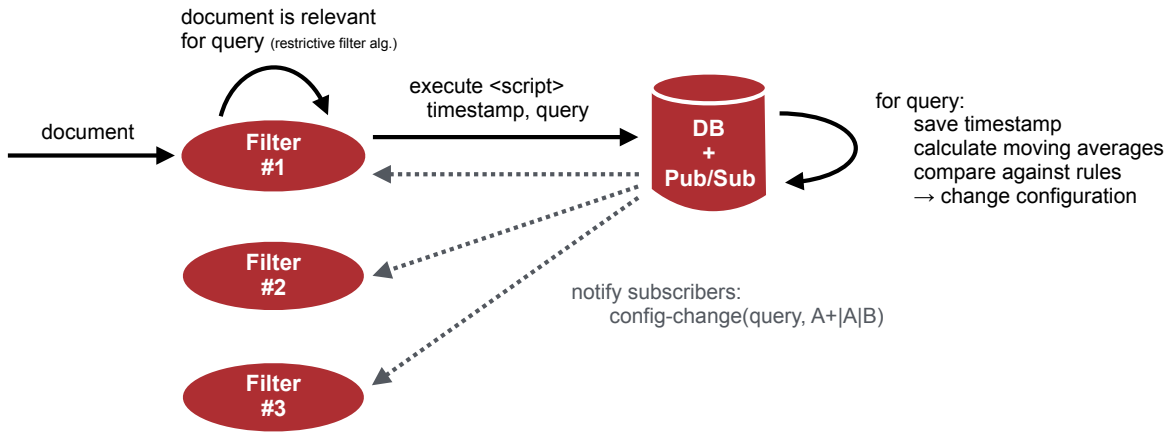

Figure 8: Implementation of the adaptive configuration switching using a database-script. For simplicity reasons, the other component instances such as Document Summarizer and Updater instances are not included in the image. The Filter instance #1 receives a document as an input. If it detects that the document is relevant for a query, the database-script is executed. It calculates new values for MA6 and MA24 and decides if a configuration switch has to be performed for target-query. If a there is a change, all subscribers are notified through the pub/sub system.

4.4 Semantic Similarity in Components

In all previous component implementations a very simple similarity measure was used, which is cosine similarity together with a vector space model using TF*ITC scores on the bag-of-words (compare section 4.1.2). Similarity between two sentences using this method is easy to compute and computationally inexpensive. However, this approach has a major downside. Even though words are stemmed before comparison, in most cases words that are similar but have another stem, like ”talk” and ”speak”, are not considered as similar by this approach. A similarity measure that incorporates similarities that are independent of the syntactic representation of the words has to incorporate semantic information or relatedness. Such an approach could be able to better find sentences in documents that are similar to the target-query. Furthermore it could better tell if sentences contain similar information, even if they do not share common words.

To further improve the adaptive summarization system, a semantic similarity measure relying on word embeddings based on word2vec [25] was used. The corpus used to train the vector space model of the word embeddings was a relatively old wikipedia dump from 2007. Current data could not be used due to restrictions that are introduced by the evaluation in section 5. Significant negative effects on the similarity measure are however not expected. The word embeddings from the trained model can be used to determine individual word similarity, because similar words are represented by vectors that lie close in the vector space, whereas vector representations of different words do not. As a similarity measure for sentence similarity, the Word Mover’s Distance [19] was used. It is based on the idea to measure similarity between sentences by finding the minimum amount of work that is required to transform one sentence into another one. The work that is required to transform one word into another is based on the distances of the respective vector representations of the words within the word embedding. In their work, Kusner et al. showed that the WMD similarity is superior to TF*IDF in a k-nearest neighbor problem-setting [19]. However this is strongly dependent on the chosen word embedding.

In the improved summarization system, WMD replaces the similarity measure of the Document Summarizer component. No further adaptions were required in this component since the MMR algorithm used for single document summarization in the Document Summarizer does not rely on any thresholds. In the Updater, the TF*ITC-based approach was not replaced, because this component implementation relies on several thresholds which in the case of a new similarity measure need to be re-calibrated. Changing the similarity measure would therefore make comparisons of the systems harder, because any potential difference could also be due to the changed threshold values.

Results of the summarization system using WMD were not fully validated during development time, however a slight increase of the number of emitted updates was observed. The overall hypotheses was that the WMD similarity measure would improve the Document Summarizer component in a way that it creates better single document summaries with more relevant content, because it can better identify sentences that are relevant to the target-query even if all words differ. Through an improved single document summarization process an overall improvement of the adaptive summarization system was expected.

A note on computational performance: The WMD semantic similarity is a more computationally expensive calculation compared to cosine similarity. WMD is based on the Earth Mover's Distance (EMD) where the computational cost of the naive solution is super-cubic. Since the sentence length in this case is limited (compare 4.1.2), the required time to compute the EMD stays at an acceptable level.

4.5 Resulting Summarization Systems

In the sections before, different summarization systems were developed and improvements and changes upon this systems were introduced. The Baseline implementation started with simple algorithms and certain drawbacks, such as the calculation of confidence scores. This approach also contains many advanced concepts, such as a strong filter and a solid MMR query-based single document summarization approach relying on TF*ITC values. In the subsequent sections, improvements based on the Baseline summarization system were created. The Updater was identified as the most interesting component of the summarization architecture, and with the Multiple Sources Updater a major change to this component was introduced. In initial tests, the multiple sources approach showed good results when the proper configuration was chosen. Unfortunately the quality of the summary was largely dependent on this configuration choice. Therefore, the adaptive approach was created which can automatically choose the configuration that is best suited for the current news-coverage of an event. Through the adaptive configuration selection, initial test showed that the adaptive approach is able to combine the advantages of all different system configurations into a single system. As a last change, the simple similarity measure of the Document Summarizer component was replaced with a more sophisticated semantic similarity measure.

The following summarization systems are fully capable of summarizing events over time with an unlimited amount of (streaming) input data. These are the major systems of this work, which are further tested in the evaluation that is presented in the next section.

- **Baseline**: This is the most basic system of this work. It was used as a solid starting point for further improvements. Besides simple algorithms, this approach also contains some advanced concepts which are computationally inexpensive.

- **Adaptive**: This is the Baseline system extended with the Multiple Sources Updater and the adaptive approach. This system is expected to consistently produce better summaries than Baseline. At the same time, this system is expected to perform nearly as good as the Baseline system in terms of computational performance.
- **Adaptive WMD**: The adaptive system with the addition of the WMD semantic similarity measure for the Document Summarizer. The hypotheses was that the semantic similarity measure produces better single document summaries and therefore leads to a better overall event summary. Computational performance is expected to be significantly lower than for Adaptive.

5 Evaluation

In the previous sections, different approaches of summarization systems were introduced in detail. Often, improvements and changes were motivated by certain hypotheses or observations made during development. In this section, a formal evaluation of the individual systems is presented. Changes and differences in terms of summarization quality are shown and assumptions from the development phase are verified. Since the aspect of the computational performance plays an important role in this work, the changes in computational performance are shown as well and real-time properties are validated.

5.1 Summary Quality

The evaluation of systems in the area of this work is non-trivial. Since the input corpus contains a huge amount of news-articles, it can not be manually annotated for evaluation purposes. The evaluation had to be able to assess the summaries that were generated by the individual summarization systems, based on a gold-standard that was derived from another source. This is what the TREC-TS 2014 evaluation [2] did, which was also used in this work. As described in section 3.4, the corpus used throughout the work already was a subset of the TREC-TS 2014 corpus with the goal of being able to compare against TREC-TS 2014 result scores. The TREC-TS 2014 evaluation can be described as follows:

1. **Events**: The evaluation consist of 15 test-events which range from 2011 to 2013. The challenge is to summarize these events over time.
2. **Gold-Standard**: For each event, so-called ”nuggets” were extracted from the edit-log of the related Wikipedia page. Nugget extraction was performed by human annotators. Nuggets represent important sub-events in the form of a full sentence or several words that were added at a certain point of time to the Wikipedia page. The timestamp of an individual edit-log entry is defined as the time when the information of the entry became ”public knowledge”. The goal for summarization systems is to produce a summary which contains information from as much nuggets as possible, while at the same time including the least amount of information not included in any nugget. Information should be included in the summary close to or before the time when it became public knowledge. Examples for nuggets from the event Costa Concordia are:
    - *”Comparisons to RMS Titanic”*
    - *”3,206 passengers and 1,023 crew members were on board”*
    - *”preliminary indications are that there may have been significant human error on the part of the ship’s Master, Captain Francesco Schettino,”*
3. **Nugget-Matching**: To assess the quality of the summary, nuggets are matched to the updates (sentences) of the summary. A nugget matches an update if the update contains the information of the nugget. This is an n:m assignment, meaning that updates may match multiple nuggets and nuggets may match multiple updates. The nugget-matching data provides crucial information that is required to calculate the result-scores. Matchings are created by human annotators, which is still the best and easiest way. The matching-process requires a certain interpretation of the

meanings, because the information of the nugget may be included in an update even if they do not share common words or certain numbers differ slightly. To limit the work that has to be done by the annotators, the TREC-TS 2014 evaluation only considers the top-60 updates (determined by confidence score) of each summary and system for the process of manual nugget-matching. The remaining updates are included if they are exact duplicates of any top-60 update from any other participating system. All other updates are considered as non-matching.

The evaluation of this work was performed for the summarization systems developed as part of this thesis. Since the nugget-matching interface of TREC-TS 2014 was not publicly available, a separate nugget-matching web interface was built in this work with a strong focus on user experience to ensure a smooth nugget-matching process. Besides nugget-matching capabilities, the interface also allows to mark updates as irrelevant to the overall topic. This adds another interesting possibility to analyze the summary. A more detailed description as well as screenshots of the nugget-matching interface are available in appendix D. For the automatic matching part of the non-top-60 updates, the matches from TREC-TS 2014 were used.

Evaluation metrics were primarily taken from TREC-TS 2014. The exact mathematical definition can be found in [2].

- **Normalized expected gain (nEG)**: Precision of the summary which is measured by the ratio of updates that are on-topic (match nuggets). Values are between zero and one with a greater value representing a better result.
- **Comprehensiveness (C)**: Coverage of the summary which is measured by the ratio of event-information that is included in the summary (∼Recall). The possible event-information is represented by the set of nuggets. Values are between zero and one with greater values representing better results.
- **Expected latency (E[Latency])**: Timeliness of the updates compared to the timestamps of the nuggets. Values are between zero and two. A value greater one indicates that the information was included in the summary before it became public knowledge (included in Wikipedia). A value less than one indicates that the information of the summary usually was outdated to a certain degree. A value near or greater than one is desired.
- **Combined target-metric ($\mathcal{H}$)**: The target-metric of TREC-TS 2014. This is the harmonic mean of the normalized expected latency gain (nEG with a latency discount/bonus) and the latency comprehensiveness (C with a latency discount/bonus). With the latency discount/bonus it could theoretically be possible to achieve scores greater than one. $\mathcal{H}$ is used to determine if a system performs better than another system. Greater values represent better results.
- **Ratio of irrelevant updates (IU)**: A non-TREC-TS 2014 metric. This is the ratio of top-60 updates that were annotated as irrelevant. IU is between zero and one, where values close to zero are desired.

By using the evaluation of TREC-TS 2014, different restrictions apply. First, it is not allowed to use any information from a point in time that is after the timestamp of a document that will be processed by the summarization system. This restriction makes sure that the simulation is similar to a real-world scenario. All summarization systems of this work meet this restriction, for example the word embeddings of the adaptive approach with semantic similarity are created from an old wikipedia dump of 2007 (compare section 4.4). As a second restriction, only sentences of the original sentence splitting

of the TREC-TS 2014 corpus are allowed to be included in the summary. These sentences, as described in section 3.4, were created based on the HTML-markup of the news-article web-pages with all HTML-tags stripped. Implementations of this work do not meet this restriction out-of-the-box, because of the different approaches used in the Filter component. All systems of this work use the raw HTML-markup as an input, remove any boilerplate content to extract only the news article text and then perform the sentence splitting. Besides any potential improvements by removing boilerplate content, this also ensures that the performance measurements represent a real-life scenario where content is usually not preprocessed. To meet this TREC-TS 2014 restriction, a simple change to the framework architecture was made to automatically replace own sentences with the equivalent sentence from the preprocessed sentences of the corpus. The document source was changed to add the preprocessed sentences as a metadata entry to each document. Since all metadata entries are passed with the documents and sentences throughout the summarization system, preprocessed sentences are accessible at any point in the framework architecture. To replace own sentences with the equivalent preprocessed sentence, a Pre-Updater component was created, which replaces all sentence content and the sentence-id entry with the most similar sentence from the preprocessed sentences of the same document. The replaced sentence then is passed to the Updater component for any further processing. A visualization of the change is provided in figure 9. With this change, all restrictions are met and the evaluation can be performed with the summarization systems of this work.

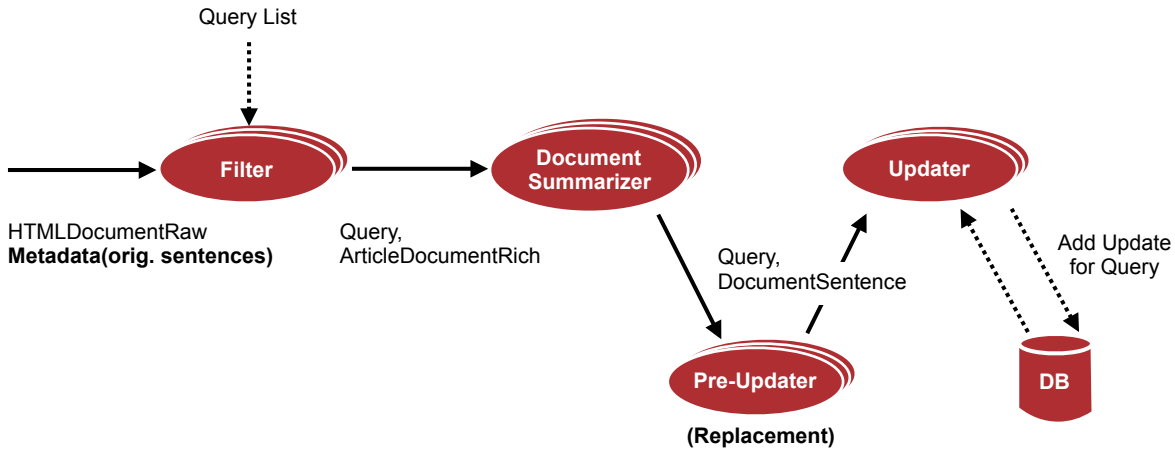

Figure 9: A change of the framework architecture to meet the TREC-TS 2014 evaluation restrictions. The input document contains the original sentences as a metadata entry. The new component "Pre-Updater" intercepts all Query/DocumentSentence pairs that would go to the Updater and automatically replaces the sentence with its equivalent from the preprocessed sentences of the same document.

5.1.1 Evaluation Procedure

For the overall evaluation of the summarization quality, there were two goals. First, to compare systems developed in this work against each other and to compare them against the TREC-TS 2014 results. Second, to verify observations and assumptions that were made during development time of the different component configurations and the adaptive approach. Therefore, the evaluation was split into a primary and a secondary part. In the primary part, the final systems of section 4.5 were evaluated. Furthermore the best system of TREC-TS 2014 (CUNLP) was re-evaluated as well, to measure devia-

tions from the official TREC-TS 2014 results. Systems of the primary evaluation therefore are Baseline, Adaptive, Adaptive WMD and CUNLP. These systems were evaluated against all events of TREC-TS 2014. Systems of the secondary evaluation are:

- **A+**: The A+ configuration of the adaptive system. This basically is the adaptive system, starting with A+ without reacting on configuration changes. The evaluation of this system and the other configurations show how well the adaptive system can combine the individual advantages of each individual configuration.
- **A**: The A configuration of the adaptive system.
- **B**: The B configuration of the adaptive system.
- **Adaptive Boilerplate**: The adaptive system without boilerplate removal. It uses the sentences of the TREC-TS 2014 corpus as an input and does not extract the article text from the HTML-markup. A comparison against Adaptive was expected to reveal advantages of the boilerplate removal process.

These systems were evaluated against a selection of 6 events, where 5 were chosen from the test-events and one event (Boston Marathon Bombing) was chosen from the events that were used during development of the systems to validate observations made during development. The selection of the 5 events from the test-events was done before any results were obtained with the goal to include a good selection of the different event-types.

To perform the nugget-matching as described earlier, the nugget-matching interface of this work was used. Three annotators were recruited for manual nugget-matching which all performed the same tasks. Individual tasks were to create the nugget-matchings for the top-60 updates of an individual summary and to annotate irrelevant updates. Because each annotator performed the nugget-matching for every summary, each system was assessed three times per event. To compensate for a potential learning-effect of the annotators, the order in which annotators worked on tasks was different for each person. Events were always treated as a block, which means that all tasks of the same event were finished consecutively. The system order was shuffled for each event with the goal to prevent the establishment of an always-comes-before relationship for systems. The event-order was manually optimized for each annotator so that events which one annotator worked on at the beginning were assigned to the other annotators at the middle and at the end. At the beginning of the annotation process, each annotator was briefed with annotation guidelines describing the task and providing examples of how to perform the nugget-matching. Examples were taken from the nugget-matching results of the TREC-TS 2013 challenge. Furthermore, annotators were briefed on the annotation of irrelevant updates. The first task for each annotator was a dummy-task with a system/event combination that was not evaluated and results from this initial task were discarded. This was meant to further reduce any potential learning effect.

5.1.2 Results

In this section, results of the evaluation are presented, discussed and further interesting results are shown, such as inter-annotator-agreement values. Result scores for TREC-TS-related metrics were calculated using the nugget-matching of the annotators and the official TREC-TS 2014 evaluation script.

The first result scores to be presented are the scores of the reference system, the best system of TREC-TS 2014 (CUNLP), compared to the scores that the same system achieved in the official TREC-TS 2014 evaluation. Results in table 5 show that scores are similar and the differences are small.

Table 5: A comparison of the results of this evaluation against the TREC-TS 2014 evaluation for the CUNLP system.

	$\mathcal{H}$	$\mathcal{H}$ (TREC-TS 2014)	Δ std.	Δ_{max}
CUNLP	0.13	0.12	0.0296	0.07

The standard deviation of the differences per event is on a medium-low level, showing that individual events usually do not differ much. The maximum difference of Δ_{max} indicates that there was no huge outlier. This shows that the evaluation results of this evaluation compared to the official TREC-TS 2014 evaluation are sufficiently similar to compare the individual system-results with official results of 2014. As a further check, the inter-annotator agreement scores between the nugget-matches of this evaluation and the TREC-TS 2014 evaluation was calculated. Agreement is based on a set of annotations, where each nugget-update match was labeled ”positive” and each non matched nugget-update combination was labeled ”negative”. Because the non-matching combinations were not explicitly annotated in the nugget-matching process, besides the usual agreement-metrics, the F1 score was calculated by only using the positive samples. This method is an effective measure for cases where only one class of labels is available, for example when there are only positive samples [14]. The agreement scores are listed in the following table 6.

Table 6: Inter-annotator agreement of the nugget-matchings from this evaluation and the TREC-TS 2014 evaluation for the CUNLP system outputs.

Simple Agreement (Non-Chance-Corrected)	Cohen’s Kappa	Krippendorff’s Alpha	F1
0.99	0.44	0.43	0.44

The high score for the simple agreement, which is not chance-corrected, shows that the distribution of the labels ”positive” and ”negative” is highly biased. With 60 updates and 60 nuggets, there are 3600 annotations when considering each non-matching pair as a negative label. The usual number of nugget-matches per task is around 30, therefore the ratio of ”negative” labels is very high. The high number of possible combinations indicates that there is a high chance of annotators missing a positive combination. Furthermore the subjectivity of the overall task is high, because the annotator often has the choice between multiple similar nuggets during a task. For this reasons the observed agreement values of 0.43-0.44 (”moderate agreement”) are on a satisfying level. Additionally, there was no annotator with an average agreement value that differed more than 0.05 from the overall average as reported in table 6. These results support the assumption that a (cautious) comparison between the TREC-TS 2014 evaluation and this evaluation can be made.

After looking to the overall comparability with the reference evaluation, the result scores of the systems developed in this work can be shown. The scores of all metrics averaged over the three annotators are listed in table 7. This table contains the averaged scores over all test-events without development-events. The exclusion of the development events does not provide a disadvantage to the compared system CUNLP, because its scores for these events were relatively low (compare table 8).

Table 7: Result scores of the primary systems for all target metrics. Scores were averaged over all non-development events. For the average TREC-TS 2014 E[Latency] as well as # Updates, it was not possible to calculate the values influence from development events, because a fine grained list over all event-values was not available for these metrics. Average values over all events are noted in parenthesis.

System	$\mathcal{H}$	nEG	C	E[Latency]	IU	# Updates
Baseline	0.12	0.11	0.23	1.05	0.03	74
Adaptive	0.17	0.13	0.26	1.23	0.07	48
Adaptive WMD	0.15	0.11	0.24	1.27	0.04	49
CUNLP	0.12	0.07	0.32	1.22	0.32	242
AVG TREC-TS 2014	*0.06*	*0.04*	*0.36*	*– / (1.29)*	*–*	*– / (8529)*

This data shows several interesting results. First, the best system of the evaluation was Adaptive, being followed by Adaptive WMD. The usage of the semantic similarity measure WMD in the Document Summarizer component for single document summarization did not provide the expected improvements. For both, nEG (precision) and C (recall), Adaptive WMD was slightly inferior to Adaptive, which is also reflected in the target-metric $\mathcal{H}$. The latency values and the percentage of irrelevant updates on the other hand improved slightly. The addition of semantic similarity did however not change the results in a drastic way, which leads to the assumption that non-major changes in the Document Summarizer component do not have a big impact on the overall results within the adaptive system. Since there is no dedicated comparison of the Document Summarizer components itself, this assumption needs further verification. Another interesting result is the high score of the target-metric $\mathcal{H}$ for Baseline, which is equal to the score of CUNLP. Baseline therefore performs extremely well. The reason for that is the high nEG score (precision) which is, compared to the nEG scores of the average TREC-TS 2014 system or even CUNLP, significantly better. When looking at the number of updates (# Updates) of all systems, the reason for these high precision values can be seen to be caused by the low amount of updates that were emitted by the systems developed in this work. The low number of updates did lead to a lower recall (C) value, however systems are more balanced in terms of precision and recall. In comparison, the average TREC-TS 2014 system outputs approximately 100 times more updates, and the best system of the challenge outputs more than three times as much updates as any (primary) system of this work. These high numbers of updates indicate that most TREC-TS 2014 systems are far from usable in a real-life scenario, because it would not be feasible for a human user to read a summary this large. Another benefit of the better balance and optimization for precision of the developed systems is the low ratio of irrelevant top-60 updates (max. 7%). CUNLP on the other hand has a high ratio of irrelevant top-60 updates of 32%. This is also an important factor for a real-life scenario, because each irrelevant update weakens the trust of users in the related system. As described before, the only drawback of the developed summarization systems is a low value for C (recall). When comparing to the the average TREC-TS 2014 system, the balance between precision and recall however seems to be better in the developed systems which is also acknowledged by the high scores of the target-metric $\mathcal{H}$.

Individual scores of the target-metric $\mathcal{H}$ for all systems and events are listed in table 8. Data shows that for test-events, Adaptive most often achieved the best score, followed by Adaptive WMD. Both systems had no outliers of extremely low (zero) scores, which shows that the adaptive systems successfully adapted to the various types of events with their different characteristics. In contrast, both Baseline and CUNLP had outliers with a near-zero score.

Table 8: Individual values for the target-metric $\mathcal{H}$ for all primary systems and events.

Type	Event	Baseline	Adaptive	Adaptive WMD	CUNLP
Used During Development	Costa Concordia	0.29	0.16	0.23	0.11
	Queensland Floods	0.12	0.17	0.13	0.07
	Boston Marathon Bombing	0.05	0.25	0.21	0.02
	Quran Burning Protests	0.21	0.23	0.21	0.23
	AVG	**0.17**	**0.20**	**0.20**	**0.11**
Unseen During Development (Test)	European Cold Wave	0.00	0.05	0.07	0.02
	Egyptian Riots	0.08	0.14	0.18	0.09
	In Amenas Hostage Crisis	0.09	0.11	0.09	0.06
	Russian Protests	0.13	0.09	0.10	0.08
	Romanian Protests	0.04	0.10	0.14	0.21
	Egyptian Protests	0.13	0.18	0.18	0.10
	Russia Meteor	0.22	0.29	0.24	0.01
	Bulgarian Protests	0.26	0.26	0.23	0.30
	Shahbag Protests	0.02	0.15	0.05	0.16
	Nor'Easter	0.20	0.21	0.21	0.17
	Southern California Shooting	0.14	0.25	0.21	0.09
	AVG	**0.12**	**0.17**	**0.15**	**0.12**

As with the CUNLP system and the comparison of the evaluation of this work and the TREC-TS 2014 evaluation, the inter-annotator agreement values were measured for the agreement of all annotators per event/system combination. Values are listed in table 9, showing similar results with a slightly lower agreement. With an agreement score of 0.40 to 0.41 this still is an acceptable result. As before, no average agreement of annotator pairs deviated more than 0.05 from the overall average.

Table 9: Average of the inter-annotator agreements between all nugget-matchings of the same event/system combination.

Cohen's Kappa	Krippendorff's Alpha	F1
0.40	0.41	0.41

Significance of the differences in system scores

Result scores showed that the adaptive system performed better than any other system tested in this evaluation. Even though the differences to Baseline, which Adaptive is based on, and CUNLP are big, the significance of these improvements is important for a final judgment. The goal therefore was to show that there is evidence that the improvements are not a result of chance. As a test, the Wilcoxon signed-rank test was chosen, which can be used to show that there is a significant difference in the median values of two distributions (which are not required to show a normal distribution).

The hypothesis are one-tailed:

$$H_0 : \tilde{x}_1 \leq \tilde{x}_2 \qquad H_1 : \tilde{x}_1 > \tilde{x}_2 \tag{8}$$

The Wilcoxon signed-rank test operates on the differences between the individual observations, in this case on the differences between the $\mathcal{H}$ scores of the same event of the two systems that should be compared. Significance levels which were tested are $p \leq 0.05$ as well as $p \leq 0.1$. The first test is a comparison between Adaptive and Baseline, with the goal to validate that the better overall score of

Adaptive is statistically significant. Using the pairwise differences of the $\mathcal{H}$ scores of table 8 (test-events only), a value of $W = 3$ from the positive ranks is obtained. Since both systems have the same scores for the event Bulgarian Protests, the population size reduces to $N = 10$. With such low populations it is common to compare W to the table of critical values for the Wilcoxon signed-rank test to check if the null-hypothesis can be discarded. For $N = 10$, the critical value for the 0.05 significance level is 10. Since $W = 3 \leq 10$, the null-hypothesis can be discarded with $p \leq 0.05$. This means that there is strong evidence that Adaptive performs better than Baseline.

The comparison between CUNLP and Adaptive was performed the same way. With the pairwise differences, a value of $W = 15.5$ with a population size of $N = 11$ is calculated. The critical value for the significance level 0.05 is 13, which means that with a greater W value of $W = 15.5$, the null-hypotheses can not be discarded with an error $p \leq 0.05$. For the significance level 0.1, the critical value is 18. In this case, the null-hypotheses can be discarded with an error $p \leq 0.1$. This means that there is evidence that Adaptive performs better than CUNLP, however it is not as strong as the evidence of Adaptive performing better than Baseline.

The same Wilcoxon signed-rank test was performed for the results of CUNLP of this evaluation and the results of CUNLP of the TREC-TS 2014 evaluation. In this case the pairwise differences of the scores of all events could be used. With a value $W = 23$ and $N = 11$ (there are 4 equal scores), the critical value for the significance level of 0.1 is 18. Since 23 is greater than 18, the null-hypotheses can not be discarded within an error level of $p \leq 0.1$. This supports the assumption that the results of this evaluation compared to the TREC-TS 2014 evaluation are rather similar.

Results of the significance test show that the observed higher scores of the Adaptive system are likely to be based on improvements of the overall summarization system instead of being a result of chance. The same results can be obtained by using the simple sign test.

Secondary evaluation

Next, results of the secondary evaluation with the different system configurations are shown. The main motivation for the evaluation of the other systems on a subset of the events was to verify observations and implications that were made during development of the systems. Fine-grained results are listed in table 10.

When looking at the results of Adaptive Boilerplate (adaptive system without boilerplate removal), the reduced number of updates is evident. The number of updates thereby is reduced significantly, sometimes Adaptive Boilerplate only emits half the number of updates compared to Adaptive. The reason is that without boilerplate removal the first sentences may contain non-article related text such as menu structures, advertisements or headlines from other news. The beginning of the text therefore is not always the beginning of the news-article. The position of a sentence in the news-article however is an important feature which is used by all the Filter components of the summarization systems developed in this work to determine if the article is relevant for a target-query. Boilerplate content at the beginning of the text therefore results in a lower number of updates and a lower comprehensiveness (recall) score C, especially for events which already had a small number of updates for Adaptive. Interestingly, for events with a higher number of updates and therefore with a broader news-coverage, Adaptive Boilerplate performs well with good result scores which are equal or better than the scores of Adaptive. This is mainly due to an increased nEG score (precision). Boilerplate removal however brings a significant advantage when looking at the ratio of irrelevant updates (IU), where the approach

without boilerplate removal has a highly increased ratio of irrelevant updates (min. increase: 0.05; max increase: 0.27). For this reasons boilerplate removal is an important factor of the summarization systems of this work.

One main contribution of this work is the approach to automatically adapt the summarization system to the news-coverage of a target-event. A comparison of the individual configurations A+, A and B as well as the adaptive system itself is important to verify the advantages of the adaptive approach. These values can also be used to check if the adaptive approach generalizes well to new and unseen events. Looking at results of A+, A and B, the number of updates is highest with A+, second highest with A and lowest with B. With the higher number of updates, a higher C (recall) score is often achieved which means that the less restrictive nature of A+ leads to a better overall event coverage. The opposite effect can be seen for nEG scores (precision), where the more restrictive configurations with less updates lead to higher values. Problematic for all systems is that they perform very well for certain events but really badly for others. This is the exact same observation that was made during development of the summarization systems and the main motivation of the adaptive approach. For most events, the overall $\mathcal{H}$ score of the adaptive approach is close to the score of the best performing single configuration. For some events the adaptive approach is even better than every individual configuration (e.g. In Amenas Hostage Crisis). This proves that Adaptive can generalize well to these unseen events and is likely to select the correct configuration for an event. The scores which are better in Adaptive compared to the individual configurations show that certain events consist of multiple parts which are best summarized using different configurations. This further indicates that the adaptive switching of configurations during runtime is an highly effective approach. Other than the individual configurations, Adaptive always finds a good balance between nEG (precision) and C (recall) values. One exception of this is the Russian Protests event, where Adaptive performed significantly worse than the A configuration, showing that further improvements for the adaptive selection of configurations are still possible. The overall scores of table 8 and the differences to Adaptive prove that Adaptive is a capable system which performs consistently well without major drawbacks.

Critics on the evaluation

The measures of TREC-TS 2014 were used in this work primarily for being able to compare against the TREC-TS 2014 results, especially against the best system that participated in the challenge. A second reason was that the time restrictions of this thesis made it infeasible to create an own evaluation, which is difficult and time-consuming for such summarization systems using a large input corpus. The TREC-TS 2014 evaluation however has a disadvantage, which is the treatment of the non-top-60 updates which are not considered for manual annotation. These updates are automatically matched against the top-60 updates of other participant systems. If another participant included the same update in the top-60 updates, the update will be considered for nugget-matching for the other system as well. This approach automatically penalizes systems that are unique and output updates that no other system found, because the chance of automatic matching would be reduced. A stronger limitation in size of the summary would be desirable. Such a size restriction for example could be dynamic, based on the duration of the event. It is important to mention that it is unlikely that this effect affected the systems developed in this work in a negative way, because of their optimization for precision which led to a smaller amount of updates.

Table 10: Results of the secondary evaluation. Differences against Adaptive are wrapped in parentheses.

Event	Value	A+	A	B	Adaptive Boilerplate
Boston Marathon Bombing *Info: Used during development*	$\mathcal{H}$	0.02 (-0.23)	0.09 (-0.16)	0.23 (-0.02)	0.24 (-0.01)
	nEG	0.01 (-0.14)	0.07 (-0.08)	0.18 (+0.03)	0.17 (+0.02)
	C	0.30 (-0.07)	0.29 (-0.08)	0.35 (-0.02)	0.29 (-0.08)
	IU	0.05 (+0.03)	0.04 (+0.02)	0.01 (-0.01)	0.18 (+0.16)
	# Updates	1735 (+1631)	237 (+133)	72 (-32)	52 (-52)
Egyptian Riots	$\mathcal{H}$	0.15 (+0.01)	0.12 (-0.02)	0.11 (-0.03)	0.22 (+0.08)
	nEG	0.06 (± 0.0)	0.08 (+ 0.02)	0.11 (+0.05)	0.10 (+0.04)
	C	0.40 (+0.01)	0.22 (-0.17)	0.08 (-0.31)	0.42 (+0.03)
	IU	0.07 (-0.02)	0.11 (+0.02)	0.00 (-0.09)	0.14 (+0.05)
	# Updates	83 (±0.00)	15 (-68)	4 (-79)	44 (-39)
In Amenas Hostage Crisis	$\mathcal{H}$	0.07 (-0.04)	0.06 (-0.05)	0.06 (-0.05)	0.14 (+0.03)
	nEG	0.07 (-0.02)	0.36 (+0.29)	0.56 (+0.47)	0.17 (+0.08)
	C	0.47 (-0.02)	0.22 (-0.25)	0.10 (-0.39)	0.42 (-0.07)
	IU	0.00 (±0.00)	0.00 (±0.00)	0.00 (±0.00)	0.11 (+0.11)
	# Updates	82 (+30)	9 (-43)	4 (-48)	32 (-20)
Russian Protests	$\mathcal{H}$	0.07 (-0.02)	0.17 (+0.08)	0.14 (+0.05)	0.15 (+0.06)
	nEG	0.03 (-0.03)	0.15 (+0.09)	0.23 (+0.17)	0.10 (+0.04)
	C	0.20 (±0.0)	0.27 (+0.07)	0.15 (-0.05)	0.24 (+0.04)
	IU	0.20 (+0.06)	0.06 (-0.08)	0.00 (-0.14)	0.31 (+0.17)
	# Updates	305 (+127)	53 (-125)	15 (-173)	128 (-50)
Russia Meteor	$\mathcal{H}$	0.24 (-0.05)	0.24 (-0.05)	0.21 (-0.08)	0.15 (-0.14)
	nEG	0.13 (-0.03)	0.15 (-0.01)	0.20 (+0.04)	0.12 (-0.04)
	C	0.36 (+0.02)	0.32 (-0.02)	0.20 (-0.14)	0.19 (-0.15)
	IU	0.04 (+0.04)	0.01 (+0.01)	0.00 (±0.00)	0.27 (+0.27)
	# Updates	163 (+121)	40 (-1)	16 (-25)	25 (-16)
Southern California Shooting	$\mathcal{H}$	0.23 (-0.02)	0.23 (-0.02)	0.18 (-0.07)	0.20 (-0.05)
	nEG	0.10 (-0.03)	0.21 (+0.08)	0.20 (+0.07)	0.17 (+0.04)
	C	0.39 (+0.07)	0.13 (-0.19)	0.08 (-0.24)	0.17 (-0.15)
	IU	0.05 (+0.02)	0.00 (-0.03)	0.00 (-0.03)	0.19 (+0.16)
	# Updates	50 (+17)	9 (-24)	6 (-27)	16 (-17)

5.2 Computational Performance

The second part of the evaluation was to measure the computational performance of the summarization systems and to verify their real-time capabilities. In this work, there are two real-time aspects. Real-time in temporal summarization means that a system is capable of emitting updates at any time when it detects a new sub-event, not just at predefined fixed intervals. Computational real-time on the other hand is the definition of an upper-bound for the time that the system is allowed to require until it finishes the processing of an item. The goal of this part of the evaluation was to show that there is such an upper-bound, as well as measuring the overall performance.

Measurements were taken on a separate dedicated machine, using the measurement data which is automatically collected by the framework implementation as described in section 3.3. The machine was a large compute server with 26 cores and a 512GB of RAM. For the first part of the test, the summarization systems were launched with only one instance per component. This means that there is no performance gain from the large amount of cores. No changes to the summarization systems were made in terms of caching and IO-operations, therefore the large amount of RAM was not utilized

during this part. The exact same systems were tested on a standard MacBook Pro for comparison, and the MacBook Pro actually performed slightly better due to a higher speed of the IO (SSD). The database for all tests ran on the same machine (in-memory). For comparison, the results of Baseline and Adaptive were analyzed. For each of the 15 events the systems were executed three times. From all the collected data, the average time per component and standard deviation values were calculated. The maximum required time for all processed items and the maximum overall time for items that triggered updates was identified. Results are listed in table 11. As expected, all components of both approaches are very fast in average. A high standard deviation throughout the components in relation to their average required computing time indicates that the processed data is very heterogeneous in terms of the overall sentence-count and sentence-sizes. A comparison of the average values for Baseline and Adaptive shows that the Document Summarizer of Adaptive requires slightly more time per item. This is due to the increased number of maximum sentences that this component can emit for Adaptive. The Updater component also shows an increased computation time which is a result of the more complex approach of the Multiple Sources Updater and its multi-step approach which results in more similarity comparisons than the Updater of Baseline.

The performance over time is an important factor to scalability and to ensure that the system supports long-running events with potentially unlimited timeframes. Filter and Document Summarizer are independent from any previous data, therefore the performance does not change over time. Updater components for both, Baseline and Adaptive use previous updates and, in the case of Adaptive, previous candidates. These lists are limited in size, performance therefore will not decrease with a large amount of processed data. A typical performance curve over an event is plotted in figure 10 which is taken from an event with a broad news-coverage (Boston Marathon Bombing). The performance of the Updater decreases only to a certain point and then is almost constant until the end of the event.

Table 11: Results of the performance measurements (milliseconds). The average processing time for an item usually is extremely low with a high standard deviation. The overall maximum computation time is close to the maximum computation time for actual updates, except for the Filter component.

System	Component	Average	Std.	Max.	Max. (Updates)
Baseline	Filter	7.89	6.17	1383.50	102.52
	Document Summarizer	1.91	2.03	68.00	67.83
	Updater	5.15	1.64	65.72	65.61
Adaptive	Filter	7.92	6.17	1401.67	101.63
	Document Summarizer	3.05	2.37	85.78	67.79
	Updater	7.86	11.04	275.16	272.10

To determine an upper-bound for the time which is required to fully process an item, the maximum computation times are used. Values are much higher than the average time required to process an item. One reason is that some sentences contain many rare words (for example documents in foreign languages) which are not found in the caches that were installed to reduce IO operations for ITC scores or other values that are kept on disk. In such cases if many cache-misses occur, the unusual high number of IO operations slows the overall summarization process. The exceptionally high maximum value for the filter components (which is present in all individual runs) however is expected to originate from third-party libraries used in the application, because Filter itself does not require any IO operations. These high maximum computation times may be caused by malformed HTML input or unusual text-patterns which could increase the complexity in the 3rd party libraries. High Max. values for Filter are consistent, but occur only individually in each run (which can also be seen from the Std.

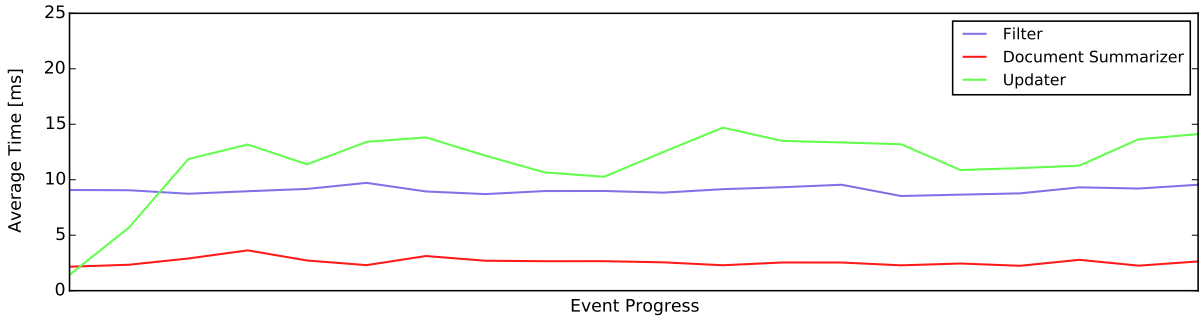

Figure 10: Performance of the individual components over time (System: Adaptive, Event: Boston Marathon Bombing).

values). When comparing Max. to Max. (Update) for the filter, the differences show that the high Max. values are not related to the updates or any feature that relates to relevant content. Since there is no obvious answer to the issue of the high maximum computing times for Filter, this aspect needs further research. Despite these values, there is an acceptable upper-bound for the computational time which the summarization system requires to fully process an item. The measurements taken during the repeated runs over the large corpus of this work provide enough evidence to specify such an upper-bound. With a sum of all maximum values (total potential worst-case performance) of 1417.69ms for Baseline and 1745.80ms for Adaptive, the upper-bound can be set to two seconds. This upper-bound was also verified with individual performance measurements performed on a standard MacBook Pro using the same system implementations.

For the system Adaptive WMD, the evaluation of the summary quality showed that there is no clear improvement compared to Adaptive. By replacing the simple cosine-similarity and TF*ITC scores with the Word Mover's Distance semantic similarity measure, there is a major increase in computational complexity at the Document Summarizer. Performance measurements showed that the average time to process an item in the document summarizer increased from 2.97ms to 131.33ms. The maximum required processing time also increased significantly from 69.13 to 575.50. Results are listed in table 12.

Table 12: Results of the performance measurements for the Adaptive WMD Document Summarizer.

Component	Average	Std.	Max.	Max. (Updates)
Document Summarizer	131.33	69.61	575.50	514.49

Parallelism

To test if the parallelism capabilities of the summarization architecture are effective, the time required to process a complete event for both alternatives, with parallelism enabled and with parallelism disabled, were timed and compared. The system which was used for this test was Adaptive, the events were Boston Marathon Bombing and the Chelyabinsk Meteor event, because they showed a broad news-coverage with a relatively high amount of items processed by all components. To prevent any (major) delay resulting from reading the corpus off the hard-drive disk, the related documents were loaded into a ramdisk (about 100GB). The non-parallelized version launched one instance for each component whereas the parallelized version launched 10 instances of the Filter and 2 instances of

the Document Summarizer and Updater components. Timestamps were taken before the start of the systems and after the system finished processing all documents inside the timeframe of the event. The expected result was a speedup from parallelization near a factor of 10.

Timing-results of table 13 show that there was a significant speedup from parallelization. Because the startup and shutdown times of the systems are included in the timings and the overall processing time of the Boston Marathon Bombing event is relatively short, the speedup for the Boston Marathon Bombing is lower than 10. For the Chelyabinsk Meteor where startup and shutdown times are smaller in relation to the overall processing time, the speedup reaches a value of 9.23 which is near the optimal speedup of 10. This shows that the summarization framewok based on Apache Storm is capable of effectively parallelizing the summarization process. This can either be used to speed up simulations or to handle a large input stream in a real-life scenario.

Table 13: Comparison of a parallelized version and a non-parallelized version of Adaptive.

Event	Non-Parallelized	Parallelized	Speedup
Boston Marathon Bombing	3151s	450s	7.00
Chelyabinsk Meteor	4892s	530s	9.23

5.3 Summary

The evaluation showed that the summarization quality of the systems developed in this work is very good, while at the same time the systems are real-time capable in terms of summarization and computational performance. The best system of this work (Adaptive) showed superior results compared against the best system of TREC-TS 2014 with a significant improvement on a medium error-level. Nonetheless comparison should not be treated as a final judgment since the systems were only tested on a subset of all the events of the official TREC-TS 2014 challenge. Surprisingly, Baseline performed very well although the component implementations of this approach only rely on simple algorithms. Adaptive was able to significantly improve the performance (low error level) against Baseline by a big margin. At the same time, Adaptive performed almost as fast as Baseline in terms of computational performance. The approach relying on semantic similarity in the Document Summarizer component, Adaptive WMD, showed unexpected results with no major differences compared to Adaptive. Changes of the similarity measure did not result in the expected improvements.

With the secondary evaluation, initial assumptions and hypothesis from the implementation phase could be verified. It showed that the process of boilerplate-removal has a positive effect on the overall summarization quality, which primarily is reflected in a lower amount of irrelevant updates that are included in the summary. Furthermore boilerplate-removal enables simple and but effective filtering approaches that rely on features related to the beginning of an article-text. The secondary evaluation also showed that the adaptive approach is effective in the configuration switching process for new and unseen events.

As a conclusion, the evaluation of the summarization systems showed that the adaptive approach combined with the Multiple Sources Updater is an effective real-time capable summarization system.

6 Conclusion and Future Work

In this work, multiple different approaches of real-time capable summarization systems were introduced, which showed a good summarization quality as well as a computationally efficient and fast summarization process. The first step towards the systems developed as part of this work was to create and implement a framework for real-time summarization. This framework and its implementation was developed to provide an abstraction of all technical details for the actual summarization system implementations, while at the same time enabling stream-processing and parallelization throughout the architecture. Actual summarization systems of this work were created on top of this framework implementation. The first system was a combination of multiple simple approaches, which was then defined as the Baseline for this work. Based on different observations, the last step of the framework, the Updater, was identified as the component which has the highest potential to achieve improvements for the overall summarization process. As a consequence, the Multiple Sources Updater was created which only allows information to be included in an update if this information is reported by multiple sources. This concept reduced the amount of irrelevant updates which the system included in the summary and increased the overall precision. Because different component configurations were required to achieve consistently good results for the different events used during development, an adaptive approach was created. Based on the detected importance of a target-event, which is measured by its news-coverage, the adaptive approach automatically switches between multiple component configurations to ensure that the best suitable configuration is chosen to summarize this event. Evaluation showed that this is the best approach developed in this work. Results revealed a superior performance compared to the Baseline approach, which was achieved without sacrificing any real-time properties or introducing limitations on potential parallelization. The comparison to a state-of-the-art system, which was the best system of the TREC-TS 2014 challenge, also showed superior results of the adaptive system. Since not all events contributed to this comparison, because some were used during development of the summarization systems, a general superiority of the adaptive approach can not be implicated with full certainty. Results are however very strong. An additional experiment with semantic similarity was created, where the simple similarity measure of the single document summarization component was replaced with a more advanced semantic similarity measure. Results of this experiment showed no distinct improvements against the standard approach, which means that more drastic changes are required to affect the overall summarization results of the adaptive system. Performance testing of the major systems of this work led to an acceptable upper-bound for the required processing time. The average performance of all components for Baseline as well as for Adaptive was extremely low, therefore a decision whether to issue a new update based on an incoming documents is usually made in short time (a couple of milliseconds).

In conclusion, the introduced systems are able to summarize events in real-time with good quality for different types of events with varying levels of news-coverage. The biggest contribution for the high quality in summarization is the adaptive selection of the individual configurations, based on the detected importance of the target-events. The framework which was created as part of this work played a significant role as well, because it provided an efficient way to build summarization systems and to experiment with new ideas. This framework could also benefit further research in the same area.

In future work, more extensions to the overall approach of the adaptive summarization system could be investigated. For example, approaches such as query expansion or machine learning may improve the overall summarization system by more accurately filtering documents or performing better single

document summarization. Query expansion in particular was an important part of the second best system of TREC-TS 2014 [35]. More experiments on semantic similarity could be performed, for example to test if semantic similarity has a higher impact in the Updater component than it had in the Document Summarizer. Results of individual components, namely Filter and Document Summarizer could be evaluated separately, measuring how good the filtering process is or how good the resulting single document summaries are. This area explicitly was left out in this work, because the amount of required effort would have exceeded the possibilities of this master thesis. The analysis and selective improvements of individual components could potentially improve the overall summarization system, especially for the adaptive approach where different specialized algorithms can be combined to form a new single system. Similar to improvements on the summarization system, improvements and automatic approaches for the evaluation would be an interesting area for future work. Through an automatic classification and scoring process with the goal to automatically judge the results of a summarization system, new approaches and optimization techniques will be possible. This could for example include methods of automatic optimization and parameter tuning. Such a transition from a manual to an automatic evaluation approach could have an even bigger impact on system performance than individual optimizations in the current evaluation setting.

References

[1] James Allan, Rahul Gupta, and Vikas Khandelwal. Temporal summaries of new topics. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 10–18. ACM, 2001.

[2] Javed A Aslam, Matthew Ekstrand-Abueg, Virgil Pavlu, Fernando Diaz, Richard McCreadie, and Tetsuya Sakai. Trec 2014 temporal summarization track overview. In *TREC*, 2014.

[3] Regina Barzilay and Kathleen R McKeown. Sentence fusion for multidocument news summarization. *Computational Linguistics*, 31(3):297–328, 2005.

[4] Jaime Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 335–336. ACM, 1998.

[5] Tsan-Kuo Chang, Pamela J Shoemaker, and Nancy Brendlinger. Determinants of international news coverage in the us media. *Communication Research*, 14(4):396–414, 1987.

[6] John M Conroy and Dianne P O'leary. Text summarization via hidden markov models. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 406–407. ACM, 2001.

[7] Terry Copeck, Anna Kazantseva, Alistair Kennedy, Alex Kunadze, Diana Inkpen, and Stan Szpakowicz. Update summary update. In *Proceedings of the Text Analysis Conference (TAC)*, 2008.

[8] Hoa Trang Dang and Karolina Owczarzak. Overview of the tac 2008 update summarization task. In *Proceedings of text analysis conference*, pages 1–16, 2008.

[9] Günes Erkan and Dragomir R Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research*, pages 457–479, 2004.

[10] Seeger Fisher and Brian Roark. Query-focused supervised sentence ranking for update summaries. In *Proceedings of the first Text Analysis Conference, TAC-2008*, 2008.

[11] Maria Fuentes, Enrique Alfonseca, and Horacio Rodríguez. Support vector machines for query-focused summarization trained and evaluated on pyramid data. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 57–60. Association for Computational Linguistics, 2007.

[12] Surabhi Gupta, Ani Nenkova, and Dan Jurafsky. Measuring importance and query relevance in topic-focused multi-document summarization. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 193–196. Association for Computational Linguistics, 2007.

[13] Vasileios Hatzivassiloglou, Judith L Klavans, Melissa L Holcombe, Regina Barzilay, Min-Yen Kan, and Kathleen McKeown. Simfinder: A flexible clustering tool for summarization. 2001.

[14] George Hripcsak and Adam S Rothschild. Agreement, the f-measure, and reliability in information retrieval. *Journal of the American Medical Informatics Association*, 12(3):296–298, 2005.

[15] Hongyan Jing. Sentence reduction for automatic text summarization. In *Proceedings of the sixth conference on Applied natural language processing*, pages 310–315. Association for Computational Linguistics, 2000.

[16] Chris Kedzie, Kathleen McKeown, and Fernando Diaz. Summarizing disasters over time. 2014.

[17] Martin Klein and Michael L Nelson. Approximating document frequency with term count values. *arXiv preprint arXiv:0807.3755*, 2008.

[18] Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. Boilerplate detection using shallow text features. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pages 441–450, New York, NY, USA, 2010. ACM.

[19] Matt J Kusner, EDU Yu Sun, EDU Nicholas I Kolkin, and WUSTL EDU. From word embeddings to document distances.

[20] Chin-Yew Lin and Eduard Hovy. The automated acquisition of topic signatures for text summarization. In *Proceedings of the 18th conference on Computational linguistics-Volume 1*, pages 495–501. Association for Computational Linguistics, 2000.

[21] H. P. Luhn. The automatic creation of literature abstracts. *IBM J. Res. Dev.*, 2(2):159–165, April 1958.

[22] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, 2014.

[23] Richard McCreadie, Romain Deveaud, M-Dyaa Albakour, Stuart Mackie, Nut Limsopatham, Craig Macdonald, Iadh Ounis, Thibaut Thonet, and Bekir Taner Dinçer. University of glasgow at trec 2014: Experiments with terrier in contextual suggestion, temporal summarisation and web tracks. In *Proceedings of TREC 2014*, 2014.

[24] Richard McCreadie, Craig Macdonald, and Iadh Ounis. Incremental update summarization: Adaptive sentence selection based on prevalence and novelty. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 301–310. ACM, 2014.

[25] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[26] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake shakes twitter users: Real-time event detection by social sensors. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 851–860, New York, NY, USA, 2010. ACM.

[27] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513 – 523, 1988.

[28] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, et al. Storm@ twitter. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 147–156. ACM, 2014.

[29] Jenine Turner and Eugene Charniak. Supervised and unsupervised learning for sentence compression. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 290–297, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[30] Lucy Vanderwende, Hisami Suzuki, Chris Brockett, and Ani Nenkova. Beyond sumbasic: Task-focused summarization with sentence simplification and lexical expansion. *Information Processing & Management*, 43(6):1606–1618, 2007.

[31] Hao Wang, Dogan Can, Abe Kazemzadeh, François Bar, and Shrikanth Narayanan. A system for real-time twitter sentiment analysis of 2012 u.s. presidential election cycle. In *Proceedings of the ACL 2012 System Demonstrations*, ACL '12, pages 115–120, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[32] Lu Wang, Hema Raghavan, Vittorio Castelli, Radu Florian, and Claire Cardie. A sentence compression based framework to query-focused multi-document summarization. In *ACL (1)*, pages 1384–1394, 2013.

[33] Elad Yom-Tov and Fernando Diaz. Out of sight, not out of mind: On the effect of social and physical detachment on information need. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 385–394, New York, NY, USA, 2011. ACM.

[34] Siqi Zhao, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. Human as real-time sensors of social and physical events: A case study of twitter and sports games. *CoRR*, abs/1106.4300, 2011.

[35] Yun Zhao, Fei Yao, Huayang Sun, and Zhen Yang. Bjut at trec 2014 temporal summarization track. 2014.

[36] Arkaitz Zubiaga, Damiano Spina, Enrique Amigó, and Julio Gonzalo. Towards real-time summarization of scheduled events from twitter streams. In *Proceedings of the 23rd ACM conference on Hypertext and social media*, pages 319–320. ACM, 2012.

Appendices

A Summarization Frontend

As an addition to the framework implementation as described in section 3.3, a summarization frontend was built with the intention to support the inspection of different summarization systems. This frontend application is compatible to all summarization systems based on the Core framework implementation. The summarization frontend consists of two separate parts: A backend application and a web-frontend. The backend application provides a REST-API which allows clients to retrieve current data of the summarization system, such as active queries, updates of a summary (by query) and system status data. Furthermore the backend application provides several web-sockets through which clients can receive updates for queries or system-status changes in real-time. To enable such functionality, the backend application connects to the same database and pub/sub system as the summarization system. From a technology perspective, the backend application is based on the Play Framework[7] and the Scala programming language. The separate web-frontend application uses the APIs and sockets from the backend application to provide a user-interface that allows the users to inspect the status and the outputs of the summarization system in real-time. A screenshot of this interface is shown in figure 11. For any update of a query, all metadata can be inspected. Since all components of the summarization system can add arbitrary metadata to a processed item, this is a key advantage which enables an effective and time-saving approach to debug a summarization system.

[7] playframework.com

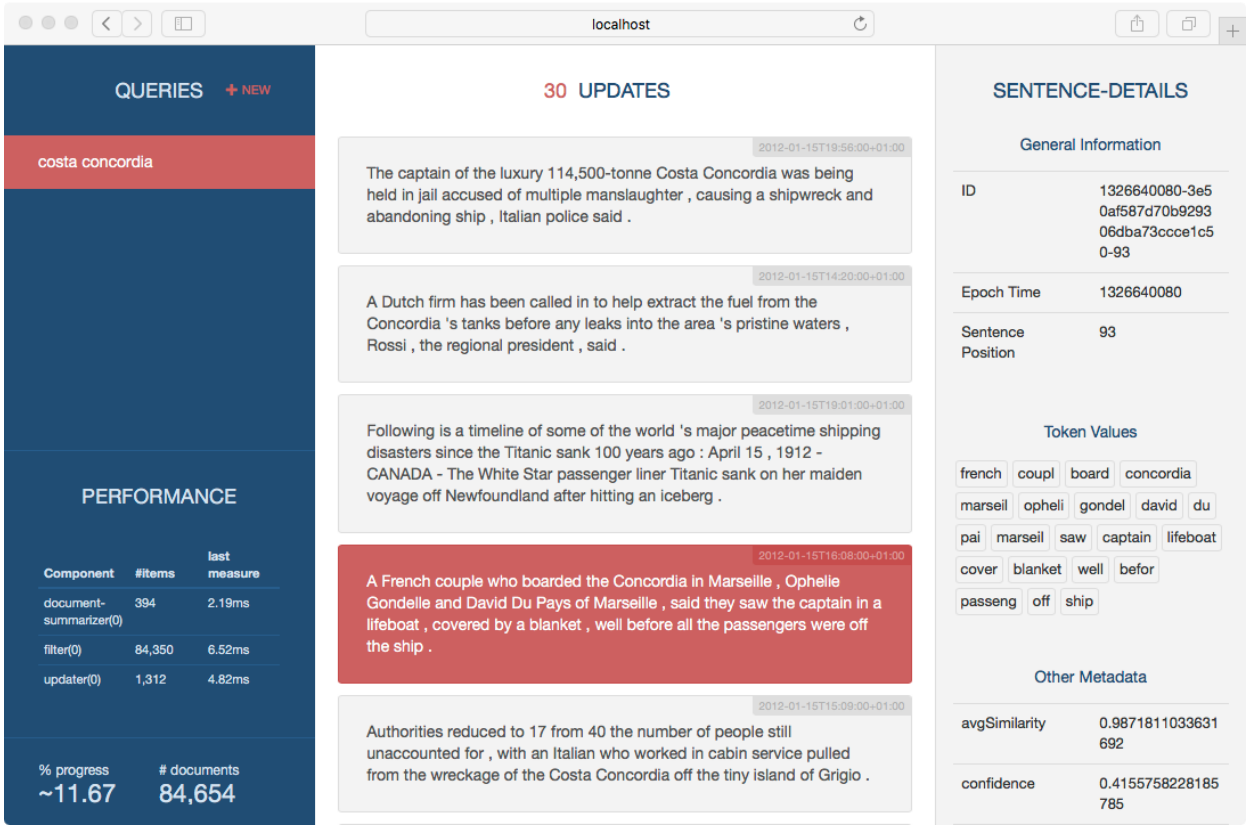

Figure 11: A screenshot of the summarization frontend. On the left hand-side a list of all queries and the system status is shown. In the middle, the updates are displayed which can be selected for further inspection. On the right hand side the inspection view shows all data and metadata of the selected update. All data updates in real-time as the summarization system continues simulating the event.

B Architectural Changes to Support an Unlimited Number of Queries

For the framework design described in section 3, there is one potential bottleneck. A large number of queries at the same time can increase the overall computation time for an individual item. The framework was explicitly designed for parallelization and stream processing to enable real-time summarization with low latency. However, if there is a large number of queries, a bottleneck within the Filter emerges. It receives a document as an input and has to decide whether the document is relevant for one or more of the target queries. The number of comparisons therefore increases linear with the number of target queries. If there are a lot of queries, the potential delay to emit an update for the last query is much higher than for the first query. Real-time properties are not met in this case, since there is no upper-bound for the delay with an unlimited number of queries.

The following architectural changes resolve this issue. As an additional component, the ”Pre-Filter” is introduced, which is placed before the filter. It receives a stream of HtmlDocumentRaw instances as an input and outputs a stream of HtmlDocumentRaw / Query pairs. Each Pre-Filter instance contains a part of the whole query list. All of the Pre-Filter instances receive every HtmlDocumentRaw instance. For each query in the query list they output the HtmlDocumentRaw / Query pair. Items from this output stream are then randomly distributed to a Filter instance which in this case only checks that the document is relevant for the single query, meaning that all checks for a single document can occur in parallel. The rest of the architecture is untouched. A visualization of this solution is shown in figure 12.

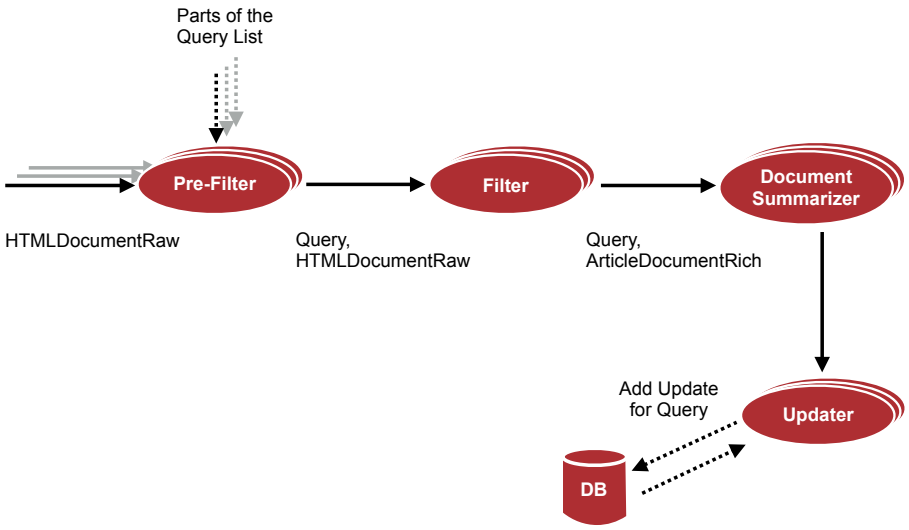

Figure 12: A visualization of a slightly modified framework architecture to resolve a potential bottleneck with an unlimited number of queries.

C Configuration Switches of the Adaptive Approach

In the following two figures 13 and 14, the algorithm switches and the amount of relevant documents over time are shown. Relevant documents are identified by the restrictive filter algorithm, therefore the numbers are rather small. However, for both events, a large amount of news-articles is processed by the summarization system (Quran Burning Protests: 483,992; Queensland Floods: 549,250).

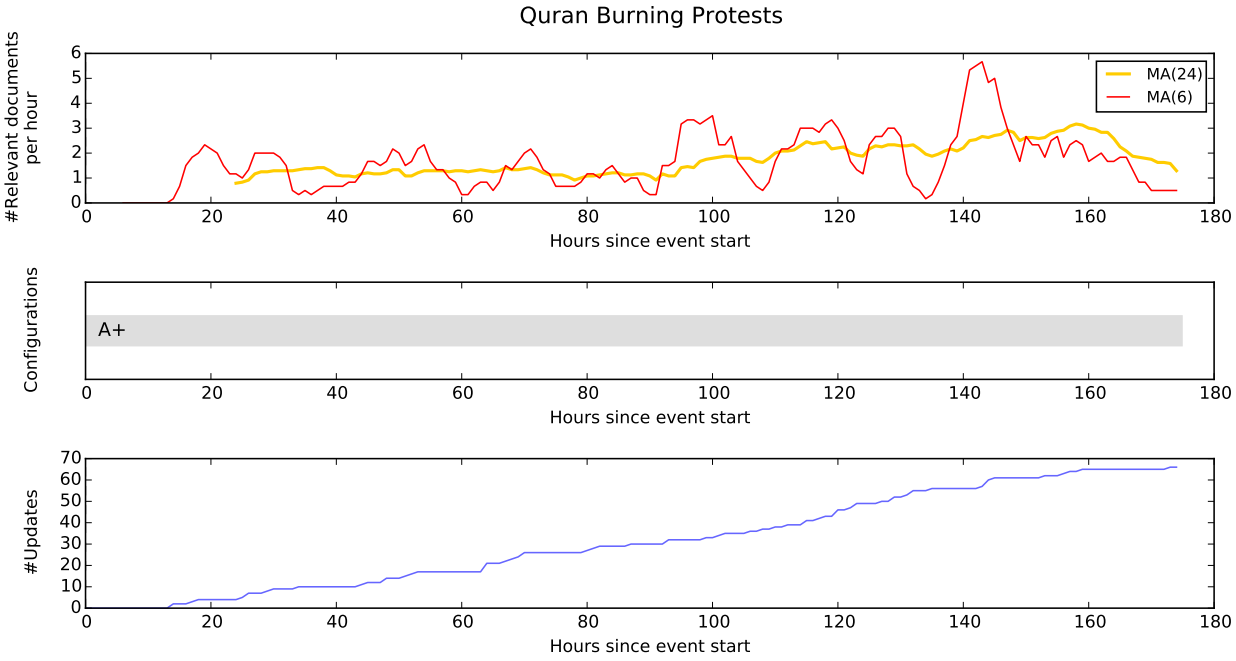

Figure 13: Configuration switches of the adaptive approach for the Afghanistan Quran Burnings event. The amount of news-articles covering this event is consistently low, therefore no configuration switches are performed.

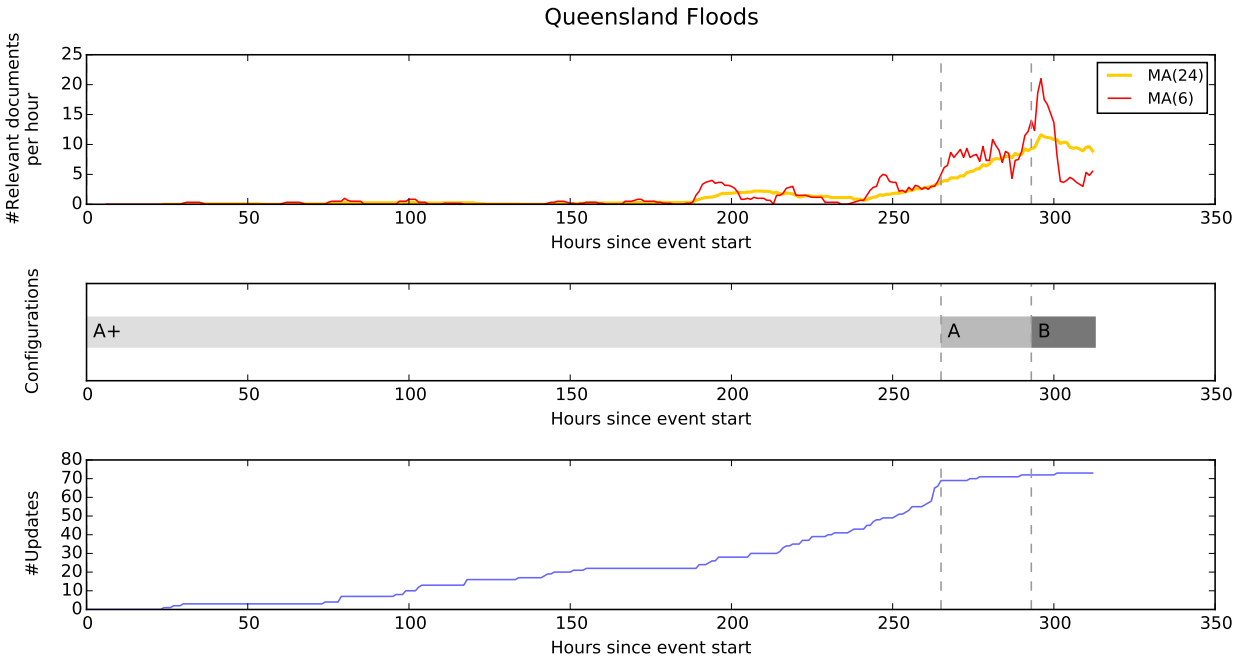

Figure 14: Configuration switches of the adaptive approach for the Queensland Floods event. After the beginning of the event, there is a relatively low interest in the event that lasts for several days. A sudden spike in news-coverage before the end of the event leads to configuration switches up to the most-restrictive configuration (B).

D Nugget-Matching Interface

Nugget-matching is the central part of the TREC-TS 2014 evaluation. Human annotators try to match the top-60 updates of a summary that was created by a summarization system for a specific event against nuggets of information (gold standard). The matching is then used to calculate the resulting scores of the system, describing its precision, recall and timeliness. Since the nugget-matching interface of TREC-TS 2014 was not available, an own version of this interface was developed as part of this work. The new interface is optimized for annotator user experience, allowing the annotator to be as efficient as possible and to always have a clear view on the current task. One annotation task consists of the top-60 updates of the summarization system for an event and all the nuggets (between 35 and 226). A good user experience therefore is required to ensure that the annotation study is feasible. In figure 15 a screenshot of the nugget-matching interface is shown. The annotator drags nuggets onto updates to create nugget-matches and can also mark updates as irrelevant for the topic by clicking the ”X”-button in the top-right corner of an update. In this specific example, the information of the first nugget is included in the second update. After dragging the nugget onto this update, a text-selection window opens. This is shown in figure 16. Within this text-selection window, the annotator selects the part of the update-text which contains the nugget-information. After the annotator confirms the selection, the nugget-matching is added. The text selection information is used in the evaluation to ensure that short and precise updates which do not contain any other non-relevant information lead to a better score than long updates with only a small portion of relevant information. After an annotator is finished with all matchings, clicking the finish button in the top-right corner of the web-page locks the task for any further editing.

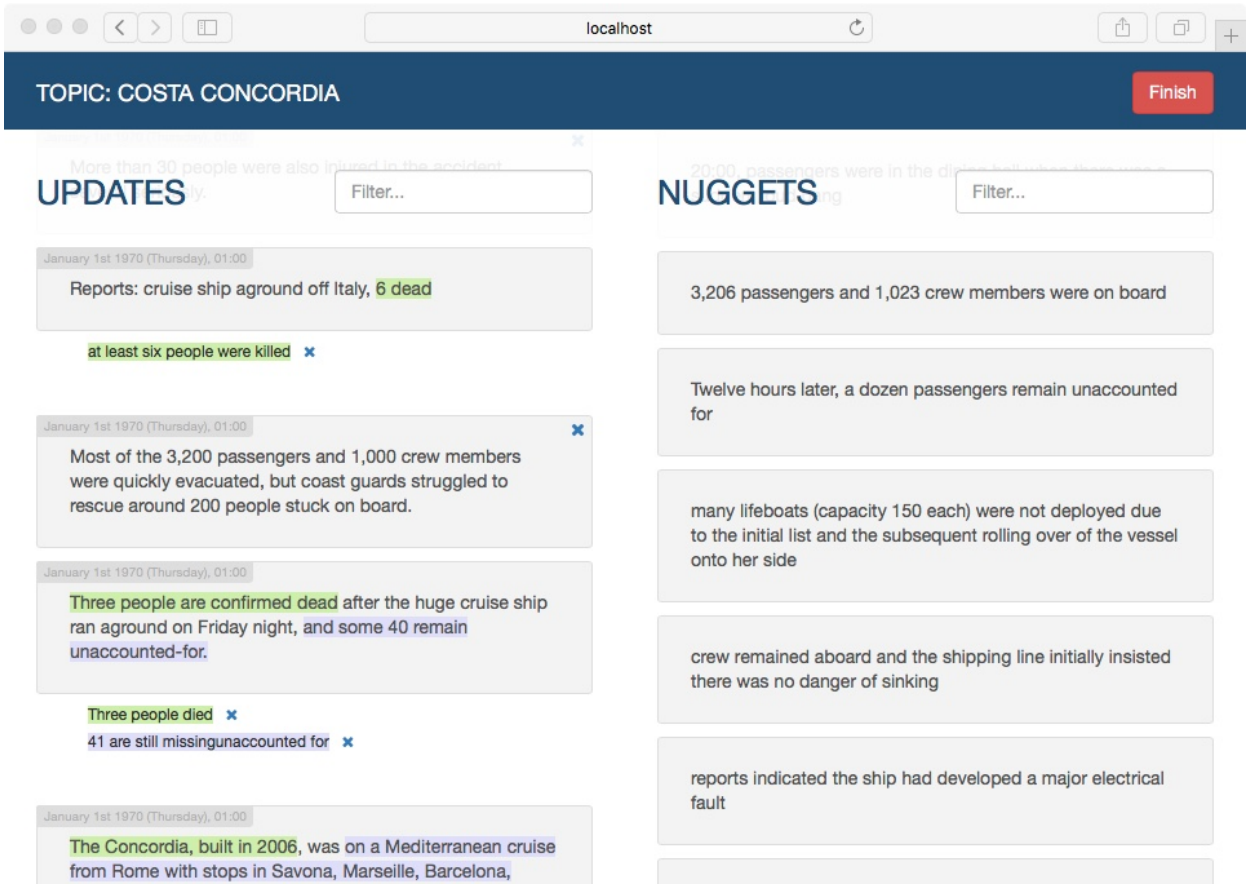

Figure 15: The nugget-matching web interface. Annotators drag nuggets from the right side onto updates on the left side. All matching nuggets of an update are listed below the update with the matching part of the update-text being color-coded. Updates can be marked as irrelevant for the topic by clicking the "X"-button in the top-right corner of an update.

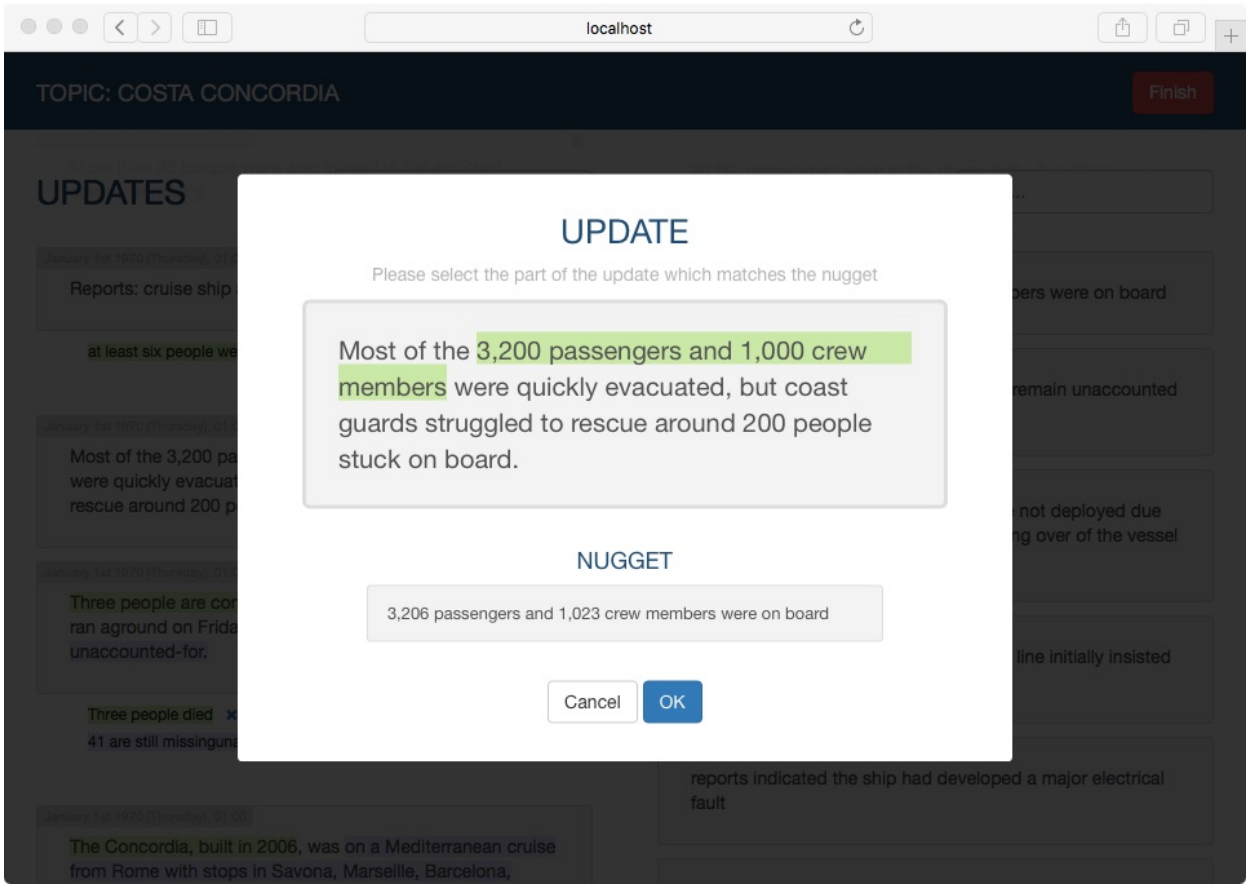

Figure 16: Text selection for a new nugget-matching annotation. When the annotator drags a nugget onto an update, the text-selection view opens. The annotator then selects the part of the update that contains the information of the nugget. After the annotator confirms the selection, the nugget-matching is created and the nugget is listed beneath the update as a match.