# Multi-Query Optimization for Semantic News Feed Query

Andinet Assefa
Department of Computer Science
Dire Dawa University, 1362, Dire Dawa, Ethiopia
andbeyes@yahoo.com

Fekade Getahun
Department of Computer Science
College of Natural Sciences
Addis Ababa University, 1176, Addis Ababa, Ethiopia
fekade.getahun@aau.edu.et

## ABSTRACT
Recently, a window-based and semantic-aware feeds querying operators to retrieve set of news items satisfying a given condition has been proposed. The use of semantic information improves the relevance of query result at the cost of degrading the efficiency of the system. To benefit from query execution on semantic information while keeping efficiency of the system, we have proposed a multi-query optimization strategy. The proposed MQO strategy accepts semantic news feed queries and examines the relationship that exists between windows defined on queries. Then, a MQE chain is generated based on the window relationship examined for the efficient manipulation of queries. Whenever a new query arrives, it is added to its appropriate chain. To validate the proposed approach, we have developed a prototype and the experimental results show that the use of our approach improves significantly the performance of the system.

## Categories and Subject Descriptors
H.2.4 [**Database Management**]: Systems - *Multimedia Databases, Query Processing*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval - *Information Filtering, Query Formulation*; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation – *Parallelism and Concurrency*.

## General Terms
Algorithms, Management, Performance

## Keyword
SNF Query, NAT, Windows Similarity Measure, MQE Chain Generator, Window Relationship

## 1. INTRODUCTION
RSS is an XML-based feed format designed for content distribution and it contains headlines and descriptions of information. It has been used in news industry to disseminate news items extracted from user favorite feed sources and consumed using RSS reader or news aggregator [1].

RSS news items are text-content rich, semantically heterogeneous, dynamic XML element and their contents are dependent on the authors' style of writing and streamed in an asynchronous and pull strategy. For efficient retrieval of news feed, retrieval functionality based on concept similarity was proposed in [2]. The authors proposed semantic-aware and window-based RSS querying operators to retrieve a set of news items satisfying a given condition while considering semantic information. It is known that semantic based query processing strategy improves the relevance of query retrieval at the cost of degrading the efficiency of the system. In stream system queries have long-running nature and are relatively permanent. Thus, devising an approach that amortizes cost using multi query optimization is imperative. In [3, 4, 5, 6, 7, 8, 9] multi query optimization is recommended as multiple concurrently active queries might share resources. To benefit from this, in this research, we have proposed a multi query optimization approach that considers the semantic similarity between queries and their defined windows.

The remainder of this paper is organized as follows. In Section 2, motivational scenario is presented. Section 3 discusses related works. Section 4 defines basic concepts used in our approach. Section 5 details our optimization approach. Section 6 presents developed prototype and experimental results. Finally, Section 7 concludes this study and draws future research direction.

## 2. MOTIVATING SCENARIO
*EasyRSSManager* [17], a semantic-aware RSS reader having a window-based RSS query components, is installed on a proxy server to process users` queries. Table 2-1 shows some of the registered users` queries. Each query is scheduled to be executed at a particular time on a given source. Queries $Q_1$ and $Q_2$ are semantically similar defined on the same source with relatively different window[1] boundary. Even though, query $Q_3$ is neither similar to $Q_1$ nor to $Q_2$, but both share the same feed source. Similarly, queries $Q_4$ and $Q_6$ are defined on the same feed source. Thus, rather than executing these queries independently, it is advisable to execute queries that share the same resources (feed source and window) simultaneously to increase the efficiency of query processor. In addition, windows defined for $Q_1$ and $Q_2$ are included in the window defined for query $Q_3$. What if we execute the queries only on one window? Doing so reduces network access time and resource/memory utilization. This is also the case for $Q_4$ and $Q_6$. In addition, queries $Q_1$ and $Q_2$, are semantically similar (i.e., similar *search value*, *attribute, and operator)* what if we execute only for $Q_1$ and apply the result for query $Q_2$ too?

This scenario shows the need to execute several queries having the same feed source on a single sharable window. Identifying semantically similar queries helps in saving computation cost and for the betterment of the performance of the system in general.

Motivated by this, we have proposed a multi query optimization approach which considers the semantic similarity of queries and windows for sharing resources among set of semantic news feed queries.

[1]Window: it is a collection of elements defined on a given feed source having start and end boundary time.


MEDES'12, October 28-31, 2012, Addis Ababa, Ethiopia.

**Table 2-1: List of semantic news feed queries registered by users**

| Query | Search Value | Attribute | RSS Operator | Feed Source | Window Boundary Time(In GMT) | |
|---|---|---|---|---|---|---|
| | | | | | Start | End |
| $Q_1$ | Car crush killed two people in Iran | Title | Select | BBC | 01:30:10 | 18:30:01 |
| $Q_2$ | Dead in Tehran car accident | Description | Select | BBC | 00:30:30 | 15:00:00 |
| $Q_3$ | Euro 2012 football latest score | Description | TopK | BBC | 00:05:30 | 19:05:00 |
| $Q_4$ | Iran nuclear programme | Title | Join | REUTERS, FRANCE 24 | 00:00:05 | 16:00:30 |
| $Q_5$ | South Sudan border town Bentiu bombed | Description | TopK | CNN | 01:00:00 | 18:05:00 |
| $Q_6$ | US president surprise world bank nominee | Title | Select | REUTERS | 03:10:00 | 15:30:10 |

The contribution of this paper is summarized as follows.

- Applying semantic similarity measure as a tool to optimize multiple queries accessing XML stream,
- Introducing semantic similarity measure for windows,
- Introducing the concept of window sharing technique into feed query processing systems, and
- Introducing the concept of chain based Multi-Query Execution (MQE) strategy for XML stream query processing system.

Finally, to the best of our knowledge, this is the first work on multi query optimization technique for semantic news feed query processing.

## 3. RELATED WORKS

In streaming applications, a number of research works have been conducted on the idea of MQO [3, 4, 5, 10, 12, 13]. Among these works, we discuss some of the most related to our approach. In [10], a multi-query optimization with schedule synchronization has been proposed in which synchronizing the re-execution times of similar queries takes advantage of computation sharing. However, the approach does not consider periodic tasks in stream applications and disregard dissimilar queries defined on the same feed sources. In [12] a paradigm for sharing of window join queries has been proposed in which slicing window states of a join operator into fine-grained window slices and forms a chain of sliced window joins. This paradigm enables to push selections down into chain and flexibly select subsequences of sliced window joins for computation sharing among queries with different window sizes. In streaming applications like publish/subscribe systems massive queries exist with different window definition and are evaluated continuously. Hence, trying to generate slice of window states for such cases are not computationally efficient. In addition, in [13] a massively Multi-Query Join Processing technique for processing a large number of XML stream queries involving value joins over multiple XML streams and documents is proposed. This technique enables the sharing of representations of input to multiple join, and the sharing of join computation.

MQO by partial aggregation continuous queries (ACQ) is also considered in [7, 14, 15]. The Panes scheme in [14] splits slide into equal sized fragments that are to be processed using the partial aggregation operator. Paired windows in [7] improve Panes by splitting slide into exactly two fragments for the purpose of minimizing the processing needed at the final-aggregation level. In [15] *Weave Share*, a cost-based multiple ACQs optimizer, that exploits weaveability to optimize the shared processing of ACQs is proposed.

Though these works contribute to stream query processing; to the best of our knowledge, none of the existing approaches consider a shared execution of stream queries based on semantic similarity between queries and their respective windows. In this paper, semantic similarity measure is used to identify the similarity between queries and it ensures a highly shared computation among a set of queries. In addition, we have introduced a formal semantic similarity measure for windows to identify window relationship types for effective sharing among multiple queries.

Shared execution addressed in [16] focuses on execution of multiple queries on the larger window first (LWF) and as alternative to execution on smaller windows first (SWF). However, overlapping windows in streaming applications should be treated wisely other than LWF and SWF strategies. Our approach handles the case of shared execution on overlapping windows. Moreover, our approach brings efficient way of sharing a window among a set of queries defined on the same source whether they are semantically similar or not. In addition, our approach handles periodic queries and provides the result of user schedule aware execution of multiple queries.

To realize the MQE objectives, our *MQE* paradigm uses a chain based data structure for faster and efficient storage supported by thread based execution. The execution is guided by a formal *MQE Rules*.

## 4. PRELIMINARIES

In this section, we provide definitions of basic concepts relevant to the realm of this paper.

**Definition 4-1 [Semantic RSS Operators]**
Semantic RSS operators [2] are query operators used to retrieve set of news items satisfying a given condition while considering semantic information. They are categorized into three: extraction, set membership and merge operators. Each of these operators accepts a set of windows and a supportive parameter set. The extraction operators are dedicated to retrieve data from the stream and it includes *Selection*, its extension *TopK*, and *Join*. The set membership operators accept two windows defined on streams and return the result of *union*, *intersection* or *difference*. The *merge* operator accepts two windows and returns the result of merging (i.e. putting them together in a given pattern) w.r.t. a given merging rule.

**Definition 4-2 [Semantic News Feed Query]**
A *semantic news feed* (SNF) *query* [2] is a persistent query Q defined on a given source S, having a predicate P. And a user SNF query is stored permanently as part of his preferences.

Given a Query $Q$, $Q.S$, $Q.P$, and $Q.\theta$ returns the source, predicate and operator respectively associated to the query. The predicate P has attribute A, RSS operator $\theta$ (extraction, set membership or merge) and search value T).

**Definition 4-3 [SNF Query Processing System]**
SNF query processing system is a time aware XML data stream processing system. The stream source the sends data in either asynchronous or synchronous manner using a push or pull strategy; and the query processer evaluates the query continuously on the incoming stream of data.

# 5. THE PROPOSED APPROACH

In this section, we have presented the detail of the proposed approach. The architecture of SNF query processing engine with a MQO component to handle the proposed optimization technique is shown in Figure 5-1. In stream processing system, user queries are relatively permanent and thus it is stored as user preference in *profile database.* Then, the *MQO Manager* component in the query engine reads queries from the profile database and prepares a MQE plan. The *Query Similarity Evaluator*, *Window Relationship Type Identifier* and *MQE Chain Generator* are its sub-components.

Generally, given a set of news feed queries; the proposed system performs MQO in three main steps.

1. Evaluates the semantic similarity of queries to generate a MQE plan for a set of semantically similar queries,
2. Identifies the window relationship type (c.f. detail in Section 5.2) that exists between queries, and
3. Generates MQE chain of queries (c.f. detail in Section 5.3) represented in singly linked list for efficient and faster execution of multiple queries at run time.

Once these three steps are done, the *MQO Manager* saves the multi query execution plan to the profile database for later use.

After the optimization is conducted, the final step in the query engine is multi query execution. The *Query Executer* reads a multi query execution plan generated by *MQO Manager* from the profile database and uses MQE rules detailed in Section 5.4 to execute multiple queries together on shared windows generated by *Window Generator* component.

The rest of this Section discusses the detail of these basic phases of MQO.

## 5.1 Query Similarity Evaluation

In stream systems, queries are relatively persistent and changes seldom thus are registered persistently in a database. It is likely that the profile database contains large number of queries with possible redundancy in query definition. Processing such redundant queries degrades the performance of the system. To avoid such anomalies, the *Query Similarity Evaluator* sub-component in the query engine applies semantic similarity to examine the extent to which user queries are similar.

Given two pair of queries $Q_1$ and $Q_2$, the semantic similarity in between is denoted as *SimQuery*($Q_1$, $Q_2$) and is defined as:

$$SimQuery(Q_1, Q_2) = \begin{cases} 0 & \text{if } Q_1.S_1 \neq Q_2.S_1 \text{ and } Q_1.\theta_1 \neq Q_2.\theta_1 \\ Sim(Q_1.P, Q_2.P) & \text{otherwise} \end{cases}$$

Two queries are different if either the source or operators are different; otherwise, their similarity is dependent on the similarity between their corresponding predicates which is computed using Algorithm 5-1. The algorithm accepts search terms, operators and attributes of two queries and returns a semantic similarity computed with vector based similarity method. It computes the angle separating the vectors representing each predicate (line 7 and 8) and similarity is computed using cosine of the vectors as shown in line 9.

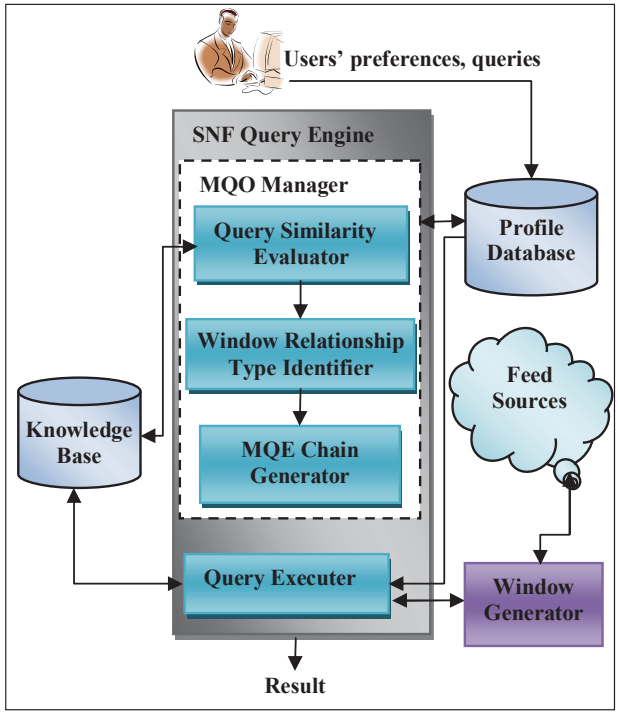


**Figure 5-1 Architecture of SNF query processing engine**

If two queries are equivalent (i.e., their query similarity is close to 1), we filter out one as a representative and apply the execution result to the other query. Thus, query processing cost will be minimized. The query processing cost is a sum of *network access cost*, *memory access*, and *execution cost*. Hence, the decision to evaluate one of semantically similar queries reduces network access time and avoids redundant holding of the same data in memory which directly reduces the query processing cost.

**Example 5-1 [Equivalent Queries]**
Referring to queries $Q_1$ and $Q_2$ in Table 2-1, they are semantically equivalent. So the result of processing query $Q_1$ can be applied to $Q_2$ or vice versa.

On the other hand, given dissimilar queries defined on the same source, it is possible to minimize query execution cost by sharing network access cost. This can be achieved by identifying the relationship between windows in queries as detailed in the next sub-section.

## 5.2 Window Relationship Type Identification

Given queries defined on the same feed, it is likely that the queries might be defined on similar or related window boundary. Such queries can be executed on a single shared window simultaneously.

Generating a window defined on the same source and sharing it

among a set of queries have the following advantages:

- It significantly decreases the network access time (NAT[2]) since a data accessed once will be used by a set of queries.
- Efficient utilization of memory space. Sharing a single window among a set of queries eliminate redundant storage of same data in memory.

**Algorithm 5-1 Query Similarity Evaluator**

```
    Input:
1.  ST1,ST2: String  // Search terms
2.  AT1,AT2: String  // Attributes
3.  OP1,OP2: String  // Operators
4.  KB: Knowledge Base
    Intermediate:
5.  V1, V2: Vector
    Output:
6.  QueSim: Decimal
    Begin:
7.   V1= BuildVectorSpace(AT1, OP1, ST1)
8.   V2= BuildVectorSpace(AT2, OP2, ST2)
                  // vector space generation
9.   QueSim = Cosine(V1,V2)
             // the cosine between vectors
13. Return⟨QueSim⟩
    End
```

To identify the exclusive relationship type between windows, we measure the similarity between windows using Definition 5-1.

**Definition 5-1 [Windows Similarity Measure]**
Given two windows $W_1$ and $W_2$, with start boundary time $S_1$, $S_2$ and end boundary time $E_1$, $E_2$ each defined in minutes respectively, the similarity between $W_1$ and $W_2$ is denoted as $SimWindow(W_1, W_2)$ and returns a vector $v$ having normalized start, end and end to start time difference. It is formalized as follow:

$$SimWindow(W_1, W_2) = v = (S, E, D)$$

Where:

$$S = \frac{S_1 - S_2}{60} \quad \text{normalized start time difference}$$

$$E = \frac{E_1 - E_2}{60} \quad \text{normalized end time difference}$$

$$D = \begin{cases} \frac{E_1 - S_2}{60} & if\ S_2 > E_1 \\ \frac{E_2 - S_1}{60} & if\ S_1 > E_2 \end{cases} \quad \text{normalized end to start time differnece}$$

A 60 minute representing an Hour is used as normalization value. The normalized values are used later in identifying the magnitude of the windows similarity vector. Accordingly, window relationship type is identified after building a row vector of similarity between each windows of a given set of queries in profile database.

Based on Definition 5-1, a window similarity value of 0 for all S, E, and D means that the two windows are defined exactly on the same time boundary. However, it is likely that two windows might be defined with some time variations and yet we need to take them as similar. For example, though windows defined for queries $Q_1$ and $Q_5$ in Table 2-1 has a 30 minutes time variation, we may need to tolerate the variations. To deal with such variations, we define minimum and maximum similarity threshold values (i.e, *minTR* and *maxTR*); and if the window similarity value falls within this range, then the two time definitions are similar.

Therefore, given an equality threshold *TEqual*, *minTR* and *maxTR* are defined as:

- $minTR = 0 - TEqual\ and$
- $maxTR = 0 + TEqual$

Based on this, window relationship type is given in Definition 5-2.

**Definition 5-2 [Window Relationship Type]**
Given two queries $Q_1$ and $Q_2$ defined on associated windows $W_1$ and $W_2$, the relationship between $W_1$ and $W_2$ is exclusive which is either *Similar, Includes, Included-In, overlaps or Disjoint* (as shown in Figure 5-2) identified using two threshold values *minTR* and *maxTR*. It is denoted as $WinRelation(W_1, W_2)$ and defined as follows:

1. $W_1$ *Similar* $W_2$ denoted as $W_1 \equiv W_2$: if the boundary time definitions of $W_1$ and $W_2$ falls within *minTR* and *maxTR* as shown in Figure 5-2(D). It is formalized as follows:

   $\text{WinRelation}(W_1, W_2) = \text{'Similar'}$

   $if(minTR \leq S \leq maxTR\ and\ minTR \leq E \leq maxTR)$
2. $W_1$ *Includes* $W_2$ denoted as $W_1 \supset W_2$: if the window boundary time of $W_2$ falls within the window boundary time definition of $W_1$ as shown in Figure 5-2 (A). It is formalized as follows:

   $WinRelation(W_1, W_2) = 'Includes'$

   $if(S < minTR\ and\ minTR \leq E \leq maxTR)\ or\ (minTR \leq S \leq maxTR\ and\ E > maxTR)\ or\ (S < minTR\ and\ E > maxTR)$
3. $W_1$ *Included-In* $W_2$ denoted as $W_1 \subset W_2$: if $W_2$, includes $W_1$ as shown in Figure 5-2 (B)
4. $W_1$ *Overlap* $W_2$ denoted as $W_1 \Cap W_2$: $W_1$ and $W_2$ overlap (as shown in Figure 5-2 (E)) if the difference between starts and also ends boundary values are below the minimum or above the maximum threshold values. It is formalized as follows:

   $\text{WinRelation}(W_1, W_2) = \text{'Overlap'}$

   $if\ (S < minTR\ and\ E < minTR)$ or $(S > maxTR\ and\ E > maxTR)$
5. $W_1$ *Disjoint* $W_2$ denoted as $W_1 \rhd\lhd W_2$: $W_1$ and $W_2$ disjoint if they are defined on different window boundary time (i.e., the End to Start boundary time difference D is close to zero). It is formalized as follows:

   $\text{WinRelation}(W_1, W_2) = \text{'Disjoint'}$

   $if(D < minTR\ or\ D = 0)$

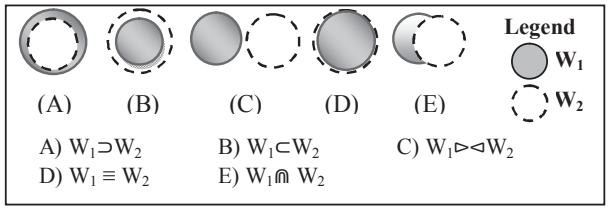


**Figure 5-2 Topology for window relationship type**

[2] NAT is the time elapsed while retrieving feed items from a specified feed source.

Algorithm 5-2 identifies the window relationship type that exists between queries. The algorithm accepts two windows, associated feed sources and equality threshold as inputs. In line 7, the algorithm compares feed sources and if they are defined on same source then, in line numbers 8 and 9, it computes windows similarity value which is a vector and extracts values of S, E, and D respectively based on Definition 5-1. In line 11 – 24, it identifies the window relationship type existing between the two windows based on Definition 5-2. Finally, if the feed sources are dissimilar, the window relationship type is Disjoint as shown in line number 26.

Table 5-1 shows the window relationship type identified between queries shown in Table 2-1.

**Table 5-1 Window relationship type for queries in Table 2-1 identified using Algorithm 5-2**

| Query | $Q_1$ | $Q_2$ | $Q_3$ | $Q_4$ | $Q_5$ | $Q_6$ |
|---|---|---|---|---|---|---|
| $Q_1$ | ≡ | ⊃ | ≡ | ⊳⊲ | ⊳⊲ | ⊳⊲ |
| $Q_2$ | ⊂ | ≡ | ⊂ | ⊳⊲ | ⊳⊲ | ⊳⊲ |
| $Q_3$ | ≡ | ⊃ | ≡ | ⊳⊲ | ⊳⊲ | ⊳⊲ |
| $Q_4$ | ⊳⊲ | ⊳⊲ | ⊳⊲ | ≡ | ⊳⊲ | ⋒ |
| $Q_5$ | ⊳⊲ | ⊳⊲ | ⊳⊲ | ⊳⊲ | ≡ | ⊳⊲ |
| $Q_6$ | ⊳⊲ | ⊳⊲ | ⊳⊲ | ⋒ | ⊳⊲ | ≡ |

Once the windows relationship types have been identified, the next step in the query engine is MQE chain generation presented in the next sub-section.

**Algorithm 5-2 Window Relationship Type Identifier**

```
    Input:
1.      W1,W2:Window       // Windows of queries
2.      FS1,FS2: String    // Feed sources of queries
3.      TEQ: Decimal       // Equality threshold
    Intermediate:
4.     minTR, maxTR, S, E, D :Decimal
5.      V: Row-Vector
    Output:
6.      WinRel: String
    Begin:
7.  IF(FS1 = FS2) Then              // Same feed source thus apply Definition 5-2
8.      V = SimWindow(W1, W2)               // using Definition 5-1
9.      S = V[0],E = V[1],D = V[2]          // start, end and start to end time difference
10.     minTR = -TEQ, maxTR = TEQ           // minimum and maximum threshold
11. IF((isBetween(S))AND(isBetween(E))) Then
12.       //isBetween checks if the given parameter is between minTR and maxTR
13.          WinRel = "Similar"
14. Else IF((S<minTR AND isBetween(E))OR(isBetween(S) AND E>maxTR)OR(S<minTR AND E>maxTR)) Then
15.          WinRel = "Includes"
16. Else IF((S>maxTRAND isBetween(E))OR(isBetween(S) ANDE<minTR)OR(S>maxTR AND E<minTR)) Then
17.          WinRel = "Included-In"
18. Else IF(D<minTR OR D = 0) Then
19.          WinRel = "Disjoint"
20. Else IF(((S<minTR)AND(E<minTR))OR((S>maxTR)AND(E>maxTR))) Then
21.          WinRel = "Overlap"
22. Else
23.          WinRel = "Disjoint"
24. End IF
25. Else         // Different feed sources thus disjoint
26.       WinRel = "Disjoint"
27. End IF
28. Return⟨WinRel⟩
    End
```

## 5.3 MQE Chain Generation

Before the actual execution of multiple queries, appropriate data structure should be used for efficient utilization of resources and faster execution of queries. In this work, we used chaining to structure queries based on their window relations. A chain is a singly linked list consists of queries with related window relationship type as defined in Definition 5-3.

**Definition 5-3 [Chain of X]**
Given n queries, $Q_1$, $Q_2$, $Q_3$ ... $Q_n$, and windows relationship type X, Chain of X, is set of queries related with relationship X. Formally, it is denoted as:

$$C_X = \{Q_1, Q_2, Q_3, \ldots, Q_n\}$$

Where:

- X= *WinRelation*($W_i$, $W_j$), *WinRelation* returns window relation type between $W_i$ and $W_j$,
- $W_i$, $W_j$ are windows associated to $Q_i, Q_j \in \{Q_1, \ldots, Q_n\}$

Accordingly, queries in a given chain will be executed on a shared window according to the MQE rules given in Section 5.4. Algorithm 5-3 is devised to generate a MQE chain. It accepts a collection of windows relationship types identified using Algorithm 5-2 and list of queries as input. In line 5-19, the algorithm generates chains of *Similar, Includes, Included-In, Overlap* and *Disjoint* type represented as a linked list. The first element in each of the chain is used as a representative element and all the elements in the chain are ordered in descending order based on their window size. Hence, a window generated for the first element in the chain can potentially be used to execute the rest of the elements in the chain.

**Algorithm 5-3 Chain Generator Algorithm**

```
     Input:
1.   WinRel: Collection-Window   // set of Window relation type
2.   Query: Array               // a set of queries
     Intermediate:
3.   SimChain, IncChain, IncInChain, OverChain, DisChain: LinkedList   // linked lists
     Output:
4.   Chain: Array      // set of chains
     Begin:            // generation of  chain
5.   For i=0 To rowLength(WinRel)
6.      For j=0 To columnLength(WinRel)
7.      IF (WinRel[i][j]= "Similar") Then
8.      addToChain(simChain, Query[i], Query[j],desc) //addToChain links queries to chain
9.      Else IF (WinRel[i][j] = "Includes") Then
10.     addToChain(IncChain, Query[i], Query[j],desc)
11.     Else IF (WinRel[i][j] = "Included-In") Then
12.     addToChain(IncInChain, Query[i], Query[j],desc)
13.     Else IF (WinRel[i][j] = "Overlap") Then
14.     addToChain(OverChain, Query[i], Query[j],desc)
15.     Else
16.     addToChain(DisChain, Query[i], Query[j],desc)
17.     End IF
18.     End For
19.  End For
20.  Return ⟨SimChain,IncChain,IncInChain,OverChain,DisChain⟩
     End
```

## 5.4 Multi Query Execution Rules

In order to execute multiple queries together, a rule based multi-query execution strategies supplemented with thread based execution have been proposed. The following are the list of MQE rules:

**MQE Rule 1.** If queries $Q_1$ and $Q_2$ are semantically similar (equivalent), then apply the execution result of one to the other.

**MQE Rule 2.** If a query Q is a member of $C_{Disjoint}$ i.e., in *Disjoint* chain, then execute Q independently.

**MQE Rule 3.** If a query Q is a member of $C_{Includes}$, $C_{Similar}$ or $C_{Included\text{-}In}$ i.e., in *Includes, Similar* or *Included-In* chain, then identify the largest chain containing *Q* and generate a window for the first element in the chain and apply MQE Rule 1. If there are more than one chain with equal size containing Q, apply MQE Rule 4 to execute Q.

**MQE Rule 4.** If there are multiple equal sized chains i.e. $C_{Includes}$, $C_{Included\text{-}In}$ or $C_{Similar}$ containing Q as their first element then execute Q on the union of the chains.

**MQE Rule 5.** If queries $Q_1$ and $Q_2$ are members of $C_{Overlap}$ with windows $W_1$ and $W_2$ respectively and if $Q_1$ is scheduled earlier than $Q_2$ then execute $Q_1$ on $W_1$ and $Q_2$ partially on the intersection of $W_1$ and $W_2$.

These MQE rules are responsible to decide on which query to execute. The rule uses the chain of X generated using the result of Algorithm 5-3 (where X is ether *Similar, Includes, Included-In, Overlap* or *Disjoint* semantic relationships between queries).

The *Query Executer* subcomponent in the query engine employs these MQE rules to execute a set of queries considering semantic information on a shared window generated by the *Window Generator* component.

# 6. EXPERIMENTATION

To realize the proposed approach, we have developed desktop prototype using C#. The prototype allows a user to subscribe their preferences and define queries, schedule queries and manage query schedule using query scheduler component, and auto execution of queries according to schedule. Figure 6-1 shows a sample user interface that allows user to define semantic news feed query and save query permanently. In addition, the user provides a schedule using the smaller window displayed on the bottom left of query definition interface. Figure 6-2 shows the execution result of user query.

The prototype is used for performance tests of the proposed MQE approach on a variety of query load. The tests were conducted on

personal computer with a single Intel Celeron CPU (2.2 GHz speed), 3 GB RAM and with a network connection speed of 54.0 Mbps. In addition, we have used WordNet for the knowledge base. We have demonstrated the performance of the system based on two experiments. The first experiment is done to show how sharing a single window for multiple queries decreases the Network Access Time (presented in sub-section 6.1). The second experiment is done to show a performance increase of our proposed approach of MQE based on the window relationship type existing between set of queries (presented in sub-section 6.2).

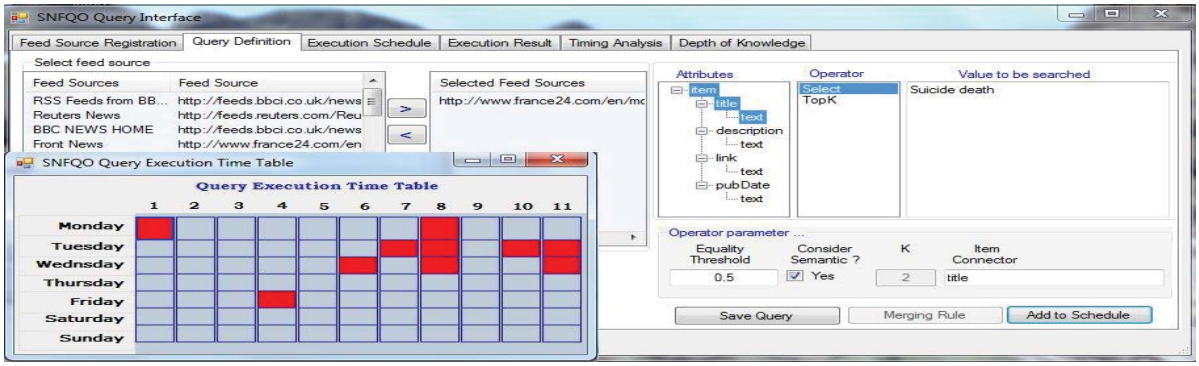


**Figure 6-1: Semantic news feed query definition window**


**Figure 6-2: User query execution result displaying window**

## 6.1 Network Access Time (NAT)

This experiment is done to show how queries defined on the same feed source share a window in determining whether they are semantically similar or not. The experiment is performed on 50 queries and Figure 6-3 shows the network access time required to generate a window for multiple queries in comparison with generation of windows for individual query (linear network access).

We made the system to process 50 queries at a time. The execution is on a pair [D, Y], where D denotes disjoint queries and Y denotes other relations i.e. *Similar, Includes* or *Overlap*. Accordingly, the black line shows the network access time required to generate windows of the 50 queries. The graph shows that the network access time is almost constant independent of the number of queries. However, using our shared network access approach, the result shows a decrease in time as the number of disjoint queries decreases in the pair [D, Y]. Initially the pair of queries was [50, 0], in which all of the query set are disjoint; and the corresponding network access time required to generate windows is almost the same as the traditional linear approach. When the number of queries related with *Includes*, *Similar* or *Overlap* window relationship type increases and *Disjoint* queries decreases in the pair, the proposed approach utilizes semantically similar feed sources to generate a common window for queries with the same window relationship type. Hence, the required time for shared network access decreases each time as shown in the red, blue, and green lines representing *Overlap, Includes* and *Similar* relationship type for Y respectively in the pair [D, Y].

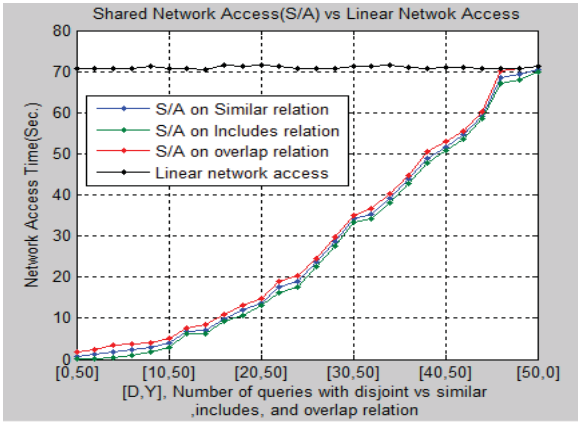


**Figure 6-3: Graph for experimental result of NAT**

## 6.2 MQE of SNF Queries

This experiment is done to show the performance of the system on executing multiple SNF queries together. Figure 6-4 shows query execution time required to execute 50 queries. Using traditional approach, the required time is almost constant time as shown by the pink line. However, using our proposed approach to execute multiple SNF queries together, the time requirement decreases as the number of disjoint query in the pair [D, X], (D for disjoint and X for *Similar*, *Includes* or *Overlap*) decreases and queries with *Includes*, *Similar* or *Overlap* increases. As a result, the lines in red, green, and blue on the graph for pairs of disjoint with *Overlap, Includes* and *Similar* respectively approaches down to the origin showing a decrease in execution time and hence an increase in the performance of the system.

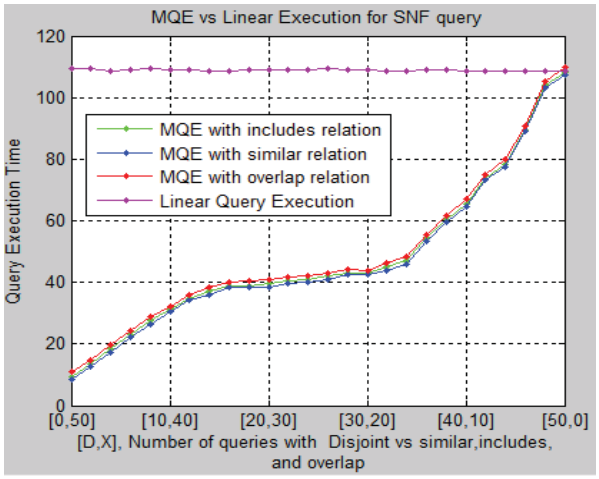


**Figure 6-4: Graph for MQE timing of queries based on window relationship type.**

Generally, as realized by both experiments the proposed multi query execution for SNF query shows a significant performance increase on the system.

# 7. CONCULUSION AND FUTURE WORK

In this paper, we proposed a multi query execution strategy for semantic news feed query. The approach first evaluates the semantic similarity of a set of queries registered in profile database, and then identifies a window relationship type among queries accessing the same feed source; and finally a MQE chain is generated for the identified window relationship types. MQE rules are used to manage the execution of queries in the MQE chain. To validate our proposal, we have developed a prototype which helps users to subscribe their favorite feed sources and define queries, schedule queries and manage query schedule using query scheduler component, and auto execution of queries according to user`s schedule.

This work opens up several avenues for future research works especially in MQE of stream queries accessing XML type stream of data in publish/subscribe systems. Currently, we are working on load shedding technique, which is the process of dropping non relevant data to increase the performance of the system that works with MQE of semantic news feed query.

# 8. REFERENCES


[1] RSS Specifications, http://www.rss-specifications.com/What-is-rss.htm, last accessed May 12, 2012

[2] Fekade Getahun, Richard Chbeir, Semantic Aware RSS Query Algebra, In Proc. of iiWAS'2010, 2010

[3] F. Fabret, H.A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha, Filtering Algorithms and Implementation for very Fast Publish/Pubscribe, In Proc. of SIGMOD, pages 115–126, 2001.

[4] M. A. Hammad, M. J. Franklin, W. G. Aref, and A. K. Elmagarmid, Scheduling for Shared Window Joins over Data Streams, In Proc. of VLDB, pages 297–308, 2003.

[5] S. R. Madden, M. A. Shah, J. M. Hellerstein, and V. Raman, Continuously Adaptive Continuous Queries over Streams, In Proc. of SIGMOD, 2002.

[6] S. Krishnamurthy, M. J. Franklin, J. M. Hellerstein, and G. Jacobson, The Case for Precision Sharing, In Proc. of VLDB, pages 972–986, 2004.

[7] S. Krishnamurthy, C. Wu, and M. Franklin, On-the-fly Sharing for Streamed Aggregation, In Proc. of SIGMOD, 2006.

[8] R. Zhang, N. Koudas, B. C. Ooi, and D. Srivastava, Multiple Aggregations over Data Streams, In Proc. of SIGMOD, 2005.

[9] A. Demers, J. Gehrke, M. Hong, M. Riedewald, and W. White, Towards Expressive Publish/Subscribe Systems, In Proc. of EDBT, 2006.

[10] Arvind A., and Jennifer W, Resource Sharing in Continuous Sliding-Window Aggregates, Technical Report. Stabford, 2004.

[11] Alasdair J. andG. Gray, A Data Stream Publish/Subscribe Architecture with Self-adapting Queries, In Proc. of OTM'05, 2005.

[12] Song W., Elke R., Samrat G., and Sudeept B.,StateSlice: New Paradigm of Multi Query Optimization of Window Based Stream Queries, VLDB '06, 2006.

[13] Mingsheng H., Alan D., and Johannes G., Massively Multi-Query Join Processing in Publish/Subscribe Systems, SIGMOD'07, 2007.

[14] J. Li., David M., Kristin T., Vassilis P., Peter A., No Pane, No Gain: Efficient Evaluation of Sliding Window Aggregates over Data Streams, SIGMOD Rec., 2005.

[15] Shenoda G., Mohamed A., Panos K., and Alexandros L., Optimized Processing of Multiple Aggregate Continuous queries, CIKM'11, 2011.

[16] Moustafa A, Michael J, Walid G. and Ahmed K., Scheduling for Shared Window Joins over Data Streams, In Proceedings of VLDB '03, 2003.

[17] Easy RSS Manager, http://dbconf.u-bourgogne.fr/EasyRSSManager, Last accessed 19 September 2012.