FIND SIMILAR DOCUMENTS WITHOUT USING A FULL TEXT INDEX

M. Scott Roth
Director of Technology,
Armedia, LLC
scott.roth@armedia.com

EMC²

Table of Contents

Disclaimer: The views, processes, or methodologies published in this article are those of the author. They do not necessarily reflect EMC Corporation's views, processes, or methodologies.

1 Introduction

Recently, an intriguing question was raised in the Documentum® Developer Forum of the EMC Community Network (ECN). The gist of the question was: during the checkin of a document, is there a quick and easy way to determine if a similar document already exists in the repository [1]? This really got me thinking. If the question had been: is there a way to determine if an *exact* duplicate of a document exists in the repository, the answer would be "yes". This determination could be made by subjecting every document in the repository to a cryptographic hash function (message digest) such as MD5 or SHA-1, saving the hash value as metadata, and looking for identical hash values upon checkin. However, because the questioner was interested in "similar" documents, cryptographic hashes were not the solution. Cryptographic hash functions are designed to detect the slightest perturbation of content and produce radically different hash values [2]. Therefore, even similar documents would produce radically different hash values whose comparison would not indicate similarity. So, how could similarity among documents in a repository be detected?

One approach would be to use the Lucene `MoreLikeThis` API. The `MoreLikeThis` API extracts salient words from a selected document and uses them to construct a full text query against its indexes [3]. Since xPlore employs the Lucene engine, this approach might be possible, but would obviously require some modification to Webtop and xPlore to implement it. In addition, constructing and executing the query could take considerable time during a checkin.

I like the notion of a hash value that represents the primary characteristics of a document and could be saved as metadata in the repository. You could then query the repository for all documents that were ±10% of a known hash value and call those documents "similar". Such a value would allow the detection of similar documents in repositories not using xPlore (or Lucene, or FAST), and could easily be implemented with an Aspect. The hash value would be the product of a different kind of hashing function where slight differences in content are ignored, and primary characteristics are highlighted. I call this value the similarity index, SI, of content.

My vision for the use of the SI in a content management system (specifically, Documentum) is represented by the following pseudo code query statement:

```
select object_id from object_type where
    similarity(SI_value,[SI]) >= 80%
```

Here, `SI_value` is the Documentum object attribute name holding a 64-bit SI value, `[SI]` is a known SI value (e.g. the SI value for a document being checked in), and the `similarity()` function performs a simple comparison of the two values to determine similarity. In this case, the query returns all documents that are 80% (or more) similar to the document represented by `[SI]`.

With a simple query statement like this, one could easily return a list of all similar documents in a repository upon checkin. At that point, the content management system could do a number of things such as: alert the user of similar content, ask the user to verify that their content is different, use this information to classify the document, apply additional keywords based upon the key words of the similar documents, etc.

This article will briefly explain the SimHash algorithm used to create SI values, and the novel implementation of a solution for identifying similarity among documents in a Documentum repository, without the aid of a full text search engine. As explained in the following pages, the solution involves an Aspect and its unique relationship with a database view to determine similarity. The solution can be leveraged in SQL or DQL, or in a simple Webtop customization.

2 Calculating and Evaluating the Similarity Index

Before diving into the Documentum implementation of this solution, a little background is necessary to understand how the SI hash values are generated and how they are evaluated in relation to one another to determine similarity.

2.1 The SimHash Algorithm

The algorithm that produces the 64-bit hash that can be evaluated in the manner envisioned is called *SimHash*[i]. The SimHash algorithm is a locality-sensitive hashing algorithm developed by Moses Charikar in 2002 [4]. Locality-sensitive simply means that instead of the algorithm being sensitive to variations in the input stream like a cryptographic hashing algorithm (e.g. MD5, SHA-1, etc.), it ignores variations (to a degree) and groups similar content together. This

concept is similar to data clustering which endeavors to group objects together that share a similar attribute. In this case, the grouping occurs via 64-bit hash value.

SimHash works by breaking the input string into *k-grams*[ii] and producing a fixed-sized *shingle*[iii] for each *k-gram*. The size of the *shingle* is the same size as the final hash, in this case, 64-bits. Each bit position of each *shingle* is reviewed[iv]. If the bit at `shingle[i]` is set (i.e. 1), then the same bit position in a temporary vector, `V[i]`, is incremented by 1. If the bit at `shingle[i]` is not set (i.e. 0), then `V[i]` is decremented by 1. Once the entire input has been evaluated, the SimHash is calculated by reviewing the temporary vector, `V`. If the bit at `V[i]` is greater than 0, then the bit at `H[i]` (the final SimHash value) is set to 1, else it is set to 0. The result of this process is a 64-bit binary number.

Table 1 contains a pseudo-code representation of the SimHash process [5].

Table 1 SimHash Pseudo-code

1. Produce a set of *shingles* (S) for the input.
2. Initialize a temporary vector (V), 64-bits in size, containing all zeroes.
3. For each *shingle* (s) in set S, if s[i] is 1 (where i = bit position), then increment V[i]. If s[i] is 0, decrement V[i].
4. Initialize the SimHash vector (H), 64-bits in size, containing all zeroes.
5. After processing all of the *shingles* in S, evaluate the temporary vector, V: if V[i] > 0, then H[i] = 1, else H[i] = 0.
6. The resulting binary number represented by vector H is the SimHash value.

Table 2 contains a simple example of the SimHash algorithm in action. To conserve space, I used a 16-bit `V` and `H`, and a 3-bit *k-gram*. To produce the *shingles*, I simply summed the ASCII values of the characters in the *k-grams* (this is not a recommended hashing technique for real-world usage, but serves well for this example). Each row in the table represents a successive loop in the algorithm (i.e. step 3 in the pseudo-code above). The input string was “Hello world”.

Table 2 SimHash Example

k-gram	shingle	shingle (binary)	V															
			0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Hel	72 + 101 + 108 = 281	0000000100011001	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	1	1	-1	-1	1
ell	101 + 108 + 108 = 317	0000000100111101	-2	-2	-2	-2	-2	-2	-2	2	-2	-2	0	2	2	0	-2	2
llo	108 + 108 + 111 = 327	0000000101000111	-3	-3	-3	-3	-3	-3	-3	3	-3	-3	-1	1	1	1	-1	3
lo_	108 + 111 + 32 = 251	0000000011111011	-4	-4	-4	-4	-4	-4	-4	2	-2	-2	0	2	2	0	0	4
o_w	111 + 32 + 119 = 262	0000000100000110	-5	-5	-5	-5	-5	-5	-5	3	-3	-3	-1	1	1	1	1	3
_wo	32 + 119 + 111 = 262	0000000100000110	-6	-6	-6	-6	-6	-6	-6	4	-4	-4	-2	0	0	2	2	2
wor	119 + 111 + 114 = 344	0000000101011000	-7	-7	-7	-7	-7	-7	-7	5	-5	-3	-3	1	1	1	1	1
orl	111 + 114 + 108 = 333	0000000101001101	-8	-8	-8	-8	-8	-8	-8	6	-6	-2	-4	0	2	2	0	2
rld	114 + 108 + 100 = 322	0000000101000010	-9	-9	-9	-9	-9	-9	-9	7	-7	-1	-5	-1	1	1	1	1
			H															
			0	0	0	0	0	0	0	1	0	0	0	0	1	1	1	1

The resulting SimHash value is: H = 0000000100001111 = 271 (decimal)

2.2 Comparing SimHash Values

SimHash values can be compared as integer numbers or as binary numbers to determine similarity. Because the SimHash algorithm could place a 1 in the left-most bit of the binary hash number (forcing the decimal representation of that number to be negative), the preferred method to compare SimHash values is to use the Hamming Distance [6]. Simply comparing these numbers based upon their decimal values could misrepresent their similarity due to differing cardinality caused by that first bit. The Hamming Distance avoids this problem by simply identifying the number of bits that differ between the binary representations of two hashes. Table 3 provides an example of comparing two SimHash values using the Hamming Distance.

Table 3 Hamming Distance Example

File 1 hash:	1010001111011011111100110100111101110110111010100001010100010000
File 2 hash:	1010001110011011111101001011111011101101110101000010101000010000
Bit differences:	1 1 11 1 1

The Hamming Distance between File 1 and File 2 in Table 3 is 6. Expressed as a percentage of the length of the hash value, the two SimHash values are only 9% different (6/64=0.09375), or 91% similar! Therefore, the lower the Hamming Distance, the more similar the files.

3 Implementation in Documentum

Now that you understand the basics of SimHash and Hamming Distances, let me explain how these concepts can be applied in a Documentum repository to identify similar documents. The implementation involves the use of an Aspect, a database view, and a few WDK components. This implementation was developed and tested on Documentum 7 using Microsoft SQL Server 2012.

3.1 Overview

The implementation described here leverages a unique relationship between an Aspect and a database view. Figure 1 depicts a general, high-level representation of how the key components of the implementation work. The `SIAspect` Aspect holds both the SI value and all the functionality necessary to generate it, compare it with other SI values, and find similar objects. The beauty of putting this data and behavior in an Aspect, as opposed to a Type-Based Business Object (TBO), is that the Aspect can be attached to any instance of an object in the repository, regardless of its type. This means the capability to identify similar content can easily be layered into an existing repository without having to change any of the existing type definitions.

Key to the `SIAspect`'s ability to find similar content is the `si_view` database view. The `si_view` is a database view that catalogs all objects in the repository with the `SIAspect` attached and calculates their Hamming Distances. The `SIAspect` is able to manage all features of the SI implementation except the ability to find similar objects directly. For this feature, the `SIAspect` must utilize the `si_view` database view. In this regard, the `SIAspect` and the `si_view` database view are inextricably bound. As you will see later, the `si_view` is also available for use by other components in the system.

To better illustrate the relationships among the components in the solution, let's walk through a hypothetical scenario. To start, let's checkin a few drafts of Lincoln's Gettysburg Address to a Documentum repository, and apply the `SIAspect`. Table 4 contains the details.

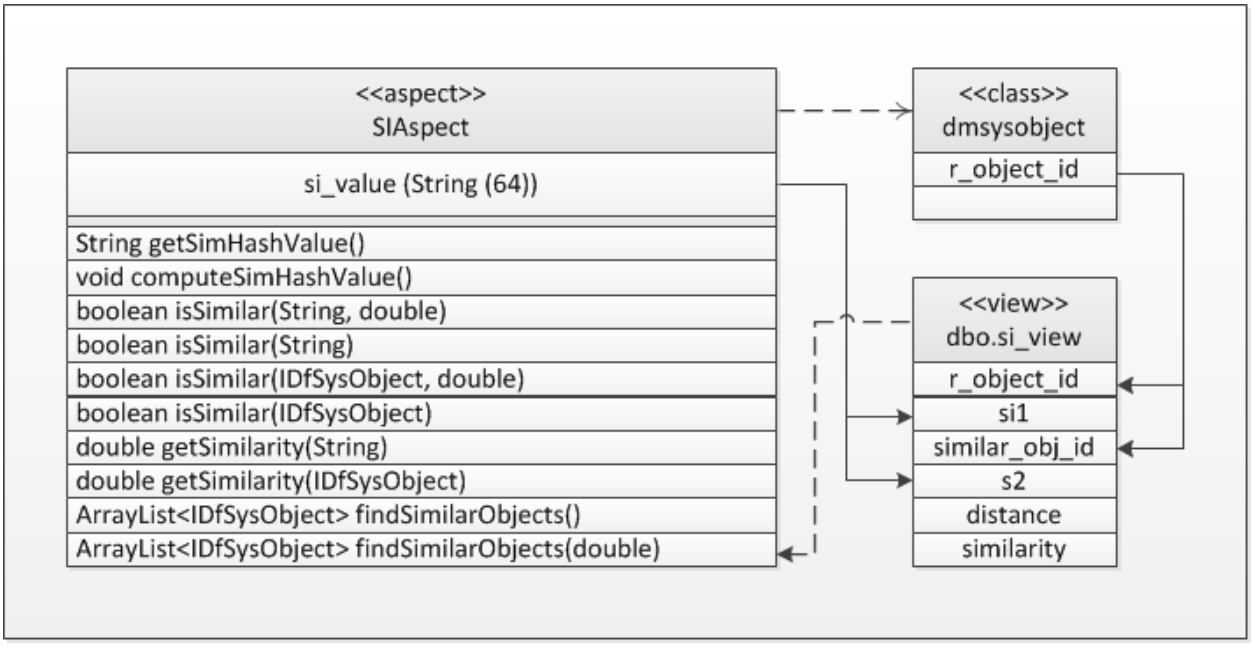

Figure 1 Similarity Index Implementation Overview

Now, suppose we are interested in finding all of the drafts of the Gettysburg Address. Start by selecting the Gettysburg_Address_FINAL.txt object and ask the repository to return all documents that are 80% similar to it. The Aspect uses the selected object's `si_value` to query the `si_view` database table. The `si_view` dynamically builds a combinatorial table that contains all of the `r_object_ids` and `si_values` for all of the objects in the repository that have the `SIAspect` attached. It then determines the distances among all these entries and converts those distances to percentages. The `si_view` then returns to the Aspect a list of `r_object_ids` for similar objects in the repository.

Table 4 Gettysburg Address Example

Document	r_object_id	si_value (16-bit for brevity)	Hamming Distance	similarity
Gettysburg_Address_FINAL.txt	0900000000000001a	1100000001000001		
Gettysburg_Address_DRAFT2.txt	0900000000000010	1100000001000010	2	87%
Gettysburg_Address_DRAFT.txt	0900000000000001	1100000001000110	3	81%
Some_Other_Document.txt	0900000000001111	0000110000100111	8	50%

3.2 The SI Aspect

Like any Aspect in Documentum, the SI Aspect was implemented using a class file, an interface file, a type definition in Composer, and some supporting JARs and libraries. Specifically, the SI

Aspect was implemented in the `SIAspect` class and `ISIAspect` interface files. The `si_aspect_type` was defined in Composer to contain the `si_value` attribute.

Table 5 contains the method prototypes defined in the *`ISIApsect`* interface file. As you can see, of the ten methods defined, there are really only five unique methods; the others are overloads of these five. The following five base methods will be discussed in greater detail:

- *`String getSimHashValue()`*
- *`void computeSimHashValue()`*
- *`boolean isSimilar(String, double)`*
- *`double getSimilarity(String)`*
- *`ArrayList<String> findSimilarObjects(double)`*

You can examine the remaining methods at your leisure in the source code archive (see Section 5).

Table 5 ISIAspect Interface File

```
public interface ISIAspect {

   public String getSimHashValue() throws DfException;

   public void computeSimHashValue() throws DfException;

   public boolean isSimilar(String si_value, double threshold)
         throws DfException;
   public boolean isSimilar(String si_value) throws DfException;
   public boolean isSimilar(IDfSysObject sObj) throws DfException;
   public boolean isSimilar(IDfSysObject sObj, double threshold)
         throws DfException;

   public double getSimilarity(String si_value) throws DfException;
   public double getSimilarity(IDfSysObject sObj) throws DfException;

   public ArrayList<IDfSysObject> findSimilarObjects(double threshold)
         throws DfException;
   public ArrayList<IDfSysObject> findSimilarObjects() throws DfException;
}
```

3.2.1 getSimHashValue()

This method simply returns an object's SimHash value as a String. Table 6 contains the code for this method. Notice that the Aspect name is pre-pended to the attribute name when it is retrieved via the DFC.

Table 6 getSimHashValue() Implementation

```
private static final String SI_ASPECT_NAME = "si_aspect";
private static final String SI_ASPECT_ATTR_NAME = SI_ASPECT_NAME +
   ".si_value";

public String getSimHashValue() throws DfException {
   return this.getString(SI_ASPECT_ATTR_NAME);
}
```

3.2.2 computeSimHashValue()

The *computeSimHashValue()* method sets the Aspect's `si_value` attribute by computing the SimHash using the `SimHash` class from the CommonCrawl project [7]. (The `SimHash` class utilizes classes in the guava [8] and fastutil [9] libraries.) Table 7 contains the code for this method. Again, note the use of the fully qualified attribute name in the DFC *setString()* method.

In this code, the SimHash value is calculated by sending the content of the object to the SimHash method as a String. The SimHash method returns the SimHash value as a 64-bit binary number rendered as a String.

Table 7 computeSimHashValue() Implementation

```
private static final String SI_ASPECT_NAME = "si_aspect";
private static final String SI_ASPECT_ATTR_NAME = SI_ASPECT_NAME +
   ".si_value";

public void computeSimHashValue() throws DfException {
   this.setString(SI_ASPECT_ATTR_NAME,computeSimHash());
}

private String computeSimHash() throws DfException {
   if (this.getContentSize() > 0) {
      ByteArrayInputStream content = this.getContent();
      DfClientX cx = new DfClientX();
      String string1 = cx.ByteArrayInputStreamToString(content);
      return pad(Long.toBinaryString(
         SimHash.computeOptimizedSimHashForString(string1)),
         SimHash.HASH_SIZE);
   } else {
      return null;
   }
}
```

3.2.3 isSimilar(String, double)

Table 8 contains the code for the *isSimilar()* method. All of the *isSimilar()* overloads eventually call this method. This method takes a `String`, representing the SimHash value to compare with this object, and a threshold value as a `double`. For example, a threshold of 0.80 will only return `true` if the two objects are 80% similar (or greater). Similarity is determined by comparing the Hamming Distance between the two SimHash values. This comparison takes place in the *getSimilarity()* method discussed in Section 3.2.4.

Table 8 isSimilar() Implementation

```
public boolean isSimilar(String si_value, double threshold) throws
DfException {

   double sim = getSimilarity(si_value);
   if (sim >= threshold)
      return true;
   else
      return false;
}
```

3.2.4 getSimilarity(String)

The *getSimilarity()* method (see Table 9) return the similarity between two objects as a percent of the overall hash length. This method utilizes the Hamming Distance calculation implemented in the SimHash class of the CommonCrawl project [7].

Table 9 getSimilarity() Implementation

```
public double getSimilarity(String si_value) throws DfException {
   int distance = getDistance(getSimHashValue(),si_value);
   return (((double) (SimHash.HASH_SIZE - distance)/(double)
      SimHash.HASH_SIZE));
}

private int getDistance(String s1, String s2) {
   long l1 = new BigInteger(s1, 2).longValue();
   long l2 = new BigInteger(s2, 2).longValue();
   return SimHash.hammingDistance(l1,l2);
}
```

3.2.5 findSimilarObjects(double)

The *findSimilarObjects()* method returns an ArrayList<IDfSysObject> of objects similar to this object (see Table 10). The double parameter represents the threshold for determining similarity. For example, a 0.80 threshold will return only objects which are 80% or greater in similarity to this object. The default threshold is 0.80, so if the overloaded method *findSimilarObjects()* is called, 0.80 is passed as the threshold value.

Table 10 findSimilarObjects() Implementation

```
private static final String SI_VIEW_TABLE = "dbo.si_view";
private static final String SI_VIEW_SIMILARITY = "similarity";
private static final String SI_VIEW_SIMILAR_OBJ_ID = "similar_obj_id";
private static final String SI_VIEW_THIS_INDEX = "r_object_id";
private static final String FIND_SIMILAR_QUERY = "select " +
   SI_VIEW_SIMILAR_OBJ_ID + " from " + SI_VIEW_TABLE + " where " +
   SI_VIEW_THIS_INDEX + " = '%s' and " + SI_VIEW_SIMILARITY + " >= %s";

public ArrayList<IDfSysObject> findSimilarObjects(double threshold) throws
   DfException {

      IDfCollection col = null;
      ArrayList<IDfSysObject> results = new ArrayList<IDfSysObject>();

   try {
      IDfQuery q = new DfQuery();
      String dql = String.format(FIND_SIMILAR_QUERY,
            this.getObjectId().toString(),Double.toString(threshold));
      q.setDQL(dql);
      col = q.execute(this.getSession(),DfQuery.DF_READ_QUERY);
      while (col.next()) {
         IDfSysObject sObj = (IDfSysObject) getSession().getObject(
            new DfId(col.getString(SI_VIEW_SIMILAR_OBJ_ID)));
         if (sObj != null)
            results.add(sObj);
      }
   } catch (DfException e) {
      throw new DfException(e);
   } finally {
      if (col != null)
         col.close();
   }
   return results;
}
```

This is the method that leverages the si_view database view. The si_view database view is discussed in greater detail in Section 3.3, but as you can see in the code, similar objects are identified by a query of the form depicted in Table 11.

Table 11 Find Similar Query

```
select similar_obj_id
from dbo.si_view
where r_object_id = '[the r_object_id for this object]' and
      similarity >= [the threshold value]
```

3.2.6 Composer Configurations

The next step in implementing the SI capability in a repository is to create the necessary Composer objects for the Aspect. This section is not meant to be a tutorial on creating Aspects using Composer; rather, it will simply highlight the configurations necessary to implement the SI Aspect. There are four primary configurations that need to be made to implement and deploy the SI Aspect: a type definition, a module definition, JAR definitions, and a library. Figure 2 depicts these configuration objects in the Composer project.

3.2.6.1 Type Definition

The SI Aspect utilizes a type definition to define the `si_value` attribute. This type is defined in the Types folder as an Aspect Type with the following characteristics:

- Name: `si_aspect`
- Attribute: `si_value; String(64); default = 0`

3.2.6.2 JAR Definitions

The second configuration is the definition of the JAR files that contain the SI Aspect, its interface file, and its libraries. The five JAR Definitions are defined as follows:

- Name: `fastutil`
  - Type: `Implementation`
  - File: `fastutil-5.0.9.jar` [9]
- Name: `guava`
  - Type: `Implementation`
  - File: `guava-14.0.1.jar` [8]

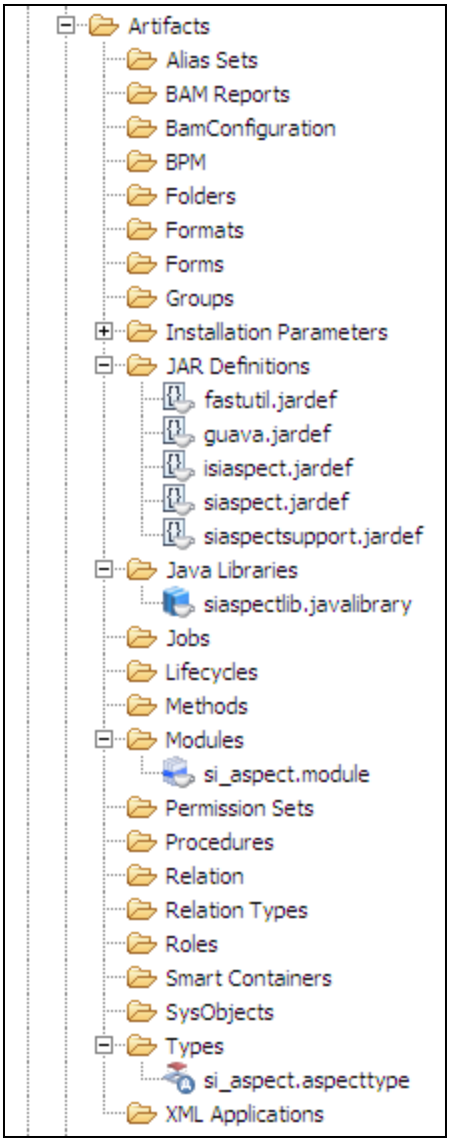

Figure 2 Composer Project

- Name: `ISIAspect`
  - Type: `Interface`
  - File: `ISIAspect.jar`
  - JAR Contents:
    - `ISIAspect.class`
- Name: `SIAspect`
  - Type: `Implementation`
  - File: `SIAspect.jar`
  - JAR Contents:

        - SIAspect.class
- Name: SIAspectSupport
    - Type: Implementation
    - File: SIAspectSupport.jar
    - JAR Contents:
        - SimHash.class,
        - Shingle.class,
        - MurmurHash.class,
        - FPGenerator.class [7]

3.2.6.3 Java Libraries

The Java Library, SIAspectLib, includes the following JAR Definitions:

- fastutil [9]
- guava [8]

This library should be downloaded with the SIAspect when it runs. However, I found this not to be the case and had to install the JAR files in the web application's /lib folder. Section 3.5.3 discusses deploying the SI solution to the web application server.

3.2.6.4 Module Definition

Finally, the definition of the Aspect itself is contained in the Modules folder.

- Name: si_aspect
- Type: Aspect
- Implementation JARS:
    - SIAspect
    - SIAspectSupport
- Class Name: com.dm_misc.similarity.SIAspect
- Interface JARS:
    - ISIAspect

3.3 The SI Database View

The workhorse of this implementation is the database view, `si_view`. This database view is constructed using a database client and is then registered with Documentum so the `SIAspect` can access it. In order to construct the `si_view` database view, you must first determine the name of the table in Documentum that stores the SI values for the Aspect, and implement a Hamming Distance function in SQL.

3.3.1 Determine Documentum SI Aspect Table

Documentum stores Aspect metadata in database tables just like any other metadata. The query[v] in Table 12 identifies the database table in which the `SIAspect` metadata is stored [10]. (This assumes the Aspect has been deployed as described in Section 3.5.1.)

Table 12 SQL to Determine Aspect Table Name

```
select a.i_attr_def as tbl
from dmc_aspect_type_s a,
   dm_sysobject_s s
where s.r_object_id = a.r_object_id and
   s.object_name = 'si_aspect'
```

The result of this query will be a table named similar to `dmi_0301e45380000259`. This table name is used in Section 3.3.3 when constructing the `si_view` database view.

3.3.2 Create SQL Hamming Distance Function

The `si_view` database view requires the use of a custom SQL function to compute Hamming Distances. The SQL[vi] to create the Hamming Distance function, *`HamDist()`*, is contained in Table 13 [11]. This function must be created in the Documentum repository database as the repository owner.

Table 13 SQL Hamming Distance Function

```
create function dbo.HamDist(@value1 char(64), @value2 char(64))
returns int

as
begin
   declare @distance int
   declare @i int
   declare @len int

   select @distance = 0,
      @i = 1,
      @len = case when len(@value1) > len(@value2)
         then len(@value1)
         else len(@value2) end

   if (@value1 is null) or (@value2 is null)
      return null

   while (@i <= @len)
      select @distance = @distance +
         case when substring(@value1,@i,1) != substring(@value2,@i,1)
            then 1
         else 0 end,
      @i = @i +1

   return @distance
end
```

3.3.3 Define si_view Database View

The SQL in Table 14 creates the `si_view` database view that compares every object in the repository with the `SIAspect` attached, to every other object in the repository with the `SIAspect` attached. In this way, it can compute the Hamming Distance and the similarity between each object. This view creates a combinatorial view of the `SIAspect` data, and can be very large. The size of the table can be computed as $n!/(2!(n-2)!)$, [12] where n is the number of objects that have the SI Aspect attached. However, to reduce the number of rows in the view, the SQL only selects objects whose similarity is 70%[vii] or greater. For example, my test repository contained 1,279 objects with the `SIAspect` attached, and resulted in a `si_view` database view with 21,864 rows (as opposed to the expectation of 817,281 if you use the formula above).

Essentially, this view is constructed by choosing an object, searching for all objects that meet the similarity criteria (i.e. 70% or greater), and entering their metadata in the view. The process is repeated for the next object and so on. If you look at the resulting view data, you will see that

the r_object_id column is held constant while the values in the similar_obj_id column vary. This view must also be created in the Documentum repository database as the repository owner.

Table 14 SQL to Create si_view Database View

```
create view dbo.si_view
as
   select a.r_object_id,
      a.si_value as si1,
      b.r_object_id as similar_obj_id,
      b.si_value as si2,
      dbo.HamDist(a.si_value, b.si_value) as distance,
      (64.0 - dbo.HamDist(a.si_value, b.si_value)) / 64 as similarity
from [dmi_table_name]_s as a
   inner join [dmi_table_name]_s as b
      on a.r_object_id <> b.r_object_id
where (a.si_value <> '') and (b.si_value <> '') and
   ((64.0 - dbo.HamDist(a.si_value, b.si_value)) / 64 >= 0.7)
```

Where [dmi_table_name] is the name of the table containing the SIAspect metadata determined in Section 3.3.1 (Table 12).

3.3.4 Register Table in Documentum

The final step in tying together the SIAspect with the empowering si_view database view is to register the si_view with Documentum. Registration of external database tables (and views) with the Content Server allows these tables to be accessed via DQL as if they were native Documentum tables. Table 15 contains the simple DQL to register the si_view database view with Documentum. Register the table from the Documentum Administrator (DA) as the repository owner.

Table 15 DQL to Register si_view Database View

```
register table dm_dbo.si_view (r_object_id string (16),
   si1 string (64),
   similar_obj_id string (16),
   si2 string (64),
   difference int,
   similarity double)
```

By default, Documentum only allows the registered table owner access to the table. Therefore, update the permissions for the registered table to grant everyone READ access to the view by using the DQL in Table 16.

Table 16 DQL to Update Registered Table Permissions

```
update dm_registered object
   set world_table_permit = 3
where object_name = 'si_view'
```

3.4 SI Webtop Components

With the implementation of the Aspect and database components in Sections 0 and 3.3, the foundational elements of the solution are complete. To expose the capability of finding similar documents to the user, I developed a simple *Find Similar* WDK interface for Webtop (see Figure 3).

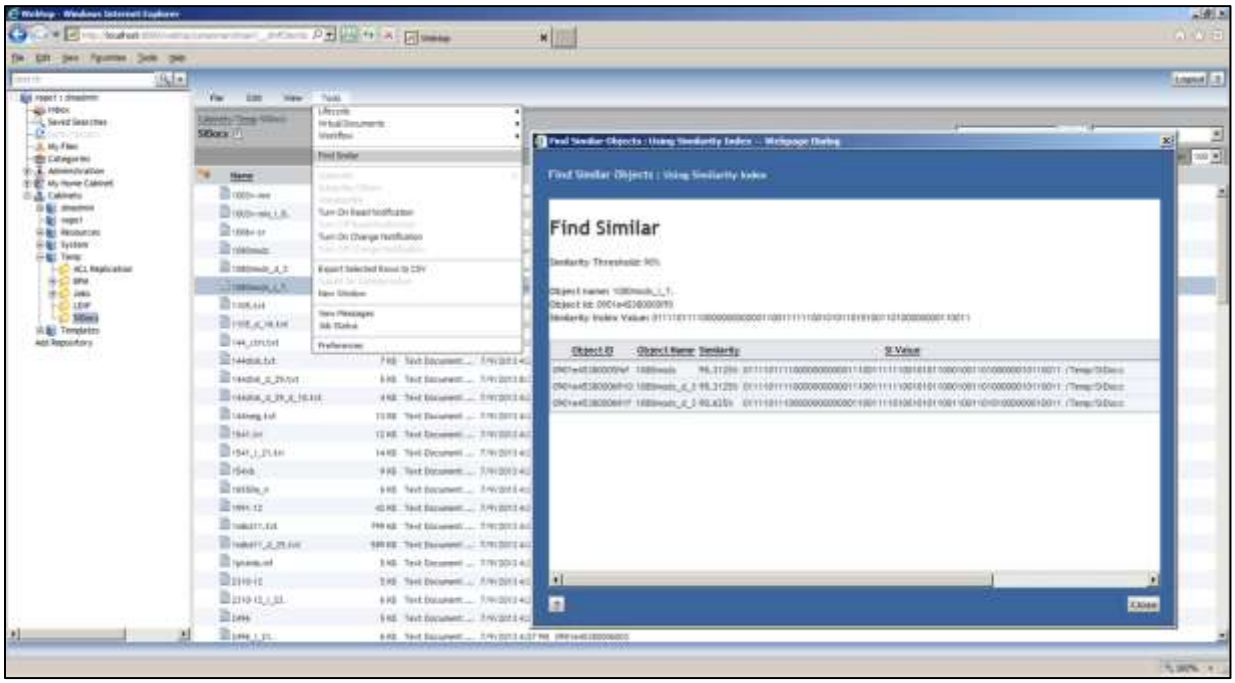

Figure 3 FindSimilar Webtop Interface

This simple interface is invoked by selecting an object in the Webtop work area, and clicking the Tools -> Find Similar menu option. The result is a modal dialog listing the objects that are similar to the one selected in the Webtop work area. The default threshold of 80% is used to determine similarity.

The Webtop component implementation class, *findSimilar* (see Section 3.4.2), leverages the SIAspect attached to the selected object to find similar objects by using the *findSimilarObjects()* method discussed in Section 3.2.5. If the selected object does not have the SIAspect attached, it is attached automatically and the SI calculated. If the object has no content, a message is returned to the modal dialog with no results.

I am only going to discuss the JSP file and the component implementation class here to demonstrate how to use and access the SI Aspect. You can review the remainder of the implementation files yourself.

3.4.1 JSP File

As depicted in Figure 3, the JSP for the findSimilar WDK component is pretty basic. It contains the following controls which are addressed by the implementation class:

- <dmf:label name = "SI_THRESHOLD">
- <dmf:label name = "OBJECT_NAME">
- <dmf:label name = "OBJECT_ID">
- <dmf:label name = "SI_VALUE">
- <dmf:datagrid name = "SI_LIST">
- <dmf:label name = "MSG">

There are some additional details in the JSPcode, but this should suffice for our discussion here.

3.4.2 WDK Component Implementation Class

The findSimilar WDK component code is shown in Table 17. The component overrides the base class Component.*onInit()* method so that as soon as the component is called, the code goes to work finding similar objects in the repository.

The most interesting parts of the code I have flagged as #1, #2, and #3. The remainder of the code handles updating the datagrid and labels on the JSP page [13].

Table 17 findSimilar.java Implementation

```
public void onInit(ArgumentList args) {
   super.onInit(args);

   try {
      // get the object
      String objectId = getInitArgs().get("objectId");
      IDfSysObject sObj = (IDfSysObject) getSession().getObject(
         new DfId(objectId));

      // check for content
      if (sObj.getContentSize() > 0) {

         // --- #1 ---

         // check for aspect - attach if needed
         if (!hasAspect(sObj,"si_aspect")) {
            // attach aspect
            ((IDfAspects) sObj).attachAspect("si_aspect", null);
            sObj.save();
            // must re-fetch the object for the aspect to take affect
            sObj = (IDfSysObject) getSession().getObject(
               sObj.getObjectId());
         }
         // get si value
         ((ISIAspect) sObj).computeSimHashValue();
         // must save again to persist the SimHash
         sObj.save();

         // --- #2 ---

         si_value = ((ISIAspect) sObj).getSimHashValue();

         // --- #3 ---

         // set similar list
         ArrayList<IDfSysObject> simList = ((ISIAspect) sObj).
            findSimilarObjects();

            // initialize the JSP datagrid
            String[] columnHeaders = new String[] {
               "r_object_id",
               "object_name",
               "similarity",
               "si_value",
               "path" };

            // create new table result set
            TableResultSet trs = new TableResultSet
               (columnHeaders);

            // loop over SI ArrayList and set column values
            for (IDfSysObject sObj2: simList) {
               String[] tableRow = new String[5];
               tableRow[0] = sObj2.getObjectId().toString();
```

```
                tableRow[1] = sObj2.getObjectName();
                tableRow[2] = ((ISIAspect) sObj).
                   getSimilarity(sObj2)*100 + "%";
                tableRow[3] = sObj2.getString("si_aspect.si_value");
                tableRow[4] = sObj2.getPath(0);
             }

             // if no results set columns to empty
             if (trs.getResultsCount() == 0) {
                String[] tableRow = new String[5];
                tableRow[0] = "No Results";
                tableRow[1] = " ";
                tableRow[2] = " ";
                tableRow[3] = " ";
                tableRow[4] = " ";
             }

             // finally, add the row to the table result set
             trs.add(tableRow);
          }

          // update datagrid
          Datagrid dg = (Datagrid) getControl("SI_LIST",Datagrid.class);
          dg.getDataProvider().setDfSession(getDfSession());
             dg.getDataProvider().setScrollableResultSet(
             (ScrollableResultSet) trs);

       } else {
          si_value = "This object has no content.";
       }

       // update the form
       Label threshold = (Label) getControl("SI_THRESHOLD",Label.class);
       threshold.setLabel("80%");

       Label objName = (Label) getControl("OBJECT_NAME",Label.class);
       objName.setLabel(sObj.getObjectName());

       Label objId = (Label) getControl("OBJECT_ID",Label.class);
       objId.setLabel(sObj.getObjectId().toString());

       Label SIvalue= (Label) getControl("SI_VALUE",Label.class);
       SIvalue.setLabel(si_value);

    } catch (DfException e) {
       System.out.println(e.getMessage());
    }
}
```

1. This `if{}` block checks whether or not the selected object has the `SIAspect` attached. If not, it tries to attach it. Once the Aspect is attached, the `dm_sysobject` must be re-fetched from the repository to retrieve the attached Aspect. The SimHash value is then calculated for the object and saved.

2. The SimHash value is retrieved from the Aspect. Note that to access the Aspect's functionality, the sysobject must be cast to the Aspect's interface type.
3. The list of similar objects is requested from the Aspect using the *SIAspect.findSimilarObjects()* method. The resulting ArrayList is processed and the required metadata is retrieved from each found object and added to the TableResultSet. After processing all of the objects in the ArrayList, the TableResultSet is added to the Datagrid.

The remainder of the code should be self-explanatory.

The findSimilar WDK component contains one helper method worth mentioning, and that is *hasAspect()*. This method simply queries an object to determine if a particular Aspect is attached to it. Table 18 contains the code for the *hasAspect()* method.

Table 18 hasAspect() Method Implementation

```
private boolean hasAspect(IDfSysObject sObj, String aspect) {
   try {
      IDfList aspectList = ((IDfAspects) sObj).getAspects();
      for (int i=0; i < aspectList.getCount(); i++) {
         if (((String) aspectList.get(i)).equalsIgnoreCase(aspect))
            return true;
      }
   } catch (DfException e) {
      System.out.println(e.getMessage());
   }
   return false;
}
```

3.5 Similarity Index Installation

Installation of this solution occurs in three primary steps, reflecting the three primary parts of the solution: the Aspect in the Documentum repository, the database, and the WDK UI components. See Section 5 for instructions to obtain the complete source code for this solution.

3.5.1 Aspect

The simplest way to install the SIAspect is to install the /bin-dar/SIAspect.dar file distributed with the Composer project using the DAR Installer. Alternatively, it can be installed directly from the project source code using Composer.

3.5.2 Database

The installation of the database components can be completed using the scripts contained in the `/sql-dql` folder of the Composer project. Please note:

- these scripts cannot be installed from Composer
- these scripts should be installed in the Documentum database
- these scripts should be run as the Documentum repository owner
- these scripts should be run in the order listed

The scripts are:

1. `create_HamDist_function.sql` – creates the *`HamDist()`* function in the database. This script must be run from an SQL client.
2. `get_aspect_table_name.sql` – returns the name of the table created to hold the `SIAspect's` SimHash values. You will need to edit the `create_si_view.sql` script to include this table name. Run this script from an SQL client only after installing the `SIAspect`.
3. `create_si_view.sql` – creates the `si_view` database view. Edit this script to include the name of the table produced by the `get_aspect_table_name.sql` script (two places).
4. `register_si_view.dql` – registers the `si_view` database view with Documentum. Run this script from the DQL editor in DA.

3.5.3 Webtop Component Deployment

Deploying the WDK components to implement the *Find Similar* feature in Webtop involves simply copying files from the Composer project into the Webtop folder structure on the web application server[viii] as follows:

- `/webtop/custom/config/find_similar_actions.xml`
- `/webtop/custom/config/find_similar_component.xml`
- `/webtop/custom/config/find_similar_menu_config.xml`
- `/webtop/custom/find_similar/find_similar.jsp`
- `/webtop/custom/strings/com/dm_misc/similarity/SIAspectNLS.properties`

- `/webtop/WEB-INF/classes/com/dm_misc/similarity/ISIAspect.class`
- `/webtop/WEB-INF/classes/com/dm_misc/similarity/SIAspect.class`
- `/webtop/WEB-INF/classes/com/dm_misc/similarity/webtop/findSimilar.class`
- `/webtop/WEB-INF/lib/fastutil-5.0.9.jar`
- `/webtop/WEB-INF/lib/guava-14.0.1.jar`

3.6 Testing

With all of the solution's components in place, you are ready to test the implementation. The first step will be to load some test content into your repository. This process is discussed in Section 3.6.1. The process of loading test content also performs a few tests whose results are discussed in Section 3.6.4. Two other types of tests are also discussed here: a database test, and a UI test.

3.6.1 Loading Test Content

In addition to the code necessary to implement the Similarity Index, one additional class exists in the Composer project. The `LoadTestContent` class loads test content into the repository, attaches the `SIAspect`, generates the SimHash, and runs tests using known and random documents.

I won't go through the details of this class here since it isn't germane for understanding the concepts or the implementation of the Similarity Index. However, it is included in the source code archive, along with the set of test documents[ix]. The `LoadTestContent` class can easily be run from Composer. In its default configuration, it will generate approximately[x] 1,279 files, import them into the `/Temp/SIDocs` folder in your Documentum repository, and attach the `SIAspect` to them.

3.6.2 Database Test

After loading your repository with test data using the `LoadTestContent` class discussed in Section 3.6.1, you can examine the contents of the `si_view` database view using the query in Table 19 (from DA). This query should return about 21,864 rows if you loaded your repository using the `LoadTestContent` class.

Table 19 si_view Test Query

```
select * from dbo.si_view where similarity >= 0.7
```

Notice the view is referenced as 'dm_dbo.si_view'. To avoid ambiguity, registered tables are referenced with a fully qualified name. Documentum provides the dm_dbo alias to always point to the repository owner, regardless of the underlying database system. The si_view database view is fully accessible from both the database (SQL) and Documentum via the registered table, and the SI Aspect. To construct a DQL query using the registered table, always reference it using the dm_dbo prefix as illustrated in Table 19.

3.6.3 Find Similar UI Test

You can also test your deployment by selecting any file in the /Temp/SI Docs folder in Webtop and selecting the Tools -> Find Similar menu option. Your result should be similar to those depicted in Figure 3. A good test document is gettysburg.txt, since it has several variations in the repository that will be identified as similar.

3.6.4 Results

In addition to loading the Documentum repository with test content and attaching the SIAspect, the LoadTestContent class also does some quick tests on the content. Table 20 contains excerpts of the results generated by executing the LoadTestContent class. Some of these results are for control files that I purposely choose and modified to represent similar content; others represent random results from the test corpus.

Table 20 LoadTestContent Results

```
sw56gu.txt
      [0010101110001010000000000111100111001110011010010010100100101010]
      found 1 similar objects

      sw56gu_dup.txt
      [0010101110001010000000000111100111001110011010010010100100101010]
      >> 1.0

killfile.faq
      [1101100111001010100010101100110111111011011010011011100100011110]
      found 1 similar objects
```

```
      killfile_i_25.faq
      [11011001110010101000101011001101111101101101001101111110000111110]
      >> 0.96875

a16.txt
      [10100011101110100100101100010110011100000001100000011100100111110]
      found 3 similar objects

      a16_dup.txt
      [10100011101110100100101100010110011100000001100000011100100111110]
      >> 1.0

      ab88.txt
      [10000010101010100100001101000110001101000001100100011101010111110]
      >> 0.8125

      ba88.txt
      [00100010101110100100101100000110001100000001000100011101010111110]
      >> 0.859375

gettysburg.txt
      [00101111010001110000101100111101110101111010010011000000001000001]
      found 6 similar objects

      getty_d_1.txt
      [00101111010011110000101100111101110101111010010011000000001010001]
      >> 0.96875

      getty_d_10.txt
      [00101101010011110000101100011101110101111010010011001000001000001]
      >> 0.9375

      getty_d_5.txt
      [00101111010001110000101100111101110111111010010010001000001000001]
      >> 0.953125

      getty_i_1.txt
      [00101111010001110000101101111101110101111010010011000000001010001]
      >> 0.96875

      getty_i_10.txt
      [00101111010001110000101100111101110101111010010011000000001010001]
      >> 0.984375

      getty_i_5.txt
      [00101111010011110000101100111101110101111010010011001000001000001]
      >> 0.96875

us-const.txt
      [11110101000110010000111101111001011110100010111100100010100111101]
      found 5 similar objects

      const11.txt
      [11110101000110010000111101111001011110100010111100100010100111101]
      >> 1.0
```

```
      us-const_d_10.txt
      [1100010100011001000011110111100101111010001001110010001010011101]
      >> 0.953125

      us-const_d_5.txt
      [1111010100011001000011110111100101111010001001110010001010011101]
      >> 0.984375

      us-const_i_10.txt
      [1111010100011001000011110111100101111010001011110010001010011101]
      >> 1.0

      us-const_i_5.txt
      [1111010100011001000011110111100101111010001011110010001010011101]
      >> 1.0

MS Word file.docx
      [1010010001101010011010001010001011001110101101111010011110000111]
      found 1 similar objects

      MS Word file_dup.docx
      [1010010001101010011010001010001011001110101101111010011110000111]
      >> 1.0

MS Word file_d_10.docx
      [1011110111101001010101011100001111011010000100111111101111000111]
      found 0 similar objects

MS Word file.pdf
      [0100010111111111000001000111111000010001010001011011111110101001]
      found 1 similar objects

      MS Word file_dup.pdf
      [0100010111111111000001000111111000010001010001011011111110101001]
      >> 1.0

MS Word file_d_10.pdf
      [0110010110101111000011000101010000110001101110011111110110001101]
      found 0 similar objects

VaticanStaircase.jpg
      [0001001011110000110011000110001000001011010111001100000111001000]
      found 1 similar objects

      VaticanStaircase_dup.jpg
      [0001001011110000110011000110001000001011010111001100000111001000]
      >> 1.0
```

In Table 20 you see:

- `sw56gu.txt` and `sw56gu_dup.txt` are 100% similar. In this case, they are exact duplicates of one another.

- killfile.faq and killfile_i_25.faq evaluate to 97% similar even though killfile_i_25.faq contains 25% more content than killfile.faq.
- a_16.txt is a control file and matched with its exact duplicate as well as two other control files, as expected.
- gettysburg.txt was found similar to all of its variations in the repository.
- us-const.txt was also found similar to all of its variations in the repository, as well as an exact duplicate that had a different name (const11.txt).
- The five binary files that were tested (MS Word file.docx, MS Word file_d_10.docx, MS Word file.pdf, MS Word file_d_10.pdf, VaticanStaircase.jpg) matched only on their exact duplicates, and did not match files known to be similar. In a way this makes sense, but highlights a weakness of this solution: all files examined for similarity must be in text format.

4 Conclusion

This article described how to implement a solution in a Documentum repository to find similar content without the use of a full text search engine. The solution capitalized on a unique relationship between an Aspect and a database view. The Aspect provided the creation and storage of a special hash value (called the Similarity Index) using the SimHash algorithm, while the database view created a correlation table of these values against which queries could be run to identify similar content.

As a final solution, I had hoped to be able to issue the query in Table 21.

Table 21 Envisioned Find Similar Query

```
select r_object_id from dm_document where
    similarity(SI_value,[SI]) >= 80%
```

Where, SI_value is the Documentum object attribute name holding the 64-bit SI value, [SI] is a known SI value, and the similarity() function performs a simple comparison of the two values to determine similarity.

The barrier to implementing a solution exactly like this was the inability to extend DQL with a custom function, namely the *similarity()* function. However, the spirit of the vision of this

solution was preserved through the interaction of the `SIAspect` and the `si_view` database view. These two mechanisms were combined in a unique relationship (illustrated in Figure 1) to provide the necessary functionality with only a small sacrifice in usability. Instead of the simple, succinct query illustrated in Table 21, the query is a little more cumbersome, but no less affective (see Table 22).

Table 22 Find Similar Query

```
select similar_obj_id from dbo.si_view where r_object_id =
          '[r_object_id]' and similarity >= 0.80
```

Where `[r_object_id]` is the object ID of the known object.

Instead of the solution querying the objects directly (as shown in Table 21), it queries the database view, which takes care of implementing the functionality not achievable directly in DQL.

The solution presented here is in no way a substitute for the capabilities of xPlore or Lucine. This Similarity Index implementation has a very narrow focus and applicability, namely: it finds syntactically similar documents based upon a hashing algorithm. It gives a good first cut at similarity among documents, but it can be fooled. For example, rearranging paragraphs in a document can impact the result of the SimHash value, thus skewing the similarity comparison, where such rearrangement does not necessarily affect Lucine's *`MoreLikeThis`* API in the same way. However, the Similarity Index does not require additional software, databases, indexes, or storage space to implement. In this regard, I believe the Similarity Index implementation discussed here is a very good, low-cost, low-risk alternative for discovering similar content in a Documentum repository.

Two final caveats about this solution; one was stated in Section 3.6.4, the other was hinted at in Section 3.3.3:

- The SI solution described herein only works for content in pure text form. The SimHash will calculate for binary files; however, there is no guarantee that a single letter change in text, translates to a single bit change in a binary format. Therefore, to truly implement this solution in a production environment, text renditions must exist for all content.

- The potential for the `si_view` database view to grow unreasonably large exists. With my 1,279 test objects, the view was already 21,864 rows long. For a modern RDBMS this is probably inconsequential; however, I believe an upper limit does exist. I did not test for this limit.

5 Get the Source Code

The source code archives for the SI Aspect can be obtained here:

- With test content files (107MB): https://app.box.com/s/z63rznnjbhu3oxmpxerz
- Without test content files (69MB): https://app.box.com/s/9g2cm0j98yvylh7ui6f2
- Only test content files (39MB): https://app.box.com/s/1vb7e8jyvara0jqwxu2d

6 Acknowledgements

Thanks Brian Yasaki and Rachael Roth for your help with the proofing of this paper and the testing of the implementation described herein; you have been invaluable to me. Thank you again.

7 About the Author

Mr. Roth has over 22 years of experience in the software industry; his past 16 years have been devoted almost exclusively to content management. Mr. Roth is a Director of Technology at Armedia, LLC and a certified EMC Technical Architect. Mr. Roth is also the author of the successful how-to book, *A Beginner's Guide to Developing Documentum Desktop Application* (ISBN: 0-595-33968-9) and regularly blogs at msroth.wordpress.com.

8 References

[1] Saladjiev. (2011, March) EMC2 Community Network (ECN). [Online]. https://community.emc.com/message/536386#536386

[2] Wikipedia. [Online]. http://en.wikipedia.org/wiki/Md5

[3] Aaron Johnson. (2008, March) cephas.net. [Online]. http://cephas.net/blog/2008/03/30/how-morelikethis-works-in-lucene/

[4] Moses Charikar. (2002, May) Similarity Estimation Techniques from Rounding. [Online]. http://www.cs.princeton.edu/courses/archive/spr04/cos598B/bib/CharikarEstim.pdf

[5] Philipp Braun. (2011, May) Brauns Brainchildren. [Online]. http://www.philippbraun.net/2011/05/handling-problematic-content-google.html

[6] Wikipedia. [Online]. http://en.wikipedia.org/wiki/Hamming_distance

[7] Common Crawl. [Online]. https://github.com/commoncrawl/commoncrawl-crawler/blob/master/src/org/commoncrawl/util/SimHash.java

[8] Guava: Google Core Libraries for Java 1.6+. [Online]. http://code.google.com/p/guava-libraries/wiki/Release14

[9] fastutil: Fast & compact type-specific collections for Java. [Online]. http://fastutil.dsi.unimi.it/

[10] M. Scott Roth. (2011, June) dm_misc. [Online]. http://msroth.wordpress.com/2011/06/13/where-do-aspects-store-atribute-values/

[11] Jeff Smith. (2007, May) Jeff's SQL Server Blog. [Online]. http://weblogs.sqlteam.com/jeffs/archive/2007/05/09/60197.aspx

[12] Wikipedia. [Online]. http://en.wikipedia.org/wiki/Combination

[13] Mirza Fazlic. (2009, February) EMC Community Network. [Online].

https://community.emc.com/docs/DOC-3032

[14] M. Scott Roth. (2011, July) dm_misc. [Online].
http://msroth.files.wordpress.com/2011/07/similarity-index.pdf

[15] M. Scott Roth. (2013, May) dm_misc. [Online].
http://msroth.files.wordpress.com/2013/05/the-similarity-index-v21.pdf

[i] SimHash is patented in US Patent #7158961.

[ii] A k-gram is simply a string of characters of length *k*.

[iii] A shingle is the result of hashing a *k-gram*. The hashing algorithm used here is cryptographic and can be of your choosing.

[iv] The input is converted to a binary string for comparison.

[v] The SQL presented in this section is SQL, not DQL (unless otherwise noted), and intended to be run from a database client.

[vi] All SQL was developed and tested on Microsoft SQL Server 2012. Your mileage may vary on other databases.

[vii] 70% was an arbitrary cutoff point, but seemed reasonable. I don't expect anyone to be interested in objects that share less than 70% similarity. However, this value can easily be adjusted to meet your needs.

[viii] I used Tomcat 7.

[ix] I tested using files from the Computers and Internet archives at www.textfiles.com, in addition to some control files I developed myself.

[x] Because there is an element of randomness to loading the files, it is not possible to state an exact number.