Praxismodul: Duplikateliminator

1.Zielbestimmung

Es ist eine Softwarelösung zu entwickeln, die einen vorgegebenen Datenbestand nach bestimmten Kriterien durchsucht, die einzelnen Dateiattribute in eine Datenbank schreibt und soweit möglich selbstständig bei Verletzung dieser Kriterien vordefinierte Aktionen ausführt. Solche Kriterien können beispielsweise doppelte Dateien sein, oder Dateien die seit langem nicht mehr benutzt wurden.

Mögliche Aktionen sollen beispielsweise das Benachrichtigen des Eigentümers der Datei (E-Mail, SMS, etc.), dass Entfernen der Duplikate (Löschen der Duplikate, ersetzen der Duplikate durch Links, etc.) sein oder im Zweifelsfall ein Eskalationskonzept (Bsp: weigert sich der Verursacher oder fällt wiederholt auf, Gespräch mit Abteilungsleiter-IT, dann Geschäftsführer.). Zur administrativen Verwaltung und Wartung des Systems ist eine Web-Oberfläche zu entwickeln, die auch nützliche statistische Auswertungen zur Verfügung stellen soll.

Ziel ist also die Vermeidung von unerwünschter Redundanz und damit verbunden die Entlastung der Speicher- und Backupsysteme.

2.Produkteinsatz

Die Lösung soll möglichst effizient vermeiden das sich alte oder doppelte Dateien auf den Dateiservern ansammeln. Weiterhin soll die Lösung weitgehend selbstständig arbeiten um die Mitarbeiter in der It-Abteilung zu entlasten und nur im Ausnahmefall eine administrative Interaktion erfordern. Aktuelle Untersuchungen im Unternehmen haben ergeben das mehr als 50GB an Speicherplatz nur durch Duplikate belegt werden.

3.Produktübersicht

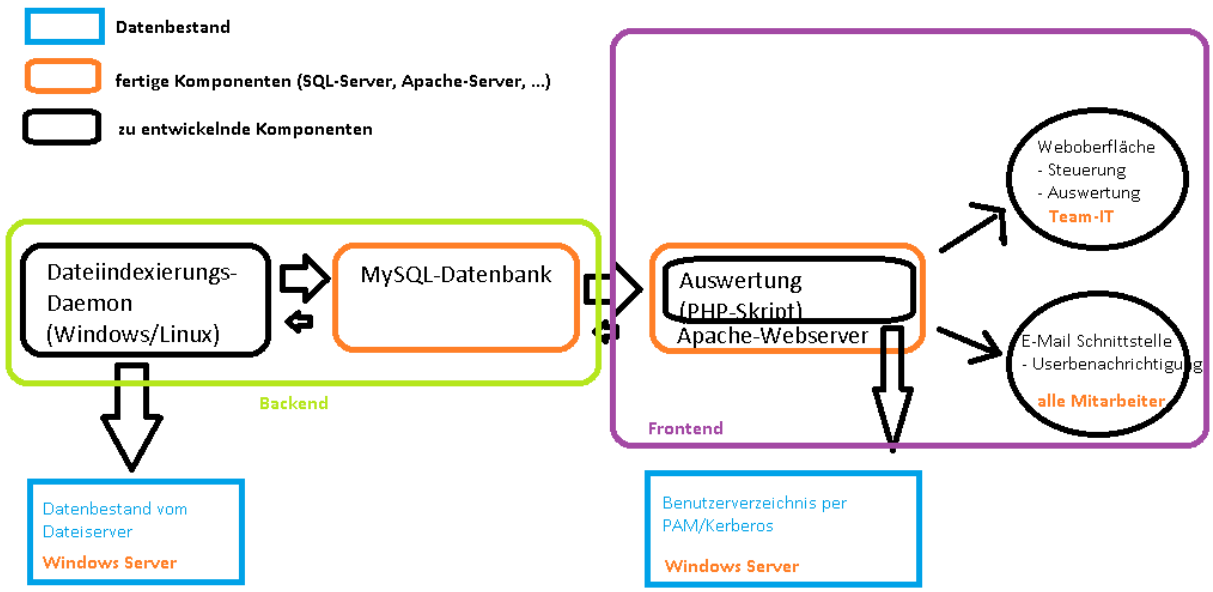

Abbildung 1: Produktübersicht schematisch

Indexierungsdienst mit MySQL-Anbindung:

- Folgende Optionen sind zu prüfen:
    - fdupes (GPL-Lizenz) um MySQL-Anbindung erweitern
        - siehe dazu MySQL Connector/C(++): http://forge.mysql.com/wiki/Connector_C%2B%2B
    - eigenen Indexierungsdienst mit MySQL-Anbindung entwickeln

Verlauf: Punkt 1 wurde nach hinreichender Prüfung verworfen, da fdupes Linux als Host-Betriebssystem vorraussetzt, die zu untersuchenden Verzeichnisse aber ActiveDirectory-Ressourcen einer Windowsdomäne sind. Es bieten sich 2 Möglichkeiten diese Ressourcen unter Linux nutzbar zu machen:

- Einbinden der Windowsfreigaben per Cifs-Mount im lokalen Kontext des Linuxnutzers
    - Nachteil: sämtliche Dateiattribute wie zB der ursprüngliche Dateieigentümer gehen verloren da sie beim mounten durch den lokalen Linuxbenutzer ersestzt werden, in dessen Kontext das mounten stattfindet
- Einbinden des Linuxhosts direkt in die ActiveDirectory-Domäne
    - Wenn überhaupt nur sehr umständlich und mit viel Improvisation möglich, daher für den Produktiveinsatz für nicht praktikabel befunden

Somit scheidet Punkt 1 aus und es wird eine komplette Neuimplementierung umgesetzt. Da die komplette IT-Landschaft der Firma über eine ActiveDirectory-Domäne abgebildet wird und als Betriebssysteme fast ausschließlich Microsoft-Produkte zum Einsatz kommen bietet sich eine Implementierung des Indexierungsdienstes in C# an.

Funktionsumfang des Indexierungsdienstes:

- Rekursives Durchlaufen vorgegebener Verzeichnisse
- Erfassung sämtlicher Dateiattribute der vorgfundenen Dateien sowie Bildung von Prüfsummen um später Duplikate erkennen zu können
- Berechung der Prüfsummen unter Zuhilfenahme der Grafikkarte (Cuda, OpenCL, etc)
- Ausgabe dieser gesammelten Metadaten über eine SQL-Schnittstelle, zB für MySQL oder MSSQL-Server
- Fortschritt der Implementierung in C#: lauffähiger Prototyp ohne die Funktionen:
    - Fehlerbehandlung SQL-Schnittstelle (Programm stürzt im Moment noch einfach ab wenn zB SQL-Server nicht erreichbar ist oder das Passwort falsch ist)
    - Berechnen der Prüfsumme mit Hilfe der Grafikkarte
    - Hinterlegung der Zugangsdaten zum SQL-Server in externer xml-Datei (bisher im Quellcode hardcoded)
- Projekt ist gehostet auf github.com: https://github.com/mkirbst/cworker

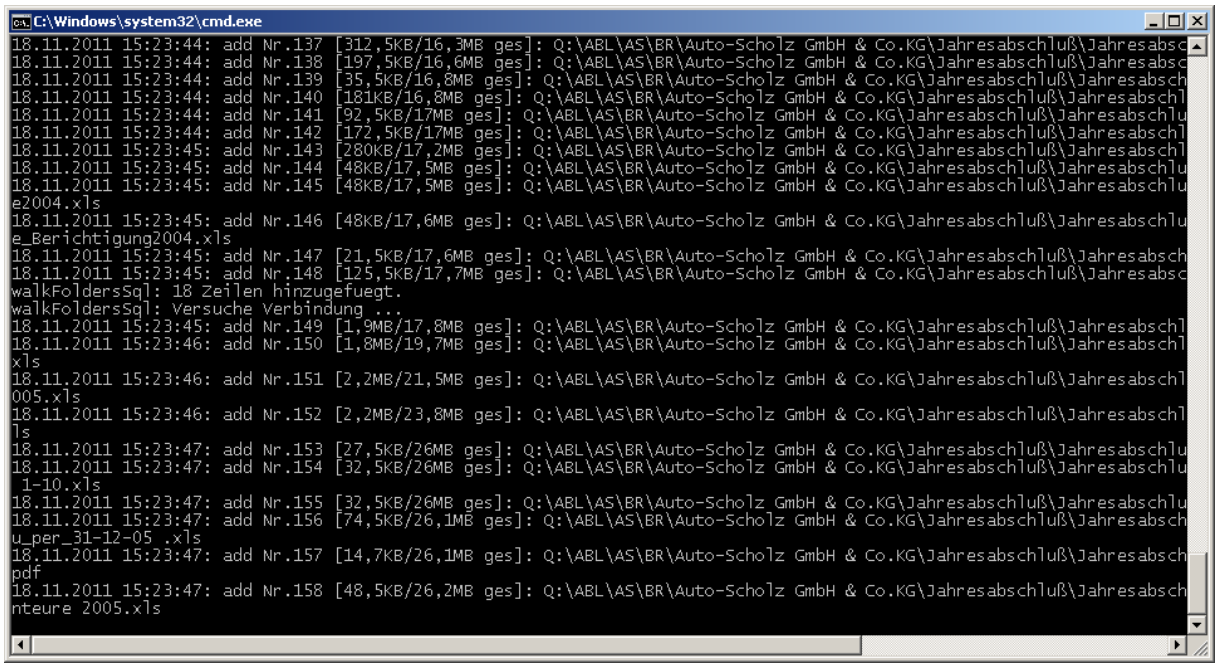

Abbildung 2: Debug-Ausgabe des Dateiindexierungsdienstes

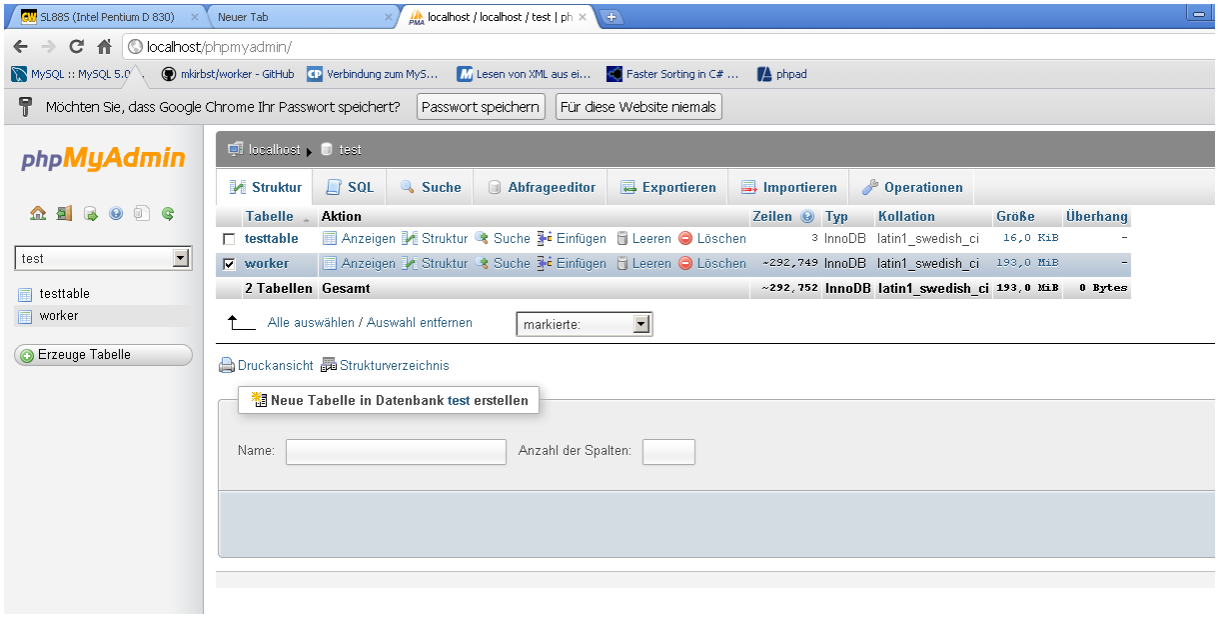

Abbildung 3: Zustand der Tabelle des MySQL-Servers nachdem ca. 200GB Daten indexiert wurden.

Quellcodelisting: Prototyp Dateiindexierungsdienst

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Security.Principal;
using MySql.Data.MySqlClient;

/** Quellenverzeichnis:
 *  walkFolders:    http://dotnet-snippets.de/dns/rekursiver-verzeichnislauf-SID462.aspx
 *  md5:            http://dotnet-snippets.de/dns/md5-hash-von-dateien-ermitteln-SID66.aspx
 *
 */

namespace worker  {

    //todo: aus xml holen und damit authentifizieren
    class creds_sql
    {
        public string username  = "";
        public string password  = "";   //todo: gegen pwhash authentifizieren
        public string hostname  = "";
        public string database  = "";
        public string table     = "";
    }


    class Program    {
        static string path = "Q:\\";
        static string logfilename = "logfile.txt";
        static int gc = 0;
        static long globalsize = 0;

        static void Main(string[] args)   {
            FileWrite(logfilename, DateTime.Now+": Starte Dateiindexierung fuer "+path+"\n\n");
            sql_trunc_table();
            walkFoldersSql(path);
        }

        private static void walkFoldersSql(string DirectorySql)
        {
            walkFoldersSql(new DirectoryInfo(DirectorySql));
        }

        private static void walkFoldersSql(DirectoryInfo disql)
        {
            int lc = 0;
            string sqlcommand = "";
            string myConnectionString = "SERVER=localhost;" +
                                        "DATABASE=test;" +
                                        "UID=root;" +
                                        "PASSWORD=password;";
            Console.WriteLine("walkFoldersSql: Versuche Verbindung ...");

            try
            {


                // Alle Verzeichnisse rekursiv durchlaufen
                foreach (DirectoryInfo subdir in disql.GetDirectories())
                {
                    walkFoldersSql(subdir);
                }

                // Alle Dateien des Verzeichnisses durchlaufen
                foreach (FileInfo fi in disql.GetFiles())
                {
                    ++gc;
                    ++lc;
                    MySqlConnection connection = new MySqlConnection(myConnectionString);
                    MySqlCommand command = connection.CreateCommand();
```

```
                    String output = DateTime.Now + ": add Nr." + gc + " [" + hrs(fi.Length) + "/" +
hrs(globalsize) + " ges]: " + fi.FullName;
                    Console.WriteLine(output);
                    FileAppend(logfilename, output+"\n");

                    connection.Open();
                    command.CommandText = "INSERT INTO `test`.`worker` (`name`, `path`, `loc`, `size`,
`csum`, `dom`, `owner`, `group`, `stime`, `atime`, `ctime`, `mtime`, `dups`) VALUES ('" + fi.Name + "',
'" + fi.FullName + "', '" + fi.FullName.Split('\\')[0] + "', '" + fi.Length + "', '" +
Datei2SHA(fi.FullName) + "', '" +
File.GetAccessControl(@fi.FullName).GetOwner(typeof(NTAccount)).ToString().Split('\\')[0] + "', '" +
File.GetAccessControl(@fi.FullName).GetOwner(typeof(NTAccount)).ToString().Split('\\')[1] + "', '" +
File.GetAccessControl(@fi.FullName).GetGroup(typeof(NTAccount)).ToString().Split('\\')[1] + "', '" +
UnixTime(DateTime.Now) + "', '" + UnixTime(fi.LastAccessTime) + "', '" + UnixTime(fi.CreationTime) +
"', '" + UnixTime(fi.LastWriteTime) + "', '1');";
                    MySqlDataReader Reader;
                    Reader = command.ExecuteReader();
                    globalsize += fi.Length;
                    connection.Close();

                }
                Console.WriteLine("walkFoldersSql: " + lc + " Zeilen hinzugefuegt.");
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }

        public static void FileWrite(String sFilename, String sLines)
        {
            StreamWriter myFile = new StreamWriter(sFilename);
            myFile.Write(sLines);
            myFile.Close();
        }

        public static void FileAppend(string sFilename, string sLines)
        {
            StreamWriter myFile = new StreamWriter(sFilename, true);
            myFile.Write(sLines);
            myFile.Close();
        }

        /**get unix timeformat
         */
        public static long UnixTime(DateTime filetime)
        {
            TimeSpan _TimeSpan = (filetime - new DateTime(1970, 1, 1, 0, 0, 0));
            return (long)_TimeSpan.TotalSeconds;
        }

        /**sha512
         */
        public static string Datei2SHA(string Dateipfad)
        {
            //Datei einlesen
            System.IO.FileStream FileCheck = System.IO.File.OpenRead(Dateipfad);
                System.Security.Cryptography.SHA512 sha512 = new
        System.Security.Cryptography.SHA512CryptoServiceProvider();
            byte[] sha512Hash = sha512.ComputeHash(FileCheck);
            FileCheck.Close();

            //in string wandeln
            string Berechnet = BitConverter.ToString(sha512Hash).Replace("-", "").ToLower();
            return Berechnet;

        }

        //humanreadable filesize
        public static string hrs(long filebytes)  {
            string hrsize = "";
            int n = 0, nachkomma = 0;

            /** Annahme: viel mehr kleine als große Dateien */
            while (filebytes > 1024)
```

```
            {
                nachkomma = (int)(filebytes % 1024);
                filebytes /= 1024;
                n++;
            }
            hrsize = filebytes.ToString();
            if (nachkomma > 100)
            {
                nachkomma = nachkomma / 100;
                hrsize += "," + nachkomma.ToString();
            }
            switch (n)
            {
                case 1: hrsize += "KB"; break;  // " KiloByte"; break;
                case 2: hrsize += "MB"; break;  // " MegaByte"; break;
                case 3: hrsize += "GB"; break;  // " GigaByte"; break;
                case 4: hrsize += "TB"; break;  //" TerraByte"; break;
                case 5: hrsize += "PB"; break;  //" PetaByte"; break;
                case 6: hrsize += "EB"; break;  //" ExaByte"; break; /*long: file.attrs can be max. 9
exabyte*/
                default: hrsize +="B "; break;  //" Byte"; break;
            }
            return hrsize;
        }

        public static bool sql_insert()
        {
            bool erg = true;

            string myConnectionString = "SERVER=localhost;" +
                                        "DATABASE=test;" +
                                        "UID=admin;" +
                                        "PASSWORD=password;";

            Console.WriteLine(": Versuche Verbindung zu: " + myConnectionString);

            MySqlConnection connection = new MySqlConnection(myConnectionString);
            MySqlCommand command = connection.CreateCommand();



            command.CommandText = "INSERT INTO `test`.`worker` (`name`, `path`, `loc`, `size`, `csum`,
`owner`, `group`, `stime`, `atime`, `ctime`, `mtime`, `dups`) VALUES ('" + "" + "',
'c:\\testdateien\\mmmm.pdf', 'c:', '1234',
'mmmmfc92da241694750979ee6cf582f2d5d7d28e18335de05abc54d0560e0f5302860c652bf08d560252aa5e74210546f369fb
bbce8c12cfc7957b2652fe9a75', 'm', 'm', CURRENT_TIMESTAMP, '1983-10-10 22:11:02', '1983-01-01 00:11:02',
'1983-10-10 22:11:11', '1');";
            MySqlDataReader Reader;
            connection.Open();
            Reader = command.ExecuteReader();
            while (Reader.Read())
            {
                string row = "";
                for (int i = 0; i < Reader.FieldCount; i++)
                    row += Reader.GetValue(i).ToString() + ", ";

                Console.WriteLine(row);
            }
            connection.Close();


            Console.WriteLine("Trenne Verbindung...");
            return erg;
        }

        public static bool sql_trunc_table()
        {
            bool erg = true;
            string myConnectionString = "SERVER=localhost;" +
                                        "DATABASE=test;" +
                                        "UID=admin;" +
                                        "PASSWORD=cNtN.5db6!;";

            Console.WriteLine("sql_trunc_table: Versuche Verbindung zu: " + myConnectionString);
            MySqlConnection connection = new MySqlConnection(myConnectionString);
```

```
            MySqlCommand command = connection.CreateCommand();
            command.CommandText = "TRUNCATE TABLE worker;";
            MySqlDataReader Reader;
            connection.Open();
            Reader = command.ExecuteReader();
            connection.Close();
            Console.WriteLine("sql_trunc_table: Trenne Verbindung...");
            return erg;
        }

        public static bool sql_dump()
        {
            bool erg = true;
            int counter = 0;
            string myConnectionString = "SERVER=localhost;" +
                                        "DATABASE=test;" +
                                        "UID=admin;" +
                                        "PASSWORD=password;";

            Console.WriteLine("sql_dump: Versuche Verbindung zu: " + myConnectionString);

            MySqlConnection connection = new MySqlConnection(myConnectionString);
            MySqlCommand command = connection.CreateCommand();
            command.CommandText = "SELECT * FROM worker";
            MySqlDataReader Reader;
            connection.Open();
            Reader = command.ExecuteReader();
            while (Reader.Read())
            {
                string row = "";
                for (int i = 0; i < Reader.FieldCount; i++)
                    row += Reader.GetValue(i).ToString() + ", ";

                Console.WriteLine(row);
            }
            connection.Close();


            Console.WriteLine("sql_dump: Trenne Verbindung...");
            return erg;
        }




    }
}
```

Ausgabe beispielhaft:

MySQL-Datenbank:

Datensatzattribute(vorläufig):

Datenattribut	Datentyp(MySQL)	Einheit	Beschreibung
timestamp	Zeitpunkt (DATETIME)	Sek. Seit 1.1.1970 (Unix-Zeitformat)	Zeitpunkt der erstellung des Datensatzes
checksum	String	MD5/SHA Prüfsumme	
files	Integer (UNSIGNED INT)		Anzahl Dateiexemplare
Path	String	Pfad1\Name1;[Pfad2\Name2;…;]	Pfade (beinhalten auch Dateiname selbst) zu den Dateien
Size	Long Int (UNSIGNED BIGINT)	Byte	Größe der Dateien
Owner	String	Eigentuemer1;[Eigentuemer2;…;]	Eigetümeruser der Datei
Group	String	Gruppe1;[Gruppe2;…;]	Eigentümergruppe der Datei
atime	Zeitpunkt (DATETIME)	Sek. Seit 1.1.1970 (Unix-Zeitformat)	Zeitpunkt des letzten Lesezugriffs
mtime	Zeitpunkt (DATETIME)	Sek. Seit 1.1.1970 (Unix-Zeitformat)	Zeitpunkt des letzten Schreibzugriffs

Primärschlüssel:

- Format: [Pruefsumme][Path]
- Begründung: In der Praxis soll der Indexierungsdienst periodisch laufen (zB wöchentlich).In dieser Zeit können die Dateien modifiziert worden sein und weist dann auch eine andere Prüfsumme auf. Über den Zeitstempel kann man in der Datenbank aber dann vor der Neuindexierung alle alten Datensätze verwerfen.

Bsp: Erzeuge MySQL-Tabelle:

```
CREATE TABLE duplicates (
        tstamp  TIMESTAMP,
        csum    VARCHAR(32) ASCII
        files   UNISGNED INT,
        path    VARCHAR(512) UNICODE,
        size    UNSIGNED BIGINT,
        owner   VARCHAR(64) UNICODE,
        group   VARCHAR(64) UNICODE,
        atime   TIMESTAMP,
        mtime TIMESTAMP
);
```

Auswertungsmodul:

-TODO-