A TYPICAL CASE OF RECOMMENDING THE USE OF A HIERARCHICAL DATA FORMAT

Elisa Margareth Sibarani[1], Paul Wagenaars[2], Guido Bakema[3]

[1]Del Polytechnic of Informatics, Toba Samosir, North Sumatra, Indonesia
elisa@del.ac.id
[2]KEMA, Arnhem, the Netherlands
paul.wagenaars@kema.com
[3]HAN University of Applied Sciences, Arnhem and Nijmegen, The Netherlands
guido.bakema@han.nl

ABSTRACT

Storing large amounts of data in an efficient way from the point of view of very fast retrieval is an important requirement for many industries like KEMA, a company offering consulting, testing and certification in the energy business. Because their relational database could not cope fast enough with the large amounts of data involved, an alternative way for data storage was proposed, called Hierarchical Data Format 5 (HDF5). HDF5 is a data model, a library, and file format for storing and managing data. Four hierarchical designs for storing and retrieving the large amounts of data involved were investigated. A benchmark was carried out in order to know which hierarchical structure would perform best. Eventually, a benchmark between HDF5 and MS SQL Server was carried out. It could be shown that HDF5 performs four (4) times better for inserting and even 200 times better for retrieving data than the MS. SQL Server.

KEYWORDS

Hierarchical Data Format (HDF), Relational Database, Hierarchical Design, Benchmarking

1. INTRODUCTION

One of the services that KEMA provides to its customers is the ability to continuously monitor distributed cable circuits for upcoming defects. The monitoring system, called Smart Cable Guard (SCG) [1], currently monitors over 100 circuits with an average of 5 discharges per circuit per minute. Regardless of the amount of circuits and pulse discharges per minute, the system should be able to complete all processes within at most one hour. The current MS SQL Server 2008 database cannot adapt to the new requirement of the user and the low query performance for inserting and retrieving more PD data. This limitation of the current database in handling the future bulks of data triggered KEMA to find alternatives for their data storage.

In order to provide a fast response service in a DBMS like MS SQL Server, various approaches and techniques were implemented, including remodelling the database design, implementing indexing and partitioning, and so on. However, these techniques could not support the high storing and retrieval performance that is needed successfully.

DOI: 10.5121/ijdms.2012.4603

To be able to fulfil the future requirement when the customer demand will grow based on the future increase of circuits to be planted, the future SCG system must be able to handle 5000 circuits with 1000 pulses every minute without decreasing the performance of accessing the data. Over time, this will produce millions of extra rows in the database, even for just one hour of partial discharges data; in total it will produce at most 300 million partial discharges per hour.
The new circuit must be controlled and monitored by the analyst and a measurement unit will sent the partial discharges to the database server located in the central office. The current system implements their storage in SQL Server and inserting such amounts of more rows into the table will make the response time to slow. This is proved from the benchmark result [2], in which the time needed to store 6 million rows of partial discharges data proved to be more than one hour.

There can be several reasons for replacing a relational solution:

1. When the data to be modeled cannot easily be represented in a relational database.
2. Handling (inserting and retrieving) massive amounts of data proves to be to slow.
3. Difficult to find information very quickly in a bulk amounts of unstructured data.

In the situation at hand, it is obviously the second reason that asks for an alternative.

An alternative data model was considered for answering the needs of the system at hand to cope with such immense amounts of data: a hierarchical file system, with tooling called HDF, because "*While the relational database approach is a logically attractive, feasible approach, but if the data for example naturally organized in a hierarchical manner and stored as such, the hierarchical approach may give better results* [6]*". "Many HDF adopters have very large datasets, very fast access requirements, or very complex datasets. Others turn to HDF because it allows them to easily share data across a wide variety of computational platforms using applications written in different programming languages. Some use HDF to take advantage of the many open-source and commercial tools that understand HDF* [3]*".*

So, doing a benchmark comparison of response times between HDF5 and SQL Server was a logical thing to do. A hierarchical data file using HDF5 was designed and implemented and those files were used to store PD data and a benchmark had to be done to compare the performance of the new storage and the current database. The findings will give clear understanding for the company whether it will be beneficial to migrate to the new HDF5 based data storage. Moreover, a list had to be made of a comparison of several important services and features already available on SQL Server and whether these are available on HDF5 as well. This list is considered to be useful for the company's decision to migrate, which will not only be based on the performance considerations but also on whether it can be guaranteed that all data can be stored safely and can be recovered when any unlike situation will happen.

The remaining sections of this paper are:

Section 2 addresses and discusses a literature review on HDF5. Section 3 concerns the design and implementation of the hierarchical data file for PD data. Section 4 presents the results of our benchmark evaluation. Section 5 explores the list of comparison between HDF5 and SQL Server. Section 6 presents conclusions.

2. HDF5

HDF stands for Hierarchical Data Format which implements a model for managing and storing data. The model includes an abstract data model and an abstract storage model (the data format), and libraries to implement the abstract model and to map the storage model to different storage mechanisms [10]. *"HDF (Hierarchical Data Format) technologies are relevant when the data challenges being faced push the limits of what can be addressed by traditional database systems, XML documents, or in-house data formats* [3]".

HDF is a data format first developed in the 1980s and currently in Release 4.x (HDF Release 4.x); whereas HDF5 is a new hierarchical data format product consisting of a data format specification and a supporting library implementation for storing and managing data [11]. It is a unique technology suite that makes possible the management of extremely large and complex data collections, which addresses some of the deficiencies of HDF4 [7]. It supports an unlimited variety of data types, and is designed for flexible and efficient I/O and for high volume and complex data.

HDF can be classified as file management system because it stores data in a plain text file contains of many lines in which one line holds one record, with fields separated by delimiters, such as commas or tabs [6]. HDF consists of four levels [8], as illustrated in figure below with the explanation as follows:

1. At its lowest level, HDF (Hierarchical Data Format) is a physical file format for storing scientific data.
2. At its highest level, HDF is a collection of utilities and applications for manipulating, viewing, and analyzing data in HDF files.

Between these levels, HDF is a software library that provides high-level APIs and a low-level data interface. Basically, everything builds from files as a base. But file systems have no metadata beyond a hierarchical directory structure and file names. In contrast, HDF5 has a schema language (metadata) to define the metadata. From [9] point of view, they categorized it as simple database systems, while others might think of it as file systems.

2.1 HDF5 Library

The HDF5 library provides a programming interface to a concrete implementation of the abstract models. Figure below illustrates the relationships between the models and implementations.

The detail descriptions of figure 1 are as follows [10]:

1. The *Abstract Data Model* is a conceptual model of data, data types, and data organization. The Abstract Data Model is independent of storage medium or programming environment.
2. The *Storage Model* is a standard representation for the objects of the *Abstract Data Model*.
3. The *Programming Model* is a model of the computing environment, which includes many platforms, from small single systems to large multiprocessors and clusters. The *Programming Model* manipulates (instantiates, populates, and retrieves) objects from the *Abstract Data Model*.

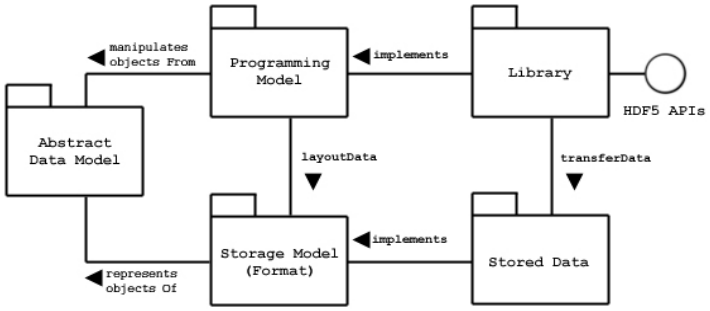

Figure 1 HDF5 Model and Implementation

4. The *Library* is the concrete implementation of the *Programming Model*. The *Library* exports the HDF5 APIs as its interface. In addition to implementing the objects of the *Abstract Data Model*, the *Library* manages data transfers from one stored form to another (e.g., read from disk to memory, write from memory to disk, etc.).
5. The *Stored Data* is the concrete implementation of the *Storage Model*. The *Storage Model* is mapped to several storage mechanisms, including single disk files, multiple files (family of files), and memory representations.

The HDF5 Library is a C module that implements the *Programming Model* and *Abstract Data Model*. The HDF5 Library calls the Operating System or other Storage Management software (e.g., the MPI/IO Library) to store and retrieve persistent data. The HDF5 Library may also link to other software, such as filters for compression. The HDF5 Library is linked to an application program, which may be written in C, C++, Fortran 90, or Java.

The application program implements problem specific algorithms and data structures, and calls the HDF5 Library to store and retrieve data. The HDF5 *Library* implements the objects of the HDF5 *Abstract Data Model*. These include Groups, Datasets, and Attributes and other objects as defined in chapter 2.2. The Application Program maps the application data structures to a hierarchy of HDF5 objects. Each application will create a mapping best suited to its purposes.

2.2. The Abstract Data Model

The Abstract Data Model (ADM) defines concepts for defining and describing complex data stored in files. The HDF5 ADM is a very general model which is designed to conceptually cover many specific models of data. Many different kinds of data can be mapped to objects of the HDF5 ADM, and therefore stored and retrieved using HDF5 [10]. The ADM is not, however, a model of any particular problem or application domain.

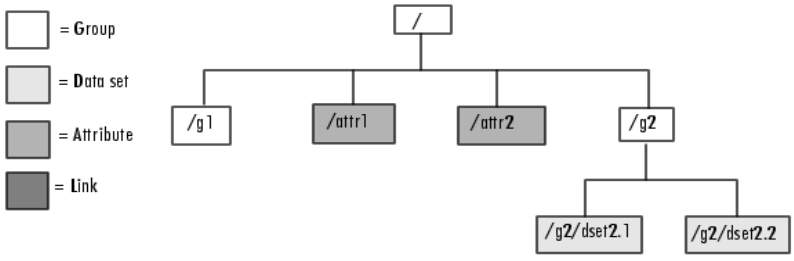

Figure 2 HDF5 Content

Users need to map their data to the concepts of the ADM, illustrated in figure above. The key concepts include:

1. *File* is a contiguous string of bytes in a computer store (memory, disk, etc.). The bytes represent zero or more objects of the model.
2. *Group* is a collection of objects (including groups).
3. *Dataset* is a multidimensional array of Data Elements, with Attributes and other metadata.
4. *Datatype* is a description of a specific class of data element, including its storage layout as a pattern of bits.
5. *Dataspace* is a description of the dimensions of a multidimensional array.
6. *Attribute* is a named data value associated with a group, dataset, or named datatype.
7. *Property List* is a collection of parameters controlling options in the library. Some properties are permanently stored as part of the object; others are transient and apply to a specific access. Each class of property list has specific properties.

2.3. The HDF5 Applications Programming Interface (API)

The API provides routines for creating HDF5 files, creating and writing groups, datasets, and their attributes to HDF5 files, and reading groups, datasets and their attributes from HDF5 files [11]. The current HDF5 API is implemented only in C.

The Naming conventions of all C routines in the HDF 5 library begin with a prefix of the form **H5***, where * is a single letter indicating the object on which the operation is to be performed, explained as follows:

1. **H5F**: **F**ile-level access routines. Example: H5Fopen, which opens an HDF5 file.
2. **H5G**: **G**roup functions, for creating and operating on groups of objects. Example: H5Gset, which sets the working group to the specified group.
3. **H5T:** Data**T**ype functions, for creating and operating on simple and compound datatypes to be used as the elements in data arrays. Example: H5Tcopy, which creates a copy of an existing datatype.
4. **H5S:** Data**S**pace functions, which create and manipulate the dataspace in which the elements of a data array are stored. Example: H5Screate_simple, which creates simple dataspaces.
5. **H5D: D**ataset functions, which manipulate the data within datasets and determine how the data is to be stored in the file. Example: H5Dread, which reads all or part of a dataset into a buffer in memory.
6. **H5P**: **P**roperty list functions, for manipulating object creation and access properties. Example: H5Pset_chunk, which sets the number of dimensions and the size of a chunk.
7. **H5A**: **A**ttribute access and manipulating routines. Example: H5Aget_name, which retrieves name of an attribute.

2.4. HDF5 File Structure

This chapter aims to describe in detail about the organization of a HDF5 file and how HDF5 file using path names and navigation to retrieve specific dataset in that file.

2.4.1. Overall File Structure

An HDF5 file is a container for storing a variety of scientific data and is composed of two primary types of object: groups and datasets. It is organized in a hierarchical structure and the main objects described below [12]:

1. HDF5 Group: a grouping structure containing instances of zero or more groups or datasets, together with supporting metadata. The metadata stores the information about the objects on disk such as names of the object, titles, data types in columns, dimension, etc.
2. HDF5 dataset: a multidimensional array of data elements, together with supporting metadata.

Any HDF5 group or dataset may have an associated attribute list. An HDF5 attribute is a user-defined HDF5 structure that provides extra information about an HDF5 object. Objects in an HDF5 file are often described by giving their full (or absolute) path names, shown as follows:

1. / : signifies the root group.
2. /foo : signifies a member of the root group called *foo*.
3. /foo/zoo : signifies a member of the group *foo*, which in turn is a member of the root group.

2.4.2. HDF5 Path Names and Navigation

A path name is a string of components separated by '/'. Each component is the name of a (hard or soft) link or the special characters '.' (meaning current group). Link names (components) can be any string of ASCII characters not containing '/' (except the string ".", which is reserved) [10].

An object can always be addressed by a *full or absolute path*, in example, starting at the root group. As already noted, a given object can have more than one full path name. An object can also be addressed by a relative path, i.e., a group plus a path starting at the group.

The structure of an HDF5 file is "self-describing", in that it is possible to *navigate* the file to discover all the objects in the file. Basically, the structure is traversed as a graph, starting at one node, and recursively visiting the nodes of the graph.

2.4.3. Reading to/Writing from a Dataset

A dataset is stored in a file in two parts: a header and a data array. The header contains information that is needed to interpret the array portion of the dataset, as well as metadata (or pointers to metadata) that describes or annotates the dataset. Header information includes the name of the object, its dimensionality, its number-type, information about how the data itself is stored on disk, and other information used by the library to speed up access to the dataset or maintain the file's integrity.

There are four essential classes of information in any header as listed below [11]:

1. **Name,** a dataset *name* is a sequence of alphanumeric ASCII characters.
2. **Datatype**, HDF5 allows one to define many different kinds of datatypes. There are two categories of datatypes: *atomic* datatypes and *compound* datatypes. Atomic datatypes can also be system-specific, or *NATIVE*, and all datatypes can be *named*:

a. *Atomic* datatypes include integer, float, string, bit field, and opaque. *(Note: Only integer, float and string classes are available in the current implementation.)*
b. *NATIVE* datatypes are system-specific instances of atomic datatypes. NATIVE datatypes are C-like datatypes that are generally supported by the hardware of the machine on which the library was compiled.
c. Compound datatypes are made up of atomic datatypes. It is one in which a collection of several datatypes are represented as a single unit, a compound datatype, similar to a *struct* in C. The parts of a compound datatype are called *members*, which may be of any datatype, including another compound datatype.
d. *Named* datatypes are either atomic or compound datatypes that have been specifically designated to be shared across datasets. Normally each dataset has its own datatype, but sometimes we may want to share a datatype among several datasets. This can be done using a *named* datatype.

3. **Dataspace**, a dataset *dataspace* describes the dimensionality of the dataset. The dimensions of a dataset can be fixed (unchanging), or they may be *unlimited*, which means that they are extendible (i.e. they can grow larger).
   Properties of a dataspace consist of the *rank* (number of dimensions) of the data array, the *actual sizes of the dimensions* of the array, and the *maximum sizes of the dimensions* of the array. For a fixed-dimension dataset, the actual size is the same as the maximum size of a dimension. When a dimension is unlimited, the maximum size is set to the value H5P_UNLIMITED.
4. **Storage layout**, the HDF5 format makes it possible to store data in a variety of ways. The default storage layout format is *contiguous*, meaning that data is stored in the same linear way that it is organized in memory. Two other storage layout formats are currently defined for HDF5: *compact,* and *chunked. Compact* storage is used when the amount of data is small and can be stored directly in the object header. *(Note: Compact storage is not supported in this release.)*. *Chunked* storage involves dividing the dataset into equal-sized "chunks" that are stored separately

The library will transfer raw data between memory and the file during the process of reading and writing operation related for dataset. To perform read or write operations, the application program must specify [12]:

1. The dataset
2. The dataset's data type in memory
3. The dataset's dataspace in memory
4. The dataset's dataspace in the file
5. The transfer properties (The data transfer properties control various aspects of the I/O operations like the number of processes participating in a collective I/O request or hints to the library to control caching of raw data. The whole implementation will use the default transfer properties.)
6. The data buffer

The steps to read to/write from a dataset must be done by obtain the dataset identifier first, specify the memory data type, specify the memory dataspace, specify the file dataspace, specify the transfer properties, perform the desired operation on the dataset, close the dataset, and close the dataspace/data type, and property list if necessary. To read to/write from a dataset, the calling program must contain the following call which can be seen in figure below.

```
H5Dread(dataset_id, mem_type_id, mem_space_id, file_space_id, xfer_plist_id, buf);
                                    OR
H5Dwrite(dataset_id, mem_type_id, mem_space_id, file_space_id, xfer_plist_id, buf);
```

Figure 3 Read/Write Dataset Routine

3. DESIGN AND BUILD HDF5 STRUCTURE

At this section, the implementation of storing PD data in HDF5 file is presented. The description begins with the possible hierarchy structure of PD in HDF5, several types for the PD hierarchy, and the comparison result which shown the best type for PD data.

3.1. PD Hierarchy Design

The proposed structure of HDF5 file to store PD data consists of several groups, datasets, and attributes. Figure 4 shows the structure of HDF5 file for PD data.

The descriptions in detail about HDF5 file structure for PD table below are listed as follows:

1. The HDF5 file always has the root group with the name "/". There will be no other objects could have the same name, because each name created must be unique and never been used within that HDF5 file. A path name is a string of components separated by '/', and this path will be used to retrieve and access dataset or attribute of an object.
2. The second level after root group is called circuit group. Based on the column which are used in the WHERE clause in the select query, whether CIRCUIT_ID or DATE_TIME will be best to be made as the group to make the dataset clustered based on the same characteristics and therefore increase the retrieving process. In this structure, CIRCUIT_ID was chosen rather than DATE_TIME because it is readable and understandable that for every circuit there must be PD data belongs to it.
3. The third level until fifth level after circuit group will be for DATE_TIME. For each part of date time, it will be split and create as separate group for year, month, and date.
4. The sixth level is for other object is called dataset which stores data elements in multidimensional arrays. This object is the place to store real data and it is located inside of date group because PD data will be inserted every day for every hour.
5. Attributes are small datasets that can be used to describe the nature and/or the intended usage of the object they are attached to. In this structure, attributes were created for date group and dataset to store the number of PD stored in that object. For example, attributes that was created in date group, called "totaline" inserted by value of the total PD stored in dataset inside of that date group. This is used for identify the size of array need to be defined if user wants to retrieve data.

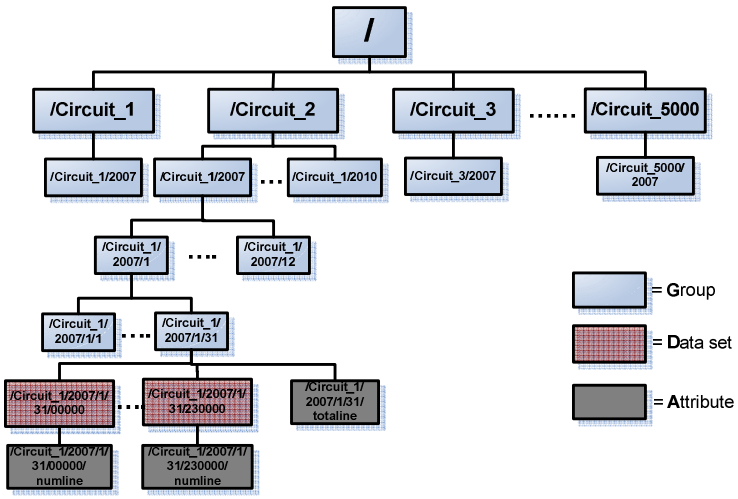

Figure 4 PD Hierarchy Design

Based on the literature, it gives opportunity to create more than one type of hierarchy structure to store PD data. Furthermore, it is possible to give different performance between different hierarchy structures. Therefore, description in detail about several HDF5 structures to store PD data will be explained in the next section.

3.2. HDF5 structure design for PD table

Several types of HDF5 structures was generated so that the proposed HDF5 structure would be the best possible one in performance for inserting and retrieving PD data. Since HDF5 is self-describing data storage therefore the structure for HDF5 could be more than one structure and each structure will be explained in detail in this chapter.

3.2.1. HDF5 Type 1

The first proposed HDF5 structure to store PD data can be seen in figure below which has almost the same structure with the structure in figure 5.

The differences between this structures and the one that is shown in figure 4 are:

1. The name of the first group is not circuit anymore, but it is called System Unit Test abbreviated as SUT.
2. Previously, the structure for the dataset to store PD is only two-dimensional array with atomic data types use integer data type to store the fractional time and charges. The new requirement needs to store five data related to a partial discharges which are: fractional time, charges, phase angle, effective measuring time and minutes of when a partial discharges occurred. Each of the data stored in different data type therefore some changes must be made due to the new requirement which are related to the data space for the dataset using a compound data types consists of integer, float and short data type.

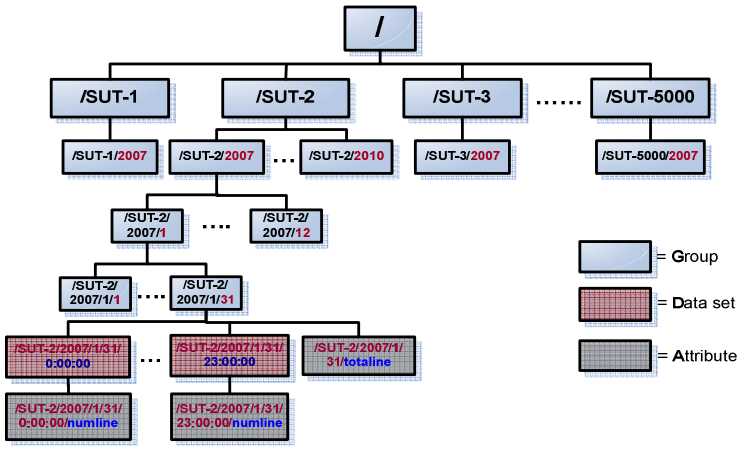

Figure 5 HDF5 Type 1

3.2.2. HDF5 Type 2

The second structure for HDF5 still has hierarchical based on system unit test and date and time, with additional group for hours. The detail descriptions from Figure 6 structure to store PD data are listed below:

1. An hour will be created as a group rather than as a dataset like the first proposed. Instead of stores many null values for effective measuring time and minutes, this structures will grouped partial discharges which is occurred at the same minute stores in one dataset. Therefore the effective measuring time value will be stored as an attribute for each minute dataset.

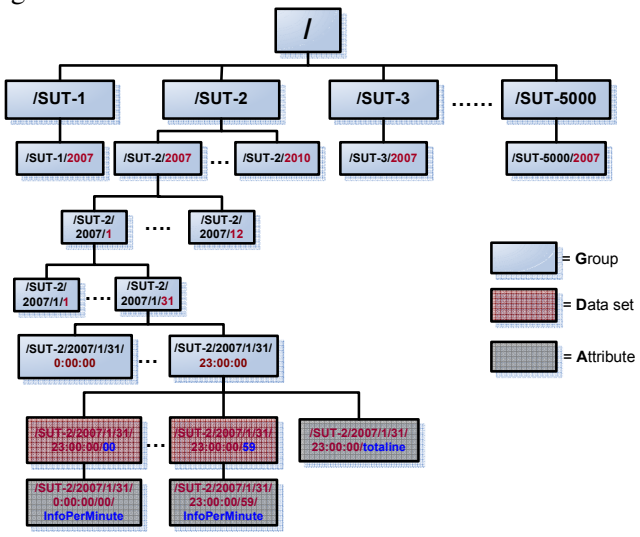

Figure 6 HDF5 Type 2

The dataset contains three dimensions related to partial discharges which are occurred for one minute. The attribute stored for each dataset is two-dimensional array which stores the number of partial discharges for that particular dataset/minute and the effective measuring time for that minute.

2. The hierarchical level for the HDF5 file structure almost the same with the first proposed structure, therefore the explanation for the first level until fifth level can be referred from chapter 3.1.

3.2.3. HDF5 Type 3

The third proposed structure for HDF5 to store PD data is the simplest between the others because it does not have any hierarchical structure in it and stores PD data directly inside a dataset under the root group. The descriptions for HDF5 file structure in figure 7 are as follows:

1. The HDF5 file always has the root group with the name "/".
2. The second level after the root group is a dataset consists of five columns with compound data type. The given name for each dataset contains the system unit test id, year, month, date, and the hour of the time when the partial discharges occurred. Each of the dataset contains one attribute that stores the number of partial discharges stored in one dataset.

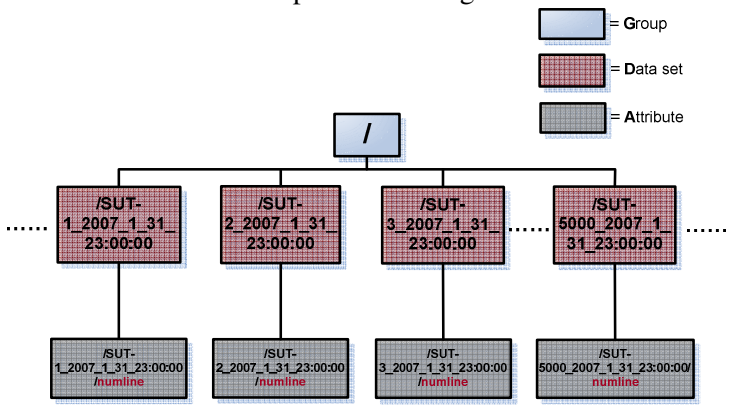

Figure 7 HDF5 Type 3

3.2.4. HDF5 Type 4

The forth proposed structure to store PD data is almost the same with the first proposed HDF5 structure but with some modification in it in order to increase the performance for inserting and retrieving data, as illustrated through figure below.

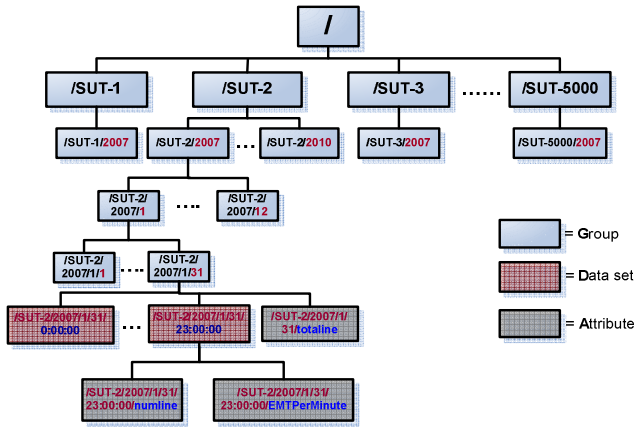

Figure 8 HDF5 Type 4

Figure above displays the structure which is almost the same with the first proposed structure from the first level until the fifth level, nevertheless it still have differences in several points which are listed below:

1. The dataset contains partial discharges data stores per hour with four (4) columns for each datasets. It is possible that increasing time for inserting PD data in line with the increasing number of columns to be inserted to the dataset. Therefore, the modification was made by reducing the number of columns to be inserted into the dataset for each hour by excluding the effective measuring time. It is important to keep the minute columns stored for each partial discharge because user need to know the exact time of when a partial discharges occurred.
2. Each dataset have two attributes, one attribute to store number of partial discharges in that dataset and the number of effective measuring time in one attribute. Another attribute stores the effective measuring time for each minute in that particular hour. Each of the attribute is a two-dimensional array using compound data type because the data stored inside of the attribute has different data type.

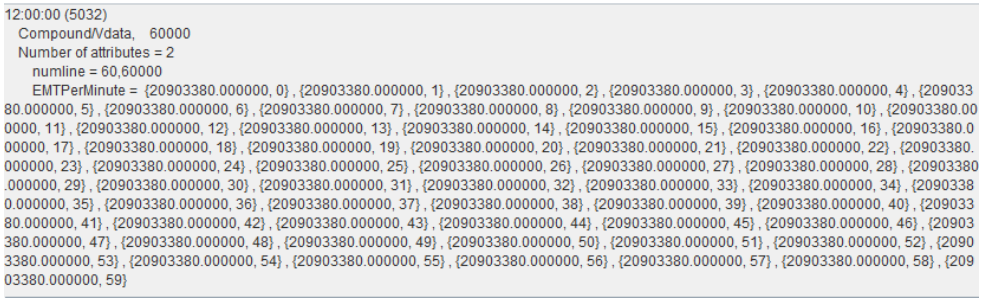

Figure 9 Attribute in Dataset in HDF5 Type 4

Figure above displays the attributes stored in the dataset clearly, from the first attribute, it consists of the number of effective measuring time stored in EMTPerMinute attribute, and the number of partial discharges data in that dataset. The second attribute stores the effective measuring time for each minute that occurs in that hour.

3.3. Comparison result

The purposes of the benchmark is to measure the time needed for inserting and retrieving PD data from each of HDF5 structure and the growing size of the data storage to store PD data. At the end, the best structure must be chosen in order to compare with the performance of SQL Server database which will be explained on Chapter 4.

The result for this benchmark can be seen in Figure 10. It is for three main focused which are the measured time for insert PD data, for retrieve PD data, and the growth size of the file during inserting a new PD data for one hour. The conclusions from the benchmark result are described below:

1. Several factors which strongly affect the performance of insert and retrieve process are (1) adding more dimensions of array in the dataset will create bad performance for insert and retrieve process (2) create many datasets in the file will spend more time that will affect the

performance for insert and retrieve becomes lower (3) by creating no hierarchy in the file (no groups) does not give any impact to the performance of HDF5 file.

2. The best structure that shows best performance in inserting and retrieving and the lowest memory consumption to store PD is type 4. By analyzing the benchmark result for type 1, type 2, and type 3, therefore types 4 is the modification of type 1 but with additional attribute for each dataset and reduce the dimension from the dataset. This is done because the structure must be able to reduce the array dimension as the space of the dataset but not to create more dataset, and the hierarchical structure still exists as well.

Figure 10 Benchmark Result between HDF5 Types

4. BENCHMARK TEST

At this section, the benchmark which is done for HDF5 and SQL Server to be able to come out with the suggestion of which data storage gives the best performance is described. The benchmark was done by following several scenarios which must be adjusted because of the limitation in the memory size of the machine. Therefore to be able to cope with as much as possible the real situation of the system, the scenario must divided based on the number of system unit test, partial discharges per record, and how many measurements was made for each hour. The detail scenario for this benchmark can be seen from table below.

Table 1 Scenario for Benchmark

Scenario	Amount of SUT	Amount of PD	Amount of hour per insertion	Measurements per hour	Total PD inserted for one hour	
					SQL Server	HDF5
1	5000	1000	2 hours (3 times) and 1 hour (4 times)	60	300000	300 million
2	5000	1000	4 hours	6	30000	30 million
3	5000	20	4 hours	60	300000	6 million

The description of each scenario in detail will be as follows:

1. Scenario 1, is based on the future demand of the system, it is the maximum number of PD that possible occurred in each system unit test. Considerably, the insertion process will be made 5 times for SQL Server and 7 times for HDF5 because the limitation in memory size of the computer. Because one measurement could contain more than one partial discharges (in this scenario will be 1000 PD), in SQL Server each measurements will be stored in one column using delimiter, but not in HDF5 storage, each partial discharge on the same measurement will be stored individually per row. Therefore, 0.3 million PD in SQL Server is similar with 300 million PD in HDF5.
2. Scenario 2 is benchmark which is made for partial discharges that measured per ten minutes within one hour therefore the measurements become 6 [4].
3. Scenario 3 is based on the real data that is currently monitored and controlled by the user. The average amount of partial discharges that could be occurred is 20 per system unit test per minute. Therefore it will be good if the current situation also count for the benchmark scenario as well.

4.1. Result for Scenario 1

The result for this benchmark can be seen in figure 11 for three main focused which are the measured time for insert PD data, for retrieve PD data, and the growth size of the file during inserting a new PD data for one hour. The explanation for each of the result is described below:

1. The first chart for insert 1 hour of PD data shows that HDF5 consistently faster to insert PD data than SQL Server, spending almost three times faster than the time needed to insert PD data by SQL Server for the first insertion process. In fact, for the next insertion process shows that the differences gradually increase become almost four times faster for HDF5 compared to SQL Server.
2. The second chart shows that the time consumption for retrieve PD with a range of amount of PD data to be retrieved steadily increased between 120 and 600. The greatest time consumption was made by SQL Server rising from 79.531 seconds to retrieve 120 PD to 275.221 seconds to retrieve 480 PD.

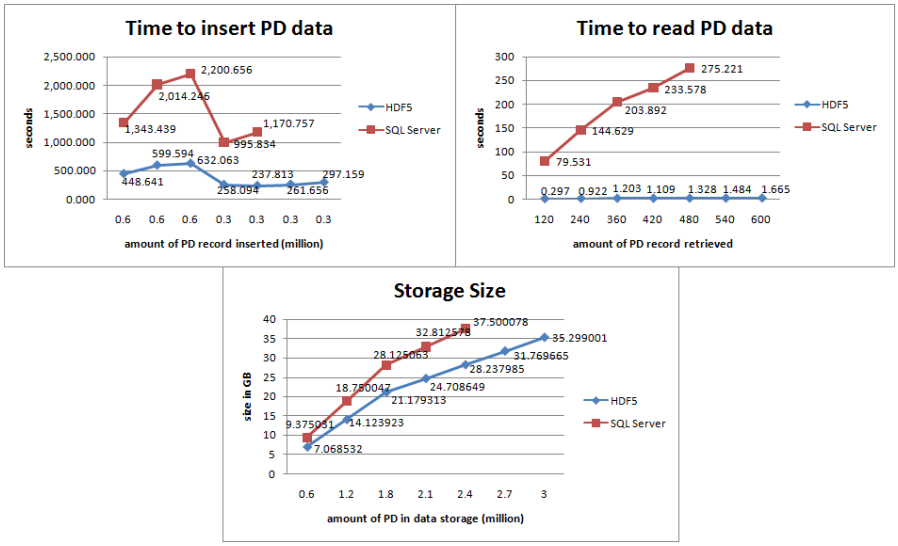

Figure 11 Benchmark Result for Scenario 1

The figure shows huge differences between SQL Server and HDF5; reaches almost 200 times faster for HDF5 to retrieve PD compared to SQL Server.

3. The third chart shows that the growth of storage size by SQL Server and HDF5 indicate overall pattern of increase. The greatest storage size was used by SQL Server which consumes 9.37 GB to store two hours PD data which is 0.6 million PD record. Statistical figures prove that HDF5 seems to be the most efficient data storage to store PD data while SQL Server follows behind.

4.2. Result for Scenario 2

The benchmark result for the second scenario can be seen in Figure 12 below, with the explanation described below:

1. The first chart shows the time to insert PD data. The benchmark measured the time to insert four hours of PD data continuously until the storage size is full. It shows that HDF5 provides better performance than SQL Server in most cases. However, in the sixteenth insert process, the time consumption by SQL Server rapidly decreases from 993.7 seconds to 438.9 seconds. It was happened probably because of the computer performance itself.

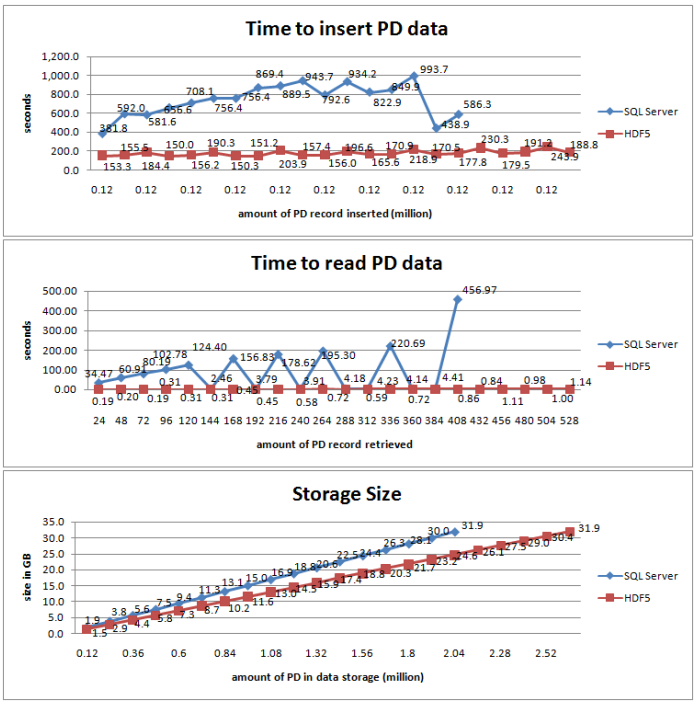

Figure 12 Benchmark Result for Scenario 2

2. The second chart shows the time consumption to read PD data from HDF5 and SQL Server. Similar to insert PD data, HDF5 also provides better read performance than SQL Server in most cases. HDF5 did consume less time than SQL Server as the amount of data stored increases.
3. The third chart shows the memory consumption by both HDF5 and SQL Server. It starts with the small gap between them, only 0.4 GB. But, as the number of PD data increases, so that

the storage gap increases. It can be seen clearly when there is 2 million of PD stored; the gap significantly increases to 7.3 GB.

4.3. Result for Scenario 3

The benchmark result for the last scenario can be seen in Figure 13 below, with explanation for each of the result is described below:

1. The first chart shows the time to insert PD data. The benchmark measured the time to insert four hours of PD data continuously until the storage size is full. On average, the performance gap was HDF5 is 16 times faster than SQL Server. The graph also proves that HDF5 far more stable than SQL Server.
2. The second chart shows the time consumption to read PD data from HDF5 and SQL Server. HDF5 also provides better read performance than SQL Server in most cases. However, SQL Server seems to have unstable time consumption to read PD because sometimes the line rapidly decreases and increases as well. It was most probably because of the computer performance itself.
3. The third chart shows the memory consumption by both HDF5 and SQL Server. It starts with significantly differences between them, 4.4 GB. Indeed, as the number of PD data increases, so that the storage gap increases. It can be seen clearly when there is only 8.4 million of PD stored; the gap significantly increases to 30.6 GB.

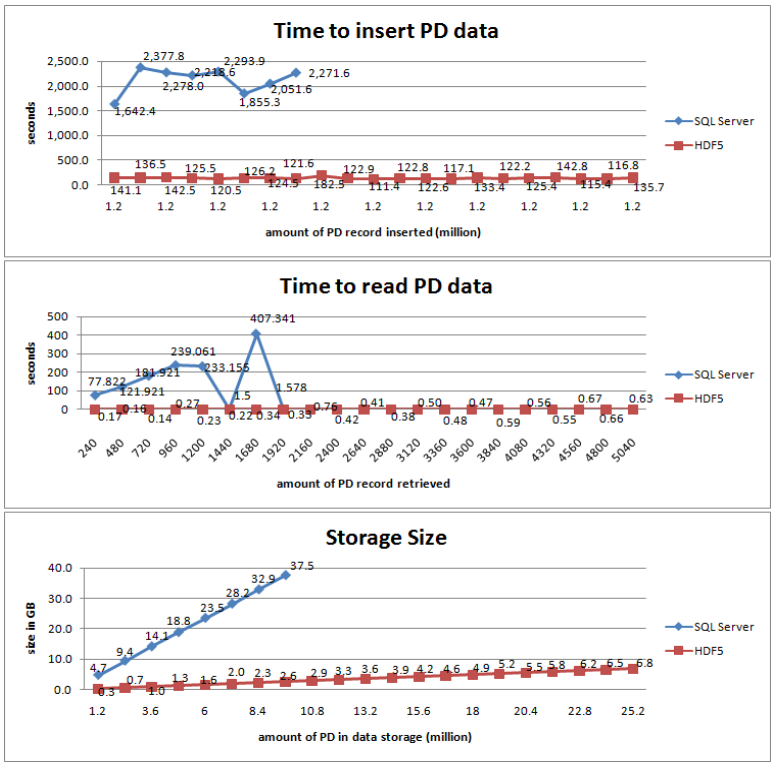

Figure 13 Benchmark Result for Scenario 3

5. LIST OF COMPARISON

It is now accepted that HDF5 is a storage that shows a better performance compared to SQL Server; indeed to rely totally on HDF5 as a storage to insert partial discharges data is not only based on the performance, but has to consider several important services and features which is already found on SQL Server. Likewise, to ensure the SCG system always running and a guarantee that all data stores safely and be able to recover whenever any unlike situation happened, SQL Server already provide all features to handle these kind of situations.

Therefore, the list of comparison for several categories that appears to be the most important factors for a reliable data storage, which can be seen in table below.

Table 2 List of Comparison between SQL Server and HDF5

	SQL Server	HDF5
Technical		
Scalability		
Partitioning	Yes	Yes
Replication	Yes	No
Distributed database	Yes	Yes
Data storage size is limited	Yes	No
Availability		
Backup	Yes	No (available in the 1.10.0 release)
Mirroring	Yes	No (available in the 1.10.0 release)
Replication	Yes	No (available in the 1.10.0 release)
Security		
Windows authentication	Yes	No (Implemented in client)
Data Encryption	Yes	No (Implemented in client)
Concurrency	Yes	Yes (but fully supported available in 1.10.1 release)
Database Recovery		
Database restore/file restore	Yes	No (available in the 1.10.0 release)
Development Tools		
Ms. Visual Studio	Yes	Yes
Java	Yes	Yes
Ease of use and maintain		
Simple structure/data model	Yes	Yes
Commonly used	Yes	No (still known for specific groups)
Easy Installation process	Yes	No installation
Easy connect from dev. tools	Yes	No (depends on bit machine, development tools, etc)

Financial		
License cost	Starts $7,171 (SQL Server Standard)	Free
Support		
Forums	Yes	Yes
Online technical supports/Helpdesk	Yes (only for Microsoft Partner)	Yes
Mailing Lists	No	Yes
Front End (GUI)	Yes	Yes (HDFView-Java-based tool and PyTables)

The descriptions for each of the category as listed in the table above are explained below:

1. There are several categories which can be grouped for a technical side of a data storage, and each of the categories are explained below:
   a. In SQL Server, there are several ways that available to support Scalability, take the example of partitioning and replication, to maintain consistency [13] or to increase the performance and able to handle growing number of data [24].
   However, HDF5 only supports partitioning but not for replication. HDF5 offers the most powerful storage option that is called chunked layout option [14]. It partitions the dataset into equalized "chunks" that are stored separately in the file. The chunking mechanism makes it possible to achieve good performance when accessing subsets of the dataset [15]. It is because data chunking reduces expensive disk seek times and improves overall I/O performance by taking advantage of spatial locality in any of dimensions.
   HDF5 also could be used as distributed data storage with client-server model that provides efficient access to remote HDF5 files. Moreover, there is no theoretical limit to the size of datasets that can be stored in an HDF5 file [3], yet SQL Server does which can only stored data until 524,258 TB [16].
   b. Availability could be handled by the mirroring feature which maintains two copies of a single database, typically located in different locations than the primary one. Replication is also support the availability features, but if the system needs the entire database available as a standby server, it will be better to use mirroring. Another feature is called backup in which the SQL Server engine will create full backup contains all data in a specific database. Unfortunately, all these features are not available at HDF5 yet, but soon after the 1.10.0 version is release, the HDF5 will handle the availability of the file by itself. The new feature to handle this feature is called Metadata Journaling [17].
   c. Security is a property of the database system that protects a database from unauthorized activity, which is usually enforced through access control by using authentication and data encryption. The authentication feature manages who can connect to the database via authentication and the data encryption protects data by storing and transmitting data in an unreadable form. However, HDF5 implements none of these features.
   d. Concurrency is achieved by using locks to help protect data and control how multiple user can access and change shared data at the same time without conflicting with each other. SQL Server fully supports this feature but it is not available yet on HDF5 until the 1.10.0 version releases.
   e. The whole database could be restored and recovered whenever undesirable situation such as massive error or huge crash of the database is occurred. It is available on SQL Server,

unfortunately not available on the current release of HDF5 file. It will be available on the next release of 1.10.0 version of HDF5 [17].

f. Development tools, is list of all development environment could connect to the data storage and for both SQL Server and HDF5, it has almost the same list of development tools that could connect to them, and the most general is Microsoft Visual Studio and Java by Sun Microsystems.

g. Ease of use and maintain, probably was found on both of these data storage in which both of it have simple structure of data model that is easily understand by users. However, HDF5 is not commonly used and still known by specific groups such as scientists, geologists, etc. In addition, SQL Server needs an installation process in order to use the database, but no need of such processes for HDF5 since it is just a library.

2. For financial side, HDF5 tend to be the best choices since it is free and most of the available features or tools that use HDF5 are free as well. In contrary, user has to pay to get the license for using SQL Server [18].
3. Another important issues regarding how far does the contribution of the company to help and support all users of their product. In this case, SQL Server and HDF5 almost have the same support in which both of them have forums and online technical supports or helpdesk [19][20]. However, the online technical support provided by Microsoft is only available for specific group which is users grouped as Microsoft Partner. In addition, SQL Server do not have mailing lists as a place for everyone who is interested in the tools either as a developer or just a user discuss about new things and solve problems together, but HDF5 does have mailing lists for their user.
4. The front-end (GUI) is available for both SQL Server and HDF5, and especially for HDF5 available several GUI for specific programming languages such as Java [21] and Python [22].

6. CONCLUSIONS

Several conclusions can be derived from the result so far, which are listed below:

1. HDF5 is not designed to work as a relational database competitor, but to be able work together for handling massive data and providing faster response time. For example, if very large tables exist in the existing relational database, then those tables can be moved to HDF5 so as to reduce the burden of the existing database while efficiently keeping those huge tables on-disk.
2. HDF5 is a general purpose library and file format for storing scientific data. HDF5 can store two primary objects: datasets and groups. A dataset is a multidimensional array of data elements, and a group is a structure for organizing objects in an HDF5 file.
3. Several aspects for consideration about HDF5 are the implementation manually for security to control the authentication and access control of different users, enabling concurrent access to protect data from inconsistency, and how to recover from crashes.
4. HDF5 is a self-defining file format for transfer of various types of data. Therefore, different structure/hierarchy in HDF5 caused different performance for inserting and retrieving process.
5. HDF5 allows hierarchical data objects to be expressed in a very natural manner, in contrast to the tables of relational database. Whereas relational databases support tables, HDF supports n-dimensional datasets and each element in the dataset may itself be a complex object.
6. HDF is open-source and the software is distributed at no cost. Potential users can evaluate HDF without any financial investment.

7. Partial discharges can be grouped as a scientific data because the data need to be analyzed and finally generate a knowledge rules after implementing some formulation to it.
8. Relational databases offer excellent support for queries based on field matching, but are not well-suited for sequentially processing all records in the database [3].
9. A file stored in HDF contains supporting metadata that describes the content of the file in detail, including information for each multi-dimensional array stored, such as the file offset, array size and the data type of array elements. HDF also allows application-specific metadata to be stored [15]. It is actually implemented while storing PD data. For each dataset, application will create attribute to store the amount of PD inserted in that dataset. Later on, this information will be used for retrieving process, because read process from a dataset needs a buffer as a place to insert the data. That buffer is an array with the same structure to the original dataset. Therefore, HDF5 provides a way in which user could create application-specific metadata to store number of PD in that dataset.
10. HDF5 could be used to store PD data and replace the current database which is MS. SQL Server 2008, supported by the following facts:
    a. PD table in SCG database is a bulk of data, which means no front-end users will access those data in the table, except an application or module. This module will perform some calculation in a purpose for statistical analysis. Therefore, concurrency and security do not impede the use of HDF5 because no user will access those data and security could be implemented in a programming level. One example implementation by manipulating File Access Control Lists in C# [23].
    b. The feeding data application is the one who will insert PD data to the database after doing match and merges from pulse files based on figure 1, is called data combiner. To be able to use HDF5 as data storage, data combiner module must be changed as well, by adding the references to HDF5 library, so then the application could write to the file. There is already a prototype application using C#.NET for inserting PD data to HDF5 and retrieve data from HDF5. Therefore, it should not be a barrier to use HDF5, since a prototype example of how to read and write PD data, already exist.
    c. To prevent from file crashes, the backup feature still not available until 1.10.1 releases. Therefore, backup process still possibly done manually in the server. It is done by create two HDF5 files as primary and secondary data storage. The synchronization process between primary and secondary data storage could be done by enabling the data combiner to insert PD data into those two files.

The list of suggestions for improvement of the HDF5 data storage is listed below:

1. From the benchmark result, it can be concluded that greater number of line of the array can affect the time consumption. Therefore, to reduce the amount of line for each dataset to store partial discharges, it is better to group the partial discharges in one minute will be stored in one row in the array using delimiter.
2. Improving access to multi-dimensional datasets in HDF5 by implementing efficient methods for performing searches for subsets, using semantic information to build multi-dimensional indexes. Indeed, it shows the performance improvements that are gained from indexing the datasets, compared to using the existing HDF library for accessing the data [15].

REFERENCES

[1] Fred Steennis, Ad Kerstens, Theo van Rijn, Jan Mosterd, Leon Bokma, Piet Soepboer, Alfred Arts, Nico van Donk and Branko Carli, "On-line Partial Discharge Monitoring for MV Cables with SCG – Practical Experiences in the Period 2007 to 2010", 8th Int. Conf. on Insulated Power Cables (Jicable), Versailles, France, 19-23 June 2011.

[2] Sibarani, Elisa Margareth, Simanjuntak, Humasak T.A. & Lumbantoruan, Rosni, (2010) "A Solution to Partial Discharges Online (PD-OL) System in KEMA", Thesis Report in Hogeschool Van Arnhem en Nijmegen (HAN) University of Applied Sciences, Arnhem, The Netherlands.

[3] Why HDF? www.hdfgroup.org. [Online] September 23, 2009. [Cited: July 4, 2010.] http://www.hdfgroup.org/why_hdf/.

[4] Oskam, Jaco (2010) PD-Online Architectural Design Document.

[5] Treinish, Lloyd A. Scientific data models for large-scale applications. www.research.ibm.com. [Online] [Cited: September 9, 2010.] http://www.research.ibm.com/people/l/lloydt/dm/DM.htm.

[6] DBMS. www.scribd.com. [Online] [Cited: June 4, 2010.] http://www.scribd.com/doc/6947984/DBMS.

[7] Products. www.hdfgroup.org. [Online] September 23, 2009. [Cited: July 4, 2010.] http://www.hdfgroup.org/products/.

[8] The HDF Levels of Interaction. www.hdfgroup.org. [Online] September 23, 2009. [Cited: July 5, 2010.] http://www.hdfgroup.org/products/hdf4/whatishdf.html.

[9] Scientific Data Management in the Coming Decade. Gray, Jim, et al. Redmond : s.n., 2005.

[10] Chapter 1 The HDF5 Data Model and File Structure. www.hdfgroup.org. [Online] March 12, 2010. [Cited: July 8, 2010.] http://www.hdfgroup.org/HDF5/doc/UG/UG_frame03DataModel.html.

[11] Introduction to HDF5. www.hdfgroup.org. [Online] February 6, 2006. [Cited: July 6, 2010.] http://www.hdfgroup.org/HDF5/doc/H5.intro.html.

[12] Group, Hierarchical Data Format (HDF). HDF5 Tutorial. University of Illinois at Urbana-Champaign (UIUC) : National Center for Supercomputing Applications (NCSA), October 1999.

[13] SQL Server Replication. msdn.microsoft.com. [Online] [Cited: August 2, 2010.] http://msdn.microsoft.com/en-us/library/ms151198.aspx.

[14] Steed, Chad A., Braud, James E and Koehler, Kim A. VGRID: A Generic, Dynamic HDF5 Storage Model for Georeferenced, Grid Data. aser.ornl.gov. [Online] http://aser.ornl.gov/steed/personal/publications/papers/2002_Steed-etal_VGRID_OCEANS.pdf.

[15] Improving Access to Multi-dimensional Self-describing Scientific Datasets. Nam, Beomseok and Sussman, Alan. p. 10.

[16] Comparison of relational database management systems. en.wikipedia.org. [Online] [Cited: August 5, 2010.] http://en.wikipedia.org/wiki/Comparison_of_relational_database_management_systems.

[17] Mainzer, John. RFC: Metadata Journaling to Improve Crash Survivability. www.hdfgroup.org. [Online] August 22, 2008. [Cited: August 1, 2010.] http://www.hdfgroup.org/pubs/rfcs/Metadata_Journaling_RFC.pdf.

[18] SQL Server 2008 R2 Pricing. www.microsoft.com. [Online] [Cited:August 9, 2010.] http://www.microsoft.com/sqlserver/2008/en/us/pricing.aspx.

[19] SQL Server 2008 Troubleshooting and Support. technet.microsoft.com. [Online] [Cited: August 9, 2010.] http://technet.microsoft.com/en-us/sqlserver/bb895929.aspx.

[20] Helpdesk and Mailing Lists. www.hdfgroup.org. [Online] [Cited: August 8, 2010.] http://www.hdfgroup.org/services/support.html.

[21] HDFView. www.hdfgroup.org. [Online] [Cited: July 1, 2010.] http://www.hdfgroup.org/hdf-java-html/hdfview/.

[22] PyTables Getting the most *out* of our data. www.pytables.org. [Online] [Cited: July 8, 2010.] http://www.pytables.org/moin.

[23] Järvinen, Jani. The Basics of Manipulating File Access Control Lists with C#. www.developer.com. [Online] [Cited: August 3, 2010.] http://www.developer.com/net/article.php/10916_3701811_1/The-Basics-of-Manipulating-File-Access-Control-Lists-with-C.htm.

[24] Microsoft, MSDN. Partitioning. MSDN Library. [Online] http://msdn.microsoft.com/en-us/library/ms178148.aspx.

Authors

Elisa Margareth SIBARANI got a Master of Information Systems Development in HAN University of Applied Sciences in The Netherlands. At the moment she is head of and lecturer at the Informatics Engineering Study Program at Del Polytechnic Institute of Informatics in Indonesia. Her research interests are mainly about information systems development, software engineering, database performance and business intelligence.

Paul WAGENAARS received the MSc. degree in electrical engineering from the Eindhoven University of Technology in The Netherlands in 2004. In 2010, he received a PhD. degree from the same university for his research on online partial discharge monitoring of medium-voltage cable systems. Since then he joined KEMA as specialist on power cable technology.

Guido BAKEMA is a graduated theoretical mathematician from Leiden University and a informatician from Nijmegen University, both in The Netherlands. Early 2011 he retired as a professor in information systems development at HAN University of Applied Sciences in The Netherlands. In 2010 he supervised the first author during her final project and master thesis writing at the end of her master study on Information Systems Development in The Netherlands.