An Integrated Approach for Data- and Compute-intensive Mining of Large Data Sets in the GRID

Matthias Röhm, Matthias Grabert and Franz Schweiggert
Institute of Applied Information Processing
Ulm University
Ulm, Germany
matthias.roehm@uni-ulm.de, matthias.grabert@uni-ulm.de, franz.schweiggert@uni-ulm.de

***Abstract*—The growing computerization in modern academic and industrial sectors is generating huge volumes of electronic data. Data mining is considered the key technology to extract knowledge from these data. Grid and Cloud technologies promise to meet the tremendously rising resource requirements of heterogeneous, large-scale and distributed data mining applications. While most projects addressing these new challenges have a strong focus on compute-intensive applications, we introduce a new paradigm to support the development of both compute- and data-intensive applications in heterogeneous environments. Combined storage and compute resources form the basis of this new approach as they allow programs to be executed on resources storing the data sets and thus are the key to avoid data transfer. A data-aware scheduling algorithm was developed to efficiently utilize all available resources and reduce data transfer of global data-intensive applications as well as support compute-intensive applications. Based on the results of the DataMiningGrid project we developed the DataMiningGrid-Divide&Conquer system that combines this approach with current Grid and Cloud technologies into a general-purpose data mining system suited for the different aspects of today's data analysis challenges. The system forms the core of the Fleet Data Acquisition Miner for analyzing the data generated by the Daimler fuel cell vehicle fleet.**

Keywords- data-intensive; data mining; Grid; MapReduce; scheduling.

I. INTRODUCTION

Increasing data volumes in many industrial and academic sectors are fueling the need for novel data analysis solutions [1]. The effective and efficient management and transformation of these data into information and knowledge is considered a key requirement for success in knowledge-driven sectors. Data mining is the key methodology to address these information needs through automated extraction of potentially useful information from large volumes of data. In the last decade there have been multitudes of efforts to scale data mining algorithms for solving more complex tasks, including peer-to-peer data mining, distributed data stream mining and parallel data mining [2] [3].

Recently, data mining research and development has put a focus on highly data-intensive applications. Google's publications on MapReduce [4][5], a special incarnation of the divide&conquer paradigm, inspired many projects working on large data sets. MapReduce frameworks, like Hadoop [6], simplify the development and deployment of peta-scale data mining applications leveraging thousands of machines. MapReduce frameworks are highly scalable because they avoid data movement and rather send the algorithms to the data. In contrast to other data mining environments these frameworks restrict themselves to a certain programming model, loosing some of the functionality provided by fully featured distributed data mining systems.

Another branch of modern distributed data mining is motivated by the sharing of heterogeneous, geographic distributed resources from multiple administrative domains to support global organizations [7] [8] [9]. This field of active research and development is generally referred to as data mining in Grid computing environments. The DataMiningGrid project addresses the requirements of modern data mining application scenarios arising in Grid environments, in particular those which involve sophisticated resource sharing. The DataMiningGrid system is a service-oriented, scalable, high performance computing system that supports Grid interoperability standards and technology. It meets the needs of a wide range of users, who may flexibly and easily grid-enable existing data mining applications and develop new grip-based approaches. The DataMiningGrid, like most related Grid systems, focused on compute-intensive applications following design principles that are correct for compute-intensive, but not for data-intensive applications. For compute-intensive application scenarios the main resource and the limiting factor is CPU-power and the focus of the system is to provide a transparent integration of multiple compute clusters. In these scenarios it is commonly assumed that the time needed to transfer the input and output data is relatively small compared to the overall execution time. These assumptions lead to an architecture build on three main components:
(1) Specialized storage servers to store input and output data as well as executables. (2) A set of compute clusters from different organizations each composed of multiple compute nodes for running the algorithms. To provide a high level of transparency, these clusters are treated as one multi-CPU resource. (3) Grid management servers for accessing and

managing the storage and compute clusters of the local organization.

In such environments data is stored on dedicated storage servers and has to be transferred to the compute nodes prior to execution. Though different scheduling algorithms have been proposed to optimize the relation between data transfer and execution time [10][11], for data-intensive applications where the limiting factor is not CPU-power but rather storage and network speed, data should not be moved at all [12].

To bring the advantages of the MapReduce paradigm into worldwide, heterogeneous computing environments we developed the DataMiningGrid-Divide&Conquer (DMG-DC) [1] system based on the concepts and services of the DataMiningGrid project. This article covers a detailed description of the scheduling algorithm and the Grid components to implement this new approach that combines different current data mining technologies into a single system. This article is organized as follows: First, we briefly revise MapReduce frameworks and introduce the more general divide&conquer paradigm for data-intensive applications in Grid environments. Then we describe the DataMiningGrid and its successor, the DMG-DC system. We also introduce a real-world data mining application based on the DMG-DC: The Fleet Data Acquisition Miner (FDA-Miner) for analyzing the data generated by the Daimler fuel cell vehicle fleet. Finally, we present system evaluation results from the FDA-Miner and discuss related technologies.

II. MapReduce and data-intensive Divide&Conquer

The tremendous amount of data generated in modern science and business applications require new strategies for storing and analyzing. As the amount of data increases, data can not be efficiently stored on a single storage server but has to be distributed over multiple machines. Google's MapReduce[4] and its open-source implementations provide frameworks to mine these distributed data sets.

The name MapReduce refers to the map and reduce functions of functional programming languages. In the context of a MapReduce framework, all applications consist of a map and a reduce function [4]. The map function reads a key/value pair and produces a set of new key/value pairs. In an intermediate step all pairs are grouped by their key values. A key and its values are presented to the reduce function which produces a list of result values.

These functions are supposed to produce the same result when applied to the whole data set or to the parts of the data set.

MapReduce frameworks build an environment for executing these map and reduce functions on a cluster. Data is split up into small chunks and stored in a distributed file system comprised of multiple standard machines acting as storage *and* compute nodes [5]. A special manager node keeps track of all data chunks and their locations in the cluster. A master process manages the execution and minimizes data movement by executing the functions on the nodes containing the data to be mined. The master identifies the nodes to use for execution by asking the distributed file system manager for the location of the data chunks. If multiple copies of a chunk are available the master schedules the execution to the least used node.

Executing the functions on the nodes that contain the data is the key to the high performance and scalability of MapReduce frameworks. As not data, but algorithms are transferred, MapReduce frameworks are perfectly suited for Clouds because they do not require information about server location and network bandwidth as traditional systems need for data scheduling.

Restricting themselves to only two functions, MapReduce frameworks are easy to program and simple to set up. However, not all data-intensive applications can be decomposed into map and reduce functions. Especially the integration of existing data mining programs including compute-intensive applications is sometimes impossible. In addition, current MapReduce-Frameworks are implemented for clusters within a single organization and are not suitable for loosely-coupled environments comprised of heterogeneous, geographic distributed resources from multiple administrative domains.

A. Divide&Conquer for data-intensive Grid applications

Recent Grid implementations provide an ideal infrastructure for inter-domain resource sharing but, as mentioned above, are geared towards compute-intensive applications. To utilize Grid systems for a wide range of data- and compute-intensive applications in such environments a different distributed computing paradigm is needed.

MapReduce can be viewed as a special form of the divide&conquer paradigm, where a problem is split into smaller sub problems that are easier to solve. Compute-intensive applications also use a divide&conquer technique to leverage the performance of compute clusters by splitting compute-intensive problems into smaller sub problems and compute these sub problems on multiple resources simultaneously. Compared to MapReduce, the general divide&conquer paradigm does not impose any restrictions on the functions or the number of processing steps and is therefore more suitable for a general purpose distributed data mining system. A data-intensive Divide&Conquer Grid processing model *DCG* might be defined as follows:

(1) The data set D to be processed can be decomposed into m subsets $d_1, \ldots, d_m$ ($\bigcup_{i \leq m} d_i = D$) and stored on multiple storage resources.

(2) An arbitrary function φ is applied to all m subsets of D in parallel. The execution of φ on resource r is denoted with φ_r.

(3) Assuming that data transfer is expensive, the function φ

is executed on a resource r which is closest to the storage of d_i

$$\varphi_r(d_i) = e_i,\ d_i \in D,\ r \in R, r \text{ closest to } d_i$$

and the results E may be processed by another function φ',

$$\varphi'_q(e_i, e_j, ...) = h_i,\ e_i, e_j, ... \in E,\ q \in R, q \text{ close to } e_i, e_j, ...$$

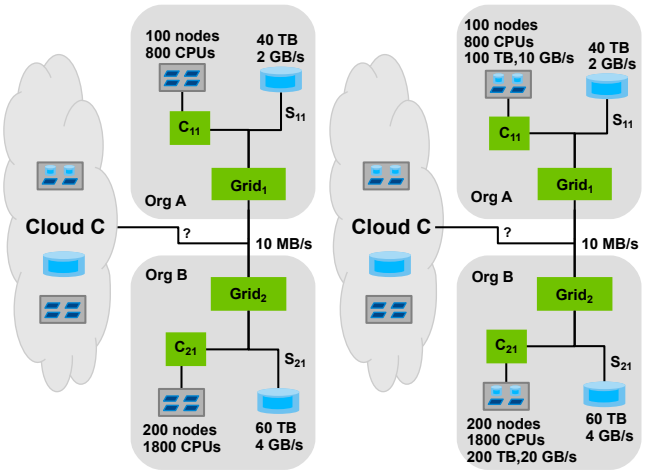
generating the result set H, which again may be processed by another function.

(4) A series of such execution steps can be represented by a direct acyclic graph, where each node is a function and the vertices symbolize the data flow between the functions.

We identified the need for three major conceptual enhancements to traditional Grids when applying the DCG approach to data-intensive applications in Grid environments:

1) *Any storage or computational resource may become a combined storage/compute resource.* This resource type forms the basis of scalable data-intensive applications as data can be processed directly on the storage location. To increase storage capacity and speed, computational resources of compute clusters may become combined resources by storing data on their local disks. It is important to point out that the combined resources are suitable for data- *and* compute-intensive applications as only new functionality is added. Combined resources fundamentally differ from current Grid concepts which impose a strict differentiation between storage and compute resources.
2) *A combined resource may provide Grid data transfer mechanisms.* As combined resource will mostly be organized within a cluster, data transfer out of the cluster is often not desired or not supported. This, again, differs from traditional Grids where each Grid storage has to implement Grid data transfer mechanisms (e.g., GridFTP).
3) *A data processing methodology avoiding input data to be transferred.* The scalability of data-intensive applications is mainly limited by two factors: the storage and network speed. Combined resources are the key to increase the overall storage speed of the system as each combined resources contributes its local storage. Avoiding input data transfer through processing the data directly on the resources storing the data, minimizes network traffic and therefore increases the scalability for data-intensive applicaitons [12].

To illustrate the benefits of these concepts for data-intensive applications, consider the example depicted in Figure 1 where a common Grid setup is shown on the left and the new concepts on the right. In a traditional Grid setup multiple clusters from different organisations are connected through Grid servers hosting the Grid middleware. Each compute cluster is treated as one single resource by the Grid middleware and is managed by a local cluster management



(a) Common Grid setup (b) DCG setup

Figure 1. A traditional Grid setup compared to the Divide&Conquer concepts.

system like Condor, LSF or SGE. Dedicated storage systems are used to store the input and output data of the compute clusters. These storage systems are optimized to support many concurrent connections for reading and writing to a single large file. Cloud resources may also be integrated into the local clusters to increase storage or compute capacity. A setup like this fits the needs of many compute-intensive applications: First an input file, or a part of it, is read by each compute resource, then a model is computed for long time and finally each compute resource writes its part of the overall solution into the output file.

The situation changes in a Grid environment with multiple compute clusters from different organisations or Cloud providers. When a compute-intensive application should be executed on a compute cluster in organisation A - or in the Cloud C - and the input data is stored in organisation B the data has to be transferred from the storage system in B to the one in A or C because a cluster can not directly access data on a storage system of another Grid site. The required data input and output transfers are often handled via special pre- and post-processing steps. As it is commonly assumed that the data transfer time is small compared to the execution time the transfer overhead is accepted. For most data-intensive applications this assumption can not be hold. In contrary, not CPU-power but storage speed is the limiting factor of many data-intensive applications. When applying the divide&conquer technique to data-intensive problems, large data sets are split into smaller subsets and distributed over multiple storage systems. An application may then process all subsets in parallel and the overall processing speed scales with the number of storage resources. In a Grid environment with a strict distinction between compute and storage resources, the only way of implementing this

approach is to add new storage resources. New dedicated storage resources not only increases costs but also the network bandwidth becomes a limiting factor - especially when resources from other Grid sites or the Cloud are used - because data always have to be transferred from a storage to a compute resource.

Here the combined resources of the DCG come into the picture. As shown on the right of Figure 1, storing data subsets on the compute nodes of a cluster saves costs, because existing resources are utilized, and increases storage space as well as the overall storage speed. In addition, data can be processed and stored on a single resource making data transfer superfluous. The combined resources are still managed by the grid middleware and the local cluster management system so that both data- and compute-intensive applications may run simultaneously on the same cluster.

The following example illustrates the differences between a setup with separated storage and compute resources (1) and with combined resources (2): An application needs to scan through 20TB to find special patterns in the data.
(1) The data is stored on storage resource S_{11}. So the input data has to be transferred to one of the compute clusters C_{11}, C_{21} or the Cloud. The scheduler decides to use the local cluster C_{11} as the connections to the other Grid instance and the Cloud have limited bandwidth and the network load is not known exactly. The cluster management system starts the data analysis program on all available resources of C_{11}. At best all 100 compute resources start reading a portion (200GB) of the input data. As the storage system S_{11} provides a maximum speed of 2GB/s, each resource processes about 20MB/s. The overall time for processing the 20TB is roughly 2.8 hours.
(2) The data was split into 20GB chunks and distributed over the 300 combined resources of the cluster C_{11} and C_{21} each now providing 1TB of storage with a speed of 100MB/s. For redundancy each chunk is stored on at least two different resources halving the overall storage space to about 150TB. To scan through the 20TB the scheduler advices the cluster management system to start the analysis program for each data chunk on a resource of C_{11} or C_{21} storing the data subset. In this setup the scheduler can choose resources from different Grid sites or the Cloud for processing as the input data has not to be transferred at all. At best 100 combined resources immediately start processing the locally stored data chunk at a speed of 100MB/s each and a total speed of 10GB/s. The overall time for processing the 20TB with combined resources comes down to about 0.6 hours.

B. A scheduling algorithm for DCG

The described data-intensive divide&conquer processing model requires not only new functionality to manage combined resources but also a method to schedule the execution of a program to a resource that is *closest* to the data. As there was no algorithm available to schedule programs close to data sets within a Grid, we developed the flexible DCG scheduling algorithm, which only needs information about the compute and storage power of the combined resources.

The inputs of the DCG scheduling algorithm depicted in Figure 2 are the set of input data D, the program p, the data locality weights λ_1, λ_2, the data transfer scheduling weights α_1 to α_4 and the data to compute ratio weights β_1, β_2. First, the scheduler obtains the current grid status and sorts the data sets of D in descending order. The ordering of D assures that the larger data sets are scheduled first and the scheduler may therefore choose among more free resources. After this init phase, the algorithm schedules each data subset d of the ordered input data D'. The main goal of the scheduler is to find a resource tuple $(\hat{r}, \hat{s})$ where: $\hat{r} \in R$ is the best (highest compute speed g_c) available execution resource that is able to execute p; and $\hat{s} \in R^d$ is the storage resource storing d with a minimal data transfer overhead to $\hat{r}$. Due to the special properties of the *Data Distance Function* f_s used to compute the data transfer overhead, the algorithm only needs to consider four execution resources per storage resource s. These four execution resources are: the storage resource s itself, the best resource within the same cluster r_c, the best resource within the same grid r_g and the best resource outside the local grid instance r_a. From the resulting $4 \cdot |R^d|$ candidate $(r,\ s)$ resource tuples the scheduler chooses the one with the highest priority as computed by the *Normed Priority Function* f_p.

The algorithm and functions are based on the following definitions:
$P :=$ all programs available in the Grid;
$R := \{r_1, \ldots, r_n\}$ is the set of all n resources of the grid;
$D := \{d_1, ..., d_m\}$ is the m data sets of the job;
$Q := \{(r, s) | r, s \in R$, r and s can exchange data directly$\}$;
$R^d := \{\ r \mid r \in R$ stores $d \in D\ \}$;
$D^r := \{\ d \mid d \in D$ is stored on $r \in R\ \}$;
$\delta_c : R \rightarrow \{1, \ldots, n_c\}$ assigns a unique number to each of the n_c clusters;
$\delta_g : R \rightarrow \{1, \ldots, n_g\}$ assigns a unique number to each of the n_g grid sites;
$g_s(Z, d, r)$ is the storage speed of resources r with respect to d defined as $g_s(Z, d, r) \geq 0$ if d is stored on r;
and $g_c(Z, p, r)$ is the compute power of resource r with respect to program p defined as $g_c(Z, p, r) \geq 0$ if r fulfills all requirements of p. Different properties of a resource may be used to define the computing and storage power of a resource, but at least the current usage - included in the overall system state Z - has to be taken into account.

The data transfer overhead of a candidate resource tuple r, s is computed by the *Data Distance Function* f_s which assigns the weights α_1 to α_4 to the storage power of s according to the distance between s and r: α_1 if d is stored on the resource itself $r = s$; α_2 if d is stored on a resource on the same cluster $\delta_c(r) = \delta_c(s)$; α_3 if

FUNCTION DCG ($p \in P$, D, α_{1-4}, β_{1-2}, λ_{1-2})
  $Z \leftarrow$ current state of the grid
  $D' \leftarrow D$ ordered by size
  $\hat{C} \leftarrow \emptyset$
  for all $d \in D'$ **do**
    $C' \leftarrow \emptyset$
    for all $s \in R^d$ **do**
      r_c with $g_c(Z,p,r_c) = \max\{g_c(Z,p,r) \mid r \in R \wedge \delta_c(r_c) = \delta_c(r)\}$
      r_g with $g_c(Z,p,r_g) = \max\{g_c(Z,p,r) \mid r \in R \wedge \delta_g(r_g) = \delta_g(r)\}$
      r_a with $g_c(Z,p,r_a) = \max\{g_c(Z,p,r) \mid r \in R \wedge \delta_g(r_a) \neq \delta_g(r)\}$
      $C \leftarrow C \cup \{ (s,\ s,\ f_s(Z,d,s,s),\ g_c(Z,p,s)),$
        $(s,\ r_c,\ f_s(Z,d,s,r_c),\ g_c(Z,p,r_c)),$
        $(s,\ r_g,\ f_s(Z,d,s,r_g),\ g_c(Z,p,r_g)),$
        $(s,\ r_a,\ f_s(Z,d,s,r_a),\ g_c(Z,p,r_a)) \}$
    end for
    Find $\hat{c} \in C$ with $f_p(\hat{c}) = \max\{f_p(c) \mid c \in C\}$
    if $f_p(\hat{c}) \leq 0$ **then**
      print Could not schedule d
    else
      $\hat{C} \leftarrow \hat{C} \cup \{\hat{c}\}$
      $Z \leftarrow$ current State of Z after choosing $\hat{c}$
    end if
  end for
  return $\hat{C}$ #Return schedule $\hat{C}$ containing the choices $\hat{c}$ for each scheduled data set
END FUNCTION

Figure 2. DCG algorithm for scheduling a program for each data set

d is on the same Grid instance $\delta_g(r) = \delta_g(s)$; and α_4 if d is on another Grid instance $\delta_g(r) \neq \delta_g(s)$. In case both resources are not able to exchange data directly, each resource needed to transfer the data set d from s to r is also considered.

FUNCTION $f_s(Z, d \in D, s \in R^d, r \in R)$
  $t \leftarrow t_\infty$
  if $(r,s) \in Q$ **then**
    $t \leftarrow |d|/g_s(Z,\ s)$
  else if $\exists\ s_1, \ldots, s_p$ with (s,s_1), ..., (s_{p-1}, s_p), (s_p, r) $\in Q$ **then**
    Choose shortest $s_1, \ldots, s_q$ with $(s,s_1), \ldots (s_q, r) \in Q$
    $t \leftarrow |d|/g_s(Z,\ s) + \sum_{i=1}^{q} |d|/g_s(Z,\ s_i)$
  end if
  if $r = s$ **then**
    $t \leftarrow \alpha_1 \cdot t$
  else if $\delta_c(r) = \delta_c(s)$ **then**
    $t \leftarrow \alpha_2 \cdot t$
  else if $\delta_g(r) = \delta_g(s)$ **then**
    $t \leftarrow \alpha_3 \cdot t$
  else
    $t \leftarrow \alpha_4 \cdot t$
  end if
  return t
END FUNCTION

As can easily be seen the data transfer scheduling weights α_1 to α_4 may be used to enforce specific data transfer policies:

- $\alpha_1 > 0$, $\alpha_2 = \alpha_3 = \alpha_4 = t_\infty$ forces the scheduler to choose a resource storing the data set d regardless of its speed.
- $\alpha_1 \leq \alpha_2$, $\alpha_3 = \alpha_4 = t_\infty$ forces the scheduler to choose a resource storing the data set d or a resource in the same cluster.
- $\alpha_1 \leq \alpha_2 \leq \alpha_3$, $\alpha_4 = t_\infty$ forces the scheduler to choose a resource belonging to the same Grid instance favoring resources storing the data or resources in the same cluster.
- With $\alpha_1 \leq \alpha_2 \leq \alpha_3 \leq \alpha_4$ the scheduler may choose any resource in the Grid, but prefers resources close to the data storage location d.

The *Normed Priority Function* is used to compute the priority of each resource as its data transfer overhead and its compute power. In addition, a *data locality* may be included into the priority calculation. The priority of each

tuple (s, r) is the linear combinaton of the weighted transfer time ($f_s(Z, d, s, r)$), the compute speed ($g_c(Z, p, r)$) and the cluster and grid data locality factors (u_{clu}, u_{grid}). To control the influence of each value on the scheduling the values are normed to $[0 : 1]$ using a non-linear norming function g_n and multiplied with a weight, where β_1 describes the importance of the data, β_2 the weight of the compute power and λ_1, λ_2 control the influence of the data locality in the scheduling process. The data locality factors u_{clu}, u_{grid} describe how much of the input data D is stored within the cluster or grid of the resource r and how fast the storage is. The factors are especially useful if the results of the execution will be combined by a subsequent step of the execution graph.

FUNCTION $f_p(r \in R, S_r \in R^D, t_r \in \Re, v_r \in \Re)$
  if $t_r \geq t_\infty \ \vee \ v_r \leq 0$ **then**
    return 0
  end if
  $u_{clu} \leftarrow \ \sum_{d \in D} \sum_{s \in R^d \wedge \delta_c(r) = \delta_c(s)} \frac{|d|}{|D|} \cdot g_s(Z, s)$
  $u_{grid} \leftarrow \ \sum_{d \in D} \sum_{s \in R^d \wedge \delta_g(r) = \delta_g(s)} \frac{|d|}{|D|} \cdot g_s(Z, s)$

  $q \leftarrow \beta_1 \cdot g_n(t_r) + \beta_2 \cdot g_n(v_r) + \lambda_1 \cdot g_n(u_{clu}) + \lambda_2 \cdot g_n(u_{grid})$
  return q
END FUNCTION

Figure 3 shows the more general DCG-MD scheduling algorithm for applications requiring multiple input data sets. In this scenario all data sets have to be available on one execution resource. First, the algorithm produces the set of candidate execution resources as the best (highest compute speed g_c) resources from all n_c clusters in the Grid and all resources storing one of the data sets $d \in D$. For each of these resources the set of storage resources with minimal transfer overhead with regard to D is generated. From all candidates the scheduler chooses the one with the highest priority f_p.

In order to support a wide range of data- and compute-intensive Grid applications without re-implementing everything from scratch it was decided to integrate the developed DCG approach into an existing grid-based data mining system. As the DataMiningGrid system described in the next section already provides many functionalities for grid-based data mining it was chosen as a basis for the DMG-DC system. Because of the missing support for data-intensive DCG applications it had to be enhanced with: (1) A distributed data registry to store, manage and locate all data subsets. (2) A resource broker implementing the scheduling algorithm described in this section. (3) A workflow manager to coordinate multiple dependant execution steps.

III. Data Mining in the Grid: DataMiningGrid

In general, a grid-enabled data mining system should support the seamless and efficient sharing of data, data mining application programs, processing units and storage devices in heterogeneous, multi-organizational environments. As data mining is used by a wide variety of users and organizations such a system should not only address the technical issues but also pay attention to the unique constraints and requirements of data mining users and applications. In the DataMiningGrid[7] project, use case scenarios from a wide range of application areas were analyzed to identify the key requirements of grid-based data mining that can be summarized as follows:

- A grid-based data mining environment should offer benefits like increased performance, high scalability to serve more users and more demanding applications, possibilities for creation of novel data mining applications and improved exploitation of existing hardware and software resources.
- Grid-enabling data mining applications should not require modification of their source code. The system should not be restricted to specific data mining programs, tools, techniques, algorithms or application domains and should support various types of data sources, including database management systems (relational and XML) and data sets stored in flat files and directories.
- To support the different user groups, intricate technological details of the Grid should be hidden from domain-oriented users, but at the same time users with a deep knowledge of Grid and data mining technology should be able to define, configure and parameterize details of the data mining application and the Grid environment.

In order to address these requirements, the DataMiningGrid system was designed according to three principles: services-oriented architecture (SOA), standardization and open technology. The early adoption of two important distributed computing standards, the Open Grid Service Architecture[13] (OGSA) and the Web Services Resource Framework[14] (WSRF)were essential for succeeding projects, like the one presented in this article. The OGSA is a distributed interaction and computing architecture based on the concept of a Grid computing service, assuring interoperability on heterogeneous systems so that different types of resources can communicate and share information. The WSRF refers to a collection of standards which endorse the SOA and proposes a standard way of associating Grid resources with web services to build stateful web services required by the OGSA.

Following these principles, the DataMiningGrid project implemented various components based on existing open technology: Data management, security mechanisms, execution management and other services commonly needed in Grid systems are provided by the Globus Toolkit 4 (GT 4) Grid middleware.

Three higher-level components for data, information and execution management form the core of the DataMiningGrid

FUNCTION DCG-MD ($p \in P$, R, D, α_{1-4}, β_{1-2}, λ_{1-2})
  $Z \leftarrow$ current state of the grid
  $R_{max} \leftarrow \{\hat{r}_i \,|\forall i \leq n_c : \hat{r}_i \in R \wedge \delta_c(\hat{r}_i) = i \wedge g_c(Z,p,\hat{r}_i) = \max\{g_c(Z,p,r)|r \in R \wedge \delta_c(r) = i\}\}$
  $R_{sp} \leftarrow \{\, r \,|\, \forall d \in D : r \in R^d \;\wedge\; g_c(Z,p,r) > 0 \,\}$
  $C \leftarrow \emptyset$
  for all $r \in R_{max} \cup R_{sp}$ **do**
    $Z' \leftarrow Z\,, \quad t_d \leftarrow 0\,, \quad S \leftarrow \emptyset$
    for all $d \in D$ **do**
      Find $\hat{s} \in R^d$ with $f_s(Z', d, \hat{s}, r) = \min\{f_s(Z', d, s, r)|s \in R^d\}$
      $t_d \;\leftarrow\; t_d + f_s(Z',\, d,\, \hat{s},\, r)$
      $S \;\leftarrow\; S \cup \{\hat{s}\}$
      $Z' \;\leftarrow$ current State of Z' after choosing $\hat{s}$
    end for
    $C \;\leftarrow\; C \cup \{(\, r,\, S,\, t_d,\, g_c(Z', p, r)\,)\}$
  end for
  Find $\hat{c} \in C$ with $f_p(\hat{c}) = \max\{f_p(c) \,|\, c \in C\}$
  return $\hat{c}$
END FUNCTION

Figure 3. DCG algorithm for scheduling a program with multiple data sets

system. The *data components* offer several data transformation and transportation capabilities to support typical data operations for data mining applications based on OGSA-DAI[15] and the Globus Toolkit data management components. The *Information Service* collects and manages all information about the data mining programs available in the system. The *Resource Broker* is responsible for matching available resources to job requests, global scheduling of the matched jobs and executing, managing and monitoring of jobs, including data stage in and out operations.

The main user interface of the system is the Triana workflow environment [16]. Triana provides a workflow editor and manager to design and execute complex workflows. Several DataMiningGrid workflow components build the interface to the GT 4 Grid infrastructure services as well as the DataMiningGrid services. The most important components allow the users to search for an application, configure its parameters, select and transform the input data sets and finally execute the configured task on the Grid.

The DataMiningGrid Application Description Schema (ADS) is the link between these DataMiningGrid components. An ADS instance covers the complete life-cycle of a data mining task. The XML document is devided into four parts: A description, information about the executable, requirements as well as monitoring and accounting information. The description contains data mining specific information about the implemented algorithm following the CRISP-DM standard and is mainly used for discovering. The next section contains all information regarding the executables implementing the algorithm, including the required libraries, options and parameters for configuring the executables. The requirements section holds the hardware and software restrictions imposed by the implementation of the executable or the user. Throughout the execution the ADS contains monitoring information and after the execution is finished, the ADS instance also stores all information related to the execution environment.

Although the necessity to address data-intensive applications was recognized in the DataMiningGrid project, due to time constraints, the project focused more on compute-intensive applications. Hence, three functionalities needed for DCG jobs are not available in the DataMiningGrid and related systems:

1) There is no server-side workflow execution component to coordinate the steps of DCG jobs.
2) There is no specialized data registry that could be used for scheduling data-intensive applications.
3) The Resource Broker [17], like other Grid resource brokers, does not provide a scheduling mechanism for combined resources and is only able to schedule and execute jobs on a clusters level. As a consequence, jobs can not be placed directly on combined storage/compute resources inside a cluster, as required by DCG jobs.

The following section describes the changes made to the DataMiningGrid system as well as the new components and features of the DMG-DC system to natively support data-intensive applications in Grid environments.

IV. DataMiningGrid Divide&Conquer system

The DMG-DC system is designed to support the different aspects of today's data analysis challenges, including compute- *and* data-intensive applications. Combined storage and compute resources form the basis of the DMG-DC system, allowing data to be stored and processed on any machine in a Grid. With this approach there is no need to transfer the input data from a storage server to a compute node prior to execution, which can significantly speed up data-intensive applications.

V. DMG-DC

Unfortunately, as discussed in the previous sections, current grid-based systems do not provide the functionality to support these combined resources because of the focus on compute-intensive applications. As a consequence, there are two different approaches to design a Grid system for combined resources that supports both, compute- and data-intensive applications: (1) Implement all needed functionality as a set of new services from scratch; (2) use or enhance existing systems and services where possible.
The architecture proposed in this article follows the later approach and builds on the DataMiningGrid system in combination with standard Grid infrastructure services. This not only reduces the implementation time but also ensures the compatibility of the system with newer versions of the Grid infrastructure.

The DataMiningGrid project already implements many features needed for grid-based data mining. The flexible and extendable design of the DataMiningGrid system made it easy to integrate the missing functionality to support data-intensive applications. Consequently, the architecture of the DMG-DC, depicted in Figure 4, does not differ significantly from the DataMiningGrid architecture [7]. The client components, the Triana workflow environment and two web-based applications, are build on top of the DMG-DC services and may also directly interact with the Grid infrastructure services. The DMG-DC services, the *Information Integrator Service* (IIS), the *Data Registry Service* (DRS) and the *Workflow Resource Broker* (WRB), provide all additional functionality to support DCG jobs in a standard Grid environment. The IIS manages all available data-mining algorithms as ADS instances and provides an interface to add and remove ADS descriptions. The DRS and the WRB services are the main building blocks to execute DCG jobs. The DRS manages all information about each data set in the Grid, including the storage location, and provides powerful functions to search for data sets. The WRB schedules and manages multiple Grid jobs in the correct sequence. In combination with the DRS the matchmaking and scheduling functions of the WRB are able to execute data-intensive jobs directly on the Grid nodes storing the data. The DRS and WRB are build on top of the Grid layer and use various

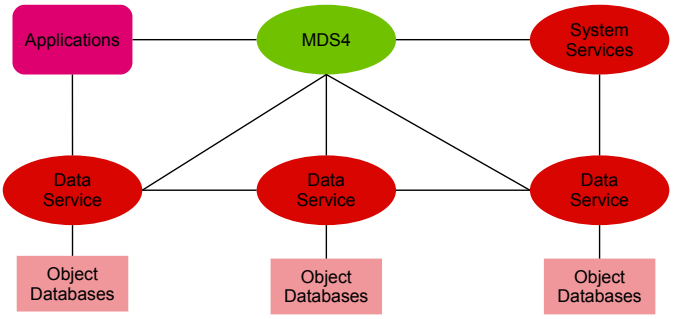


Figure 5. The DMG-DC Data Registry Service architecture

GT 4 Grid services like the MDS4, the RFT or the WS-GRAM [18]. The Grid layer also provides all functionality to integrate the resources of the different organisations into a single virtual organisation. In most setups all resources of an organisation are managed by one Grid server instance that is connected to the grid servers of all other organisations. Multiple compute or combined resources within an organisation are typically controlled by a cluster management system like Condor, Sun Grig Enging or LSF.

A. Data Registry Service

The data registry is the central component for executing DCG jobs in a Grid, as it provides the locations of all data sets to the Resource Broker. Without this information the Resource Broker would not be able to schedule jobs to the nodes containing the data to be mined. The developed distributed registry consists of a number of WSRF-compliant Data Registry Services that store user-defined metadata describing the data sets available in the Grid. In contrast to the distributed file system of a MapReduce framework, the DRS only stores information about the data, leaving the actual storage to database management or file systems. Therefore, DCG jobs can process data stored in any storage system and are not limited to a specific distributed file system.

The DRS stores metadata in user-defined categories which specify a list of logical and physical attributes describing the data. Logical attributes hold information about the content or creation process of the data set, whereas physical attributes include storage location, size or data format information. When a new data set is registered with a category, a logical and a physical object is created with unique object names. These objects contain the logical/physical attributes of that data set and are used to model replication: A logical object references one or more physical objects.

A single DRS may store the metadata information of all data sets in the Grid. To improve reliability and performance several DRS may run on different sites in the Grid. As depicted in figure 5 multiple DRS automatically form a peer-to-peer network, forwarding client search requests and category information to the appropriate DRS.

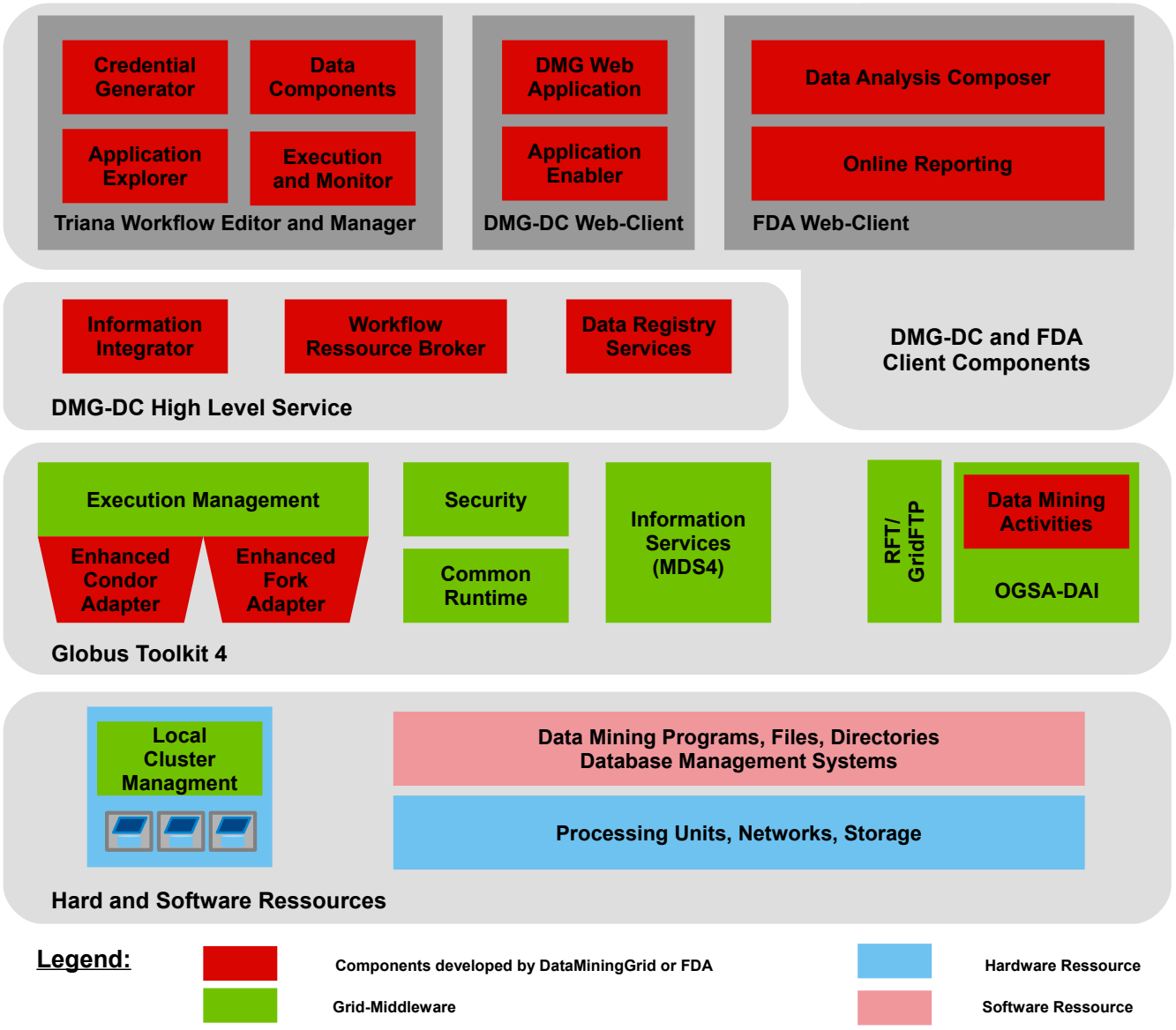


Figure 4. The DMG-DC architecture

As many data analysis applications need mechanisms to select and process arbitrary subsets of the data stored in the Grid, the DRS provides an advanced search enabling clients to search for data using multiple attributes within a single query. This distinctive feature of the DRS is not available in other Grid data registries like the Globus Toolkit Replica Location Service [19].

B. *Workflow Resource Broker*

The new requirements arising from DCG jobs led to the development of the DMG-DC Workflow Resource Broker. The WRB was designed not only to support DCG jobs but also to include all features of current Grid resource brokers like the DataMiningGrid Resource Broker [17]. The two central new features of the WRB are the workflow execution manager and the advanced job scheduler, able to schedule jobs close to the data.

As depicted in Figure 6, the WRB consist of 5 components communicating through well-defined interfaces:
Clients connect to the *workflow manager* to submit workflows, monitor and manage workflow execution. A workflow consists of one or more jobs, each described by an ADS instance, and dependencies between these jobs. The workflow manager is responsible for executing all jobs as specified in the workflow. To start the execution of a single job, the workflow manager sends the corresponding ADS instance to the *ADS execution* component. The execution

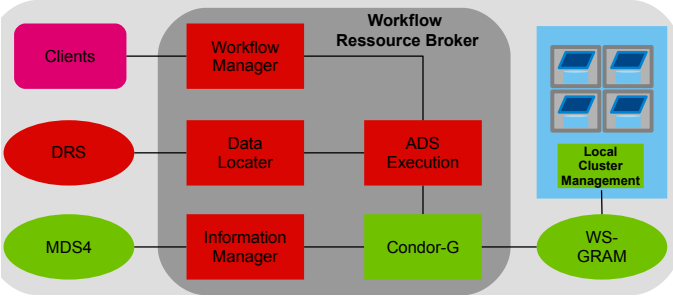


Figure 6. The components of the Workflow Resource Broker.

component analyzes the ADS instance and connects to the *data locator* to get the locations of all data sets specified in the ADS instance. The data locator acts as an interface to different data registries, although currently only DRS is supported. The execution component combines the data locations with the ADS definitions and generates a Condor-G job description. In addition to the standard job description parameters, like executable and arguments, the generated description also contains a Condor-G representation of the developed scheduling algorithm enabling DCG jobs to run on nodes storing the selected data sets.

Condor-G [20] is a powerful Grid task broker providing advanced scheduling, execution and managing capabilities as well as an uniform interface to different Grid execution management systems. A key feature of Condor-G is its ClassAd mechanism to describe jobs and compute resources with specific ClassAd attributes for jobs and compute resources. Based on these descriptions jobs can be matched and scheduled to suitable compute resources. ClassAds can also be used to implement various different scheduling policies through defining $requirements$ and $rank$ expressions. As current Grid implementations do not provide resource information as ClassAds we developed the *information manager*. The information manager collects all information for each computational resource from the Grid information system - currently only an interface to the Globus Toolkit Monitoring and Discovery System (MDS4) is implemented - and translates it into ClassAds. To implement the Divide&Conquer scheduling algorithm the information manager adds two new ClassAds for combined resources: the $ClusterInstance$ and the $GridInstance$. With these new ClassAds, each combined resource can be uniquely identified and directly used for scheduling.

When receiving a job, Condor-G matches the job with all available resources, as defined by the respective resource and job ClassAds. DCG jobs contain special $requirements$ and $rank$ expressions that configure the Condor-G scheduler according to the scheduling algorithm described above. After the matchmaking step, the job is submitted to the execution management service - in case of the Globus Toolkit, the WS-GRAM - of the Grid instance providing the best match. The job submitted to the WS-GRAM contains an element instructing the WS-GRAM to produce a job description for the local cluster management system - currently only Condor is supported - that insures the job is only started on the chosen resource. The WS-GRAM of this instance then submits the job to the cluster management system that manages the chosen resource. Figure 7 shows the interaction of all components while executing a single DCG job: (1) A client searches for suitable algorithms in the MDS4 and receives an ADS description. (2) The client configures the algorithm parameters, the input/ouptut data and optionally the scheduler parameters α_x and λ. Input data can be specified by logical object names or a search query. Steps (1) and (2) may be repeated several times to compose a complex workflow. (3) The client submits a single ADS or a workflow description to the WRB. (4) The WRB parses the workflow and starts processing of the root ADS descriptions of the workflow. First it retrieves the current status of all compute resources from the MDS4. Then it queries the DRS for the physical attributes, including the storage location, of all input data. Based on this information it configures a Condor-G job description implementing the developed scheduling algorithm and starts the scheduling. For each of the 4 data subsets the scheduler initiates the execution on one of the combined resources storing the data. (5) As the 4 chosen resources are located in the organisations A and B Condor-G creates WS-GRAM job descriptions and submits them to the WS-GRAMs. (6-7) The WS-GRAMs parses the description and initiates the transfer of the executables from organisation D to the local storage. (8) The WS-GRAM translates the job description to the format of the local cluster management system and submits it. (9-10) The local cluster management system parses the job description with the restriction to only use the chosen resources and starts the execution on these resources. (11a - 11c) The local cluster management system monitors the execution. The WS-GRAM periodically polls the status of the job from the local cluster management system. Condor-G in turn polls the WS-GRAMs and provides the information to the WRB. (12-13) As the user specified a storage resource in organisation C as the output data location, the WS-GRAM initiates the transfer of the results.

VI. FDA-Miner

The presented DMG-DC system forms the basis of the Fleet Data Acquisition Miner (FDA-Miner) for analyzing the data generated by the Daimler fuel cell vehicle fleet. The Daimler AG has been involved in fuel cell technology for more than 15 years and has released the largest fleet of zero emission fuel cell vehicles in the world with more than 100 vehicles [21]. The purpose of these operations is to test these vehicles in the hands of selected customers in everyday operations under varying climatic conditions, traffic conditions and driving styles in different locations

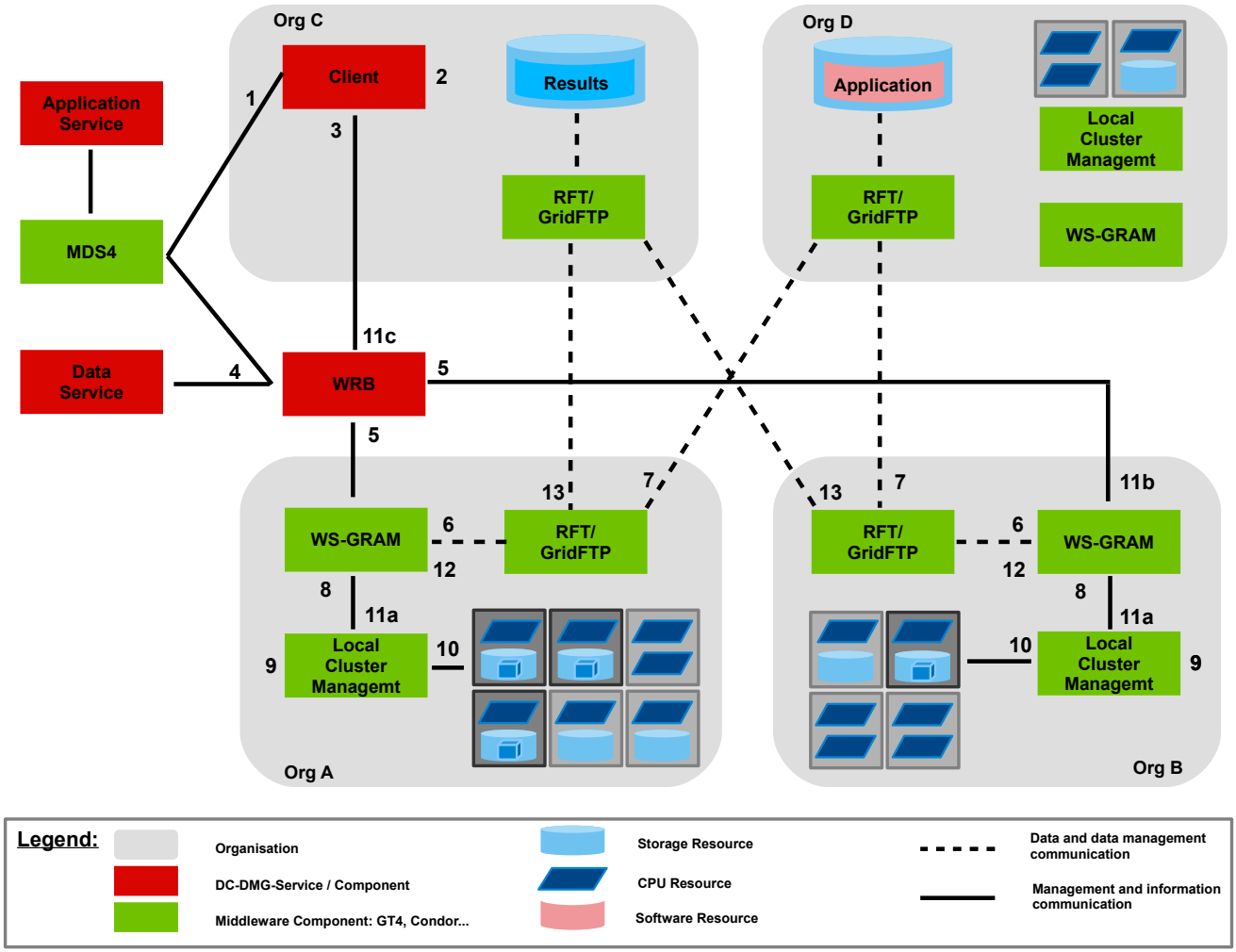


Figure 7. Interaction of the different components while running a Divide&Conquer job.

worldwide. In order to gain the most experience for future fuel cell vehicle development, a fleet data acquisition system has been developed which continuously records all relevant parameters of vehicle operation, such as the fuel cell voltage, current and temperatures. The enormous amount of world-wide distributed data produced by the fleet - over 4 million kilometers have been recorded - and the needs for compute-intensive data analysis methods were the key drivers behind the development of the DMG-DC system [22].

The FDA-Miner provides a user friendly web-based data analysis application for mining the fuel cell data. In addition to specialized visualization and reporting features, it offers a flexible front end to configure customized data mining tasks. The application uses the services of the DMG-DC to retrieve information about the available data and analysis programs. For each user defined task, the application creates the appropriate ADS instances and workflow definitions and submits it to the WRB.

The FDA-Miner programming toolbox supports users implementing specialized DCG data mining algorithms. The toolbox provides Perl and C modules to read the fuel cell data sets and templates for parallel data processing and combination steps.

A common usage scenario of the FDA-Miner is the generation and testing of models for different key components of the fuel cell system. First the user defines a model, e.g., an Artificial Neural Network, to analyze or simulate a specific component then the data to train the model is selected. In most cases only data points with special properties are of interest. In a first data-intensive processing step these data points have to be filtered out of all available data by a DCG job. The resulting training data is relatively small and is transferred to a fast machine for the compute-intensive model training. In the next step the model is applied to a subset or the complete data set to evaluate the model.
As model definition - model training - model evaluation is

an iterative process and new data is recorded constantly, this process is repeated frequently. With its ability to efficiently execute data- and compute-intensive programs the FDA-Miner helps reducing the overall processing time of such applications and therefore enables shorter development cycles of fuel cell vehicle components.

VII. Evaluation

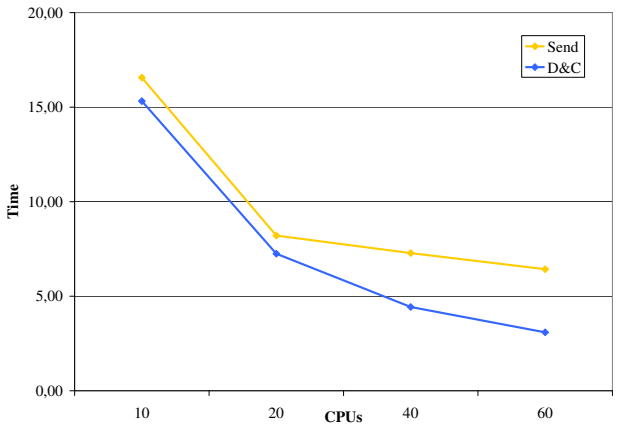
The FDA-Miner has been already heavily used in production and successfully computed thousands of data- and compute-intensive jobs. The following evaluation therefore focuses on the advantages of the DCG functionality of the DMG-DC system compared to traditional Grid systems, like the DataMiningGrid, with dedicated storage servers. The evaluation setup consisted of 9 dual quad core machines with direct attached storage connected over a 1 GBit Ethernet network. To measure the performance of the DCG functionality, a subset of the fuel cell data was randomly distributed over all 9 machines and each file was placed on at least two machines. Traditional Grid systems were represented by a representative scenario with 1 storage server serving the data to 8 compute nodes.

A typical FDA-Miner data-intensive analysis job - filtering the data and computing various statistical properties - was executed on both setups. Figure 8 shows the overall time for performing this job while varying the number of CPUs and the size of the data set. The results demonstrate that the DMG-DC (blue curve), like MapReduce systems, scales well for data-intensive analysis jobs. When the number of CPUs is increased more machines and their storage are utilized, leading to a higher overall data throughput. As no input data is transferred, there is no transfer overhead and the CPUs can leverage the complete storage speed of the machine.

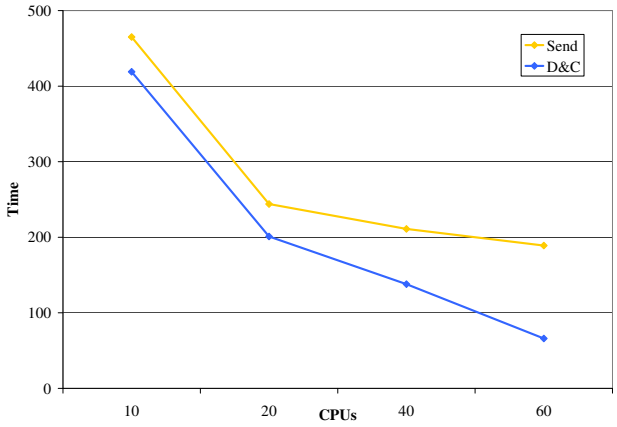
Traditional Grid systems (orange curve) on the other hand have to send the data to the compute nodes first. This not only introduces an additional overhead but also limits the processing performance to the storage speed of the file server. Depending on the network bandwidth and the storage speed the transfer time may, for simple data filtering operations, even exceed the time for data processing. Scaling of these systems is also limited by the number of concurrent connections the storage server can handle without dropping network throughput. In the presented evaluation setup a single storage server can only deliver enough data to server about 20 CPUs and therefore does not scale above that point. Adding additional dedicated storage servers is the only way to increase the system performance. Then distributed file systems like Lustre[23] have to be used if a single file system is required.

VIII. Related work

Recently, various systems and approaches to grid-based data mining and MapReduce have been reported in the literature. Some of those, that are particularly relevant to the DMG-DC system, are briefly reviewed here.



(a) 11694 data sets



(b) 256530 data sets

Figure 8. Execution time of a job in Send and Divide&Conquer mode.

The GridBus resource broker [24] provides functions for scheduling data- and compute-intensive applications. In combination with the Storage Resource Broker[25] GridBus is able schedule data-intensive jobs based on various different metrics, including network bandwidth and utilization. As GridBus follows the common separation between storage and compute resources, data has to be transferred prior to execution and therefore it does not provide DCG or similar functionality at the moment. In addition GridBus is not compatible with current grid standards, in particular WSRF.

The Cactus [26] broker was developed to support compute-intensive numerical calculations in Grid environments. It is based on MPICH-G and the Globus Toolkit and requires that applications have to be written in MPI. This restriction makes it almost impossible to integrate existing data mining applications. In addition Cactus is not WSRF-compliant and does not provide any notion of data aware

scheduling to support data-intensive DCG applications.

The GridWay Meta-Scheduler [27] is the built in resource broker of the Globus Toolkit. GridWay provides common resource brokering functions including data aware scheduling. Storage and compute resources are treated separately so that data has to be transferred prior to execution and there is no support for DCG processing.

Data Mining Grid Architecture (DMGA) [28] focuses on the main phases of a data mining process: pre-processing, data analysis and post-processing. The proposed architecture is composed of generic data Grid and specific data mining services. WekaG is an implementation of DMGA based on the data mining toolkit Weka and the Globus Toolkit 4. The DMGA itself is a flexible architecture to build grid-based data mining system. But its only implementation WekaG is restricted to Weka. The service based approach offers high flexibility but implies resource utilization issues as a service can only use the local resources and WekaG provides no DCG or similar functionality at the moment.

Anteater [2] is a web-service-based system to handle large data sets and high computational loads. Anteater applications have to be implemented in a filter-stream structure. This processing concept and its capability to distribute fine-grained parallel task make it a highly scalable system. Due to the restriction on a filter-stream structure Anteater shares some downsides of MapReduce frameworks: Applications have to be ported to this platform which makes it almost impossible to integrate existing applications.

GridMiner [9] is designed to support data mining and online-analytical processing in distributed computing environments. GridMiner implements a number of common data mining algorithms, some as parallel versions, and supports various text mining tasks. Two major differences between GridMiner and DMG-DC are the DCG functionality and that the latter complies with the recent trend towards WSRF.

Knowledge Grid (K-Grid) [8] is a service-oriented system providing grid-based data mining tools and services. The K-Grid system can be used for a wide range of data mining and related tasks such as data management and knowledge representation. The system architecture is organized into a high-level K-Grid services and a core-level K-Grid service layer, which are built on top of a basic Grid services layer. K-Grid incorporates some interesting features for distributed data mining but no DCG or similar functionality is available at the moment.

Hadoop [6] is the most well known open source implementation of Google's MapReduce paradigm. Hadoop's MapReduce framework is build on top of the Hadoop distributed file system (HDFS) containing all data to be mined. The map and reduce functions are typically written in Java, but also executables can be integrated via a streaming mechanism. MapReduce frameworks like Hadoop do not offer the functionality to efficiently execute compute-intensive applications on a cluster, making them unsuitable for a general-purpose data mining system. Hadoop On Demand in combination with the SUN Grid Engine try to overcome these limitations by running Hadoop on top of a cluster management system, thus adding another layer of complexity. Still, the resources to use for MapReduce are reserved exclusively for Hadoop and can not be used by other jobs. Hadoop and similar MapReduce frameworks simplify the development and deployment of data-intensive applications on local clusters and cloud resources but, in contrast to the DMG-DC system, these frameworks are currently not suited for large-scale, heterogeneous environments comprised of multiple independent organizations.

IX. Conclusion

In this article, we introduced the DCG, a divide&conquer approach for data-intensive applications in Grid environments. The new concept of combined Grid resources in combination with the developed data location aware scheduling algorithm provides an infrastructure to build scalable data-intensive applications in worldwide, heterogeneous environments. The scheduling algorithm and the implemented Resource Broker also support compute-intensive applications so that both data- and compute-intensive applications can be implemented in one single system. The developed DMG-DC system not only provides the functionality to run diverse data- and compute-intensive data mining applications but also supports the complete data mining process end-to-end.

The FDA-Miner, a real world data analysis application, uses the distinct features of the DMG-DC system to efficiently mine the data of the whole Daimler fuel cell vehicle fleet. The FDA-Miner evaluation results highlight the advantages of the DMG-DC compared to traditional Grid systems.

Future work may include the integration of other data management systems like the Globus Toolkit Data Replication Service[29] or the Storage Resource Broker, adding support for other local cluster management system than Condor and a generalized version of the FDA-Miner programming toolbox.

References

[1] M. Röhm, M. Grabert, and F. Schweiggert, "A generalized mapreduce approach for efficient mining of large data sets in the grid," in *1. International Conference on Cloud Computing, GRIDs, and Virtualization, CLOUD COMPUTING 2010*, Lisbon, Portugal, 2010, pp. 14–19.

[2] D. Guedes, W. Meira, and R. Ferreira, "Anteater: A service-oriented architecture for high-performance data mining," *IEEE Internet Computing*, vol. 10, no. 4, pp. 36–43, 2006.

[3] S. Datta, K. Bhaduri, C. Giannella, and H. Kargupta, "Distributed data mining in peer-to-peer networks," *IEEE Internet Computing*, vol. 10, no. 4, pp. 18–26, 2006.

[4] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," 2004, pp. 137–150. [Online]. Available: http://www.usenix.org/events/osdi04/tech/dean.html

[5] S. Ghemawat, H. Gobioff, and S. T. Leung, "The google file system," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 29–43, 2003.

[6] T. White, *Hadoop: The Definitive Guide*, 1st ed. O'Reilly Media, 2009.

[7] V. Stankovski, M. Swain, V. Kravtsov, T. Niessen, D. Wegener, M. Röhm, J. Trnkoczy, M. May, J. Franke, A. Schuster, and W. Dubitzky, "Digging deep into the data mine with datamininggrid," *IEEE Internet Computing*, vol. 12, no. 6, pp. 69–76, 2008.

[8] A. Congiusta, D. Talia, and P. Trunfio, "Distributed data mining services leveraging wsrf," *Future Generation Computer Systems*, vol. 23, no. 1, pp. 34–41, 2007.

[9] B. Peter and W. Alexander, "Grid-aware approach to data statistics, data understanding and data preprocessing," *International Journal of High performance Computing and Networking*, vol. 1, no. 6, pp. 15–24, 2009.

[10] R. McClatchey, A. Anjum, H. Stockinger, A. Ali, I. Willers, and M. Thomas, "Data intensive and network aware (diana) grid scheduling," *Journal of Grid Computing*, vol. 5, pp. 43–64, 2007.

[11] S. Venugopal and R. Buyya, "A set coverage-based mapping heuristic for scheduling distributed data-intensive applications on global grids," in *In proceedings of the 7th IEEE/ACM International Conference on Grid Computing(Grid06*. IEEE CS press, 2006.

[12] K. Ranganathan and I. Foster, "Decoupling computation and data scheduling in distributed data-intensive applications," in *HPDC '02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*. IEEE Computer Society, 2002, pp. 352–358.

[13] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, "The physiology of the grid: An open grid services architecture for distributed systems integration," *Global Grid Forum*, 2002. [Online]. Available: http://www.globus.org/alliance/publications/papers/ogsa.pdf

[14] K. Czajkowski, D. F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe, "The ws-resource framework," 2005. [Online]. Available: http://www.globus.org/wsrf/specs/ws-wsrf.pdf

[15] A. Mario, K. Amy, P. N. W., E. Andrew, L. Simon, M. Susan, M. Jim, and P. Dave, "The ws-dai family of specifications for web service data access and integration," *SIGMOD Rec.*, vol. 35, no. 1, pp. 48–55, 2006.

[16] I. Taylor, M. Shields, I. Wang, and A. Harrison, "The triana workflow environment: Architecture and applications," in *Workflows for e-Science*, I. Taylor, E. Deelman, D. Gannon, and M. Shields, Eds. Secaucus, NJ, USA: Springer, New York, 2007, pp. 320–339.

[17] V. Kravtsov, T. Niessen, V. Stankovski, and A. Schuster, "Service-based resource brokering for grid-based data mining," in *Proceedings of The 2006 International Conference on Grid Computing and Applications*, Las-Vegas, USA, 2006.

[18] I. T. Foster, "Globus toolkit version 4: Software for service-oriented systems." in *NPC*, ser. Lecture Notes in Computer Science, H. Jin, D. A. Reed, and W. Jiang, Eds., vol. 3779. Springer, 2005, pp. 2–13.

[19] M. Ripeanu and I. Foster, "A decentralized, adaptive replica location mechanism," in *HPDC '02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*. Washington, DC, USA: IEEE Computer Society, 2002, p. 24.

[20] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-g: A computation management agent for multi-institutional grids," *Cluster Computing*, vol. 5, no. 3, pp. 237–246, July 2002.

[21] J. Friedrich, R. Schamm, C. Nitsche, J. Keller, B. Rehfus, T. Frisch, and M. Röhm, "Advanced on-/offboard diagnostics for a fuel cell vehicle fleet," in *Society of Automotive Engineers SAE World Congress 2008*, 2008.

[22] M. Röhm, J. Keller, and T. Hrycej, "Data mining fuel cell fleet data for stack degradation analysis," in *Fuel Cell Seminar, San Antonio*, 2007.

[23] S. C. Simms, G. G. Pike, and D. Balog, "Wide area filesystem performance using lustre on the teragrid," in *in Proceedings of the TeraGrid 2007 Conference*, 2007.

[24] S. Venugopal, R. Buyya, and L. Winton, "A grid service broker for scheduling e-science applications on global data grids," *Concurrency and Computation: Practice and Experience*, vol. 18, pp. 685–699, 2006.

[25] A. Rajasekar, M. Wan, and R. Moore, "Mysrb & srb: Components of a data grid," in *HPDC*, 2002, pp. 301–310.

[26] G. Allen, W. Benger, T. Dramlitsch, T. Goodale, H.-C. Hege, G. Lanfermann, A. Merzky, T. Radke, and E. Seidel, "Cactus grid computing: Review of current development," in *Euro-Par 2001 Parallel Processing*, ser. Lecture Notes in Computer Science, R. Sakellariou, J. Gurd, L. Freeman, and J. Keane, Eds. Springer Berlin / Heidelberg, 2001, vol. 2150, pp. 817–824.

[27] E. Huedo, R. S. Montero, and I. M. Llorente, "A framework for adaptive execution in grids," *Softw. Pract. Exper.*, vol. 34, no. 7, pp. 631–651, 2004.

[28] M. Perz, A. Sanchez, V. Robles, and P. Herrero, "Design and implementation of a data mining gridaware architecture," *Future Generation Computer Systems*, vol. 23, no. 1, pp. 42–47, 2007.

[29] A. Chervenak, R. Schuler, and C. Kesselman, "Wide area data replication for scientific collaborations," in *In In Proceedings of the 6th International Workshop on Grid Computing*, 2005.