Graywulf System Documentation

László Dobos

# Introduction

Graywulf is a framework and API to implement distributed database systems over a cluster of Microsoft SQL Servers. It aims to support many distributed database scenarios: system registry, database allocation, query partitioning, distributed joins, scheduling etc.

Graywulf is written in C#, uses SQL Server for the registry and data operations but can read remote data sources, such as data files, MySQL, Postgres etc.

Write about the users prospective

Not just build a cluster, but also publish data

# Graywulf concepts

Graywulf was designed to be a platform for distributed database applications with the following goals.

* Simplify database cluster management via a simple web-based management console.
* Provide an API to build data loading pipelines.
* Provide an API to implement data management jobs.
* Provide (primarily read only) access to a federation of databases co-located on a high performance database server cluster via simple queries.
* Provide a simple, yet powerful web-based user interface that allows users to build there scientific data processing workflows from raw data analysis to press quality plots.

**Cluster:** A collection of machines that are managed together and work together in a controlled way. Clusters usually consist of a head node, or controller, that stores the system registry and runs the job scheduler, as well as worker nodes which run SQL Server instances containing the data. Worker nodes might also run delegated jobs, such as data import and export tasks.

**Federation:** A collection of databases that belong to the same application, field of research, etc.

**Slice:** Databases that are two large to fit on a single server might be sliced into multiple smaller databases. Slicing most of the time happens along the primary key of the largest fact table, carefully observing foreign key constraints. While Graywulf has had the concept of slices since the beginning, sliced databases are not implemented yet.

**Dataset:** In standard TSQL, tables and function are identified by their three part names: database.schema.objectname. In case of linked servers, one can use four part names to identify tables. In Graywulf the syntax is slightly extended to add a so called dataset name in the following form: dataset:schema.objectname. Datasets are distinguished from databases for a series of reasons. First of all, a dataset is most of a time a collection of databases (a set of mirrors, a set of slices, etc.) and not a single database. In a typical federation, different catalogs of data are distinguished by their dataset name.

**MyDB:** Downloading query results after each execution is inefficient for many reasons. list problems To avoid all these problems, users get a scratch database called MyDB for their own purposes at registration where all query results are written by default. Users then can either further process these tables, or download them in various formats.

**Jobs:** All long-running operations are implemented in Graywulf as jobs. Jobs are stored in a queue until execution, when they are picked up by a scheduler that executes them. The job system is designed to be highly extensible and federations can easily implement their own jobs or inherit from the generic Graywulf jobs. System maintenance tasks are also jobs, often recurring ones.

**Partitioned query:** If a database is mirrored to multiple servers, a single query can be partitioned into smaller pieces that run in parallel on multiple machines and results are gathered upon completion. Graywulf implements a very simple model of query partitioning. When a partitioning clause is added to the table that comes first after the FROM keyword in a SELECT query, statistics about the partitioning key are gathered, partition bounds are determined and finally the query is executed in parallel on all machines where the necessary data is available.

**Sliced query:** If a database is slices, i.e. it spans multiple servers due to limited space on individual servers queries must be executed on all slices and results must be gathered upon completion. This query type is not yet implemented.

**Distributed join:** Users may write queries that can reference databases throughout the cluster, even databases that are not available on the same server. Certain queries can reference MyDB tables and even tables on remote data sources (servers not managed by Graywulf). When query references tables from a single database only, it can be always executed on a single server but once we allow multiple databases, it is likely to happen that the required data are not available co-located. Joins between tables residing on separate servers are called distributed joins. Graywulf already supports distributes queries, but the implementation is very simple. A database server is chosen with the most data readily available and all other tables are copied to that particular machine. Simple logic is used to determine a subset of columns and rows of remote tables that are absolutely necessary to execute a join.

**Vertically partitioned table:** Wide tables usually take up too much space to be mirrored to multiple servers. Also, most columns are only accessed by a few percent of the queries while most queries use a few percent of the columns. It might be worth to split the table into to and store the less frequently used columns on a separate, slower sever with larger storage. Vertically partitioned tables could be even implemented transparently, not requiring the users to explicitly do a join between the two table parts. Vertically partitioned tables are not yet implemented in Graywulf. This concept is frequently called lazy join too.

**Metadata:** While database columns, function parameters, variables have only a name and a data type in SQL, most physical quantities in science carry additional information about a piece of data than a value and a name. Graywulf extends the database schema with metadata that can store a physical unit, a human readable description and a software readable content identifier for each column and parameter.

# Basic operations by users

The main objective of Graywulf is to provide a development platform for distributed and federated databases. Users can access database federations by writing queries. Graywulf provides a web-based user interface to interact with the system consisting of a schema browser, a query editor, a set of tools to manage myDB, import and export features, a plotting tool and data sharing tools.

## Schema Browser

Queries can only be written in the knowledge of the underlying schema, thus an efficient schema browser is essential. As Graywulf is designed for scientific applications, detailed descriptions on the contents of the tables and columns are also necessary. The Graywulf schema browser is based on the same schema library classes that are used internally for query processing, and can benefit from schema caching, etc.

## Query editor

The Graywulf query editor is based on an open source javascript code editor component that supports syntax highlighting. Future plans include adding code auto-completion functionality.

## MyDB

Query results and uploaded data appear in MyDB. Users have the ability to drop and rename MyDB tables. Future plans include complete schema management tools where database objects could be freely edited, indices created, metadata added, tables duplicated, etc.

## Export and import data

Data tables can be directly imported into MyDB tables. The web user interface provides a form for uploading data files. File types can be automatically detected form file extensions, or set manually. The data import framework can automatically detect data columns (in case of formats without schema description), add identity columns and append metadata. As files are uploaded via the browser, import operations are done synchronously and as coordinated by the web server. Future plans include allowing uploading zip archives that may contain multiple data files. Instead of uploading files via the browser, fetching files from HTTP, FTP, etc. will be available. The currently supported data format is CSV but the format framework is designed to be extensible.

Data from MyDB can be exported into various formats. Data exports are scheduled as jobs and coordinated by the controller machine. Data export operations are executed by the worker nodes of the cluster and not the web server.

## Plot tables

Future plans include adding a plotting tool to the user interface that will enable users to create print quality plots. Either a tool based on gnuplot or python’s matplotlib will be implemented.

## Share data

There are no means of sharing data tables among users in the current versions.

# Basic operations by administrators

Setting up Graywulf on a cluster of servers is not a trivial task, but does not require software developer skills. Most basic setup steps can be done with command-line tools and once the admin web site is running, it can be used to manage the cluster.

## Creating a cluster

Graywulf

Creating a federation

Creating a database definition

Creating physical databases

Loading data

Publishing data

# An overview of Graywulf modules

Graywulf consist of the following modules.

**Registry:** Stores system information about the status of the cluster in a central SQL database. Hardware configuration is kept down to the RAID volume level to support optimized allocation of databases. Every physical database is reflected in the registry down to the file level to support database mirroring, moving etc. The registry also contains a complete user database.

**Schema:** This module is a collection of classes to reflect the schema of SQL databases with additional (custom implemented) metadata. The schema manager class supports caching of schemas to minimize schema read request toward the database servers. Schema connectors are implemented for SqlServer, MySQL and Postgre, but the framework is extensible. The schema browser web interface is built on top of the schema module.

**SQL parser:** The system consists of a SQL language parser automatically generated from a grammar at compile time. The parser builds a parsing tree from the tokens of a SQL statement that can be later used to analyze the query. Many parser tree node types are extended manually to support the analysis of the query. The SQL name resolver identifies every reference to tables, columns and function calls in the parsing tree generated by the SQL parser, and associates them with the underlying database schema. The name resolver can identify unbound names (identifiers that do not exist in the database) and collect the set of tables and columns that are necessary to execute a query.

**Format:** The format module is an API to implement data file format connectors that allow reading files as if they were data tables, and serialize data tables into various data file formats.

**IO:** The IO module implements data import and export operations. It is based on the Format module but extends its functionality. It can do resilient file copies between servers, but also run bulk inserts and bulk copies from remote servers.

**Activities:** Graywulf implements a set of activities that function as building blocks of workflows that in turn implement jobs that can, for example, execute complex, long running queries, or data export operations, etc. An activity, for example, can export a data table into a file or execute a query on a remote server and store the results in a local cache table.

**Scheduler:** Long-running operations in a distributed database environment are implemented as jobs. In Graywulf, every job is a workflow (built on top of .Net Workflow Foundation) of multiple steps that may contain asynchronous operations and parallel loops. The Graywulf Scheduler can queue and execute such jobs reliably.

**Remote Service:** Task delegation is essential in multi-machine environments. The Graywulf Remote Service module is a lightweight implementation that allows activating and executing user code on any machine of a cluster from the scheduler.

**Logging:** All operations are logged into a central SQL Server database. Logging can be done at such granularity that log record can be used for debugging.

**Web interfaces:** Graywulf provides a web interface for system administration, a user interface and a .Net single sign on interface. The admin web site allows managing the system registry directly. The user interface allows users to browse the database schemas, write and execute queries, track query execution, get information about their MyDBs, import and export data and access help. The authentication service is a central web site for user registration and authentication that can be shared among .Net web sites and services running under the same domain name.

# Graywulf Registry

The Graywulf registry is a complete description of the logical and physical configuration of the database cluster. The registry is a hierarchically organized collection of entities. Entities are also organized into for main groups: Cluster, Federation, Layout, Jobs and Security.

## Cluster group

The cluster group contains entities that reflect hardware components. A cluster consists of servers organized into server roles. Each server can have multiple disk volumes and database server instances. The cluster group contains the following entities.

**Cluster:** The root entity of the registry hierarchy is a single cluster. A single Graywulf installation might control multiple clusters, but each cluster is administered separately, so in the administration web site, only a single cluster will appear and no new cluster can be created. A cluster is a collection of database server machines, databases allocated, as well as application running on them.

**Machine Role:** Machine role is a collection of machines that serve the same purpose in a server cluster. By default, the ‘Controller’ role is used to store the Graywulf registry and execute the job scheduler, whereas the ‘Node’ role is created for database server worked nodes that execute queries, data import and export operations, etc.

**Machine:** A single physical machine of the cluster.

**Disk Volume:** Represents a physical disk volume, usually a big RAID volume, that can be used to store databases. Graywulf can use this information to optimize database allocations for the hardware. To support data movement, data volumes must be shared on the local network and share UNC path is stored for each disk volume.

**Server Version:** Defined on the machine role level and identifies SQL Server versions. Graywulf supports multiple SQL Server versions running side by side, and each server instance is associated with a server version.

**Server Instance:** Defined on the machine level and represents a SQL Server instance running on that particular machine.

## Federation group

The Federation group contains entities that describe the logical configuration of the system. A single cluster can serve multiple purposes, for example, used for two different fields of research. Therefore, database federations and services are combined into domains. Federations are collections of database definitions, which are essentially collections of databases with the same schema, but possibly different data. The Federation group contains the following entities.

**Cluster:** Represents the entire cluster. It is the root of the entire registry and is the same node as in the Cluster node, see above.

**Domain:** A domain a collection of database federations (applications) belonging to the same field of interest. Federations belonging to the same cluster can share certain settings, for example users can register into a domain, instead of individual federations, and share their identity across the services provided within a domain.

**Federation:** A federation is a loose collection of databases definitions that can be accessed in parallel to combine the information contained in them. A federation of databases is usually access by one or more applications are services.

**Database Definition:** A database definition is an abstract representation of a set of physical databases with *identical schema*. A database definition can refer to a set of mirrored databases, or various versions of the same database. A database definition can also be a set of sliced databases, databases that are two large to be stored on a single server and have to be slices (partitioned) and distributed over multiple nodes. It is very important that all databases within a database definition must have exactly the same schema. If the schemas differ, multiple database definitions have to be created. Database definitions can be created under a federation, or directly under the cluster. The latter ones are databases that are shared by all application of the cluster, for example, temporary storage.

**Database Version:** Database versions can be defined to distinguish different variants of the same database. It is not intended to do actual database versioning, but to differentiate between different states of operation. For example, during a data loading process, a ‘hot’ database can serve queries while a ‘warm’ database replicates the same data for redundancy and a ‘cold’ version is used for merging in new data. Once loading is completed, database versions are rotated. Another use case of database versions is when a ‘hot’ version serves queries while a ‘mini’ version of the same database contains a subset of the data only and is used to gather statistics before query execution.

**File Group Definition:** Each database definition has multiple file groups. Data tables will be associated with these file groups when the databases are allocated. Storage requirements for each file group have to be set manually and Graywulf later can use this information to generate database files automatically across multiple disk volumes.

**Slice:** Large databases that do not fit on a single server can be sliced up (partitioned) to span multiple servers in the future. This functionality is not yet implemented. A database definition always has at least one slice called ‘FULL’, meaning that databases are monolithic.

**Partition:** SQL Server supports partitioning of data tables. In Graywulf, partitions for each slice can be defined and the partitioning functions for each physical database are generated automatically. Not fully implemented, and not used with monolithic databases.

**Deployment Package:** Not yet implemented.

**Remote Database:** Remote databases are databases that are not controlled by the Graywulf system but can be referenced in queries. Data on remote servers will be cached before query execution. Graywulf implements some basic logic to pre-filter data on remote servers and fetch only those parts of remote tables that are absolutely necessary to execute a query. Currently MSSQL, MySQL and PostgreSQL are the supported system.

## Layout group

Federations consist of database definitions. Database definition are only abstract representations of physical databases as they can represent sets of actual databases, for example, a set of mirrored databases, databases with different versions of data or databases that are sliced (partitioned) over a collection of servers. The layout group, consequently, consists of the description of the mapping of the logical system configuration (Federation group) to the actual physical configuration of the system (databases existing on the cluster nodes). The layout group contains the following entities.

**Cluster:** Same as described above.

**Domain:** Same as described above.

**Federation:** Same as described above.

**Database Definition:** Same as described above.

**Database Instance:** While database definitions are only abstract representations of a collection of databases with identical schema, database instances refer to actual databases residing on the cluster nodes. A database instance is always an ascendant of a database definition in the configuration hierarchy, but also tagged with a database version and associated with a slice.

**File Group Instance:** File group of a physical database, associated with a database instance and partition. Partitioned databases are not yet fully implemented.

**File Instance:** Refers to a physical file on one of the cluster servers. This information is used when databases are allocated, moved or mirrored. Database files are associated with the disk volumes in the cluster configuration, consequently their local and UNC path can always be figured out easily.

## Jobs group

The jobs group contains the entities describing job queues, job types and actual jobs. Queues can be defined on the cluster or federation level and associated with machines. Certain jobs are defined on the cluster level, such as regular maintenance jobs, but most jobs are defined by the federations, such as different types of query jobs.

**Queue Definition:** write this

**Queue Instance:** write this

**Job Definition:** write this

**Job Instance:** write this

## Security group

**Security group:** The security group contains entities that identify users and user groups. Users and groups can be defined either on the cluster level (this option is for administration purposes) or on the domain level. As a consequence, users registered to a domain may share their identity among the different federations (application) in the same domain.

**User Group:** write this

**User:** write this

Activities

# Schema and metadata

Graywulf features a metadata library that can almost fully reflect the schema of SQL Server databases. The schema browser was written from scratch and does not rely on SQL Server Management Objects (SMO) for many reasons. First of all, the schema library supports additional schema providers to other platforms, such as MySql or Postgres. The schema library supports caching of the schema on the client side which reduces network traffic significantly when the schema is to be reused by different components of the system. Components using the schema library are the SQL name resolves (see SQL Parser section) and the schema browser.

The schema library extends standard SQL database schema with metadata. Metadata is a human/software readable extension to describe the contents of tables, column, parameters etc. The basic metadata entries are a summary, a detailed description that can contain examples, a content identifier field for software clients and a physical unit of the quantities stored in the variable.

Since the Graywulf system parses every SQL query, it can extract detailed information on how output tables and columns are generated from the source tables and columns. This will allow us in the future to extract detailed provenance information from the queries and set metadata on the generated tables automatically.

The user query interface of Graywulf features a schema browser that can display the schema and metadata of all configured datasets in a unified way.

# SQL Parser

SQL Parser is a library to analyze SQL queries prior to execution. It consists of five main modules: a parser generator with a SQL grammar, a SQL parser that is generated with the former but contains many hand-coded extensions, a name resolver, a basic SQL validator and a set of code generators.

The parser generator is a compile-time tool that generates the parser classes from a grammar. As a specialty, grammars are formulated as C# expressions, which enable checking grammar consistency at compile time. The parser generator supports grammar inheritance, a feature that enables writing extensions to another grammar without having to modify the original. The parser generator was written from scratch to generate parser tree nodes exactly in the form a C# SQL parser requires. The generated parser is a simple backtracking one, rules are implemented in the parsing tree node classes. All generated classes are marked partial, so custom implementation to the generated code can be added very easily. The SQL grammar is freely extensible but currently supports a slightly limited version of the SELECT statement only.

The SQL Parser library contains a name resolver which is a very important part of the system. The name resolver can identify nodes in the parsing tree that reference tables, functions calls, columns etc. and matches them with the underlying database schema. As a result, a query executor can figure out lots of things about the query, for example, get the list of tables and columns that have to cached from remote servers before query execution.

The name resolution process runs recursively on subqueries, starting with the innermost query and traversing the hierarchy outwards. 1) Default dataset names and schema names are inserted into table and table-valued function identifiers. 2) Tables and table-valued functions are resolved after the FROM clause. 3) Columns and function calls in the SELECT list, ON criteria, WHERE, HAVING and GROUP BY, etc. clauses are resolved. The last clause to process is the ORDER BY.

The name resolver currently supports the following.

* Resolve table and table valued function references
* Resolve column references in all clauses
* Support multiple occurrences of the same table in the same query (with different aliases)
* Support subqueries
* Support UNION-type multi-part queries
* Can generate CNF and DNF of logical expression
* Can determine the most restrictive where clause that can be applied to a table before joins

The SQL Parser library contains code generators. Code generators visit the parsing tree and convert the nodes back to SQL text. Different generators can be implemented for various flavors of SQL, currently simple code generators exist for MySQL and Postgres, beside SQL Server. Code generators can also generate special queries to fetch parts of tables from remote servers.

# Data formats

The Graywulf Format API contains classes that help implementing new data formats that can be read and written by the framework.

Internally, each data file is read as a table via an IDataReader implementation. Files are written by an IDataWriter, row by row. The IDataWriter takes data rows from an IDataReader. A single file, if the file format supports it, might contain multiple tables, called blocks.

Additional features supported: compressed files, compressed archives of multiple files (not yet complete).

# IO building blocks

Reliable file copy

Database mirroring

Bulk imports and exports

Read and write local files, FTP, dropbox, etc.

# Activities and jobs

Jobs as workflows

Workflows are built from activities

Async activities with time-out and cancellation logic

Retry logic

Logging from activities

Predefined activities

Dynamic workflows

# Scheduler

The Graywulf Scheduler is responsible of executing long-running jobs. In Graywulf, all jobs are implemented as workflows. Workflows are graphs of sequences of activities that have to executed in a particular order. Activities can be inside loops and certain loops can be executed in parallel. Beside executing jobs, the scheduler is also responsible

On startup, the Graywulf scheduler loads the system configuration from the Registry.

Load system config on startup

Keep track of system status, not yet implemented

Reload system config eventually, not yet implemented

Queue and execute jobs

Job polling

Enforce time-out

Cancel jobs

Handle job exceptions

Pausing and persisting jobs

Jobs are binaries, yet new jobs can be added at runtime

Multiple versions of the same job supported

Conserving memory

# Task delegation via Remote Service

Scheduler cannot execute actual code

Queries are easily delegated to database servers

Other computation is delegated to worker nodes via the Remote Service module

Implementation details

# Logging

What log contains

How to navigate log

Debugging practices

# System setup

How to build, install and configure a Graywulf cluster

## Hardware configuration

## Platform configuration

### Hardware configuration

### Windows configuration

### Database Server configuration

#### Configure tempdb

By default, tempdb files are located on the system disk. To move them to the RAID volume, execute:

ALTER DATABASE tempdb

MODIFY FILE (NAME = tempdev, NEWNAME = tempdev\_0, FILENAME = '[path\_to\_data\_volume\_0]\tempdev\_0')

ALTER DATABASE tempdb

MODIFY FILE (NAME = templog, NEWNAME = templog\_0, FILENAME = '[path\_to\_data\_volume\_0]\templog\_0')

Restart the server for the changes to take effect. It is important to move the files first and set the file size later because files are moved to the new location only after a server restart and growing the files residing ont he system volume may eat up all the disk space and crash the system.

In case of multiple volumes, add more files:

ALTER DATABASE tempdb

ADD FILE (NAME = tempdev\_1, FILENAME = '[path\_to\_data\_volume\_1]\tempdev\_1', SIZE = 50GB, FILEGROWTH = 0)

ALTER DATABASE tempdb

ADD LOG FILE (NAME = templog\_1, FILENAME = '[path\_to\_data\_volume\_1]\templog\_1', SIZE = 10GB, FILEGROWTH = 0)

You can verify the settings by executing

exec sp\_helpfile

It is very important to turn off file growth as a runaway query may easily eat up all the disk space on a server.

### Setting up a Windows account for the services.

In order to centrally manage security of the system, a Windows domain account is required. All Graywulf services will run under or impersonate themselves under this domain account. We suggest to name this account MYDOMAIN\Graywulf. By default, the account should only have basic domain user privileges with no remote desktop access.

The following checklist can be used to configure the permission this account must have:

* Member of domain users
* Full access to all data directories on the worker nodes
* Full access to the network shares of all data directories of the worker nodes
* Windows user account added on all SQL Servers
* Member of dbcreator role on all SQL servers on the worker nodes
* Full control access to %windir%\temp on the webserver

Create graywulf domain account, very important to run everything under this account

* grant access to the shared directories, pay attention to mounted volumes!
* DB creator under sql server
* run service under this account
* grant access to %windir%\temp so serializer classes can be generated on the web server

Setting up the Graywulf Registry

Setting up the log

Setting up the database for workflow persistence

- Create an empty database called Graywulf\_Persistence

- Run SqlWorkflowInstanceStoreSchema.sql and SqlWorkflowInstanceStoreLogic.sql located in %windir%\Microsoft.NET\Framework64\v4.0.30319\SQL\en

Installing the bulk-op server

Open port 5055

Setting up the Graywulf Cluster Administration Console

How to run the admin console with windows credentials

Creating a Federation

Configuring MyDB location

Configuring existing monolithic databases

Setting up the front-end for a federation

# Developing Databases for Graywulf

# Credits

# References

# Copyright