the Availability Digest

GRIDSCALE® – A Virtualized Distributed Database

July 2008

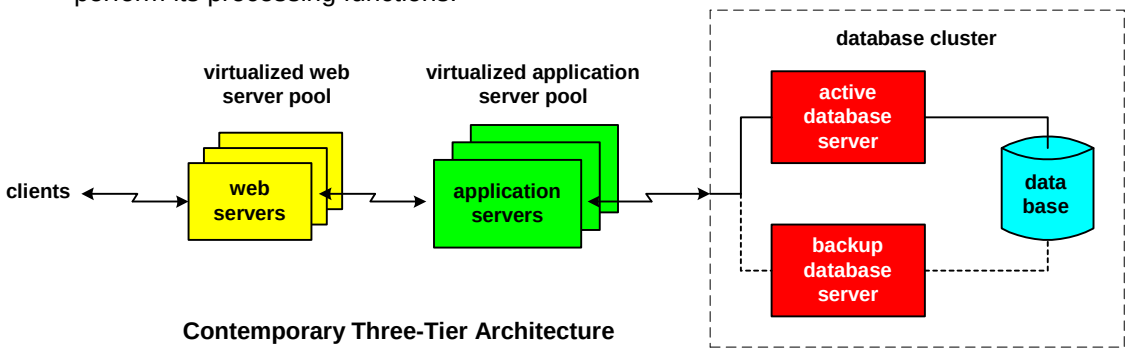
As its name implies, GRIDSCALE, from xkoto, Inc. (www.xkoto.com), is a scalable grid of database servers. GRIDSCALE virtualizes a pool of database servers so that they appear to be a single, consistent database server to the applications they serve. The database servers may be geographically dispersed for disaster tolerance. The failure of any one database server is transparent to the users of the database. This virtual database is readily scalable by adding or subtracting additional database servers on-the-fly.

The Three-Tier Syndrome

Contemporary Three-Tier Architecture

Common practice today is to implement systems in three tiers:

- The *Presentation Tier* manages data received from and sent to the clients of the system. Perhaps the most widespread example of a presentation tier is that comprising web servers.
- The *Application Tier* processes requests from clients received from the presentation tier and returns results to the clients through the presentation tier.
- The *Database Tier* provides the application database needed by the application tier to perform its processing functions.



Contemporary Three-Tier Architecture

In typical systems today, the presentation tier and the application tier are virtualized. That is, they comprise a pool of similar servers, any of which can provide the required processing. Should a server fail, it can be removed from the pool; and the rest of the pool will continue to carry the

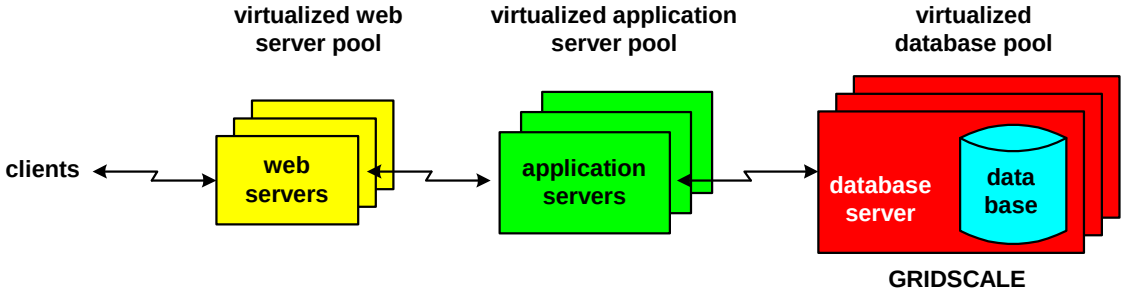
system load with no interruption to the users. If more capacity is needed in a tier, additional servers can be added to help carry the load.

However, the database tier is typically a single system – often a cluster. Although it may be configured as an active/backup system with redundant data storage such as RAID, it is not scalable without installing larger servers since only one server is active at a time. Furthermore, failover is not transparent to the user. Clusters typically take minutes to failover, and that is only if there has been no database corruption caused by the failure of the primary server.

Virtualized Three-Tier Architecture

There are great benefits to be obtained if virtualization could be extended to the database tier. By virtualizing a pool of database servers, the system can be made impervious to database failures. Furthermore, query loads can be spread among the individual database servers, thus allowing scalability by adding additional database servers.

This is the mission of GRIDSCALE.



Virtualized Three-Tier Architcture

GRIDSCALE

GRIDSCALE provides a single-image view of a pool of database servers, each managing its own private database. The database servers may be geographically distributed to provide disaster tolerance. If a database server is lost, it is automatically evicted from the server pool; and transactions are handled by the remaining database servers. GRIDSCALE maintains its own SQL statement log to resynchronize an evicted server upon its return to the pool.

GRIDSCALE ensures database consistency so that all databases are always in the same state. It does this through asynchronous transaction replication. Each operation within a transaction is sent simultaneously to all database servers to be executed. GRIDSCALE guarantees that each server will execute transactions in exactly the same order so that read consistency is maintained across the pool.[1]

GRIDSCALE provides high performance even while managing synchronized copies of the database. Unlike synchronous transaction replication systems, which must wait for all of the database servers to commit the transaction before completion is returned to the application, GRIDSCALE will return completion as soon as the transaction completes on just one database server. Typically, there is at least one local database server so that transaction response time is substantially that of a standalone transaction processing system.

Queries are load-balanced across the database servers. A read operation is directed to the database server that is most likely to deliver the fastest response. A byproduct of this is that for

[1] GRIDSCALE can also be used in an "operation" mode in which the atomic element is a single read or write operation rather than a transaction. The description of this mode is the same as that for the transaction mode, where a single operation can be considered a transaction.

write-seldom, read-often applications, a GRIDSCALE virtual database is nearly linearly scalable by adding additional database servers to spread the read load. In addition, query-only applications may run independently on the database servers.

GRIDSCALE supports views, triggers, materialized views, and stored procedures. It also replicates DDL commands entered by operations personnel to reconfigure the virtualized database.

While GRIDSCALE works with SMP systems, GRIDSCALE also allows the use of heterogeneous commodity servers as the database servers rather than large SMP systems. Since all GRIDSCALE database servers are running in an active/active configuration and are contributing to load handling, two or more small systems can match the capacity of a much more expensive pair of large SMP systems needed for a database cluster. This is because only one of the systems in a cluster can be active at any one time. Currently, GRIDSCALE supports IBM's DB2 database running on IBM AIX, Linux, Microsoft Windows, or Sun Solaris. Support for SQL Server databases is in beta now, and support for other databases will be released in the future.

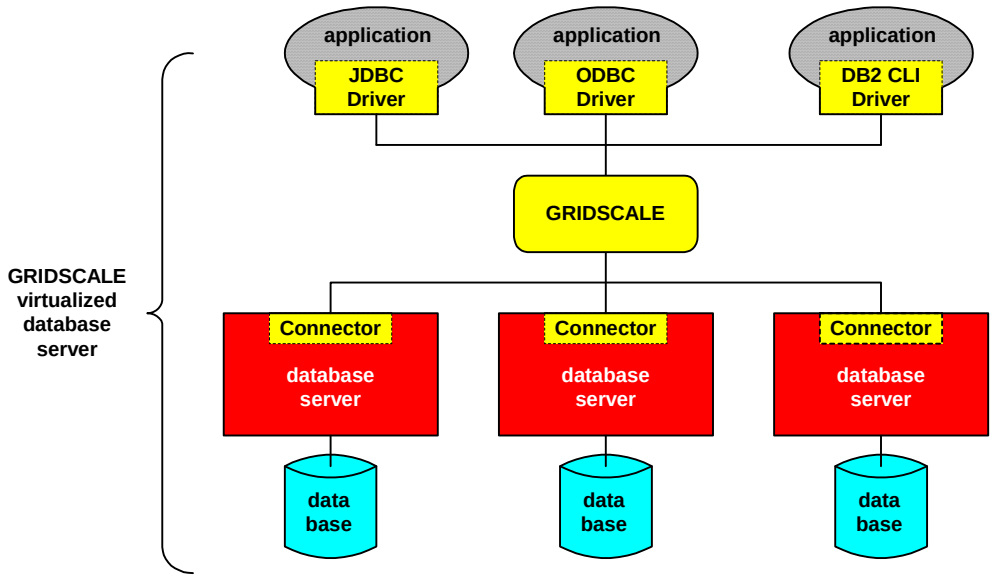
GRIDSCALE is noninvasive. It can be installed for use by any application without modification provided that the application makes database calls via ODBC, JDBC, DB2 CLI, native DB2 SQL calls, or SQL Server with .NET calls.

GRIDSCALE is managed from a single point via a web or command line interface.

GRIDSCALE Architecture

GRIDSCALE comprises three components:

- *Application Drivers* that provide standard interfaces to the GRIDSCALE virtualized database.
- A *GRIDSCALE server* that orders application requests and passes them to the database servers in such a way as to guarantee consistency.
- *Database Connectors* that manage the execution of transaction I/O requests in the order specified by the GRIDSCALE server.



Application Drivers

GRIDSCALE provides drivers for applications to use to access GRIDSCALE servers. These drivers support ODBC, JDBC, and DB2 CLI as well as native DB2 SQL calls. SQL Server .NET calls will be supported. If an application uses one of these interfaces to make its database calls, it can use GRIDSCALE without modification simply by linking in the appropriate driver.

GRIDSCALE Server

The GRIDSCALE Server (GSS) is the heart of the GRIDSCALE system.[2] It receives all database transaction requests from the applications and serializes them for the database servers.

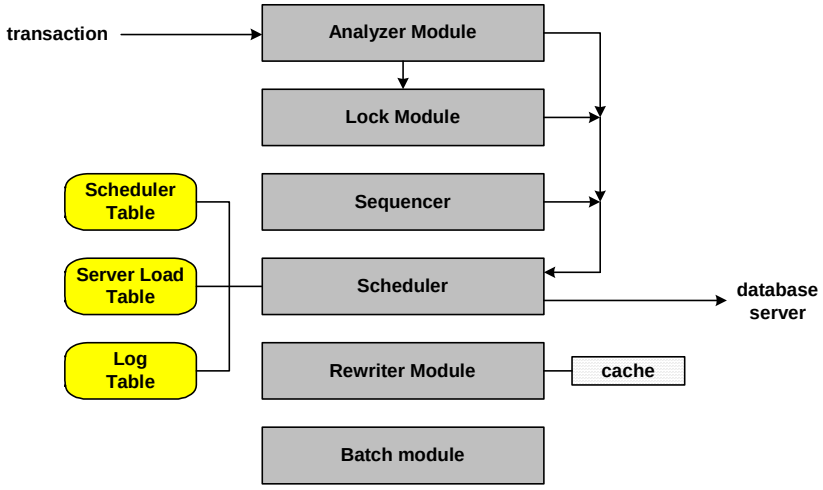
The GSS analyzes each operation within a transaction and determines which are read and write operations. It sends all writes (inserts, updates, deletes) to all database servers.

It analyzes each read request to determine which database server is best able to handle the read request. It may select a database server based on the server's load, on its computing capacity, and on the distance separating the server from the GSS. Having selected a server, the GSS then sends the read request to that server but also sends a virtual read request to all of the other servers so that they may appropriately lock the row being read.

Each read or write request is sent with an appropriate lock and a sequence number. The sequence numbers guarantee that all operations are performed by each database server in exactly the same order.

The GSS typically runs on its own hardware server, where best practices suggest a two-processor, two-gigahertz. server running Linux, AIX, Solaris, or Windows. An active/backup pair of servers may be configured to protect against a GSS failure.

A GSS comprises the following six modules that in concert process a transaction request.



GRIDSCALE Database Load Balancer

Analyzer Module

The Analyzer Module parses the SQL statement and determines which read and write requests are being made. It passes these as elemental operations to the other modules in the GSS.

[2] Ferguson, Gregory; Heisz, Jeffrey; Tung, David; Jamal, Muhammad Mansoor; Kassam, Ariff; Method and System for Load Balancing a Distributed Database, U.S. Patent 20070203910; August 30, 2007.

Lock Module

The Lock Module determines what kind of lock is to be carried with each data operation in order to guarantee consistency of the database. The ANSI SQL isolation levels are supported:

- *Read Uncommitted* (dirty reads) - Unimpeded reading of a row or record is provided.
- *Read Committed* - If a transaction reads a row, it will always get the same data on subsequent reads of that row.
- *Repeatable Read* - Only the results of committed transactions are available to be read.
- *Serializable* - Once a transaction reads a row, no other transaction may modify that row until the original transaction has been committed.

A lock carries four pieces of information: a *Lock Number*, which uniquely identifies the lock and associates it with a particular row or record; a *Lock Type*, which specifies whether the lock is a read or a write lock; a *Lock Scope*, which specifies how long the lock is to be held (for the read/write operation or until the transaction is committed); and a *Lock Sequence*, which specifies the sequential position of its associated operation with respect to all other operations.

Sequencer

The Sequencer receives the results from the Analyzer Module and assigns a sequence number to each of the transaction operations that will access a particular database row or record. Sequence numbers are assigned so that the transaction operations will be ordered across all transactions, guaranteeing that all transactions and all operations within each transaction are executed in exactly the same order in each database server. The sequence number assigned to each database read/write operation is associated with the lock for that operation, as described above.

Scheduler

The Scheduler tracks the execution of the respective transactions and their operations on the database. It receives the operations that make up a transaction from the Analyzer Module, the locks associated with each operation from the Lock Module, and the sequence number associated with each lock from the Sequencer.

The Scheduler determines the distribution of each of the operations among the database servers. Write operations (inserts, updates, deletes) are sent to all database servers along with their associated locks and sequence numbers.

For read operations, the Scheduler determines the database server best capable of handling the read. This determination is based on the current server loads, the power of the servers (if heterogeneous servers are used), and the communication latency (i.e., the distance) of the servers from the GSS. The read request is sent to the preferred database server. However, a virtual form of the read request is sent to all other database servers so that they may apply the appropriate read lock to their copies of the row or record for the duration of the transaction. In this way, the system query load is balanced among all of the database servers in the virtualized pool.

Should a read operation fail due to a server problem or a network problem, the GSS can submit the read request to another database server.

The Scheduler tracks the responses from the various database servers and determines when a response is to be returned to the application. For reads, the result is returned as soon as the Scheduler receives the response from the selected database server.

For writes, completion is returned to the application as soon as one of the database servers has reported a successful write to the Sequencer. Similarly, for transactions, the commit response is returned to the application as soon as one database server has reported a successful commit to the Scheduler. In the event that all servers return an error, an error status is returned to the application.

The Scheduler has three tables that it uses to execute its functions:

- The *Scheduler Table* tracks the results of database operations. An entry is made into the Scheduler Table for each operation received by the Scheduler. Among other information, a table entry contains the operation, the operation's sequence number, and the response to the operation by each of the database servers.

  The response field contains success/fail responses from each database server. It may also include other information such as status, warnings, and parameter return information. It is the Scheduler Table response field that determines when the Scheduler can respond to the application. A read operation is complete when the read data is returned from the selected database server. A write operation or a transaction commit is complete when the first success response is received from a database server.

  Once an operation has been committed or rolled back by all database servers, its entry is deleted from the Scheduler Table.

- The *Server Load Table* tracks the current load on each database server. This load information is returned to the Scheduler typically with each operation and may include the number of operations waiting to be performed at the database server, the processing speed of the server, and the distance separating the server from the GSS. It is the Server Load Table that the Scheduler consults to determine to which database server to send a read request.

- The *Log Table* contains an entry for every committed operation (it is a standard transaction log as found in most transaction managers). The operations associated with a transaction are written to the Log Table before the transaction is committed.

  The Log Table is used to recover a database server that has left the database pool for any reason – planned or unplanned (a crash) – when that server is to be returned to service.

If the Scheduler finds that one database server has failed to complete an operation that was successful at all other database servers, it can evict that server from the database server pool for analysis and repair.

Rewriter Module

The Rewriter Module analyzes the SQL statements contained within a transaction and stores the template represented by that transaction in the module's cache memory. This is the standard SQL *prepare* function. If a later SQL statement that matches this template is received, then only the values in each field need to be sent to the database servers rather than the lengthy SQL text statement.

In addition, the Rewriter Module looks for any fields that are nondeterministic. That is, the value of such a field received from the various database servers may be different. Examples of such fields

are random numbers and time stamps. The Rewriter Module will replace these fields with specific values so that they will be deterministic across the pool of database servers.

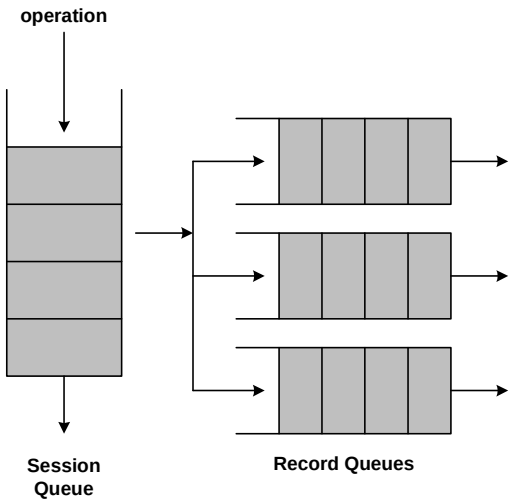
Batch Module

The Batch Module determines whether a set of operations in a single transaction can be sent as a batch to the database servers. An example of a set of operations that can be batched is a sequence of insert operations.

Database Connectors

A Connector accepts sequenced operations from the GSS and manages their executions by the database server. A Connector has available to it a Session Queue for each client session and a Record Queue for each row or record in the database.

A single operation can affect several database rows or records. When an operation is received, it is placed in the appropriate Session Queue along with all of its locks and sequence numbers. Lock and sequence number entries are also entered into the Record Queues for each record that is affected by this operation. The queues are ordered by sequence number.



The operation can be executed when all of its components - the Session Queue entry and all Record Queue entries - are at the head of their respective queues. Upon completion, the completion status and data, if any, are returned to the Scheduler in the GSS.

The Connector can detect potential deadlocks. If it does so, one of the conflicting operations is aborted.

For recovery purposes, the Connector stores the last transaction that it successfully completed.

Networking

All communication between the GSS and its clients and database connectors is over high-speed TCP/IP links using the GRIDSCALE Database Routing Communication Protocol (DRCP). These links may be encrypted.

Continuous Availability

GRIDSCALE provides many high-availability and continuous-availability features.

Active/Active Database Servers

All database servers are running in an active/active pool. Therefore, not only can the query load be distributed among all servers, but the loss of any server is transparent to clients. A server can be evicted from the server pool because its heartbeat is lost, because it does not respond, or because it gives an erroneous result.

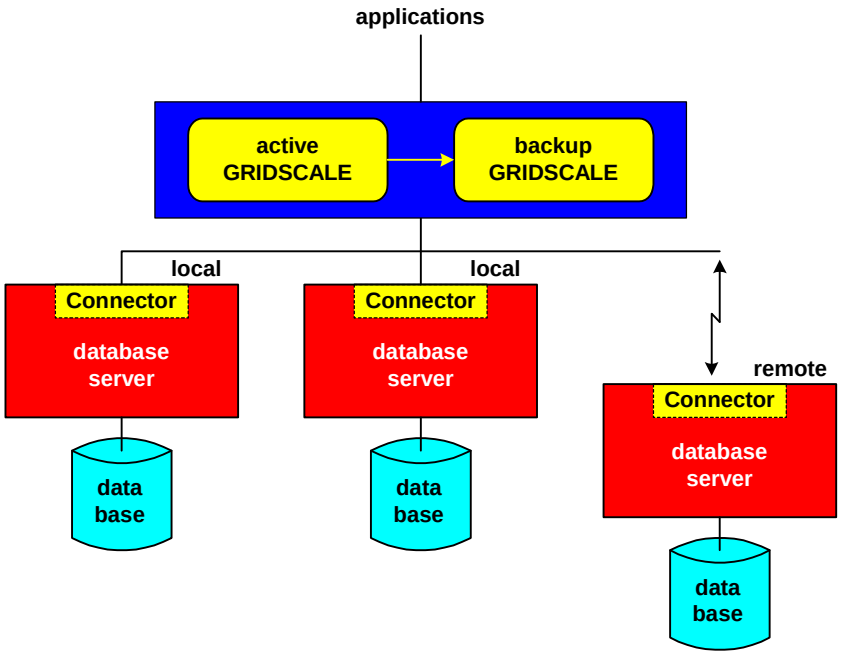
In the event of a server eviction, transaction load is simply redistributed to the remaining database servers.

Database Server Recovery

When a database server that had been removed from the server pool is to be reinstated, the GSS can query it to determine the last transaction that the database server committed. Using this as a marker, the GSS can then replay all subsequent transactions to the recovering database server from the GSS Log Table, thus bringing the database server back into synchronization with the database server pool.

Disaster Tolerance

One or more of the database servers can be located at a data-center site remote from the data center at which the GSS is located. This provides tolerance to a disaster that takes out the local site. In the event of an outage of the local site, the application servers can reconnect to the remote database server and can continue operation.



Eliminating Planned Downtime

Via GRIDSCALE's management console, a database server can be *frozen*. This action removes it from the server pool but maintains the GSS connection to that database server.

The frozen database server may then be upgraded or maintained. When it is ready to be returned to service, it is *unfrozen*. The queue of transactions that had occurred while it was frozen is drained to it from the Log Table, and the database server is returned to service.

In this way, upgrades may be rolled through the database pool one node at a time.

GRIDSCALE Server Crash

The GSS is the heart of the GRIDSCALE environment. Should it be lost, the virtual database would be down. To protect against this, the GSS can be configured as a primary/backup pair. In this case, the memory-resident Scheduler Table and the Load Table are replicated to the backup system upon each update. In that way, the backup GSS is prepared to take over instantly.

Takeover is subsecond once a primary failure is detected. Failover detection typically takes about five seconds (loss of five one-second heartbeats).

To provide continued support for database server recovery, the Log Table is also replicated to the backup system. The completion of a transaction is noted in both the primary and backup Log Tables before the transaction is reported to the application as committed.

Should the site containing the GSS be taken out by a disaster of some sort, operations can continue with a remote data center if one has been provided.

Management

GRIDSCALE provides a web interface and a command-line interface to configure and manage the virtual database environment. Via this interface, database administrators can add and remove database servers, freeze and unfreeze database servers, monitor the performance of database servers, view alerts generated by GRIDSCALE for detected faults or anticipated problems, and generate performance reports.

Administrators can issue DDL commands to reconfigure the database. These commands will be sent synchronously to all database servers for simultaneous execution.

Graphical dashboards are provided to show the relative performance of each database server, server workloads, transaction rates, and other system parameters.

Performance

The GRIDSCALE virtual database is easily scalable by adding additional nodes on-the-fly. One test run by an xkoto customer showed the capability for a four-node system to process 15,000 reads per second.

GRIDSCALE virtualization does add some minor overhead to database processing. However, tests have shown a nearly linear scalability, with each database server after the first adding about 85% of its capacity to the virtual database. This has been demonstrated for systems up to eight nodes and is projected to continue for up to thirty nodes.

xkoto

xkoto is a fairly new venture. It is a privately held company founded in Canada in 2005, and it first delivered product in 2006. It now has installations throughout North America and Europe in the fields of financial services, healthcare, travel, telecommunications, and retail.

Its partners include IBM, Microsoft, VMware, Citrix, Sun Microsystems, and HP. The GRIDSCALE product has been thoroughly tested by IBM for the DB2 database, and IBM currently provides product support for GRIDSCALE.

Where did the name “xkoto” come from? It is reportedly a transliteration of two Greek words loosely translated as “out of darkness, out of chaos.”

Summary

GRIDSCALE brings virtualization for its supported databases to the final tier of today’s popular three-tier architecture. Entire data centers now can be virtualized pools of servers providing their respective functions with active/active load-sharing and continuous availability with no failover. As with the presentation and application tiers, the database tier is readily scalable by adding database servers to the database pools.