


## Table Of Contents


- Overview
- Examples
- Maven
- JDBC Driver
- Client Router
- Hadoop Pig Integration
- Performance Tips
- Scalar
- Scripts
- System Properties


# Overview

This document provides various **miscellaneous** information about GridGain products that augments the main documentation:

- Installation Instructions (that come with every distribution)
- Developer Manuals
- White Papers, Datasheets, APIs

# Examples

This chapter describes structure of all examples shipped with GridGain including examples for Java, Scala, PHP with REST access as well as configuration and other miscellaneous files.

> Use pom.xml to import examples into IDE of your choice

## GRIDGAIN_HOME variable

Don't forget to setup `GRIDGAIN_HOME` environment variable pointing to `distro` folder under GridGain installation root.

## 'config' folder

This folder contains GridGain XML configuration files. Each example always has documentation on what specific configuration (i.e. configuration file) it requires.

## 'libs' folder

This folder contains necessary libraries for running examples.

## 'clients' folder

This folder contains examples for C++, .NET/C#, PHP, and Java clients.

## 'java' folder

This folder contains Java examples for all major GridGain APIs. There are over 50 different examples for different GRidGain subsystems.

## 'scalar' folder

This folder contains `Scalar` examples. Scalar is an advanced Scala-based DSL for GridGain.

This is the simple example of full featured MapReduce app in Scalar (equivalent to Java example above):

```
package org.gridgain.scalar.examples

import org.gridgain.scalar.scalar
import scalar._
import org.gridgain.grid.GridClosureCallMode._

object ScalarWorldShortestMapReduce {
    def main(args: Array[String]) {
        scalar {
            val input = "World shortest mapreduce application"

            println("Non-space characters count: " +
                grid$ @<(SPREAD, for (w <- input.split(" "))
                    yield (() => w.length), (s: Seq[Int]) => s.sum)
            )
        }
    }
}
```

### 'gar' folder

This folder contains example of deploying GridGain compute task using GAR file protocol.

# Maven

GridGain provides repository for its Maven artifacts.

> GridGain repository is located at http://www.gridgainsystems.com:8085/nexus/content/repositories/external

## Maven Artifacts

GridGain Maven repository has 5 artifacts:

- gridgain-datagrid - contains jars and dependencies for In-Memory Data Grid
- gridgain-streaming - contains jars and dependencies for In-Memory Streaming
- gridgain-hadoop1 - contains jars and dependencies for In-Memory Accelerator for Hadoop (Hadoop 1.x version)
- gridgain-hadoop2 - contains jars and dependencies for In-Memory Accelerator for Hadoop (Hadoop 2.x version)
- gridgain-platform - contains jars and dependencies for all GridGain editions.

# JDBC Driver

JDBC Driver allows to get distributed data from GridGain cache using standard SQL queries and standard JDBC API. Use it to connect to your GridGain In-Memory Data Grid using standard JDBC API from your favorite SQL viewing tool.

> Note that GridGain JDBC driver allows to only query cached objects in read-only fashion. All updates, inserts, or deletes must be done directly through GridGain caching API.

## Query Execution

Queries are executed differently depending on the cache mode. For Replicated caches data is available on every node and it is enough to execute the query directly on the node the JDBC driver is connected to.

For Partitioned caches data is partitioned across multiple nodes and therefore GridGain must automatically distribute query. This has effect on how distributed joins will work. If you are

executing a join between Replicated cache and Partitioned cache (same as Star Schema), then joins will always work without additional effort. If you are executing a join between different classes within Partitioned cache, then you need to make sure that entities being joined are properly collocated.

For example, when joining Person with Organization you must make sure that all Persons working for the same Organization are cached on the same node as Organization itself. GridGain has several ways to support such colocation. Please refer to documentation for `org.gridgain.grid.cache.affinity.GridCacheAffinityKeyMapped` annotation and several affinity examples under `org.gridgain.examples.datagrid.affinity` package for more information.

## Limitations

Data in GridGain cache is usually distributed across several nodes, so some queries may not work as expected since the query will be sent to each individual node and results will then be collected and returned as JDBC result set. Keep in mind following limitations (not applied if data is queried from one node only, or data is fully co-located or fully replicated on multiple nodes):

- `Group by` and `sort by` statements are applied separately on each node, so result set will likely be incorrectly grouped or sorted after results from multiple remote nodes are grouped together.
- Aggregation functions like `sum`, `max`, `avg`, etc. are also applied on each node. Therefore you will get several results containing aggregated values, one for each node.
- Joins will work correctly only if joined objects are stored in collocated mode. Refer to `org.gridgain.grid.cache.affinity.GridCacheAffinityKey` javadoc for more details.
- Note that if you are connected to local or replicated cache, all data will be queried only on one node, not depending on what caches participate in the query (some data from partitioned cache can be lost). And visa versa, if you are connected to partitioned cache, data from replicated caches will be duplicated.

## SQL Notice

Driver allows to query data from several caches. Cache that driver is connected to is treated as default schema in this case. Other caches can be referenced by their names.

Note that cache name is case sensitive and you have to always specify it in quotes.

## Dependencies

JDBC driver depends on main GridGain JAR and with all its dependencies. So if you are using JDBC driver in any external tool, you have to add main GridGain JAR will all dependencies located under `GRIDGAIN_HOME/libs` folder.

## Configuration

Internally JDBC driver **is based on GridGain Java client**. Therefore, all client configuration properties can be applied to JDBC connection.

JDBC connection URL has the following pattern: `jdbc:gridgain://<hostname>:<port>/<cache_name>?nodeId=<UUID>`

Note the following:

- Hostname is required.
- If port is not defined, `11211` is used (default for GridGain client).
- Leave `<cache_name>` empty if you are connecting to default cache.
- Provide `nodeId` parameter if you want to specify node where to execute your queries. If not provided, then random node at the specified host will be used. If node with provided ID doesn't exist, exception is thrown. Note that local and replicated caches will be queried on this node only, while partitioned cache will be queried across all participating nodes.

Other properties can be defined in `Properties` object passed to `DriverManager#getConnection(String, Properties)` method:

| Name | Description | Default | Optional |
|---|---|---|---|
| **gg.client.protocol** | Communication protocol (`TCP` or `HTTP`). | `TCP` | Yes |
| **gg.client.connectTimeout** | Socket connection timeout. | `0` (infinite timeout) | Yes |
| **gg.client.tcp.noDelay** | Flag indicating whether `TCP_NODELAY` flag should be enabled for outgoing connections. | `true` | Yes |
| **gg.client.ssl.enabled** | Flag indicating that `SSL` is needed for connection. | `false` | Yes |
| **gg.client.ssl.protocol** | SSL protocol (`SSL` or `TLS`). | `TLS` | Yes |
| **gg.client.ssl.key.algorithm** | Key manager algorithm. | `SunX509` | Yes |
| **gg.client.ssl.keystore.location** | Key store to be used by client to connect with GridGain topology. | | No (if `SSL` is |

| | | | |
|---|---|---|---|
| | | | enabled) |
| **gg.client.ssl.keystore.password** | Key store password. | | Yes |
| **gg.client.ssl.keystore.type** | Key store type. | `jks` | Yes |
| **gg.client.ssl.truststore.location** | Trust store to be used by client to connect with GridGain topology. | | No (if `SSL` is enabled) |
| **gg.client.ssl.truststore.password** | Trust store password. | | Yes |
| **gg.client.ssl.truststore.type** | Trust store type. | `jks` | Yes |
| **credentials** | Client credentials used in authentication process. | | Yes |
| **gg.client.cacheTop** | Flag indicating that topology is cached internally. Cache will be refreshed in the background with interval defined by `gg.client.topology.refresh` property (see below). | `false` | Yes |
| **gg.client.topology.refresh** | Topology cache refresh frequency (ms). | `2000` | Yes |
| **gg.client.idleTimeout** | Maximum amount of time that connection can be idle before it is closed (ms). | `30000` | Yes |

## Example

```
// Register JDBC driver.
Class.forName("org.gridgain.jdbc.GridJdbcDriver");

// Open JDBC connection.
```

```
Connection conn =
DriverManager.getConnection("jdbc:gridgain://localhost/cache");

// Query persons' names
ResultSet rs = conn.createStatement().executeQuery("select name from
Person");

while (rs.next()) {
    String name = rs.getString(1);

    ...
}

// Query persons with specific age
PreparedStatement stmt = conn.prepareStatement("select name, age from
Person where age = ?");

stmt.setInt(1, 30);

ResultSet rs = stmt.executeQuery();

while (rs.next()) {
    String name = rs.getString("name");
    int age = rs.getInt("age");

    ...
}
```

# Client Router

Router allows remote clients to connect to grid without direct access to the network where grid is running. It accepts requests on the same protocol as grid rest server, either directly via TCP or over HTTP, and routes those requests to the specified grid nodes.

## Starting router

GridGain supports two different protocols for client connections and therefore two kinds of router are provided:

- `org.gridgain.client.router.GridTcpRouter`
- `org.gridgain.client.router.GridHttpRouter`

All router instances are managed by `org.gridgain.client.router.GridRouterFactory`. An

instance of `GridTcpRouterConfiguration` or `GridHttpRouterConfiguration` should be passed to the factory in order to start a new router.

### Standalone Startup

Router can be started as a standalone process by executing `ggrouter.sh` or `ggrouter.bat` scripts located under `GRIDGAIN_HOME/bin` folder. See `clients/java/config/router/default-router.xml` for configuration example. This file has all configuration properties commented out - feel free to uncomment them as needed.

If no configuration file is passed as a parameter to startup script, then `clients/java/config/router/default-router.xml` is used by default. You can start router with custom configuration as follows:

```
bin/ggstart.sh path/to/my-router-config/xml
```

### Embedded Startup

Router can also be started in embedded fashion from code. As a default configuration implementation you can use `GridTcpRouterConfigurationAdapter` or `GridHttpRouterConfigurationAdapter` depending on weather you will be using TCP-based or HTTP connectivity. Note that all configuration parameters are optional and you can run router with default configuration just by passing freshly created configuration instance.

Below is an example on how to start router with non-default configuration.

```
GridTcpRouterConfigurationAdapter cfg =
    new GridTcpRouterConfigurationAdapter();

cfg.setHost("router.appdomain.com");
cfg.setPort(11211);

cfg.setServers(Arrays.asList(
    "node1.appdomain.com:11211",
    "node2.appdomain.com:11211"));

GridRouterFactory.startTcpRouter(cfg);
```

## Client configuration

Note that clients

should be specifically configured in order to use router. Please refer to

`org.gridgain.client.GridClientConfiguration.getServers()` and `org.gridgain.client.GridClientConfiguration.getRouters()` for more details.

# Hadoop Pig Integration

GridGain's `In-Memory Accelerator For Hadoop` edition is based on the industry's first high-performance dual-mode in-memory file system that is 100% compatible with HDFS. GridGain File System (GGFS) is a plug-and-play alternative to the disk-based Hadoop HDFS enabling up to 10x faster performance for IO and network intensive Hadoop MapReduce jobs running on any Hadoop distribution.

Unlike any other file system, GGFS can work as either a **standalone file system** in Hadoop cluster, or work in tandem with HDFS, providing a **primary caching layer** for the secondary HDFS. As a caching layer it provides highly configurable read-through and write-through behavior. In either case GGFS can be used as a drop-in alternative for, or an extension of, standard HDFS providing an instant performance boost.

## PIG installation (optional)

GGFS was tested with PIG version `pig-0.11.1` and hadoop version `hadoop-2.0.3-alpha`. We assume here that you have installed Hadoop and GridGain and configured GGFS. For hadoop and GGFS installation instructions please refer to `readme-hadoop.pdf`.

For initial PIG installation refer to PIG documentation available on http://pig.apache.org/docs/r0.11.1/

For GGFS integration you should do the following:

1. Copy these files from `GRIDGAIN_HOME/libs` folder and `GRIDGAIN_HOME` folder to `${HADOOP_HOME}/share/hadoop/common/lib`:
   - gridgain-{version}.jar
   - jline-2.9.1.jar
   - mail-1.4.7.jar
   - spring-beans-3.2.2RELEASE.jar
   - spring-core-3.2.2RELEASE.jar
   - tika-core-1.3.jar
   - tika-parsers-1.3.jar
2. Make sure `HADOOP_CONF_DIR` environment variable is pointing to GGFS hadoop configuration directory as it is described in `in-memory-hadoop-readme.pdf`.

After these steps are completed you can start GGFS nodes, Hadoop MapReduce and run PIG scripts referring to GGFS file system. Please review to `in-memory-hadoop-readme.pdf` for information of how to start GGFS with Hadoop.

## GGFS & PIG Performance Considerations

In this section we will describe usage patterns that should be used to exploit maximal GGFS performance.

## File Creation

Try to make your file writes are on local-node only.

By default GGFS will spread each file across the grid, which will involve a lot of network traffic which will slow down the file write. In most Hadoop applications task files are created on all nodes in the cluster and file sizes are roughly equal (often this can be easily achieved by simply increasing number of mappers or reducers in each particular task). In this case these files may be created with replication factor equal to `1` which will force GGFS to place whole file on local node (the node on which this write is performed). We will call such files `collocated`. Note that GGFS will throw out of space error if `collocated` file is being written and no space left available on local node (even if other nodes have enough free space).

Replication factor may be specified at file creation either in direct API call to file system or by setting `dfs.replication` configuration property either globally in `core-site.xml` file or on per-mapreduce-task basis.

## File Scanning

Try to make your reads as consecutive as possible.

We noticed that PIG generates task splits based on file system block size and does not adjust consecutive splits to a single task. With large files this can result into excessive number of map tasks generated, and overhead for task start will be comparable to file read time. If MapReduce task reads collocated file, you can take advantage of knowledge that whole file resides on single node and override block size for this task. As a result, PIG will generate fewer tasks which will result in better performance.

For example, if you have 6 source data files colocated on 6 8-core nodes and each file is 4 GBytes, then it will make sense to set block size equal to 512MBytes, so that 8 splits will be generated for each file yielding 100% CPU utilization. If block size is less then 512Mbytes, more then 8 splits will be generated for each node.

You can override GGFS block size by setting `fs.ggfs.block.size` to task configuration.

> **NOTE:** Overriding this property will not change actual file layout in GGFS neither in collocated mode, nor in distributed. It will only affect the value provided to Hadoop MapReduce task.

## Buffer Sizes

Generally, the larger the buffer you provide for GGFS streams, the better performance you will get on file writes and scans. For our configuration optimal values are `128k` buffer for output streams

and `2M` buffer for input streams (it may depend on hardware and on operating system you use).

You can set buffer sizes either via direct API calls on file system or by setting `io.file.buffer.size` configuration property.

# Performance Tips

> **About These Tips**
>
> GridGain provides one of the most sophisticated APIs and features for in-memory data and compute processing. It is also shipped with a long list of examples and variety of configuration files and options. Out of the box, GridGain's **default configuration is tuned for examples and simple evaluation.** In the same time such configuration is often inadequate in production, testing or benchmarking environments. This documents provides basic overview of some of the key performance and configuration tips that you can employ to make GridGain work dramatically faster in many cases.

## Multithreaded Access to GridGain

If you are experiencing somewhat slow performance for cache updates, you should ask yourselves whether you are utilizing full computing power (all the cores) on your machine. GridGain is multi-threaded internally, but if you are doing sequential operations one after another from a single thread, then you are not using multithreading. Generally it makes sense to use the amount of threads of about 2 or 3 times the number of cores for populating grid. All GridGain APIs are thread-safe, so you don't have to worry about any concurrency issue when populating data.

## Disable Internal Events Notification

GridGain has rich event system to notify its users about almost everything that happens in the system. However, if you do not subscribe for certain events or your application does not depend on GridGain events at all it will be better to disable all or some events. See `GridConfiguration.getIncludeEventTypes()` configuration parameter for more information.

## Configure Thread Pools

By default GridGain has it's main thread pool size set to available CPU count. Thread pools are defined by `GridConfiguration.getExecutorService()` and `GridConfiguration.getSystemExecutorService()` configuration properties. In most cases keeping a thread per core will result in faster application performance, since there will be less context switching and CPU caches will work better. However, if you are expecting that your jobs will block for I/O or any other reason, it may make sense to increase the thread pool size. Try to change pools sizes to see if it affects application throughput. Using `vmstat` utility to monitor

context switching may help here.

## Tune Communication Buffers

`GridTcpCommunicationSpi` by default uses connection buffers which is best for most scenarios. However, if you are sending large messages across network (e.g. large job arguments or results, cache keys or values, etc...) and estimated or measured average message size is greater than communication buffer size (defined by `GridTcpCommunication.setConnectionBufferSize()`) it will be better to disable buffer altogether and add more threads to the thread pools defined by `GridConfiguration.getExecutorService()` and `GridConfiguration.getSystemExecutorService()` configuration parameters.

## Use Externalizable Whenever Possible

It is a very good practice to have everything that is transferred over network implement `java.io.Externalizable`. These may be cache keys or values, jobs, job arguments, or anything else that will be sent across to other grid nodes. Implementing `Externalizable` may sometimes result in over 10x performance boost over standard serialization.

## Use Collocated Computations

GridGain enables you to execute MapReduce computations in memory. However, most computations usually work on some data which is cached on remote grid nodes. Loading that data from remote nodes is very expensive in most cases and it is a lot more cheaper to send the computation to the node where the data is. The easiest way to do it is to use `GridProjection.affinityRun()` method; however GridGain has plenty of `mapKeysToNodes()` methods to help users figure out custom data ownership within Grid.

## Use Data Loader

If you need to upload lots of data into cache, use `org.gridgain.grid.GridDataLoader` to do it. Data loader will properly batch the updates prior to sending them to remote nodes and will properly control number of parallel operations taking place on each node to avoid thrashing. Generally it provides performance of 10x than doing a bunch of single-threaded updates.

See
`org.gridgain.examples.cache.loaddata.dataloader.GridCacheDataLoaderExample`
under *examples* folder for more details.

## Use Bulk Loader

Often data needs to be loaded from an external persistent store, for example, to make sure that all caches have all required data at startup or at any point later on. The best way to do that is to

utilize `org.gridgain.grid.cache.GridCache.loadCache(...)` method which ultimately delegates to `org.gridgain.grid.cache.store.GridCacheStore.loadAll(...)` method. This method works for all types of caches and essentially makes sure that every node can load data for itself independently from others. The biggest advantage of this approach over using `GridDataLoader` is that it is local to every individual node and requires no distribution.

See `org.gridgain.examples.cache.loaddata.storeloader.GridCacheStoreLoaderExample` under *examples* folder for more details.

## Turn Off Peer Class Loading

While peer class loading is very convenient in development, it does carry a certain overhead and should be turned off in production. To do so, you should explicitly set `GridConfiguration.isPeerClassLoadingEnabled()` configuration property to false.

## Batch Up Your Messages

If you can send 10 bigger jobs instead of 100 smaller jobs, you should always choose to send bigger jobs. This will reduce the amount of jobs going across the network and may significantly improve performance.

## Tune Initial Cache Capacity

To avoid internal resizing of cache maps you should always provide proper cache start size - not doing so can significantly hurt performance as some CPU cycles will be spent on GridGain resizing internal cache maps instead of application logic. You can configure cache start size via `GridCacheConfiguration.getStartSize()` configuration property.

## Tune Near Cache

When using Partitioned cache, GridGain will front this cache with local Near cache to make sure that if entry does not belong to local partitions, it will still be cached in a smaller local cache for better performance on next access.

However, most usages of GridGain happen from collocated computations, i.e. computations submitted to the grid are usually routed to the nodes where the data resides automatically. In cases like this, using Near cache is redundant, as all data access happens from memory anyway. To save on performance, you can disable Near cache by setting `GridCacheConfiguration.isNearEnabled()` configuration property.

## Data Replication vs. Data Invalidation

By default cache will propagate all data changes to other nodes (e.g. back up nodes) to keep data consistent. However, if data does not need to be replicated (e.g. it can always be recreated by

reading through from persistent store), then sometimes it makes sense to invalidate remote copies instead of updating them. Invalidation messages are much smaller and have significantly less impact of network and performance. Use invalidation with caution as it will effectively remove all remote data copies on any update.

## Tune Off-Heap Memory

If you plan to allocate large amounts of memory to your JVM for data caching (usually more than 10GB of memory), then your application will most likely suffer from prolonged lock-the-world GC pauses which can significantly hurt latencies. To avoid GC pauses use off-heap memory to cache data - essentially your data is still cached in memory, but JVM does not know about it and GC is not affected.

The only configuration property to set to enable off-heap memory is `GridCacheConfiguration.getMaxOffHeapMemory()` which will tell GridGain how much off-heap memory to make available for your application. By default off-heap memory is disabled.

## Tune Swap Storage

First of all, if you don't plan to use swap storage (i.e. disk overflow storage), you should not change any default swap settings (swap storage is disabled by default). If you do need to use swap storage, then you should enable it via `GridCacheConfiguration.isSwapEnabled()` configuration property.

## Tune Query Indexing

There are several configuration properties that you should watch out for here. First of all and most importantly, if you don't plan to use cache queries at all, you should disable indexing altogether via `GridCacheConfiguration.isQueryIndexEnabled()` configuration property.

If you do plan to use cache queries, you should properly enable/disable indexing of primitive keys and values on `GridH2IndexingSpi`. You should enable indexing for primitive keys by setting `setDefaultIndexPrimitiveKey()` to `true` on the SPI only if you plan to use primitive cache keys in your cache queries. The same goes for indexing primitive values controlled by `setDefaultIndexPrimitiveValue()` property.

Also, if for every value class you don't plan to have different key classes (essentially every value class has one key class), set `setDefaultIndexFixedTyping()` on the SPI to `true`. This way GridGain will store key types as corresponding SQL types instead of binary form which provides faster performance for key lookups.

## Tune Eviction Policy

If you don't plan to over-populate your cache, i.e. if you don't need any eviction policy at all, then you should disable eviction policy altogether via

`GridCacheConfiguration.isEvictionEnabled()` configuration property.

If you do need GridGain to make sure that data in cache does not overgrow beyond allowed memory limits, you should carefully choose the eviction policy you need. Most likely you will need either FIFO or LRU eviction policies shipped with GridGain, however depending on your application, you may need to plugin your own custom eviction policy. Regardless of which eviction policy you use, you should carefully chose the maximum amount of entries in cache allowed by eviction policy - if cache size overgrows this limit, then evictions will start occurring. Usually max size is controlled by `setMaxSize()` configuration property on the instance of eviction policy.

You should also almost always configure `setAllowEmptyEntries()` configuration property to `false`. By default, GridGain will keep entries with `null` values in cache to preserve some other properties of the entry, like time-to-live, for example. However, if you don't use time-to-live then most likely you should discard the entry once it gets expired or invalidated.

## Use Write-Behind Caching

If you can afford for your persistent store to be behind your in-memory cache, then use *write-behind* caching. When write-behind is enabled, GridGain will batch up cache updates and flush them to database in batches in the background which can often provide significant performance benefits. You can enable *write-behind* caching via `GridCacheConfiguration.isWriteBehindEnabled()` configuration property.

## Tune Cache Preloading (Repartitioning)

There are a couple of parameters you need to worry about here to make sure that cache preloading (or repartitioning) does not have negative impact on your application performance:

- You should configure preload batch size to make sure it is appropriate for your network. Default is 512KB which means that by default preload messages will be about 512KB, however, you may need to set this value to be higher or lower based on your network performance. This parameter is defined via `GridCacheConfiguration.getPreloadBatchSize()` configuration property.
- When preloading large data sets, the CPU and network can get over-consumed with preloading messages, which consecutively may slow down the application performance. In this case you should enable data preload throttling which helps tune the amount of time to wait between preload messages to make sure that preloading process does not have any negative performance impact. Note that application will continue to work properly while preloading is still in progress. Preload throttling is disabled by default and can be enabled via `GridCacheConfiguration.getPreloadThrottle()` configuration parameter.

## Use Key Batching

When mapping keys to nodes for affinity job execution it is always more efficient to group multiple keys into a job before sending it to a node. Essentially if you have million of keys, then it is always better to split them up into batches, say of 10,000, then mapping keys batch-by-batch to data nodes, and executing a job by sending a collection of keys directly to the node they are cached on. A sample code snippet which maps a batch of keys to remote nodes looks like following:

```
List<MyKey> batchKeys = new ArrayList<MyKey>(batchSize);

for (int i = startBatch; i < endBatch; i++)
    batchKeys.add(MyKey(i));

// Map keys to nodes.
Map<GridRichNode, Collection<GridDemoKey>> mappings =
cache.mapKeysToNodes(batchKeys);

// Collocate computations and data.
for (Map.Entry<GridRichNode, Collection<GridDemoKey>> e :
mappings.entrySet()) {
    GridRichNode node = e.getKey();
    Collection<MyKey> keys = e.getValue();

    // Send colocated job to specified node.
    node.run(new MyJob(keys));
}
```

Batching keys as suggested above can often significantly improve performance vs. sending a job per key.

# Scalar

Scalar is a Scala-based **Domain Specific Language** (DSL) that is build on top of GridGain APIs and runtime. The main purpose of Scalar is to adopt GridGain APIs to Scala language and provide simpler and smoother usage model for those who chose to use Scala language when working with GridGain. We highly recommend Scala as the main language to be used with GridGain as it provides much reacher environment for distributed development with GridGain.

Scalar does not add any new significant functionality to GridGain but rather provides implicit and explicit conversions to adapt Java APIs to Scala language. Keep also in mind that you can use GridGain APIs directly from any Scala-based application and there is no technical need for Scalar - but with Scalar many things are easier and more natural when working with GridGain and Scala language.

# Scripts

### ggstart.{sh | bat}

This script allows to start grid node with either default configuration (if nothing is specified) or with specific configuration when configuration file is passed in.

### ggvisorui.{sh | bat}

This script starts GUI-based Visor - GridGain Admin Console. GridGain Visor provides a single unified operations, management and monitoring console across all GridGain products and for any applications and systems built with GridGain.

### ggvisorcmd.{sh | bat}

This script starts CLI-based Visor that is based on Scala REPL. Visor gets loaded by loading Scala script *bin/visor.scala*. Note that you can freely modify script *bin/visor.scala* to load custom commands, pre-built scripts, etc.

### ggrouter.{sh | bat}

This script starts up a router instance for remote clients to connect to. Router serves a role of gateway to tunnel remote client communication to the grid as opposed to having remote clients connect to individual nodes directly.

### include/ directory

All scripts in `include/` directory are for internal use only and should not be executed directly.

# System Properties

The following system properties can be used to affect GridGain runtime during its start. They can be set in standard `ggstart.{sh|bat}` scripts, in IDE runtime configurations, SBT/Ant/Maven settings, etc.

## *General*

### GRIDGAIN_RESTART_CODE

This property is used internally to pass an exit code to loader when GridGain instance is being restarted.

## GRIDGAIN_DAEMON

Presence of this system property with value `true` will make the grid node start as a daemon node. Node that this system property will override `GridConfiguration#isDaemon()` configuration.

## GRIDGAIN_HOME

Provides GridGain installation folder.

## GRIDGAIN_NO_SHUTDOWN_HOOK

If this system property is set to `false` - no shutdown hook will be set.

## GRIDGAIN_NO_DISCO_ORDER

System property to disable requirement for proper node ordering by discovery SPI. Use with care, as proper node ordering is required for cache consistency. In other words, setting this property to `true` will lead to **inconsistent data** in in-memory data grid. When set to `true`, any discovery SPI can be used with distributed cache, otherwise, only discovery SPIs that have annotation `@GridDiscoverySpiOrderSupport(true)` will be allowed.

## GRIDGAIN_UPDATE_NOTIFIER

If this system property is set to `false` - no checks for new versions will* be performed by GridGain. By default, GridGain periodically checks for the new* version and prints out the message into the log if new version of GridGain is available for download.

## GRIDGAIN_STARVATION_CHECK_INTERVAL

This system property defines interval in milliseconds in which GridGain will check thread pool state for starvation. Zero value will disable this checker.

## GRIDGAIN_NEAR_GET_MAX_REMAPS

This property defines the maximum number of attempts to remap near get to the same primary node. Remapping may be needed when topology is changed concurrently with get operation.

## GRIDGAIN_PROG_NAME

Name of the system property defining name of command line program.

## GRIDGAIN_SUCCESS_FILE

Name of the system property defining success file name. This file is used with auto-restarting functionality when GridGain is started by supplied `ggstart.{bat|sh}` scripts.

## GRIDGAIN_LOCAL_HOST

Name of system property to set system-wide local IP address or host. If provided it will override all default local bind settings within GridGain or any of its SPIs. Note that system-wide local bind address can also be set via `GridConfiguration#getLocalHost()` method. However, system properties have priority over configuration properties specified in `GridConfiguration`.

## GRIDGAIN_FUTURE_SYNC_NOTIFICATION

Name of the system property or environment variable to activate synchronous listener notification for future objects implemented in GridGain. I.e. closure passed into method `GridFuture#listenAsync(GridInClosure)` will be evaluated in the same thread that will end the future. See `GridFuture#syncNotify()`.

## GRIDGAIN_FUTURE_CONCURRENT_NOTIFICATION

Name of the system property or environment variable to activate concurrent listener notification for future objects implemented in GridGain. I.e. upon future completion every listener will be notified concurrently in a separate thread. See `GridFuture#concurrentNotify()`.

## GRIDGAIN_STOPWATCH

System property to enable or disable stop watch (`true` or `false`, default is `false`). If enabled, GridGain will watch various internal execution points and will print out stats at the end.

## GRIDGAIN_STOPWATCH_PRINTSTEP

If `GG_STOPWATCH_ENABLED` is `true`, then this property will control whether or not to print out every execution step or not.

## GRIDGAIN_DEPLOYMENT_MODE_OVERRIDE

System property to override deployment mode configuration parameter. Valid values for property are: `PRIVATE`, `ISOLATED`, `SHARED` or `CONTINUOUS`. See `GridDeploymentMode` and `GridConfiguration#getDeploymentMode()`.

## GRIDGAIN_MAX_COMPLETED_TX_COUNT

Property controlling size of buffer holding completed transaction versions. Such buffer is used to detect duplicate transaction and has a default value of `102400`. In most cases this value is large

enough and does not need to be changed.

### GRIDGAIN_MAP_CONCURRENCY_LEVEL

Concurrency level for all concurrent hash maps created by GridGain.

### GRIDGAIN_SLOW_TX_WARN_TIMEOUT

Transactions that take more time, than value of this property, will be output to log with warning level. `0` (default value) disables warning on slow transactions.

### GRIDGAIN_OVERRIDE_MCAST_GRP

System property to override multicast group taken from configuration. Used for testing purposes.

### GRIDGAIN_REFLECTION_CACHE_SIZE

System property to override default reflection cache size. Default value is `128`.

### GG_JOBS_HISTORY_SIZE

System property to override default job processor maps sizes for finished jobs and cancellation requests. Default value is `10240`.

### GRIDGAIN_CONFIG_URL

System property to hold optional configuration URL.

### GRIDGAIN_PRELOAD_RESEND_TIMEOUT

System property to hold preload resend timeout for evicted partitions.

### GRIDGAIN_MARSHAL_BUFFERS_RECHECK

System property to specify how often in milliseconds marshal buffers should be rechecked and potentially trimmed. Default value is `10,000ms`.

### GRIDGAIN_DISABLE_HOSTNAME_VERIFIER

System property to disable `HostnameVerifier` for SSL connections. Can be used for development with self-signed certificates. Default value is `false`.

## *Logging*

### GRIDGAIN_DEBUG_ENABLED

Property identifying whether GridGain debug is enabled or not. If enabled, the standard logger configuration will be used, otherwise logger settings will be bypassed and debug won't be printed.

### GRIDGAIN_LOG_GRID_NAME

If this system property is present the GridGain will include grid name into verbose log.

### GRIDGAIN_SCRIPT

Defines GridGain property indicating support for ANSI escape sequences. Set it to any non-`null` value to indicate that runtime support ANSI escape sequences.

### GRIDGAIN_ANSI_OFF

GridGain property explicitly disabling ANSI coloring of log messages. Set it to any non-`null` value to indicate that runtime does not support ANSI escape sequences.

### GRIDGAIN_NO_ASCII

If this system property is present (any value) - no ASCII logo will be printed.

### GRIDGAIN_QUIET

Set to either `true` or `false` to enable or disable quiet mode of GridGain. In quiet mode, only warning and errors are printed into the log additionally to a shortened version of standard output on the start. Note that if you use `ggstart.{sh|bat}` scripts to start GridGain they start by default in quiet mode. You can supply `-v` flag to override it. Note also that in quiet mode no other log is produced (no INFO or DEBUG even if they are enabled in the log configuration).

### GRIDGAIN_DFLT_LOG4J_DEBUG

If set to any value will set default log level for Log4j to `DEBUG`.

### GRIDGAIN_ASSERT_SEND_DISABLED

System property to disable emailing of assertion errors. See `GridThread`.

## *SMTP, Emails*

### GRIDGAIN_SMTP_HOST

Name of the system property or environment variable to set or override SMTP host. If provided - it will override the property in grid configuration. Note that GridGain uses SMTP to send emails in critical situations such as license expiration or fatal system errors. It is **highly** recommended to configure SMTP in production environment. See `GridConfiguration#getSmtpHost()`

## GRIDGAIN_SMTP_PORT

Name of the system property or environment variable to set or override SMTP port. If provided - it will override the property in grid configuration. Note that GridGain uses SMTP to send emails in critical situations such as license expiration or fatal system errors. It is **highly** recommended to configure SMTP in production environment. See `GridConfiguration#getSmtpPort()` and `GridConfiguration#DFLT_SMTP_PORT`

## GRIDGAIN_SMTP_USERNAME

Name of the system property or environment variable to set or override SMTP username. If provided - it will override the property in grid configuration. Note that GridGain uses SMTP to send emails in critical situations such as license expiration or fatal system errors. It is **highly** recommended to configure SMTP in production environment. See `GridConfiguration#getSmtpUsername()`.

## GRIDGAIN_SMTP_PASSWORD

Name of the system property or environment variable to set or override SMTP password. If provided - it will override the property in grid configuration. Note that GridGain uses SMTP to send emails in critical situations such as license expiration or fatal system errors. It is **highly** recommended to configure SMTP in production environment. See `GridConfiguration#getSmtpPassword()`.

## GRIDGAIN_SMTP_FROM

Name of the system property or environment variable to set or override SMTP FROM email. If provided - it will override the property in grid configuration. Note that GridGain uses SMTP to send emails in critical situations such as license expiration or fatal system errors. It is **highly** recommended to configure SMTP in production environment. See `GridConfiguration#getSmtpFromEmail()` and `GridConfiguration#DFLT_SMTP_FROM_EMAIL`.

## GRIDGAIN_ADMIN_EMAILS

Name of the system property or environment variable to set or override list of admin emails. Value of this property should be comma-separated list of emails. If provided - it will override the property in grid configuration. Note that GridGain uses SMTP to send emails in critical situations

such as license expiration or fatal system errors. It is **highly** recommended to configure SMTP in production environment. See `GridConfiguration#getAdminEmails()`.

### GRIDGAIN_SMTP_SSL

Name of the system property or environment variable to set or override whether or not to use SSL. If provided - it will override the property in grid configuration. Note that GridGain uses SMTP to send emails in critical situations such as license expiration or fatal system errors. It is **highly** recommended to configure SMTP production environment. See `GridConfiguration#isSmtpSsl()` and `GridConfiguration#DFLT_SMTP_SSL`.

### GRIDGAIN_LIFECYCLE_EMAIL_NOTIFY

Name of the system property or environment variable to set or override whether or not to enable email notifications for node lifecycle. If provided - it will override the property in grid configuration. See `GridConfiguration#isLifeCycleEmailNotification()`.

### GRIDGAIN_SMTP_STARTTLS

Name of the system property or environment variable to set or override whether or not to use STARTTLS. If provided - it will override the property in grid configuration. Note that GridGain uses SMTP to send emails in critical situations such as license expiration or fatal system errors. It is **highly** recommended to configure SMTP in production environment. See `GridConfiguration#isSmtpStartTls()` and `GridConfiguration#DFLT_SMTP_STARTTLS`.

## *Jetty, REST*

### GRIDGAIN_JETTY_HOST

This property allows to override Jetty host for REST processor.

### GRIDGAIN_JETTY_PORT

This property allows to override Jetty local port for REST processor.

### GRIDGAIN_REST_MAX_TASK_RESULTS

This property allows to override maximum count of task results stored on one node in REST processor.

## *Visor SSH*

## GRIDGAIN_SSH_HOST

System property to hold SSH host for visor-started nodes.

## GRIDGAIN_SSH_USER_NAME

System property to hold SSH user name for visor-started nodes.

## GRIDGAIN_SSH_LOG_DIR

System property to hold custom log directory for visor-started nodes.