

Table Of Contents

- 1. GridGain In-Memory Database
- 2. GridGain Installation
 - 2.1 Check GridGain Installation
 - 2.2 Running GridGain Examples
 - 2.3 Configure GridGain Node Discovery
- 3. Starting Grid Nodes
- 4. Management & Monitoring with Visor
- 5. Scala Integration
- 6. Javadoc & Scaladoc

1. GridGain In-Memory Database

GridGain's In-Memory Database is distributed, object-based, ACID transactional, in-memory, SQL+NoSQL key-value database. Let's look at each characteristic individually:

Charactersitic	Description
Distributed	Unlike sharding systems - GridGain is distributed by default and has been built this way from the ground up. In other words, GridGain allows you to install itself on multiple computers and it will automatically coordinate data management across these computers seamlessly for the user so that the entire cluster can be viewed as a single in-memory database. Moreover, you can seamlessly add and remove nodes from topology and GridGain will properly handle the data repartitioning, transactions, indexing, etc. NOTE: GridGain has been independently tested and have production installations with over thousands computers in the grid.
SQL+NoSQL Key-Value	Data in in-memory database is kept in a form of key/value pairs. In a simplistic approximation you can think of in- memory database as a collection of distributed hash maps. Both keys and values can be of any type (including any containers or user-defined types). It is important to note that key/value representation doesn't mean primitive types or byte-arrays only. GridGain just as easily supports document APIs where value represent JSON or any other document- style representation. GridGain provides many interfaces to query and process its data: SQL, Key-Value, MapReduce, File System, MPP and MPI-like processing.
Transactional	Unlike most of NoSQL products that provide only Eventual Consistency (EC) model, GridGain provides full ACID (Atomicity, Consistency, Isolation, Durability) compliant transactions that ensure guaranteed consistency. Moreover, GridGain provides various performance optimizations that allow users to relax some of the ACID properties in exchange for even greater performance.
In-Memory	GridGain treats memory (on-heap and off-heap) as a primary media to store application data. It also natively supports disk-based swap space to integrate with non-volatile storage like HDDs, SSDs, SANs, etc. as well as cache stores that allow GridGain in-memory database to read-through from and write-through to an existing data stores like SQL RDBMS, HDFS, HBase, etc. However, the main algorithms of GridGain (ACID transactional system, MapReduce, SQL processing, etc.) are heavily optimized for in-memory processing.
Object-Based	As we mentioned above - both keys and values can be of any type supported by JVM. That allows to store any business domain objects of any internal complexity in in-memory database directly without any need to marshall and demarshall from any external representations such as binary, XML or JSON protocols. This in turn provides two key advantages: 1) Developers can work directly with domain objects enjoying natural, convenient and simple to use APIs, and 2) Object-based storage leads to better performance as objects don't have to be constantly marshaled from

Key Features

In-Memory Database edition includes everything provided by In-Memory HPC edition.

GridGain In-Memory Database provides the following key features:

- Distributed java-based in-memory key-value store
- Distributed ANSI SQL and Lucene in-memory query capability over cached data
- PESSIMISTIC and OPTIMISTIC ACID distributed transactions
- Affinity routing with compute grid
- Local, fully replicable, and partitioned cache types
- Partitioned caches with active backups
- · Support for data replication and invalidation in sync/async modes
- · Concurrent, transactional, and delayed data preloading
- Pluggable expiration policies (LRU, LIRS, random, time-based)
- · Read-through and write-through logic with pluggable cache store
- Synchronous and asynchronous cache operations
- MVCC-based concurrency with HyperLocking technology
- Pluggable network segmentation (split brains) resolution
- Pluggable data overflow storage
- READCOMMITTED, REPEATABLEREAD, SERIALIZABLE isolation levels
- JTA/JCA transaction integration
- Write-behind cache store support
- Datacenter replication
- Customizable/pluggable data indexing support
- JDBC driver for in-memory object data store
- Off-Heap BigMemory support
- Tiered storage with on-heap, off-heap, swap space, SQL, and Hadoop
- Zero deployment for data
- OOP and FP-based APIs for Java and Scala

2. GridGain Installation

GridGain distribution comes in a ZIP file that simply needs to be unzipped and GRIDGAIN_HOME environment variable should be set to point to distro folder under GridGain installation.

You can use any tools to perform this installation on multiple computers. There are no additional steps required for GridGain installation in such multi machine setup.

Installation requirements:

- 1. Windows, Linux, or MacOS environment.
- 2. Java 7 (latest update is advisable).
- 3. Point JAVA_HOME environment variable to your JDK or JRE installation.
- 4. Point GRIDGAIN_HOME environment variable to the distro folder under GridGain installation.

2.1 Check GridGain Installation

To verify GridGain installation, you can execute the GridGain startup script.

The following command will startup GridGain with default configuration using Multicast node discovery.

bin/ggstart.{shlbat}

The following command will startup GridGain with default configuration using TCP node discovery for all nodes running on local host.

bin/ggstart.{shlbat} config/tcp/spring-tcp-vm.xml

If GridGain was installed successfully, the output from above commands should produce no exceptions or errors. Note that you may see some warnings during startup, but this is OK as they are meant to inform that certain functionality is turned on or off by default.

You can execute the above commands multiple times on the same machine and make sure that nodes discover each other. Here is an example of log printout when 2 nodes join topology:

... Topology snapshot [nodes=2, CPUs=8, hash=0xD551B245]

You can also start GridGain Management Console, called Visor, and observe started nodes show up on Visor Dashboard. To startup Visor in GUI mode, you should execute the following script:

/bin/ggvisorui.{shlbat}

2.2 Running GridGain Examples

GridGain comes with many well documented examples. All examples have documentation about how they should be started and what the expected outcome should be. Start off by running simple examples, such as examples under org.gridgain.examples.hpc.helloworld package for HPC, org.gridgain.examples.database.putget package for database, org.gridgain.examples.ggfs package for GGFS, ogr.gridgain.examples.streaming package for streaming, and then move to others.

For affinity colocation with HPC edition (i.e. colocation of computations with data) you should take a look at examples in **org.gridgain.examples.database.affinity** package. You should also take a look at examples under **org.gridgain.examples.database.loaddata** to see how GridGain to see various ways of loading data into GridGain including **realtime** example which loads and queries data concurrently in real time.

If you are using Eclipse or IntelliJ IDEA you can play with examples using pre-configured projects that are shipped with GridGain.

2.2.1 IDEA Users

IDEA users should open IDEA project file called idea_users_open_this_file.ipr located under GridGain installation.

2.2.2 Eclipse Users

Eclipse users should follow instructions specified in eclipse_users_read_here.txt under GridGain installation.

2.2.3 Other IDEs

If you are using IDE other than Eclipse or IDEA, you can still view GridGain examples by creating a project manually in IDE of your choice. Make sure to include gridgain-x.x.x.jar file and and libraries located under libs folder as dependencies into your project.

2.3 Configure GridGain Node Discovery

When using TCP-based discovery make sure to update configuration for IP Finder with actual IP addresses like so:

```
<property name="discoverySpi">
<bean class="org.gridgain.grid.spi.discovery.tcp.GridTcpDiscoverySpi">
<property name="ipFinder">
<bean class="org.gridgain.grid.spi.discovery.tcp.ipfinder.vm.GridTcpDiscoveryVmIpFinder">
<property name="addresses">
<list>
<list>
<value>10.1.2.3:47500</value>
<value>10.1.2.4:47501</value>
</list>
</property>
</bean>
</property>
```

On startup, GridGain node will try to connect to the specified IP addresses one-by-one until it succeeds.

NOTE: you are only required to specify at least 1 IP address of the grid node that will be started first - other IP addresses are optional.

3. Starting Grid Nodes

Grid nodes can be started by executing bin/ggstart.{shlbat} script and passing a relative path to GridGain configuration file. If no file is passed, then grid nodes are started with default configuration using Multicast discovery protocol. Here is an example of how to start GridGain node with non-default configuration:

`bin/ggstart.sh config/tcp/spring-cache.xml`

Note that only TCP discovery is supported for In-Memory Database (Multicast discovery is not supported). This means that you cannot use GridGain In-Memory Database with default configuration and should always pass configuration file with cache configuration along with TCP discovery configuration. Example of such configuration can be found in examples/config/tcp/spring-cache.xml configuration file.

4. Management & Monitoring with Visor

GridGain comes with GUI and CLI (command) based DevOps Managements Consoles delivering advance set of management and monitoring capabilities. Visor GUI is based on a standalone Java application and CLI version is built on top of Scala REPL providing fully scriptable and customizable DevOps environment.

To start Visor GUI console, execute the following command:

`bin/ggvisorui.sh`

To start Visor in console mode you should execute the following command:

`bin/ggvisor.sh`

On Windows, run the same commands with .bat extension.

NOTE: Visor GUI console has a much richer set of functionality over Visor command-based console. You should always prefer Visor GUI console whenever possible.

Here is an example of Visor Database Tab which provides overall view on database.

000		Visor – GridGain A	Admin Console			
<u>File</u> <u>V</u> iew						
Cache: meta × ANDAE: C1ED6E7	E × 🗣 Node: E83BB058 ×	🕽 Host: 192.168.1.2 × 🛛 🖼 Viewer 🛛 🐨 Telemetry	Profiler: ggfs ×	🔛 Profiler: ggfs (snapshot 💅 Streaming	: 17:27:40) × 🏐 File Manager: 1 ×	🗐 Cache: data 🚿
Cache Nodes: 2 Total Heap C Total Caches: 2 Total Heap U	apacity: 8.9CB Total Caches Size: Ised: 4.2CB Total Caches Keys:	76.9MB Total Off-Heap Size 198.0K Total Off-Heap Key	e: 0 Total Swap Si s: 0 Total Swap K	ze: 0 eys: 0		🗍 Undock 📮 Refresh
Show 30 ▼	min 🕑 Hits « 🗹 Misses	x 🗹 Reads x 🗸 Wi	rites 🛪 🗹 🗖 Comn	nits » 🗹 🦳 Rollbacks »	🖌 🗌 CPU x 🔽 🗌 Heap	*
25%		<u>`````````````````````````````````````</u>	W///// 17	w//w/////// 20	17:25	M-MMMM
Nodes With Caches: 2 Selected: 1 Hosts	: 1 Filter:			Free Heap %		ation
ClED6E7E 127.0.0.1 E83BB058 127.0.0.1	Jul 14, 13:02:04 4h 26r Jul 14, 13:02:04 4h 26r	n 3s 8 20 n 3s 8 20	.80% (0% 53.13% 0% 53.13%		_64 10.8.4 _64 10.8.4
			RestartStop			
			📌 Node 🗐 Host			
\$ Switch			✓ Ping Add To Group ► Image: Second seco			
Caches: 2 Selected: 1 Name 1 Mode	Cache Config G F	Partitions 🛛 🙀 SQL 🛛 🕵 C ear Size DHT Size Primary S	選 Threads 🔊 VisualVM	Swap 📳 Preload	i 😼 Load 🔄 Reset Writes Read, % Com	mits Rollbacks Commit, %
data PARTITIONED meta REPLICATED	6.9MB 238Bytes 30.6K 31.7MB 479Bytes 69.4K	0 30.6K 30. 0 0 69.	Second License	.7K 59.36% 129.7K .4K 99.93% 58.4M	134.7K 49.06× 11 716.0K 98.79× 31	.4.9K 0 100× .4.9K 2 100.00×
		-	Select All			
			G Copy			
Connected: ON Last Update: 17:28:1	0 H/N/C: 1/2/8 CPU Load	: 9.72% Free Heap: 53.13	« Up Time: 4h 26	im 3s Update: 3	s 📿 6 🧘	5302M of 5845M

Here is an example of Visor Cache Tab which provides view on individual cache.

ARTITIONED Mer. Size 7,000 Entr Sh	mory Size: 3.6MB :: 15.8K y size: 237Bytes 1000 30 ▼ min	Near Size: 0 DHT Size: 15.8 Primary size: 15.8	Off-heap Size: 0 Off-heap Keys: 0	Swap Size: 0 Swap Keys: 0		🗇 Unde	ock 🗣 Refresh 🛛 🛛 C	
Sh	now 30 🔻 min							
		Hits 🛪	Misses % 🖌 Read	ds %	Commits %	Rollbacks %		
		15:18:21 PM Hit ratio: 60%						
		The factor dow						
15.15		15:20		15.2	5	15-20		
13.13	Casha M	15.20		13.2	2	13.30		
SOL R Cle	Cache Ma	Avg. Ratio	Reads/s Avg.	Ratio Hits/s	Avg.	Ratio Commits/s	Min Dev. Max	
	Reads:	68.1K 49.0%	50 Hits: 40.5	K 59.4% 30	Commits: 60.3K	100% 44 Parts:	4.4K 11.3% 5.6P	
npact 🛛 🛃 Swa	ар	Avg. Ratio	Writes/s Avg.	Ratio Misses/s	Avg.	Ratio Rollback/s	Min Dev. Max	
oad 🛛 🐑 Res	set Writes:	70.7K 51.0%	52 Misses: 27.6	K 40.6% 20	Rollbacks: 0	0% 0 Keys:	7.1K 10.5% 8.7H	
losts: 1 Filter:								
lode IP	Start Time	Un Time	CPUIs CPUI load	% CC load %	Eree Hean %		ormation	
7.0.0.1	Jul 09, 15:11:42	22m 9s	8 7.	39% OC Load, %	85.16%	₫ Mac OS X x86_64 10.8.4		
7.0.0.1	Jul 09, 15:11:42	22m 9s	8 38.	24% 0%	85.16%	📹 Mac OS X x86_64 10.8.4		
	15:15 SQL Receiption of the second s	15:15 SQL Cache Me mpact Swap ad Reset Virtes: Writes: tosts: 1 Filter: Kode IP Start Time 7.0.0.1 Jul 09, 15:11:42	15:15:21 PM Ht ratio 600 15:15 15:15 3QL QL QL Reads: 68.1K 49.0s Avg. Ratio Writes: 70.7K 51.0s 4osts: 1 Filter: 4ode IP Start Time Up Time 7.0.0.1 Jul 09, 15:11:42 22m 9s	15:15 15:20 15:15 15:20 SQL Cache Metrics Avg. Ratio Reads: 68.1K 49.0x 50 Hits: 40.5 Writes: 70.7K S1.0.4 52 Misses: 27.6 Nots: 1 Filter: Node IP Start Time Up Time CPUs Avg. 8 7.0.0.1 Jul 09, 15:11:42 22m 9s 8 38.	15:15 15:20 15:15 15:20 3QL Cache Metrics Arg. Ratio Arg. Ratio Writes/s Avg. Ratio Writes/s Avg. Ratio Writes: 70.7K S1:1 Filter: Vertes: 71.0x S2 Misses: 27.6K 40.6 IP Start Time Up Time CPUs CPU Load, % CC Load, % 7.0.0.1 Jul 09, 15:11:42 22m 9s 8 38.24x 0x	15:15 15:20 15:15 15:20 15:15 15:20 SQL Question (Construction) Avg. Ratio Avg. Ratio Avg. Ratio Avg. Ratio Mitsels/s Avg. Reads: 68.1K 40.5K 59.4s 30 Commits: 68.1K 49.0x 50 Hits: Writes: 70.7K 51.0x 52 Misses/s Avg. Ratio Misses/s Avg. Ratio Writes: 70.7K 51.0x 52 Misses: 2 Reset Itime Up Time CPU Load, % CC Load, % Free Heap, % 7.0.0.1 Jul 09, 15:11:42 22m 9s 8 38.24s 0s 85.16s 7.0.0.1	15:15 15:20 15:25 15:30 15:15 15:20 15:25 15:30 30L QL Avg. Ratio Reads/s Avg. Ratio Keads/s Avg. Ratio Commits/s add Reads: 68.1K 49.0h 50 Hits: 40.5K 59.4s 30 Commits: 60.3K 100h 44 Parts: aad Reads: 68.1K 49.0h 50 Hits: 40.5K 59.4s 30 Commits: 60.3K 100h 44 Parts: aad Reads: 68.1K 49.0h 52 Misses/s Avg. Ratio Rollback/s 0 0 Keys: iosts: 1 Filter: Image: 27.6K 40.6% 20 Rollbacks: 0 0 Keys: iosts: 1 Filter: Image: 27.6K CLoad, % CC Load, % Free Heap, % Osi Info iosts: 1 Filter: Image: 27.39% 8 7.39% 0% 85.16% Mac OS X iode IP Start Time Up Tim	

5. Scala Integration

GridGain provides a very nice and easy to use DSL for Scala users called Scalar . If you like Scala, take a look at Scalar examples located under examples/scalar folder.

6. Javadoc & Scaladoc

We spend significant amount of time on writing and maintaining Javadoc and Scaladoc for our product. All documentation is shipped with it and you can find it under docs/javadoc and docs/scaladoc sub-folder respectively.