



GridGain 3.0

White Paper - July 2010

GridGain 2.0 Retrospect

Since its first release in the summer of 2007 GridGain software has become the leading Java-based distributed computing middleware with over 100,000 downloads and over 2,000,000 starts worldwide while working on any managed infrastructure - from a single Android device to thousands of nodes in the cloud. Today GridGain software starts every 20 seconds around the globe and hundreds of businesses and organizations use GridGain daily in their software development projects.



In 2008 GridGain was the first Java-based grid computing middleware that was independently tested to scale linearly up to 2048 processing cores on Amazon EC2 cloud infrastructure.

Goals Beyond 2.0

With the release of GridGain 2.0 a new set of goals was developed based on feedback from GridGain's users, market analysis and internal research and development. These goals have set the direction for GridGain 3.0 design and development and can be put in five general categories:

❖ Core Functionality Improvements

GridGain had to continue to enhance its functionality in core areas such as REST interfaces, discovery and communication capabilities, extended support for 3-rd party products while retaining the maximum backward compatibility. Continuing improvements in stability under large loads in complex topologies should remain the central focus of the overall system hardening.

❖ Even Simpler Usage Model

Although simple, productive and powerful APIs behind GridGain was a marquee feature that set GridGain apart from older technologies at the beginning, GridGain Systems' research in functional programming and domain specific languages showed that even simpler model can

be developed that would introduce distributed operation right at the language level enabling a *fundamental shift* in how cloud-enabled applications can be developed.

❖ **Easier Cloud Scalability**

GridGain was one of the first Java-based grid computing middleware that worked natively on any managed infrastructure including private and public clouds - yet further improvements in discovery and communication subsystems were needed to ensure that GridGain can work out-of-the-box in the complex environment of hybrid clouds comprising of internal, external and public infrastructures.

Furthermore, along with simplified usage model the cloud topology elements must be brought onto API level allowing developers to directly manipulate cloud resources from within their business applications.

❖ **Native Data Grid**

Since its first version GridGain provided integration with number of existing in-memory data grid products such as JBoss Cache, Oracle Coherence and GigaSpaces. However, such integrations often provided end-users with incoherent and less than ideal experience further compounded by the lack of number of fundamental features such on-demand peer-to-peer class loading, advanced transactions supporting eventually consistent modes, comprehensive distributed query capabilities, and non-trivial affinity based distribution.

Next version of GridGain had to provided fully integrated and cloud enabled in-memory data grid with industry leading feature set.

❖ **Pay-Per-Usage Pricing Model**

One of the most sought after change in GridGain was the change in pricing model. In fact, many customers pointed out a clear disconnect between fixed price support subscription and pay-per-usage pricing from cloud providers.

Next version of GridGain had to have a clear pay-per-usage pricing model that would remove the inherited cost impedance between GridGain software and cloud providers.

GridGain 3.0

GridGain 3.0 is the culmination of almost 20 months of extensive research and development by GridGain Systems.

Although the product effectively doubled in size and most of its APIs have been significantly enhanced, it is largely backward compatible and in majority of cases the migration from the previous version only requires a simple recompilation. Such longevity of the design is a sound testament to GridGain core design principles.

Building up on goals set in after GridGain 2.0 release, GridGain 3.0 introduces the host of new features.

Post-Functional Java and Scala Distributed Middleware

GridGain 3.0 is the industry first JVM-based post-functional distributed middleware that combined traditional object-oriented programming approach with comprehensive functional programming support. The result - *is a powerful, flexible and highly expressive APIs that are simple and productive to use.*

To support functional programming GridGain 3.0 introduced two new features:

- ❖ *Java-based functional programming framework* built from the ground up in GridGain 3.0 providing its users with the most comprehensive functional programming capabilities for Java programming language.
- ❖ *ScalaR - Scala-based internal Domain Specific Language (DSL)* built on top of GridGain 3.0 Java-based functional core allowing Scala developers native access to GridGain functionality.

With support for functional programming in Java and availability of internal DSL for Scala, GridGain 3.0 becomes the only distributed middleware that is native to both Java and Scala programming languages.

New Java and Scala functional APIs (additionally to object-oriented and AOP-based) enable developers to significantly simplify many often used distributed operations - yet gain in readability and expressiveness of their business logic build on top of GridGain 3.0.

Consider the following example of the method that calculates the number of non-whitespace characters in a string.

In Java:

```
public int length(String msg) {
    return msg.replaceAll("\\s", "").length();
}
```

In Scala:

```
def length(msg: String) = msg.replaceAll("\\s", "").length
```

In order to grid-enable it (for the sake of this example) we would split input string into substrings separated by whitespaces, distribute the calculation of length for each string to the remote nodes and finally aggregate all sub-lengths from remote nodes to a final value of non-whitespace length for the input string.

Here is how this could be coded in Java using GridGain 2.0 AOP-based approach:

```
@Gridify(taskClass = Task.class)
public int length(String msg) {
    return msg.replaceAll("\\s", "").length();
}

class Task extends GridifyTaskSplitAdapter<Integer> {
    @Override
    protected Collection<? extends GridJob> split(int gridSize,
        GridifyArgument arg) throws GridException {
        String msg = (String)arg.getMethodParameters()[0];
    }
}
```

```

String[] words = msg.split("\\w", 0);
Collection<GridJob> jobs = new ArrayList<GridJob>();
for (final String word : words) {
    jobs.add(new GridJobAdapter() {
        @Override
        public Object execute() throws GridException {
            return word.length();
        }
    });
}
return jobs;
}

@Override
public Integer reduce(List<GridJobResult> results) throws GridException {
    int length = 0;
    for (GridJobResult res : results) {
        length += (Integer)res.getData();
    }
    return length;
}
}

```

The code above creates a grid task that “knows” how to split and aggregate the original method that is marked with `@Gridify` annotation to link it with its task.

The boilerplate code for such task, however, can in most cases be eliminated since the actual logic consists of just two simple functions:

- ❖ One that produces a collection of closures based on the input string,
- ❖ and second one that sums up collection of integers.

With GridGain 3.0 this problem can be solved more effectively using functional APIs. Instead of AOP we will simply modify the *original code to work in a distributed fashion*. This usage paradigm is central to new GridGain 3.0 functional design.

In Java (using GridGain 3.0 functional APIs):

```

public int length(String msg) {
    return G.grid().reduce(SPREAD, F.yield(msg.split("\\s", 0),
        F.<String, Integer>c1("length")), F.sumIntReducer());
}

```

In Scala (using GridGain 3.0 and ScalaR DSL):

```

def length(msg: String) = scalar !!< (
    for (w < msg.split("\\s", 0)) yield () => w.length(),
    (s: Seq[Int]) => (0 /: s)(_+_))
)

```

As seen in these examples the simplification is dramatic. And it is not only cosmetic - it highlights the actual business logic leaving the distribution plumbing completely behind the scene.

Another important point is that distribution logic is brought onto language level in Java and even more so in Scala. In Java, closures and typedefs significantly reduce the boilerplate code, and in Scala the distributed operations brought completely on an operator syntax level via

internal Scala DSL making complex distributed cloud operations no different syntactically or semantically from the standard Scala code.

Fully Integrated Middleware: Compute + Data + Cloud

GridGain 3.0 is the first version of GridGain software that delivers on its original vision: *to provide highly integrated, cohesive and easy to use distributed middleware that combines computational grid, data grids and auto-scaling on any managed infrastructure.*

Additionally to the industry leading computational grid, GridGain 3.0 introduces two new features that amount to the most significant technical enhancements in GridGain 3.0:

- ❖ State of the art in-memory data grid (a.k.a. distributed caching)
- ❖ Advanced API-level control of cloud operations

GridGain 3.0 Data Grid subsystem is fully integrated into the core of GridGain and is built on top of the existing functionality such as pluggable auto-discovery, communication, and marshaling, peer-to-peer on demand class loading, and support for functional programming. Among its key features are:

- ❖ Expiration policies (LFU, LRU, time-based)
- ❖ Named caches
- ❖ Read-through and write-through logic with pluggable cache store
- ❖ Synchronous and asynchronous cache operations
- ❖ Pluggable data overflow storage via new swap space SPI
- ❖ Pessimistic, optimistic and eventually consistent transactions
- ❖ JTA/JCA integration
- ❖ Data replication and data invalidation in synchronous and asynchronous modes
- ❖ Partitioned cache with active replicas
- ❖ Advanced distributed query capability including SQL based, Lucene based, H2 text based and predicate based scanning with support for pagination, local and remote filtering, transformation and reduction
- ❖ Full integration for compute grid for non-trivial affinity based routing
- ❖ Functional and object-oriented APIs

GridGain 3.0 is also the first data grid featuring *zero-deployment capability* enabling users to simply bring up default GridGain nodes online and they immediately become part of the data grid topology and can store any user objects without any need for explicit deployment of user's classes.

GridGain 3.0 also introduces advanced API-level control of cloud operations. Anchored by a new cloud SPI with three out-of-the-box implementations for EC2, RackSpace and in-memory cloud, it enables developers to control any managed infrastructure right from the code of their applications largely removing any need for 3rd party cloud management solutions to perform such operations as:

- ❖ Starting, stopping and managing virtual instances

- ❖ Querying cloud resources such as images, storage devices, network quotas, etc.
- ❖ Changing virtual instance profile (where supported)

In hybrid cloud deployments GridGain 3.0 provides transparent topology and unified view with a single API on the entire virtual cloud comprised of any number of physical cloud providers. This unification greatly simplifies auto-scaling capabilities of business application built with GridGain.

GridGain 3.0 also provides cloud *strategies* and *policies* that enable automated elasticity as well as automated SLA/QoS control capabilities. Both are completely pluggable and are managed by GridGain runtime.

To go along with cloud enhancements GridGain 3.0 Enterprise Edition features two built from the ground up SPI implementations for discovery and communication. These new implementations are designed specifically to work in a large hybrid cloud environment with one-directional connections, and through-cloud routing capabilities.

These new discovery and communication implementations support following non-trivial hybrid deployments:

- ❖ Multiple private/public clouds with no direct connectivity
- ❖ Multiple private/public cloud with out-connectivity or in-connectivity only
- ❖ Geographically distributed hybrid cloud
- ❖ Single cloud with WAN/LAN/VPN connectivity between nodes

GridGain Visor

GridGain 3.0 Enterprise Edition introduces GridGain Visor - a pluggable and scriptable command line management and monitoring tool. Some of its key features are:

- ❖ Allows to “script” various operations on GridGain deployment
- ❖ Interactive and command modes
- ❖ Fully extensible via user defined pluggable commands
- ❖ Seamless connectivity to the running GridGain deployment
- ❖ Some of the available out-of-the-box commands:
 - ❖ Review and monitor topology
 - ❖ Monitor status
 - ❖ Get statistics
 - ❖ Query data grid
 - ❖ Query distributed events

Enterprise and Community Editions

One of the significant changes in GridGain 3.0 is introduction of two editions of GridGain software:

- ❖ GridGain 3.0 Enterprise Edition

❖ GridGain 3.0 Community Edition

Enterprise edition is built on top of the Community Edition and adds host of the additional features. While Community Edition remains open sourced and free, the Enterprise Edition is available on commercial base only.

Following table shows GridGain versions and their respective licenses:

GridGain Version	License	Open Source?
1.0 (released 2007)	LGPL	Yes
2.0 (released 2009)	LGPL + Apache 2.0	Yes
3.0 Community Edition	GPLv3	Yes
3.0 Enterprise Edition	Commercial	No

GridGain Systems will continue offer Academic and OEM licenses for both Community and Enterprise Editions.

Pay-Per-Usage Distributed Middleware

With introduction of Enterprise and Community Edition, GridGain 3.0 also introduces a new pay-per-usage pricing model for its Enterprise Edition. GridGain 3.0 becomes the first distributed middleware that works on any managed infrastructure - from a single Android device to a thousands of nodes in the cloud - that offers a single and unified pay-per-usage pricing.

GridGain 3.0 also incorporates a unique built-in *idle-detection technology* that prevents charging for an active and running node if it is idling for more than hour. Idling is defined as not performing any user operations and not storing any user data as part of the data grid.

This technology allows GridGain customers to safely over-provision yet pay for the actual use only. This removes the cost penalty for such frequent over-provisioning scenarios such as planned over-capacity for anticipated load spikes, staging and QA environments, hot standbys, scheduled builds, disaster recovery sites, etc.

Following tables provides basic detailed information (1) on GridGain 3.0 support subscription, per-pay-usage pricing (2), and enterprise-wide license:

Edition	Pay-Per-Usage USD\$/CPU/Month	Professional Support USD\$/CPU/Month (4)	Enterprise Support USD\$/CPU/Month (4)	OEM/Unlimited (3)
GridGain 2.x	Free	\$20	\$25	N/A
3.0 Community	Free	\$15	\$20	N/A

3.0 Enterprise	\$30	N/A	Included	Contact Sales
----------------	------	-----	----------	---------------

- (1) Prices are in USD and are subject to change. Consult www.gridgain.com for the latest pricing.
- (2) 1 CPU = 1 Core on multicore CPU architectures as reported by OS.
- (3) Enterprise License includes unlimited CPUs with each deployment not exceeding 5,000 CPUs.
- (4) One-time setup fee is extra. 10 business day setup period. One month free if purchased annually.

Volume support subscription discounts:

- ❖ Any number of CPUs over 100 gets automatic 10% discount on any subscription level
- ❖ Any number of CPUs over 1000 gets automatic 20% discount on any subscription level