In-Memory Data Grid

White Paper

GridGain Systems, 2013

Table of Contents:

Re-Imagining Ultimate Performance

What is In-Memory Computing?

Data volumes and ever decreasing SLAs have overwhelmed existing disk-based technologies for many operational and transactional data sets, requiring the industry to alter its perception of performance and scalability. In order to address these unprecedented data volumes and performance requirements a new solution is required.

> In-Memory Computing is characterized by using high-performance, integrated, distributed memory systems to manage and transact on large-scale data sets in real time, orders of magnitude faster than possible with traditional disk-based technologies.

With the cost of system memory dropping 30% every 12 months In-Memory Computing is rapidly becoming the first choice for a variety of workloads across all industries. In fact, In-Memory Computing paves the way to a lower TCO for data processing systems while providing an undisputed performance advantage.

In-Memory Data Grid at a Glance

What is an In-Memory Data Grid?

As traditional approaches to application architecture based on spinning disk technologies struggle to keep up with the ever expanding data volumes and velocities inherent in today's enterprise applications, a faster, scalable alternative is required, and organizations are increasingly considering an In-Memory Data Grids as the cornerstone of their next generation development efforts.

In-Memory Data Grids (IMDG) are characterized by the fact that they store all of their data in-memory as opposed to traditional Database Management Systems that utilize disk as their primary storage mechanism. By utilizing system memory rather than spinning disk, IMDGs are typically orders of magnitude faster than traditional DBMS systems.

Keeping data in memory is not the only reason why IMDGs perform significantly faster than disk-based databases. The main reason for performance difference are the actual differences in the architecture. IMDGs are specifically designed with memory-first and disk-second approach where memory is utilized as a primary storage and disk as a secondary storage for backup and persistence. Since memory is a much more limited resource than disk, IMDGs are built from ground up with a notion of horizontal scale and ability to add nodes on demand in real-time. IMDGs are designed to linearly scale to hundreds of nodes with strong semantics for data locality and affinity data routing to reduce redundant data movement.

Disk based databases, on the other hand, are designed with disk-first and memory-second architecture and are primarily optimized for disk-based, block device access. Even though disk-

based databases do employ caching, generally caching is just a thin layer holding a small portion of overall data in memory, while absolute majority of the data is still stored on disk. When data access pattern goes beyond of what little can fit in cache, disk-based system suffer from heavy paging overhead of data being brought in and out of memory. This is true for traditional SQL RDBMS as well as for vast majority of new NoSQL and NewSQL systems that are predominately disk-based.

Since disk is virtually unlimited resource when compared to memory, most disk-based systems can often hold the whole or most of the data set and are rarely designed to horizontally scale simply because there is no need for it from data storage standpoint. Such architecture makes disk based systems much less suited for parallel processing and often results in possible database overloading and thrashing.

IMDGs, however, due to linear scalability and in-memory data partitioning, can localize processing logic to the nodes where the data actually resides. This makes IMDGs much more suited for parallel processing as multiple CPUs across the cluster can be utilized to process computations that work on different data sets in parallel.

Ability to collocate computations with the data makes IMDGs much more suited for parallel processing than traditional disk-based systems.

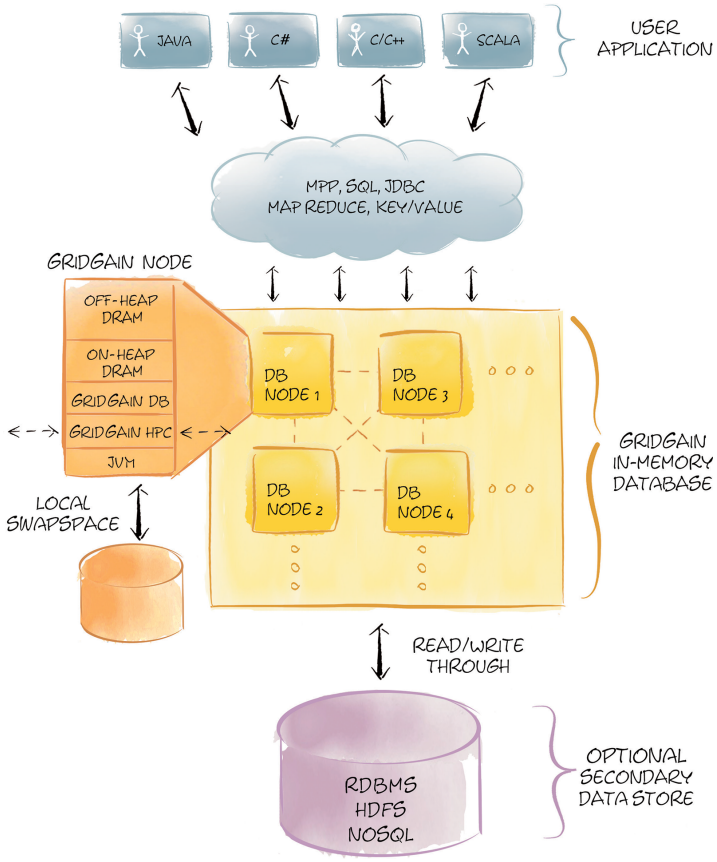
In-Memory Data Grids and Fast Data

What problems does In-Memory Data Grid solve?

Gartner defines the “three V’s” of Big Data as: Velocity, Volume and Variety. Being able to easily handle these three considerations are core to a Fast Data strategy. For example, Fast Data often encompasses the ingestion and analysis of streaming and/or transactional data (velocity). While there is certainly an ever-growing amount of data (volume) and the sources and types of data in the enterprise are expanding (variety), the key notion to understand about Fast Data is the type of the latencies at which data can be processed to produce actionable insights or machine-driven decision-making. Succinctly, the key notion that makes data “Fast” rather than just “Big” is the actionable analysis of information in near real-time.

It is helpful to consider some of representative Fast Data use cases where GridGain IIMDG is used:

Real-time Fraud Detection
The ever-increasing speed of commerce and the growing sophistication and organization of the people intent on criminal activity create an extremely difficult problem for financial institutions. Transaction processing gateways no longer simply have to deal with a high-volume of transactions, they now must be able to determine, in milliseconds, the legitimacy of a given transaction. This task is further complicated by the fact that transactions cannot be considered



in isolation, but must be considered in the context of multiple transactions as well as transaction history. As the sophistication of the people committing the fraud has increased, traditional post facto analysis has become effectively useless. By the time an alert is generated, the fraudsters have discontinued use of the credit card and have moved on to the next target. With a Fast Data solution, transactions can be analyzed in real-time for suspicious patterns as compared against historical purchase history to detect fraud in real time and deliver a decision on the legitimacy of a transaction in real time.

Biometrics And Border Security

Border security is a growing concern in today's global climate. The ability for front-line border security personnel to have at their disposal the most up-to-date information and risk models is critical. With a Fast Data solution, things such as passport scans and biometric information for every incoming passenger presenting themselves at a port of entry can be quickly processed against vast data sets as well as run through complex risk models to identify potential security risks before entry is granted.

Financial Risk Analytics

The speed with which market data can be ingested and more importantly analyzed against open positions in a given portfolio is a key component to a given firm's competitiveness and ultimately profitability. As market data ages, it becomes less useful for driving trading decisions leaving the slower firm at a disadvantage. Consequently, data must be ingested and risk calculations performed as quickly as possible. With a Fast Data solution, the streaming data can be consumed as fast as it is available from the information provider and risk calculations can be performed in parallel in near real-time. Furthermore, more complex modeling can be done with no drop in latencies simply by adding additional nodes.

The commonalities across these use cases are apparent in that they all share a low-latency SLA and require complex analytics to be performed across a high volume of data. In the following section GridGain's IMDG is compared to other solutions.

GridGain In-Memory Data Grid vs. Other Solutions

What makes GridGain In-Memory Data Grid a unique solution?

In-Memory data grids have been around for a while, but recently the availability of comparatively inexpensive memory has triggered development of a variety of new in-memory data management platforms.

One of distinguishing features for most IMDGs is the way data replication is supported. There are several common approaches here ranging from no replication at all to full replication where every node sees absolutely identical data set. Perhaps the most interesting approach is partitioned approach where each node gets a chunk of data it is responsible for. Systems that support this approach are usually most scalable as with addition of more nodes in the cluster the more data can be stored in memory. However, what makes an IMDG a powerful solution is not efficient support for some replication mode, but ability to use different replication modes together on different data sets within the same application, and ability to easily cross-query between these data sets in real-time.

GridGain In-Memory Data Grid product supports local, replicated, and partitioned data sets and allows to freely cross query between these data sets using standard SQL syntax.

Another important characteristic of an IMDG system is data accessibility and search features of the provided query language. Most IMDGs will provide access to the data by primary key. Much fewer systems will provide a custom expressive query language for richer data querying capabilities. Very few systems will allow to use standard SQL for data querying. Systems which allow for execution of distributed joins on the data are usually the most rare. Ability to execute standard SQL joins in distributed environment is usually one of the hardest features to support, but when done right, provides most elegant and efficient way for accessing data.

GridGain In-Memory Data Grid product supports standard SQL for querying in-memory data including support for distributed SQL joins.

GridGain In-Memory Data Grid Features And Key Concepts

What are the key features and concepts of GridGain In-Memory Data Grid?

The goal of In-Memory Data Grid is to provide extremely high availability of data by keeping it in memory and in highly distributed (i.e. parallelized) fashion. GridGain In-Memory 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, marshaling, on-demand class loading, and support for functional programming.

Local, Replicated and Partition Distribution Modes

GridGain provides 3 different modes of cache operation: Local, Replicated, and Partitioned:

Local Mode

Local mode is the most light weight mode of cache operation, as no data is distributed to other cache nodes. It is ideal for scenarios where data is either read-only, or can be periodically refreshed at some expiration frequency. It also works very well with read-through behavior where data is loaded from persistent storage on misses. Other than distribution, local caches still have all the features of a distributed cache, such as automatic data eviction, expiration, disk swapping, data querying, and transactions.

Replicated Mode

In REPLICATED mode all data is replicated to every node in the grid. This cache mode provides the utmost availability of data as it is available on every node. However, in this mode every data update must be propagated to all other nodes which can have an impact on performance and scalability.

As the same data is stored on all grid nodes, the size of a replicated cache is limited by the amount of memory available on the node with the smallest amount of RAM. This means that no matter how many grid nodes you have in your data grid, the maximum amount of data you can cache does not change.

This mode is ideal for scenarios where cache reads are a lot more frequent than cache writes, and data availability is the most important criteria for your use case.

Partitioned Mode

Partitioned mode is the most scalable distributed cache mode. In this mode the overall data set is divided equally into partitions and all partitions are split equally between participating nodes, essentially creating one huge distributed memory for caching data. This approach allows you to store as much data as can be fit in the total memory available across all nodes. Essentially, the more nodes you have, the more data you can cache.

Unlike Replicated mode, where updates are expensive because every node in the grid needs to be updated, with Partitioned mode, updates become cheap because only one primary node (and optionally 1 or more backup nodes) need to be updated for every key. However, reads become somewhat more expensive because only certain nodes have the data cached. In order to avoid extra data movement, it is important to always access the data exactly on the node that has that data cached. This approach is called affinity colocation and is strongly recommended when working with partitioned caches.

Off-heap Memory

Off-Heap overflow provides a mechanism in grid by which grid cache or any other component can store data outside of JVM heap (i.e. in off-heap memory). By allocating data off-heap, JVM GC does not know about it and hence does not slow down. In fact you can start your Java application with a relatively small heap, e.g. below 512M, and then let GridGain utilize 100s of

Gigabytes of memory as off-heap data cache. One of the distinguishing characteristics of GridGain off-heap memory is that the on-heap memory foot print is constant and does not grow as the size of off-heap data grows.

GridGain provides a tiered approach to memory by allowing data to migrate between On-Heap, Off-Heap and Swap storages. By doing so, GridGain avoids the well known issues with JVM Garbage Collection (GC) when attempting to utilize large on-heap memory configurations.

Traditionally, GC pauses were mitigated by starting multiple JVMs on a single server. However, this is less than ideal for applications that need to co-locate large amounts of data in a single JVM for low latency processing requirements.

Distributed ACID Transactions

GridGain In-Memory Data Grid supports Distributed ACID Transactions that use a two-phase commit (2PC) protocol to ensure data consistency within the cluster. 2PC protocol in GridGain has been significantly optimized to ensure minimal network chattiness and lock-free concurrency.

Distributed data grid transactions in GridGain span data on local as well as remote nodes. While automatic enlisting into JEE/JTA transactions is supported, GridGain Im-Memory DBMS also allows users to create more light-weight transactions which are often more convenient to use.

GridGain transactions support all ACID properties that you would expect from any transaction, including support for Optimistic and Pessimistic concurrency levels and Read-Committed, Repeatable- Read, and Serializable isolation levels. If a persistent data store is configured, then the transactions will also automatically span the data store.

HyperLocking® Technology

HyperLocking is one of ways to achieve even better performance with GridGain 2-phase-commit (2PC) transactions.

In standard 2PC approach a separate lock will be acquired, either optimistically or pessimistically, for every key within transaction. If keys are backed up on other nodes, then these locks will have to be propagated to backup nodes, sequentially, which may impose additional latencies and network overhead.

When HyperLocking is used, only one lock per transaction is acquired even though transaction may be modifying 1000s of cache entries. In pessimistic mode the lock is acquired at the beginning of transaction, and in optimistic mode it is acquired at commit phase. The requirement is that all elements within transaction are grouped by the same affinity key or partition ID.

HyperLocking can provide up to 100x performance boost over standard 2PC approach and one of the main reasons why GridGain outperforms most of the other data grid or database systems by an order of magnitude.

Multi Version Concurrency Control (MVCC)

GridGain In-Memory Data Grid utilizes an advanced and optimized implementation of MVCC (Multi Version Concurrency Control). MVCC provides practically lock-free concurrency management by maintaining multiple versions of data instead of using locks with a wide scope. Thus, MVCC in GridGain provides a backbone for high performance and overall system throughput for systems under load.

In-Memory SQL Queries

GridGain In-Memory Data Grid provides the ability to query data using standard SQL. SQL can be issued via API-based mechanisms, or via a read-only JDBC interface.

There are almost no restrictions as to which SQL syntax can be used. All inner, outer, or full joins are supported, as well as the rich set of SQL grammar and functions. The ability to join different classes of objects stored in the DBMS or across different caches makes GridGain queries a very powerful tool. All indices are usually kept in memory, either on-heap or off-heap, resulting in very low latencies for query execution.

In addition to SQL queries, GridGain also supports Text queries by utilizing Lucene text engine for efficient indexing of text data.

Data Preloading

Whenever a node joins or leaves the topology the remaining nodes will make an attempt to preload or repartition all values from the other nodes. This way, the data in the grid remains consistent and equally balanced. In the case where data is replicated to all nodes the new node will try to load the full set of data from the existing nodes. When data is partitioned across the grid, the current node will only try to load the entries for which the current node is either primary or back up.

Data preloading can be either synchronous or asynchronous. In synchronous mode distributed caches will not start until all necessary data is loaded from other available grid nodes. Essentially existing data grid nodes will keep operating as usual, but the new nodes will not be allowed to start until they have all the necessary data.

If asynchronous mode is turned on, which is default behavior, distributed caches will start immediately and will load all necessary data from other available grid nodes in the background. This mode is most efficient preloading mode and should be utilized unless application logic has strong requirements for synchronous mode.

Delayed and Manual Preloading

There are some cases when it's not efficient to start preloading right after a new node starts or an existing node leaves. For example, if you start multiple nodes one after another, you may want to wait until all nodes are started to avoid multiple preloading processes to take effect one after another for each individual node start. Or, if you plan to restart existing nodes, it's better to start pre-loading only after nodes are restarted, and not right after they are stopped.

To support such cases, GridGain provides a way to delay pre-loading. Purposely delaying preloading process often allows to significantly reduce network overhead and improve system performance.

Note that delaying preloading does not have any effect on data consistency. The data in IMDG always remains consistent, regardless of when preloading starts or completes.

Pluggable Persistent Store
In-memory data grids are often used in conjunction with an external persistent data store, such as a disk-based database or a file system. Whenever persistent store is configured, loading data and updating data in data store is automatically handled by the system. Moreover, persistent store plugs into IMDG transactions, so updating underlying persistent store is part of the same transaction as IMDG update, and if one fails the other fails as well.

Read-Through and Write-Through
Properly integrating with persistent stores is important whenever read-through or write-through behavior is desired. Read-through means that data will be automatically read from the persistent store whenever it is not available in cache, and write-through means that data will be automatically persisted whenever it is updated in cache.

All read-through and write-through operations will participate in overall cache transactions and will be committed or rolled back as a whole.

Refresh-Ahead
Refresh-ahead automatically reloads data from the persistent store whenever it has expired in memory. Data can expire in memory whenever its time-to-live value is passed. To prevent reloading data every time data is expired, refresh-ahead ensures that entries are always automatically re-cached whenever they are nearing expiration.

Write-Behind Caching
In a simple write-through mode each cache put and remove operation will involve a corresponding request to the storage and therefore the overall duration of the cache update might be relatively long. Additionally, an intensive cache update rate can cause an extremely high load on the underlying persistent store.

For such cases GridGain In-Memory Data Grid offers an option to perform an asynchronous storage update also known as write-behind. The key concept of this approach is to accumulate updates and then asynchronously flush them to the persistent store as a bulk operation.

In addition to obvious performance benefits, because cache writes simply become faster, this approach scales a lot better, assuming that your application can tolerate delayed persistence updates. When the number of nodes in topology grows and every node performs frequent updates, it is very easy to overload the underlying persistent store. By using a write-behind approach one can maintain a high throughput of writes in In-Memory Data Grid without bottlenecking at the persistence layer. Moreover, In-Memory Data Grid can continue operating

even if your underlying persistence store crashes or goes down. In this case the persistence queue will keep storing all the updates until the data grid comes back up.

Fault Tolerance and Data Resiliency

A common misconception about In-Memory Data Grids in general is that you lose all data in the event of a node failure. This is simply not the case if you have architected your grid topology correctly. In a partitioned scenario, you are free to configure as many back-ups as you wish. In the event of primary node failure, the grid will automatically promote the backup node to primary and data access will continue uninterrupted.

Furthermore, you can write your data through to underlying persistent storage in a variety of ways. By writing through to persistent storage, you can then ensure that data is not lost in the event of node failures.

Datacenter Replication

When working with multiple data centers it is important to make sure that if one data center goes down, another data center is fully capable of picking up its load and data. When data center replication is turned on, GridGain In-Memory Data Grid will automatically make sure that each data center is consistently backing up its data to other data centers (there can be one or more).

GridGain supports both active-active and active-passive modes for replication. In active-active mode both data centers are fully operational online and act as a backup copy of each other. In active-passive node, only one data center is active and the other data center serves only as a backup for the active data center.

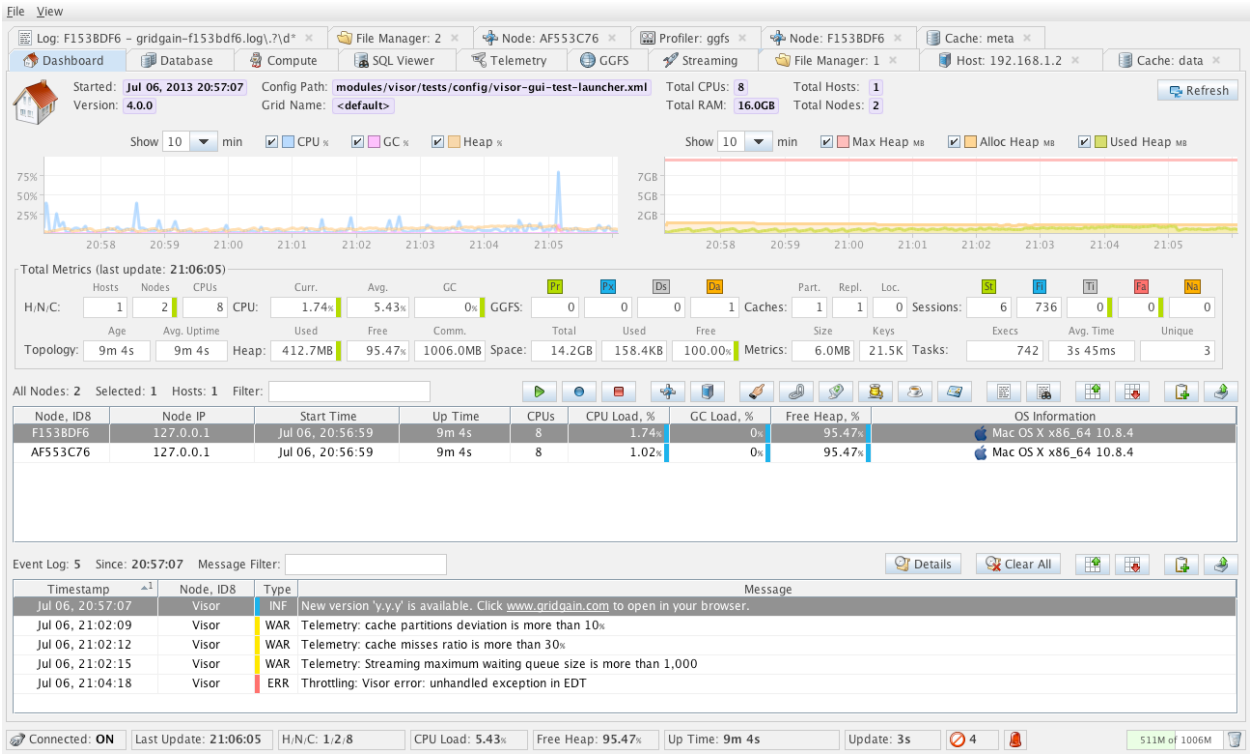
Datacenter replication can be either transactional or eventually-consistent. In transactional mode, a transaction will be considered complete only when all the data has been replicated to another datacenter. If the replication step were to fail, then the whole transaction will be rolled back on both data centers. In eventually-consistent mode, the transaction will usually complete before the replication is completed. In this mode the data is usually concurrently buffered on one data center and then gets flushed to another data center either when the buffer fills up or when a certain time period elapses. Eventually-consistent mode is generally a lot faster, but it also introduces a lag between updates on one data center and replication to another.

If one of the data centers goes offline, then another will immediately take responsibility for it. Whenever the crashed data center goes back online then it will receive all the updates it has missed from another data center.

Management

GridGain In-Memory Data Grid, as any other GridGain platform product, comes with a comprehensive and unified GUI-based management and monitoring tool called GridGain Visor. It provides deep operations, management and monitoring capabilities.

A starting point of the Visor management console is the Dashboard tab which provides an overview on grid topology, many relevant graphs and metrics, as well as event panel displaying all relevant grid events. To manage and monitor the data nodes you can select the Data Grid Tab which will display detailed information about the In-Memory IMDG. You can also manage and monitor individual caches within the data grid.



End-to-End Stack & Total Integration

What are different GridGain editions?

GridGain provides full end-to-end stack for in-memory computing: from high performance computing, streaming, and data grids to Hadoop accelerators, GridGain delivers a complete platform for low-latency, high performance computing for each and every category of payloads and data processing requirements. Total integration is further extended with a single unified management and monitoring console.

Platform Products

GridGain's platform products are designed to provide uncompromised performance by providing developers with a comprehensive set of APIs. Developed for the most demanding use

cases, including sub-millisecond SLAs, platform products allow to programmatically fine-tune large and super-large topologies with hundreds to thousands of nodes.

In-Memory HPC	Highly scalable distributed framework for parallel High Performance Computing (HPC).
In-Memory Data Grid	Natively distributed, ACID transactional, SQL and MapReduce based, in-memory object key-value store.
In-Memory Streaming	Massively distributed CEP and Stream Processing system with workflow and windowing support.

GridGain Foundation Layer

What are the common components across all GridGain editions?

GridGain foundation layer is a set of components shared across all GridGain products and editions. It provides a common set of functionality available to the end user such clustering, high performance distributed messaging, zero-deployment, security, etc. These components server as an extensive foundation layer for all products designed by GridGain.

Hyper Clustering®
GridGain provides one of the most sophisticated clustering technologies on Java Virtual Machine (JVM) based on its Hyper Clustering® technology. The ability to connect and manage a heterogenous set of computing devices is at the core GridGain's distributed processing capabilities.

Clustering capabilities are fully exposed to the end user. The developers have full control with the following advanced features:

- Pluggable cluster topology management and various consistency strategies
- Pluggable automatic discovery on LAN, WAN, and AWS
- Pluggable "split-brain" cluster segmentation resolution
- Pluggable unicast, broadcast, and Actor-based cluster-wide message exchange
- Pluggable event storage
- Cluster-aware versioning
- Support for complex leader election algorithms
- On-demand and direct deployment
- Support for virtual clusters and node groupings

Zero Deployment®
The zero deployment feature means that you don't have to deploy anything on the grid – all code together with resources gets deployed automatically. This feature is especially useful during development as it removes lengthy Ant or Maven rebuild routines or copying of ZIP/JAR files. The philosophy is very simple: write your code, hit a run button in the IDE or text editor of your choice and the code will be automatically be deployed on all running grid nodes. Note that

you can change existing code as well, in which case old code will be undeployed and new code will be deployed while maintaining proper versioning.

Advanced Security
GridGain security component provides two levels by which security is enforced: cluster topology and client connectivity. When cluster-level security is turned on, unauthenticated nodes are not allowed to join the cluster. When client security is turned on, remote clients will not be able to connect to the grid unless they have been authenticated.

SPI Architecture And PnP Extensibility
Service Provider Interface (SPI) architecture is at the core of every GridGain product. It allows GridGain to abstract various system level implementations from their common reusable interfaces. Essentially, instead of hard coding every decision about internal implementation of the product, GridGain instead exposes a set of interfaces that define the GridGain's internal view on its various subsystem. Users then can use either provided built-in implementations or roll out their own when they need different functionality.

GridGain provides SPIs for 14 different subsystems all of which can be freely customized:

- Cluster discovery
- Cluster communication
- Deployment
- Failover
- Load balancing
- Authentication
- Task checkpoints
- Task topology resolution
- Resource collision resolution
- Event storage
- Metrics collection
- Secure session
- Swap space
- Indexing

Having ability to change the implementation of each of these subsystems provides tremendous flexibility to how GridGain can be used in a real-world environment. Instead of demanding that other software should accommodate GridGain, GridGain software blends naturally in almost any environment and integrates easily with practically any host eco-system.

Remote Connectivity
GridGain products come with a number of Remote Client APIs that allow users to remotely connect to the GridGain cluster. Remote Clients come for multiple programming languages including Java, C++, REST and .NET C#. Among many features the Remote Clients provide a rich set of functionality that can be used without a client runtime being part of the GridGain cluster: run computational tasks, access clustering features, perform affinity-aware routing of tasks, or access in-memory data grid.

Summary

The case for in-memory computing is actively winning converts. Analyst firm Gartner says that in 2012, 10% of large and medium-sized organizations had adopted in-memory computing in some capacity. By 2015, that figure will have more than tripled to 35%.

GridGain's new In-Memory Accelerator For Hadoop product extends the performance value chain to Hadoop distributions while also significantly cutting an organization's storage costs. It's flexibility, scalability and plug-n-play architecture allow for seamless integration and improved velocity of analytics and reporting.

###