GridGain 3.0

High Performance Cloud Computing

Real-Time Cloud Computing 2
Compute Grid 3
In-Memory Data Grid 4
Zero Deployment 5
SPI-Based Architecture 5
Write Once - Scale on Any PaaS or IaaS 6
Java, Scala and Groovy 6
Advanced Load Balancing 9
Pluggable Fault Tolerance 9
Monitoring with GridGain Visor 9

Overview

Since its first release in the summer of 2007 GridGain software has become the leading JVM-based distributed computing middleware with tens of thousands of downloads and over 5,000,000 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 10 seconds around the globe and hundreds of businesses and organizations use GridGain daily in their software development projects.

Usage Growth

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.

© Copyright 2005-2011 GridGain Systems, Inc.

Real-Time Cloud Computing

GridGain is a JVM-based application middleware that enables companies to easily build highly scalable **real-time compute and data intensive distributed applications** that work on any managed infrastructure - from a small local cluster, to private grid, to large private, public and hybrid clouds.

To achieve this capability GridGain developed the only middleware in the world that integrates two fundamental technologies into one cohesive product:

- Computational Grid
- In-Memory Data Grid

These two technologies are axiomatic for any real-time distributed application as they provides the means for co-located parallelization of processing and data access - the cornerstone capability for enabling flat linear scalability under extreme high loads.

Having these two technologies for the first time fully integrated into one product eliminates the integration cost as well as dramatically reduces the learning curve of different APIs, different configurations, different management and monitoring solutions.

GridGain also provides many general features that make development of distributed applications easier and productive such as zero provisioning and zero deployment model, support for AOP and functional programming, streaming MapReduce processing, and integration with wide variety of 3rd party projects.

With GridGain business applications can:

- Work in a zero-deployment mode
- Scale up or down based on demand
- Cache distributed data in data grid
- Colocate data and computations for best utilization of resources
- Run sql and text queries against cached data
- Speed up long running task using MapReduce
- Use distributed thread pools
- Evenly distribute the workload on the grid
- Effectively exchange messages
- Auto-discover all grid resources
- Execute closures on the grid
- Grid-enable existing Java and Scala code

GridGain natively supports three most widely used JVM languages:

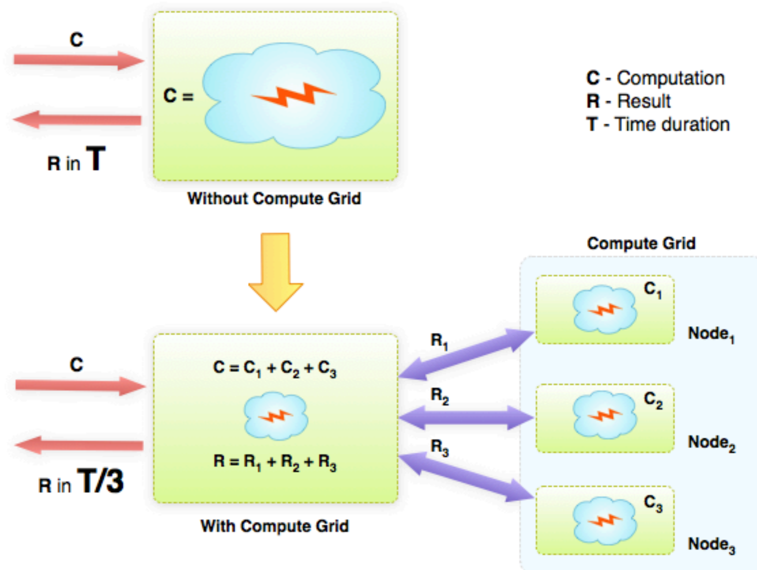
- Java 6
- Groovy 1.8 and Groovy++
- Scala 2.9

© Copyright 2005-2011 GridGain Systems, Inc.

Compute Grid

Computational grid technology provides means for distribution of processing logic, i.e. parallelization of computations on more than on computer.

More specifically, computational grids or MapReduce type of processing defines the method of splitting original computational task into multiple sub-tasks, executing these



sub-tasks in parallel on any managed infrastructure and aggregating (reducing) results back to one final result.

GridGain provides the most comprehensive computational grid and MapReduce capabilities on the market today:

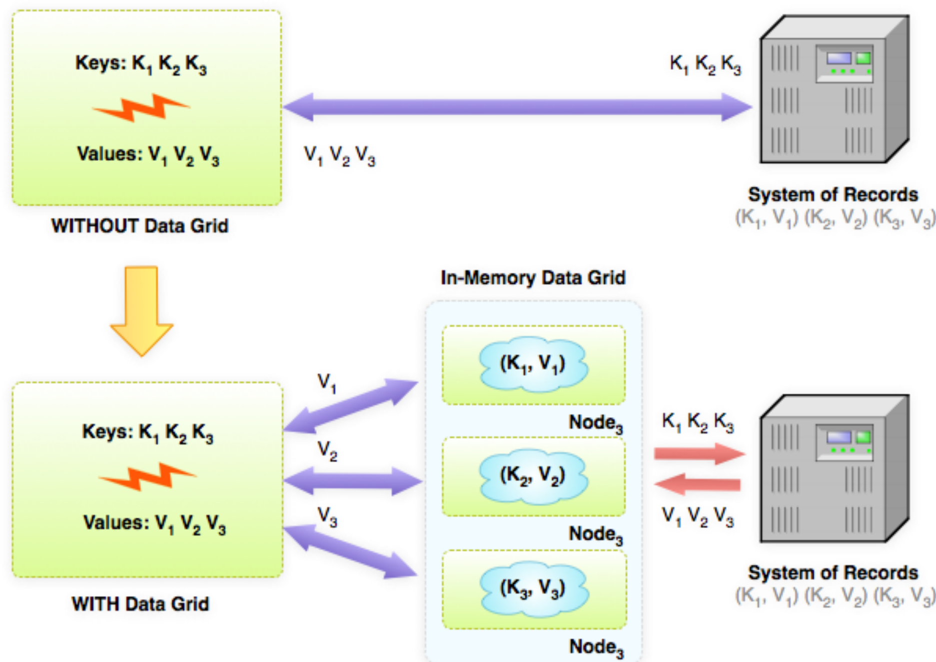
- Direct API for split and aggregation
- Fully dynamic and adaptive MapReduce
- Pluggable failover, topology and collision resolutions
- Distributed task session/context
- Distributed continuations
- Distribution recursive split
- Support for streaming MapReduce
- Node-local cache
- AOP, OOP/FP-based, synch/asynch execution modes
- Support for direct closure distribution in Java, Scala and Groovy
- Cron-based scheduling
- Direct redundant mapping support
- Zero deployment with P2P class loading

© Copyright 2005-2011 GridGain Systems, Inc.

- Partial asynchronous reduction
- Direct support for weighted and adaptive mapping
- State checkpoints for long running tasks
- Early and late load balancing
- Affinity routing with data grid

In-Memory Data Grid

In-Memory Data Grids (IMDGs) provide capability to parallelize the data storage by storing partitioned data in memory closer to application. IMDGs allow to treat grids and clouds as a single virtualized memory bank that smartly partitions data among the participating computers' memory while providing various caching and accessing strategies.



The goal of IMDGs 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 as well as tightly integrated with Compute Grid. It 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:

- Local, full replicable and partitioned cache types

© Copyright 2005-2011 GridGain Systems, Inc.

- Pluggable expiration policies (LRU, LIRS, random, time-based)
- Named caches
- Read-through and write-through logic with pluggable cache store
- Synchronous and asynchronous cache operations
- Pluggable data overflow storage via swap space SPI
- PESSIMISTIC, OPTIMISTIC and EVENTUALLY_CONSISTENT transactions
- READ_COMMITTED, REPEATABLE_READ, SERIALIZABLE isolation levels
- JTA/JCA integration
- Data replication and data invalidation modes in synchronous and asynchronous modes
- Partitioned cache with configurable active replicas (a.k.a active backups)
- Advanced distributed query capability
  - SQL based, Lucene based, H2 text, and predicate based queries
  - Result for pagination
  - Local and remote filtering, transformation and reduction
- Full integration for compute grid for dynamic affinity routing
- OOP and FP-based 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.

Zero Deployment

GridGain is the first fully distributed application middleware providing zero deployment capability where all necessary classes and resources are P2P loaded on demand. GridGain further provides 4 different modes of peer-to-peer deployment supporting the most complex deployment environments like custom class loaders or WAR/EAR files:

- PRIVATE - the most restrictive "share-nothing" deployment mode
- ISOLATED - only resources within the same tasks will be shared
- SHARED - default deployment mode; less restrictive than ISOLATED
- CONTINUOUS - similar to SHARED but no classes undeployment after master node left

Zero P2P deployment technology enables users to simply bring up default GridGain nodes online and they immediately become part of the data and compute grid topology and can store any user objects or perform any user tasks without any need for explicit deployment of user's classes or resources.

SPI-Based Architecture

Service Provider Interface (SPI)-based architecture is at the core of configuration and customization capabilities of GridGain. Based on the same technological principles as OSGi, GridGain exposes all major functional areas of its infrastructure via SPI allowing developers to customize practically every aspect of GridGain functionality from

© Copyright 2005-2011 GridGain Systems, Inc.

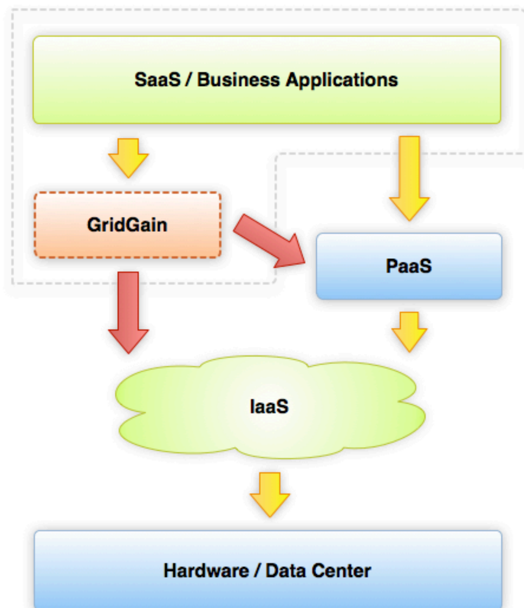
communication between nodes, auto-discovery and topology management to deployment, load balancing and collision resolution.

Unified configuration of SPIs enables LEGO-like approach to assembling your GridGain runtime with specific set of SPI implementations. In the same time compute and in-memory data grids applications (the actual business logic that is running on the grid) are kept unaffected by different SPIs running underneath.

Write Once - Scale on Any PaaS or IaaS

One of the key characteristics of GridGain architecture is its independence from any specific IaaS or PaaS giving its users a freedom and flexibility to scale on any infrastructure or deployment platform.

GridGain is an application middleware and is located on the stack closest to the



application layer. GridGain can work with any PaaS or directly with any IaaS or both in the same time. This gives a tremendous flexibility to how GridGain can be used in development, testing and production deployment.

Java, Scala and Groovy

GridGain is the industry first JVM-based real-time distributed middleware that provides native support for all three major JVM languages:

- Java 6
- Scala 2.9
- Groovy 1.8 and Groovy++

© Copyright 2005-2011 GridGain Systems, Inc.

While GridGain is written natively in Java it provides two Domain Specific Languages (DSLs):

- Scalar - Scala-based DSL for GridGain
- Grover - Groovy++ based DSL for GridGain

Consider the following three examples of the same application calculating Pi number in the distributed context written in Java, Scala and Groovy (fully compilable source code that works identically on one local node or on thousands of nodes in the cloud; **bold** highlights the GridGain specifics):

Using GridGain Functional Java APIs:

```
 1 public final class GridPiCalculationExample {
 2     private static final int N = 1000;
 3
 4     private static double calcPi(int start) {
 5         double acc = 0.0;
 6
 7         for (int i = start; i < start + N; i++)
 8             acc += 4.0 * (1 - (i % 2) * 2) / (2 * i + 1);
 9
10         return acc;
11     }
12
13     public static void main(String[] args) throws GridException {
14         G.start();
15
16         Grid g = G.grid();
17
18         try {
19             System.out.println("Pi estimate: " +
20                 g.reduce(SPREAD, F.yield(F.range(0, g.size()),
21                     new C1<Integer, Double>() {
22                     @Override public Double apply(Integer i) {
23                         return calcPi(i * N);
24                     }
25                 }), F.sumDoubleReducer())
26             );
27         }
28         finally {
29             G.stop(true);
30         }
31     }
32 }
```

Using Scalar - Scala-based DSL for GridGain:

```
 1 object ScalarPiCalculationExample {
 2     private val N = 10000
 3
 4     def main(args: Array[String]) = scalar { g: Grid =>
 5         println("Pi estimate: " +
```


© Copyright 2005-2011 GridGain Systems, Inc.

```
 6            g @<[Double, Double] (SPREAD, for (i <- 0 until g.size()) yield () =>
 7                calcPi(i * N), _.sum)
 8        )
 9    }
10
11    def calcPi(start: Int): Double =
12        (start until (start + N)) map (i =>
13               4.0 * (1 - (i % 2) * 2) / (2 * i + 1)) sum
14 }
```

Using Grover - Groovy++ based DSL for GridGain:

```
 1 @Typed
 2 @Use(GroverProjectionCategory)
 3 class GroverPiCalculationExample {
 4     private static int N = 10000
 5
 6     static void main(String[] args) {
 7         grover { Grid g ->
 8             println("Pi estimate: " +
 9                 g.reduce$(
10                     SPREAD,
11                     (0..< g.size()).collect { { -> calcPi(it * N) } },
12                     { it.sum() }
13                 )
14             )
15         }
16     }
17
18     private static double calcPi(int start) {
19         (start..<(start + N)).inject(0) { double sum, int i ->
20             sum + (4.0 * (1 - (i % 2) * 2) / (2 * i + 1))
21         }
22     }
23 }
```

First of all, notice how similar all three examples are yet they are written in a native APIs of the host language. Notice also that all three example use FP-based approach to solving a Pi calculation problem - even with Java.

Functional APIs (additionally to OOP and AOP-based) enable developers to significantly simplify many distributed operations - yet gain in readability and expressiveness of their business logic build on top of GridGain.

Another important point is that distribution logic is brought onto language level in Java and even more so in Scala and Groovy. In Java, closures and typedefs significantly reduce the boilerplate code, and in Scala and Groovy the distributed operations brought completely on an operator syntax level via internal DSLs making complex distributed cloud operations no different syntactically or semantically from the standard Scala of Groovy code.

© Copyright 2005-2011 GridGain Systems, Inc.

Advanced Load Balancing

GridGain provides both early and late load balancing that are defined by load balancing and collision (scheduling) resolution SPIs - effectively enabling full customization of entire load balancing process. Early and late load balancing allows adapting the grid task execution to non-deterministic nature of execution on the grid.

In fact, grid environment is often heterogeneous and non-static, tasks can change their complexity profiles dynamically at runtime and external resources can affect execution of the task at any point. All these factors underscore the need for proactive load balancing during initial mapping operation as well as on destination node where jobs can be in waiting queues.

Pluggable Fault Tolerance

Failover management and resulting fault tolerance is a key property of any grid computing infrastructure. Based on its SPI-based architecture GridGain provides totally pluggable failover logic with several popular implementations available out-of-the-box. Unlike other grid computing frameworks GridGain allows to failover the logic and not only the data.

With grid task being the atomic unit of execution on the grid the fully customizable failover logic enables developer to choose specific policy much the same way as one would choose concurrency policy in RDBMS transactions.

This allows to fine tune how grid task reacts to the failure, for example:

- Fail entire task immediately upon failure of any of its jobs (fail-fast approach)
- Failover any failed job to other nodes until all nodes are exhausted for this job (fail-slow approach)

Monitoring with GridGain Visor

Additionally to built-in JMX instrumentation 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 management operations on GridGain deployment
- Provides fully interpreting interactive mode for calling out GridGain APIs
- Fully extensible via user defined pluggable commands
- Seamless connectivity to the running GridGain deployment

Some of the available out-of-the-box commands:

- User defined alerts
- Start and stop of remote nodes
- Review and update of licensing information

© Copyright 2005-2011 GridGain Systems, Inc.

- Review and monitor topology
- Trace executed tasks
- Monitor status
- Get various node/task/data grid statistics
- Query data grid
- Query distributed events

Visor also provides unique interactive interpreter for GridGain and Scala combination. Just like Scala REPL provides Scala developers the ability to quite test their code and immediately see the result in interactive interpreting mode - GridGain Visor does the same for GridGain-based code.

Developers can easily type in a quick distributed expression using GridGain and Scala APIs into Visor and it will be executed interactively giving the the user an instance feedback without any need for compilation or any build steps.

© Copyright 2005-2011 GridGain Systems, Inc.