Inside GigaSpaces XAP

Technical Overview and Value Proposition

Introduction

GigaSpaces eXtreme Application Platform (XAP) is an enterprise application virtualization platform that provides a solution for end-to-end scalability of the application and its data under extreme latency and load requirements. XAP is a consolidated platform that combines the GigaSpaces in-memory data grid with a fully elastic application platform for complete application scalability, from the load balancer down to the database.

XAP is the only platform that enables end-to-end scalability with a single product, and as a single-product solution, it provides the joint benefits of increased performance and cost reduction.

XAP is designed to meet the mission-critical needs of a wide range of businesses, with advanced monitoring and management capabilities, high-level automation of operations, cloud readiness that supports private, public, or hybrid architectures, and complete interoperability: XAP provides a solution for scalability in any environment, language, and API, without dictating a specific development framework or environment.

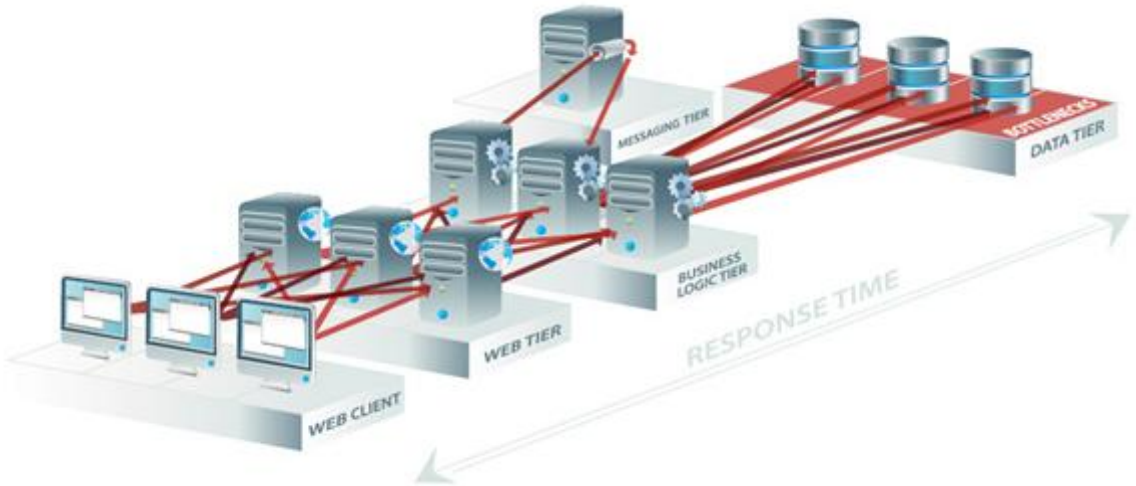
This document outlines the technical foundations of GigaSpaces XAP and the "secret sauce" behind the product's unique capabilities, the Space-Based Architecture (SBA).

The Challenge

What's Wrong Tier-Based Architectures

Most modern enterprise applications are built on top of three or four physical tiers. The first tier is the data tier, which in most applications comes in the form of a relational database. The second is the business logic tier, which implements the application's core functionality. In the JEE world, this tier is manifested using JEE application servers. In many cases, there is also a separate web tier which implements the presentation part of the application. The web tier is usually fronted by a load balancer.

Many applications also use a messaging tier to enable reliable asynchronous communication with the application and to implement event-driven processing patterns. The business logic services consume messages from the messaging tier as they arrive, and process them. To achieve high availability and higher processing capacity, all of the tiers use a clustered configuration. The result is three or four different clusters that communicate with each other over the network, and as a whole, form the enterprise application.



When analyzing tier-based architecture, there are a number of apparent problems:

- **It is hard to manage** – each tier has a different clustering model that requires different expertise. This causes a number of issues:
  - **It is expensive** – organizations have to buy separate licenses for each of the components, and usually have to hire experts to install and manage each of the tiers. This is very costly. In addition, clustering some of the above components can be a non-trivial effort even for the most experienced professionals.
  - **It is hard to monitor** – tracking and monitoring so many components in a real life scenario is very challenging. You will often need a dedicated tool to be able to do that.
  - **It is hard to troubleshoot** – when something goes wrong, it is very hard to determine exactly what happened, since there are so many moving parts in this architecture.
  - **It is hard to deploy** – deploying the application and integrating all the pieces together can be very challenging as well. It takes quite an effort to make all the tiers play well with one another, let alone configure them to reliably cooperate when processing business transactions.
- **It is bound to static resources** such as physical disks and host names. This means it is very hard to install such an application in virtualized environments, because these are very dynamic in nature.
- **Throughput and latency can only go so far** – a business transaction typically goes through most if not all of the tiers before it completes. This involves many network hops between the tiers and within each tier (assuming it is clustered).

  Also, ensuring business transaction reliability involves disk writes at various points in the process. Both network and disk I/O significantly limit the scalability and increase the latency of the business transactions. The result is that **Tier-Based Architectures cannot be predictably scaled**. If the system load increases and more processing capacity is needed, adding more hardware to the mix does not guarantee more capacity. The strong reliance on disk and network I/O inherently limits the capacity of the system. Furthermore, in many cases adding more nodes to some of the tiers (such as the data tier) actually increases the latency of each operation rather than maintaining it constant, and does not increase throughput at all due to the overhead of synchronizing the cluster nodes.

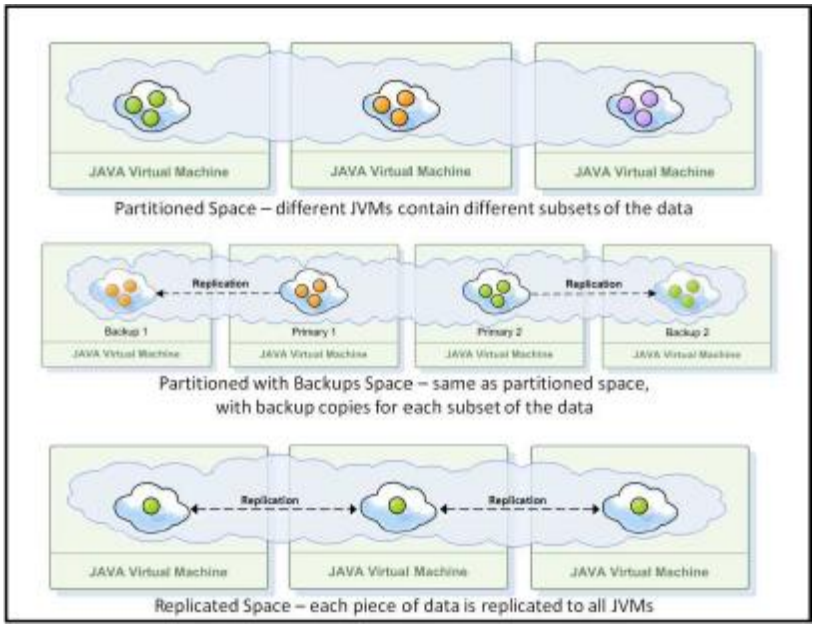
Why Caches and Data Grids Are Not Enough Anymore

The most common solution to latency and scalability issues is to front the database with an in-memory data grid, such as the one developed by GigaSpaces. This is a step in the right direction that provides partial relief and is relevant for read-mostly scenarios. Note that in many data grids, this is limited to scenarios where data is fetched, based on a unique identifier, since this is the only mode of retrieval the data grids support. Although this solution may be relevant for certain scenarios, it is not ideal for the following reasons:

- It adds yet another tier for which licenses need to be purchased. This tier also has to be developed, integrated, and managed with the other tiers, thus increasing the overall complexity of the application and making it even more costly to set up and maintain.
- It is limited to very specific read-mostly scenarios, and does not solve the latency and scalability issues of write-mostly scenarios.

Foundations of GigaSpaces XAP

The Space



At the core of GigaSpaces XAP is the GigaSpaces' best-of-breed in-memory data grid (IMDG), also known as the Space.

The Space is a scalable, high-performance, reliable data grid implementation. Its primary API is inspired by the JavaSpaces specification and the powerful tuple space model. Nevertheless, it contains much richer functionality, supporting modern paradigms like POJO-based data objects, Java 5 generics, and dependency injection, as expected from any modern data grid implementation.

The GigaSpaces IMDG supports multiple clustering topologies (partitioned, replicated, master/local, and more) and enables you to store vast amounts of data in the memory of your data grid instances, while maintaining high availability through replication to peer instances in the cluster.

Key to the application platform capabilities of XAP are the multiple ways XAP can be used:

- **Same Data, Any API** – XAP can be used from a wide variety of client APIs, including a schema-free document API, the Java Persistence Architecture (JPA), a SQL interface, memcached, in addition to the native API. What's more, all data is accessible from any other API, so you can use a document API to write data, and the JPA to read that same data.
- As a clustered in-memory **data repository**, the IMDG supports various methods to access your data. You can do so by using SQL-like queries, Query by Example, or even the limited, yet well-known Map API. You can do so from Java, .Net or C++ programs, and even transparently share data between these languages.
- As a clustered, ultra-fast, in-memory **message bus**, the IMDG allows you to register for data updates that occur in it, e.g. when a new object of a certain type is written to it, or an existing object that matches a certain query or criterion is updated. The change is propagated to your event listeners as it happens, either in a point-to-point or a publish-subscribe model.
- As a **distributed platform for running your application's code**, the IMDG supports cluster-wide execution of code. This enables easily turning the IMDG into a highly scalable processing grid. You can run your code on the entire set of machines on which the IMDG is running, on some of them, or on just one of them. Furthermore, as part of your processing code, you can access the local data stored on each machine that the code is running on at in-memory speed. If you choose to execute your code on more than one machine, you can use the IMDG's built-in support for the map/reduce design pattern to leverage the power of the entire grid.

These capabilities enable XAP to provide a full middleware stack that includes data access, messaging, and business logic execution within a single platform. This gives you the value of simplicity, and saves the need to purchase, learn, integrate, and manage multiple middleware components.

Full Support for Transactions

Another important enabler for XAP's application platform capabilities is the GigaSpaces IMDG's built-in transaction support, and the ability to operate transactionally on the grid's in-memory data. You can use fully ACD-compliant transactions to modify the Space's in-memory data, read from the Space, and receive notifications from the Space, all while still maintaining the ultra-high performance of in-memory data access. The Space transaction management mechanism supports both local transactions that are executed on a single data grid instance, and global transactions that are executed on multiple data grid instances.

The IMDG also supports XA transactions, and is able to take part in a global XA transaction

Flexible Database Integration

The Space integrates easily with all major relational databases via the JDBC API and the Hibernate ORM framework. It supports the following integration scenarios out-of-the-box:

- **Cache Warm-up:** Enables pre-loading data from the database before the IMDG becomes available to clients.
- **Cache Read-through:** Enables reading data from the database, in case the data is not present in the IMDG on read operations.
- **Cache Write-through:** Enables writing data to the database as it is written to the IMDG (naturally this is done at the speed of the database, rather than the in-memory data grid).
- **Cache Write-behind:** Unique to GigaSpaces, this feature enables asynchronous, yet reliable, change propagation from the data grid into your database, without impacting the performance of your application due to the speed of the database.

Cross-Language Support and Interoperability

Although the core Space runtime is implemented in Java, the IMDG also supports .Net and C++ out-of-the-box. This enables you to access the Space from multiple environments, from any language with Java, C++, or .Net integrations, and even run .Net and C++ code within the Space, utilizing the data grid's compute resources and co-location with the data.

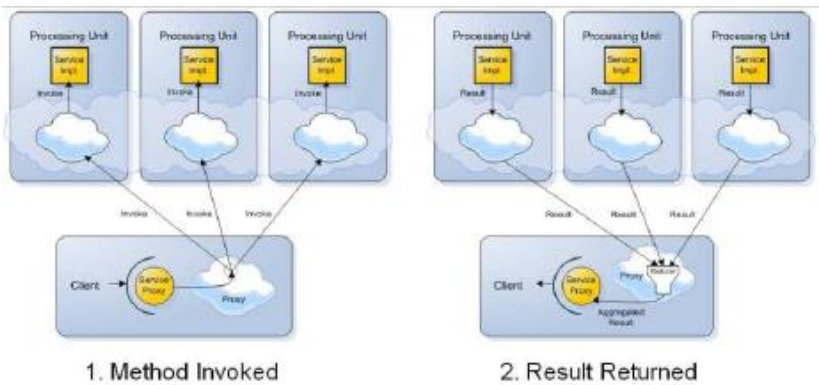
Another important aspect of this capability is the interoperability between these languages, so that one client can write an object in C++, another can read it in .Net, and another can update it in Java. The IMDG supports interoperability between all primitive types, primitive wrappers, collections (Lists, Maps) and even custom user-defined classes.

This enables you to benefit from an optimized high-performance interoperability solution which can be used to integrate various applications, or access the same application from various environments.

Beyond the Space – A Full Stack of Middleware Features

GigaSpaces XAP also provides various higher-level services on top of the GigaSpaces IMDG. These services, along with the IMDG's basic capabilities, provide the full stack of middleware features that you can use to build your application.

Event containers use the IMDG event processing and messaging APIs to abstract the code from all the low level details of handling the event (such as event registration with the IMDG, transaction initiation, etc). This has the benefit of abstracting your code from the low-level APIs, and enables it to



1. Method Invoked

2. Result Returned

focus on your business logic and the application behavior.

Space-based remoting uses the IMDG messaging and code execution capabilities to enable application clients to transparently communicate with server-side services, through an application-specific interface. Using the Space as the transport mechanism for the remote calls enables location transparency, high availability, and parallel execution of the calls without changing the client code.

Spring Integration

The IMDG programming model is integrated tightly with the popular Spring framework. This enables you to gain all the benefits of Spring, such as dependency injection, declarative transaction management, and a well defined application life cycle model.

The higher-level services (remoting and event processing) are also tightly integrated with Spring, and follow the Spring framework's proven design patterns. GigaSpaces XAP provides a set of well-defined Spring bindings, utilizing Spring's support for custom namespaces, enabling you to easily configure and wire GigaSpaces components within Spring.

Perhaps the greatest benefit of the Spring integration is the ability to isolate your application from most of the proprietary GigaSpaces APIs, and reduce the migration effort from traditional environments to GigaSpaces XAP.

Implementing Scalable Middleware Using Space-Based Architecture

Space-Based Architecture (SBA) is a term coined by GigaSpaces to describe an architectural pattern for building highly-scalable, reliable enterprise applications that solve all of the pain points in the traditional tier-based approach. SBA uses the following principles:

- **Virtualize:** Use the IMDG as the virtualized middleware layer, in which data is maintained reliably, and which also serves as a messaging bus. Application clients communicate with business logic services over the Space, using the messaging and remoting support described above. This is done using the Space's well-defined Spring abstractions, which isolate most of your code from the proprietary GigaSpaces API and enable you to focus on the business logic.
- **Co-locate:** Put the business logic and the IMDG services in the same physical address space so that all communication between the services and the Space is done in-memory and incurs no disk or network overhead, achieving unparalleled performance. This self-sufficient unit is called a "processing unit".
- **Back up:** To achieve reliability and high availability, you can create backup processing units. The primary processing unit replicates all changes made to the backup processing unit. If the primary processing unit fails, the backup takes over, and the same business services (now running on the backup) kick in, and continue to operate on the Space from the last committed point. The system automatically allocates a new instance of a backup processing unit for the newly-promoted primary.
- **Partition and scale out:** The IMDG's built-in support for partitioned topology is used to scale applications. In a partitioned topology, each processing unit is attached to a certain partition, and

works only on the data in that partition. GigaSpaces XAP provides the means to decide how the data is partitioned, enabling you to minimize or completely eliminate the need to retrieve data from other partitions when processing business transactions. To scale your application, simply add more partitions. This is merely a configuration change, and does not affect the application's code. Because the processing unit is self-sufficient, and does not depend on other components to process transactions (such as a centralized database or messaging server), the application scales predictably.

- **Put the Database where it belongs:** the last step is to use the IMDG's Write-behind to database support, to asynchronously, yet reliably, offload the result of each transaction to the database. This makes sure that data ends up in the database as before, but since the database is no longer part of the transaction's latency path, the transactions are much faster. Another side effect is that the database performance and high availability is not as critical as before, which makes it possible to use a cheaper and easier to manage database configuration.

The SLA-Driven Runtime Environment

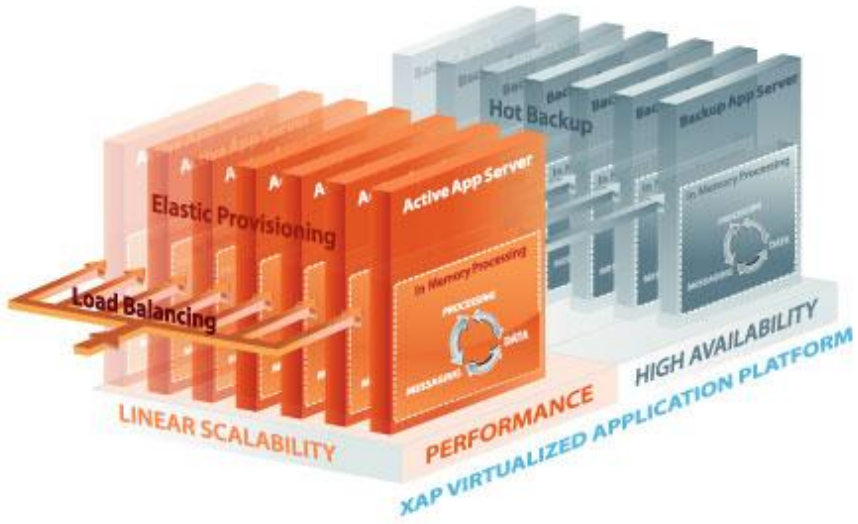
The final piece in the puzzle of Space-Based Architecture is how to deploy and run the SBA processing units. This involves a number of simple steps:

- Package the processing unit as a standard .jar file (or .war file, in case it contains a web interface) which contains all the code and definitions required to run it (IMDG configuration, number of partitions, number of backups, etc.). The processing unit is configured using a Spring-compliant XML configuration, and is in fact a Spring application with a number of GigaSpaces extensions to define elements related to the processing unit's deployment and runtime behavior.
- Deploy the processing unit using the GigaSpaces tooling (UI, CLI, etc.) into the SLA-driven runtime environment. The environment enforces the processing unit's runtime definitions on the deployment, while keeping it isolated from the actual physical runtime. That way you can move processing unit instances between hosts with a simple drag-and-drop operation. You can also add instances at the click of a button, and deploy the processing unit on any runtime environment, virtualized or not virtualized, without any configuration change. Furthermore, the deployment environment makes sure these definitions are always maintained, even in the event of a node failure. For example, it enforces that a primary and backup processing unit instance will not reside on the same node. If a certain instance fails, it fails over to its backup, but the environment automatically starts another instance as a backup on another node to make sure there is no single point of failure.

- Equally importantly, you can also define proactive behaviors to be applied to your processing unit by the environment. Here are two examples of what you can achieve:
  - By using XAP's extensive monitoring and cluster control facilities, you can monitor the CPU utilization on all the nodes in the cluster, and if a certain node becomes over-utilized, you can start another instance of your processing unit, or relocate some of the existing instances to another node. This allows you to tackle issues before they actually become issues and guarantee the system's performance under any load.
  - Your application only uses the resources it actually needs at any given point in time. We refer to this trait of the application as elasticity. For example, you can monitor the overall number of requests your application is processing. If it goes above a certain threshold, you can increase the number of processing unit instances on the fly. But you can also do the opposite – if it goes below a certain threshold, you can actually decrease the number of instances, thus freeing up unnecessary resources held by the application. This becomes very appealing when your application is running in a virtualized environment such as public clouds, where you pay per use and would like to optimize the amount of resources your application is using

Putting it All Together – GigaSpaces XAP Value Proposition

GigaSpaces XAP is built upon the following cornerstones:

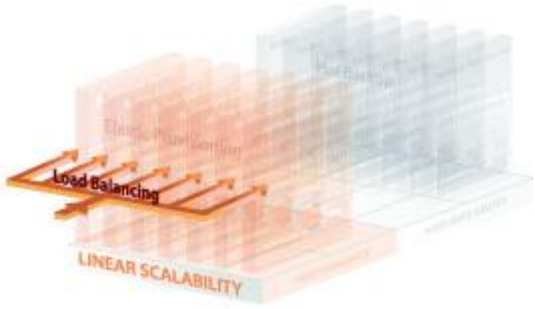


Performance, High Availability, Linear Scalability

With GigaSpaces, the entire application – from the load balancer on the front end to the database on the back end – runs completely in-memory, with all the tiers collapsed into one lightweight container. When the system needs to scale to meet increased loads, XAP dynamically expands your application onto additional physical resources – automatically and on-demand, to meet any SLA. Resiliency is guaranteed with in-memory backup within each container, and by mirroring data to a traditional database outside the runtime.

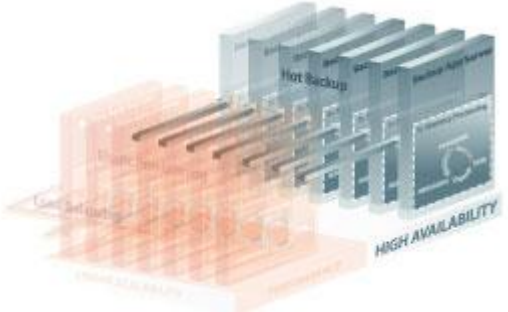
Performance

XAP solves performance bottlenecks by caching terabytes of application data close to the business logic that needs to access this data. In other words, physical I/O, database connection pool and network bandwidth, all known to be transaction delays, are taken out of the response-time equation. Application processing is done in-memory on a single platform, achieving superior performance.



Linear Scalability

XAP eliminates the endless routine of optimizing code and tuning SQL queries, which often results in only marginal improvements. Instead, XAP promises linear scalability, optimized load balancing, and smart resource utilization. XAP enables partitioning data into self-contained processing units, and provides mechanisms to elastically deploy application units to handle any load, and to dynamically allocate resources for optimized utilization



High Availability

High availability is a basic necessity for enterprise mission-critical systems, and often even a regulatory obligation. XAP guarantees zero downtime, with hot backup and automatic recovery from failure, plus multi dimensional monitoring capabilities to quickly locate operational and functional problems

XAP End-to-End Scaling Delivers the Following Business Values:

Future-Proof Your Business

Maintain consistent Quality of Service under any load – XAP scales dynamically and linearly to meet peak loads, with no need to over-provision. Smart load balancing and elastic provisioning enable scaling to meet fluctuating loads.

Ensure Business Continuity

XAP includes built-in high-availability features such as automatic self-healing, online replication, and seamless, fast failover to backup processing units to guarantee business continuity at all times.

Improve Performance

XAP solves performance bottlenecks by bringing terabytes of application data into memory and co-locating the data with the application's business logic that needs to access it.

Get Faster Time-to-Market

XAP's built -in performance, scalability, and high-availability save you expensive development and testing time, enabling your team to focus on the business rather than the infrastructure.

Reduce Costs Substantially

Dramatic reduction in hardware infrastructure, through use of blazing-fast and scalable in-memory data access, optimized resource utilization, and dynamic deployment and provisioning. All this is achieved for the most demanding, mission-critical applications on standard, commodity servers.

About GigaSpaces

GigaSpaces Technologies provides a new generation of application virtualization platforms. Our flagship product, eXtreme Application Platform (XAP), delivers end-to-end scalability across the entire stack, from the data all the way to the application. XAP is the only product that provides a complete in-memory solution on a single platform, enabling high-speed processing of extreme transactional loads. XAP was designed from the ground up to support any cloud environment – private, public, or hybrid – and offers a pain-free, evolutionary path from today's data center to the technologies of tomorrow.

Hundreds of organizations worldwide are leveraging XAP to enhance IT efficiency and performance. Among our customers are Fortune Global 500 companies, including top financial services enterprises, telecom carriers, online gaming providers, and e-commerce companies.

www.gigaspaces.com contact@gigaspaces.com