TECHNOLOGY WHITE PAPER

AZUL SYSTEMS®

Supercharging the Java Runtime

Redefining scalability and responsiveness through Java runtime innovation

This paper will review the current limitations of enterprise Java applications and the advantages of a highly innovative and elastic JVM and elastic runtime that can provide guaranteed predictability, even under load.

Executive Summary

Java-based applications are the lifeblood for many Global 1000 companies. Every day online products and services organizations rely on key platforms such as portals, eCommerce and trading systems to achieve their business objectives. But in recent years these applications, which have heavy transaction loads and data-intensive operations, have come under increased pressure to provide greater capacities with better consistency, while achieving lower TCO. Add the additional IT burden of supporting virtualization and the latest Cloud (i.e. SaaS/PaaS) technologies to improve "business agility" and it is understandable why many IT organizations are struggling to keep up with business innovation, despite choosing Java as their core application infrastructure.

Unfortunately, Java Virtual Machines (JVMs) – the very bytecode engine that gives Java its power and versatility are at the root of much of the difficulties for IT, since Java runtime innovations haven't kept pace with the latest technology trends or advancements over the last decade. Conventional Java runtimes (i.e. JVMs) are limited in a number of problematic ways that prevent applications from meeting business goals, including:

- Rigidity: fixed memory allocation size (i.e. –Xmx) for each JVM instance
- Inefficiency: proliferation of small (2-3 GB) instances used in an attempt to scale applications
- Instability: inconsistent behavior outside of a small operating range, particularly when under load
- Complexity: poor visibility and management within and across a "proliferation" of JVMs
- Topology: ill-suited for performance-centric virtualized and Cloud deployments

To remove these barriers and allow applications to cost-effectively meet business needs, a new approach is needed that frees Java execution from the limitations of conventional JVMs and the rigidities of standard Operating Systems. This paper will review the current limitations of enterprise Java applications and the advantages of a highly innovative and elastic JVM and elastic runtime that can provide guaranteed predictability, even under load.

IT Trends

Supporting New Application Dynamics

Over the past decade, large transaction processing applications with heavy resource requirements, such as eCommerce, trading systems and web portals have become mainstream due to the growth of the Internet. These business-critical applications, most based on the Java platform, face challenges that were inconceivable just a few years ago. Performance, scalability, availability, low latency and response time levels that were once needed only by the largest airline reservation systems or automated teller machine networks are now routine requirements. Accordingly, many businesses are struggling to meet new demands and support business goals.

Embracing Linux and Commodity Hardware Platforms

To support this rapid growth in transaction volumes, enterprises are growing IT infrastructure at an accelerating pace and the associated capital, administration and facilities costs are spiraling out of control. Consequently, most datacenters are pursuing replatforming on Linux and commodity hardware to increase infrastructure utilization and agility while reducing complexity. This shift moves applications from the traditional deployment model on a single large, high-capacity server to pools of physical resources.

TABLE OF CONTENTS

The latest x86 servers can now be purchased with dozens of CPU cores and hundreds of gigabytes of memory for low cost, offering compelling price/performance levels. However, with conventional Java Virtual Machines (JVMs), each application instance can utilize only a small fraction (e.g. 1/100th) of these resource-rich machines. Thus, each machine becomes host to dozens or even hundreds of small Java instances. Migrating these complex hosts requires time-consuming planning, coding and risk mitigation, limiting enterprises' ability to quickly lower overall infrastructure TCO.

Improving Time-to-Market

As enterprises seek to lower TCO and develop competitive advantage through new, innovative applications and features, their initiatives are often slowed by deployment complexities associated with managing a large number of application instances and weeks of JVM tuning needed for production deployments. Just making minor changes to existing Java applications must be carefully planned to reduce risk, since many instances must be updated at the same time and tuning errors can cause 'out of memory' crashes. As organizations struggle to realize the benefits of these powerful IT trends, most enterprises have quickly discovered that Java runtime limitations have slowed their adoption and muted potential gains.

WHY DO ENTERPRISES NEED A BETTER JVM?

- Rescue Java apps that are unreliable, slow to respond or are timing out end users
- Extend the lifespan of older Java apps that weren't designed for today's business needs
- Allow companies to enter new markets and launch new business initiatives based on Java apps with confidence
- Make it easy for developers and architects to create applications to handle modern data- and transaction-heavy systems
- Help companies get applications and features to market faster by providing a JVM that works great out of the box with little tuning
- Simplify Java deployments and help solve production issues fast

Why Doesn't Java Always Scale Well?

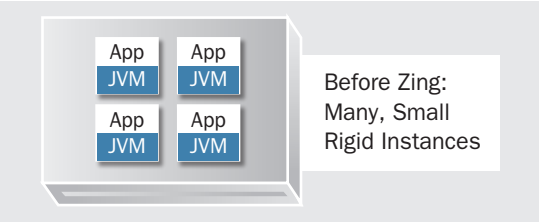
The Java Conundrum

As application needs evolve and enterprises pursue new business initiatives such as Cloud services, the Java runtime is increasingly becoming a barrier to successful deployments. Without greater Java runtime scalability and elasticity, few enterprises will be able to realize the business benefits of new innovative frameworks (e.g. Spring), advancements in commodity severs (massive core and memory) and new deployment paradigms (e.g. SaaS/PaaS).The core limitation of conventional JVMs is their inability to efficiently consume more than a few gigabytes of memory without performance penalties due to garbage collection.

Critical issues with current Java runtimes and deployment platforms include:

Rigid and non-elastic. When Java instances are deployed, they are fixed in memory size and do not change dynamically based on real-time demand or workload. This results in over-provisioning to handle peak workloads and creates inherent operational inefficiency and low resource utilization.

Unstable under load. As enterprises attempt to scale their Java applications to satisfy the continued growth of users as required by the business, they quickly discover the "fragility" of Java runtimes under load. Even though 64-bit JVMs allow larger memory heap sizes, JVM instances are not able to practically utilize more than 3-4 GBs of heap memory before applications begin to experience unacceptable behaviors caused by Java's garbage collection process (e.g. inconsistent response times). This memory size limitation causes significant architectural complexities as developers attempt to work around this basic scale problem. Because a single Java instance can't scale beyond a few GBs, more instances are added to provide application capacity, causing greater complexity and costs.



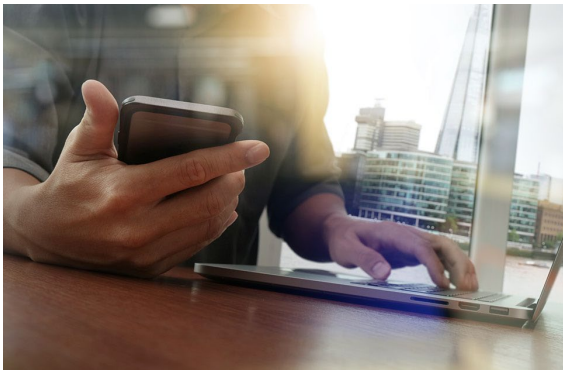
WE DO STRANGE THINGS WITHIN SERVERS NOW…

- Java runtimes "misbehave" above ~2-4 GB of memory due to garbage collection
- We use 50 JVM instances to fill up a ~$10K, ~128 GB server.
- We use distributed software solutions within a single commodity server
- The problem is in the software stack – artificial constraints on memory per instance

Solve garbage collection, and you've solved the problem

Inefficient. Commodity x86 servers continue to grow in overall capacity, with single servers providing dozens of CPU cores and hundreds of gigabytes of memory at very low prices. However, given the practical memory size limitations of individual JVMs (as discussed above), the gap between the physical server resources available and what a given Java application instance can practically use, is ever-widening. As a result, in order to achieve high server utilization, IT groups are forced to deploy hundreds of Java instances per physical server (at numbers greater than 3:1 per core), creating an overwhelming number of instances that need to be managed and maintained. The fact that Java instances are not able to dynamically share memory resources between them further exacerbates the sprawling instance and low server utilization problems.

Complex to manage. The proliferation of Java instances has created a significant burden to manage and monitor them. Large enterprises may have tens of thousands of Java instances, each of which must be individually managed, maintained, and monitored. Java runtimes are also notorious for their "black box" nature, with limited ability to debug and tune applications in production environments, significantly extending time-to-market for new applications and time-to-resolution for issues.



Mismatched for new business models. Many companies are considering new business initiatives centered on Cloud services. The ideal Cloud deployment, whether public or private, allows an application to scale to meet the needs of the business without requiring intimate knowledge of the physical server infrastructure. Cloud-based services rely on strict SLAs and quality of service (QoS) to attract and keep customers. Current Java runtimes, however, often experience significant performance degradation and throughput fluctuations when deployed on Cloud infrastructure, and require painstaking tuning in order to scale on a given server. This prevents companies from pursuing potentially lucrative opportunities, since SLA violations are costly, and launching new customers can cause response time issues or even outages.

As enterprises seek to lower TCO and develop competitive advantage through new, innovative applications, the quest for more memory-efficient and scalable Java runtimes is growing. In addition, commodity servers continue to improve in overall capacity, outpacing conventional Java runtimes' ability to efficiently consume these physical resources. With commodity servers now exceeding 48 cores and a terabyte of memory, enterprises need to rethink whether their current deployment paradigms and choice of Java runtimes to ensure they can support the needs and growth targets of the business.

Solution

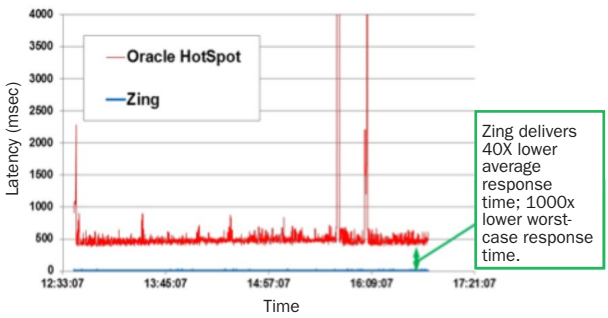
Zing: The New Foundation for Elastic, Scalable Java Deployments

Offering unprecedented levels of scalability and throughput, the Zing JVM shatters Java scalability barriers and enables existing Java applications to smoothly and reliably scale to dozens of CPU cores and hundreds of gigabytes of memory in a single instance. Optimized for Linux and x86, Zing's elastic architecture can automatically scale individual Java application instances up and down in both core count and memory size, based on real-time demands. Because Zing uses Azul's "awarding-winning" and highly innovative Pauseless Garbage Collector (a.k.a. Azul C4 - Continuously Concurrent Compacting Collector), application instances can grow to any size without variations in response times associated with conventional JVM Garbage Collectors.

Zing Components

In addition to a 100% Java-compatible JVM which installs and launches like any other commercial JDK/JVM, Zing provides a management and monitoring platform with an integrated, application-aware resource management and process controller and a true, zero-overhead, always-on production-time diagnostic and tuning tool.

Better Java Performance, Consistency and Reliability for All Types of Apps



As an example of the dramatic capabilities of Zing, consider the real-world results shown in the picture above. Zing delivered an average response time 40X lower than a tuned Oracle JVM, and a 1000X reduction in worst-case response time. Same application, unchanged code – different JVMs.

ZING BENEFITS

- Consistent performance, even under heavy load
- ~40X lower average response times and ~1,000X lower worst-case response times
- Ultra-low latency

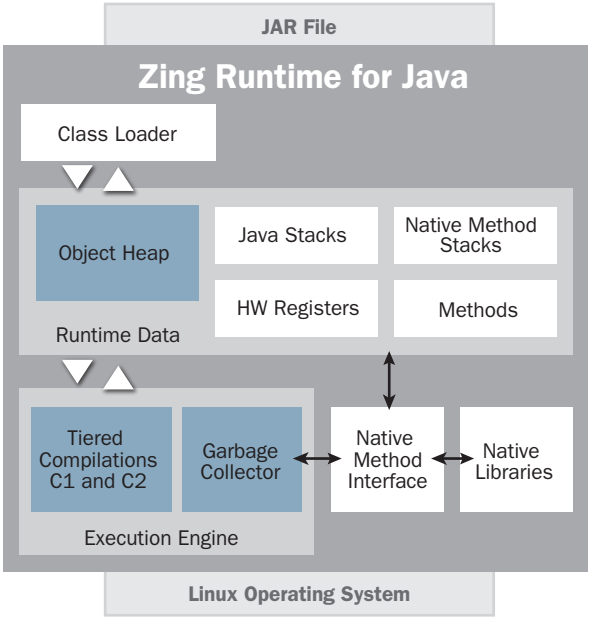
How Zing Works

Eliminating Application Hiccups

Java performance and scalability start at the JVM. Conventional JVMs are rigid and fixed in size at start up. In addition, the underlying OS or hypervisor does not work in cooperation with the JVM to provide greater elasticity as demand changes. Constrained to just a few GBs per instance, Java applications can experience varying response times as single instances experience higher user-loads or greater memory demands (such as parsing a large XML payload). As application loads and memory demands increase, these artificially small memory heap limits (usually around 2-4 GBs) can directly impact application performance and throughput, resulting in negative business impact and unmet SLAs. Because of these conventional JVM limitations, existing solutions simply do not meet the needs of today's business-critical Java applications.

In contrast, the Zing JVM utilizes Azul's proven technology to break Java scale barriers and increase throughput and consistency. The Zing JVM seamlessly replaces the existing JVM. Simply point your application or startup scripts to use the Zing JVM. This process requires no coding changes and is completely transparent to the rest of the deployment. The result is applications that achieve unmatched throughput and response time consistency with no outliers associated with GC pauses experienced with conventional Java runtimes.

Available for multiple operating systems, the Zing JVM is optimized for Linux but can also offload Java workloads from non-x86 systems such as SPARC Solaris, enabling enterprises to cost-effectively migrate applications to the latest-class of x86 servers without changing their operating system and without the conventional migration costs and risks.



Key Zing Innovations

Azul C4 (Continuously Concurrent Compacting Collector). Java garbage collection automatically frees the heap space used by objects that are no longer referenced by the application. This makes the programmer's life easier, since he or she no longer has to keep track of and free allocated memory. Automating this process reduces the number of bugs in the final program and saves developers plenty of headaches. In conventional JVMs, heap space is allocated per instance when the application is launched and is, therefore, rigidly limited at runtime. IT organizations spend hours tuning JVMs to the right amount of heap. Too much, and garbage collection pauses interrupt processing, too little and the instance is 'out-of-memory' and crashes. The Java garbage collector also has to spend time preventing heap fragmentation and ensuring room for new objects. When too much fragmentation occurs, new objects may not fit in the largest continuous space and ALL conventional collectors must now make room through a "stop the world" (i.e. pause) event that is linear with the size of the heap.

When application loads increase, Java applications typically will suffer from more frequent business-interruptive garbage collection pauses. Despite IT's continued efforts to tune away GC occurrences, "compaction is inevitable" and only occurs more frequently under load.

Zing implements the unique C4 garbage collector that utilizes concurrent compaction, enabling the application to continue executing while remapping memory. This keeps all pause times consistently low – preventing spikes and making applications predictable and consistent in response time. The C4 collector and its performance is also independent of memory heap size, so a single instance can start at 30 GB and dynamically grow because of load and variability up to 2 TB or more. This allows applications to enjoy a smooth, wide operating range insensitive to data size, concurrent sessions and throughput. Each instance can employ hundreds of GBs of heap memory, achieving unmatched scale and consistently fast response times in a simplified deployment with far fewer, larger instances.

Elasticity. With the elimination of long garbage collection pauses, application SLAs are no longer impacted by growing memory heaps. Freed from the practical restriction on heap size, Zing provides elasticity, so instances grow and shrink memory as needed. In conventional JVMs, administrators set a rigid heap size configured for the worst case scenario plus a safety factor, which leads to inefficient use of resources, since unused memory is inaccessible to other instances that could use it. With Zing, sizing doesn't have to be a "dark art". Each instance is assigned a certain amount of 'Committed' memory (i.e. -Xmx) to guarantee the live set and performance, but also has access to two pools of shared memory. 'Performance' memory is available as needed to maintain consistent response times under load, and 'Insurance' memory is accessible to instances to survive peaks and memory leaks. In operation, these pools are managed by the Zing Resource Controller, which automatically scales individual Java application instances up and down based on real-time demands, and dynamically maps and allocates resources based on configurable policies. The result is better resource utilization and reliability, as instances under load elastically grow into unused memory space and return it when it's no longer needed, preventing out of memory errors, slowdowns and crashes.

Visibility. Conventional Java runtimes are notorious for their "black box" nature. Although rich Java development tools exist, none of them can be used in production environments without significantly impacting running applications. With this limited ability to debug and tune, the time it takes for IT to resolve production-time issues significantly increases. With Zing Vision (ZVision), a true, zero-overhead diagnostic and tuning tool that is integrated into the Zing JVM, developers can now capture application metrics and performance data (at the JVM level) for the running application with no impact on operations and no performance degradation. This visibility tool offers a uniquely detailed view of all performance aspects of the application and runtime, including thread, memory, and IO profiles. ZVision provides always-on visibility into production workloads that enables fast diagnosis of issues when and where they happen.

ZING COMPONENTS

Zing Java Virtual Machine (Zing JVM). A highly scalable, full-featured, 100% Java-compatible JDK based on HotSpot optimized for Linux and x86 that transparently replaces your existing JVM

Zing Vision (ZVision). A true, zero-overhead, always-on production-time diagnostic and tuning tool instrumented into the Zing JVM

Zing Benefits

Conventional JVM	Impact	Zing JVM	Result	Benefits
Limited to <12 GB of heap memory/instance	Complex deployments of up to hundreds of small instances	Up to hundreds of GBs of heap memory/ instance	Simplified deployments with fewer, larger instances	• Scalability – 100x larger heap sizes with no GC pauses • Reliability – Fewer moving parts • Manageability – Fewer, larger instances • Consistency – response times not subject to garbage collection pauses
Limited, often intrusive monitoring tools	Difficult and time-consuming to find and fix production issues	True, zero-overhead, always-on production time diagnostics and tuning tools	Fast production issue resolution	• Reliability – Find and fix potential issues before they become downtime • Visibility – fine-grained monitoring of capacity utilization and resource usage at the instance level
Complex deployments where each instance has to be managed individually	As the deployment grows, it becomes unmanageable	Control all Java applications from a single, centralized console	Ability to share resources dynamically and map and allocate resources based on configurable policies	• Scalability – Scale elastically to accommodate sudden changes in load • Faster time to market – No need for fine-tuning of memory size • Reliability – Assign additional resources to instances to survive peaks and leaks • Manageability – Single view to all applications in the deployment

Zing Versatility

Zing provides a robust foundation for all types of Java applications. With Zing, applications elastically scale from a few cores and GBs of memory dozens of cores and hundreds of gigabytes of memory. Zing is easy to install and transparent to existing applications. Zing is ideal for:

- Large or variable numbers of concurrent users
- Microservices-based architectures
- High or variable transaction rates
- Low latency requirements
- Large data sets
- Caching, in-memory data processing
- ESBs, SOA, messaging, NoSQL, Search applications
- Multi-tenant SaaS, Platform-as-a-Service (PaaS)
- Virtualized and Cloud deployments

Zing Benefits for Common Deployment Models

Conventional JVM

- Many, rigid instances
- Complex
- Fragile
- Inefficient, with low utilization
- Inconsistent
- Hard to grow
- Limited headroom
- Costly to operate

With Zing

- Fewer, elastic instances
- Simpler
- More robust
- Efficient, with high utilization
- Responsive
- Easy to scale
- Massive shared headroom
- Lower TCO

Summary

At Azul, we have one thing in mind; ensure that customers can rely on Java-based applications for their business-critical services, grow their businesses and launch new initiatives without worrying about scalability, reliability or outsize infrastructure investments. To that end, we deliver the best JVM in the world, a very stable, superior, robust JVM with predictable high-performance and low latency capability, a JVM with all the features built in. Zing eliminates most of the typical JVM challenges and performance and scaling barriers, so your team can focus on the features and functions of your application or service, and you can focus on growing your business.

Contact Azul

To discover how the Zing JVM can make your Java deployments more scalable, more elastic, more reliable, and less complex, contact Azul today.

Azul Systems, Inc.
385 Moffett Park Drive, Suite 115
Sunnyvale, CA 94089 USA
+1.650.230.6500

www.azul.com
info@azul.com