Java CoP

Java HTTP Session Replication

A Study of Performance Impacts and Costs-benefits

Presented by the Java Community of Practice

Table of Contents

Abstract

Session Replication (SR) is often confused as the only and most critical component for High Availability. Rather, SR is one of many high availability components, and specifically SR retains user data at the Application Tier. For large-scale, JEE systems where transactional or user data is stored in sessions, the trade-off between performance, implementation costs and benefits need to be considered. For many organizations, the benefits of SR do not outweigh the investment costs, and therefore an analysis should be conducted prior to taking on the requirement. This paper explores the performance considerations, various architectures for implementing session replication and presents a case study of an organization faced with the challenge of a technical requirement of enabling session replication.

Introduction

Session Replication (SR) is a component of high availability at the Application Tier. For large-scale, JEE systems where transactional or user data is stored in sessions, the trade-off between performance, implementation costs and benefits needs to be considered. System performance is impacted by resource consumption (CPU, Memory), Java Virtual Machine (JVM) memory management processes, object serialization, and network latency; and for many organizations the benefits of session replication are diluted by performance factors. Therefore, prior to investing in the implementation of session replication, the business must analyze the associated costs and benefits.

Assumptions and Scope

- The JVM referenced in this document is Oracle JRockit R27 (64-bit)
- The Application Server referenced in this document is Oracle Weblogic 10.3 (64-bit)
- The Operating System referenced in this document is Oracle Solaris 10 (64-bit)
- The physical Host referenced in this document is the Oracle Sun SPARC T5240
- Data points presented in this document are specific to the environment in which they were collected
- Each application is developed according to various functional and technical requirements, which impacts the size of session objects

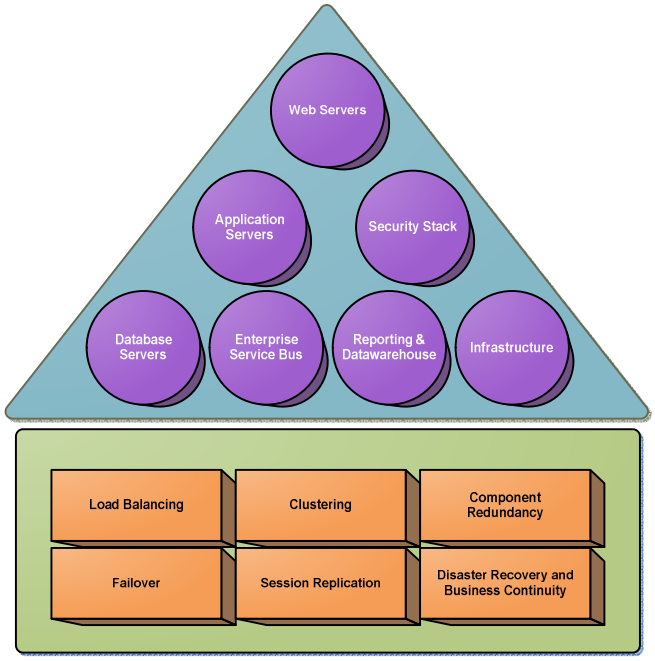
Given these assumptions, it must be understood that each system must be independently evaluated to identify performance optimization opportunities based on unique characteristics. This paper explores the performance considerations, various architectures for implementing session replication and presents a case study of an organization faced with the challenge of a technical requirement of enabling session replication.

High Availability

High Availability (HA) is a design requirement for a system to be available in continuous operation or according to agreed upon durations. HA is often described in terms of the percentage of time per year the system is available, for example 99.9% is no more than 8.76 hours of downtime per year. There are many system components or design principles that support HA, and there is often a misconception that session replication is the only, or most critical design element. The following section reviews basic high availability components and highlights the benefits each provides.

Components of High Availability

To achieve High Availability goals, a number of components must be considered within a given architecture and infrastructure. These components include, but are not limited to:



- **Load Balancing** is a methodology for distribution of workload across specific resources assigned to process the specific tasks, therefore making effective use of resources to increase throughput and minimize response times
- **Clustering** refers to the concept of grouping similar components to appear as a single processing unit
- **Component Redundancy** ensures that critical components are duplicated to increase reliability and eliminate weakest-links; includes software and hardware
- **Failover** is a fault-tolerance, system design principle which automates the action of switching between redundant components when one abnormally terminates
- **Session Replication** ensures user session data is persisted for availability during failover scenarios
- **Disaster Recovery and Business Continuity** ensures system availability in case of a major data center disaster, where high-end architectures support active, real-time data replication over high speed fiber

These foundational concepts support availability of application architecture components, including Web Servers, Application Servers, Security Servers, Databases, Enterprise Service Bus, Data Warehouse, and Hardware Infrastructure. It is important to note that each of the foundational concepts/components may impact/interfere with others and therefore must be considered together rather than in isolation.

Session Replication

HTTP Session Replication (SR) is a component of high availability (in a clustered configuration) at the Application Tier, for maintaining session state and functional continuity in case of an application server failure. As the user's session information is modified during normal operations[1], the alterations are managed by the application container[2] and can be made available to nodes in the cluster. In situations where a user's primary application server fails, the user's request is re-directed and processed by a secondary node, unbeknownst to the end-user.

Architecture Design Options

When designing a JEE application, Architects have a variety of options that can be considered to support SR. Each of these options has an impact on how the system will scale and ultimately perform when SR is enabled. Therefore, the pros and cons need to be evaluated for each specific environment.

Memory Based Persistence

With Memory Based Persistence, session objects are stored within the applications heap space. Weblogic provides two methods for maintaining session in memory – synchronous and asynchronous. Asynchronous provides the benefit of not directly affecting the transaction time, while introducing the risk of "stale" session state if a node were to fail because the container replicates the sessions only after a certain time. On the other hand, the synchronous method ensures real-time HTTP state at the expense of computing resources.

In addition to storing the replicated sessions in memory, the JVM will spend additional processing time in serialization and transfer of replicated session data between application server nodes.

File Based Persistence

With File Based Persistence, session objects are stored on the file system, and can be shared across multiple application server nodes. While the solution reduces the memory requirements for the application's heap and reduces network latency, writing to disk is significantly slower than in-memory operations. Additionally, the overhead required to synchronize access to the file store among the application cluster can become a major bottleneck.

---

[1] The HttpSession, setAttribute(…) function is the only means by which the container will replicate changes

[2] In order to support HTTP replication, objects in session (and their attributes) must be serializable; see Weblogic documentation for complete details

Database Persistence

Sessions are maintained within the Database using a Java Database Connectivity (JDBC) driver to store and retrieve sessions. Since the database is a shared resource, the session objects do not need to be serialized across application server nodes nor do the sessions need to be stored in the application's heap. Additionally, if the entire application cluster were to fail, session data would still survive. Weblogic supports Asynchronous and Synchronous methods, similar to the Memory Based Persistence.

Distributed Cache

Sessions are stored within a separate JVM, external to the application, using Oracle Coherence caching solution – Weblogic provides the capability of integrating Coherence specifically for this purpose. The cache is shared across the application cluster, therefore eliminating the need to serialize and replicate sessions across nodes. As with the Database and File Persistence methods, Distributed Cache provides the benefits of alleviating memory requirements from the application's heap.

Dedicated Hardware Solution

Oracle has designed a hardware-based solution for infrastructures that require extensive compute power, named Exalogic Elastic Cloud, which addresses a number of performance issues faced by IT organizations. The Oracle Exalogic "Elastic Cloud", which includes Coherence and Infiniband, is a recently engineered system comprised of both hardware and software components.

The Exalogic solution is built on a high bandwidth and low-latency InfiniBand I/O fabric that connects all major system components, to enable high-throughput, low latency shared storage directly attached to the I/O fabric.

Exalogic includes tuned and optimized Oracle Solaris components, and is designed to improve performance of applications which:

- Use in-memory session state replication
- Have large memory requirements

Addressing Performance Impacts

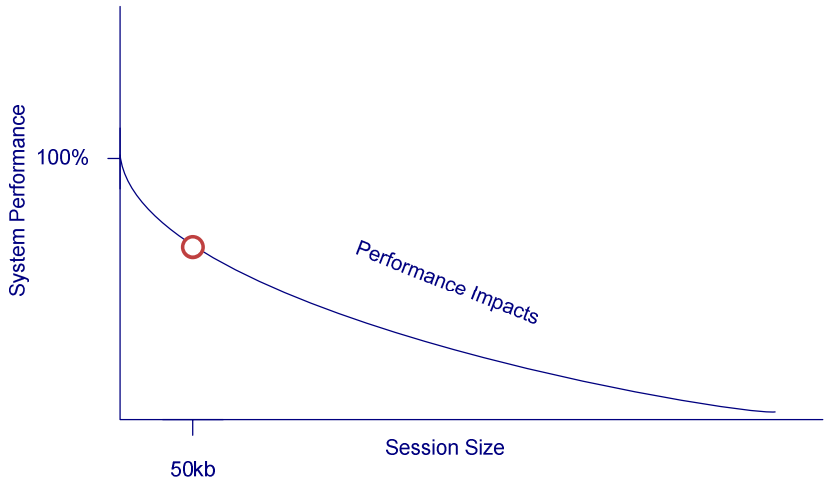
Consider the following tactics to address performance impacts session replication:

- **Reduce memory management overhead**
  - If utilizing Memory Based Persistence, appropriately size the application's JVM heap space and tune Garbage Collection to account for the additional session objects
  - Invalidate sessions when they are no longer alive rather than letting them expire
  - Offload the session objects from the main application's heap space through external storage such as DB Based Persistence, File Based Persistence, Distributed Cache, or a Dedicated Hardware Based solution
- **Reduce serialization overhead**
  - Analyze session object sizes at a global and transactional level to ensure only the most imperative data is being stored
  - Analyze code for frequency of session updates and decrease appropriately

    - Utilize a shared storage mechanism such as DB Based Persistence, File Based Persistence, Distributed Cache, or a Dedicated Hardware Based solution
- **Minimize session object size**
    - Only store what's absolutely needed in session
    - Terminate dead results from session objects (e.g. search results)

Understanding Costs and Benefits of Session Replication

While SR provides safeguards against data loss, its benefits come with performance impacts which are generally addressed through expensive investments. As session sizes increase, system performance rapidly degrades due to many factors, including resource consumption, time spent in memory management processes, serialization, network latency, system load, and frequency of modifications to session attributes – the following graphic provides a notional illustration.



In order to understand the benefits and costs associated with Session Replication, the business needs to develop an analysis model. Primary contributors to implementation costs include acquisition of new hardware, software licenses, development time, and testing time. The range of solutions will obviously vary in costs, on the low-end being simple configurations and development tasks, while advanced domain specific solutions such as the Oracle Exalogic, Exadata, and Infiniband package will add significant infrastructure costs. The following metrics provide a generic starting point for the analysis model:

Estimated Losses per Year:

- Application Server Failures Per Year
- Business Transactions Lost Per Year
- Percentage of IT Ops Time Spent Addressing Application Server Failures
- Costs Due to Damaged Reputation
- Total Cost of Application Server Failures

Estimated Investment Costs:

- Hardware Acquisition Costs
- Software Acquisition Costs
- Design, Development and Implementation Costs
- Functional QA and Performance Testing Costs
- Total Costs of Session Replication

By utilizing an agreed upon model, decision makers will be better informed prior to pursuing session replication in the application architecture. For an application that experiences relatively little downtime and little loss of mission critical data, the costs of application server failures may be minimal (e.g. < $5,000), while the costs to implement session replication will be significant (> $3M) – the benefits do not outweigh the costs in such a scenario.

Case Study

Background

A client's non-functional system requirements included Session Replication at the application tier. While stress testing the application, it was determined that enabling session replication would cause complete degradation of application performance. Various session replication options were evaluated, including Memory Based Persistence, File Based Persistence, and Database Persistence. Additionally, impacts of reduced user load and increased heap spaces were explored. This particular application is a large-scale, n-tier application that serves 58 distinct business units that are geographically separated – each unit has its own business processes, rules and data.

Application Profile

- Language - Java EE
- Lines of Code - 5 Million
- Design - N-Tier
- User Base per Hour - 3,960 Average; 6,600 Peak
- Database Size - 720 GB
- Types (Class) of Transactions – 21
- Business Units Served (Custom Application Deployments) - 58

Technical Environment

- Operating System - Oracle Solaris 10, 64bit
- Web Server - Oracle iPlanet 7
- Application Server - Oracle Weblogic 10, 64bit
- Virtualization - Oracle Zones
- Database - Oracle DB 11g, 64bit
- Load Testing - HP LoadRunner 11
- Disk Configuration – NFS v3

Performance Test Configurations

- Number of Virtual Users – 3,960
- Duration
    - Login - 10 Minutes (staggered login for the user base)
    - Warmup - 30 Minutes
    - Load Test (Measured Period) - 60 Minutes
    - Logoff - 10 Minutes
- Business Scenarios – 15
- Business Transactions – 394
- Target % of Transactions Meeting SLAs – 100%
- Target Transaction Throughput – 250k per hour
- Target Average CPU Utilization – < 70%

- Target Average Memory Utilization – < 70%

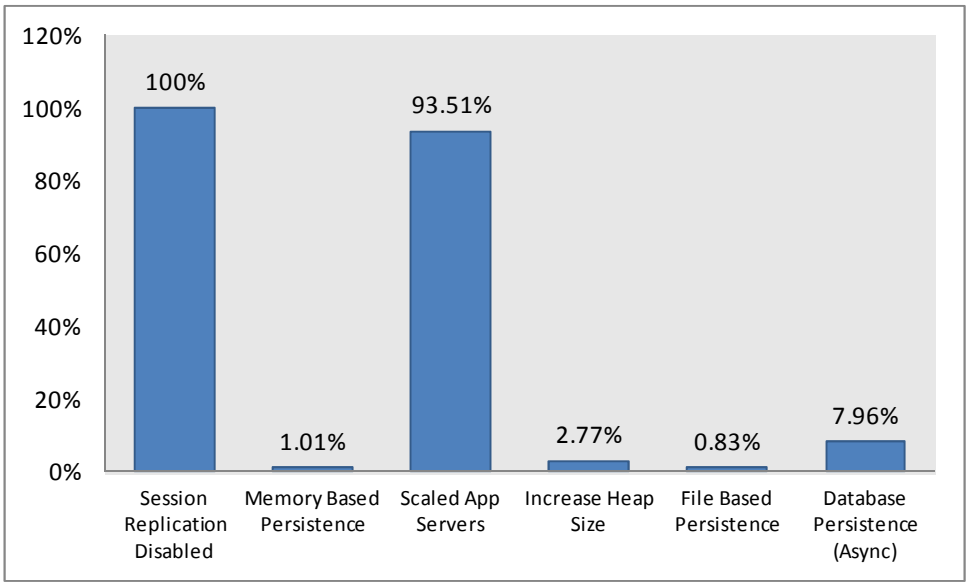
Configurations and Performance Test Scenarios

The following session replication configurations were evaluated during performance testing efforts.

Test Name	Persistence	Heap Size (GB)	User:Server Ratio	JDBC Per WL	General Observations
Session Replication Disabled	Disabled	12	660:1	300	System met SLAs and will be used as the benchmark for comparison
Memory Based Persistence	In Memory	12	660:1	300	Significant amount of processing time was spent in serialization (Code Hotspots) and memory management processes; observed unacceptable performance
Scaled Application Servers	In Memory	12	90:1	300	The system scaled well and met SLA objectives with similar performance to the benchmark; observed acceptable performance
Increased Heap Size	In Memory	32	660:1	300	Marginal increase in performance; observed unacceptable performance
File Based Persistence	File System	12	660:1	300	Increased read/write frequency and latency times to disk caused significant bottlenecks; observed unacceptable performance
Database Persistence	Database	12	660:1	300	Experienced a significant number of failed transactions and queuing of DB connections from the application tier which led to poor performance; observed unacceptable performance

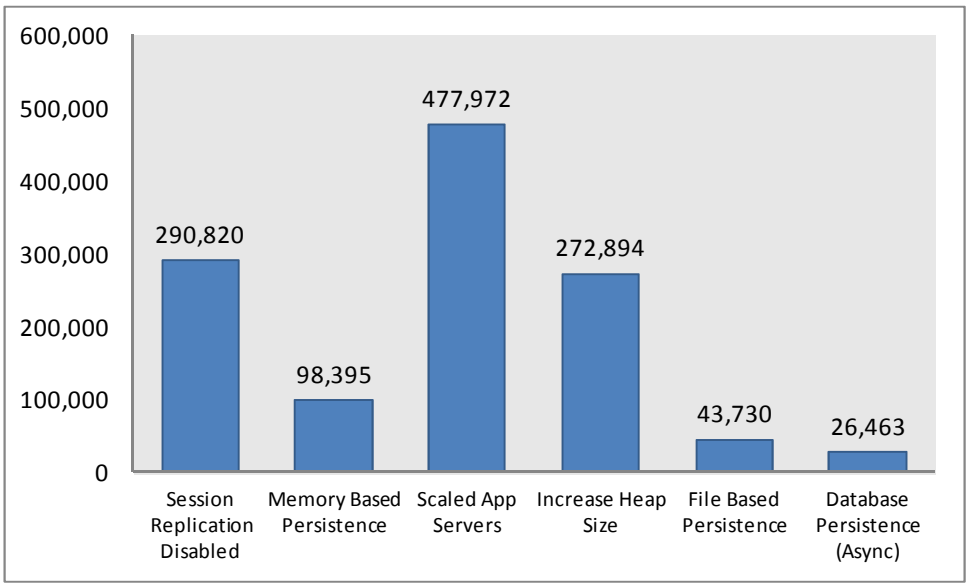
Transaction Response Times

The following graph illustrates the capability of the system to perform against defined SLAs. Transaction performance is expressed in terms of percentage of transactions that met SLAs during the performance test. The results indicated favorable performance from scaling the application servers, while all other session replication configurations could not meet targets.



Transaction Throughput

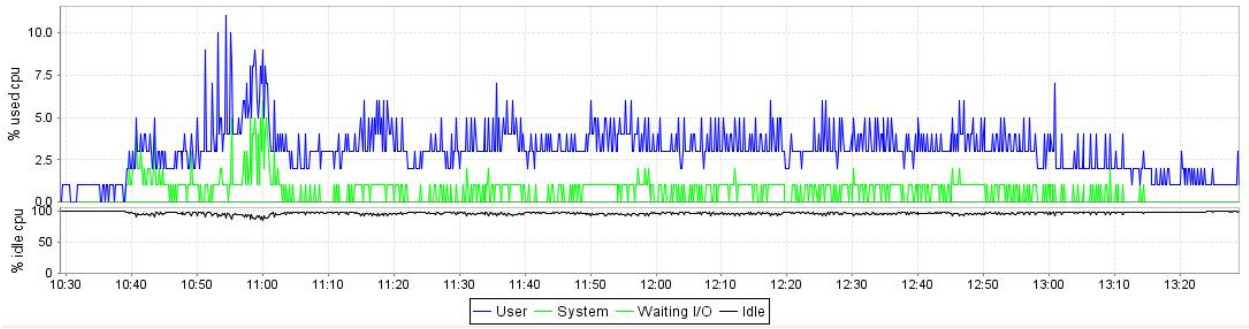
The system must meet expected hourly throughput of business transactions, as defined by the client. The following graph illustrates the systems capability to meet those requirements according to each session replication configuration. Session Replication Disabled, Scaled Application Servers, and Increased Heap Size were the only configurations to meet or exceed expectations; however, an Increased Heap Size yielded unfavorable transaction performance.



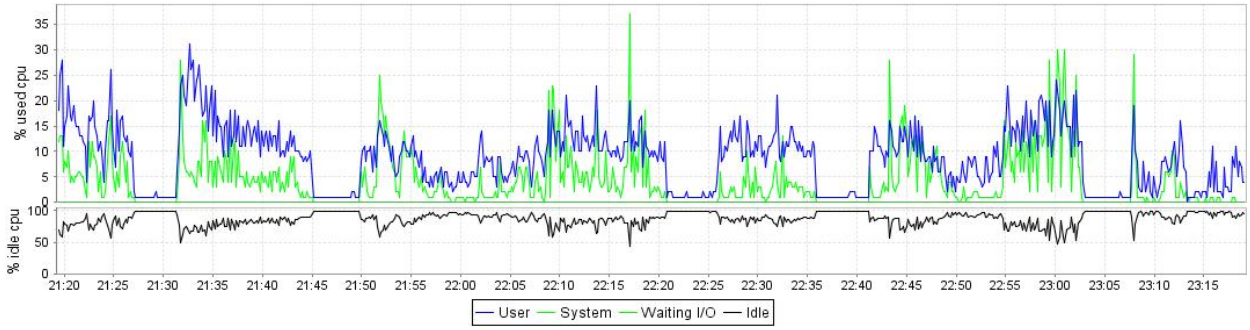
CPU Performance

The following graphs compare the hourly CPU consumption of Memory Based Persistence and Scaled Application Servers against the benchmark. While scaling the application servers yielded similar CPU consumption patterns with session replication disabled, the memory based persistence showed obvious signs of increased CPU consumption and erratic behavior, including multiple spikes of system (kernel) processes above user processes (applications) – the OS is conducting a significant amount of work, which is indicative of a resource constraint or inefficient use of resources.

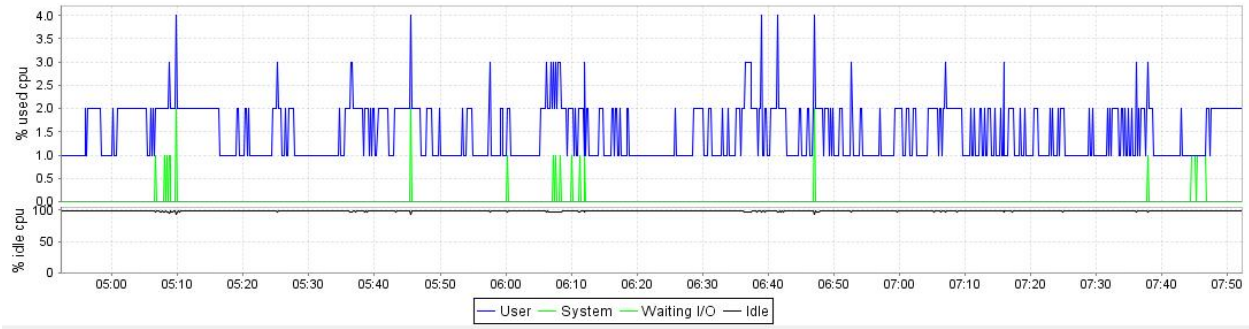
Session Replication Disabled



Memory Based Persistence



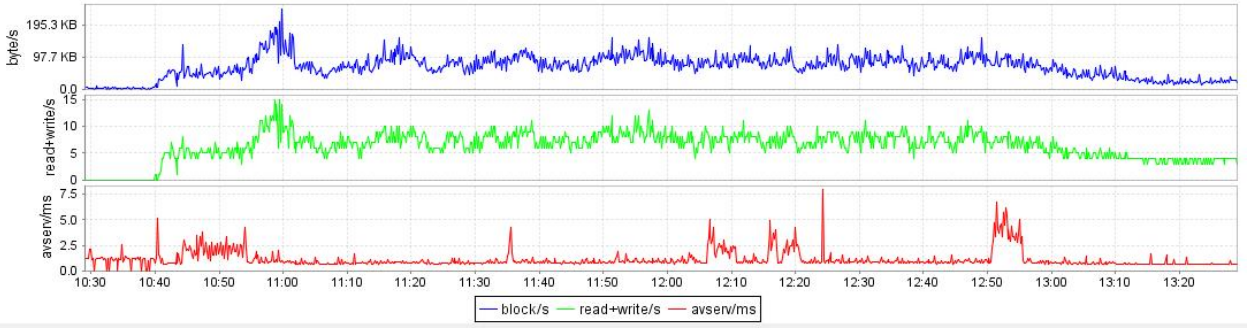
Scaled Application Servers

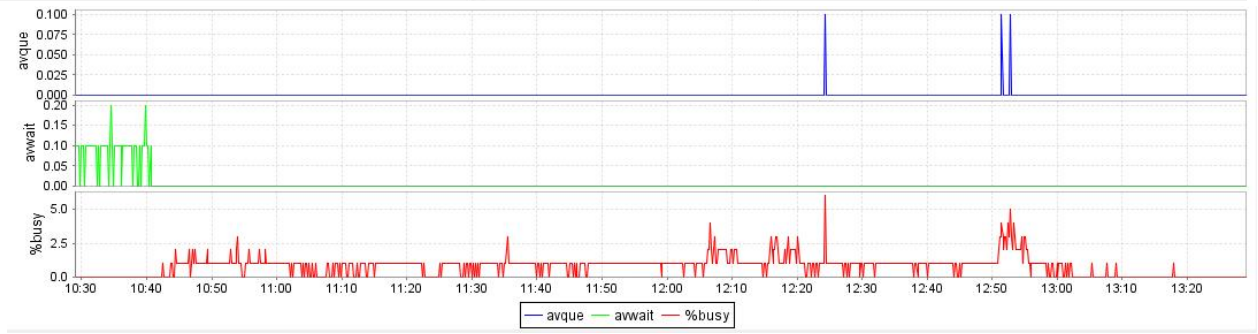


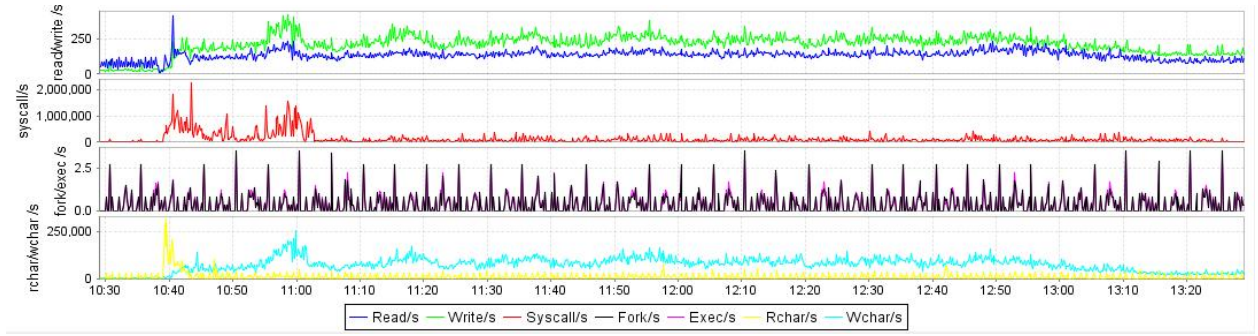
Physical Disk Performance

The following graphs illustrate the performance impacts of File Based Persistence against the benchmark. It is evident that the system spent significantly more time in waiting for the disk to serve requests (avserv/ms), significantly more data was written per second (blocks/s), frequency of writing/reading from disk increased, and the percentage of time the disk was busy increased. Most notably, the amount of bytes written per second increased from roughly ~95kb to ~950kb, while the number of read/writes per second increased from roughly 250/sec. to 50,000/sec. While the disk remained fairly idle at 5% busy per second, the system was heavily utilizing the disk and therefore expensive I/O operations led to poor performance and bottlenecks.

Session Replication Disabled







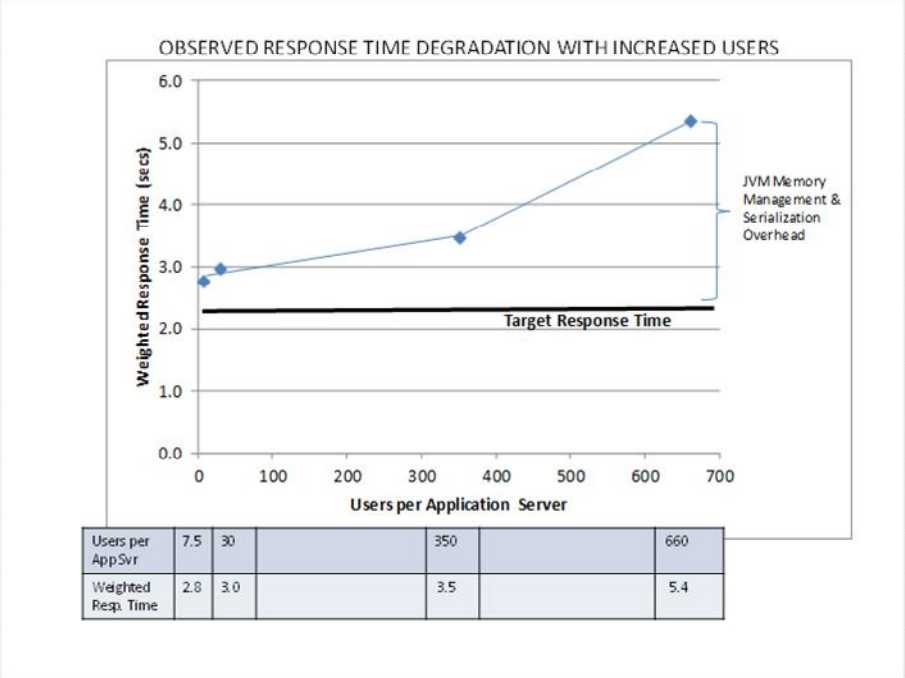
File-Based Persistence

Conclusions

Impacts From User Load

The single greatest determinant and indicator of increased performance degradation with Memory Based Persistence on the current configuration is the number of users per application server. Response times are reasonable with a low number (<50) of users per application server, and response times progressively degrade when there are larger numbers of users (>100) per application server.

Current peak load conditions and planned application server cluster configurations call for the support of 6,600 users with 6 application servers, resulting in a ratio of 1,100 users per application server. Data indicates undesirable response time degradation at 3,960 user load on 6 application servers, or 660 users per application server. The recommendation is to reduce this ratio to approximately 300 users per application server.



Path Forward

As observed through the performance testing exercises, session replication adds a significant overhead to this application. The only probable solutions for this application are Session Replication Disabled and Scaled Application Servers. Other options exist, but were not evaluated for their performance, including the use of a Distributed Cache (Oracle Coherence) or an advanced hardware solution (Oracle Exalogic).

In order to scale the servers, the architecture would need an additional 38 Weblogic servers with supporting infrastructure changes, including five additional physical host servers, increased physical memory per machine, and other supporting architecture components. The estimated costs for this solution are in the $2M range, while the estimated losses per year for the organization are less than $3,000.

Although implementing session replication is feasible, the benefits to the organization are marginal and do not outweigh the costs. After reviewing the results of the testing effort and costs-benefits analysis, the client decided against pursuing session replication across the enterprise. Instead, as an economical approach, session replication would be made available as an “a la carte” option to the business units.

Contributors

Primary contributors

Sheel Kapur - skapur@deloitte.com

Mr. Kapur has extensive experience leading teams through all phases of the SDLC for the delivery of custom enterprise solutions. His background encompasses proficiency in JEE application design and development, software architecture, systems security engineering, business process management, performance engineering, and CMMI Level 3 processes.

Java Community of Practice

The Java Community of Practice is a community of technology professionals whose primary focus is Java based technology and its applications. The community provides a platform for collaboration, exchange of ideas and encouraging contemporary best practices among its members. Our goal is to continuously enhance our capability to deliver technology solutions across different industries.

More information about the Java Community of Practice as well as additional collaboration and Java resources can be found at our KX site here.