

Middleware and Web Services

Lecture 6: High Availability and Performance

doc. Ing. Tomáš Vitvar, Ph.D.

tomas@vitvar.com • @TomasVitvar • <http://vitvar.com>



Czech Technical University in Prague

Faculty of Information Technologies • Software and Web Engineering • <http://vitvar.com/courses/mdw>



Modified: Sun Nov 26 2017, 22:57:02
Humla v0.3

Good Performance

- What influences good performance?
 - *Number of users and concurrent connections*
 - *Number of messages and messages' sizes*
 - *Number of services*
 - *Infrastructure – capacity, availability, configuration, ...*
- How can we achieve good performance?
 - *Infrastructure*
 - *Scalability, failover, cluster architectures*
 - *Performance tuning*
 - *Application Server, JVM memory, OS-level tuning, Work managers configuration*
 - *Service configuration*
 - *Parallel processing, process optimization*

Overview

- **Infrastructure**
 - *Load Balancers*
 - *Cluster Architecture*
- **Performance Tuning**

Definitions

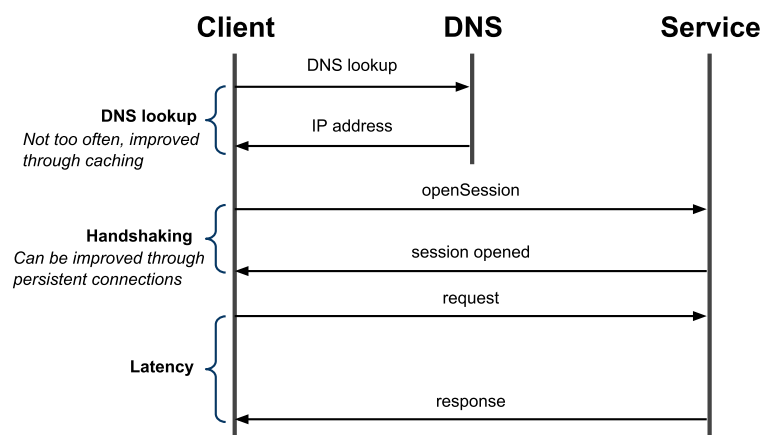
- **Scalability**
 - *server scalability*
 - *ability of a system to scale – when input load changes*
 - *users should not feel a difference when more users access the same application at the same time*
 - **horizontal scaling**
 - *adding new instances of applications/servers*
 - **vertical scaling**
 - *adding new resources (CPU, memory) to a server instance*
 - *network traffic*
 - *bandwidth capacity influences performance too*
 - *service should limit the network traffic through caching*
- **Availability**
 - *probability that a service is operational at a particular time*
 - *e.g., 99.9987% availability – downtime ~44 seconds/year*

Definitions (Cont.)

- **High Availability**
 - When a server instance fails, operation of the application can continue
 - Failures should affect application availability and performance as little as possible
- **Application Failover**
 - When an application component performing a job becomes unavailable, a copy of the failed object finishes the job.
 - Issues
 - A copy of the failed object must be available
 - A location and operational status of available objects must be available
 - A processing state must be replicated
- **Load Balancing**
 - Distribution of incoming requests across server instances

Performance Metrics

- **Latency**
 - A client-side metric

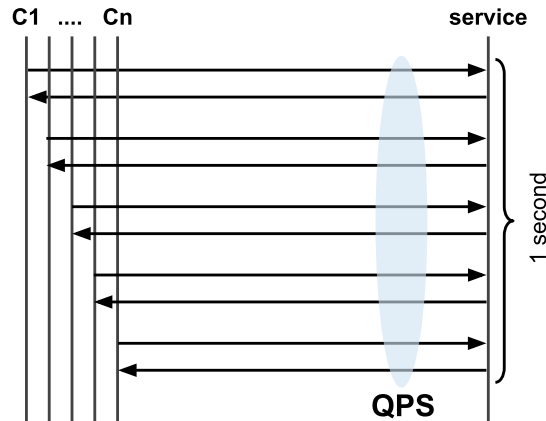


- CPU intensive service or a bad configuration of a service
 - consider asynchronous processing when CPU intensive
- Writing to a data store

Performance Metrics

- Queries/Requests per Second (QPS)

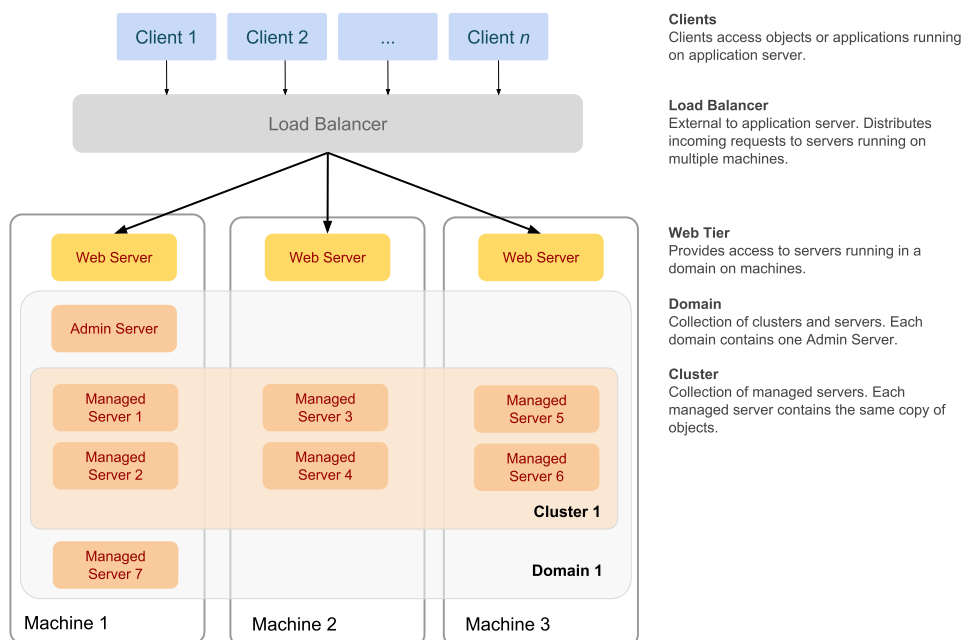
– *A server-side metric*



– *Caching may improve performance*

→ *even if data changes often, with high QPS caching improves a lot*

Infrastructure Example – Weblogic



Best Configuration Practices

- Domain configuration
 - A server is an admin server or a managed server
 - Each server is running on a separated JVM
 - A physical machine may run one or more servers
 - There should be at least two managed servers running on one machine
 - This provides a better performance
(as opposed to one server running on one machine)
 - A domain can have clustered or unclustered servers
- Load balancers (LB)
 - Load Balancers are not part of the domain
 - They are external to Weblogic server
 - There is usually one HW LB and several SW LBs
 - Software LB
 - Realized by the Web Tier (Apache HTTP server)
 - Redirects requests too all managed servers in a domain (across multiple machines)

Overview

- Infrastructure
 - *Load Balancers*
 - *Cluster Architecture*
- Performance Tuning

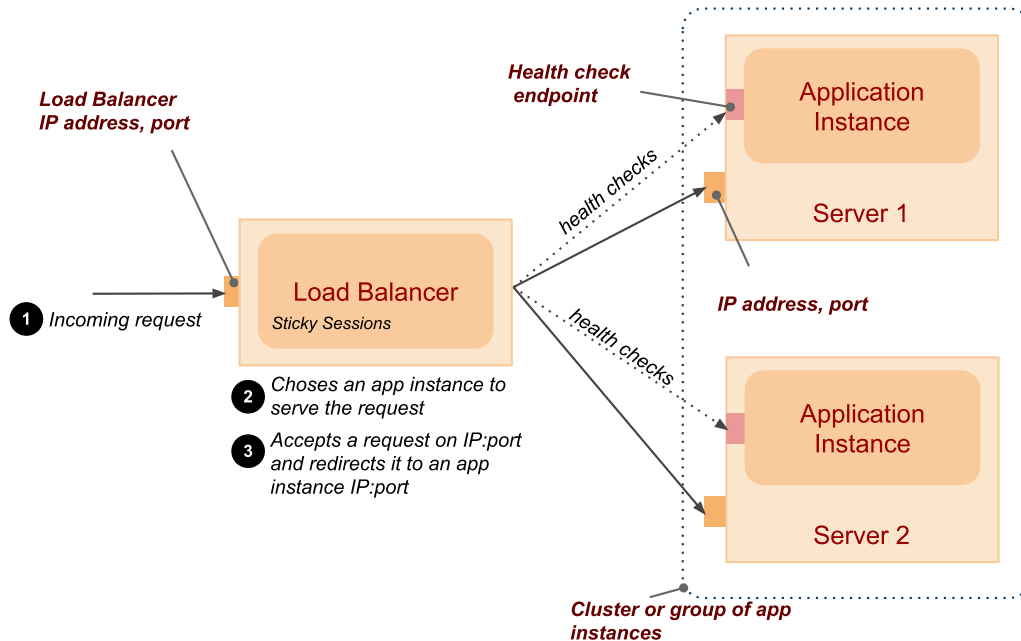
Load Balancing

- Distributes a load to multiple app/object instances
 - App instances run on different machines
 - Load sharing: equal or with preferences
 - Health checks
- Types
 - DNS-based load balancer
 - DNS Round Robin
 - NAT-based load balancer (Layer-4)
 - **Reverse-proxy load balancer** (Layer-7)
 - application layer
 - Sticky sessions
 - JSession, JSession-aware load balancer
 - Client-side load balancer
 - LB run by a client
 - a client uses a replica-aware stub of the object from the server

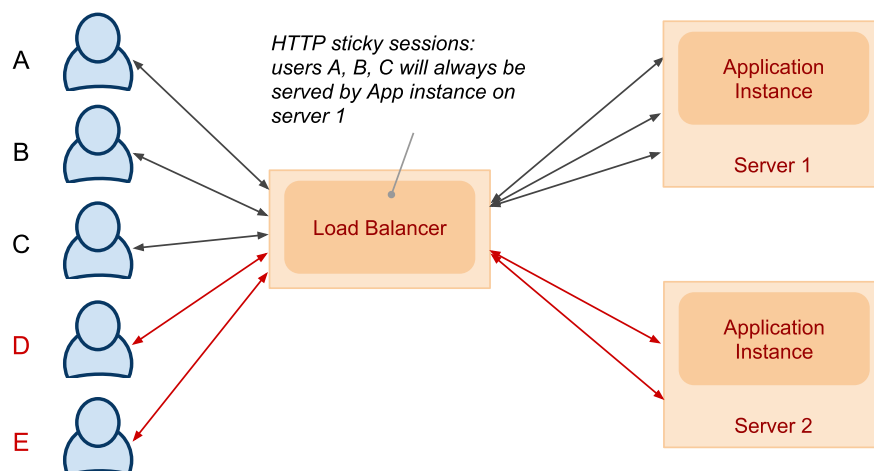
DNS-based Load Balancer

- DNS Round Robin
 - A DNS record has multiple assigned IP addresses
 - DNS system delivers different IP addresses from the list
 - Example DNS A Record:
`company.com A 147.32.100.71 147.32.100.72 147.32.100.73`
- Advantages
 - Very simple, easy to implement
- Disadvantages
 - IP address in cache, could take hours to re-assign
 - No information about servers' loads and health

Reverse Proxy Load Balancer



HTTP Sticky Sessions Example



- How to identify a server that hosts the session state
 - Passive cookie persistence – LB uses a cookie from the server
 - Active cookie persistence – LB adds its own cookie

Types of Load Balancers

- Software

- *Apache mod_proxy_balancer*
 - *HTTP Session persistence – sticky sessions*
- *WebLogic proxy plug-in*

```
1 <Location /soa-infra>
2   SetHandler weblogic-handler
3   WebLogicCluster czfmwapp03-vf:8001,czfmwapp04-vf:8001,czfmwapp05-vf:800
4 </Location>
5
```

/soa-infra is a first part of an URL path that rules in this **Location** will be applied (this is a standard Apache configuration mechanism)

czfmwapp{N} is a hostname that corresponds to a virtual IP to which the managed server JVM processes is bounded (using the tcp port **8001**).

WebLogicCluster specifies the list of servers for load balancing

- Hardware

- *Cisco, Avaya, Barracude*

Round-Robin Algorithm

- Uses

request – client request with or without a cookie information

server_list – a list of servers that can process the request

rbinx – round robin index

sticky_sessions – associative array of pairs **<session_id,server>**

unhealthy_treshold – a number of negative consecutive health checks before moving the server to the "unhealthy" state.

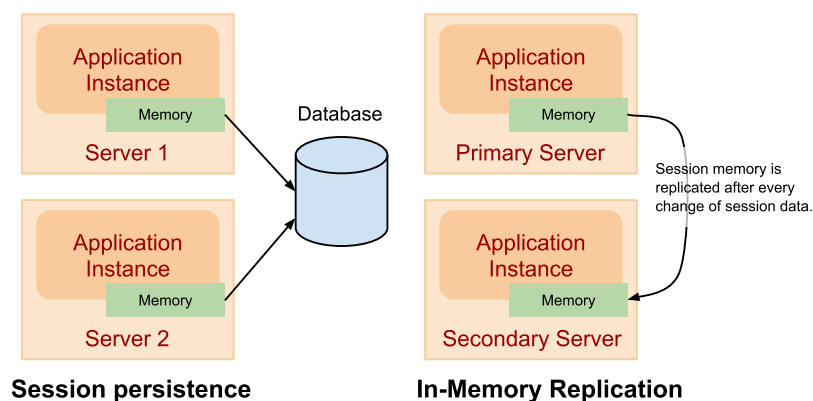
- Round Robin Algorithm

- if **session_id** exist in the **request** and in **sticky_sessions**
 - send the **request** to the server **sticky_sessions[session_id]**
- otherwise
 - send the **request** to the **rbinx** server in the **server_list**
 - extract **session_id** from the response from the server
 - if the **session_id** exist, add a pair **<session_id;server_list[rbinx]>** to **sticky_sessions**
 - increase **rbinx** by one or reset it to **0** if it exceeds the length of **server_list**

Health Check

- Health Check
 - For each server in the `server_list`
 - call the server's healthcheck endpoint
 - if a number of failed health checks for the server exceeds the `unhealthy_threshold`
 - remove the server from the `server_list`
 - if the server was unhealthy and there was a successful healthcheck
 - add the server back to the `server_list`

Session State Persistence and Replication



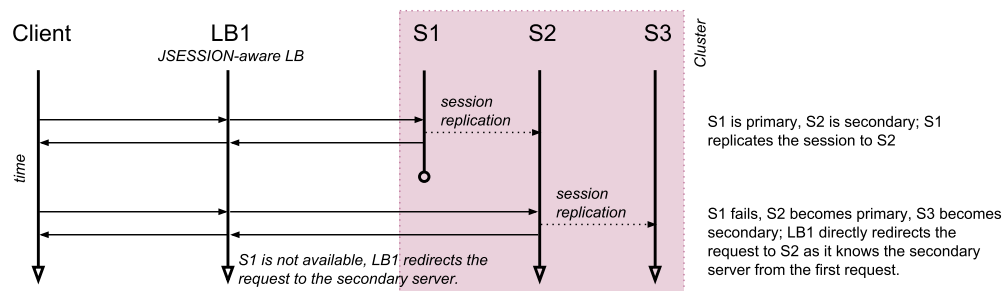
- Session persistence
 - Session information is maintained in the database
 - Does not require sticky sessions
 - Implements `HttpSession` interface that writes data to the DB
- In-memory replication
 - A **primary server** holds a session state, the **secondary server** holds its replica.
 - Information about primary and secondary servers are part of `JSession`

In-Memory Replication

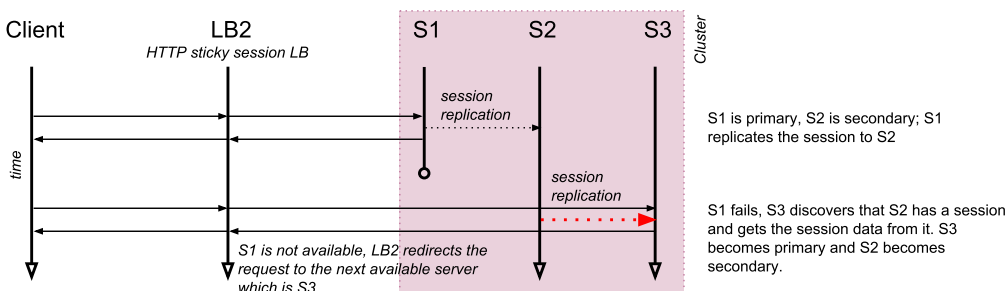
- Session format
 - It's a cookie
 - `JSESSIONID=SESSION_ID!PRIM_SERVER_ID!SEC_SERVER_ID!CREATION_TIME`
 - `SESSION_ID` – session id, generated by the server to identify memory associated with the session on the server
 - `PRIM_SERVER_ID` – ID of the managed server holding the session data
 - `SEC_SERVER_ID` – ID of the managed server holding the session replica
 - `CREATION_TIME` – time the session data was created/updated
- How LB uses this information
 - LB has information whether the server is running or not (via healthchecks)
 - if the primary server is running, it redirects the request there
 - if the primary server is not running, it redirects the request to the secondary server directly
 - if primary and secondary servers are not running, it redirect the request to any other server it has in the list – this may cause side effects!

In-Memory Replication Scenarios

Scenario A: JSession-aware load balancer



Scenario B: HTTP sticky session load balancer



Overview

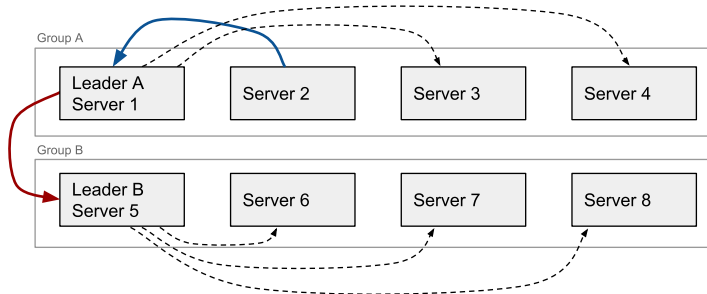
- Infrastructure
 - *Load Balancers*
 - *Cluster Architecture*
- Performance Tuning

Overview

- Cluster capabilities
 - *A group of servers (aka cluster members) act together to serve clients' requests*
 - *Cluster is transparent to clients*
 - *Servers can run on the same machines or on different machines*
 - *Cluster's capacity can be increased by adding servers to the cluster*
 - *Servers in a cluster may have the same copy of objects and they are aware of each other objects*
 - *objects: applications, JMS destinations, RMI objects*
 - *See [Cluster-wide JNDI tree](#) in Lecture 4*
- Cluster Messaging Protocols
 - *When servers need to send messages to other members of the cluster*
 - **Unicast** - *one-to-one communication using TCP/IP sockets*
 - **Multicast** - *one-to-many communication*
 - *Cluster services that rely on the cluster messaging protocol*
 - *Cluster Membership*
 - *JNDI Replication*
 - *There are services using persistent RMI connections between cluster members*
 - *such as during in-memory replication of HTTP session information*

Unicast

- Overview
 - Uses standard TCP/IP sockets to send messages across cluster members
 - Uses a **group leader strategy** to limit the number of sockets required to send messages within the cluster
- Groups in the cluster
 - Cluster is split into **M** groups with **N** servers, each group has a leader
 - Servers communicate with the group leader, group leaders communicate with servers in the group and other group leaders
 - When a group leader dies, the group elects another group leader
 - There is up to **MxN** network messages for every cluster message



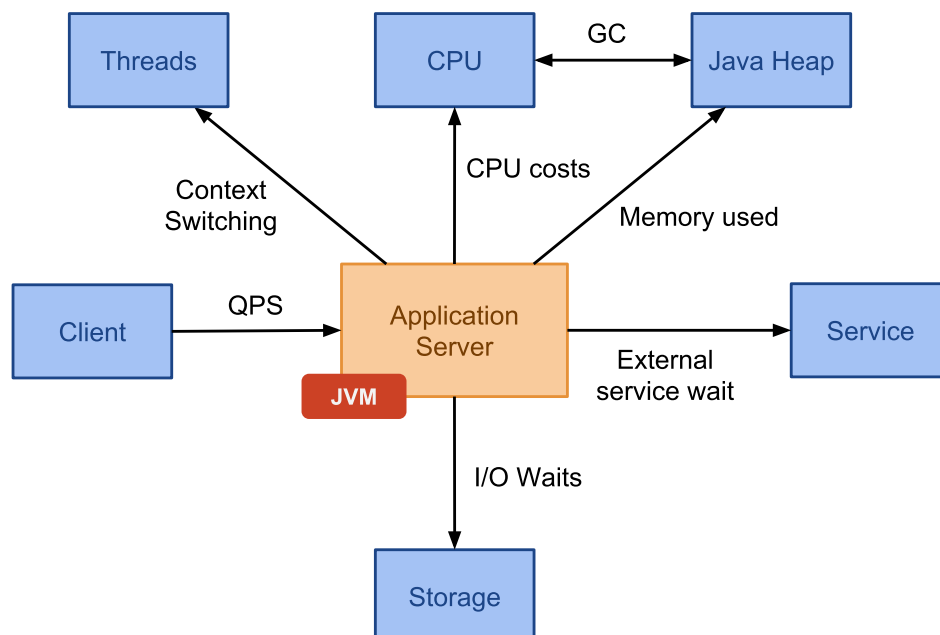
Cluster Services

- Cluster Membership
 - Cluster members maintain their view on what servers are in the cluster
 - Each server sends heartbeat messages to the cluster
 - others know it is alive
 - Each server receives heartbeat messages
 - When a server misses a number of messages from a cluster, it removes the server from the list until it receives the next heartbeat message.
- JNDI Replication
 - Provides each server with a cluster-wide view of the JNDI tree.
 - Servers send JNDI update messages to the cluster when an object is bound or removed to their local tree.
 - When a server leaves the cluster
 - other members remove its JNDI bindings from their tree.
 - When a server joins the cluster
 - The server asks other server for a JNDI state dump.

Overview

- Infrastructure
- Performance Tuning

Performance Limiting Factors



Monitoring

- Important to understand performance
 - *DevOps monitoring trends*
- What you need
 - *Collect → Filter → Store → View → **Tune***
 - *Metrics, dashboards, alerting, log management, reporting, tracing capabilities*
 - *It is necessary to organize metrics well in order to understand what is going on*
 - *Start from a high-level process, detail to technical components*
- Source
 - *Application server*
 - *usually management beans with JMX interfaces*
 - *log files (access logs, server logs, etc.)*
 - *OS*
 - *many utilities available out of the box*
 - *open sockets, memory, context switches, I/O performance, CPU usage*
 - *Database*
 - *applications may write metrics to the DB*
 - *SQL scripts to collect metrics*

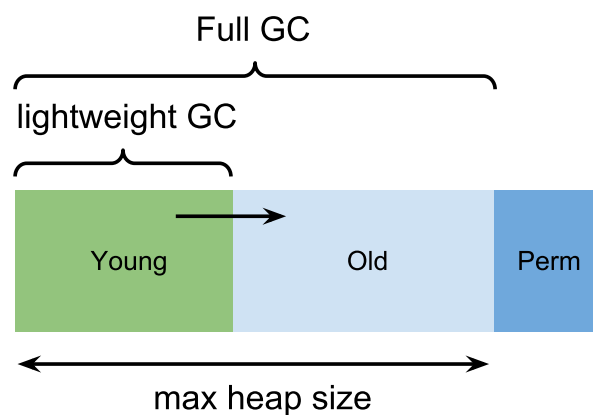
Monitoring Tools

- Commercial Monitoring Solutions
 - *Application server vendor usually offers a monitoring solution*
 - *AppDynamics, Oracle Enterprise Manager, Splunk*
 - *Google stackdriver, Amazon AWS CloudWatch*
- Open source examples
 - *Elasticsearch + LogStash + Kibana*
 - *InfluxDB + Telegraph + DataGraph*

Tuning – A Layered Approach

- Application server can be tuned at multiple layers
 - *Service configuration optimization*
 - *Transport-level tuning*
 - *Application Server Tuning*
 - *JVM Memory Tuning*
 - *OS Tuning*
- Lower levels are cheaper to tune

Memory Allocations



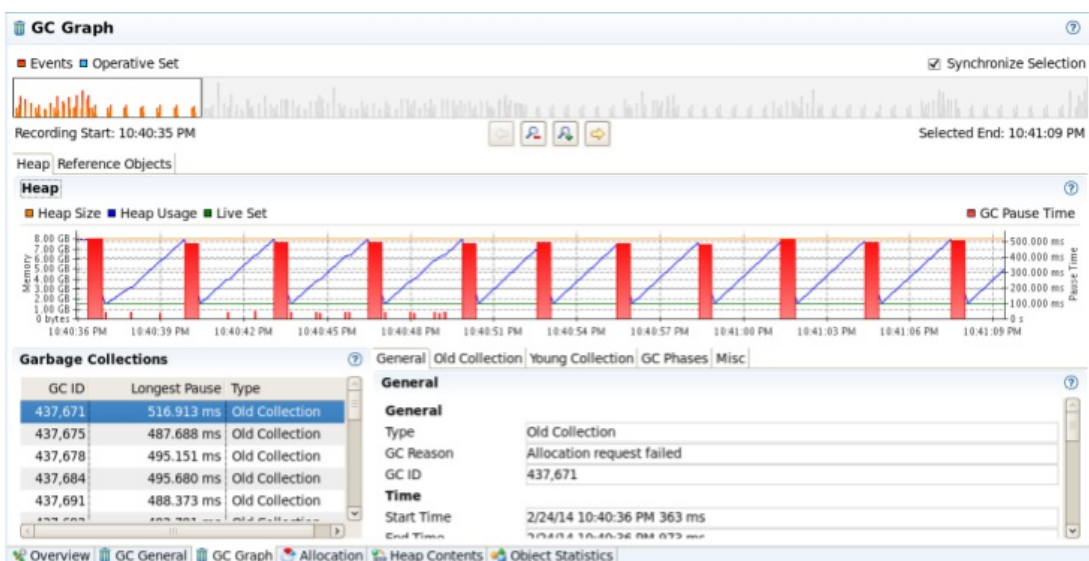
- Generations
 - *Young – objects get allocated in this space initially*
 - *Old – objects get promoted to old from young*
 - *Perm – space for permanent allocations, e.g. objects describing classes and methods*

Garbage Collection

- Steps to move objects around
 1. Objects are created in young
 2. When young is full, the live objects are copied to old, dead are discarded – **lightweight GC**
 3. When young is full and no space in old → the full GC frees the old space – **Full GC** – nothing is running in JVM, the application stops – **Too frequent full GC has an impact on performance**
- A memory leak or inadequate heap allocation
 - Old is out of space → full GC will run often (or continuously)
 - High CPU utilization, The server will not be able to process/respond to requests

Heap Size and GC Runs

- Heap Size and GC runs
 - Wrong heap size allocation – too small or memory leaks
 - GC full runs too often, this has a negative impact on performance

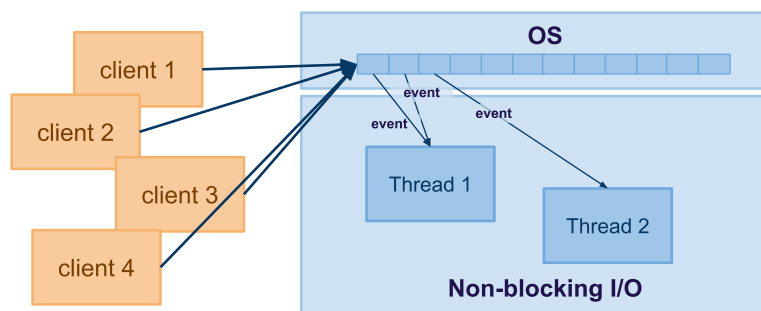


JVM Memory Tuning

- JVM Memory Parameters
 - Xms – initial java heap size
 - Xmx – maximum java heap size
 - XX:NewSize – the initial size of the heap for young generation
 - XX:MaxNewSize – the maximum size of the heap for young generation
- General recommendations
 - Xms and -Xmx should be set to the same value
(do not allow the heap to grow → limit the overhead)
 - XX:NewSize and -XX:MaxNewSize should be set to the one half of maximum heap
 - Example, 1GB heap size
 - Xms1024m -Xmx1024m -XX:NewSize=500m -XX:MaxNewSize=500m

Asynchronous I/O: Recall

- Connections maintained by the OS, not the Web app
 - The Web app registers events, OS triggers events when occur



- Characteristics
 - Event examples: new connection, read, write, closed
 - The app may create working threads, but controls the number!
→ much less number of working threads as opposed to blocking I/O

Work Manager Configuration

- Work Manager
 - Controls the number of thread allocated to processing of requests
 - In WLS is called a dispatch policy
 - Can be assigned to OSB proxy services
 - Parameters
 - **maximum threads** (**max**) – maximum number of working threads
 - **capacity** (**cap**) – maximum number of connections
 - maximum connections waiting to be processed: **cap** - **max**
 - refused connections: when number of connections is > **cap**
- Inbound throttling
 - A dispatch policy applied to a single proxy service
 - Rejected connections will not be processed