

# 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: Tue Nov 22 2016, 20:46:32  
Humla v0.3

# Good Performance

- What influences a 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 a 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

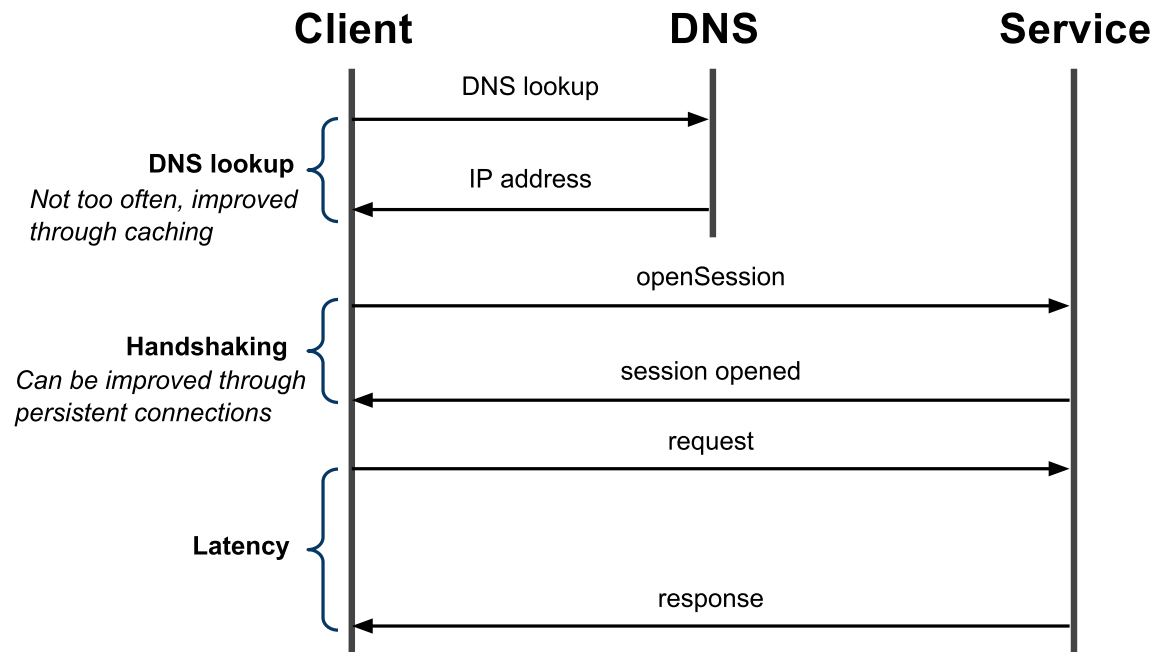
- Scalability
  - *server scalability*
    - *ability of a system to scale – when input load changes*
    - *users should not feel a difference when more users access the same application at the same time*
    - **horizontal scaling**
      - *adding new instances of applications/servers*
    - **vertical 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*

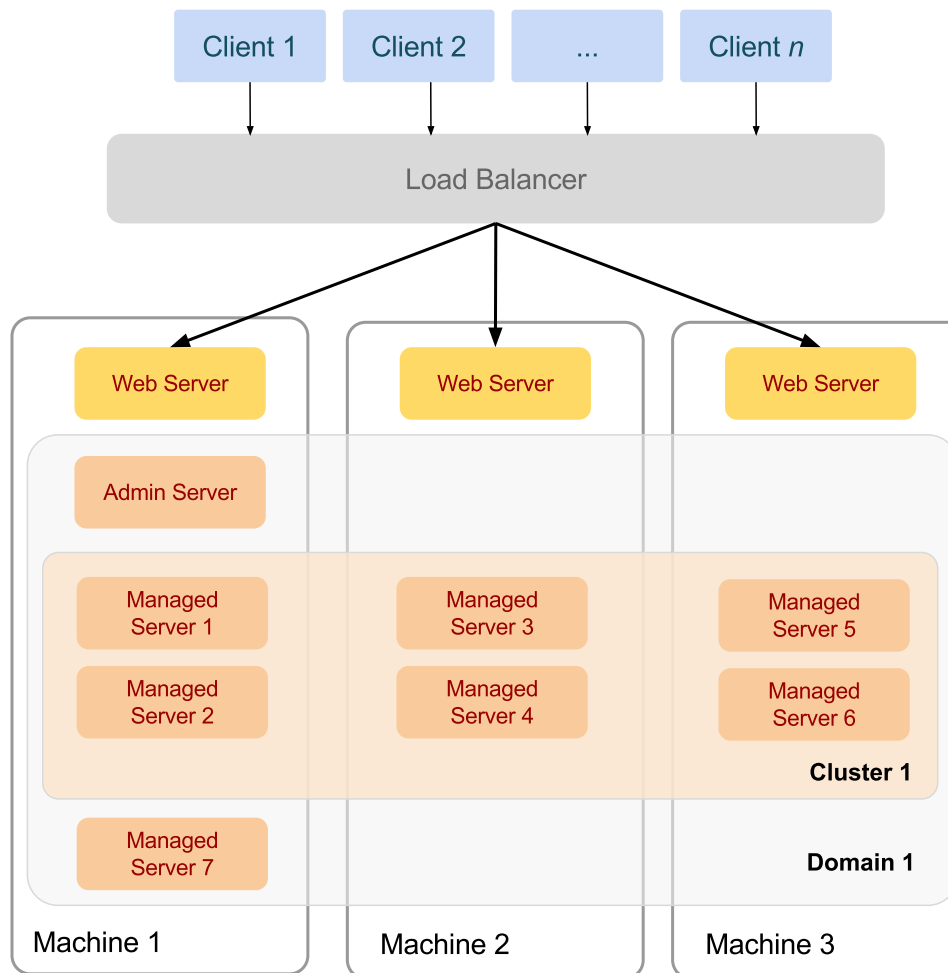
# Performance Metrics

- 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



## Clients

Clients access objects or applications running on application server.

## Load Balancer

External to application server. Distributes incoming requests to servers running on multiple machines.

## Web Tier

Provides access to servers running in a domain on machines.

## Domain

Collection of clusters and servers. Each domain contains one Admin Server.

## Cluster

Collection of managed servers. Each managed server contains the same copy of objects.



# 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

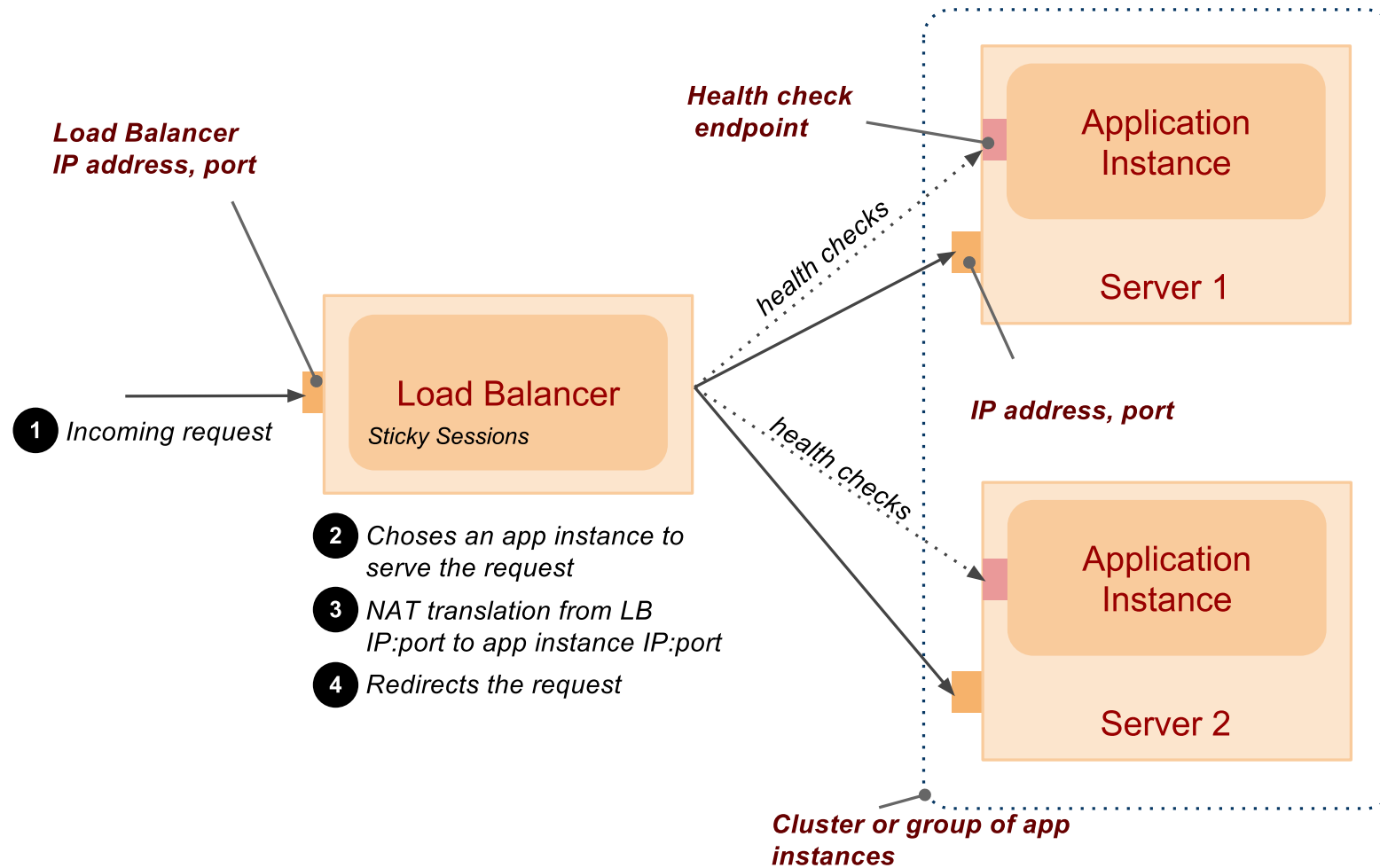
# Load Balancing

- 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:8001
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

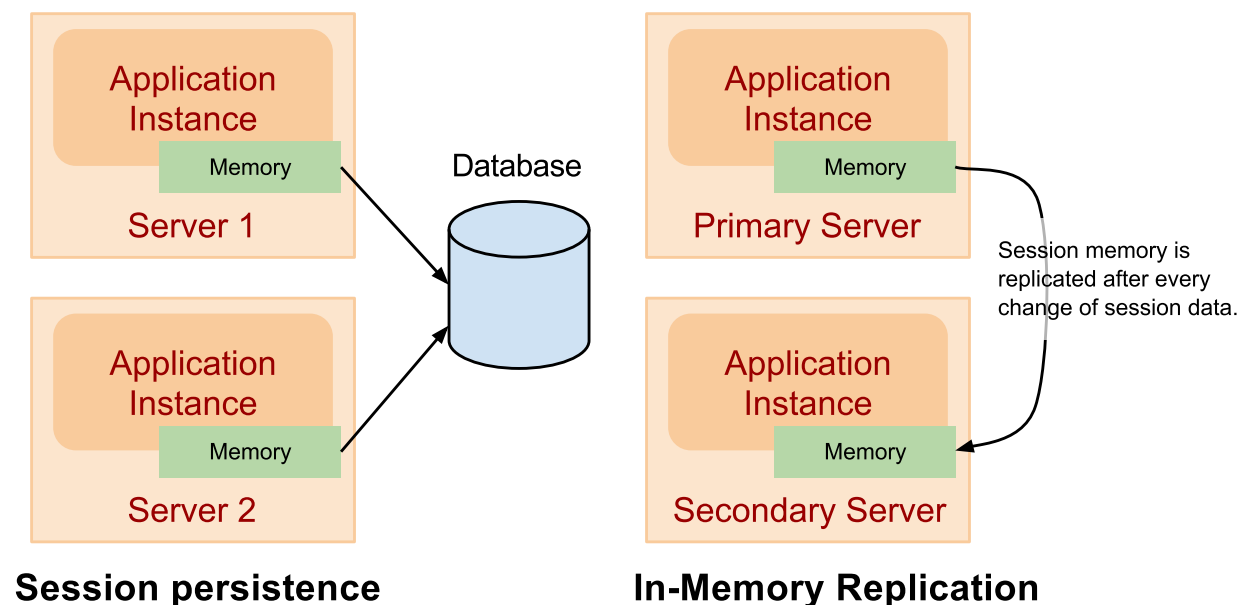
- *Cisco, Avaya, Barracude*

# Round-Robin Algorithm with Health Check

- Uses
  - *request* – client request with or without a cookie information
  - *server list* – a list of servers that can process the request
    - e.g. **WebLogicCluster** value (see previous slide)
  - *unhealthy treshold* – a number of negative consecutive health checks before moving the server to the "unhealthy" state.
- Steps
  - *if a cookie exist in the request that identifies a server*
  - *always use that server*
  - **health check**
  - *LB polls the servers' healthcheck endpoints*
  - *if a number of health checks exceeds the unhealthy threshold*
    - *LB removes the server from the server list*
  - *if a server was unhealthy and a there was a successful healthcheck*
    - *LB adds the server to the server list*



# Session State Persistence and Replication



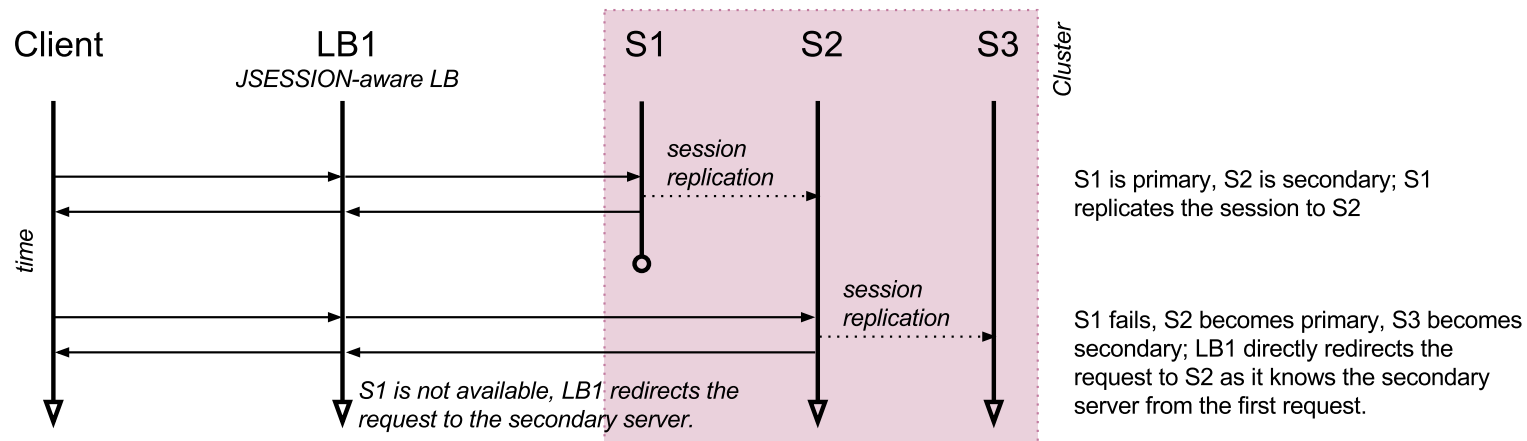
- Session persistence
  - Session information is maintained in the database
  - Does not require sticky sessions
  - Implements `HttpSession` interface that writes data to the DB
- In-memory replication
  - A **primary server** holds a session state, the **secondary server** holds its replica.
  - Information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

# In-Memory Replication

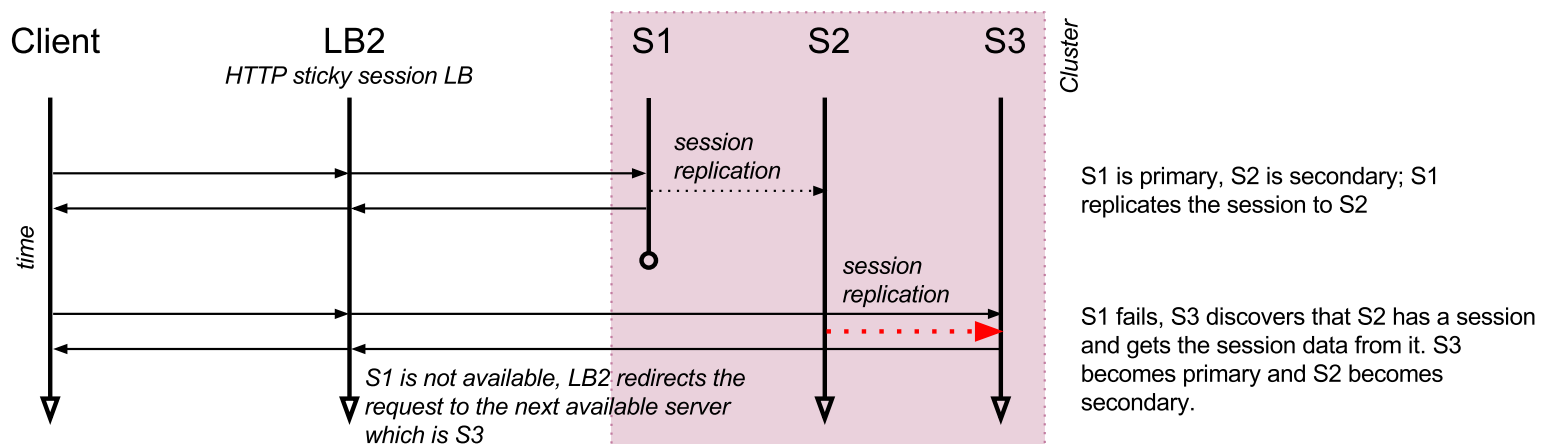
- Session format
  - *It's a cookie*
  - `JSESSIONID=SESSION_ID!PRIM_SERVER_ID!SEC_SERVER_ID!CREATION_TIME`
    - `SESSION_ID` – *session id, generated by the server to identify memory associated with the session on the server*
    - `PRIM_SERVER_ID` – *ID of the managed server holding the session data*
    - `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

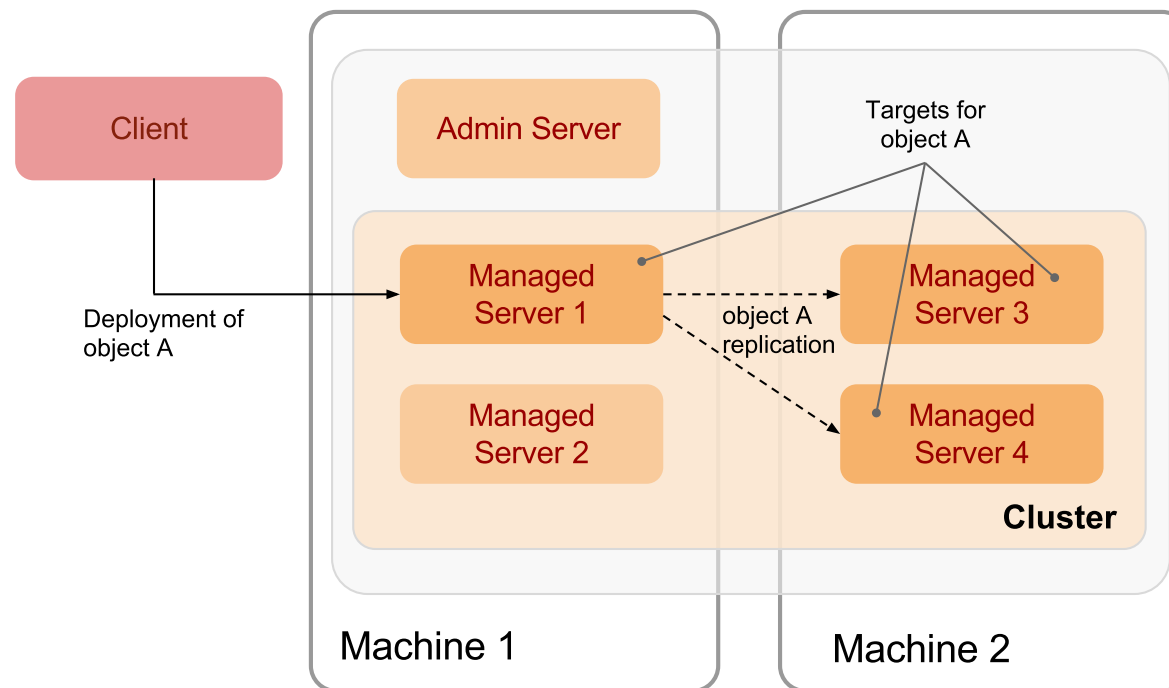
- Infrastructure
  - *Load Balancers*
  - *Cluster Architecture*
- Performance Tuning

# Overview

- Cluster
  - *A group of servers that act together to serve client requests*
  - *Cluster appears to clients as a single application server*
  - *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*
- Communication in the cluster
  - *peer-to-peer communication using IP sockets*
  - *IP multicast which servers use to broadcast availability of objects and heartbeats*
- Configurations
  - *Objects deployed to to all servers in a cluster*
    - *Cluster-wide JNDI tree allows to look-up clustered objects*
  - *Servers in a cluster may get replicated through migration*

# Deployment to Cluster

- Deployment of an object
  - Client deploys to one managed server in the cluster
  - Object gets replicated to its targets
    - Targets can be configured for the object, usually all servers but can be selected servers
    - See [Lecture 4](#) for the definition of the object

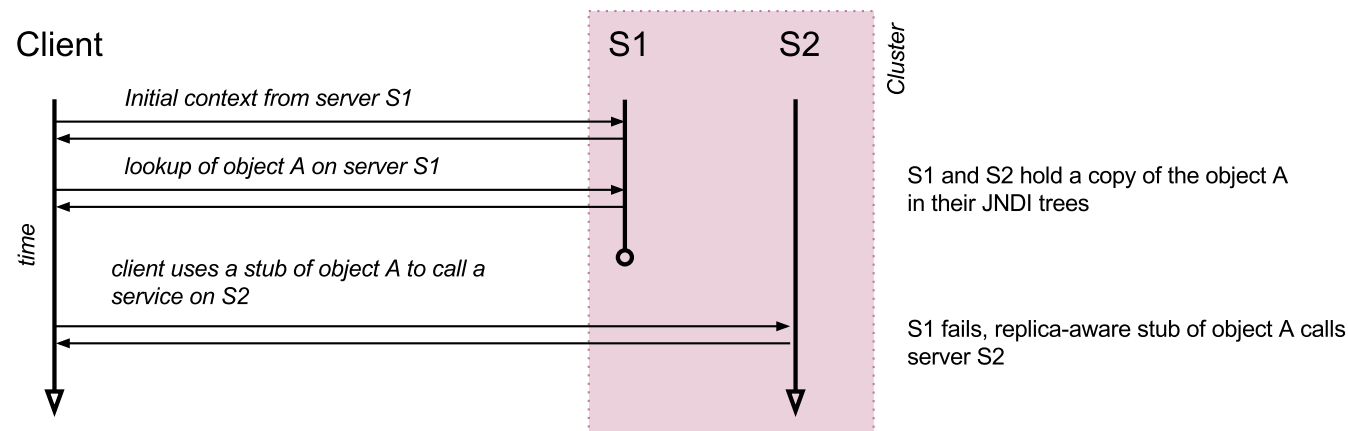


# Object Failover

- Failover

- *Failover = ability to locate an object on another server that holds a copy of the object without an impact on the performance and configuration*

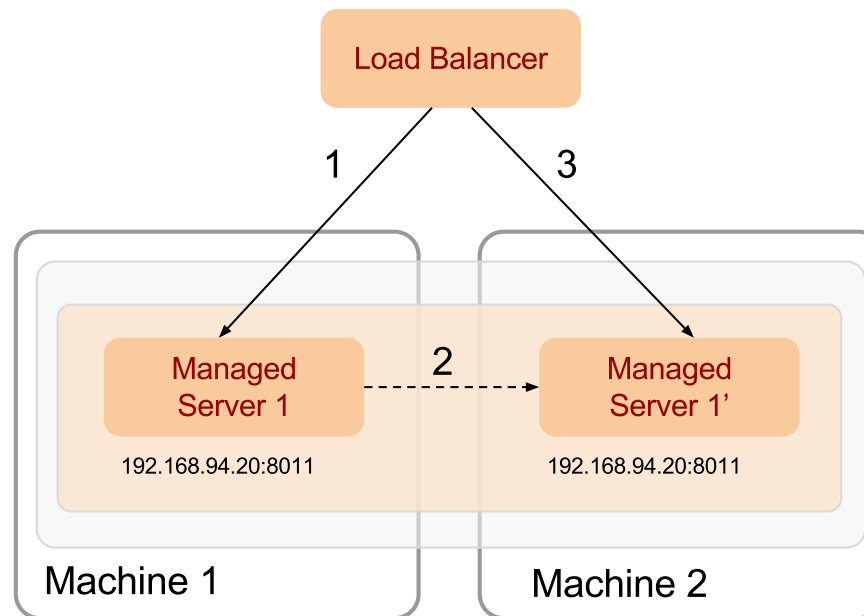
## Replica-aware stub of object A, failover in cluster



- *A client gets a stub of the object by calling **lookup** on the context*
- *A client uses the stub of the object to access the object on the server*
- *When a server fails, replicate-aware stub calls the next server that holds the object copy*

# Server Failover

- Failover
  - *Failover = ability to relocate the server to another machine without an impact on the performance*



- *Managed server listens on **virtual\_IP:port***
- *A load balancer forwards a request to **virtual\_IP:port***
- *When the server moves, **virtual\_IP:port** remains the same*



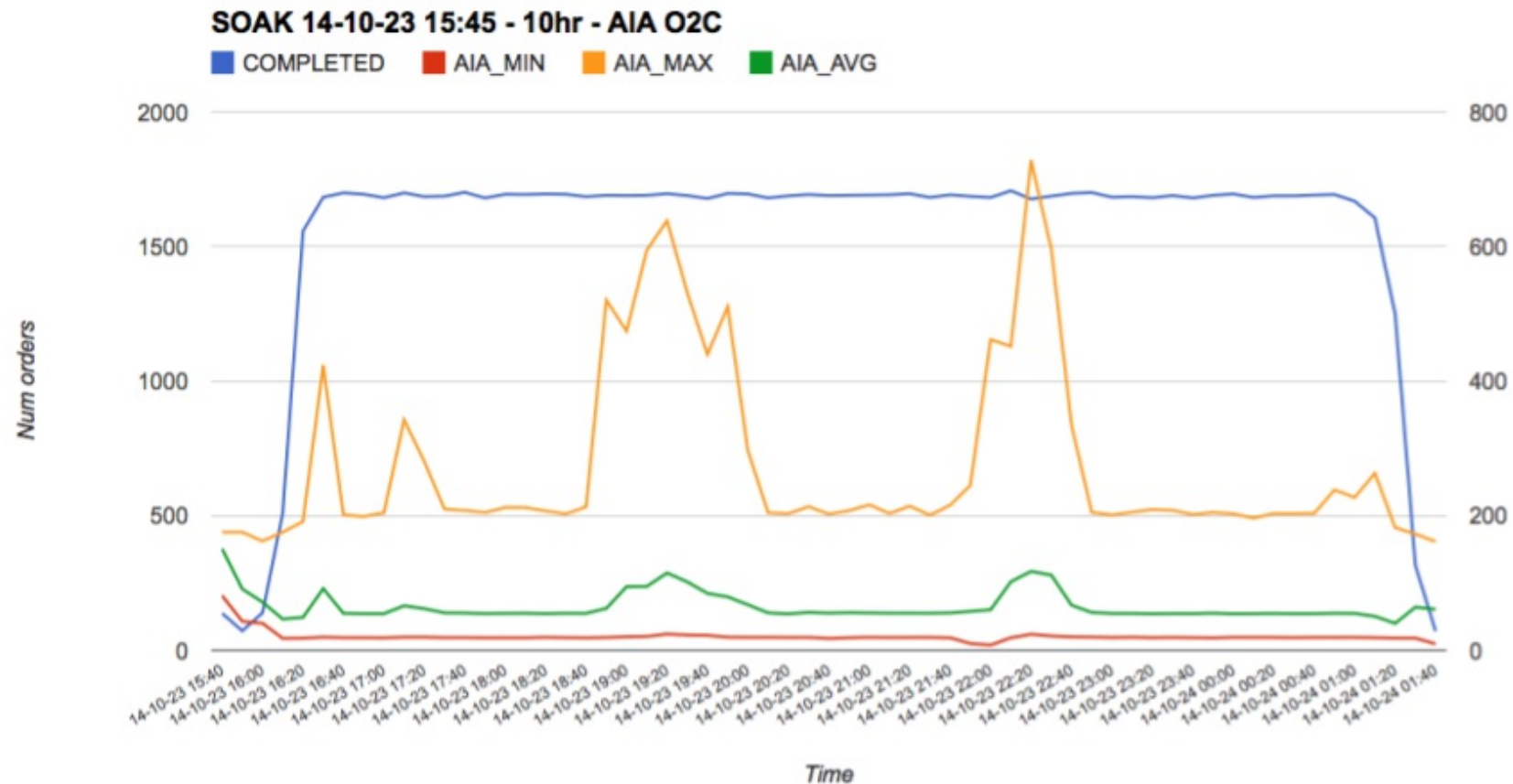
# Overview

- Infrastructure
- Performance Tuning

# Performance Limiting Factors



# Example Performance Testing

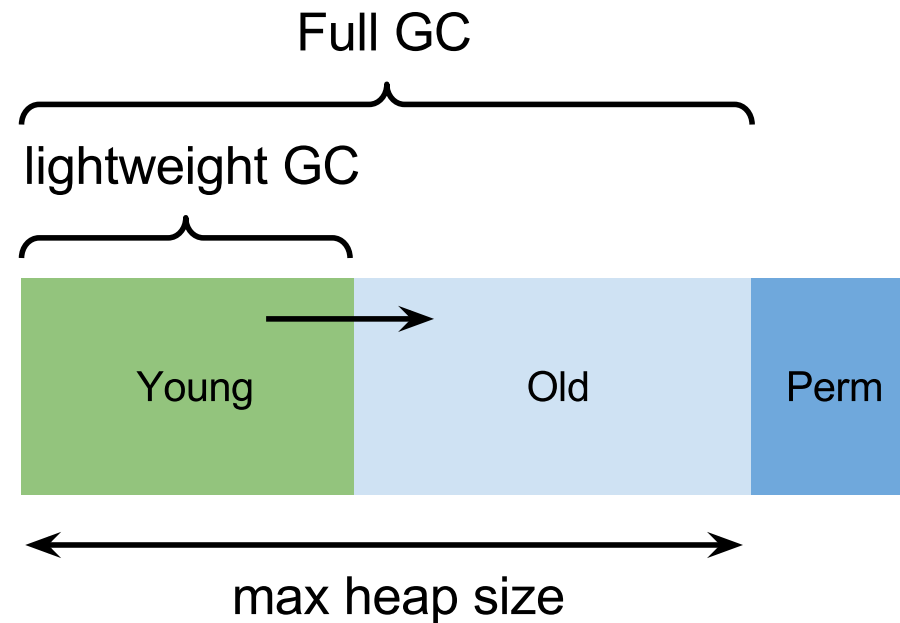


- *Completed* – number of completed orders
- *MIN, MAX, AVG* – a minimum/maximum/average processing time in 10 minutes
- *At 18:30–20:20 was a performance issue with OMS environment*

# 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



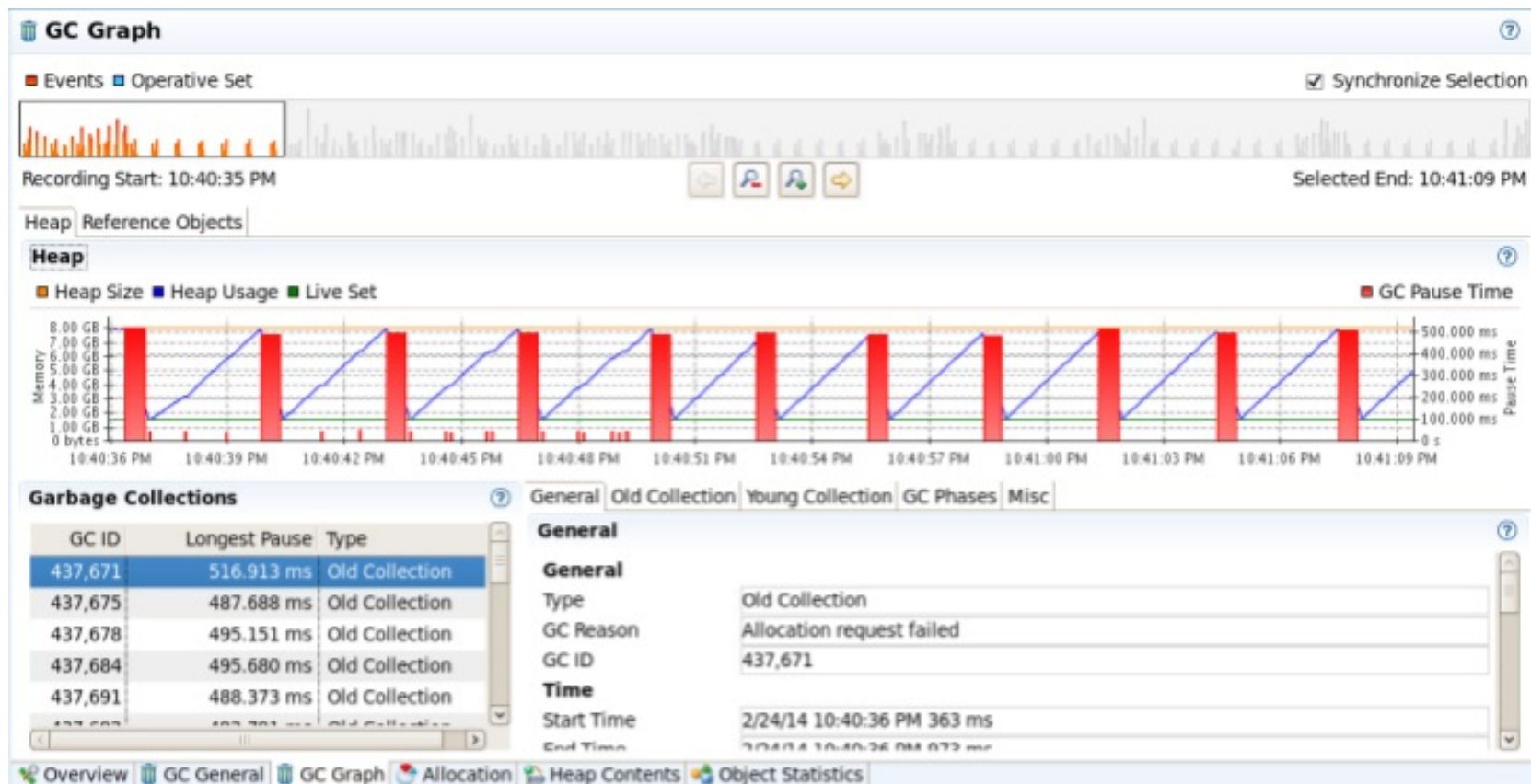
- Generations
  - *Young* – objects get allocated in this space initially
  - *Old* – objects get promoted to old from young
  - *Perm* – space for permanent allocations, e.g. objects describing classes and methods

# 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, ESB 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

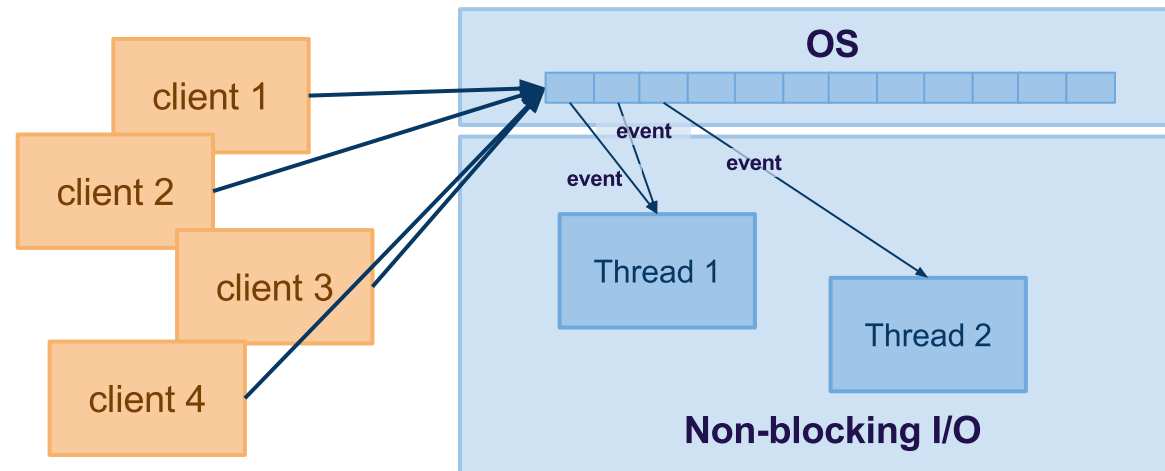
- Example, 1GB heap size

- 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