Middleware Architectures 1

Lecture 6: High Availability and Performance

doc. Ing. Tomáš Vitvar, Ph.D.
tomas@vitvar.com • @TomasVitvar • https://vitvar.com

Czech Technical University in Prague
Faculty of Information Technologies • Software and Web Engineering • https://vitvar.com/lectures

Modified: Sun Dec 05 2021, 22:09:30
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*
    $\rightarrow$ *Scalability, failover, cluster architectures*
  - *Performance tuning*
    $\rightarrow$ *Application Server, JVM memory, OS-level tuning, Work managers configuration*
  - *Service configuration*
    $\rightarrow$ *Parallel processing, process optimization*

Overview

- Definitions
- Load Balancers
- Performance Tuning

Definitions

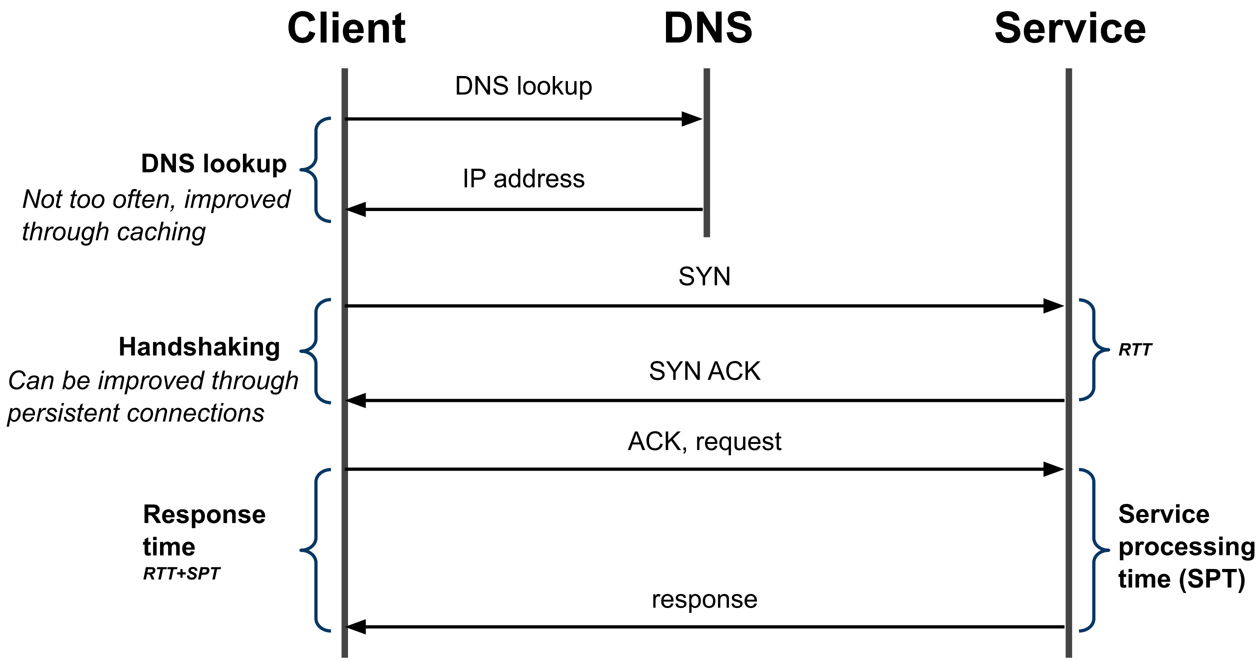
- Scalability
  - *server scalability*
    - *→ ability of a system to scale – when input load changes*
    - *→ users should not feel a difference when more users access the same application at the same time*
    - *→ **horizontal scaling***
      - *→ adding new instances of applications/servers*
    - *→ **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*
- SLA – Service Level Agreement
  - *Guarantee of service availability*
  - *When availability is below a guaranteed value, a customer can get a discount*

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

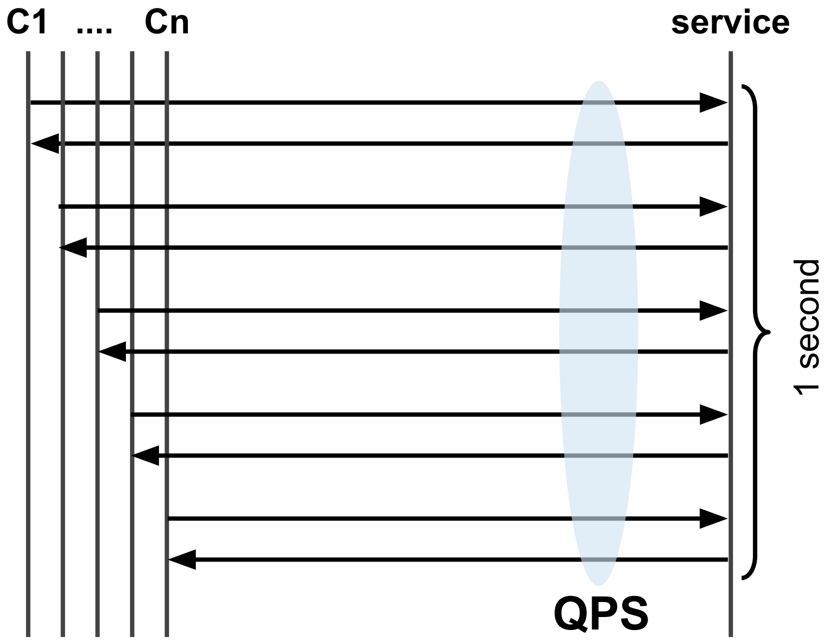
- Response Time
  - *A client-side metric*



  - *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*

Overview

- Definitions
- Load Balancers
- Performance Tuning

Load Balancing

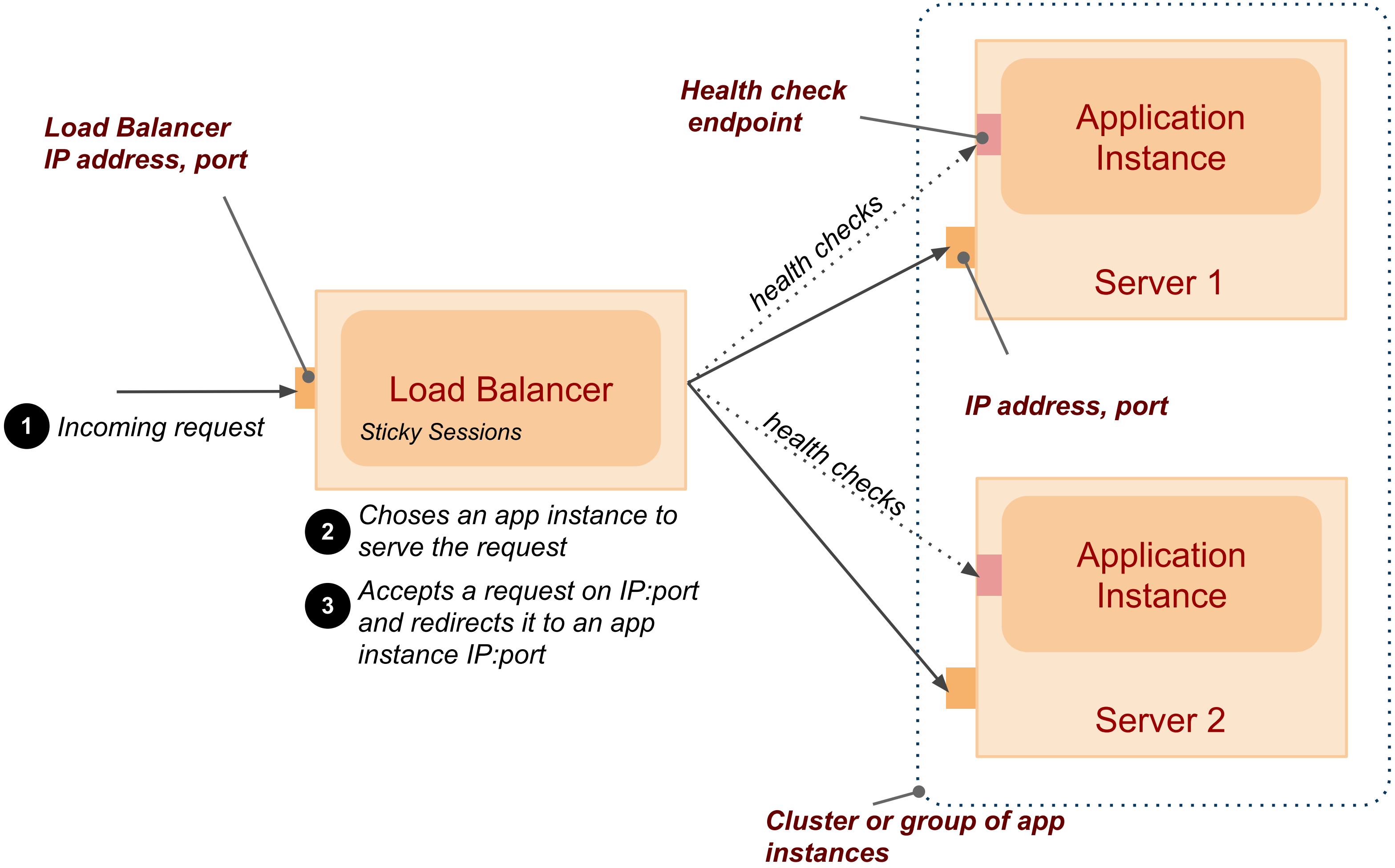
- Distributes a load to multiple app/object instances
  - *App instances run on different machines*
  - *Load sharing: equal or with preferences*
  - *Health checks*
- Types
  - *DNS-based load balancer*
    - → *DNS Round Robin*
  - *NAT-based load balancer (Layer-4)*
  - ***Reverse-proxy load balancer** (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

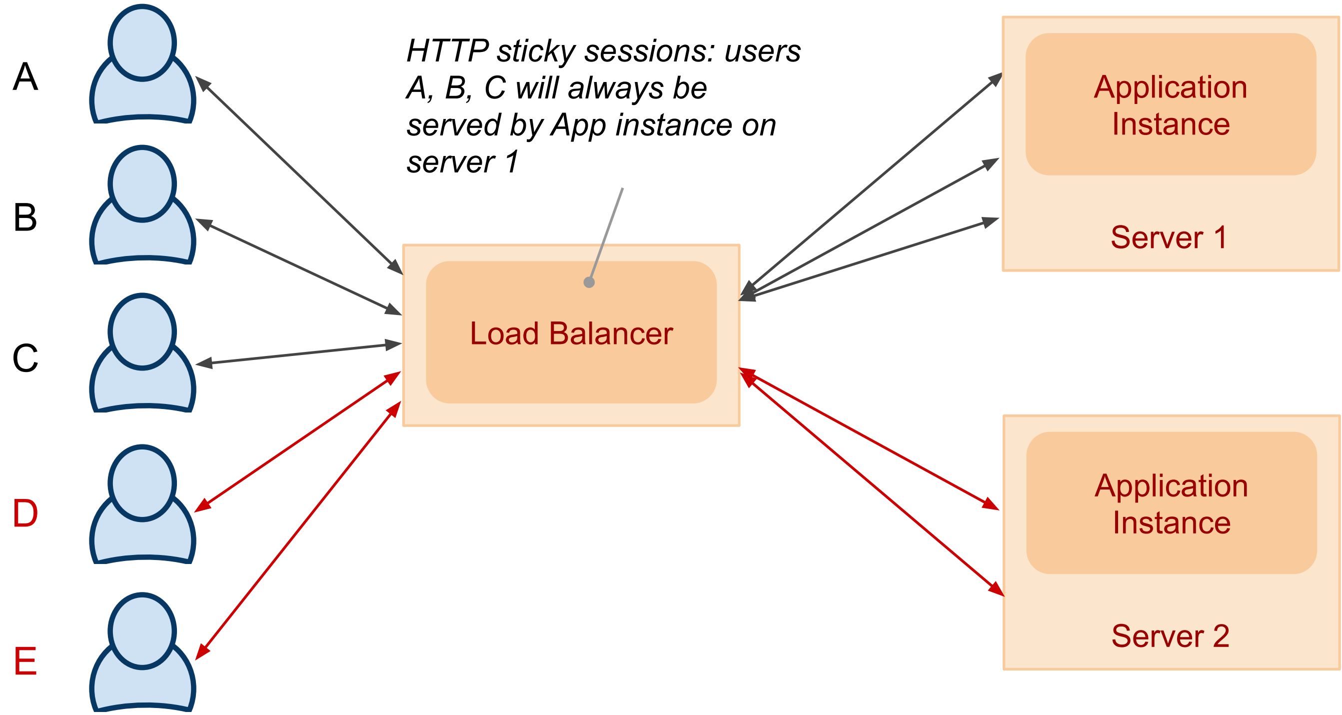
- DNS Round Robin
  - *A DNS record has multiple assigned IP addresses*
  - *DNS system delivers different IP addresses from the list*
  - *Example DNS A Record:*

    ```
    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`*, NGINX*
    - → *HTTP Session persistence – sticky sessions*
    - → *Various configuration options*
  - *WebLogic proxy plug-in*

```
<Location /soa-infra>
    SetHandler weblogic-handler
    WebLogicCluster czfmwapp03-vf:8001,czfmwapp04-vf:8001,czfmwapp05-vf:8001
</Location>

```

`/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)*

`WebLogicCluster` *specifies the list of backend 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 backend servers that can process the request*

  `rbinx` – *round robin index*

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

  `unhealthy_treshhold` – *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 heatlhcheck 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 a there was a successful healthcheck*
      - → *add the server back to the* `server_list`

Backend Server Selection Options

- Backend server with a weight and a backup server
  – *NGINX example:*

```
http {
    upstream backend {
        server backend1.example.com weight=5;
        server backend2.example.com;
        server 192.0.0.1 backup;
    }

    server {
        location / {
            proxy_pass http://backend;
        }
    }
}
```

- Least connections
  – *A request is sent to a server with the least number of active connections*
- Least time
  – *A request is sent to a server with the lowest average response time and the lowest number of active connections*
  – *Time can be:*
    → *Time to receive the resonse header*
    → *Time to receive full response body*

Backend Server Selection Options (Cont.)

- Limiting the Number of Connections
  - *Maximum number of connections per backend server*
  - *Number of connections in the queue*

```
upstream backend {
  server backend1.example.com max_conns=3;
  server backend2.example.com;
  queue 100 timeout=70;
}
```

- Hash (ip hash, generic hash)
  - *A server to which a request is sent is determined from the client IP address or an arbitrary value (string, request URL, etc.)*
- Server Slow-Start
  - *This prevents a recently recovered server from being overwhelmed*
  - *During server slow-start, connections may time out*

    → *This may cause the server to be marked as failed again.*

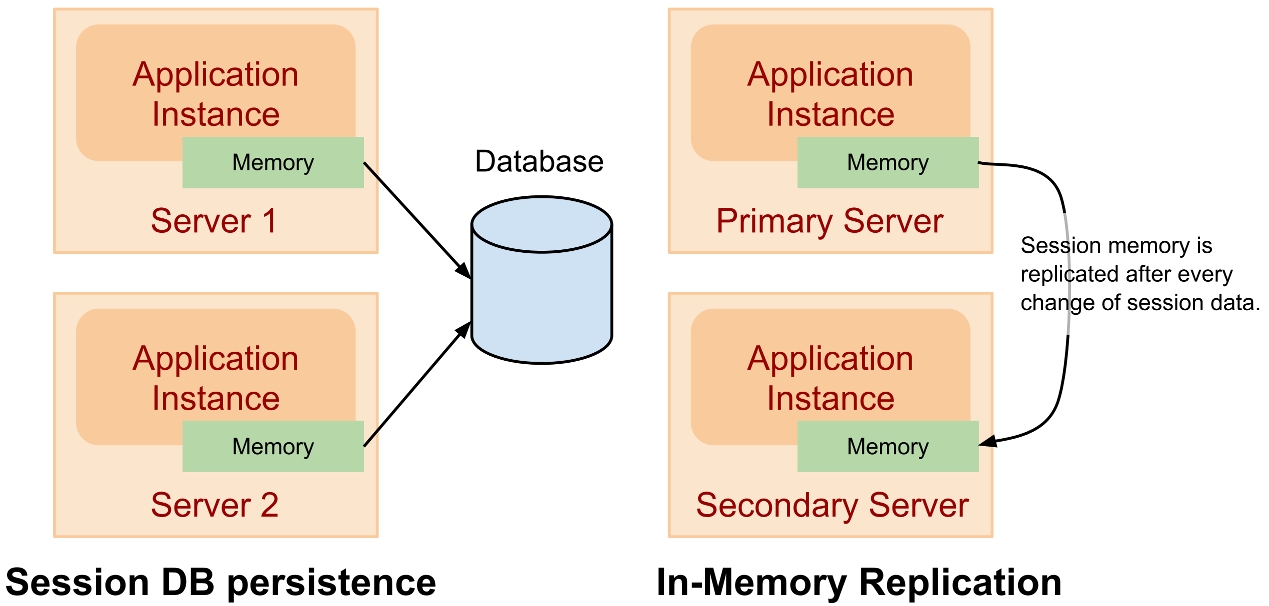
Session Persistence

- Session Persistence
  - *Sticky cookie*
    - → *A cookie defined by the load balancer for every client*

```
upstream backend {
    server backend1.example.com;
    server backend2.example.com;
    sticky cookie srv_id expires=1h domain=.example.com path=/;
}
```

  - *Sticky learn*
    - → *LB finds a cookie by inspecting requests and responses*
    - → *LB uses the cookie for subsequent redirection*

Session State Persistence and Replication



- Session DB persistence
  - *Session information is maintained in the database*
  - *Does not require sticky sessions in LB*
  - *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*

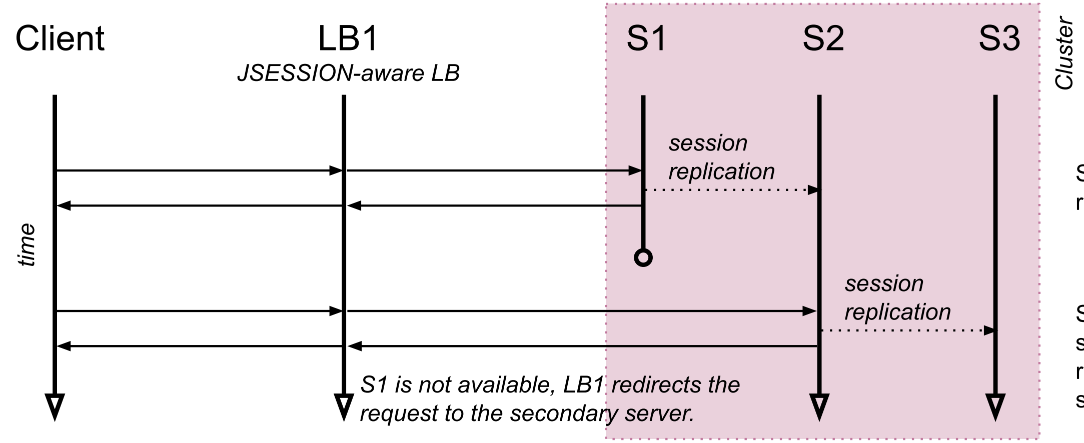
    `PRIM_SERVER_ID` – *ID of the managed server holding the session data*

    `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

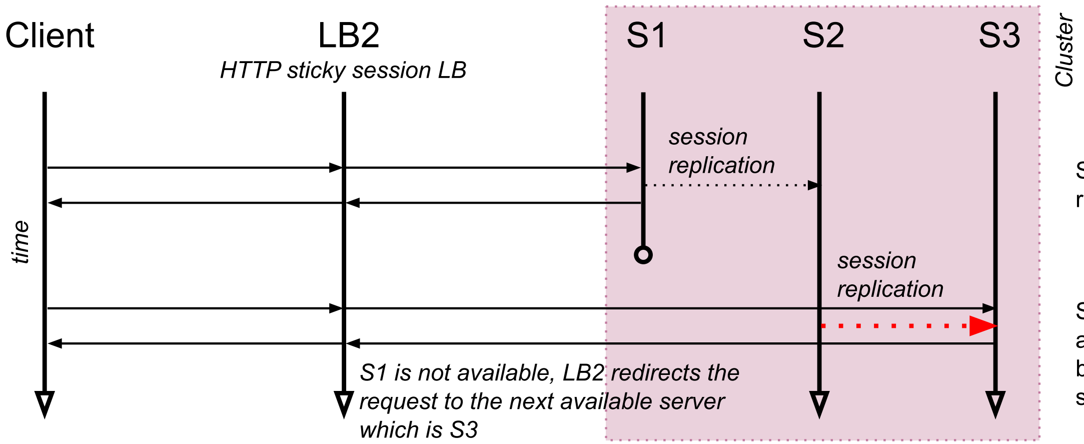
Scenario A: JSession-aware load balancer



S1 is primary, S2 is secondary; S1 replicates the session to S2

S1 fails, S2 becomes primary, S3 becomes secondary; LB1 directly redirects the request to S2 as it knows the secondary server from the first request.

Scenario B: HTTP sticky session load balancer



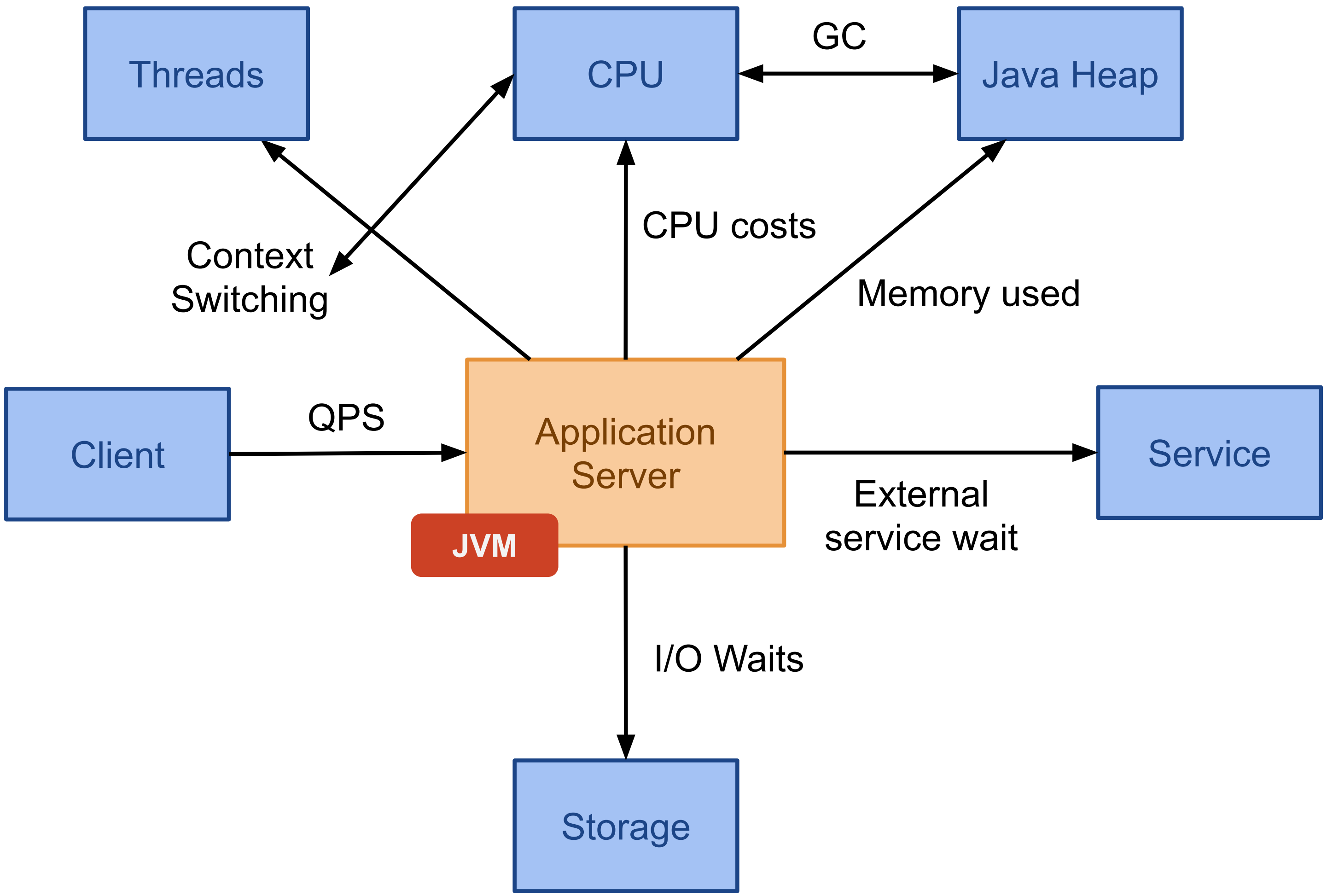
S1 is primary, S2 is secondary; S1 replicates the session to S2

S1 fails, S3 discovers that S2 has a session and gets the session data from it. S3 becomes primary and S2 becomes secondary.

Overview

- Definitions
- Load Balancers
- 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*