Performance, Scalability and High-availability of Enterprise Applications

Miroslav Blaško

miroslav.blasko@fel.cvut.cz

Winter Term 2017

Contents

Motivation

Motivation

There are applications for which it is critical to establish certain availability, consistency, performance etc.

- **How can we define/measure such non-functional application's requirements ?**
- **What techniques/tools can we use to provide such application ?**

Core concepts

Understanding Core Concepts

- **Mission-critical application** is an application that is essential to the survival of a business or organization, i.e. failure or interruption of the application significantly impacts business operations.
- Important properties of such application
  - How well it can be adapted to handle bigger amounts of work ? *(scalability)*
  - How well it provides useful resources over time period ? *(availability)*
  - What is rate of processing over specified workload and time period ? *(performance)*

Scalability of an application

- **Scalability** is property of an application which defines
  - how it can be easily expanded to satisfy increased demand for network, processing, database access, file-system resources etc.
  - how well it handles the increased amount of work
- There are 2 ways to scale an application
  - **horizontally (scaling out)** – expanding by adding new nodes with identical functionality to existing ones.
  - **vertically (scaling up)** – expanding by adding processor units, main memory, storage or network interfaces to a node.

Horizontal Scaling Example

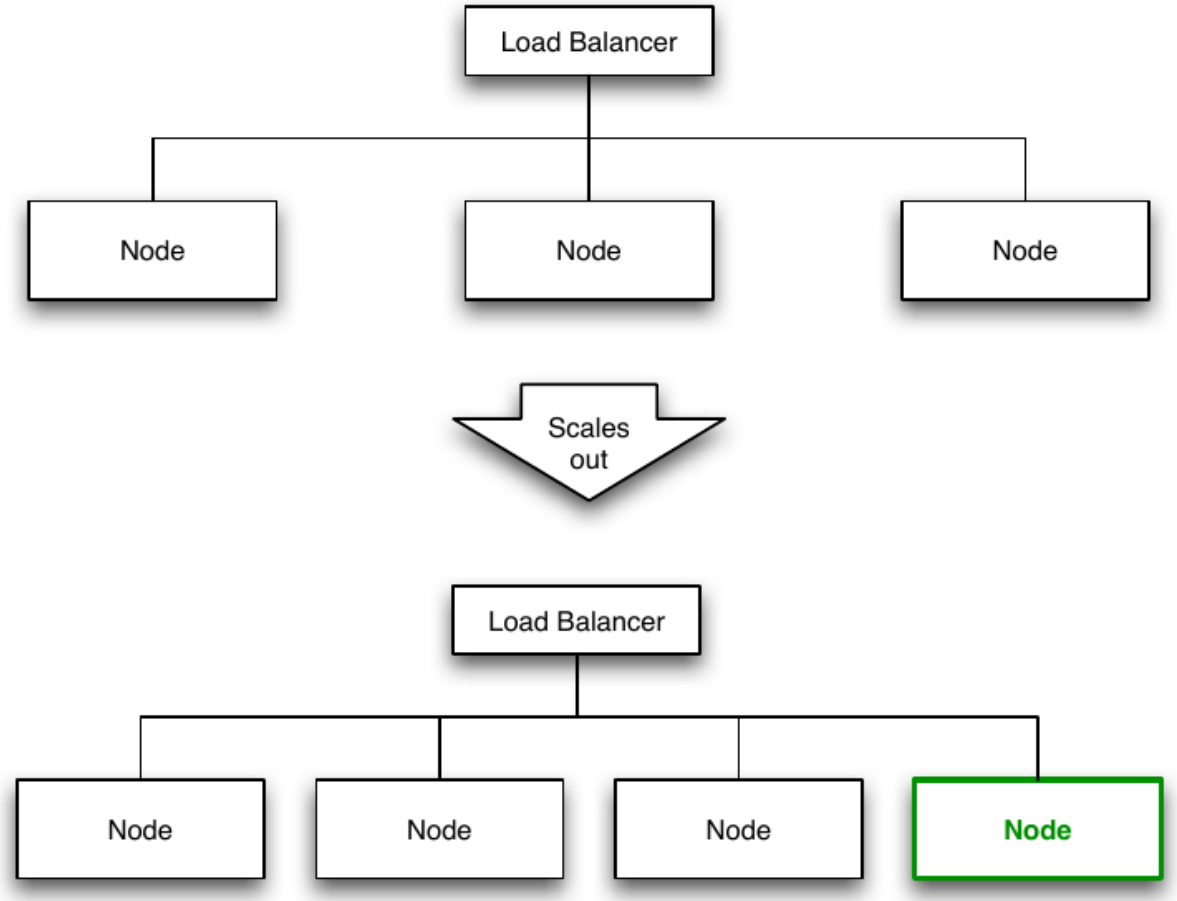

Figure: Clustering Example – horizontal scaling of SOA systems/web services by adding more servers nodes to a *load-balanced network* [1].

Vertical Scaling Example

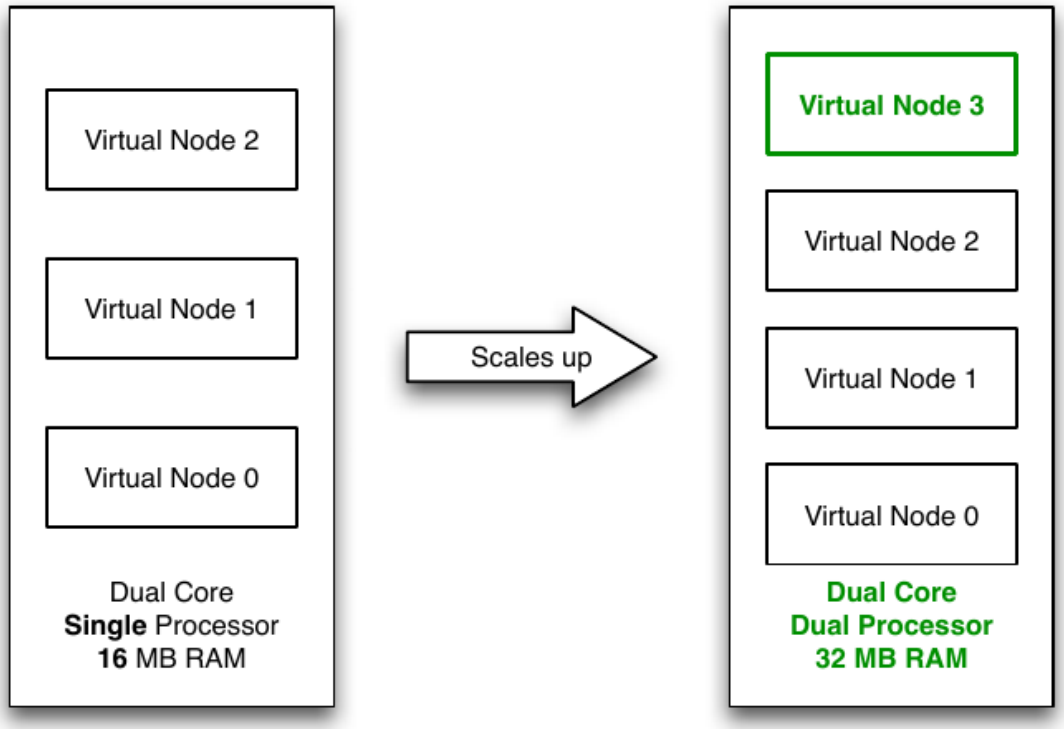

Figure: Virtualization Example – vertical scaling of hosting services by increasing number of processors, the amount of main memory to host more virtual servers [1].

High-availability of an application

- **Uptime**, **(downtime)** is time during which application is running (not running). Sometimes *uptime*, *downtime* is used to express its probability.
- **Availability** describes how well an application provides its assumed functions over particular time period, expressed in percentages (%) as $A = (1 - t_{uptime}/t_{unplanned\_downtime}) * 100$
- Note, that *uptime* and *availability* are different concepts.
- **High-availability** characterizes applications that is obliged to have availability close to 100 %.

Measuring availability

Availability	**Downtime per year**	**Downtime per week**	**Downtime per day**
90% ("one nine")	*36.5 days*	*16.8 hours*	*2.4 hours*
95%	18.25 days	8.4 hours	1.2 hours
97%	10.96 days	5.04 hours	43.2 minutes
98%	7.30 days	3.36 hours	28.8 minutes
99% ("two nines")	*3.65 days*	*1.68 hours*	*14.4 minutes*
99.9% ("three nines")	*8.76 hours*	*10.1 minutes*	*1.44 minutes*
99.99% ("four nines")	*52.56 minutes*	*1.01 minutes*	*8.66 seconds*
99.999% ("five nines")	*5.26 minutes*	*6.05 seconds*	*864.3 milliseconds*
99.9999% ("six nines")	*31.5 seconds*	*604.8 milliseconds*	*86.4 milliseconds*
99.99999% ("seven nines")	*3.15 seconds*	*60.48 milliseconds*	*8.64 milliseconds*

Table: Measuring Availability – vendors typically define availability as given number of "nines".

Service Level Agreement (SLA)

Service Level Agreement (SLA) defines obligations of involved parties in delivering and using an application e.g.

- minimal/target levels of availability
- maintance windows
- performance and metrics for its evaluation
- billing
- consequences for not meeting obligations

Techniques

Load balancing

- **Response time** defines amount of time system takes to process a request after it has received one. In remote calls (e.g. web services) it's often used term **latency** referring to response time lowered by processing time of the request on a server.
- **Throughput** defines number of transactions per second that application can handle.
- **Load balancing** is a technique for minimizing *response time* and maximizing *throughput* by delegating requests among multiple nodes.
- *Load balancer* is responsible for routing requests to available nodes based on scheduling rules.

Load Balancer

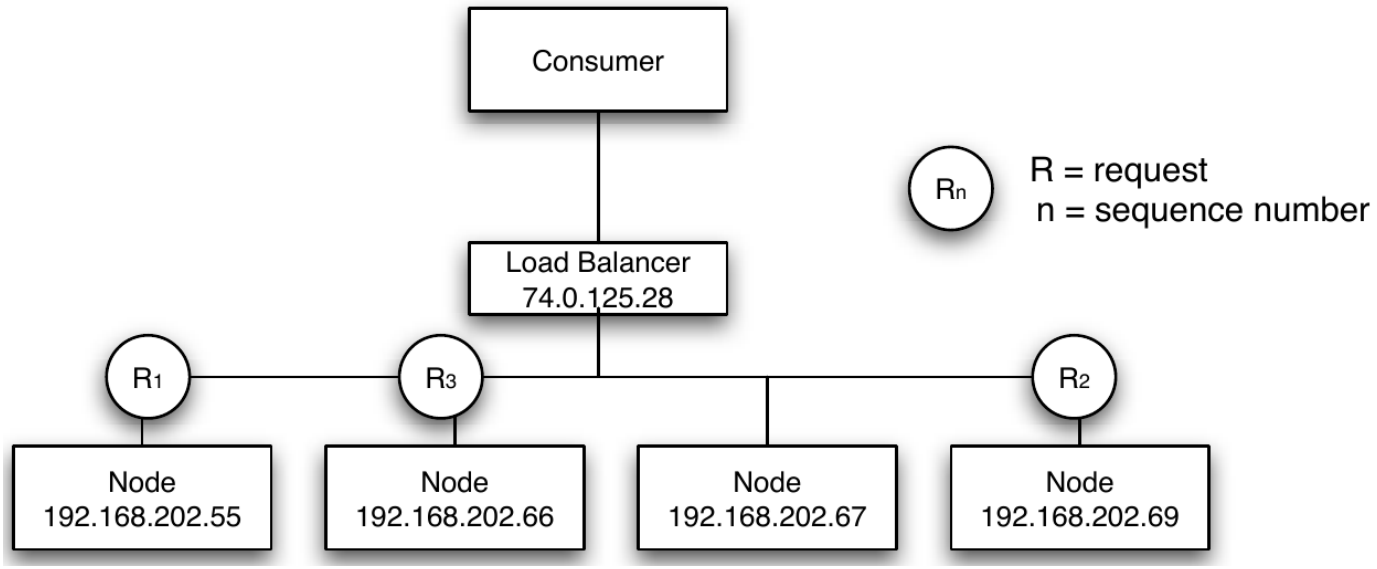

Figure: Load Balancer. It uses scheduling rules to decide which request will be served by which node [1].

Sticky Load Balancer

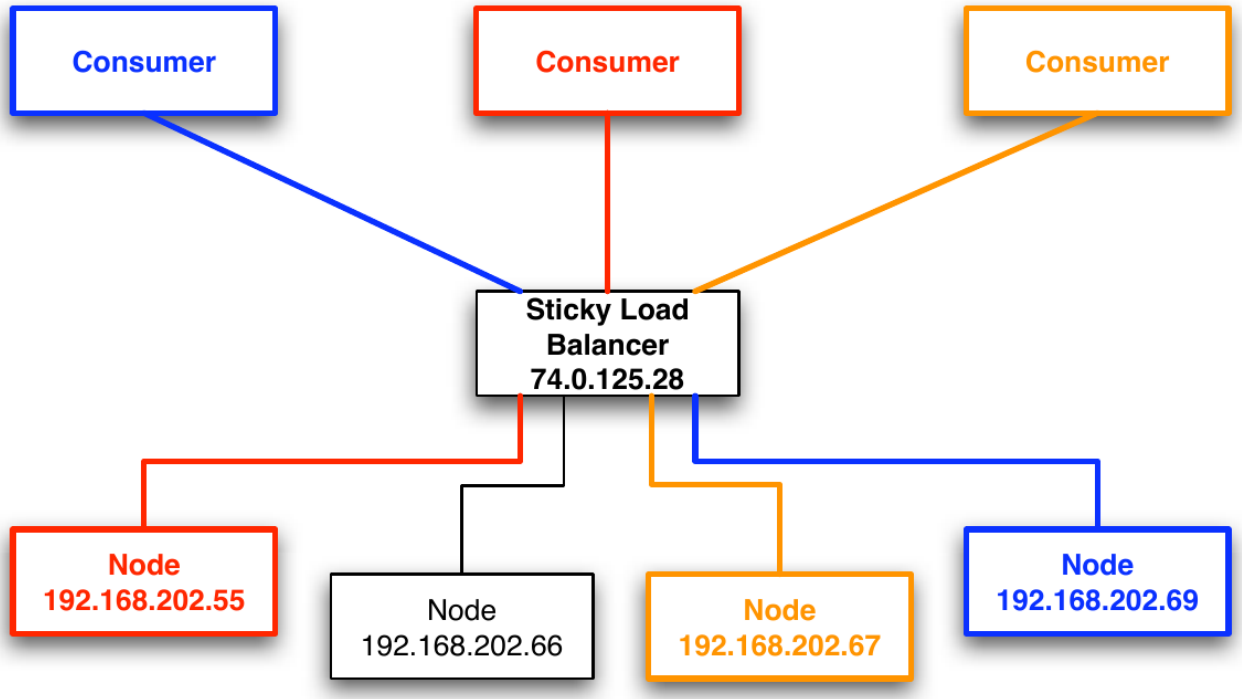

Figure: Sticky Load Balancer – stateful applications require persistent/sticky load balancing, where consumer is guaranteed to maintain a session with specific node [1].

Common Features of Load Balancers

- *asymmetric load distribution* – different loads are assigned to different nodes
- *priority activation* – if loads gets too high, some standby nodes are activated
- *content filtering* – modifies traffic on the way through
- *firewalling* – deciding whether traffic might pass through an interface or not base on security rules
- *TCP buffering* – buffer responses from servers for slow clients

Caching

Caching is a technique for sharing data among multiple data consumers. It is useful for data that are expensive to compute or fetch. E.g. stateful load balancing requires data sharing among the service providers.

- implemented by index tables where *key* is used to retrieve cached entry (datum)
- query for datum using cache can lead to *cache hit* or *cache miss*
- *cached data* can be refreshed according to different *policies*
  - *write-through* – a synchronous write
  - *write-behind* (write-back) – updated only if dirty datum is requested
  - *no-write allocation* (write-around) – only reads are beeing cached

Write-through with No-write Allocation

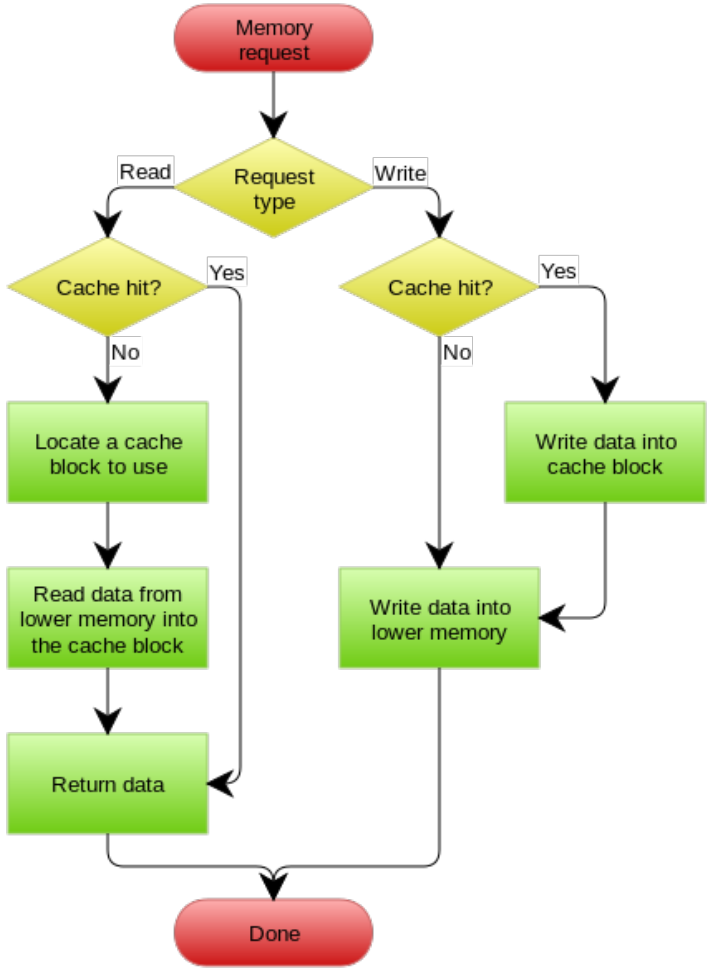

Figure: A write-through cache with no-write allocation taken from `https://en.wikipedia.org/wiki/Cache_(computing)`

Write-behind Cache with Write Allocation

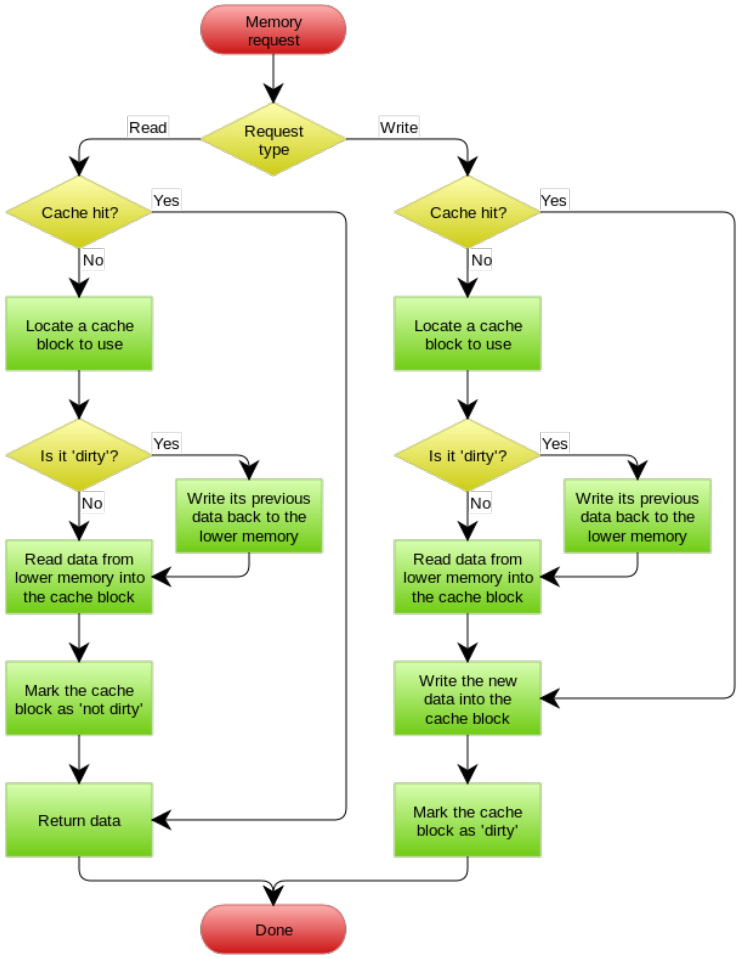

Figure: A write-behind cache with write allocation taken from `https://en.wikipedia.org/wiki/Cache_(computing)`

Caching types

- **application cache**
  - implicit vs. explicit application caching – with little/no participation of a programmer (e.g. Terracotta) vs. using caching API (e.g. memcached)
- **web cache**
  - *client side* (browser) vs. *server side caching*
  - *web-accelerators* – operates on behalf of the server of origin (e.g. content distribution networks, Akmai)
  - *proxy caches* – serve requests to a group of client accessing same resources. Used for content filtering and reducing bandwidth usage (e.g. Apache)
- **distributed cache** – implemented across multiple systems that serves requests for multiple customers and from multiple resources (e.g. distributed web cache Akmai, distributed application cache memcached)

Clustering

- **Cluster** is group of computer systems that work together in a form that appears from the user perspective as a single system.
- *Load-balancing cluster* (Active/Active) – distributes load to redundant nodes, while all nodes are active at the same time offering full-service capabilities
- *High-availability cluster* (Active/Passive) – improves service availability by redundant nodes eliminating single points of failures.

Load-Balancing Cluster

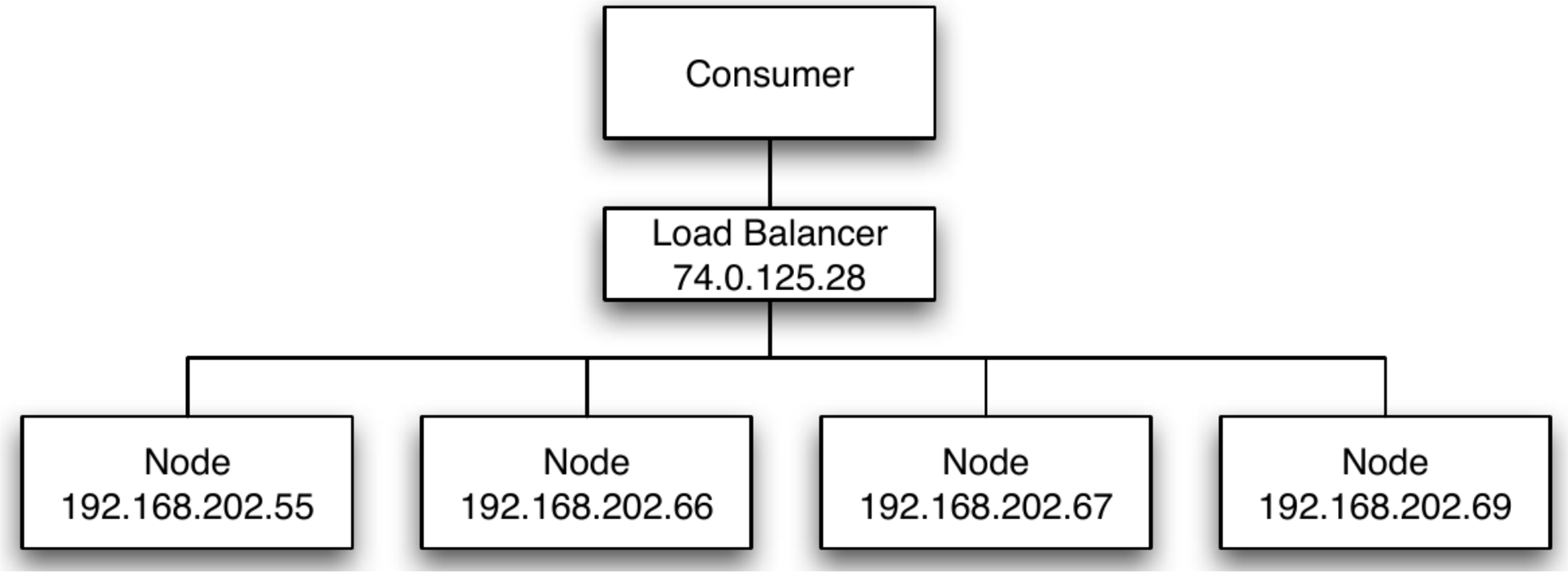

Figure: Load-Balancing Cluster (Active/Active) [1]

High-Availability Cluster

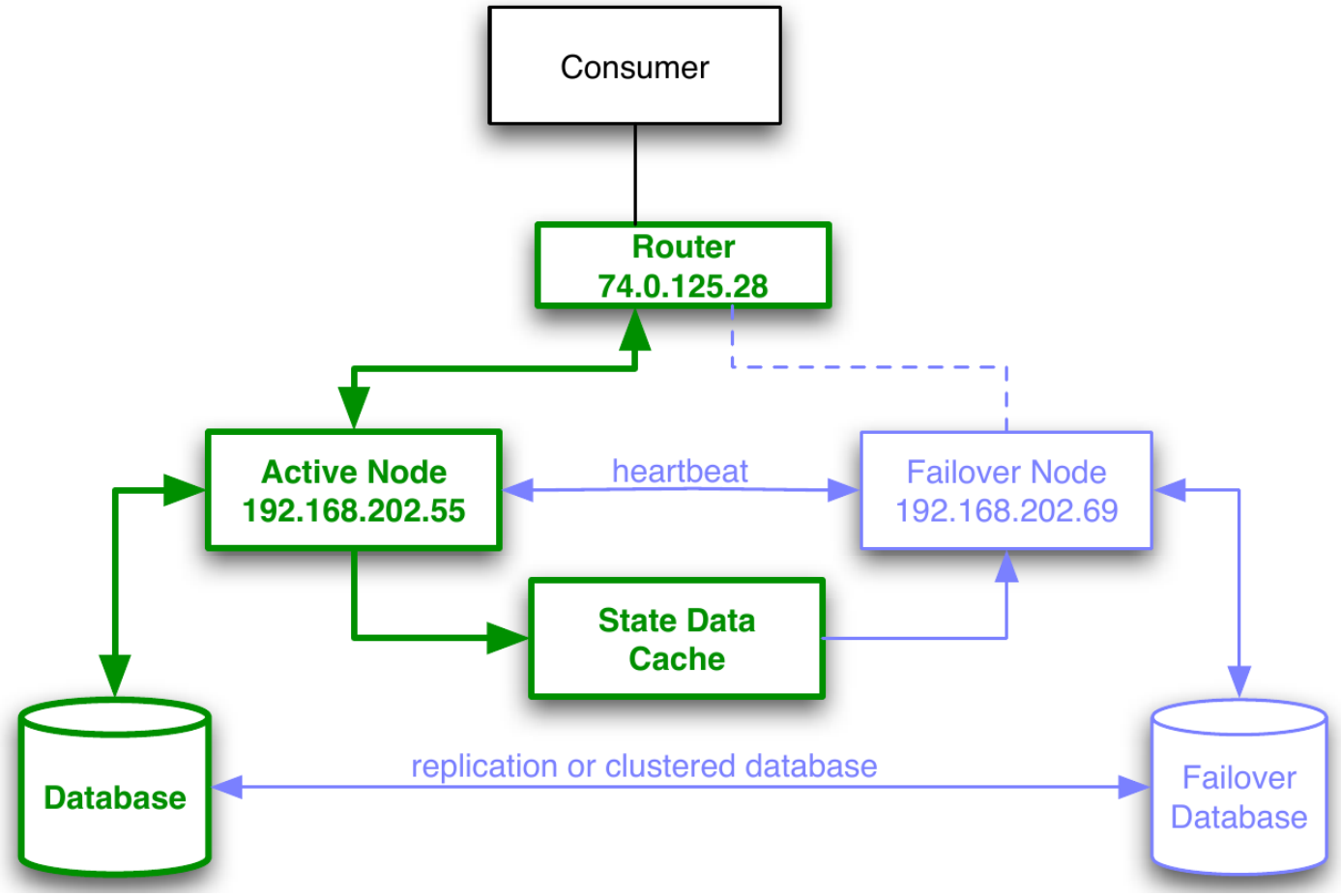

Figure: High-Availability Cluster (Active/Passive) [1]. It uses "heartbeat" to detect if nodes are ready and routing mechanism to switch traffic if a node fails.

Principles to Achieve High Availability

- Elimination single points of failure – adding redundancy so failure of a component does not cause failure of the entire application
- Reliable crossover – ability to switch to from failing node to new node without loosing
- Detection of failures as they occur – failing node should maintain activity, not user's attention.

Cloud Computing

- **Cloud Computing** is a type of internet-based computing where applications are running on distributed resources owned and operated by a third-party.
- Typically used for end-user applications
- Service models within cloud computing :
  - Software as a Service (SaaS) – using providers application with limited control over the application e.g. CRM, emails, virtual desktop
  - Platform as a Service (PaaS) – using providers services, libraries, tools with control over deployed application e.g. execution runtime, database, web-server, development
  - Infrastructure as a Service (IaaS) – control over operating system but not underlaying infrastructure e.g. virtual machines, servers, load balancers, network

System performance testing

- **Performance** refers to application throughput with specified workload and period of time.
- Performance specifications are typically documented in SLA document
- Troubleshooting performance issues requires multiple types of testing such as
  - *endurance testing* – identifies resource leaks under the continuous, expected load
  - *load testing* – show application behavior under a specific load
  - *spike testing* – shows application behaviour under dramatic changes in load
  - *stress testing* – identifies the breaking point for the application under dramatic load changes for extended periods of time

Tools

Tools for critical-mission applications

- Spring/JSR-107 Cache API (java libraries)
- Netbeans Profiler, Intellij Idea Profiler or VisualVm (profiling)
- Apache JMeter or Gatling (performance testing by scripts)
- Apache Server (load balancing and high-availability)

Demo application

Spring Cache Abstraction

- @Cacheable – triggers cache population
- @CacheEvict – triggers cache eviction
- @CachePut – updates the cache without interfering with the method execution
- @Caching – regroups multiple cache operations to be applied on a method
- @CacheConfig – shares some common cache-related settings at class-level

Caching with Spring vs. JSR-107 annotation

Spring	**JSR-107**
@Cacheable	@CacheResult
@CachePut	@CachePut
@CacheEvict	@CacheRemove
@CacheEvict(allEntries=true)	@CacheRemoveAll
@CacheConfig	@CacheDefaults

Table: Alternative annotations within Spring and JSR-107

Experiments Reporting Tool Application

- VisualVM profiling
- Intellij Idea Memory Monitor plugin
- JMeter load testing
- Testing cache

The End

Thank You

Resources

1. E. Ciurana, Scalability & High Availability, 2009
   https://dzone.com/storage/assets/4333-rc043-010d-scalability_3.pdf
2. Spring Framework Reference, http://docs.spring.io/spring/docs/current/spring-framework-reference