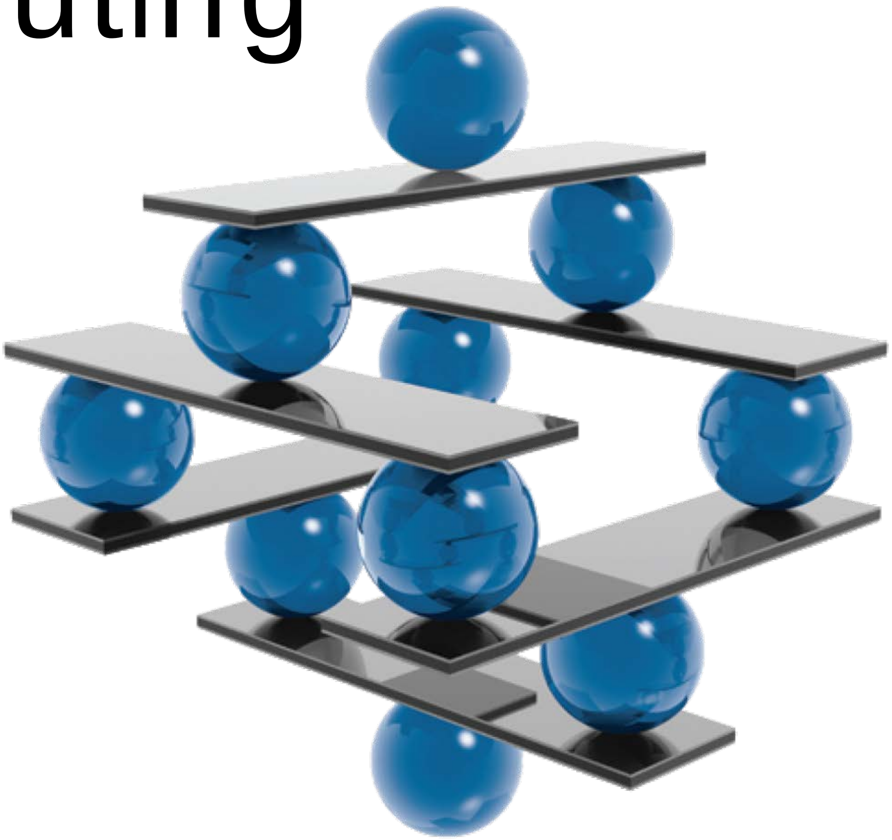
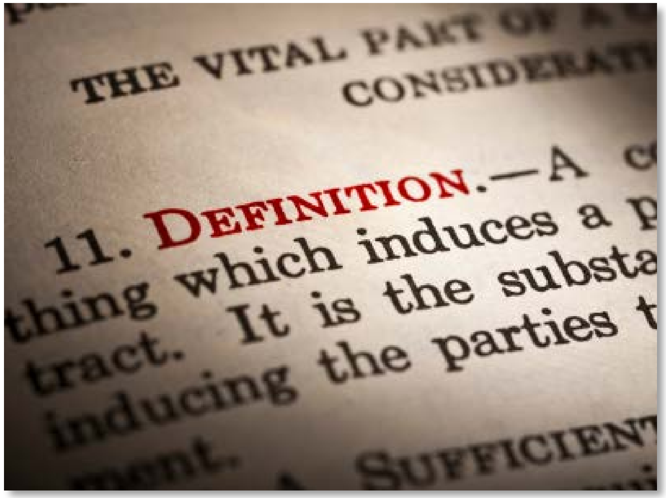
Peer-to-Peer and Cloud Computing



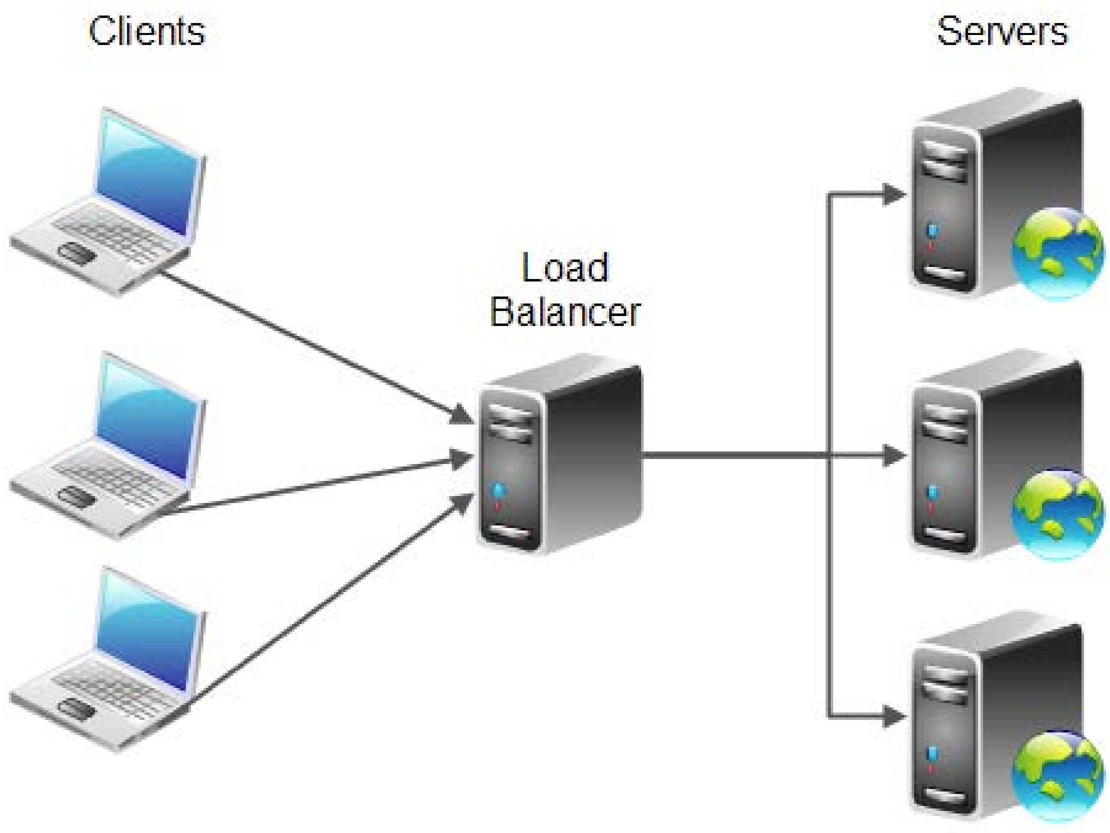
Load Balancing Concepts

1. Introduction
2. Naive Approaches in a Nutshell
3. Honey Bee-inspired Load Balancing
4. Summary



What is Load Balancing?

- Distribution of **workload** (tasks, requests, jobs) among available **resources** (i.e. servers, allocation-units).
- Resolve imbalance: equally **distribute** load among available resources.
- User/Client is re-directed to a server that is currently capable of serving his **request.**
- User/Client does not notice re-direction and thinks that he is communicating with one single server throughout the entire session.


Clients
Load
Balancer
Servers

Why do we need Load Balancing?

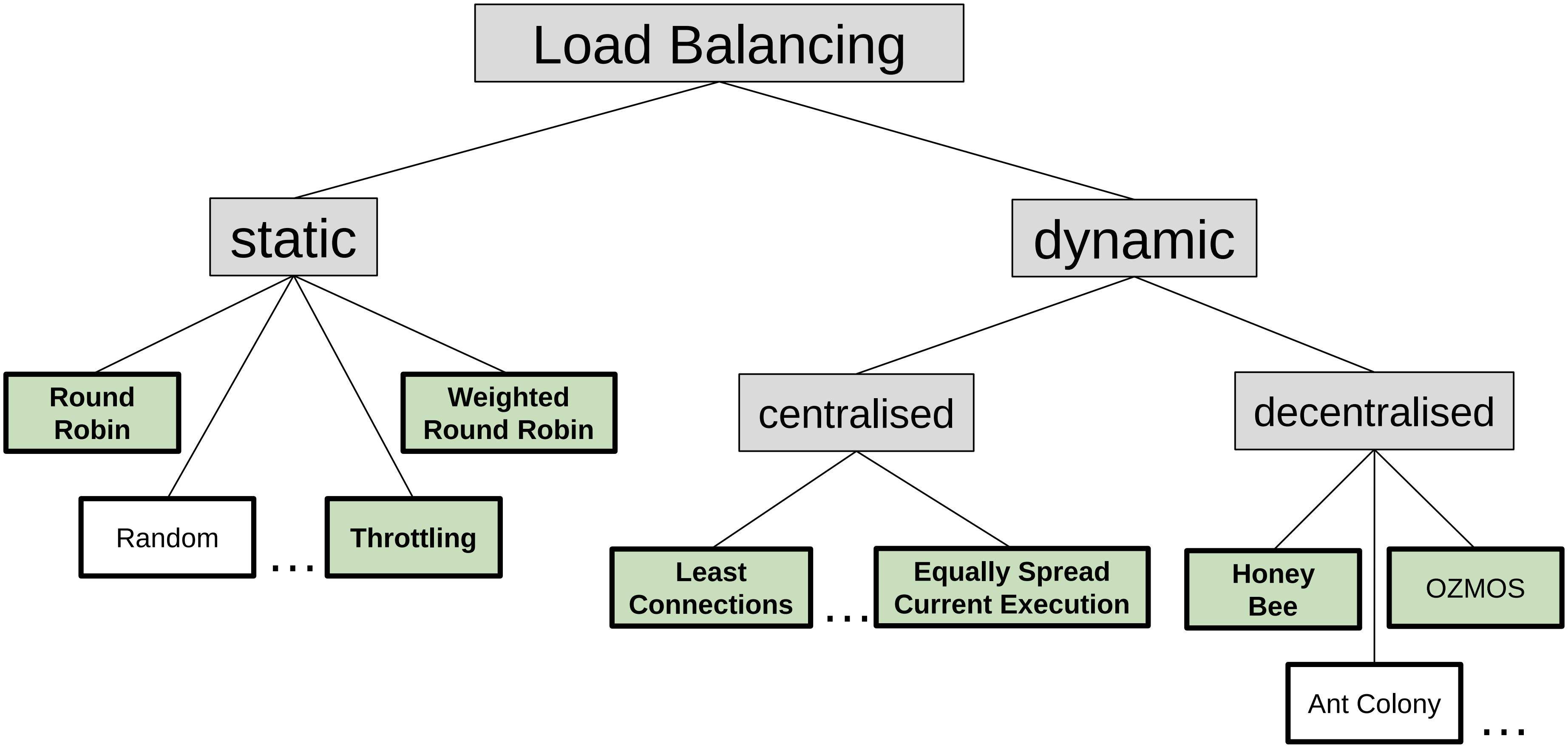
- **Quality of service**: e.g. availability, reliability, robustness, performance
- **No SPoF**: load will be assigned to alternative servers in advance
- **Cost efficiency**: avoidance of idling servers (in contrast to one or more overloaded servers)
- **Faster response:** increased overall throughput (think of user-experience)
- **Flexibility**: new hardware can be added easily

Challenges

- **Uncertainty**: amount of incoming tasks (e.g. HTTP-requests)
- Heterogeneous resources/servers
  - Varying performance and capabilities of servers
  - Different classes of tasks: not every server is capable of serving any request
- Scalability of the distributed systems (Grids, Clouds, P2P) must not be limited, e.g. by an overloaded central Load Balancer → SPoF

- Not any of the existing approaches meet the requirements resulting from the challenges mentioned above.
- Some of them are of static nature → do not consider the current situation in terms of actual workload.
- **Dynamic** and **decentralised** approaches have been investigated and are still topics of current research.

Classification of Load Balancing Algorithms



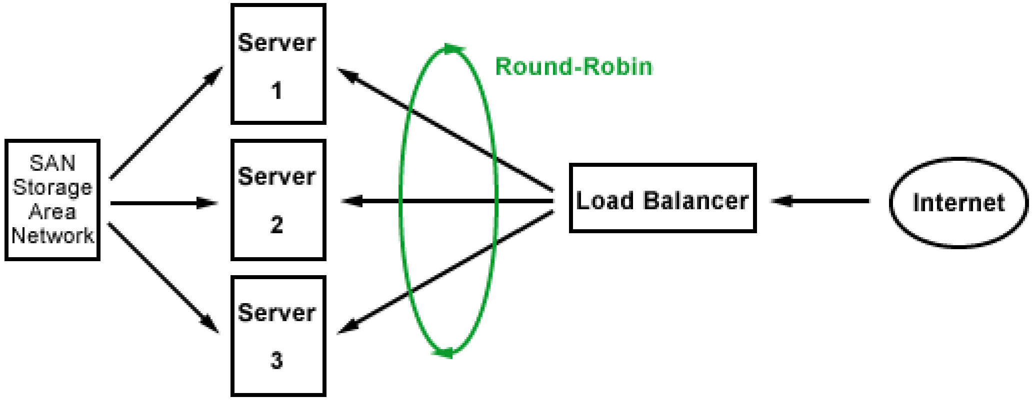
* We will have a look at the highlighted ones

1. Introduction
2. Naive Approaches in a Nutshell
3. Honey Bee-inspired Load Balancing
4. Summary

Static Approach: Simple Throttling

- **Servers** are assigned a limit/threshold for a maximal accepted load
  - Number of active connections
  - Number of processing requests
  - etc.
- **Load Balancer** maintains a list of all available allocations units (i.e. servers) and their current state.
- New request
  - Load Balancer traverses list
  - Selects first server whose threshold has not exceeded.
- Request is assigned to that allocation unit.

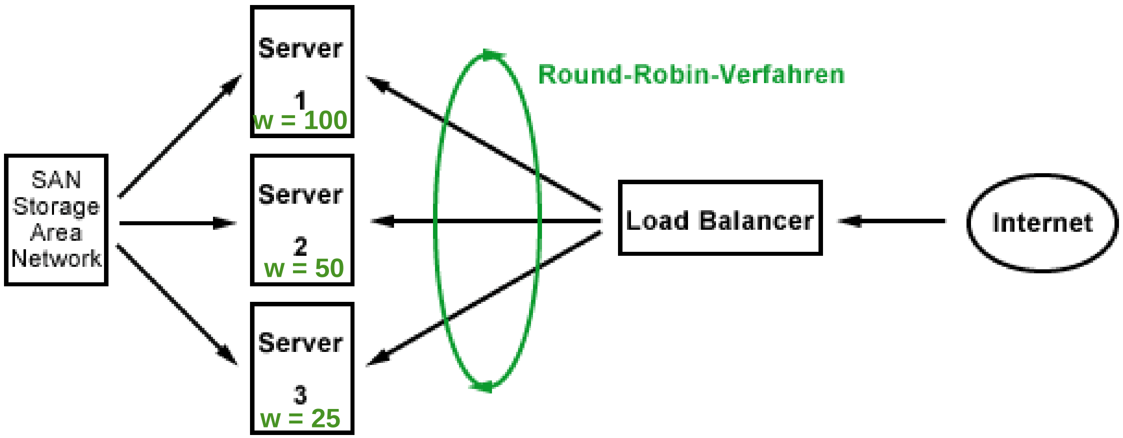
- Sequential distribution of incoming requests among available servers
- Sequence is **fixed** and **not dependent on the current load** of the incorporated servers →static
- Servers should all provide **identical** (web) services and be equipped with **equal** computational capacity
- Under this precondition, **round robin** is a simple and effective method for balancing load originating from incoming requests



Source: Elektronik Kompendium

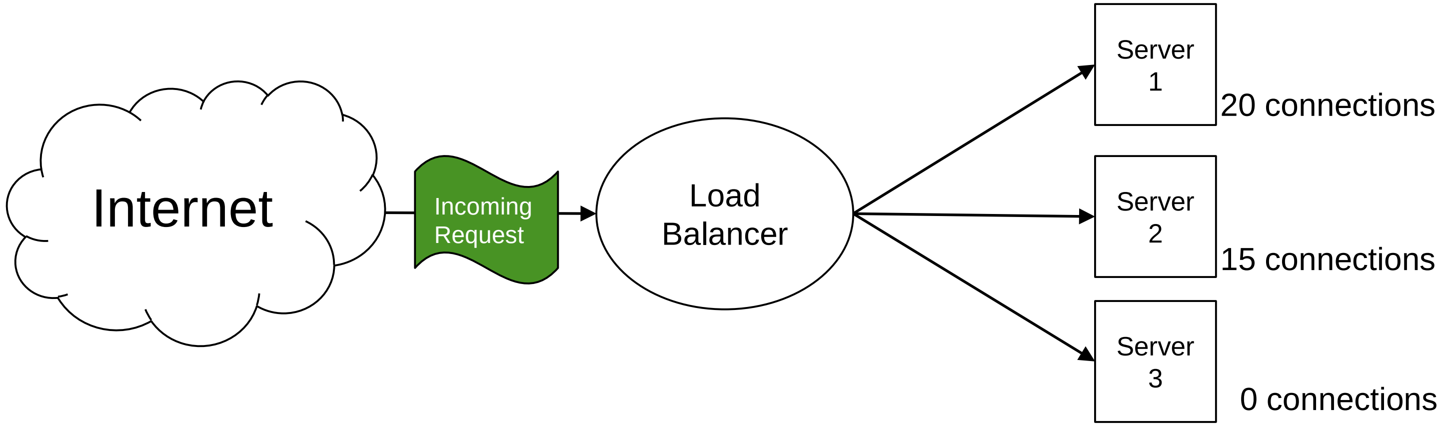
Static Approach: Weighted Round Robin

- Overcomes the drawback of the conventional approach.
- Useful if the servers do not all have the same capacity.
- Administrator can **pre-assign the capacity** of available servers in terms of a **weighting**, i.e. the server with the highest weight is the most efficient one.
- Thus, Server 1 always is assigned two consecutive requests before Server 2 gets his first one for instance.



Source: Elektronik Kompendium

- Considers the amount of currently **active connections** on each participating server.
- An incoming request is scheduled to the server with the smallest number of active connections.
- **Same preconditions** as for the Round Robin approach should be met for this method.



Dynamic centralised approach: Equally Spread Current Execution

- **Avoid load imbalance** among the available servers
- Requests are migrated to other servers with less workload.
- Incoming requests are **queued** and spread around all available servers to achieve a **nearly equal load** among all allocation units.

- Imaginable drawbacks
  - **Migration costs** for re-allocating requests from one server to another.
  - **Complex requests** (e.g. time-consuming computations on the server-side): migration might be difficult or not possible at all due to interrupting and purging at the old location as well as setting up at the new location.

1. Introduction
2. Naive Approaches in a Nutshell
3. Honey Bee-inspired Load Balancing
4. Summary

Motivation

- Server farms, or rather **Internet-server colonies** may co-host multiple web services
- Customers purchase **resources** offered by providers of such Internet-server colonies
- Problems
  - limited **number of available servers**
  - **unpredictable amount of HTTP requests** to reach the hosted web-services
- **Optimised server allocation policy** is a challenging task from the provider‘s point of view

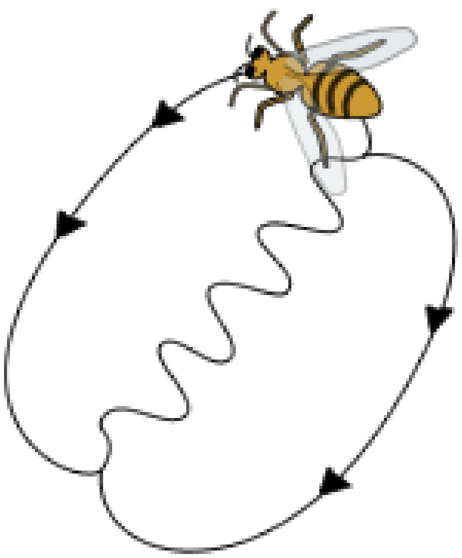
Goal: Maximise provider‘s **revenue** and satisfy customer‘s **demand** by allocating available resources efficiently

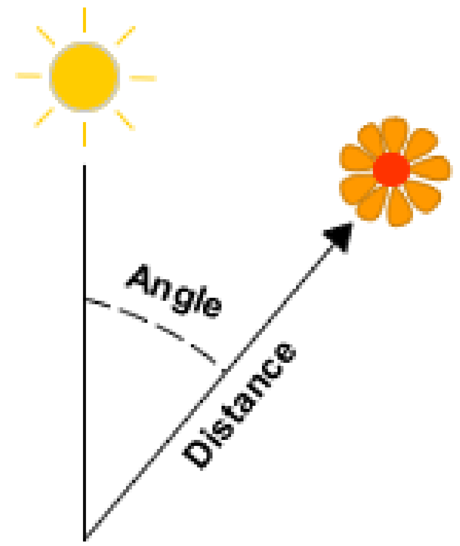
- Nature-inspired approaches
- Swarm intelligence definition:
  – *Collective execution of primitive interactions amongst individuals (viewed at the microscopic abstraction level), leads to a complex global (macroscopic) behaviour that could not be accomplished by any single member of the group/system.*
- Swarm intelligence can be characterised by
  1) Self-Organisation → decentralised and unsupervised
  2) Adaptiveness → dynamically varying environment
  3) Robustness → Accomplishing global goal even in the presence of faulty acting individuals
- Examples: Ants, bees, termites, wasps

Honey Bees Collecting Food/Nectar (1)

- Scenario of swarm intelligence observed from honey bee colonies
- **Forager bees:** dynamically allocated in a self-organised manner to collect nectar from food sources
- Allocation with regard to the **profitability** of the food sources (nectar quality, distance to hive,…)
- Forager bees provide **feedback** regarding the profitability of the visited food sources
- Additionally: *scout bees* **explore** for new flower patches
- Feedback mechanism is called *waggle-dance*

Waggle dance





- The length of the waggle-dance is composed of two factors:
  1) A *response threshold* determined by the current nectar flow into the hive (known from interaction with receiver bees)
  2) A *profitability rating* of the advertised flower (a function of nectar-quality, -bounty and distance to the hive)
- Inactive forager bees observe a waggle-dance and decide whether to follow the dancer to the advertised flower patch or not
- Observing foragers do <u>not</u> acquire global knowledge by means of observing every waggle dance, but rather select a certain waggle-dance randomly

- Analogy from nature: Allocate *forager bees* among food sources (flower patches)

Bee colony	Server colony
Forage site (flower patch)	Service/Request queue
Flower patch volatility	unpredictable HTTP request
Forager bee	single server / allocation unit
Forager group collecting from one source	Virtual server serving specific HTTP request / service queue
Bee colony	ensemble of servers
Forager round trip time	time cost
Nectar quality	value-per-request-served

Remember: Co-hosting of several web services

- Model
  - M groups of n servers called *virtual servers* $VS_j, \quad j = 0, \dots, M-1$
  - M *service queues* Q_j facing certain type of HTTP request
  - Server $s_i \in VS_j$ serving Q_j is paid c_j cents per served request

- Honey Bee-inspired server allocation
  - $s_i \in VS_i$ posts advert with duration $D = c_j \cdot A$ with probability p after it has finished serving the last request
  - A denotes an *advert scaling factor* → constant
  - Any server s is either a *forager* s^f or a *scout* s^s

After specific time interval T_i.

- Honey Bee-inspired server allocation (continued)
  - If server is forager s_i^f it reads an advert randomly with probability r_i
  - If server is scout s_i^s it selects a virtual server $VS_j, j = 0, \dots, M-1$ randomly
  - A server s_i determines its profitability by comparing its own *profit rate* P_i with the *overall profit rate* of the colony P_{colony} (by means of a Lookup-Table, see below)
    - $P_i = \frac{c_j R_i}{T_i}$, where R_i denotes the number of requests served by s_i and T_i is a certain time interval
    - $P_{colony} = \frac{1}{T\_colony} \sum_0^{(M-1)} c_j R_j$, where R_j denotes the total number of requests served by VS_j and T_{colony} again is a given time interval

Profit Rate	P[Reading] r_i
$P_i \leq 0.5 P_{colony}$	0.60
$0.5 P_{colony} < P_i \leq 0.85 P_{colony}$	0.20
$0.85 P_{colony} < P_i \leq 1.15 P_{colony}$	0.02
$1.15 P_{colony} < P_i$	0.00

<< Lookup Table for dynamic assignment of r_i

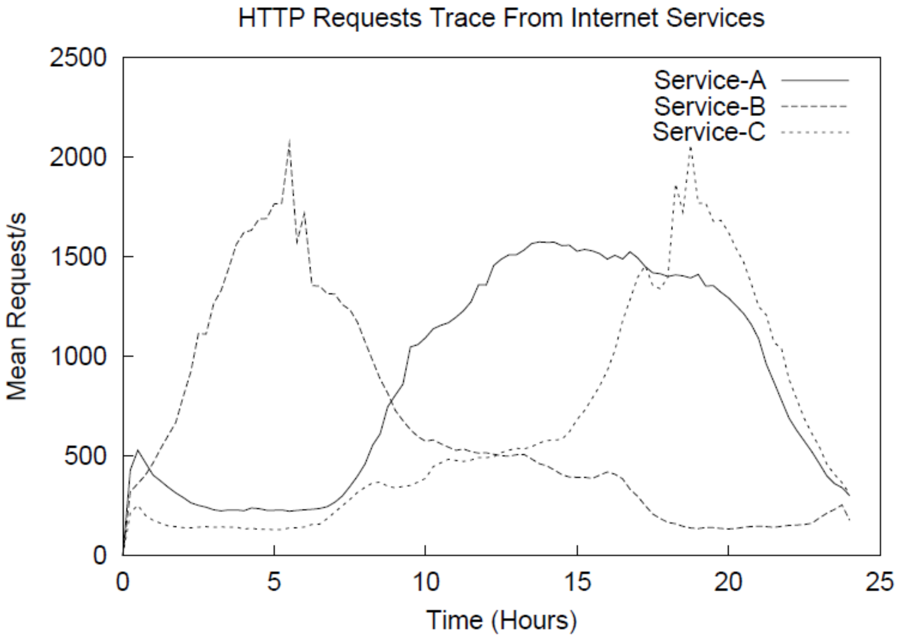
[NaTo03]

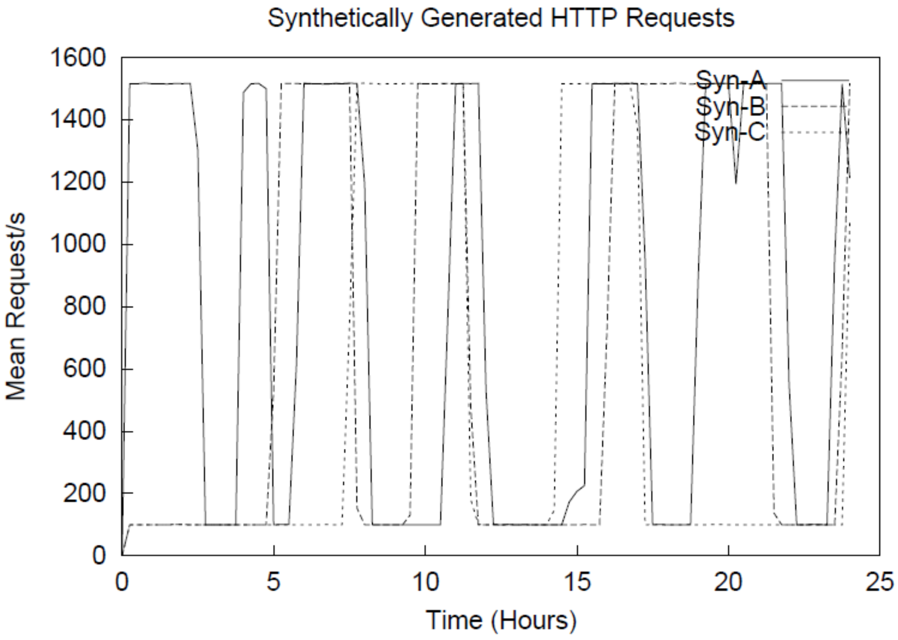
- Honey Bee-inspired server allocation (continued)
  - Resulting effect: a server is more likely to read an advert (observe a waggle dance) when it serves a low-profit queue Q_j
  - Approach resembles the behaviour of real honey bee colonies
  - with the difference that here a server may skip reading an advert $(r_i = 0)$,
  - whereas in real bee colonies the foragers decide to dance or not
  - depending on global knowledge regarding the hive‘s *nectar-intake-rate* and the time until a receiver bee is available (response threshold).
- Aims in maximising the provider‘s revenue rather than minimizing the load of particular servers.
- With slight modifications for the profit rate definition P_{colony} the latter objective should be easily accomplishable.

Evaluation Scenario

- Comparing performance of honey-bee algorithm with three benchmark algorithms: *Omniscient*, *Greedy* and *Optimal-Static* configuration
- **Real HTTP-request traces** from a commercial service provider as well as synthetic traces following an inhomogeneous Poisson process come into operation
- Experiments conducted for Internet Server Colony with **2 and 3 virtual servers** (i.e. different web-services)
- Virtual servers are composed from a total of **50 physical servers**
- Performance metric: **Total revenue** earned by the entire server colony

HTTP-request arrival patterns





[NaTo03]

Load Balancing based on Honey Bee Allocation: Evaluation (2)

- **Omniscient**
  - **Upper bound** on possible profitability
  - Informationally impossible and computationally prohibitive in practise
  - Computed by means of dynamic programming (**complete knowledge** required)

- **Greedy**
  - Standard **heuristical** approach
  - „What would have been optimal during the preceding time period“
  - Reallocate servers based on the optimal profit that could have been achieved given the HTTP-arrival pattern from the **past time-interva**l

- **Optimal-static** (server re-allocation takes place only seldom)
  - Using the best from among all static (fixed) allocations
  - Many SLA providers do not change their allocations more than once a month
  - Choose the best out of **k possible server allocations** for a time-horizon split into N discrete time-steps

- Experimental results for synthetically generated HTTP-requests

Revenue ($) for synthetically generated HTTP requests		
Algorithm	**2-VS**	**3-VS**
Omniscient	970.636	1.119.400
Honey Bee	868.949	1.003.050
Greedy	836.077	968.087
Optimal-Static	810.510	860.363

[NaTo03]

- Honey Bee algorithm shows good adaptability to varying HTTP-request arrival patterns
- However, for a low variability Greedy approach outperforms Honey Bee algorithm by approx. 1.2 %

Revenue-per-served-request: c_j = 0.5 cent
Simulation duration: 24 h period

- Experimental results for real internet service provider traces

Revenue ($) from real Internet service traces		
Algorithm	**2-VS**	**3-VS**
Omniscient	1.066.440	1.336.960
Honey Bee	1.050.110	1.238.470
Greedy	1.043.400	818.040
Optimal-Static	844.822	1.108.360

[NaTo03]

- Honey Bee algorithm again outperforms Greedy and Optimal-Static approach and performs within 1.55% (for 2-VS) respectively 7.95% (for 3-VS) of the Omniscient

Revenue-per-served-request: c_j = 0.5 cent
Simulation duration: 24 h period

- Introduction
- Naive Approaches in a Nutshell
- Honey Bee-inspired Load Balancing
- Summary

Summary

- A brief overview of **Load Balancing strategies** was presented
- Some simple and basic approaches were sketched.
- Additionally, a **self-organised and nature-inspired** method was discussed in more detail.
- Provide a basic understanding of the **necessity** of Load Balancing and an insight how this important challenge can be achieved.
- Load Balancing is a broad research field that could easily fill a designated lecture considering the underlying network technologies and specific implementations in more detail.

Literature

[NaTo03] Sunil Nakrani and Craig Tovey. 2003. On Honey Bees and Dynamic Server Allocation in an Internet Server Colony. To appear in *Proc. 2nd International Workshop on the Mathematics and Algorithms of Social Insects*, Atlanta, Georgia, USA. December 15-17, 2003

[NaTo04] Sunil Nakrani and Craig Tovey. 2004. On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers. *Adaptive Behavior - Animals, Animats, Software Agents, Robots, Adaptive Systems* 12, 3-4 (September 2004), 223-240.

Load balancing figure taken from

http://tutorials.jenkov.com/software-architecture/load-balancing.html