Software Architecture

Distributed Architecture

Types of Distributed Architecture Style

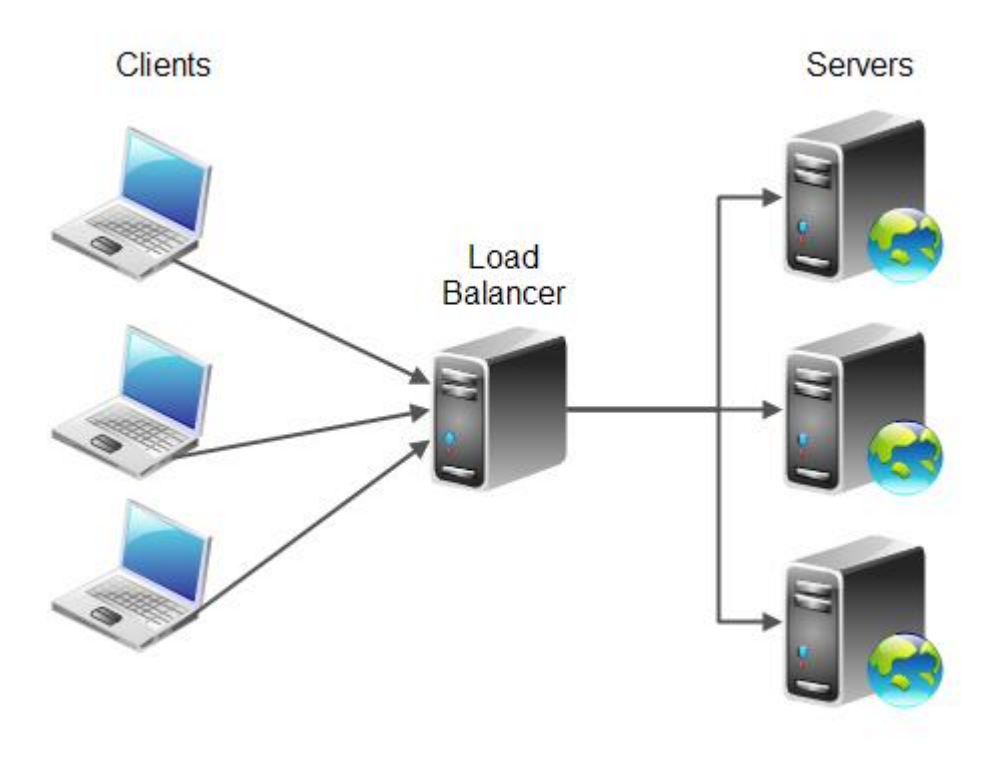
- Client-server
- Multitier
- Proxy
- Dispatcher (Load Balancer)
- P2P
- Broker
- Service-oriented architecture
- Microservice architecture

Dispatcher (Load Balancer) Architecture Style

Eunmi Choi
Kookmin University

Dispatcher(Load Balancer) Architecture Style

- Synopsis
  - a middleware architecture used in distributed computing to make a server farm appear to clients as a single server so that the server farms achieve high scalability and high availability through server load balancing



서버들의 군을 이루어주는 마스터 - 슬레이브 구조에서
마스터에 디스패쳐를 올려두면 서버들의 군(Server Farm,동종의 서버들이 군집)에 리퀘스트를 전달해줌.

클라이언트 입장에서는 싱글 시스템 이미지(외부에서 보면 서버가 하나인 것 같아 보임)로 접근할 수 있는 구조임.

서버 팜 구조는 스케일 아웃이 잘 되고 서비스가 항상 가용한 상태를 유지할 수 있음.

모든 서버들이 밸런스 있게 움직임.
운영을 여러개를 하면서 필요한 리퀘스트를 나누어 분산시켜서 처리하는 형태임!

Dispatcher(Load Balancer) Architecture Style: Context

- A single server that handles all the incoming requests may not have the capacity to handle high traffic volumes once the business becomes popular.
  - In such a situations companies often start to *scale up*: they upgrade the existing infrastructure by buying a larger box with more processors or add more memory to run the applications.
  - Scaling up, though, is only a short-term solution. And it's a limited approach because the cost of upgrading is disproportionately high relative to the gains in server capability.

스케일 업은 시스템을 업그레이드 시키는 것. 하나의 서버에서 하이 트래픽을 처리할 수 없으면 스케일 업을 하는 경우가 있는데 이거는 좋아지긴 하지만 6개월 정도밖에 효과가 없음. 더 좋은게 나오니까. 그래서 단타적인 것.

- Scaling up, though, is only a short-term solution. And it's a limited approach because the cost of upgrading is disproportionately high relative to the gains in server capability.
  - For these reasons most successful Internet companies follow a *scale out* approach.
  - Application components are processed as multiple instances on server farms, which are based on low-cost hardware and operating systems. As traffic increases, servers are added.

d ㅋ 러개를 붙여서성능을 받쳐 주는것. 스케일어빌리티는 스케일아웃에 대한것이다.

Dispatcher(Load Balancer) Architecture Style: Forces

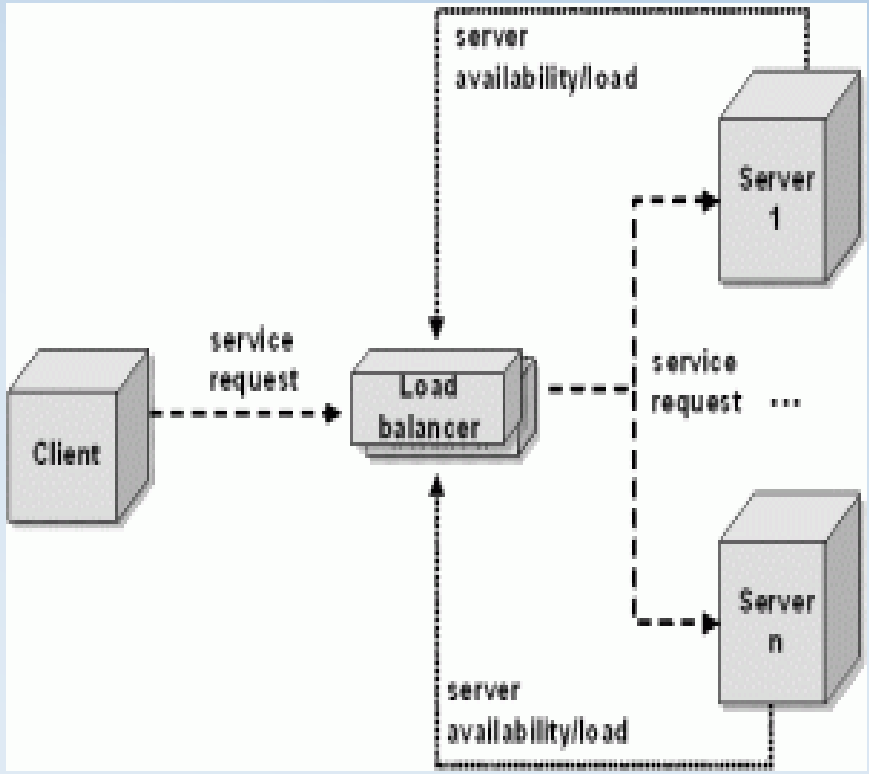
- On the software side, you must design applications so that they can run as multiple instances on different servers.
  - You do this by splitting the application into smaller components that can be deployed independently.

    디스패쳐를 서버에서 쓸때 고려해야 하는 것은?
    HTTP 를 예로 들자면, 세션에 대해 유지하는게 없음(Stateless). 어플리케이션의 세션을 유지하는게 필요함.
    Stateful하게 하기 위해 세션을 유지하게 하는게 필요함!
- A more challenging problem arises when the application components are stateful with application-session data.

- On the infrastructure side, the processing load must be distributed among the group of servers (Server Load Balancing).

Dispatcher(Load Balancer) Architecture Style - Solution

- Structure
  - In a widely used server load balancing architecture, the incoming request is directed to a dedicated server **load balancer** that is transparent to the client.
    - Based on parameters such as availability or current server load, the load balancer decides which server should handle the request and forwards it to the selected server.
    - To provide the load balancing algorithm with the required input data, the load balancer also retrieves information about the servers' health and load to verify that they can respond to traffic.



로드밸런서에서 일들을 나눠주고 안에선 여러 일들을 나눠 가지도록 함. 로드 밸런서 알고리즘)
1. 라운드 로빈(순차적으로 일을 줌)을 많이 씀. : 리퀘스트가 유사할 때 많이 씀.
2. 리퀘스트가 유사하지 않으면? = 이종의 리퀘스트. 다른 종류의 시스템을 접속하는 리퀘스트라면) 마스터는 서버들에 대한 프루빙을 함. 네트워크 대역폭, 메모리의 가용량, ...(녹음) 등등을 체크하면서 리퀘스트에 따라 적절한 서버에게 분배시킴. - > 스테이트를 가져와서 힙에 올려도 되고 테이블로 관리해도 됨.

rk용성

Availability vs. Scalability

- **Availability** is defined by *uptime* -- the time between failures. (Downtime is the time to detect the failure, repair it, perform required recovery, and restart tasks.)
  - During uptime the system must respond to each request within a predetermined, well-defined time. If this time is exceeded, the client sees this as a server malfunction.
  - High availability, basically, is redundancy in the system: if one server fails, the others take over the failed server's load transparently. The failure of an individual server is invisible to the client.

시스템이 항상 가용한 상태. 가용한 상태란, 전체시스템에서 서비스를 개죄하는,,,? 시간. = 업타임(< - > 다운타임 , 페일이 일어나서 디텍트 하고 어쩌구 저쩌구 해서 리커버리 하기 까지) 퍼센트로 표시함.

시스템이 계속해서 붙여나가는 역할. 원하는 역할. 원하는 퀄리티 오브 서비스
여러가지 성능 평가, 롣드들을 얼마나 확장성 있게 하는가,,
주어진 리퀘스트가 서버에 분산되어 가는데 로드가 적절한 서버에 들어가게 되고 거기에 우리가 어베일어빌리티를 측정함
다른 영역으로도 할 수 있다. 스케일 아웃을 할 숫 있다...

Availability vs. Scalability

- **Scalability** means that the system can serve a single client, as well as thousands of simultaneous clients, by meeting quality-of-service requirements such as response time.
  - Under an increased load, a high scalable system can increase the throughput almost linearly in proportion to the power of added hardware resources.
- High scalability is reached by distributing the incoming request over the servers. If the load increases, additional servers can be added, as long as the load balancer does not become the bottleneck. To reach high availability, the load balancer must monitor the servers to avoid forwarding requests to overloaded or dead servers. Furthermore, the load balancer itself must be redundant too.

Dispatcher(Load Balancer) Architecture Style - Solution

디스패처를 어느 레이어에서 하느냐도 달라짐.
- 트랜스포트는 레이어 4에서 함. 특징은 게런티. 모든 패킷이 레벨 4에서 하면 제대로 도착 가능함! 트랜스포트 레이어에서 로드 밸런스를 하는 경우, 4계층에서 패킷을 받아 어디로 보낼지 결정.
- 7계층인 경우 의미론적으로 로드 밸런싱 하는 경우 ㅁ낳이 씀.

- Implementations
  - In general, server load balancing solutions are of two main types:
    - *Transport-level* load balancing -- such as the DNS-based approach or TCP/IP-level load balancing -- acts independently of the application payload.
    - *Application-level* load balancing uses the application payload to make load balancing decisions.

도메인 네임 서버 시스템. 도메인 네임에 따라 아이피 어드레스를 달리줌

프라이빗 아이피를 줌

  - Load balancing solutions can be further classified into software-based load balancers and hardware-based load balancers.
    - Hardware-based load balancers are specialized hardware boxes that include application-specific integrated circuits (ASICs) customized for a particular use. Hardware-based load balancers are often used for transport-level load balancing. In general, hardware-based load balancers are faster than software-based solutions. Their drawback is their cost.

하드웨어적으로 품

그냥 하드웨어를 낌. 4계층 스위치를 사다 끼는 것. 이러면 하드웨어적으로 asics 에 따라 안쪽에 있는 군들을 등록 해놓으면 라운드로빈으로 접근해 하나씩 할 수 있도록 알고리즘을 돌림. 빠르게 로드 밸런싱을 함.

소프트웨어 적으로 품

    - In contrast to hardware load balancers, software-based load balancers run on standard operating systems and standard hardware components such as PCs. Software-based solutions runs either within a dedicated load balancer hardware node or directly in the application.

리눅스 lvs 에서 많이 씀. 여기선 로드 밸런ㅅ를 짤 수 있다.
로드밸런서 알고리즘을 만들 수 있음~

Dispatcher(Load Balancer) Architecture Style -Examples

하나의 dns에 여러 아이피 어드레스를 줌. 필요에 따라 dns에 접속하면 하나씩 줌.

- **DNS-based load balancing**
  - The DNS-based approach is based on the fact that DNS allows multiple IP addresses (real servers) to be assigned to one host name.
  - DNS is an efficient solution for global server load balancing, where load must be distributed between data centers at different locations.
  - Often the DNS-based global server load balancing is combined with other server load balancing solutions to distribute the load within a dedicated data center.

레벨 4에서 주로 함. 마스터에게 실제로 주는 아이피 어드레스를 주고, 뒤에는 프라이빗 어드레스가 있음. 아이피의 낭비도 없음. nat 방식을 해서 네트워크 트렌스레이트만 함. 안에서 나갈땐 마스터에서 대표 ip로 바꿔 줌.
대표를 통해 들어가고 나오고 함.

- **TCP/IP server load balancing**
  - TCP/IP server load balancers operate on low-level layer switching. A popular software-based low-level server load balancer is the Linux Virtual Server (LVS).

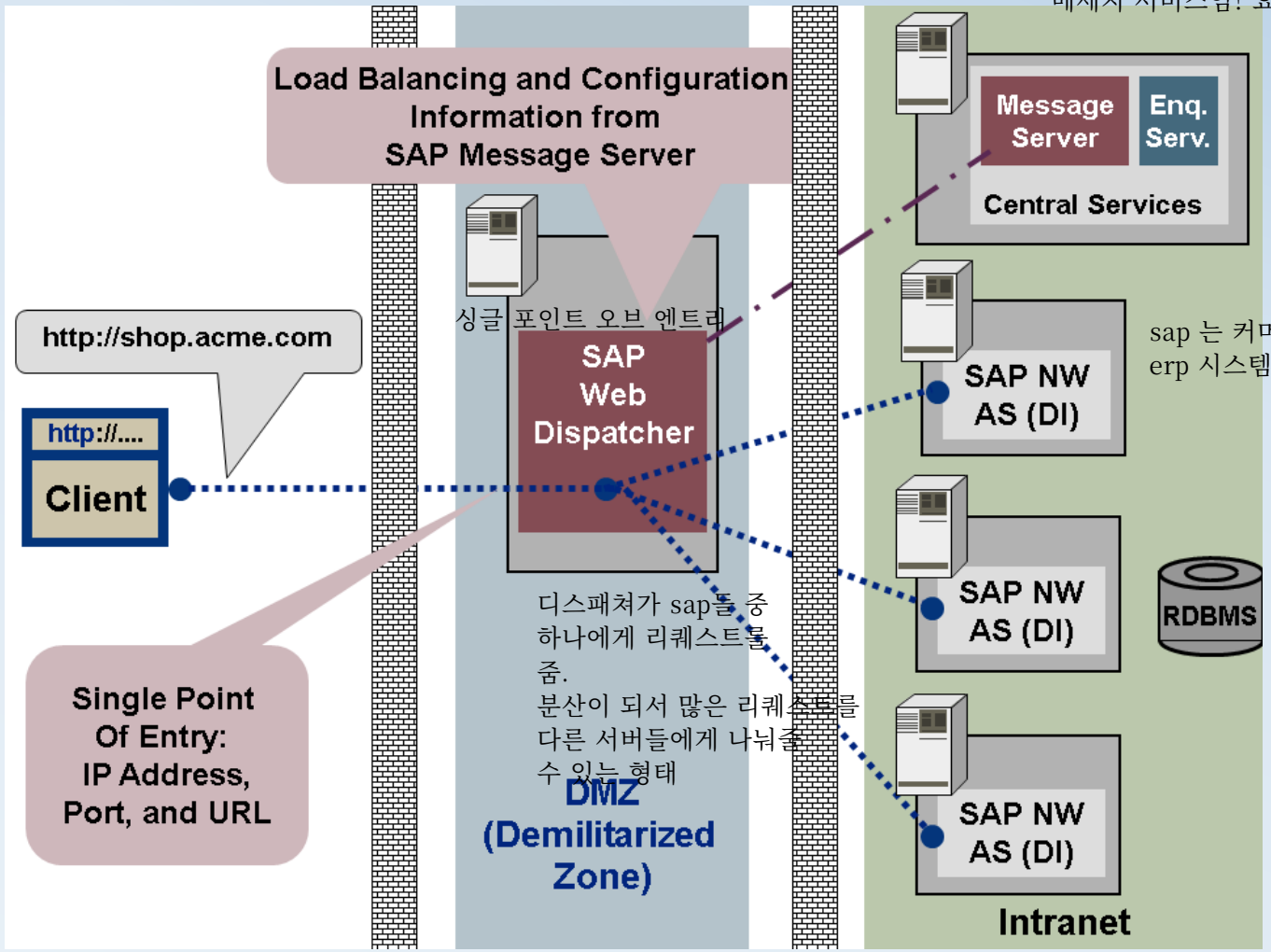
이것도 tcp/ip 레벨에서 함~

  - The incoming requests on a TCP connection are forwarded to the real servers by the load balancer, which runs a Linux kernel patched to include IP Virtual Server (IPVS) code.

버츄얼 서버역할로 마스터가 진행된다.

Dispatcher(Load Balancer) Architecture Style -Examples

메세지 서비스임! 요기에 등록된 곳으로 메세지를 보낼 수 있게 함



sap 는 커머셜 버전의 어플리케이션 서버.
erp 시스템도 sap 머신임.

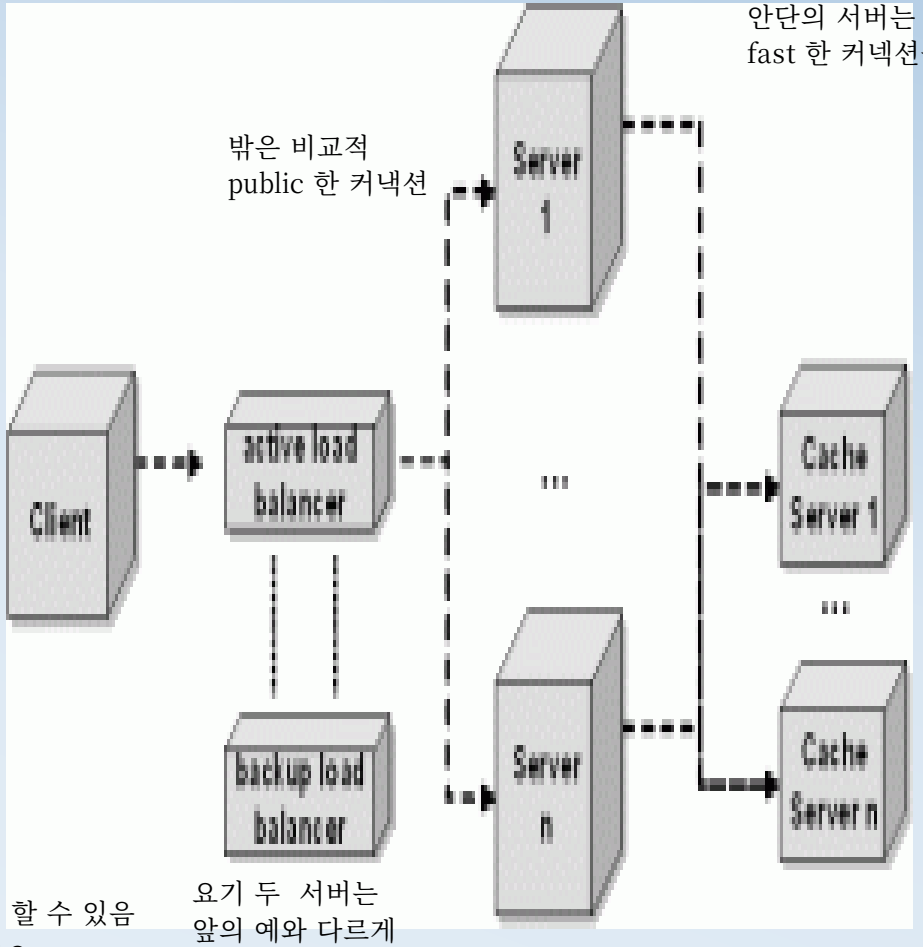
Dispatcher(Load Balancer) Architecture Style - Solution

stateless vs stateful
원하는 세션을 유지하고 싶으면 세션에 대한 오브젝트를 유지해야 하는데 이를 위한 캐시 서버를 둠.
세션의 번호를 가지고 있으면 그 제너럴 아이피를 가지고

- Implementations
  - Caching
    - Low-level server load balancer solutions such as LVS reach their limit if application-level caching or application-session support is required. **Caching** is an important scalability principle for avoiding expensive operations that fetch the same data repeatedly. A cache is a temporary store that holds redundant data resulting from a previous data-fetch operation.
    - If a cache is used on the server side, straying requests will become a problem. One approach to handle this is to place the cache in a global space. memcached is a popular distributed cache solution that provides a large cache across multiple machines.

안단의 서버는
fast 한 커넥션을 하게 함

밖은 비교적
public 한 커넥션



요기 두 서버는
앞의 예와 다르게
백업한 로드밸런서가
있음.

1. active - active하게 할 수 있음
허브로 리슨을 하고 있음.
2. active - passive
액티브는 자기가 처리하고 패시브한테
노티파이만 해줌. 패시브하게 따라가다 페일러가 일어나면 액티브가 됨 3. 스페어도있음 . 로그보다가 갈아낌.