AARHUS UNIVERSITET

Microservices and DevOps

DevOps and Container Technology

Horizontal Scaling and Session Management

Henrik Bærbak Christensen

AARHUS UNIVERSITET

Scalability Quality Attribute

Why?

- Scaling ~ ‘make bigger’ in some sense

- The two main Quality Attributes in Architectural sense

- ***Performance***
  - Handle more work, and/or handle it faster (latency)
    - Two persons dig twice as fast as one person…
- ***Availability***
  - Ability to handle work in case of one service failing
    - Two persons are less likely to be sick at the same time
      - (give and take a pandemic ☹)

Scalability [Bass et al., 2012]

- Vertical Scalability (Scale up)
  - *Adding more resources to a physical unit*
    - *More RAM, more Disk, more CPU*

Higher Availability ?

- Horizontal Scalability (Scale out)
  - *Adding more resources to logical units*
    - More servers

Higher Availability ?

- In cloud computing *Elasticity*
  - *Add/remove VMs to resource pool*

AARHUS UNIVERSITET

According to Bass et al.

Scalability

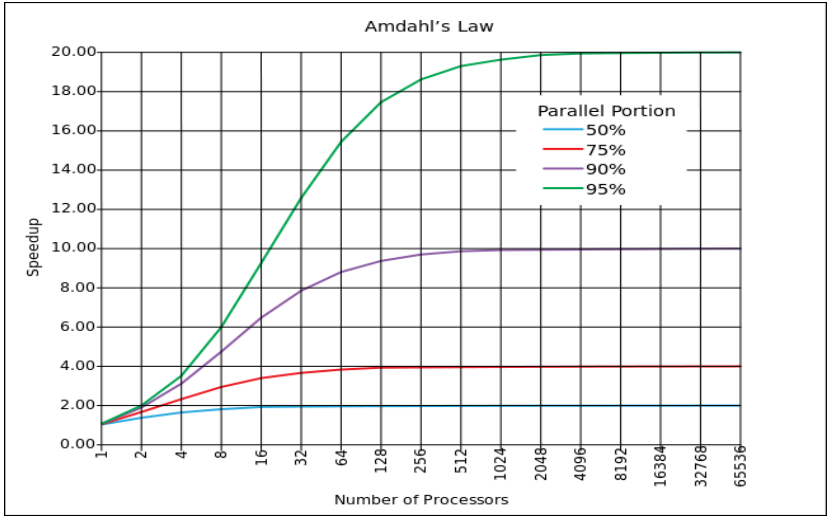
Two kinds of scalability are horizontal scalability and vertical scalability. Horizontal scalability (scaling out) refers to adding more resources to logical units, such as adding another server to a cluster of servers. Vertical scalability (scaling up) refers to adding more resources to a physical unit, such as adding more memory to a single computer. The problem that arises with either type of scaling is how to effectively utilize the additional resources. Being *effective* means that the additional resources result in a measurable improvement of some system quality, did not require undue effort to add, and did not disrupt operations. In cloud environments, horizontal scalability is called *elasticity*. Elasticity is a property that enables a customer to add or remove virtual machines from the resource pool (see Chapter

AARHUS UNIVERSITET

Performance?

- Performance
  - Twice the work? No, now more overhead coordinating!
    - Much more on that in my ‘Software in Architecture’ fagpakke…

- Amdahl’s law
  - *Speedup is limited by the portion that can be run in parallel*



AARHUS UNIVERSITET

Availability Calculations

- If a system of *n* replicated servers in which each server has a probability, *p*, of failing, then the system has total probability
- p^n of failing
- Ex
  - p = 5% (0.05)
    - (72 minutes every 24h)
  - n = 3
  - Overall failure rate:
  - $0.05^3 = 0.000125 = 0{,}125$ per mille
    - (10 seconds every 24h)

That is:
1 - p^n availability

AARHUS UNIVERSITET

SkyCave Scalability

- Ex: Some 100.000 users on SkyCave
  - One user ~ one request every two seconds
  - That is: 50.000 requests per second

- Issues:
  - Try it with ‘socket.cpf’
    - It is *single-threaded server*
      - If ‘quote’ talks to the quote service but it is *slow responding* (say 10 secs)
      - Then *all 100.000 users will experience a 10 second delay!*

AARHUS UNIVERSITET

SkyCave Scalability

- Issues:
  - The ‘http.cpf’ is a *thread-pooled server (Jetty)*
    - So less chance of all waiting for a quote, but…
    - The server may hit hardware limits in
      - IO transfer to/from DB
      - CPU at 100%
- One solution: Horizontal Scaling:
  - *Make several instances of ‘daemon’ available for processing*
    - *Two can (nearly) do twice as much as one*
      - *Coordination effort – shared resources*

Load Balancers

One Example of Horizontal Scaling

Load Balancer

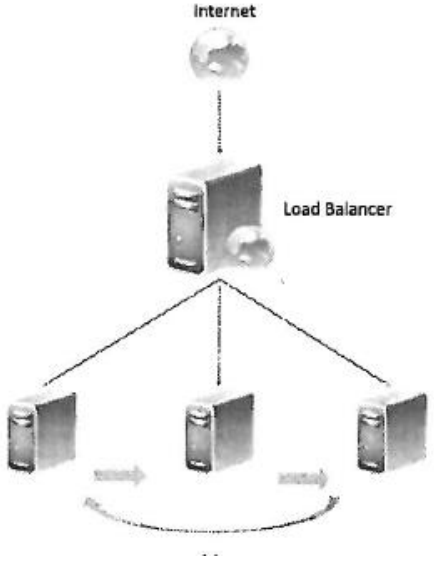
- Load Balancer
  - *Makes the pool of servers under the load balancer appear as a single server with high computing capacity.* *[Bahga et al., 2014]*

  - Basically a *reverse proxy server* that distribute requests to a pool of servers based upon some algorithm

Reverse proxy

From Wikipedia, the free encyclopedia

In computer networks, a **reverse proxy** is a type of proxy server that retrieves resources on behalf of a client from one or more servers.



Examples

- HA Proxy
  - TCP/HTTP proxy
    - Used by github, airbnb, instagram, stack overflow, …
- Nginx
- Round-robin DNS
  - Multiple ip addresses associated with single domain
    - Can also give some very weird problems
- Hardware load balancers
- Docker Swarm's ingress network
- *Messaging* systems
  - Can load balance… and a lot more…

AARHUS UNIVERSITET

Session Management

Statefull versus Stateless

Servers and State

- The main requirement on any server that is load balanced is:

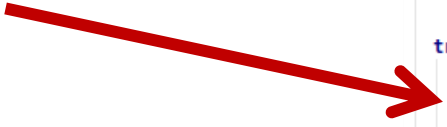
It must be stateless

- Stateful:
  - Has cached/stored state about given session with a client *in the server* itself
- Stateless
  - No stored state about given session *in the server itself*

AARHUS UNIVERSITET

Sessions?

- Some domains do not require sessions
  - Simple web browsing
  - Simple data storage
- Others domains, sessions are vital
  - Shopping basket while web shopping
  - Game interaction
  - SkyCave
    - Who is exploring?



```
@Override
public ReplyObject handleRequest(String objectId, String operationName, String payload) {
  ReplyObject reply = null;

  // objectId is a mangling of both the player's playerId and sessionId,
  // so we have do demangle it to get the two parts
  String[] parts = Marshalling.demanglePlayerAndSessionId(objectId);
  String playerId = parts[0];
  String sessionId = parts[1];

  // Payload is delivered as JsonArray from the server request handler.
  JsonParser parser = new JsonParser();
  JsonArray array = parser.parse(payload).getAsJsonArray();

  try {
    // Fetch the player object from the name service
    Player player = objectManager.getPlayerNameService()
            .get(playerId);
```

AARHUS UNIVERSITET

Architectural Design

- Ok, so SkyCave has already encapsulated session management?

```java
public interface PlayerNameService extends ExternalService {

  /**
   * Get the player object corresponding to the given player id.
   *
   * @param playerID
   *          the id of the player
   * @return null if no player id is stored in the name service, ot
   *         object
   */
  Player get(String playerID);

  /**
   * Add a player instance under the given player id
   *
   * @param playerID
   *          id to store player instance under
   * @param player
   *          the player instance to add to service
   */
  void add(String playerID, Player player);
```

```java
// Create player domain object. To ensure dependency injection,
// we have to let the factory create it
CaveServerFactory factory = objectManager.getFactory();

Player player = factory.createPlayerServant(theResult, subscription.getPlayerID(), objectManager);
// new PlayerServant(theResult, subscription.getPlayerID(), objectManager);

// Enter the player object reference into the name service
objectManager.getPlayerNameService().add(player.getID(), player);
```

CaveServant

Encapsulate what varies
Program to an interface
Favor object composition

AARHUS UNIVERSITET

Session Handling

- Bahga et al., 2014, lists the possible practices:
  - Sticky sessions
    - All requests from session ‘a’ are routed to the same server
  - Session database
    - State of session ‘a’ is stored in a database, which all servers retrieve session state from
  - Browser cookies (client session)
    - State of session ‘a’ is stored in client, and sent along to server in each request
  - URL re-writing

AARHUS UNIVERSITET

Sticky Sessions

- *All requests from session ‘a’ are routed to same server*

- Of course, requires the load balancer to have some logic to make that happen
  - Example: MQ systems can route messages on ‘topics’
    - A topic could be ‘skycave.server.1’
      - Begin forwarded to ‘server1’ because MQ can link ‘*.*.1’ topics to that particular server

- Benefits/Liabilities?

AARHUS UNIVERSITET

Session Database

- *State of session ‘a’ is stored in a database, which all servers retrieve session state from*

- Simple load balancing – just ‘round robin’ would do…

- Benefits/Liabilities?

Client Session

- *State of session ‘a’ is stored in client, and sent along to server in each request*
  - Browser cookies
- Again, load balance is simple
- Benefits/Liabilities?

AARHUS UNIVERSITET

Discussion

- What kind of “session management” is used in SkyCave at the moment?
  - Sticky session, session database, client sessions
- Could we code a ‘client session’ approach?
  - What would it require?
  - And… Why can we not use the approach ☺ ?

Discussion

- Why do we scale?
  - A) to get improved availability
    - Then sticky sessions are problematic
  - B) to get improved performance
    - Then it puts a demand on the session database – which is?

AARHUS UNIVERSITET

Scaling Databases

Statefull services (databases)

Important!

- So, … we will return to that in the second course…

- Redundancy and Replication are the key techniques.
  - NoSQL db's all support it out of the box (to my knowledge)
  - And the SQL ones have followed suit (to my knowledge ☺)

Summary

- More users means more resource demand
- Answer: Add resources (or become more efficient)
  - Horizontal or Vertical
- Load Balancing:
  - Make a server cluster appear like one server
- Session Management
  - Handle that different servers are ‘hit’ by given client