Performance Testing

- Audience warmup - warmup is a very useful phase of software performance testing, to get the system into a stable state, so let's warm up our system with a few questions
    - Who has their own car or drives regularly?
    - Who knows what is the top speed of this? What is the engine power?
    - Who has ever written an application?
    - Who knows what is the performance of this application?
    - And what did you measure?
- Types of tests
    - Load testing - the simplest performance test, figuring out whether the system behaves correctly under the given load, which can be represented by number of concurrent users, size of the requests, number of the requests in the given time frame, or any combination
    - Stress testing - finding the upper limits of the application throughput, can be used to verify application robustness while reaching or overcoming the upper throughput boundary => DoS
    - Soak or Endurance testing - verification that the application can handle large load for a long period of time, we search for performance degradation and memory leak
    - Benchmark
        - What is benchmark?
        - In general, it is a set of performance tests
    - Scalability testing - verification that the system can scale up when deployed in larger scale (number of nodes in a cluster, more performant HW, more users…), scale down is important as well
- What can we measure? What quantity/value? What is a performance test? What should we look for?
    - transactions/requests/messages per second
    - Response time
    - Service time vs Response time
    - accessibility - resp. time - influence on users
    - throughput/utilization - how efficient is the application why utilizing resources
    - avg, max, min, sliding window
    - What does a typical performance requirement look like?
        - in 95% response time is under Xms
        - number of concurrent users
        - reliability - accessibility in time, failure rate
- How can we measure?
    - Whitebox
        - Test directly in the application code
        - code instrumentation
        - change of application code can significantly influence the application behaviour
    - Blackbox
        - we do not know what is inside, what is the real behaviour of the system
        - application access - Web UI? Do I measure the app. or the UI?
        - UI is rarely the culprit, there can be a proxy server, router, load balancer
- Test parameters - what can we specify
    - Concurrent users/transactions
    - Message size
    - request content - WS - special characters
    - Request peaks
    - transactional testing - message backlogs - generate and then commit
- Environment influence

    - the test environment must be as much similar to prod. as possible
    - https://tweaked.io/
    - Application logging
    - Library versions
- Influence of the measurement tool
    - jMeter vs PerfCake
    - Definition of an ideal tool (page 15 in thesis by Ron Šmeral)
    - http://is.muni.cz/th/256206/fi_m/
- Reporting
    - results processing
    - RAW picture
- Reproducable/repeatable results
    - Absolutely the same environment
    - Network traffic
    - Updates, indexing in background
    - Multiple nodes to generate load
    - Dedicated database server
    - Results processing and storage - automated - PerfRepo
- Results comparison
    - Before and after an app change
    - New version comparable to the old one
    - Baseline compared to competitor's app
    - BPMS - simulation of actions of a process, then real process, counting expected slow down
- Important attributes of a perf. measurement tool
    - Clustering
    - Power/Performance!
    - Influence on the measured app
    - Extendability
    - Supported protocols
    - Injection profiles
    - Usability - some tools require installation of an http server, used via browser
    - Results reporting
    - Message templates and sequences
    - license/price
    - Warmup detection
        - How do I know the system is warmed up and provides reliable results?
    - Memory leak detection
- Tools overview
    - jMeter
    - Gatling
    - Faban
    - Grinder
    - PerfCake
- Hints for performance measurement
    - Running in a virtual environment
        - current topic
        - users are running the apps in VM
        - necessity to measure the performance even here
        - timer/stopwatch precision
        - generate load out of VM
    - Response time vs. throughput

        - Single thread, resp. time = 5ms, throughput = 1 / resp. time = 200 msgs/s
        - 5 threads, avg. resp. time = 5ms, throughput = 1 / resp. time = 200 msgs/s
        - What is wrong?
        - In fact, we have 5x 200 msgs/s
        - Using integrals for non-constant thread count
    - Coordinated omission
        - 100 threads, avg. resp. time 1ms, test runs for 10s, we collect 1 000 000 measurements
        - Then, application freezes for 10s, all threads hit this barrier, we collect 100 measurements
        - Result: 1_000_000 x 1ms + 100 x 10_000ms = 2_000_000ms / 1_000_100 = 1.999ms avg. resp. time - isn't this a strange number?
        - How about the histogram? 90% = 1ms, 95% = 1ms, 99% = 1ms, 99.9999% = 1ms (1_000_100 / 100 = 10_001, 1 / 10_001 = 0.00009999, 100 - 0.00009999 = 99.9999), 100% = 10_000ms
        - How to solve this?
            - adding clients/threads/requests to get the same number of records
            - less requests in both phases
            - fix the numbers mathematically
    - Monitoring
        - watching the perf. of a deployed app in the real environment for the purpose of an audit and analysis
        - the load is generated by real users
        - we just passively collect information
- Application profiling
    - when we run the test, we discovered an issue but we do not know what is really wrong
    - first step is to run the test once more - to reproduce the problem and minimize the negative impact of the environment
    - how about adding println to the code?
    - usage of CPU and memory
        - memory - heap, permgen/metaspace
        - threads, classes
        - GC activity
    - monitor - threads and their state - running, sleep, wait, park, monitor
    - usage of the memory by individual objects - number of instances, used memory
    - where do the threads spend most of the time in the code - sorted in a tree according to the call hierarchy
    - jvisualvm (Oracle JDK) / visualvm (open source projekt)
        - basic Java tool
    - jprofiler
        - advanced - heap walker - it is possible to figure out where were the large objects allocated
        - it can analyze usage of JDBC, JMS, JTA, JCA, RMI, Servlets, sockets, files and other standard APIs that interact directly with the system and external resources
        - it can display the source of locks and monitors
        - snapshot comparison
    - details will be demonstrated at the exercise