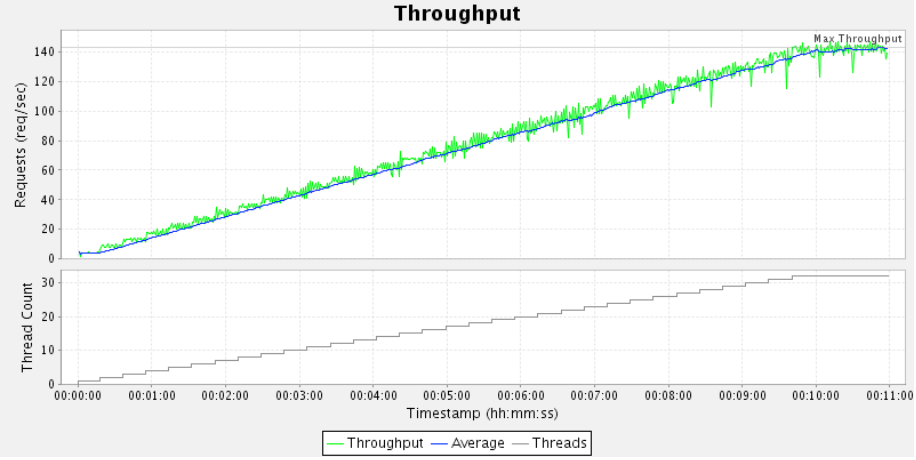
Bilag 2a

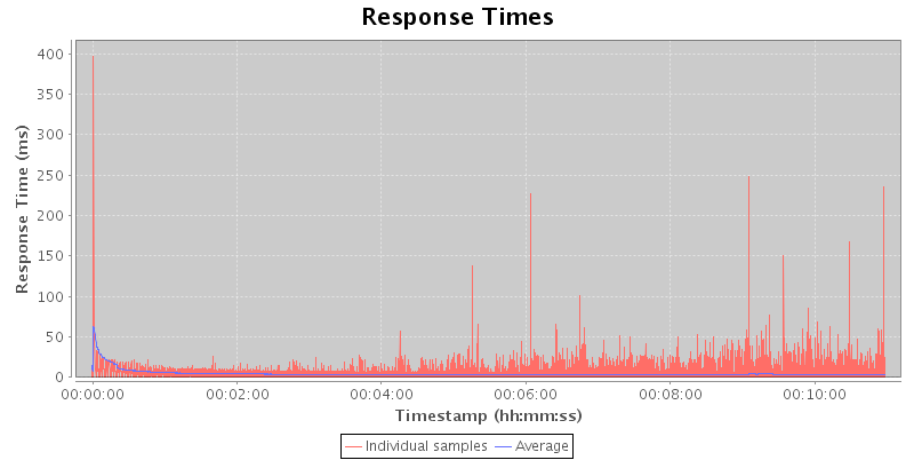
LAKESIDE A/S
Åbogade 15
8200 Århus N
+45 2160 7252

cvr-nr: 2545 0442
info@lakeside.dk
www.lakeside.dk

Kvalitetssikring af Software

Verificering af Performance





Version	Dato	Ansvarlig	Kommentarer
0.1	01-11-2011	KSR	Første udkast
0.2	28-11-2011	JRI	1. review af JRI
0.3	29-11-2011	ADS	Konsolideret med ADS notat
0.4	30-11-2011	JRI	Opdateret afsnit 3.1 og 3.2
0.9	01-12-2011	ADS	Endelig version
1.0	05-12-2011	ADS	0.9 opløftet til 1.0

LAKESIDE

Indholdsfortegnelse

1. Introduktion

1.1 Baggrund

Det er alment kendt at it-systemer, især nyudviklede it-systemer, skal kvalitetssikres inden det tages i anvendelse. De krav, der er stillet til systemet, skal verificeres inden systemet skal overgå fra et udviklings-/etableringsprojekt til drift.

Historisk set har kvalitetssikring af it-systemer haft fokus på test af systemets funktionalitet, dvs. systemer er testet igennem uden fejl eller mangler.

Denne ensidige fokus på funktionelle krav har ledt til, at mange systemer er blevet sat i drift, og efterfølgende trukket tilbage eller sat i stå, på grund af performance-problemer i forbindelse med igangsætning, udrulning eller efter at have været i brug et stykke tid, hvor datamængderne er vokset.

Disse dårlige erfaringer har igennem de seneste år øget fokus på en ny side indenfor kvalitetssikring af it-systemer: Performancetests.

Nærværende dokument har tre formål:

- at indføre læseren i performancetest-området, herunder definere forskellige aspekter af performance og performancetest
- at foreslå forskellige testtyper, der, hvis de bliver udført omhyggeligt, vil kvalitetssikre et it-system optimalt i forhold til performance
- at foreslå nogle tekniske værktøjer til udførelse af performancetests

1.2 Afgrænsninger

Denne rapport gennemgår principperne for kvalitetssikring af IT systemer via performancetests. Disse principper skal ses som et supplement til et allerede eksisterende system til kvalitetssikring af funktionalitet.

1.3 Metode og rapportens opbygning

Rapporten består i al væsentlighed af to kapitler:

Efter nærværende introduktion vil rapporten i kapitlet ”Verifikation af Software Performance” gennemgå de forskellige typer at tests, der bør foretages som led i kvalitetssikring af it-systemer. Målet med dette kapitel er at sikre, at læseren har en overordnet forståelse af performancetestkonceptet.

De forskellige typer tests vil herefter i kapitel 3.2 og frem blive gennemgået, mht. formål og forventet resultat.

1.4 Målgruppe og læsevejledning

Den primære målgruppe for rapporten er medarbejdere i it-arkitekturenheder, systemudviklere, driftsmedarbejdere, samt medarbejdere med ansvar for kvalitetssikring.

For at få fuldt udbytte af rapporten bør man tidligere have haft berøring med kvalitetssikring af IT systemer.

Igennem rapporten vil anbefalinger blive fremhævet i bokse (se eksempel nedenfor). Anbefalingerne uddrager essensen af det pågældende afsnit af rapporten.

<Anbefalingstekst>

Rapporten gør ikke ellers brug af særlige notationsformer eller modelleringsmetoder.

2. En introduktion til softwarekvalitet

Kvalitetssikring af softwareapplikationer kan overordnet opdeles i sikring af applikationens interne og eksterne kvalitet.

Intern kvalitet

Den interne kvalitet handler om den interne sundhed i systemet, kvalitet af kode og af dataskemaer, overholdelse af systemets arkitektur, og dermed hvor let applikationen er at vedligeholde og forbedre. Intern kvalitet kan sjældent observeres af en slutbruger, men er ofte ret åbenlys for de der udvikler, vedligeholder og drifter applikationen.

Manglende intern kvalitet fører dog ofte på sigt til manglende ekstern kvalitet. Man anvender ofte begrebet *teknisk gæld*, som illustrerer at manglende intern kvalitet påfører udviklere en rentebyrde (forøgede omkostninger til vedligehold) indtil gælden er afviklet. Er gælden stor og rentebyrden høj, kan det resultere i, at applikationen "lammes" idet omkostninger ved videreudvikling bliver uforholdsmæssig stor.

Ekstern kvalitet

Den eksterne kvalitet handler om, hvorvidt systemets funktionelle og ikke-funktionelle krav er opfyldt.

De funktionelle krav går på systemets opførsel fra en brugers synspunkt og specificeres typisk i f.eks. Use Cases eller User Stories. De ikke-funktionelle krav udtrykkes typisk i en bred vifte af forskelligartede krav, f.eks. brugervenlighed, tilgængelighed (for f.eks. handicappede) samt krav til svar- og oppetider. En væsentlig del af de ikke-funktionelle krav er kravene til et systems *performance*[1].

Definition af Performance:

Et udtryk for hvor godt en applikation udnytter den it-infrastruktur den afvikles i givet nogle belastningsmønstre. Performance måles i:

- Svartid målt i tid per serviceleverance
- Ydelse målt i antal serviceleverancer per tidsenhed
- Tilgængelighed målt i tid per medgået tid

En applikation med god performance er således:

- en applikation der leverer det fornødne ydelse i forhold til det forventede i den givne it infrastruktur
- en applikation der leverer svarene hurtigt nok i forhold til de forventninger, der er til applikationen i den pågældende it infrastruktur
- en applikation der er til rådighed, når der er brug for den. Heri ligger også at applikationen skal være robust over tid
- en applikation der ikke kræver unødigt meget it-infrastruktur til at kunne levere ovennævnte ydelse og svartid.

[1] se også Wikipedias definition af performance [WPERF]

2.1 *Verifikation af kvalitet*

Den eksterne kvalitet verificeres hovedsageligt ved tests og (i mindre omfang) ved inspektion.

Den interne kvalitet verificeres typisk ved en blanding af inspektion af f.eks. kildekode og dokumentation kombineret automatiserede værktøjer til at kontrollere kodekvalitet og overholdelse af arkitektur.

Verifikation og test af såvel intern som ekstern kvalitet bør foretages i forbindelse med alle væsentlige ændringer i systemet (igangsætning af nye versioner). Det er derfor fordelagtigt med en høj grad af automatisering af denne. Erfaringen viser, at manuel test ved mange gentagelser bliver prohibitivt dyrt, og derfor i praksis ikke vil blive foretaget så tit som ønskeligt. Foretages verifikationen kun lejlighedsvist, vil problemer kunne nå at eskalere inden de opdages, og vil derfor ofte være dyrere at rette op på. En høj grad af automatisering er derfor ikke blot ønskelig, men i praksis nødvendig for effektivt at sikre kvaliteten.

3. Verifikation af Software Performance

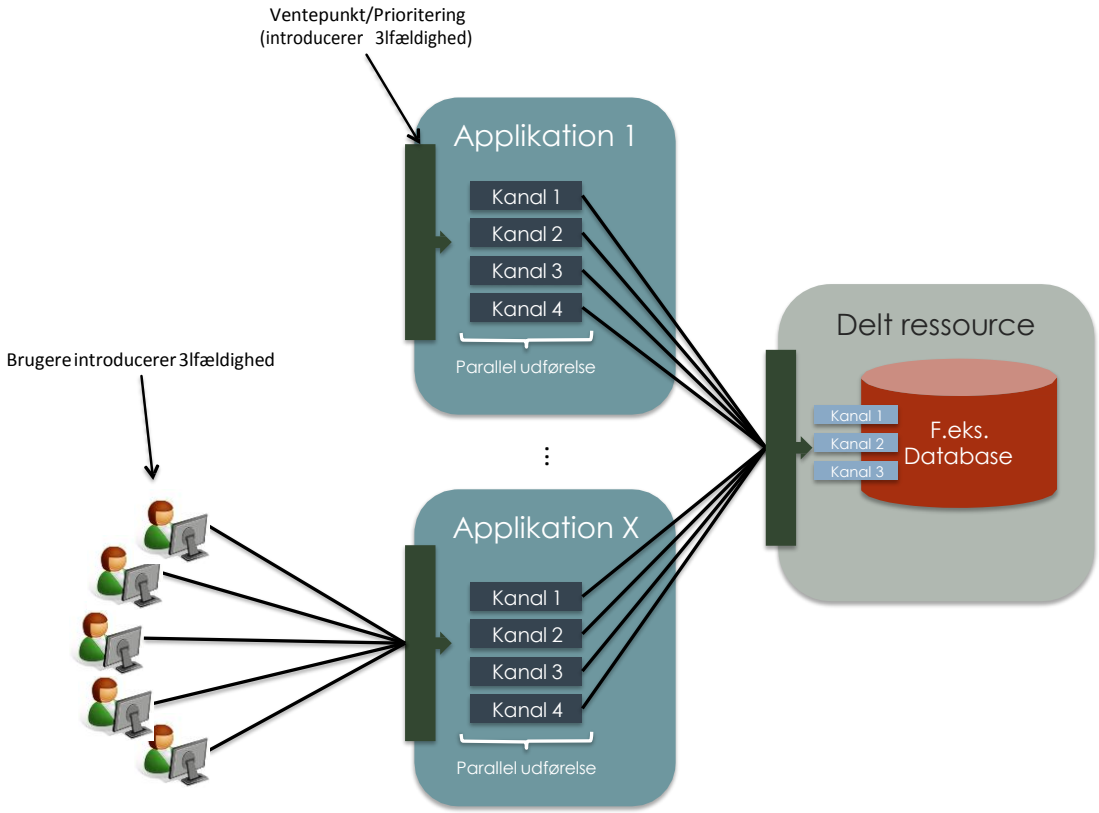
Nærværende notat omhandler retningslinjer i forhold til verifikation af software performance. Som nævnt ovenfor handler performance om svartid, ydelse og tilgængelighed. I det følgende behandles hver del for sig, og der foreslås forskellige typer af tests, der kan verificere disse aspekter af performance.

3.1 Verifikation af Svartidsperformance

Svartidsperformance måles typisk som responstid per forespørgsel, dvs. den tid det tager det samlede system at producere et svar på en modtaget forespørgsel.

I de fleste kørende systemer vil der være et tilfældighedsaspekt involveret i afsendelse, transport og modtagelsen af forespørgsler. Effekten af dette ses typisk ved at forespørgsler kommer i større eller mindre "samtidige klumper". Nogle gange vil disse "klumper" være større end antallet af parallelle kanaler hvormed applikationen kan behandle forespørgsler, hvilket medfører kødannelse ved indgangen til applikationen.

For at komplicere sagen yderligere, vil applikationer i mange tilfælde have behov for at anvende delte ressourcer, dvs. ressourcer som også andre applikationer bruger. Igen kan der være rift om de tilgængelige ressourcer, hvilket introducerer tilfældighed i form af forskellig svartid fra den delte ressource (se Figur 1).



Figur 1 - I et komplekst system vil der være en række punkter der introducerer tilfældighed

Alle disse tilfældighedspunkter vil fra en anvender blive set som udsving fra den ideelle svartid. Udsvingene vil dog følge et vist mønster – man siger at de følger en sandsynlighedsfordeling. Det er derfor relevant ikke kun at kigge på gennemsnitlige svartider, men også på hvor meget og hvordan svartiderne varierer. Det kan derfor være relevant at se på flere mål af svartider:

Hyppigt anvendte mål af svartider:

- Gennemsnitlig svartid
- Median. Halvdelen af svartiderne er mindre end denne.
- 95 % fraktil. 95 % af svartiderne er mindre end denne.
- 99 % fraktil. 99 % af svartiderne er mindre end denne.

Performancekravet vil typisk være formuleret som krav til svartider ved forventet normal belastning evt. suppleret med nogle krav til svartider ved forventet forhøjet belastning i nogle bestemte tidsrum.

Da en applikation oftest indeholder en række forskellige funktioner (operationer) der i varierende omfang belaster systemets ressourcer, skal svartiden altid testes under en passende fordeling af operationer og med en belastning, der bedst muligt passer til den man kender fra (eller forventer i) det kørende system.

Kvalitetssikring af svartider handler derfor om at sikre, at applikationen i en sammenlignelig it-infrastruktur og med en simuleret produktionslignende belastningsprofil giver nogle tilfredsstillende svartidsmålinger.

Udførelse af svartidstest:

- Definer kravene til svartider.
- Definer ved hvilken sammensætning af kald disse krav skal være opfyldt.
- Simuler en passende belastning med den definererede fordeling.
- Mål på svartiderne under denne belastning hhv. belastningsfordeling.
- Vurder hvor følsomt resultatet er overfor yderligere belastning.
- Lav målinger som faciliterer performancetuning i tilfælde af utilfredsstillende performance.

3.2 Verifikation af Ydelsesperformance

Ydelsesperformance består af en række målinger, der tilsammen danner et billede af, hvor meget et system kan yde, og hvordan systemet agerer i takt med, at belastningen øges.

For at kortlægge et systems ydelsesperformance, udføres typisk en testportefølje bestående af følgende test:

- **Ramp-up test**. Tester parallellitet i applikationen og måler den optimale og den maksimale ydelse.
- **Skalerbarhedstest.** En skalerbarhedstest foretages på systemet for at vise hvorledes systemet reagerer ved en forøgelse af tilgængelige ressourcer**.**

- **Loadtest**. Belastningen på systemet øges ved at variere forskellige parametre, f.eks. ændring af operations-sammensætning, forøgelse af datamængder i forespørgsler, forøgelse af datamængder i svar.

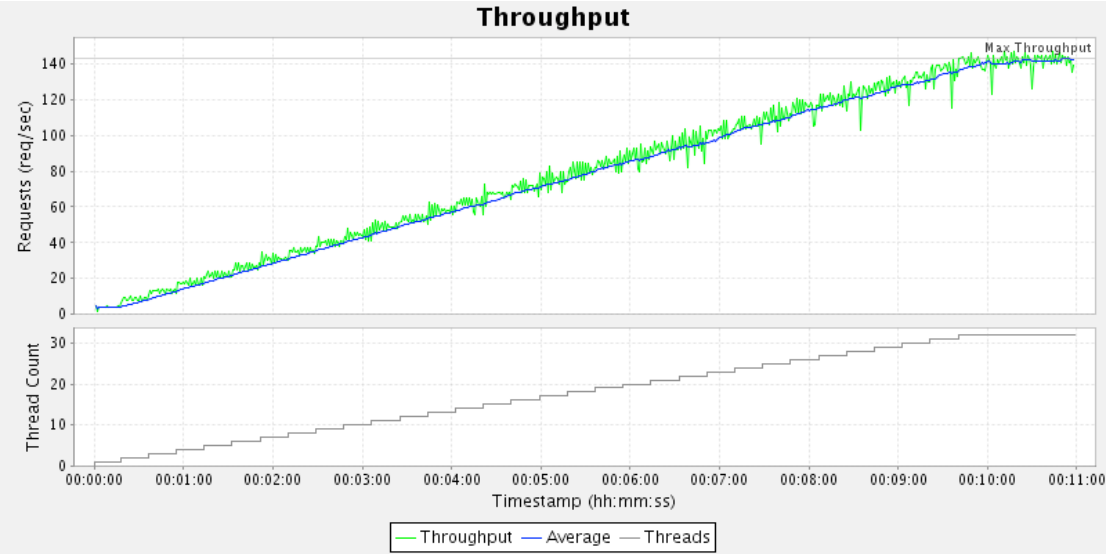
3.2.1 Ramp-up test

Ramp-up test. En ramp-up-test foretages på systemet for at identificere systemets optimale og maksimale ydelse.

En ramp-up-test udføres ved trinvist at øge belastningen på systemet og samtidig overvåge svartider og svartidsudsving, indtil man observerer en stagnation eller fald i ydelse. Ud fra antallet af behandlede beskeder pr. tidsenhed, kan man efterfølgende plotte en graf, og identificere hhv. optimum[2] og maksimal[3] ydelse.

Udførelse af Ramp-up test:

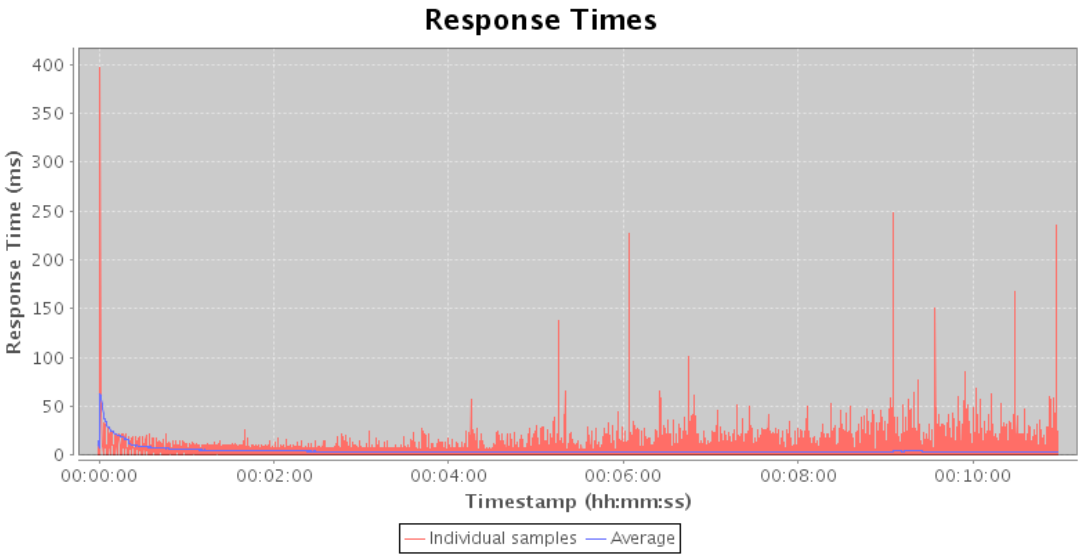
- Definer kravene til (*througput*).
- Definer ved hvilken sammensætning af kald disse krav skal være opfyldt.
- Foretag en test med stigende belastning med den definerede sammensætning
- Mål på ydelse ved hver belastning
- Mål på svartider, svartidsvarians, ressourceforbrug ved hver belastning
- Ud fra ydelsessignaturen kan ”Max. throughput” og ”Optimal throughput” beregnes.



Figur 2 - Ydelsesprofil der følger af en "Ramp-up" test (Chronos graf)

[2] Optimum er punktet på grafen, lige før linjens hældning falder. Efter dette punkt vil svartiden pr. besked stige, og overordnet set vil systemet herefter køre med nedsat performance.

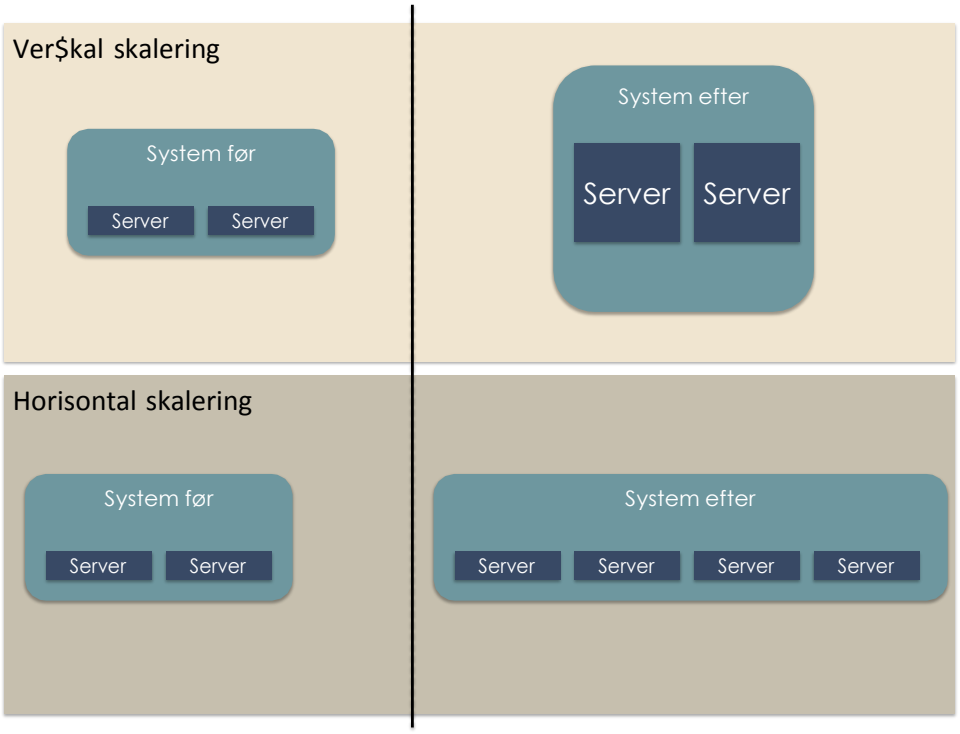
[3] Maksimal ydelse er punktet på grafen, lige før antallet af behandlede beskeder pr. tidsenhed begynder at falde.



Figur 3 - Udvikling af svartider og varianser i forbindelse med "ramp-up" (Chronos graf)

3.2.2 Skalerbarhedstest

En skalerbarhedstest foretages på systemet for at vise hvorledes systemet reagerer ved en forøgelse af tilgængelige ressourcer igennem en enten vertikal eller horisontal ressourceudvidelse. Vertikal skalering handler om at tilføre flere ressourcer til den eksisterende infrastruktur (mere RAM, mere båndbredde, kraftigere maskiner), mens horisontal skalering handler om at tilføre flere ressourcer i ”helheder” (f.eks. flere uafhængige servere).



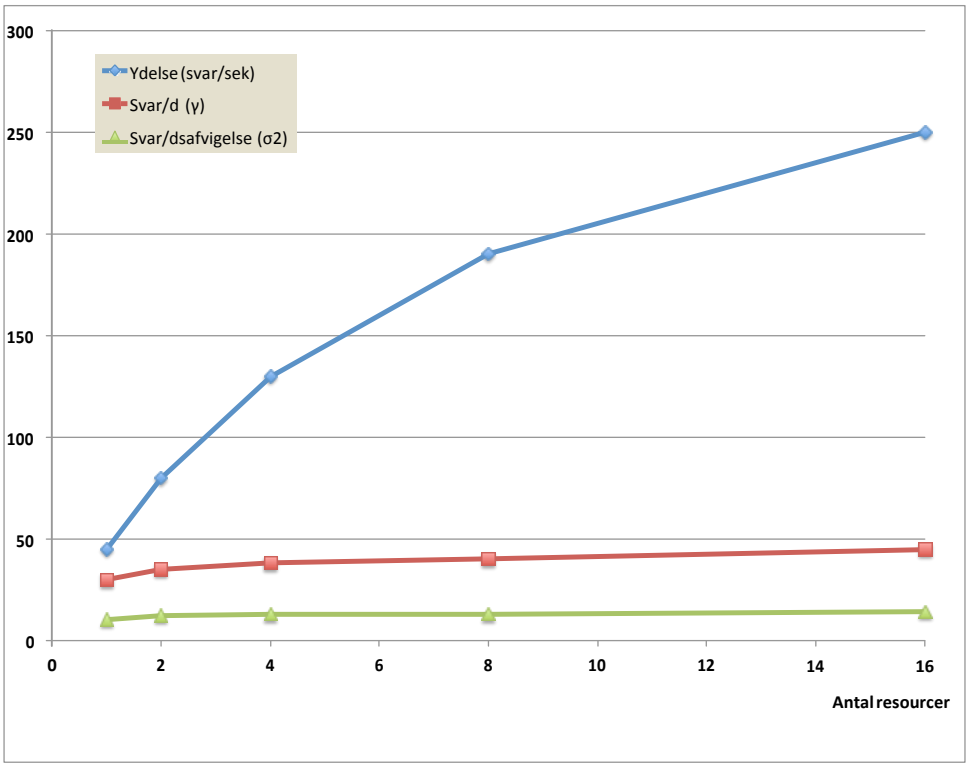
Figur 4 - Illustration af forskellen mellem vertikal og horisontal skalering.

Det perfekte system skalerer enheds-lineært, dvs. dobbelt så høj ydelse ved dobbelt så mange ressourcer. Oftest begrænses skaleringsmulighederne dog af eksterne fælles ressourcer, f.eks. databaser, og krav til disse om at kunne håndtere komplekse datatilstande i systemet. Skalerbarhed er et kompleks område, og et system er ikke nødvendigvis designet forkert, blot fordi det ikke skalerer lineært. Lineær eller nær-lineær skalering er dog stadig målet.

En skalerbarhedstest udføres ved at belaste systemet med passende baggrundsbelastning, og herefter måle svartiden og antal behandlede beskeder pr. tidsenhed. Herefter udvides ressourcerne og testen afvikles igen og resultaterne sammenlignes.

Udførelse af Skalerbarhedstest:

- Foretag en ”Ramp-up” test med minimale ressourcer
- Find den optimale ydelse og registrer svartidsinformationer ved denne ydelse (gennemsnit, fordeling, varians, etc.)
- Forøg de tilgængelige ressourcer
- Foretag på ny en ramp-up test og registrer målingerne igen målinger ved optimal ydelse
- Foretag denne skalering et tilstrækkeligt antal gange til at man kan danne sig et billede af tendensen, når målingerne sammenlignes på en graf (linearitet, logaritmisk tendens eller lignende)
- Ud fra tendenser kan de nødvendige ressourcer estimeres i forhold til de ydelseskrav, der er til systemet



Figur 5 - Eksempel på resultater fra skaleringstest.

3.2.3 Load tests

En loadtest foretages på systemet for at kortlægge, hvordan systemet reagerer, hvis brugsmønsteret ændrer sig. Loadtestens formål er altså at identificere situationer, hvor en et ændret brugsmønster kan skade systemet svartidsperformance.

Testen udføres normalt kun på de mest sandsynlige ændringer i brugsmønstret. Hvis brugsmønstret ikke forventes at ændre sig, kan denne test evt. udelades.

Selve testen udføres på samme måde, som ovennævnte ramp-up test, men med andre parametre. Efterfølgende sammenlignes optimum og maksimum med ramp-up-testen for den aktuelle baggrundsfordeling.

Udførelse af Load test:

- Afdæk de mest sandsynlige kommende udviklinger i kalds-sammensætning.
- Foretag en "Ramp-up" test, hvor hver af disse sammensætninger simuleres.
- Sammenlign ydelses- og ressourcemålinger med målingerne foretaget med den forventede produktionssammensætning af

3.3 Verifikation af stabilitetsperformance

Stabilitetsperformance består af en række tests, der tilsammen danner et billede af, om et system er stabilt, og hvordan systemet agerer ved ekstrem høj belastning.

For at kortlægge et systems stabilitetsperformance, udføres typisk en testporteføljе bestående af følgende test:

- **Stresstest**. Tester systemets evne til at håndtere en ekstrem overbelastning i kortere eller længere tid.
- **Spiketest.** Tester systemets evne til at håndtere pludselige skift i belastning**.**
- **Udholdenhedstest**. Systemets holdes kørende i en længere periode, mens en række driftsparametre overvåges, hvorved det verificeres, at systemet ikke degraderer over tid.

3.3.1 Stresstest

Stresstest. En stresstest foretages for at sikre, at systemet ikke bliver ustabilt under ekstremt højt load. Formålet med testen er dels at vise driftsmæssig stabilitet, men også sikkerhed, idet systemet skal være resistent overfor f.eks. et DOS[4] angreb.

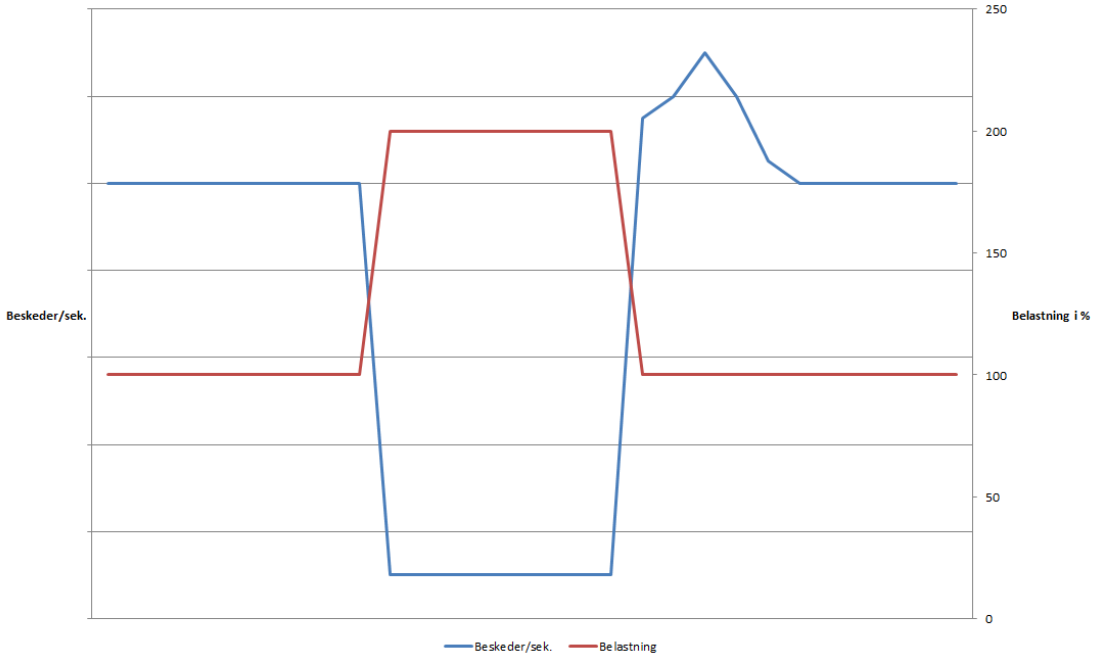
En stresstest udføres ved at overbelaste systemet i en sådan grad, at systemet ikke længere kan honorere svartiderne og samtidig begynder at afvise forespørgsler. Efter en periode med ekstrem belastning, sættes belastningen ned til normal igen, og system skal herefter returnere til normale svartider.

[4] DOS = "Denial Of Service" angreb er et forsøg på at gøre et system utilgængeligt for brugerne ved at overbelaste systemet [DOSwiki]

Belastningen under en stresstest skal sættes så højt, at man tydeligt kan observere ressource udsultning. Typisk udsættes systemet initielt for en belastning svarende til 2 gange peak[5].

Udførelse af stresstest:

- Definer kravene til ”ekstrem” belastning.
- Definer den forventede varigheden af perioden efter belastningen trappes ned og til systemet kører normalt.
- Foretag en test med ekstrem belastning.
- Overvåg at systemer ikke fejler, men afviser beskeder pænt.
- Mål på svartider, svartidsvarians, ressourceforbrug under den ekstreme belastning.



Figur 6 - Eksempel på resultat fra stresstest.

Som det ses af eksemplet udviser det tænkte system den ønskede kvalitet. Systemet bliver presset af den øgede belastning, og begynder at afvise beskeder. Systemets performance degraderer fordi, belastninger bliver for stor, så totalt set, behandler systemet færre beskeder pr. sekund. Efter der skrues ned for belastninger, stabiliserer systemet sig efter en kort periode, og kører derefter igen normalt.

3.3.2 Spiketest

Spiketest. En spiketest foretages for at sikre, at systemet kan håndtere pludselige skift i belastningsniveau.

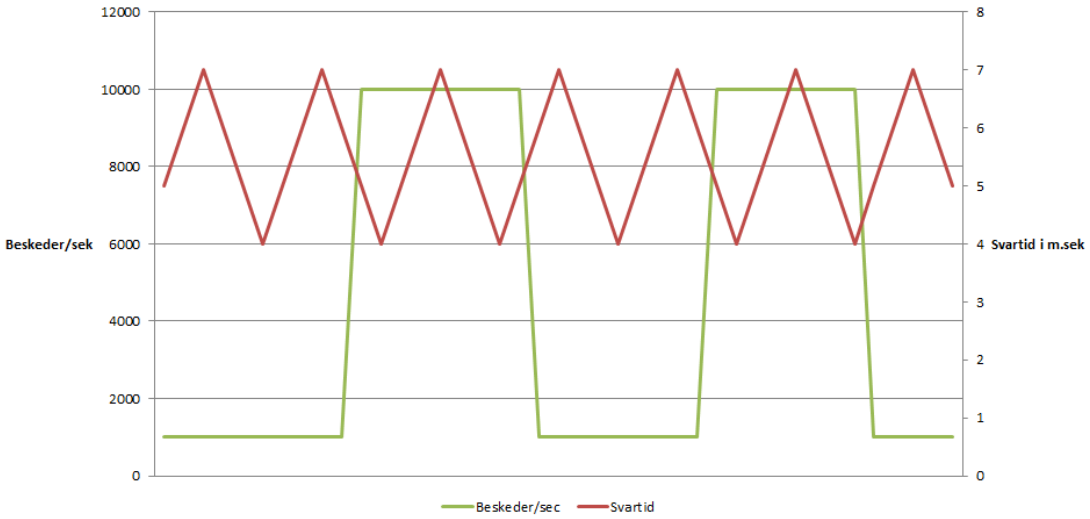
[5] Peak defineres som den maksimale forventede belastning et system udsættes for. For de fleste IT systemer er der oftest tale om en såkaldt ”peak period”, dvs. et bestemt tidsrum, hvor system er ekstra belastet. Typisk er denne periode af begrænset varighed.

En spiketest udføres ved at belaste systemet normalt i en kort periode, herefter pludseligt skrue belastningen op til peak i kort periode, for herefter at skrue ned igen.

Under disse skift i belastning, skal systemets svartider være upåvirket, dvs. systemet skal fortsat overholde sine svartidskrav.

Udførelse af spiketest:

- Definer kravene til spiketesten, dvs. størrelse, varighed og antal af spikes.
- Foretag en spiketest
- Mål på svartider, svartidsvarians, ressourceforbrug under testen.



Figur 7 - Eksempel på resultatet fra en spiketest.

Som det ses af eksemplet udviser det tænkte system den ønskede kvalitet. Systemet svinger svartidsmæssigt en smule, hvilket er normalt, men svartiderne er generelt upåvirkede af belastningsgraden, så længe systemets maksimale kapacitet ikke overstiges.

3.3.3 Stabilitetstest

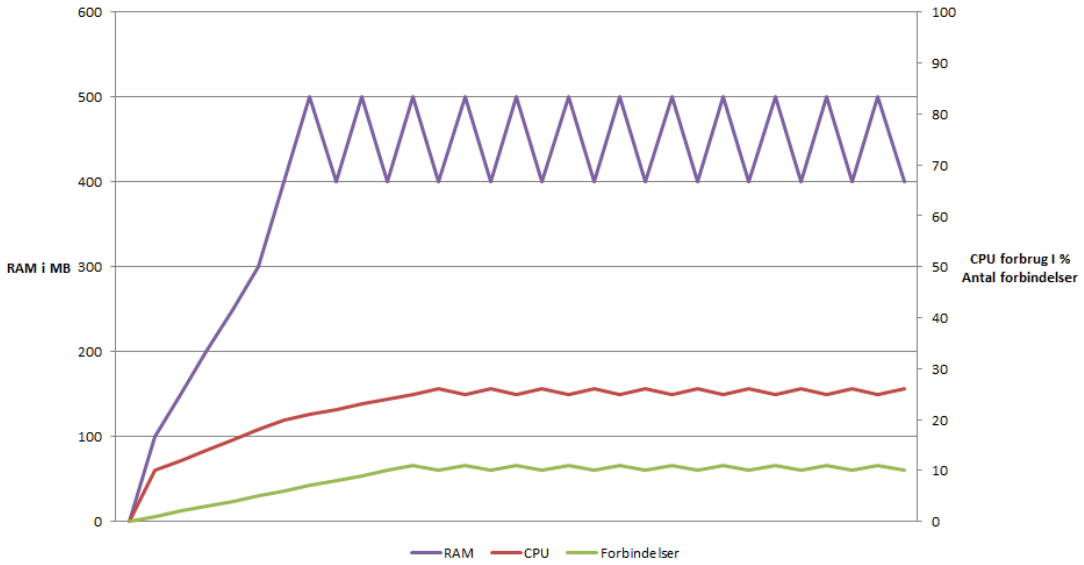
Stabilitetstest. En stabilitetstest foretages for at sikre, at systemet ikke degraderer hen over tid.

Testen udføres ved at belaste systemet over i længere periode, mens man overvåger systemets ressourceforbrug.

Under testen er det normalt, at systemet har en initialt stigende ressourceforbrug, men ressourceforbruget bør efter en kort indkøringsperiode stabiliseres.

Udførelse af stabilitetstest:

- Foretag en stabilitetstest
- Mål svartider, svartidsvarians, ressourceforbrug under testen.



Figur 8 - Eksempel på resultat fra stabilitetstest.

Som det ses af eksempel, så ud det tænkte system de ønskede kvaliteter. System har et stigende ressourceforbrug i perioden efter opstart, men alle ressource stabiliseres efter en kort indkøringsperiode, og svinger derefter indenfor et relativt lille bånd.

4. Konklusion

Et moderne IT system bør altid gennemgå et kvalitetssikringsforløb inden det frigives til det de endelige slutbrugere. Erfaringerne viser dog, at en ensidig fokus på funktionalitet ikke i tilstrækkelig grad sikrer, at systemet er velfungerende og anvendeligt for slutbrugeren. Mange projekter har oplevet forsinkelse under udrulning, nogle gange flere år, grundet manglende kvalitetssikring af performance, inden overdragelsen af projektet til drift.

Det er derfor Lakesides klare anbefaling, at ethvert moderne IT system også kvalitetssikres på performance siden, og at performancetests indarbejdes, så det bliver en naturlig del af udviklings- og kvalitetssikringsprocessen.

De udgifter der er associeret med denne ekstra kvalitetssikring er hurtig sparet, ved at flere hundrede medarbejde kan udføre deres arbejde uden forsinkelser, og at udrulningsplaner overholdes.

Det er Lakesides erfaring, at der selv i de bedst designede systemer, vil kunne opstå situationer, hvor systemet enten udsultes eller ikke performer korrekt. Med faste procedurer for kvalitetssikring af performance, vil langt de fleste af disse situationer blive fundet under udviklingsforløbet, og kan derved ofte løses billigere og hurtige, end hvis de først identificeres når systemet er sat i drift.

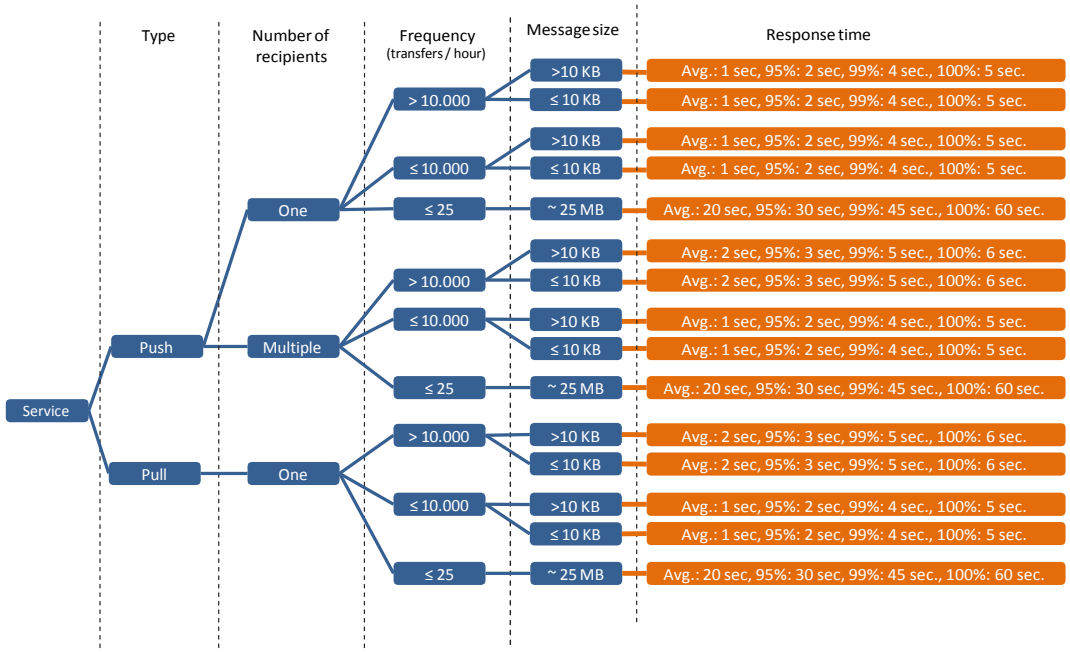
5. Bilag

5.1 Performancetest forudsætninger

Inden en performancetest kan afvikles er der en række forudsætninger, der skal være på plads:

- En klar specifikation af hvilke snitflader skal testes.
- En klar specifikation af hvordan snitflader testes, herunder input/output parametre og forventede datamængder.
- En klar specifikation af forventede svartider for hver testet snitflade, set fra slutbrugeren.
- En klar specifikation af hvordan svartid defineres, dvs. hvor tiden måles fra, og hvordan eksterne komponenter indregnes.
- En klar specifikation af baggrundbelastning baseret på eksisterende systemer eller forventet brugsmønster.
- Såfremt systemet er afhængig af en underliggende database, skal der udarbejdes en klar specifikation af forventet størrelse på databasen under kørslen, og fordelingen af data i databasen.

Disse data danner tilsammen en svartidsmatrice, der dels angiver antallet af nødvendige tests, men også hvordan hver tests udføres og godkendes.

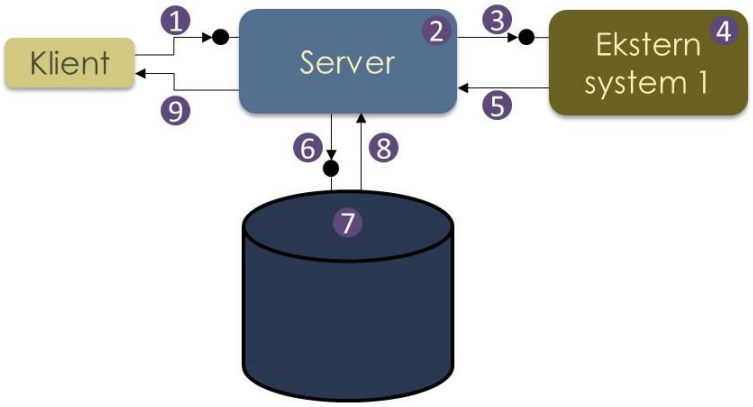


Figur 9 - Eksempel på svartidsmatrice for en enkelt snitflade.

5.2 Performancetest svartidsfordeling

Et system er sjældent en selvstændig isoleret komponent, men indgår oftest i et komplekst IT system, hvor andre komponenter og systemer kan påvirke performance. En forudsætning for performancetesten er derfor også en klar specifikation af, dels hvordan en svartid måles, og dels hvordan svartiden fordeles.

Når en slutbruger udfører en handling i en klient, observerer denne en svartid fra systemet. Denne svartid er summen af en række opgaver, som brugeren har affødt ved at udføre den enkelte handling i klienten. Der skelnes derfor oftest mellem et overordnet svartidskrav registreret hos slutbrugeren og så svartidskrav til det enkelte komponenter. Summen af alle indgående opgaver, skal dog altid være under slutbruger svartidskravet.

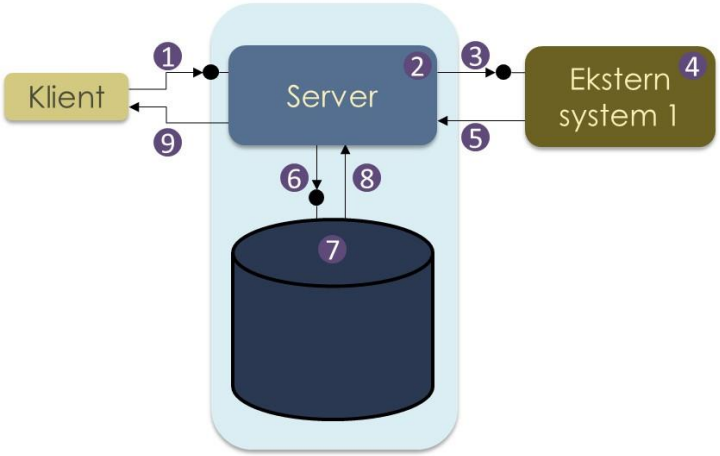


Figur 10 - Eksempel på affødte opgaver.

Som det fremgår af Figur 10, så består et yderste forsimplet eksempel af ni affødte opgaver:

1. Netværkstid mellem klient og server.
2. Behandlingstid i serveren
3. Netværkstid mellem server og ekstern komponent.
4. Behandlingstid i ekstern komponent.
5. Netværkstid mellem ekstern komponent og server.
6. Netværkstid mellem server og egen database.
7. Behandlingstid i egen database.
8. Netværkstid mellem egen database og server.
9. Netværkstid mellem server og klient.

Et rent serversystem opdeles typisk, så netværkstid og eksterne systemer ikke indgår i serverens svartidsansvar. Det betyder, at en performancetest af et serversystem vil typisk foretages direkte på systemets snitflader, se Figur 11.



Figur 11 - Eksempel på afgrænsning af svartidsansvar.

5.3 Performancetestplan

Ideelt er testmiljøet en identisk kopi af produktionsmiljøet, idet dette vil resultere i de bedst mulige forhold for performancetesten. Typisk er dette dog ikke økonomisk eller teknisk muligt, og performancetestmiljøet afviger derfor oftest i større eller mindre grad fra produktionsmiljøet. Det er derfor en forudsætning for performancetesten, at disse afvigelser klarlægges, og at konsekvensen af afvigelserne dokumenteres.

Udover dokumentation af miljøet, skal der udarbejdes en testplan for performancetesten, så testmiljøet kan reserveres, og evt. påvirkninger af eksterne komponenter kan koordineres.

En testplan bør som minimum afsætte 2 gennemløb af hver test, så målinger kan verificeres og evt. uforudsete udefrakommende påvirkninger ikke forsinker testplanen.

I forbindelse med testplanen skal der også etableres krav til omkringliggende komponenter, såfremt systemet har afhængigheder til sådanne. Dette kan f.eks. være andre systemer eller en underliggende database. Såfremt en eller flere af disse eksterne komponenter ikke er tilgængelig under testen, så skal det dokumenteres, i hvilket omfang det påvirker test, og hvordan der kompenseres for manglen.

5.4 Performancetest afvikling

Til afvikling af performance testen benyttes typisk to værktøjer:

- Værktøj til afvikling af baggrundsbelastning.
- Værktøj til måling af svartider.

Til afvikling af baggrundsbelastning benyttes oftest et distribueret klientsystem, der kan afvikles for flere forskellige maskiner samtidig. Derved kan man generere en realistisk baggrundsbelastning, uden at klientmaskinerne bliver en flaskehals.

Der findes på markedet i dag både en række kommercielle og ikke kommercielle produkter til afvikling af distribueret baggrundsbelastning. Blandt de ikke kommercieller er Apache JMeter og Grinder de mest udbredte.

Til måling af svartider findes der flere muligheder, lige fra lavpraktiske brug af et stopur til avancerede programmer, der registrerer hvornår et skærmbillede er færdigt med at indlæse.

I et system, hvor klienterne ikke er en del af leverancen, f.eks. webservices, kan man typisk benytte baggrundsbelastningsværktøjerne til også at verificere svartider, idet disse systemer oftest også logger eksekveringstider for de forespørgsler der foretages.

5.5 Andre parametre

I forbindelse med en performancetest, er der en række andre parametre der er interessante, ud over selve svartiden:

- CPU belastning.
- Hukommelsesforbrug.
- Åbne filer.
- Åbne netværksforbindelser.

Disse parametre indgår alle i en samlet vurdering af et systems performance.

6. Referencer og kilder

Reference-id	Indhold / Overskrift	Henvisning
[WPERF]	What is Computer Performance?	http://en.wikipedia.org/wiki/Computer_performance
[DOSwiki]	Denial-of-service attack	http://en.wikipedia.org/wiki/Denial-of-service_attack