SISU

PUBLIKATION 96:01

RAPPORT – JANUARI 1996

Prestanda för interaktiva Web-system

– en arbetsmetod för test och optimering av Web-system

Peter Johansson

SVENSKA INSTITUTET FÖR SYSTEMUTVECKLING

SISU

Sammanfattning

För den som sätter upp en interaktiv Web-tjänst är det viktigt att tjänsten är tillräckligt snabb. Annars riskerar den att förlora sina användare. Problemet är att det är svårt att säga vilken kapacitet ett Web-system har i förväg. En anledning till detta är att ett Web-system är uppbyggd av många olika komponenter, som serverprogram, gatewayprogram, databashanterare m m. Denna rapport syftar till att ge en arbetsmetod för att mäta prestandan hos ett Web-system samt att optimera det för att klara uppsatta prestandakrav.

För att kunna göra prestandatester krävs att vissa förutsättningar först fastställs. Det är bl a uppgifter om antalet användare, hur de beter sig vid användningen, datamängder och liknande. Utan dessa kan inga realistiska tester göras. Det bästa underlaget för att sätta dessa förutsättning är att ha resultat från systemet i drift sen tidigare. Om inte detta finns så måste uppskattningar av användningen göras, något som kan vara väldigt problematiskt.

Själva testmetoden går ut på att användningen simuleras. Datorer kopplade till samma lokala nätverk ställer frågor till Web-systemet på ett sätt som försöker efterlikna vanliga användares. Web-systemet testas på dettas sätt som en svart låda där alla olika komponenter testas tillsammans och inte var för sig.

Om Web-systemets prestanda inte är tillräckligt bra enligt de prestandamål som satts upp så måste den optimeras. Varje komponent kan optimeras var för sig, t ex antalet serverprocesser, programmeringsspråket för gatewayprogrammen m m. Men de måste också optimeras tillsammans. De ska alla använda gemensamma resurser som minne och processor. Det finns många olika variabler att ta hänsyn till vid denna optimering.

När väl prestandan är tillräckligt bra så är det dags att driftsätta Web-systemet. Då gäller det att övervaka den för att se hur den verkliga belastningen blir. Ligger denna nära den kritiska belastningen är det dags att göra nya tester och optimeringar.

Innehåll

1 Introduktion

Att i förväg säga vilken kapacitet en Web-server kommer att få är mycket svårt. Web-serverns prestanda kan variera stort beroende på dator, operativsystem, serverprogram, konfiguration, storlek på dokument m m.

En Web-server för interaktiva tjänster, ett Web-system, består av många olika komponenter (bild 1) som alla påverkar prestandan. För några av komponenterna, som serverprogrammet, finns det vissa möjligheter att förutsäga påverkan till viss del då de alltid utför samma uppgift. Andra komponenter däremot, som gatewayprogram och databas, kan ha stora variationer i påverkan beroende på tillämpning då olika tillämpningar kan utföra väldigt olika uppgifter. T ex ett program som beräknar summan av värden från ett formulär kräver betydligt mindre datorkraft än ett som gör en sökning i en databastabell innehållande 10 000 rader.

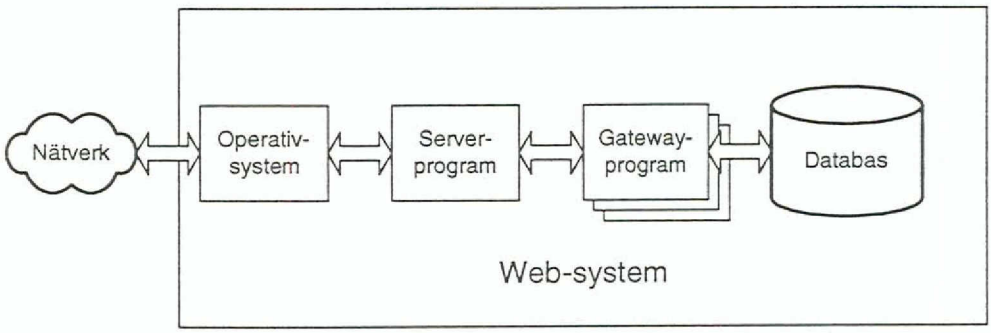

Bild 1 *Ett Web-system består av många olika program som kommunicerar med varandra.*

Denna rapport beskriver en metod för att testa och optimera ett interaktivt Web-system. I kapitel 2 beskrivs de testparametrar som först måste fastställas innan relevanta tester kan göras. När detta är gjort kan tester med simulerad användning göras enligt en metod som beskrivs i kapitel 3. Resultaten från testerna kan användas för att optimera de olika komponenterna i Web-systemet, vilket kapitel 4 behandlar. Kapitel 5 tar sedan upp saker att tänka på när systemet har tagits i drift.

2 Fastställ testparametrar

För att kunna göra realistiska prestandatester av ett Web-system så måste först vissa testparametrar fastställas, som t ex antalet användare som ska hanteras, datamängder i databasen etc. Dessa uppgifter används för att skapa indata för testerna och för att verifiera att tjänsten uppfyller prestandakraven. Det är mycket viktigt att testparametrarna anges korrekt, eftersom nyttan av testresultaten beror på hur väl dessa stämmer överens med verkligheten.

Testparametrarnas värde varierar beroende på användarnas beteende, t ex hur ofta och vilka delar av tjänsten de använder. Beteendet kan vara svårt att förutsäga, så om det finns information från tidigare användning av Web-systemet så kan denna vara mycket värdefull.

2.1 Förutsättningar för testet

Den viktigaste parametern för testerna är det genomsnittliga accessmönstret för Web-systemet, d v s hur ofta olika funktioner och dokument efterfrågas. Ett exempel kan vara att 50% av inkommande request är för bilder och färdiga HTML-dokument, 20% är för ett visst gatewayprogram som ställer frågor mot databasen, 25% för ett annat gatewayprogram som även det arbetar mot databasen och de sista 5% är för ett gatewayprogram som inte behöver tillgång till databasen. Varje sådan post måste sedan delas upp i ytterligare mindre delar. T ex så är var 10:e request av bilder/HTML-dokument för en speciell bild, var 20:e för ett visst HTML-dokument o s v. För gatewayprogrammen görs en uppdelning på samma sätt, fast då uppdelat på parametervärdena.

Det kan vara svårt att ta fram ett sådant accessmönster. Då kan gamla serverloggar vara mycket värdefulla och användas för att ta fram den statistik som behövs för att få fram ett relevant mönster. Ännu större problem blir det om accessmönstret måste uppskattas. Följande exempel visar hur en sådana uppskattning kan göras:

> AB Småannonser (ABS) har ett Web-system där användarna kan söka bland radannonser. Antag att en användare kommer in i systemet via ABS hemsida som består av ett färdigt HTML-dokument och tre bilder. Användaren går därefter vidare till söksidan, som förutom det färdiga dokumentet innehåller två bilder varav den ena är samma logotyp-bild som finns på hemsidan. Här gör användaren en sökning, med 50% sannolikhet att hon fyller i textfältet med ett sökord och 50% att hon inte gör det. Programmet som gör sökningen skapar ett dokument med en lista med rubrikerna från matchande annonser. Med 75% sannolikhet nöjer sig användaren med träfflistan och väljer att titta på en annonsen, vilket hon gör genom att klicka på annonsens rubrik i listan. Detta resulterar i att ett annat gatewayprogram exekverar och skapar dokumentet med annonstexten. Om användaren inte hittade någon annons i träfflistan så gör hon en ny sökning, denna gång med ett sökord. Användaren kommer nu troligen att nöja sig med resultatlistan och väljer därefter att titta på en annons.

Från detta beteende sammanställs sedan antalet requests till olika dokument för att ta reda på accessmönstret. När det gäller HTML-dokumenten och bilderna är de enkla att räkna:

Dokument	# requests
/ (hemsidan)	1
/sök.html	1
/logo.gif	2
/bild1hem.gif	1
/bild2hem.gif	1
/bild1sök.gif	1

Något svårare blir det att sammanställa antalet requests för programmen. Till en början var det 1*0,50=0,50 requests utan parameter till sökprogrammet. Sedan var det 1*0,50=0,50 requests med sökord (värdet för detta antas inte spela någon roll, däremot spelar det roll om det finns eller ej). Med 25% sannolikhet görs en ny sökning med ett sökord, d v s 1*0,25=0,25 requests. Användaren väljer till slut att titta på en annons, vilket resulterar att annonsprogrammet får en request. Sammanställt för gatewayprogrammen blir det:

Dokument	# requests
Sökprogram m parameter	0,75
Sökprogram utan parameter	0,5
Annonsprogram	1

Tyvärr är det inte riktigt så här enkelt att göra beräkningarna. Läsprogrammet cachar troligen logotyp-bilden från hemsidan och hämtar inte den på nytt när söksidan begärs. Det skickas alltså bara en request för denna bild. För att göra det ännu mer komplicerat så är det inte säkert att några färdiga dokument skickas överhuvudtaget om de finns i läsprogrammets cache sedan tidigare. Om det är så eller ej beror på hur ofta användaren kommer tillbaka till tjänsten.

Ett annat slags accessmönster är användarnas nätverksförbindelser. Långsamma och snabba förbindelser ger olika förutsättningar för belastningen av Web-systemet. För en intern tjänst kan det antas att alla användare kommer ha snabba förbindelser. För många publika tjänster däremot kommer en stor del av användarna ha långsamma modemförbindelser till någon Internet-leverantör.

För att Web-systemet ska belastas riktigt vid testerna så måste databasen innehålla samma informationen som kommer att finnas där när systemet är i drift. Om det är stor omsättning på informationen så bör den åtminstone innehålla representativ information, både mängd och innehåll. Innehåller den för lite är det risk att

sökningarna går snabbare än de borde och belastningen därmed blir för liten. Likaså bör innehållet vara liknande det i drift så att sökningarna under testet belastar systemet korrekt.

2.2 Prestandakrav

För att kunna avgöra om Web-systemets prestanda är tillräcklig måste det finnas prestandamål. Detta är krav när det gäller svarstider, t ex hur länge ska användaren behöva vänta i normala och värsta fall. Olika krav kan sättas på olika slags frågor. Som exempel kan requests för databassökningar tillåtas ta längre tid än de för bilder. Dessa värden jämförs med testresultaten för att verifiera att systemet har den kapacitet som krävs.

Nästa parameter som behövs är antalet requests som kommer att genereras under en tidsperiod. Denna kan räknas fram från antalet samtidiga användare som arbetar mot Web-systemet vid ett givet tillfälle, nämligen det när belastningen antas vara störst. Även här behöver användarnas beteende uppskattas. Exemplet nedan visar hur denna beräkning kan gå till:

> ABS har den största tillströmmningen av användare vid lunchtid. Antag att maximalt 80 användare arbetar med deras Web-system vid ett tillfälle och att användarna har beteendet som beskrivits tidigare. För att komma vidare måste uppskattningar göras för hur länge de läser dokumenten:
>
> Hemsidan innehåller väldigt lite information och användaren kan snabbt klicka sig vidare till söksidan. I genomsnitt tar det 15 sek innan söksidan laddas. Söksidan kräver något mer tankearbete, användarna ska ju bestämma sig för sökvillkor, så kanske 45 sek. Träfflistorna kan vara rätt långa, så användarna får spendera en del tid där innan de väljer att gå vidare, antag att detta tar t ex 1 min 30 sek. Totalt blir det 2 min 30 sek = 150 sek. Fast 25% av användarna gör en ny sökning och genomgång av träfflistorna, vilket tar ytterligare 2 min 15 sek = 135 sek. I genomsnitt spenderar därför en användare (0,75*150+0,25*(150+135))=185 sek. Under denna tid har användaren gjort 8,25 requests, d v s en request ca var 22:a sekund. 80 samtidiga användare gör att ABS Web-system hanterar 80*(1/22)=3,6 requests/sekund = 13000 requests/timme.

Detta beräknade värde avgör hur många requests som Web-systemet måste klara att hantera enligt de fastställa förutsättningarna i accessmönster och användarnas beteende.

3 Test av Web-systemet med simulerad användning

En metod för att testa Web-systemets kapacitet är att simulera användarnas beteende. Testresultaten visar hur hårt systemet tål att belastas innan det inte klarar att hantera fler requests. Om uppmätt prestanda uppfyller de uppsatta målen är systemet sedan färdigt att sättas i drift eller i annat fall så måste det optimeras. Testresultaten kan också användas som underlag för att göra optimeringar.

3.1 Genomförande av test

Testet görs med hjälp av program som exekverar på en eller flera datorer kopplade till Web-systemet via ett lokalt nätverk, se bild 2. Programmen belastar systemet med requests på samma sätt som en användare gör enligt det tidigare definierade access-mönstret. I testet fungerar Web-systemet som en svart låda. På så sätt testas hela systemet och inte enbart de olika komponenterna var för sig. Genom att ändra belastningen kan den simulerade användningen varieras:

- Variera antalet samtidiga användare som skickar requests till Web-systemet. Observera att detta inte motsvarar det antalet användare som jobbar mot systemet samtidigt. De simulerade användarna skickar hela tiden en ny request direkt när de fått svar på den tidigare.
- Variera hur accessmönstret av dokument som efterfrågas ser ut. Se kapitel 2.

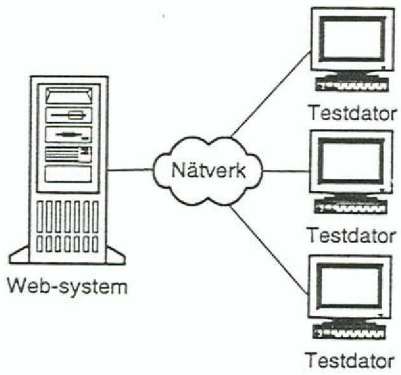

Bild 2 *Datorer simulerar användare genom att ställa frågor över nätverket mot Web-systemet för att testa dess prestanda.*

Ett program som kan användas för att utföra ett test enligt denna metod är WebStone från Silicon Graphics. Mer information om WebStone finns i kapitel 6.1.

Testprogrammet kommer bara att mäta svarstider och antalet uppkopplingar som Web-systemet hinner hantera. Med hjälp av olika slags verktyg kan även andra delar av systemet mätas. Det kan vara hur mycket olika resurser används, t ex minne och processor, vilka program som används, hur databasens konfiguration

fungerar m m. Dessa mätverktyg körs då på Web-systemets värddator under tiden simuleringen pågår. Vilka verktyg som finns beror mycket på vilket operativsystem och vilken databashanterare som används på värddatorn.

3.2 Resultaten

Resultaten kommer dels från de loggfiler som simuleringsprogrammet skapar och dels från mätverktyg till operativsystem och databashanterare. Loggfilerna ger information om svarstider och storlek på de dokument som hämtades. Dessa värden kan användas för att få maximala och normala svarstider, antalet misslyckade requests, antalet requests per sekund och storlek på nätverkstrafiken. Mätverktygen kan användas för att se hur datorns olika resurser använts, som minne, hårddisk och databashanterare. Dessa resultat kan användas som underlag för optimeringar.

Genom att göra flera simuleringar där antalet simulerade användare varieras och dokumentblandningen hålls konstant går det att se hur Web-systemet hanterar ökat antal requests. Diagram 1 visar hur antalet fel, d v s requests som systemet inte hinner besvara innan time-out ges, och den genomsnittliga svarstiden stiger med en ökning av antalet requests per timme. Ofta kan det finnas en kritisk belastning, en tydlig gräns när systemet inte längre klarar av att hantera flera requests. Denna syns i en snabb ökning av antal fel, se exempel i diagram 1a, och en ökning av genomsnittliga svarstiden, se diagram 1b.

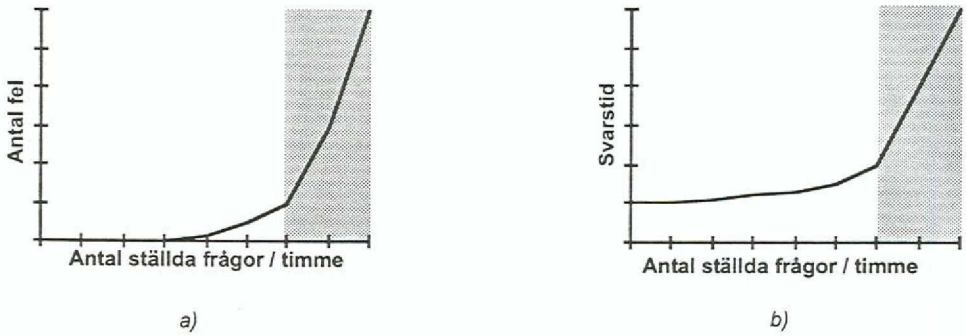

Diagram 1 *Hög belastning ökar antal fel och svarstid. a) visar hur antalet fel ökar med antalet ställda frågor per timme. Över en viss belastning kommer alla ytterligare frågor resultera i fel vilket betyder att systemet har begränsad kapacitet. b) visar hur svarstiden varierar med antalet ställda frågor. Den ökar sakta i början men blir fort längre när belastningen når en viss gräns.*

Dessa kurvor kan jämföras med de uppskattade värdena för användningen av Web-systemet. Ligger dessa värden under den kritiska belastningen med god marginal så har Web-systemet kapacitet för att hantera den tänkta användningen och systemet kan tas i drift. Det är viktigt att tänka på att den kritiska belastningen är den maximala belastningen som systemet kan hantera vid en given tidpunkt. Vid planering av användningen måste man därför ta hänsyn till att belastningen inte vid något tillfälle kan överstiga detta värde.

Om värdena befinner sig nära eller bortom den kritiska belastningen betyder det att kapaciteten inte är, eller snabbt riskerar att inte vara, tillräcklig. Då är det dags att optimera Web-systemet.

4 Optimeringar

Många olika komponenter är inblandade för att realisera databasfrågor i ett interaktivt Web-system. För optimal prestanda krävs att varje del är noga intrimmad. Om någon komponent fungerar dåligt så är det risk att hela Web-systemet får dålig prestanda.

4.1 Var optimeringar kan göras

Det finns flera möjliga flaskhalsar i hanteringen av en request. Serverprogrammet som tar emot en request måste hinna att hantera alla requests, annars köas de. Ineffektivt skrivna gatewayprogram drar ner prestandan. En dålig databasstruktur gör att databashanteraren får arbeta onödigt mycket. På alla dessa ställen kan optimeringar göras för att prestandan ska bli så bra som möjligt.

Serverprogrammet

När en request kommer till Web-systemet så tar ett serverprogram hand om denna. Beroende på dokumentnamn kan serverprogrammet skicka tillbaka ett färdigt dokument, eller så kan programmet utföra någon annan funktion för att se till att dokumentet skapas, t ex genom att starta ett gatewayprogram.

Serverprogrammet måste kunna hantera flera samtidiga requests för att de inte ska köas och svarstiden därmed blir för lång. Användare med långsamma förbindelser till Web-systemet blockerar nya förbindelser till serverprogrammet ända tills hela dokumentet är överskickat. En vanlig modell idag för att hantera många samtidiga förbindelser är att flera parallella serverprocesser används. När en request kommer till Web-systemet så dirigerar en huvudserverprocess den till en ledig serverprocess, som tar hand om den och väntar tills hela dokumentet är skickat till användaren innan det tar hand om en ny request. Så länge det finns lediga serverprocesser så kommer requesten genast att behandlas, se bild 3a. Om alla serverprocesser är upptagna så kommer requesterna placeras i kö tills någon process blir ledig vilket syns i bild 3b. En annan modell är att inte använda flera serverprocesser utan serverprogrammet är skrivet för att hantera flera requests parallellt. Denna metod är resurssnålare och börjar bli vanlig i nya serverprogram.

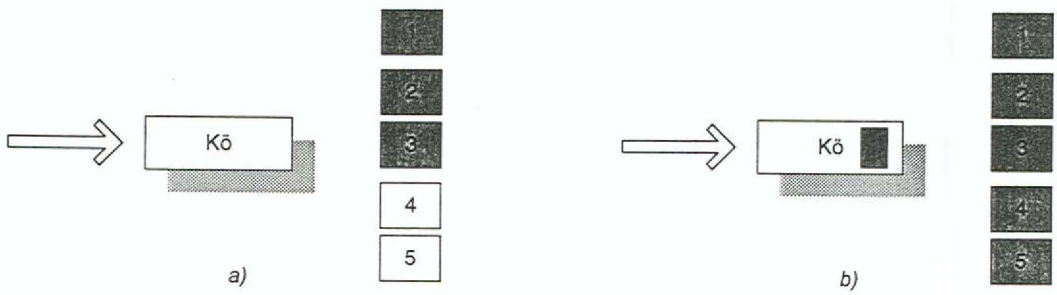

Bild 3 *Requests till Web-systemet tas om hand av serverprogram. Skuggade rutor är requests. I a) finns lediga serverprocesser, vilket inte finns i b) där en requests väntar i kön på att någon process ska bli ledigt.*

För att få ut så mycket som möjligt av systemet måste ett lagom antal serverprocesser användas. Lämpligt antal konfigureras efter den tänkta användningen och kapaciteten hos systemets värddator. Allt för många serverprocesser tar mycket resurser av dator och operativsystem, och drar därför ner prestandan på datorn, samtidigt som för få ger långa svarstider men inte utnyttjar datorn fullt ut. Om många användare har långsamma nätverksförbindelser behövs fler serverprocesser än om de har snabba förbindelser.

Användarens läsprogram etablerar en ny förbindelse till Web-systemet för varje dokument som hämtas. Problemet är att ett dokument som användaren ser i sitt läsprogram ofta är uppdelat på flera separata dokument i Web-systemet. Varje bild är exempelvis ett eget dokument. En del läsprogram kan ha flera parallella förbindelser till systemet och låser på så sätt upp många serverprocesser. Ett sätt att komma runt detta finns i de senaste versionerna av läs- och serverprogrammen. Istället för att använda en ny förbindelse för varje dokument så skickas fler dokument över en förbindelse. Detta ger snabbare överföring och minskar lasten på systemet då färre uppkopplingar görs, som alla kräver datorkraft.

Kopplingen mellan serverprogram och databas

Det finns flera sätt att integrera serverprogram och databaser. Vanligast är att använda fristående program, s k gatewayprogram, som serverprogrammet startar för att behandla en request från användaren. När gatewayprogrammet exekveras etablerar det en förbindelse till databashanteraren och ställer därefter en eller flera databasfrågor för att hämta information för att skapa dokumentet. Detta lämnas över till serverprogrammet som skickar det vidare till användaren. Gatewayprogrammet avslutas när dokumentet är färdigskapat. Nästa gång programmet ska användas startas det på nytt.

Gatewayprogrammen kan ha stor inverkan på Web-systemets prestanda. Om programmen är långsamma blir Web-systemet långsamt. Förutom att svarstiden blir lång så belastar programmen datorn så att andra gatewayprogram får vänta längre på gemensamma resurser som processortid och filer. Viktigt är också att databasfrågorna är effektiva. Dåligt skriven SQL-kod gör att databashanteraren får arbeta onödigt mycket, vilket bl a påverkar exekveringen av andra gatewayprogram som samtidigt gör sökningar i databasen.

Programmeringsspråket som används för gatewayprogrammen kan också påverka prestandan. Kompilerade program är vanligen snabbare än interpreterade program. Vilken effekt detta får på prestandan beror på hur mycket programmet utför. Gatewayprogrammen innehåller normalt få instruktioner så i många fall är det inget problem att använda program skrivna i interpreterande språk. Viktigare är att programspråket har en effektiv koppling till databasen.

Arkitekturen med separata serverprogram och gatewayprogram i bild 1 är den idag vanligaste lösningen. Den är bekväm på så sätt att det är enkelt att skriva och testa nya funktioner. Nackdelen är att ett externt program måste startas för varje request, något som tar mycket resurser på värddatorn. Detta beror på att programmet först måste laddas in från disk och sedan göra olika initieringar, som en

uppkoppling till databashanteraren, innan själva bearbetningen kan sätta igång. Dessutom måste det resulterande dokumentet slussas via serverprogrammet tillbaka till användaren, vilket blir ett extra steg.

Gatewayprogram som inte används så ofta påverkar inte systemets prestanda nämnvärt. Program som används kontinuerligt kan det däremot löna sig att göra något åt. En lösning kan vara att lägga in ny funktionalitet direkt i serverprogrammet och på så sätt undvika det extra steget att starta ett externt program för varje request. Flera serverprogram har möjlighet att göra detta, t ex Netscapes och Spyglass program. Via programmeringsgränssnitt kan ny funktionalitet läggas in i serverprogrammet. Referenser till några sådana programmeringsgränssnitt finns i kapitel 7.3.

Databas

Databasen innehåller information som gatewayprogrammen använder för att skapa dokumenten. När gatewayprogrammen startas etableras en förbindelse till databashanteraren. Flera program kan samtidigt vara uppkopplade mot databashanteraren. Gatewayprogrammen ställer frågor mot databasen och skapar från svaren de dokument som sänds tillbaka till användaren. I databasen är informationen organiserad för att programmen ska kunna hitta den information de söker efter.

En dåligt strukturerad databas kan innebära avsevärda negativa effekter på prestandan. Databashanteraren får arbeta intensivare med att hitta den information som söks. Genom en väldesignad databasstruktur går sökningar snabbare och databashanteraren blir mindre belastad. Detta problem är inte på något sätt specifikt för Web-system utan gäller vid all slags användning av databaser.

Databashanteraren kan oftast konfigureras på olika sätt för att förbättra prestandan. En inställning som kan påverka prestandan är allokering av primärminne för databuffertar. Mycket allokerat minne gör att mycket av informationen i databasen kan placeras i minnet så att den inte behöver hämtas från disk vid varje sökning. Detta gör att frågorna kan besvaras snabbare. Till många databashanterare finns verktyg som kan hjälpa till att beräkna hur stora databuffertar som behövs.

En del databashanterare klarar inte att många användare ställer frågor samtidigt utan att prestandan sjunker. En anledning till detta kan vara att databashanteraren försöker växla mellan frågorna så att en parallell bearbetning görs. Med för många frågor så använder databashanteraren mer tid åt att växla mellan frågorna än att besvara dem. Därför kan det vara nödvändigt att begränsa antalet samtidiga användare som kommer åt databashanteraren. Detta kan göras på olika sätt:

- Databashanteraren konfigureras så att bara ett visst antal användare får söka i databasen samtidigt. De andra kan köas upp eller så kan ett meddelande sändas tillbaka till användaren som meddelar att "för många användare söker just nu i databasen, vänligen försök igen".
- Antalet serverprocesser begränsas. På så sätt kan bara lika många användare vara inne i databasen samtidigt som det finns processer. Om detta tillvägagångssätt används så bör två separata omgångar processer användas där den ena

tar hand om databasfrågor och den andra levererar alla andra typer av dokument, se bild 4. Om inte detta görs kommer alla requests efter t ex bilder köas upp om alla serverprocesser jobbar med databasfrågor. Då drabbas denna typ av requests bara för att antalet samtidiga databasanvändare ska minskas.

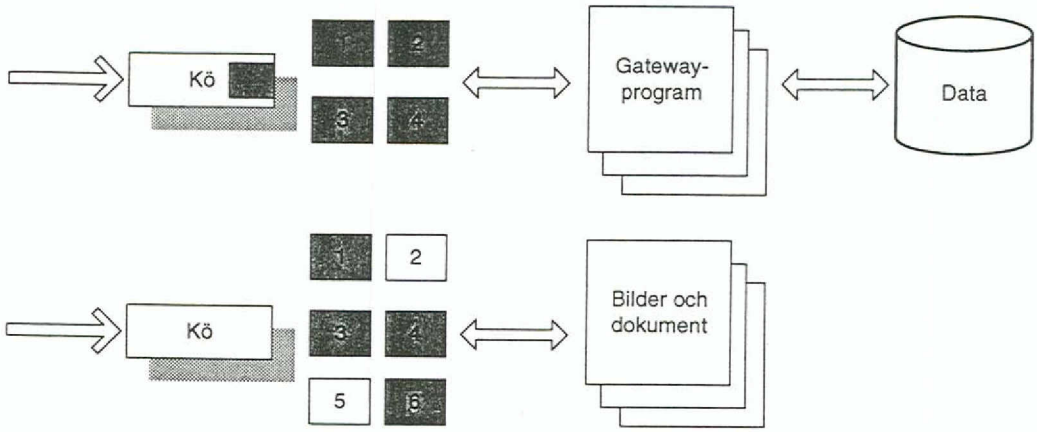

Bild 4 *Två omgångar serverprocesser tar hand om inkommande requests på olika portnummer. Den övre hanterar requests som genererar databasaccesser. I detta fall finns fyra serverprocesser som alla arbetar med requests och en request ligger i kön. Den undre hanterar vanliga dokument och där finns det lediga serverprocesser.*

Hårdvara och operativsystem

Prestandan beror givetvis också på den dator som används för Web-systemet. Serverprogram, gatewayprogram och databashanterare kräver alla datorresurser. Många av följande resurser kan vara inblandade och påverka prestandan:

- Primärminnet är för litet. Varje program tar utrymme i minnet. Om primärminnet tar slut så flyttas hela eller delar av ett eller flera program ut på hårddisken för att göra plats för programmet som exekverar just nu. Denna procedur tar väldigt mycket tid. Om detta händer hela tiden kan det vara en anledning till kraftigt försämrad prestanda. Ytterligare en anledning att öka primärminnet är för att utöka antalet databasbuffertar och på så sätt snabba upp databassökningar.
- Hårddisken är för långsam. Detta problem märks framförallt om stora delar av databasen ligger på disk och inte i primärminnet. Likaså får det effekt om program flyttas till disk tillfälligt för att göra plats för andra program i primärminnet.
- Processorn får arbeta maximalt hela tiden och har därför ingen mer kapacitet att ge. Detta kan uppstå om databasfrågorna kräver mycket bearbetning.
- Operativsystemet är ineffektivt eller felaktigt konfigurerat. Ofta handlar det om att drivrutiner för nätverket inte är konfigurerade för den typ av trafik WWW genererar. Hur operativsystemet fördelar minne och andra resurser mellan olika program kan också ha effekt på prestandan.

Nätförbindelse

Den nätförbindelse Web-systemet har ut till användarna kan vara en begränsande faktor. Ett Web-system skapar framför allt utgående nättrafik. Från användningsloggarna går det att ta reda på hur stor nättrafiken är vid olika tidpunkter. Om belastningen under långa perioder är nära den totala kapaciteten måste åtgärder vidtas. Antingen kan kapaciteten på förbindelsen ökas eller så kan dokumentens storlek minskas.

Olika typer av dokument skapar olika mängd nättrafik. Ett sätt att få ner trafiken kan vara att minska storleken på bilderna. Förutom att det minskar belastningen på nätförbindelsen så får användaren upp dokumenten snabbare i läsprogrammet. Ett annat sätt kan vara att minska storleken på träfflistorna från sökningar. Långa träfflistor är ofta besvärliga att leta genom för användaren så det spelar ingen roll att dessa kortas ned. En bättre lösning är att göra det lätt för användaren att förbättra sökvillkoren för en sökning om listan skulle bli alltför lång och samtidigt bara visa de första träffarna.

4.2 Sammanfattning och några tips

För att kunna göra en ordentlig optimering av ett Web-system krävs ingående kompetens om alla dess komponenter. Det viktigaste är dock att först lägga arbetsinsatsen där den gör mest nytta, d v s där chansen är störst att göra optimeringar med begränsad insats, för att sedan göra mindre förbättringar om det finns resurser tillgängliga. Nedanstående punkter kan ge några tips på var optimeringar kan göras för att förbättra prestandan hos Web-systemet:

- Undersök hur fördelningen av processortid ser ut. Om operativsystemet arbetar mycket tyder det på att processer kopieras fram och tillbaka till disk. Utökning av primärminnet kan avhjälpa detta problem. Om det är databashanteraren som tar den mesta processortiden så kan processorn behöva uppgraderas. Om detta inte är möjligt kan det vara nödvändigt att balansera systemet så att även andra program får processortid, vilket kan göras genom att tillåta färre samtidiga användare i databasen. Om processorn aldrig arbetar maximalt kan det betyda att antalet serverprocesser bör ökas och/eller att fler samtidiga användare bör tillåtas i databasen.
- Undersök om antalet databasbuffertar bör utökas. Använd eventuella verktyg som följer med databashanteraren.
- Testa programmeringsspråkets koppling till databashanteraren. Skriv några korta program som gör samma sak, fast i olika språk. Gör tester med dessa för att se om det är några större skillnader i prestanda.
- Vilken modell använder serverprogrammet för samtidig hantering av frågor? Parallell hantering utan att skapa flera processer är mest resurssnål och har bäst möjligheter att bli snabbast rent tekniskt sett. Annars bör ett lagom antal serverprocesser användas. Många requests av stora dokument och/eller gatewayprogram låser processerna under längre tid och då krävs många serverprocesser.

Det behövs även fler serverprocesser om en stor del av användarna har långsamma förbindelser.

- Gatewayprogram som används ofta bör kanske göras som utökningar av serverprogrammet.
- Om databashanteraren arbetar mycket så se över databasstruktur och koden för databasfrågor. En optimering som kan göras är att använda indexfiler.
- Primärminnet är en resurs som alla processer ska dela på. Det kan vara en av de viktigaste hårdvaruoptimeringar att göra.

4.3 Användning av fler datorer

Resonemanget hittills har mest handlat om att Web-systemet består av enbart en dator. De olika programmen försöker då konkurrera om tillgängliga resurser. Genom att dela upp belastningen på fler datorer så kan prestandan höjas. Det finns flera olika sätt att göra detta. Ett är att använda fler datorer som innehåller alla funktioner och har identiskt innehåll. Inkommande requests sprids mellan dem för att minska lasten på var och en. En annan variant är att olika funktioner läggs på olika datorer:

- En dator tar hand om alla requests, d v s serverprogram, alla färdiga dokument och gatewayprogram finns på en dator. Databashanteraren finns på en annan dator som gatewayprogrammen kopplar upp sig emot för att få svar på databasfrågorna.
- En dator med serverprogram tar hand om alla requests som rör färdiga dokument. På en annan dator finns databashanterare och serverprogram. Alla requests som kräver tillgång till databasen dirigeras till denna dator.

4.4 Testa på nytt

Efter gjorda optimeringar är det dags att testa på nytt enligt metoden i kapitel 3. Om resultaten nu visar att Web-systemet har tillräcklig kapacitet så är det färdigt att tas i drift. I annat fall måste nya optimeringar göras. Denna cykel upprepas tills man är nöjd med prestandan.

5 Web-systemet i drift

När Web-systemet tas i drift måste det övervakas. Detta måste göras för att kontrollera den verkliga belastningen. Testerna visar bara om systemet fungerar väl i en kontrollerad laboratoriemiljö under givna förutsättningar. De kan inte visa alla tänkbara problem. Nedan följer ett exempel på ett problem som inte testerna upptäcker:

> Vid testning används ett snabbt lokalt nätverk (10 Mbps). Antag att tjugo användare skickar requests samtidigt om olika filer med en storlek på runt 30 kB styck och att fem serverprocesser betjänar dessa requests. Det tar bara bråkdelen av en sekund att skicka tillbaka varje fil, så kön med requests till serverprocesserna kommer snabbt att behandlas, troligen på någon sekund. Antag istället att tjugo riktiga användare var uppkopplade med modem med en hastighet på 15 kbps och skickar samma requests. Nu skulle det ta 300/15=20 sekunder för de första fem att få svar på sina requests, ytterligare 20 sekunder för de fem nästa, och så vidare. Med tjugo serverprocesser skulle alla bara behöva vänta 20 sekunder tills hela dokumenten var överskickade. I modemfallet behövde inte varje serverprocess arbeta särskilt hårt, de väntar den största delen av tiden på att få sända nästa del av filen och systemet belastas inte nämnvärt av detta. Det kan däremot bli problem om tjugo användare med snabba förbindelser kopplar upp sig mot systemet om tjugo kopior av serverprogrammet körs.

Detta och andra problem kan vara näst intill omöjliga att få reda på. Därför går det inte att lita på att testerna visar på alla prestandaproblem. Med övervakning kan några problem kanske fångas upp.

5.1 Analysera

Genom att analysera loggarna från serverprogrammet under tiden systemet är i drift går det att få ut en hel del användbar information. Bl a går det att mäta den verkliga belastningen av systemet. Detta gäller både fördelningen av olika dokument samt hur mycket de används. Ett användningsområde för informationen kan vara att uppdatera de uppskattade testparametrarna med de verkliga. Genom att göra nya tester med denna indata så blir resultaten vid test av den kritiska belastningen bättre än tidigare.

För att analysera serverloggarna finns flera program tillgängliga. De presenterar framförallt statistik på vilka dokument som hämtas men även hur mycket information som skickats. I kapitel 6.3 finns referenser till olika analysprogram.

5.2 Nya optimeringar och tester

Om den dagliga övervakningen visar att den verkliga belastningen ligger nära den simulerade kritiska belastningen så måste nya optimeringar och tester göras tills prestandamålen åter är uppfyllda. På samma sätt kan nya tester göras om förutsättningarna ändras. Exempelvis om nya funktioner läggs in, ny layout m m. Detta ger nya värden för den kritiska belastningen.

6 Program för test och optimering

Som hjälp vid testning och optimering finns olika program tillgängliga. Här finns några exempel på några sådana som kan användas. Flera av dem kan hämtas ner via Internet.

6.1 Testprogrammet WebStone

WebStone är ett prestandatestprogram från Silicon Graphics. Själva testprogrammet fungerar enbart på Unix-datorer. Web-systemet kan däremot vara av vilken typ av dator som helst, bara den är kopplad till samma lokala nätverk. Mer information om WebStone finns på adressen

http://www.sgi.com/Products/WebFORCE/WebStone/

6.2 Prestandaverktyg för operativsystem och databashanterare

Verktyg för prestandamätningar och -optimeringar är beroende på operativsystem och databashanterare. För mer information om dessa hänvisas till dokumentation och annan tillgänglig litteratur för dessa.

6.3 Analysprogram för serverloggar

Det finns många program för att analysera serverloggar. Tabellen nedan innehåller några exempel:

Program	Beskrivning	Mer information
getstats	Skapar HTML-dokument över användning per tidsenhet, varifrån användarna kommer, vilka dokument som hämtats m m. Parametrar från formulär eller från kommandoraden.	http://www.eit.com/software/getstats/getstats.html
analog	Liknande getstats.	http://www.statslab.cam.ac.uk/~sret1/analog/
gwstat	Grafer som visar användningen av Web-servern.	http://dis.cs.umass.edu/stats/gwstat.html

Information om andra liknande analysprogram finns på Yahoo:

http://www.yahoo.com/Computers_and_Internet/Internet/World_Wide_Web/HTTP/Servers/Log_Analysis_Tools

7 Referenser

På Internet finns en del information som rör prestanda för Web-servrar. Här är en förteckning över dokument som använts till denna rapport

7.1 Testning

Dessa dokument handlar om olika testmetoder och varför man ska göra prestandatester.

Dokumentnamn	Dokumentadress
World Wide Web Server Benchmarking	http://www.sgi.com/Products/WebFORCE/WebStone/
World Wide Web Server Performance	http://www.sun.com/cgi-bin/show?sun-on-net/Sun.Internet.Solutions/performance/index.html
Why is a performance Web server important? WebForce FAQ	http://www.sgi.com/Products/WebFORCE/wffaq.html #whyserver

7.2 Prestandaresultat

Det har gjorts några prestandatester av olika serverprogram och olika serverdatorer. Detta är några av dem:

Dokumentnamn	Dokumentadress
Performance Benchmark Tests of Unix Web Servers Using APIs and CGIs	http://home.netscape.com/comprod/server_central/performance_whitepaper.html
Open Market Secure Webserver 1.1 och Netsite Commerce Server 1.12	http://www.zdnet.com/~pcweek/sr/1113/tsecure.html
Performance of Several HTTP Demons on an HP 735 Workstation	http://www.ncsa.uiuc.edu/InformationServers/Performance/V1.4/report.html
WebSite Performance Analysis	http://website.ora.com/devcorner/white/wsperf.html

7.3 Programmeringsgränssnitt för serverprogram

Funktionaliteten kan utökas av användarna i några serverprogram. För att göra detta möjligt har leverantörerna skapat programmeringsgränssnitt till programmen.

Dokumentnamn	Dokumentadress
THE NSAPI VERSUS THE CGI INTERFACE	http://home.netscape.com/newsref/std/nsapi_vs_cgi.html
Spyglass Server Application Development Interface	http://www.spyglass.com/techspec/adi_spec.html
THE NETSCAPE SERVER API	http://home.netscape.com/newsref/std/server_api.html