

När hög tillgänglighet inte blir hög

Nyckeln är balansgång mellan teknikåtgärder, komplexitetsrisker och kostnader



2011-05-25: Sven-Håkan Olsson

UNDVIK KÄNDA FALLGROPAR Vi lägger ofta ner både pengar och möda på att försöka skapa stabila it-applikationer med hög teknisk tillgänglighet. För många tjänster är tillgängligheten, "uptime", helt avgörande – att åstadkomma minst 99,99 procent tidsandel är ett vanligt verksamhetskrav. Ändå misslyckas vi förvånansvärt ofta.

När jag håller it-arkitektkurser brukar jag passa på att be om handuppräkring bland deltagarna. *Jag frågar hur många som har råkat ut för att deras plattform har gett stillestånd trots att den har varit designad för hög tillgänglighet och "failover"*. 25-30 procent brukar räcka upp handen. Det är ett förskräckligt dåligt betyg, direkt från verkligheten.

Ibland handlar mina konsultuppdrag om att utvärdera applikationer som visat sig ge för dålig tillgänglighet och haverera för ofta. Det är mycket intressant att bena ut orsaker. Vid något enstaka tillfälle har det inte gått att fastställa grundorsaken, men oftast går det tydligt att i efterhand se problemmönstren.

Och vad är det då som inte brukar fungera? Här kommer några exempel, utan inbördes ordning:

- Rätt konfiguration och inställningar har inte vidmakthållits över tid. Till exempel kanske man installerat en operativsystemsupdate som borde åtföljts av en konfigurationsändring i failoverdelen. Man har bytt till ett modernare system där inställningarna borde ha varit annorlunda. Eller också har man uppgraderat två klustermedlemmar var för sig och inte fått dem kompatibla efteråt.

- Eftersom failover oftast är komplext att sätta upp är risken för operatörsmisstag i största allmänhet stor.
- Planerad failoverlösning har i praktiken inte varit påslagen. Förekommer faktiskt förvånansvärt ofta!
- Det kan visa sig att ett felscenario inte ingår i det som failoverleverantören designat för. Vi hade till exempel fallet att AIX nästan fullständigt stannat till följd av ett disksystemsfel, men att operativsystemet inte gjorde failover till friskt reservsystem eftersom det inte var specificerat för att göra så i just den här situationen.
- Fenomenet "fel-men-inte-komplett-fel". Ett exempel var en halvt avgrävd kabel som knappt gav kontakt men någon gång ibland släppte igenom ett datapaket. Kommunikationslösningen som skulle ha gjort failover till en reservkabel fattade hela tiden beslutet att inte koppla över, trots att användarna upplevde att kommunikationen helt stannat av.
- Olika failoverlösningar i olika skikt ovanpå varandra interagerar på ett svåröverblickbart och resultatlöst sätt. Skikten kan exempelvis vara kabelfailover, operativsystemsfailover, filsystemfailover (ofta för så kallade SAN), applikationsserverfailover, databasfailover. Man kan till och med få självsvängning i helhetslösningen på grund av icke förväntad samverkan mellan skikt – systemet kanske inte hinner göra något vettigt eftersom det hela tiden byter mellan normal- och reservsystem.
- Dålig driftövervakning som inte fångar upp att en fungerande failover faktiskt skett till reservsystemet, men att normalsystemet nu är stendött sedan en vecka. Alltså finns ingen failovermöjlighet kvar vid ett nytt fel!
- Failover testas inte regelbundet. Och om det faktiskt testas så är testerna alldeles för snälla. Man måste brutalt dra ur kablar, stoppa processer, brotta ner operativsystem!

Detta är den viktigaste lärdomen: Schemalägg brutala failovertester, helst något olika varje gång.

Höga kostnader

Man ska vara medveten om att teknikplattformar som påstås ge hög tillgänglighet har en rejäl prislapp. Dels rör det sig om redundans, det vill säga dubbleringar av hårdvara, med tillhörande licenser. Dels är det avancerade mjukvaror som ska hantera automatisk överkoppling, failover, för applikationsservrar, integrationslösningar, ESB:er (Enterprise Service Bus), webbservrar och databaser. Licenspriserna är ofta oblyga.

Eftersom leverantörerna tjänar bättre på dessa avancerade lösningar än de enkla kan detta driva fram en överförsäljning och en överanvändning. Som köpare gäller det att se upp.

En annan kostnad är tidsåtgången för att hantera och konfigurera failover-lösningar eftersom de ofta är mycket komplexa. En tredje kostnad är kompetens. För att säkra

tillgången på kunnig personal behöver man ofta räkna med minst tre specialistpersoner för att klara personalomsättning, sjukdom och semester. Medarbetare måste också hållas välutbildade i lösningen och ges möjlighet att arbeta praktiskt med produkterna för att verkligen vara redo när det bränner till och något stannar.

Ytterligare en kostnad är utökad programmeringstid. I åtskilliga fall måste programkoden skapas utifrån en failoverlösningens specifika egenskaper. Testning kostar också mycket och kan vara synnerligen komplex att utföra för att verkligen prova ut relevanta stopp-scenarios.

För att klara testning behövs dessutom ett extra system för failovertest som har PRECIS samma konfiguration som produktionssystemet; samma uppsättning datorer, noder och mjukvarulicenser (även om testsystemet inte behöver dimensioneras för lika hög lasttålighet som produktionssystemet). Detta är en mycket rejäl kostnad. En tillräckligt rymlig budget för ett sådant testsystem glöms ofta bort initialt.

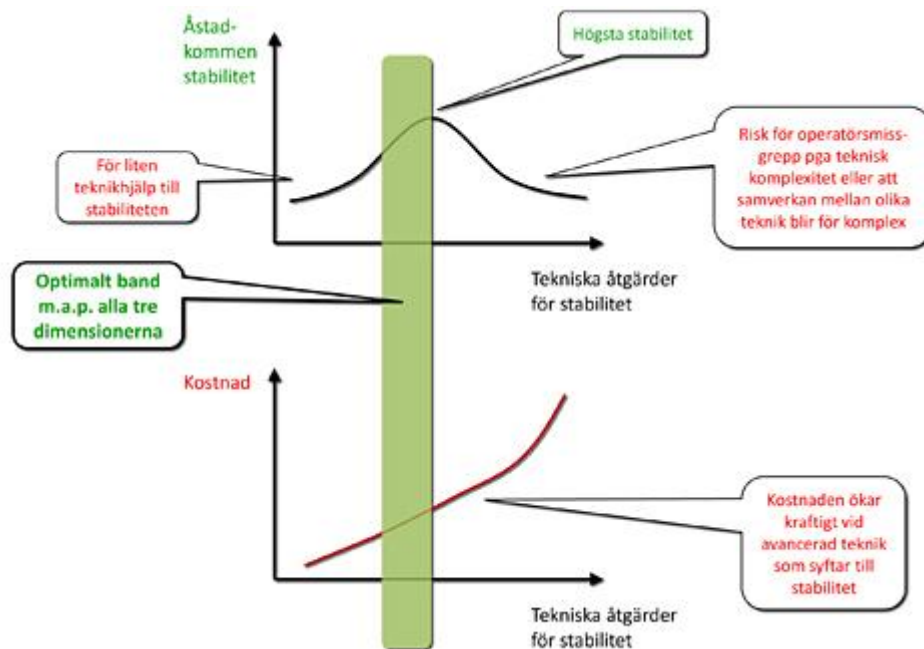
Optimering

Av genomgången ovan kan man förstå att hög komplexitet är en fiende till bra tillgänglighet. Det blir helt enkelt alltför svårt för driftpersonalen och systemutvecklarna att göra rätt. Här talar erfarenheten från faktiska stopp sitt tydliga språk.

Å andra sidan kan många tekniska åtgärder för att öka stabilitet ge klart högre teknisk tillgänglighet, men medför tyvärr samtidigt ökande komplexitet.

Kostnaden ökar också kraftigt om man inför avancerade failoversystem.

Det gäller alltså att optimera tre faktorer mot varandra: Teknikåtgärder för stabilitet, komplexitetsrisker och kostnad. Följande figur försöker illustrera detta.



Den övre kurvan exemplifierar att om vi inte inför några failoversystem alls så får vi dålig tillgänglighet. Men inför vi alltför komplexa failoversystem så får vi också dålig tillgänglighet. Det bästa läget återfinns mellan dessa ytterligheter.

Ett exempel kan vara att failover på webbservernivå oftast är mindre komplicerad, tack vare http-protokollets enkelhet. Om det går, välj alltså hellre en sådan stabilitetsökande åtgärd än applikationsserverfailover som vanligen är mycket mer komplex.

Den undre kurvan vill visa att kostnaderna ökar brant när man tillför avancerade failoverlösningar. Om man ska hitta bästa optimering kan det således bli i det "gröna band" som är markerat i figuren. Alltså, något till vänster om max stabilitet, eftersom kostnaden är lägre vänsterut.

Några sammanfattande råd

- Köp inte av slentrian alltför avancerade failoverlösningar, utan optimera istället på ett genomtänkt sätt relativt komplexitetsrisker och kostnad.
- Testa failover regelbundet och brutalt. Testa, testa, testa...
- Ett helt annat sätt att öka tillgängligheten är att minska beroendet av att alla datorresurser måste vara tillgängliga varje millisekund. Gör exempelvis genom att bygga mer köbaserade, asynkrona system (om färskhetskraven på informationen tillåter det).

Betraktelsen i den här trendspaningen har främst handlat om traditionella arkitekturer som till exempel webb > applikationsserver > relationsdatabas.

En del nya arkitekturer som nu provas ut kan gå att bygga på ett enklare sätt för att få hög stabilitet - några ord att internetsöka på inom denna nya sektor är REST, BASE, NoSQL, ATOM och idempotency. En nackdel här kan dock vara att användarna ofta har svårt att veta när en uppdatering verkligen "tagit". Den här samlingen nya arkitekturer tänkte jag återkomma till i en kommande trendspaning.



Sven-Håkan Olsson sysslar just nu med en iPad-tjänst för bolagsstyrelser och nämnder. I övrigt är han oberoende konsult som särskilt arbetar med att kombinera verksamhetsnytta med teknikhöjd. Han har en lång karriär sedan 70-talet som it-konsult (it-arkitektur, systemdesign, programmering, reviewer, utredningar, kursledning). Sven-Håkan är också medgrundare av Know IT och var dess teknikchef 1990-2003. Han utsågs till en av "Sveriges topputvecklare" av Computer Sweden 2008 och 2010.

Sven-Håkan håller regelbundet kurser åt Dataföreningen Kompetens. Läs gärna mer på hans blogg www.definitivus.se.

[Sven-Håkan Olsson](#)