Veckans teknikspaning

Rädda ditt data

Säkerhetskopiering och återställning av asynkrona system

2013-06-03: Sven-Håkan Olsson

SÄKERSTÄLL DATA En applikation som har hand om information med höga krav på måste ha en återställningsrutin som inte tappar bort data efter en krasch. En sådan rutin behöver vara ganska komplex och bestå av både manuella och automatiska steg.

Ganska ofta stöter jag på helt otillräckliga rutiner för säkerhetskopiering – och framförallt för återställning. Inte sällan handlar det om att utvecklarna tänker "Det där får väl driften ordna sedan" och att driftgänget tänker "Vi har inte hört om några speciella krav, så vi skapar väl en standardbackup". Men för system som har någon slags asynkron funktionalitet duger detta oftast inte alls.

Vad menas då med asynkront? Vardagliga funktioner som inläsningar av batchfiler från andra system bör faktiskt räknas som asynkrona ur den här synvinkeln. System där det ingår en kö-komponent, där EDIFACT-data kommer in, eller att det kommer in asynkrona aviseringar via ett nav eller en buss, räknas som asynkrona.

Det kan också handla om att applikationen är en del i en EDA (Event Drive Architecture) eller att systemet hämtar nyttodata från RSS/Atom. En asynkron karaktär kan ofta vara väldigt lämplig för en applikation i sig, och för dess integrationer. Men det finns nackdelar när det gäller just säkerhetskopiering och återställning.

Grundproblemet

Grundproblemet är att hitta konsistens för allt data för en viss tidpunkt. Ta följande exempel: En relationsdatabas kan oftast återställas till läget alldeles före diskkraschen skedde eller före applikationsbuggen uppkom som ställde till det i databasen – förutsatt att transaktionsjournalen överlevt. Men ungefär samtidigt som kraschen inträffade så pågick databasuppdateringar orsakade av asynkrona stimuli.

Hur vet jag nu exakt vilka köinläsningar som ingick i det som sedan återställdes och hur många som gick förlorade? Hur vet jag vilka delar av en batchfil som lyckades bli inlagd om inläsningen pågick vid kraschen? Eller vilka av batchfilerna, om det var flera på gång?

Här har olika teknikplattformar olika egenskaper, det kan finnas rollback-commit innefattande både kö och databas så att problemet minskar, men med alla integrationer inräknade brukar det alltid finnas ett tidsglapp kvar där man inte riktigt kan veta säkert vad som kommit med.

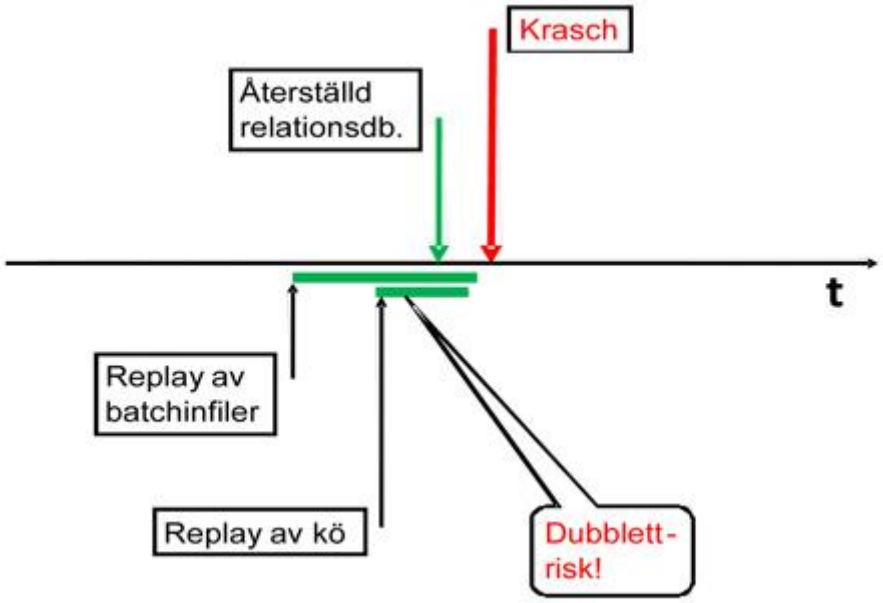
Man börjar inse att en korrekt återläsningsprocedur efter en krasch eller katastrof många gånger inte kan vara en helautomatisk operation. Någon mycket applikations- och integrationskunnig måste faktiskt undersöka förhållandena innan man kan godkänna driftsstart igen, läsa driftloggar och applikationsloggar samt göra detektivarbete.

Visst kunde man tänka att ALLA felvarianter skulle täckas i en förprogrammerad restore-rutin men då blir i sin tur den rutinen mycket komplex, svårtestad och en risk i sig.

Korrekt återställning och ominläsning

Hur går då en korrekt återställning till? Jo, om man använder relationsdatabas så återställer man först den till så färskt läge som möjligt, baserat på dess transaktionsjournal. Därefter behöver man ofta kunna köra om ett antal inläsningar av asynkrona händelser, kö-poster eller batchfilrader/batchfiler.

Man måste helt enkelt backa en bit i tiden så man är säker på att allt som inte kommit med i relationsdatabasen blir med. Många kölösningar klarar av att utföra sådan återuppspelning, liksom RSS/Atom. En batchfilinläsning kan man köra om från början, om man hållit ordning på filerna.



Självklart får inte till exempel autogirodebiteringar eller lönetransaktioner bearbetas dubbelt. Men eftersom det alltså ofta inte går att hamna i exakt rätt ögonblick kommer några av dem säkert att dyka upp igen trots att de redan finns i den återställda databasen. För att undvika det måste varje post som ska läsas in asynkront få någon slags unik identitet eller tidsstämpel, så att mottagande system har en chans att skilja agnarna från vetet. Ett engelskt namn på sådan dubblettverkanseliminering är **idempotency**.

Se upp med sänd asynkron information

Problemet handlar inte bara om mottagning. Om det kraschade systemet sänder asynkron information till andra system kan det också bli problem. Man kunde tänka sig att ett löpnummer vore bra som identitet för att skapa idempotemcy men det är inte alltid fallet.

Jag har sett applikationer där det senaste löpnumret kunde återställas till läget precis före kraschen, men några påföljande nummer hade trots allt hunnit skickas ut till ett annat system. Efter återställningen skickades alltså samma nummer ut igen vilket omöjliggjorde idempotemcy i det andra systemet och skapade felaktigheter hos mottagaren. Bättre är alltså någon slags annan unik transaktionsidentitet såsom ett så kallat guid/uuid eller andra typer av unika identiteter.

Annan datalagring

I de fall man inte använder en traditionell relationsdatabas utan någon annan lagringsprincip så blir man vanligen av med lyxen att ha en inbyggd

transaktionsjournal. Tidsglappet där man förlorar uppdateringsdata kan bli stort. Om det handlar om viktigt operativt data måste man då kanske tillföra en egenprogrammerad transaktionsjournal.

I andra sammanhang kan det räcka med att logga inkommande händelser och ha en överenskommelse med parter som man tar emot ifrån om att de lovar att spara gammalt data och att klara av återspelning. Skulle man använda en av de nya s.k. NoSQL-databaserna så har vissa av dem journalhållning, men inte alla – då gäller det alltså att se upp så att inte viktig data slarvas bort.

Molnlagring har sällan journal utan försöker ge dataskydd genom spegling. Men notera att spegling inte ger minsta skydd mot operatörsmissgrepp eller mot att applikationsbuggar eller plattformsfel raderar viktigt data!

Ett nära relaterat problem är att vanliga filsystem bara brukar ha backuptagning en gång på natten. Det innebär att en batchfil som var halvinläst i återställd databas kanske är försvunnen efter kraschen och då måste återsändas från ursprungssystemet.

Ett annat problem jag har råkat på är att en applikation genererade bilder som lagrades i filsystemet och pekades ut av en tabell i databasen. Men i och med en lyckad återställning av databasen hade den bara tappat några sekunders data, medan en mängd filer som pekades ut var borta eftersom de skapats efter backupen på natten. Det innebar ogiltiga filpekare varför applikationen inte ens gick att starta. Här behövdes alltså en förändrad undantagshantering så det åtminstone gick att få igång systemet.

Sammanfattning

En applikation som har hand om information man har höga krav på måste ha en återställningsrutin som inte tappar bort data efter en krasch. En sådan rutin kan behöva vara ganska komplex och bestå av både manuella och automatiska steg.

Observera **att transaktionsjournaler och indataloggar SKA lagras på ett helt annat disksystem än där det operativa datat lagras**. Alltså inte över huvud taget i samma SAN-disksystem.

Det finns buggar även i SAN (vet jag av egen erfarenhet) som inte får tillåtas ge problem för både databasen och för journalen. Dessutom finns det alltid risk att operatörsmissgrepp gör att SAN:et i sig får problem, liksom när mjukvaruuppdateringar i SAN:et införs eller när migreringar mellan olika SAN ska göras.

Slutligen: **återställning måste testas, testas, testas, testas...** inte bara när systemet införs, utan periodiskt under hela dess levnad. Kanske kan man använda billiga virtuella testservrar som noga efterliknar driftservrarna så att inte kostnaden för ett återställningstestsystem blir så stor – det vore rätt så nervöst att behöva testa återställning direkt i driftsystemet, även om det också kan behöva förekomma.

Här kan SAN vara nyttiga för att möjliggöra tillfälliga stora filytor där man kan testa återläggning. Kanske kan flexibla molntjänster också komma ifråga för de här testerna.

Sven-Håkan Olsson sysslar just nu med en läsplattetjänst för bolagsstyrelser och nämnder. I övrigt är han oberoende konsult som särskilt arbetar med att kombinera verksamhetsnytta med teknikhöjd. Han har en lång karriär bakom sig sedan 70-talet som it-konsult (applikationsarkitektur, systemdesign, programmering, reviewer, utredningar, kursledning). Sven-Håkan är medgrundare till KnowIT där han också var teknikchef 1990 - 2003. Han utsågs till en av "Sveriges topputvecklare" av Computer Sweden. Sven-Håkan håller regelbundet kurser åt Dataföreningen. Läs gärna mer på hans blogg definitivus.se samt på styrelsemöte.se.

Sven-Håkan Olsson