DIS - Skalerbarhed 1 - Publish-Subscribe vs. Message Queues

Publish-Subscribe

Publish-subcribe er en metode til at abonnere til at distribuere data

SIP
Session Initiation Protocol, der blandt andet bruges til beskedudveksling.

Pubsub

Eksempel Når en enhed abonnerer på en kanal, og når der sker noget i den kanal, så får alle abonnenter besked – dvs. alle enheder der har bedt om at få noget besked, får besked.

Publish-subscribe fungerer ofte via websockets eller long-polling.

Long polling
er en teknik til at få klienten til at spørge om noget og så lade vær med at svare med det samme. Long polling anvender HTTP, som er en protokol, som er baseret på request-response - "nu vil jeg gerne have noget, så nu skal du svare mig med noget".

Udfordringer

Publish-subscribe kræver ofte en vis mængde systemressourcer.

Skalering foregår typisk via en database, fx Redis, som ofte skal clusteres.

Heartbeating
kan udgøre en væsentlig mængde trafik og kræve flere ressourcer end selve beskederne, der bliver sendt frem og tilbage

Publish-subscribe er ofte implementeret via WebSockets, men mange firewalls tillader ikke protokollen

Heartbeating

centralt begreb når man snakker computere, når man snakker publish-subscribe (også alt muligt andet) - det går dybest set ud på, at én enhed (en computer fx) spørger en anden enhed (en anden computer), om computeren er der eller ej. Når man sender en besked til den anden enhed "Er du der?", og den anden enhed svarer: "Yes, jeg er der stadig" eller "Nej, jeg er der ikke"

Hvorfor gør man det? Det gør man fordi, når vi snakker om nu hvor man er online, så er man jo aldrig online hele tiden. Det handler om, at to enheder er aldrig forbundet med hinanden over nettet, det er en illusion. Kun det øjeblik hvor de kommunikerer med hinanden og sender data. I mellemtiden kunne den ene sagtens lave noget andet. Derfor anvendes heartbeating til at finde ud af om den anden vedkommende fortsat er der gennem en sådan beskedudveksling.

Heartbeating kan anvende flere ressourcer end beskeden der sendes - i kilobytes/bytes kan heartbeating godt fylde mere end en besked der indeholder A eller B (er der/ikke er der), der for kan det være **omkostningstungt** også.

Det er ikke kun websockets, men også HTTP-requests. Dermed er publish-subscribe et eksempel på noget, der kan hjælpe med at skalere. Når man sender HTTP-requests til en server, er der typisk meget overhead forbundet med det. HTTP request har en stor header målt i bytes (hvem er den, hvilket sprog den taler nogen gange, hvor den er fra osv.), sådan er det ikke med web sockets, som er mere socket baseret kommunikation. Så i stedet for at der er 1000 servere der via HTTP requests spørger livescore.com hvordan det står til en specifik kamp, så kan man bruge publish-subscribe i stedet for at spare på ressourcer, hvilket gør det mere skalerbart.
Dermed er publish-subscribe et eksempel på noget, der kan hjælpe med at skalere. Skalering er nøglen i de fleste distribuerede systemer.

Direkte / Indirekte kommunikation

Direkte kommunikation er når to enheder taler med hinanden (fx bluetooth og kabler)

Indirekte kommunikation er når computerne taler med hinanden gennem en server. (Server er tredjepart) (fx facebook - messenger beskeder)

Indirekte kommunikation foregår altid mellem mindst to parter, der taler via et mellemled

Hvorfor ikke bare kommunikere direkte?

Mange eksempler på at vi kommunikere direkte i hvedagen - fx når vi ringer til hinanden, det er svært at lægge over på en anden. Men et beskedsystem giver det god mening, at man sender noget videre til en server, som sendes videre til den anden part - her behøves man ikke være bundet til hinanden samtidig.

Afsender og modtager skal ikke være online samtidig. Derfor kommunikerer man ofte ikke direkte! Emails er et andet eksempel på indirekte kommunikation.

WebSockets

Websockets er en protokol (ligesom HTTP), hvor en server kan sende data til en klient uden at klienten har spurgt først. Så når vi sidder og chatter inde på messenger, så bruger vi websockets

Flooding

Man har en socket som abonnerer på en message, så tager den en funktion som parameter. Den broadcaster så til den samme kanal med username - det, der så sker, at alle der abonnerer på message kanal, får en besked, der bliver sendt ud. Når beskeder sendes ud til alle, så er der tale om flooding. Det kan være både godt og skidt. Hvis vi har livescore.com, så er det smart med flooding, da alle er derinde vil tænkes at have nyt om kampe der spilles. Når man sender en besked til én via messenger eksempelvis, så er det ikke flooding. I stedet for broadcaste til alle, så sender man specifikt til ét ID fremfor en gruppe.

Message Queues

Der er meget inden for IT der ikke er så komplekst, når man tænker det. Generelt set er det smart at kunne **parkere noget et sted**. Hvorfor er det smart med message queues? Fordi man kan parkere noget et sted, en buffer, inden det er klar til at blive brugt et specifikt sted.

Løs kobling

Når man har en beskedkø så skaber det løs kobling mellem applikation - løskobling er når to ting ikke afhænger så meget af hinanden, kan være måden måden man har implementeret noget kodeteknisk fx med interfaces. Servere, der er stærkt afhængig af hinanden (tætkoblet) eller ikke afhængig af hinanden (løskoblet)

Den kan sørge for at modtage beskeder og sende dem videre til videre til behandling og både en efter en og parallelt for bedre udnyttelse af ressourcer på tværs af netværk. Det skaber **asynkronitet**.

Atomicitet

Sikrer atomicitet Begreb der dækker over, at enten gør man noget fuldt og hændt ellers gør man det slet ikke. Man bruger det meget i databaser, man bruger det i transaktioner - man bruger det "atomart" - så ændrer man nogle data i nogle SQL kald, og man gør det kun, hvis alle de ændringer man foretager, lykkedes. Så hvis noget fejler, så ruller man det tilbage og ændrer det ikke. Meget udbredt inden for SQL

Sender - message queue - receiver

Når man snakker beskedkøer i IT-brabchen, det er noget der tilhører Enterprise-delen af vores fag. IBM laver det (mest kendt for det med websfear(?)), Microsoft laver det også. Når IBM kommer ud til SKAT og sælger dem en beskedkø, så er der et mønster der hedder, at man har en sender , der sender en besked, og vi har en receiver (modtager). Og når man så sender afsender en besked (vi vil ha fat i årsopgørelse), og man skal vente, fordi receiver ikke er klar til at modtage, så opstår der en beskedkø (vi har sendt en besked til en API der taler med en beskedkø), indtil receiver er klar til at modtage.

Hvorfor gør man så det?

- Når vi får vores foreskudsopgørelse, så opretter man disse beskedkøer, sådan så når alle vil ind og se opgørelsen, at systemet ikke bare gå ned. Sker også ved salg af iPhones ved nylancering fx.
- Det er en måde bevidst at forsinke systemet, så det fortsat virker. Helt konceptuelt har man noget kode, man modtager og sender afsted gennem.

Websphere, RabbitMQ, ØMQ, Resque

Publish-Subscribe vs. Message queues

Publish-Subscribe

- **Real-time**
- **Sporadisk**
- **Forbinder klienter**
- **Ingen ambitioner om konsistens**

Den ene er real time og sporadisk. Publish subscribe kan man vælge at abonnere noget og gå væk igen, så går man bare væk igen. Igen med LiveScore. Det er generelt set en service hvor man kan gå ind og se hvad scoren er i forskellige fodboldkampe. Og når vi går ind på livescore.com tilmelder vi deres publish-subscribe system, og når vi så lukker igen, så lukker vi bare igen. Og når vi så går ind på livescore.com igen, så går vi bare ind igen

Message queues

- **Kronologisk**
- **Struktureret**
- **Forbinder servere**
- **Konsistens afvikling**

Man lægger besked nummer 1,2,3,4,5 i kø - det er kronologisk og meget struktureret. Hvis alle går ind på SKAT.dk og alle går ind og beder om forskudsopgørelse, så bliver det lagt i kø hos SKAT – de har sådan nogle servere. Den tager besked efter besked efter besked kronologisk - det er meget struktureret. Derfor er beskedkøer typisk en måde at forbinde server på - en måde man bruger til at skalere på. Det sikrer konsistens afvikling

Noter

Når der er flere versioner af sandheden, som to eller flere computere skal blive enige om.

Det er et meget normalt problem i distribuerede systemer – de skal blive enige om noget, jf. distribueret system er der altid forsinkelse i. Det er der i alt hvad vi laver

Hvorfor skiftede Microsoft til indirekte kommunikation? Pga. kontrol og overvågning. NSA vil gerne have mulighed for at aflytte vores Skype samtaler. Kontrol ødelagde Skype. Indirekte kommunikation er derfor ikke altid godt

Skype er et eksempel på scaling-gone-wrong. Det handler om at bruge den rette teknologi i den rette sammenhæng

DIS handler overordnet set om konsistens og eventual **consistency** (**konsistens over tid**), **concurrency (noget der sker samtidigt), kausaliteter (noget sker i en rækkefølge)**, det handler om andre begreber, der er meget akademiske. Noget, der ikke er konsistens, er noget der typisk ikke har et forløb. Det vender vi tilbage til senere i den sidste forelæsning.