Problembeskrivelse informatik

Moderne software-udvikling handler i større grad om afvikling af kode i Cloud-miljøer end lokale maskiner og egne server-maskiner, som man traditionelt har fokuseret på. Dette fremhæver nogle andre problematikker.

Den første af disse problematikker vi vil arbejde med er måden hvorpå man betaler for Cloud-services. Betaling afregnes på baggrund af afviklingstid. Det vil sige man betaler på baggrund af hvor lang tid, dvs. hvor langsom ens kode er. Derfor er der et proportionalitetsforhold mellem afviklingstid og afregningspris. Dvs. det er i programmørens interesse i at ræssonere om og reducere afviklingstid.

Den anden problematik er i forhold til korrekthed af programmet man skriver. Dette problem, ligesom det første i hvis grad, er ikke specifikt til Cloud-miljøet, men det kan have nogle andre, muligvis forværrede konsekvenser i dette miljø. Derfor er der også har en interesse for programmørens, nemlig at ræssonere om og reducere mængden af fejl.

Det første problem kan løses eller forbedres af af forskellige performance-diagnostiske værktøjer. Disse giver programmøren et indblik i afviklingen af koden med fokus på afviklingstiden.

Et konkret eksempel på såddanne diagnostisk værktøj er en *flamegraph*. En flamegraph fortæller helt konkret med visuelle bjælker hvor meget af den totale afviklingstid bliver brugt i hver enkelt funktion. Dette kan programmøren bruge til at at se, hvilke funktioner, dvs. hvilke dele af koden de bør fokusere på.

Det andet problem relaterer til at kunne finde fejl i koden. Dette gøres eksempelvis med kodetest, som afvikler dele af koden og programmatisk tester resultatet.

Et konkret eksempel på diagnostisk værktøj i denne forstand, er *code coverage detection*. Dette værktøj fortæller programmøren konkret linje for linje, hvilket kode bliver kørt og hvilket kode, der ikke bliver kørt.

Dette kan programmøren bruge til at se, om *code tests* faktisk tester det relevante kode.

Problembeskrive teknikfag

Vores produkt tager form af et udviklingssystem bestående af et compilerværktøj og afviklingsmiljø, en programmør kan bruge til både at afvikle og diagnosticere programmer.

Flow'et fungerer sådan: En programmør skriver sit program i sproget, vi har specificeret. Programmøren bruger derefter vores compilerværktøj til at oversætte programmet i kildekode til *byte*-kode, som vores afviklingsmiljø kan afvikle. Programmøren eller en anden operatør bruger så afviklingsmiljøets værktøjer til at studere afviklingskarakteristika af programmet.

Compileren består af at række transformationer, der i den ene ende tager kildekode som input, udfører statisk checking og validerer programmet og producerer byte-kode som output. Compileren går igennem forskellige steps, såsom *parsing, symbol resolution, type checking* og *lowering.*

Afviklingsmiljøet er et program, der kan tage byte-kode som input, afvikle det og vise forskellige grafiske og tekstuelle diagnostikker og informationer om afvikling til programmøren eller en anden operatør. Det grafiske information vises gennem et web-interface.

De grafiske informationer består eksempelvis af *flamegraph* og *code coverage detection*. En flamegraph viser afviklingstiden i hvert funktion i koden. Code coverage detection viser hvor mange gange hver af alle linjerne i et program bliver afviklet.

Afviklingsmiljøet sætter alle værktøjer til rådighed for programmering, der er nødvendigt for programmøren, for at udvikle og afvikle brugbare programmer. Dette inkludere en måde hvorpå hukommelse og andre systemressourcer bliver håndteret i koden og i afviklingsmiljøet.