Mobila molntjänster (halvfärdig)

Kandidatavhandling i Datateknik

Axel Fogel

Handledare: Annamari Soini

Våren 2015

Innehållsförteckning

1. Inledning

Antalet mobila enheter i värden översteg antalet människor under år 2014. Idag finns det ungefär 7,4 miljarder mobila anslutningar med 3,7 miljarder unika abonnenter. Antalet mobila anslutningar ökar fem gånger snabbare än världens befolkning, med en årlig ökning på ungefär 6 % [1][2]. Även smarttelefonernas antal har ökat explosionsartat och man antar att det under 2015 kommer att finnas över 2 miljarder smarttelefonanvändare världen över. År 2017 kommer uppskattningsvis hälften av alla mobiltelefoner att utgöras av smarttelefoner och en tredjedel av världens befolkning kommer att äga en smarttelefon [3]. Även surfplattorna tar stora marknadsandelar. Trots att det inte är mer än fem år sedan Apple lanserade sin första surfplatta, uppskattar man att det år 2015 kommer att säljas fler surfplattor än persondatorer [4].

Den enorma ökningen av mobila enheter har lett till att även internetanvändningen skiftat mot det mobila. Idag är den mobila enheten medelamerikanens främsta verktyg för att komma åt internet. Sammanlagt utgör den mobila användningen 60 % av den totala internetanvändningen och av den mobila användningen sker över 80 % via mobilapplikationer [5].

Kravet på mobiltelefonen har under de tio senaste åren ändrat totalt. Från att ha varit en apparat med vilken man kan ringa och skicka korta meddelanden har den utvecklats till ett verktyg som användas till allt mer avancerade uppgifter. Idag används mobiltelefonen som kamera och musikspelare, för att nätsurfa och besöka sociala medier samt för e-post, kalender och övriga arbetsrelaterade uppgifter. Telefonen, och mobila enheter överlag har med andra ord tagit över många av de uppgifter som man traditionellt sätt använt persondatorer till. Detta har möjliggjorts av en mycket snabb utveckling av telefonen, som i många avseenden idag är lika kraftig som persondatorerna för tio år sedan [6].

Trots den otroliga takt med vilken mobila enheter har utvecklats har inte alla tekniska delområden hängt med i utvecklingen. Den mest begränsande faktorn, vad gäller mobila enheters utvecklande, är batterierna, vilka inte har utvecklats i

samma takt som den övriga tekniken. Eftersom enheterna skall vara så kompakta som möjligt kan man inte heller utrusta dem med stora batterier. Detta betyder istället att processorerna måste vara så energieffektiva som möjligt, vilket i sin tur leder till att prestandan lider [7] (kolla källa ännu). Trots energieffektiva enheter är de flesta smarttelefoners batteritid föga imponerande och över ett dygns batteritid är snarare ett undantag än en regel.

Mobila molntjänster (*Mobile Cloud Computing*) är ett framlagt förslag på hur man skulle kunna övervinna vissa av de begränsningar som idag finns på mobila enheter. Kort beskrivning av vad mobila molntjänster är och en beskrivning av vad jag kommer att gå igenom i avhandlingen.

2. Molntjänster

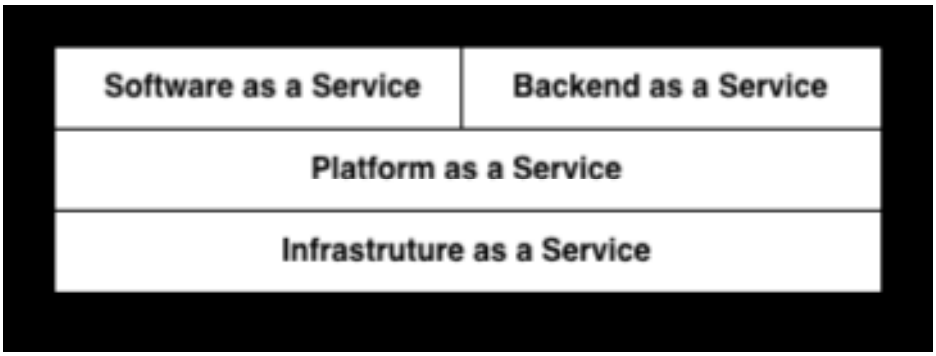
The National Institute of Standards and Technology (NIST) har definierat molntjänster som följande: ”Molntjänster är en modell för att möjliggöra ubikvitär, bekväm, på begäran tillgänglig nätanslutning till en delad pool av konfigurerbara dataresurser (t.ex. nätverk, servrar, lagringsutrymme, applikationer och tjänster) som skall kunna tilldelas och frigöras snabbt och med så lite administration som möjligt” (fri översättning från engelska) [8].

Det finns även många andra beskrivningar av molntjänster, och begreppet är en aning diffust. De tre utmärkande egenskaperna för molntjänster är enligt Ashraf et al. dock:

- Datorresurser kan köpas på begäran från ett till synes oändligt lager.
- Utgifterna för datorresurser ändras från att vara stora investeringskostnader till att vara rörelsekostnader. Detta eliminerar risken för att över- eller underskatta resursmängden som behövs.
- Kapaciteten kan snabbt skalas upp och ner, och betalning sker baserat på användning [9].

Fördelarna med molntjänster jämfört med den traditionella IT-modellen, kan i många fall vara betydande. Molntjänster möjliggör snabb driftsättning av nya applikationer, funktionaliteter och tjänster vilket gör IT-omgivningen mer mottaglig för förändringar i kraven som ställs på den. Riskerna med att under- eller överskatta resursbehoven elimineras också eftersom resurserna snabbt kan ökas och minskas vid behov, och kostnaderna baseras på använd resursmängd. Detta är speciellt användbart för företag vars resursbehov kraftigt varierar. Användningen av molntjänster eliminerar företags behov att själva uppehålla dyr IT-infrastruktur i och med att det är molnleverantören som tar hand om allt dylikt underhåll. För speciellt små företag kan molntjänster även leda till ökad säkerhet eftersom de inte nödvändigtvis har resurserna eller kunnandet för att uppehålla önskad säkerhetsnivå [10].

Molntjänster medför också ett flertal risker och nackdelar. I och med att processorkapaciteten och alla data är utlokaliserade, är ett företag som använder sig av molntjänster helt beroende av nätverksförbindelsen till leverantören samt leverantörens tillgänglighet. Likaså medför utlokaliseringen av data vissa säkerhetsrisker i och med att företagen inte själva kan påverka eller ens nödvändigtvis granska säkerheten med vilken data lagras i molnet. Ett annat problem är avsaknaden av driftskompatibiliteten mellan olika leverantörer. Kunden kan i vanliga fall inte enkelt flytta data och program från en leverantör till en annan. Detta betyder att företag blir låsta till en leverantör vilket kan vara ett problem i fall där en viss tjänst inte erbjuds leverantören eller då leverantören går i konkurs [11].



Figur 1 Servicehierarkin [19]

Molntjänster brukar vanligtvis delas upp i tre olika kategorier: Infrastructure as a Service, Platform as a Service och Software as a Service. Förutom dessa finns det även en fjärde kategori, Backend as a Service, som inte är lika etablerad som de övriga, men som är relevant för mobila enheter. Figur 1 visar servicehierarkin som ofta brukar framställas i nivåer eftersom de olika tjänsterna bygger på varandra [19]. Gränserna mellan dessa är dock inte helt klara utan tjänster kan ha karaktärsdrag från flera olika kategorier [12]. I det följande kommer jag att gå igenom de olika kategorierna och belysa skillnaderna mellan dem samt nämna olika leverantörer av de olika tjänsterna.

2.1 Software as a Service

I Software as a Service (SaaS) erbjuds kunden applikationer eller fullständiga mjukvarusystem, som finns på leverantörens servrar och är åtkomliga för kunden via Internet. Tjänsterna kan antingen vara gratis, i vilket fall inkomster genereras av till exempel reklam, eller avgiftsbelagda, antingen på basis av användning eller genom en fast prenumerationsavgift. Kunden behöver således inte betala för hårdvaruinfrastrukturen eller för programlicenser. Software as a Service är ett bra alternativ då mjukvaran inte behövs hela tiden, utan behovet varierar kraftigt. Exempel på sådan mjukvara är program för bokföring eller beskattning. Även mjukvara som endast behövs för en kortare tid, till exempel ett projekt, eller mjukvara som måste vara åtkomligt via nätet eller mobilt, kan löna sig att köpa som Software as a Service. Däremot är SaaS inte det bästa alternativet då bland annat snabba realtidsberäkningar måste utföras eller då det redan finns en lokal lösning som uppfyller företagets behov [12]. Kända exempel på leverantörer av Software as a Service är Salesforce och Facebook [9].

2.2 Platform as a Service

I Platform as a Service (PaaS) är det, som namnet säger, en plattform som levereras över nätet. PaaS används ofta för utveckling av webbapplikationer eftersom man med hjälp av det snabbt kan utveckla applikationer utan att behöva bry sig om den underliggande infrastrukturen och mjukvaran. Dessutom kan applikationerna vanligtvis köras som en del av tjänsten [9]. Oftast består PaaS av tjänster för att utveckla, testa, agera värd för och underhålla applikationer i en enhetlig utvecklingsmiljö, som också tillåter förändringar i belastningen. Även verktyg för projektplanering och kommunikation kan ingå för att underlätta samarbetet mellan olika utvecklingsteam. Detta betyder att Platform as a Service är ett bra alternativ i projekt där det finns flera utvecklare eller där många utomstående parter måste kunna samverka. PaaS passar också för rörlig systemutveckling samt för projekt där man vill automatisera testningen och driftsättningen. Däremot är PaaS inte det bästa alternativet ifall man vill ha större frihet att välja programmeringsspråk och verktyg eller ifall man har behov av att modifiera underliggande hårdvara. Bland de kändaste leverantörerna av Platform as a Service finns Google App Engine och Microsoft Azure Services [12].

2.3 Infrastructure as a Service

Den stora skillnaden mellan Infrastructure as a Service och Platform as a Service är att medan PaaS vanligtvis kräver en viss applikationsmodell samt tillhörande mjukvara och bibliotek så har man i IaaS en större frihet att välja systemen man vill använda [9]. I Infrastructure as a Service köper man IT-infrastruktur, t.ex. servrar, lagringsutrymme och operativsystem som en service på distans i stället för att själv bygga upp egna datacenter. Således kan företag undvika att själva stå för stora anskaffningsinvesteringar och dyra underhållskostnader. IaaS är också mycket skalbar, så företag riskerar inte att missbedöma resursbehovet. Således är Infrastructure as a Service lämpligt i fall där resursbehovet är mycket fluktuerande eller i fall där resursbehovet växer i en sådan takt att kunden har svårt att anpassa hårdvaran till behovet. IaaS lämpar sig också för nya företag, som saknar resurser att investera i hård- och mjukvara. Slutligen är IaaS också ett bra alternativ i fall där behovet av infrastruktur endast är temporärt. Emedan IaaS har stora fördelar när det kommer till skalbarhet, kan dess prestanda inte alltid uppnå den på ett lokalt system. Således är IaaS inte den bästa lösningen i fall där mycket hög prestanda krävs. Det kändaste systemet av typen IaaS är Amazon EC2 [12].

2.4 Backend as a Service

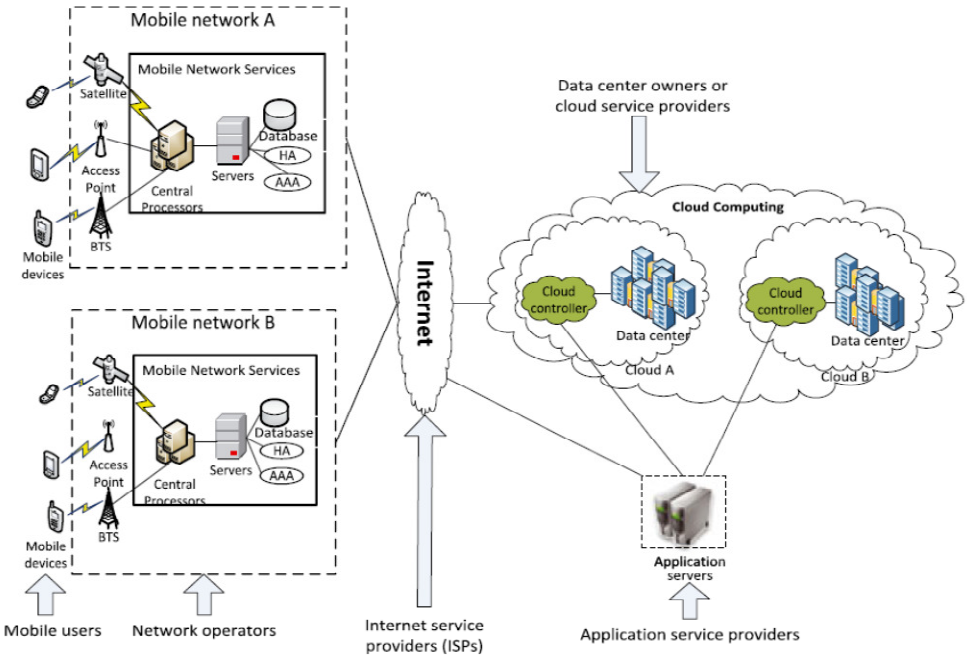
Backend as a Service är en ny typ av molntjänst som utvecklats under de senaste åren och är ett sätt för utvecklare av webb- och mobilapplikationer att koppla sina applikationer till molnresurser, såsom lagring och beräkningskraft. Marknaden för Backend as a Service kommer uppskattningsvis att växa från 216,5 miljoner dollar år 2012 till 7,7 miljarder dollar år 2017. Emedan det finns likheter mellan PaaS och BaaS, är den senare direkt ämnad för mobila applikationer och för att underlätta alla delmomenten i utvecklingsprocessen. Målet är att skapa en infrastruktur som tar hand om skalningen, prestandan och säkerheten såväl som data- och nätverksproblem ofta associerade med mobila applikationer. BaaS erbjuder också användarhantering, push-notifikationer, integration av sociala medier, virtuell handel, geolokalisering och övriga egenskaper som krävs för utveckling och användning av applikationer.

Till de ledande BaaS-leverantörerna hör Appcelerator och Appery.io men även de stora molnleverantörerna som Amazon Web Services och Salesforce har börjat

modifiera sina produkter så att de bättre uppfyller de krav som ställs av utvecklare av mobila applikationer [13].

3. Mobila Molntjänster

Trots att termen mobila molntjänster har existerat nästan lika länge som konceptet molntjänster har begreppet inte entydigt definierats. I sin enkelhet är mobila molntjänster en infrastruktur där datalagring och databehandling kan ske utanför den mobila enheten. Det mobila molnet kan dock definieras på olika sätt. Vanligtvis syftar termen på en struktur där databehandling och -lagring flyttas från den mobila enheten till en betydligt kraftigare datastruktur i molnet. Således kan man kringgå de begränsningar som ställs av enhetens hårdvara och potentiellt få en lösning som är mer energieffektiv och förlänger enhetens batteritid eller en som har högre prestanda än enheten. Figur 2 visar en typisk arkitektur för mobila molntjänster, dock kan detaljerna variera. Här är den mobila enheten via ett mobilnät kopplad till resursrika servrar [14]. Man kan också med det mobila molnet syfta på en lösning där de mobila enheterna själva är delar av ett mobilt moln. De enskilda enheternas resurser utnyttjas tillsammans med stationära enheter, om sådana finns tillgängliga, för att bygga upp ett icke-hierarkiskt nät. Ett tredje alternativ är en lösning som använder sig av så kallade *cloudlets*. I denna modell flyttas arbetsbördan till lokala *cloudlets*, små datorer med nätanslutning till molnet. Dessa *cloudlets* skulle finnas på allmänna platser såsom kaféer och fungerar som mellanhänder till molnet, vilket kunde minska latensen och lösa bandbreddsproblem [15].



Figur 2 Arkitektur för mobila molntjänster[14]

Jag kommer i detta kapitel först att ta upp de problem som potentiellt skulle kunna lösas med hjälp av mobila molntjänster. Jag kommer också att gå igenom de begränsningar och utmaningar som ännu finns. Efter det kommer jag att gå närmare in på beräkningsavlastning (*Computation Offloading*) som är ett av de främsta koncepten inom mobila molntjänster.

3.1 Fördelar med mobila molntjänster

Den mobila kommunikationen och de mobila enheterna står i dagens läge framför ett antal utmaningar. Till de mobila enheternas främsta problem hör idag begränsad batteritid, begränsade lagrings- och beräkningsresurser samt bristande pålitlighet. Mobila molntjänster har potential att förbättra enheterna på alla dessa punkter. Dessutom ärver mobila molntjänster också alla de egenskaper som vanliga molntjänster har, det vill säga dynamisk tilldelning av resurser och skalbarhet med mera.

De lösningar som presenterats för att lösa enheternas batteriproblem har ofta krävt förändringar i enheternas strukturer eller hårdvara vilket ofta är dyrt eller rentav ogenomförbart. Mobila molntjänster och framför allt beräkningsavlastning är ett koncept som har potential att minska enheternas energiförbrukning avsevärt. Beräkningsavlastning betyder att komplexa arbetsdryga beräkningar flyttas från den mobila enhetens begränsade hårdvara till stora resursrika servrar i molnet. Denna avlastning minskar processoranvändningen i den mobila enheten vilket förlänger batteritiden. Experiment har visat att batteritiden i vissa fall drastiskt kan öka med hjälp av denna teknik. För stora matrisberäkningar kan enhetens energibesparing vara upp till 41 %. Besparingen i vissa spel kan vara upp till 45 % [14]. Förutom att förlänga batteritiden kan beräkningsavlastning i vissa fall till och med förkorta den totala beräkningstiden för tunga beräkningar. För till exempel ansiktsigenkänning har man i experiment lyckats minska latensen från 19 sekunder till mindre än 2 sekunder genom att flytta beräkningarna till ett näraliggande moln [16]. I kapitel 3.3 kommer jag mer ingående gå igenom olika förslag på hur beräkningsavlastning kunde genomföras.

Förutom processorkraft har mobila enheter också ofta kraftigt begränsad lagringskapacitet. I dagen läge finns det flera tjänster för att lagra och komma åt filer i molnet. Förutom enkel lagring kan det i tjänsterna även ingå funktioner för att till exempel bearbeta bilder, vilket vanligtvis är energikrävande på enheterna. Överflyttningen av data och beräkningar till molnet leder även till en högre grad av pålitlighet eftersom data och processer finns säkerhetskopierade på flera ställen. Dessutom kan molnen erbjuda säkerhetstjänster på distans för mobila enheter. Exempel på sådana tjänster är virusskanning, spårning av skadliga program samt auktorisering. Alla dessa tjänster kan köras på distans, på datorer med mycket högre resurser, vilket förbättrar prestanda och batteritid [14].

3.2 Utmaningar

På grund av kombinationen av två olika teknikområden står mobila molntjänster inför många tekniska utmaningar. Förutom de problem som även gäller vanliga molntjänster, det vill säga kravet på tillgänglighet, säkerhet, och kompabilitet

mellan olika leverantörer är det främst mobiliteten som ställer krav på systemen. En stor utmaning är bandbredden eftersom resurserna i trådlösa nät är betydligt mer begränsade än i traditionella fasta nät [14]. Utvecklingen av fjärde och femte generationens mobiltelefoni kommer dock förhoppningsvis att förbättra bandbredden för mobila användare. Även femtocellnätverk är en lovande teknologi för att överkomma begränsningarna i mobila nät. Utöver dessa nya teknologier har det även föreslagits olika system där resurserna delas mellan användarna som sedan hjälper varandra att ladda ner data som skulle vara för stort för att ladda ner på egen hand. Dessa system är dock något begränsade eftersom de bygger på att användare vill ladda ner samma sak, eller är villiga att hjälpa varandra [14].

En annan utmaning som speciellt gäller mobila molntjänster är servicetillgängligheten. Riskerna för trafikstockningar, nätverksfel och signalproblem är större än för molntjänster via fasta nät. Behovet för kvalitetsgaranti är dock lika stort för mobila som icke-mobila molntjänster. Forskning som gjorts i dessa problem förslår lösningar där antingen användaren eller nätverksnoderna mer effektivt försöker hitta nya rutter till molnet ifall kopplingen bryts [14]. Ytterligare ett problem som utvecklarna av mobila molntjänster är tvungna att ta ställning till är heterogeniteten i trådlösa nätverk. Mobila enheter skall ha möjlighet att koppla till molnet via flera olika mobila teknologier med olika gränssnitt, till exempel LTE Advanced, CDMA2000 eller WLAN. Oberoende av teknologi skall de krav som ställts på mobila molntjänster uppfyllas. Ett förslag på en lösning är användningen av IRNA (*Intelligent Radio Network Access*) för att sköta om heterogeniteten av mobila nät [14].

3.3 Beräkningsavlastning

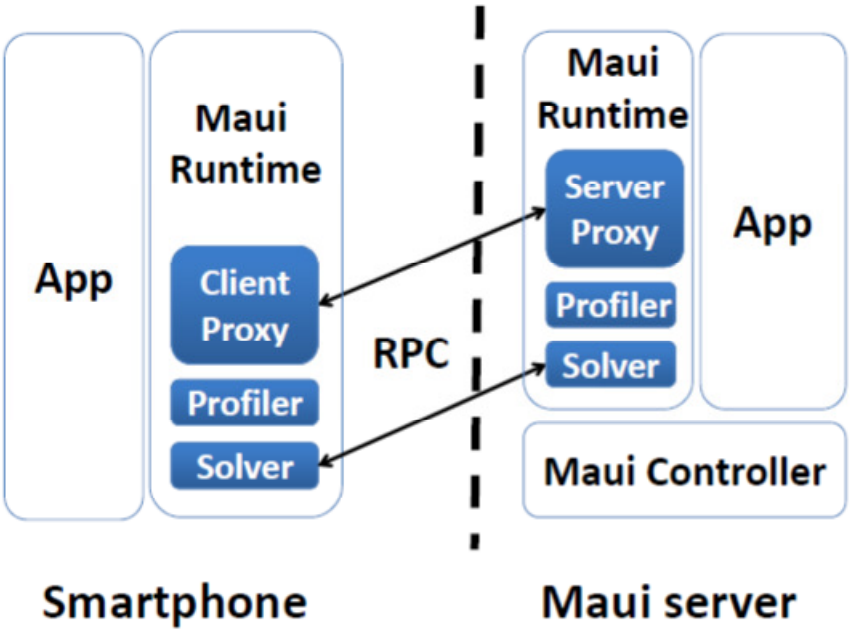
Ett delområde inom mobila molntjänster är beräkningsavlastning (*Computation Offloading*), vilket är en teknologi som försöker lösa mobila enheters resursproblem genom att migrera en del av beräkningarna till molnet. Vid avlastning flyttas kontrolldata och tillstånd till en server som fortsätter körningen av en viss del av programmet och sedan returnerar resultatet. Baserat på ett set kriterier bestäms huruvida avlastningen är fördelaktigt och avlastning sker endast

i de fall som det lönar sig. Eftersom kvaliteten på mobila nät varierar måste avlastningsbeslut konstant omprövas. Avlastning måste också kunna göras via flera olika mobila gränssnitt och dessa borde användas på ett sätt som optimerar avlastningen. Programmerarens roll varierar beroende på avlastningsmetod. I vissa metoder spelar programmeraren en stor roll i besluten om vilka delar av programmet som skall avlastas. I andra metoder däremot, behöver programmeraren endast lite eller inte alls ta ställning till avlastningen [17].

Jag kommer till följande att redogöra för två olika beräkningsavlastningsmetoder, MAUI [16] och CloneCloud [18], presenterade 2010 respektive 2011. Först kommer jag att presentera de båda metoderna var för sig och beskriva deras grundidéer. Metoderna har många likheter sinsemellan så jag kommer därefter att jämföra metoderna med varandra och jämföra experimentella resultat.

3.3.1 MAUI

MAUI är ett system som genom avlastning av kod maximerar den potentiella energibesparingen utan att ställa höga krav på programmeraren. Många tidigare implementeringar av fjärrexekvering (*Remote Execution*) valde vanligtvis ett av två tillvägagångssätt för att avlasta kod. Det första sättet avlastade hela processer eller rentav hela program till molnet. Detta ställer inte så höga krav på programmeraren men lösningen är inte heller optimerad för varje enskilt program. I den andra lösningen var det programmeraren som bestämde vilka delar av programmet som skulle avlastas och på vilket sätt avlastningen skulle bero på nätverkets karaktär och prestanda. Denna lösning är mer optimerad för enskilda program men kräver god kunskap av programmeraren. MAUI kombinerar dessa två tillvägagångssätt. Programmeraren markerar de metoder i programmet som kan avlastas. Varje gång en metod anropas använder MAUI sitt optimeringsramverk för att bestämma huruvida metoden skall avlastas eller köras lokalt. Optimeringen sker baserat på data som MAUI Profiler samlat från enheten och nätverket. Systemets främsta syfte är att minska på enhetens energiförbrukning, men även prestandan förbättras i vissa fall [16].



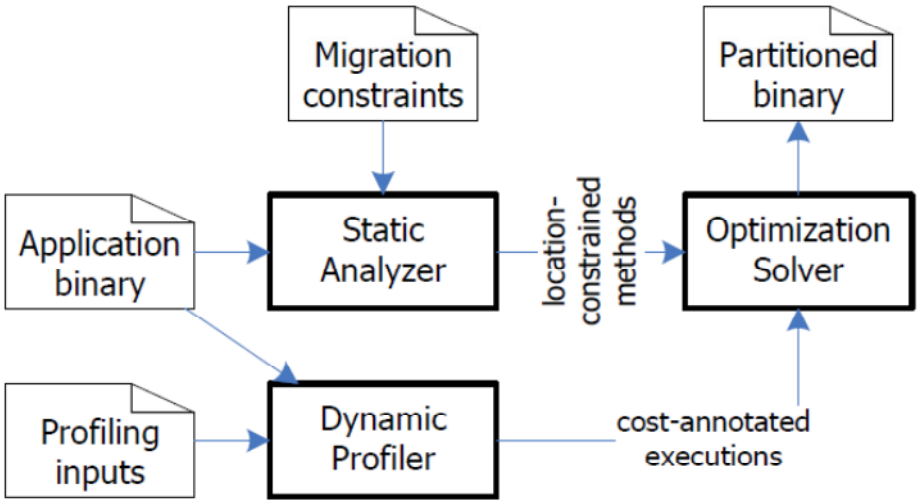
Figur 3 MAUI:s arkitektur [16]

MAUI skapar två versioner av en mobil applikation, en som körs lokalt och en som körs i molnet. MAUI använder sig av Microsoft .NET Common Language Runtime vilket betyder att skillnader i arkitekturen mellan den mobila enheten och servern kan ignoreras. Figur 3 visar MAUI:s arkitektur. Serversidan består av fyra olika delar. Server Proxy tar hand om kontrollen och dataöverföringen för metoder som avlastas. Profiler samlar data om programmets energianvändning och nätverksförbindelsen. Solver bestämmer baserat på profilers data, när det lönar sig att avlasta beräkningar. Slutligen finns Controller som tar hand om autentisering och resursallokering till inkommande förfrågningar. På den mobila enheten finns tre olika delar. Profiler och Client Proxy har ungefär samma uppgift som motsvarande delar på serversidan. Utöver dessa finns också ett gränssnitt till Solver, som körs i molnet för att spara energi [16].

I kapitel 3.3.3 till 3.3.7 kommer jag mer ingående gå igenom uppdelningen av programmet, profileringen av förhållandena samt lösningen av avlastningsproblemet samt jämföra dessa delar med CloneClouds motsvarighet.

3.3.2 CloneCloud

CloneCloud påminner i många avseenden om MAUI. CloneCloud löser, liksom MAUI, avlastningsproblemet med en optimerare som använder data samlat av en profilerare. Däremot delar CloneCloud utan programmerarens hjälp upp programmet i delar som kan avlastas och delar som inte kan det. Detta betyder att alla applikationer, utan manuell modifiering, kan avlastas till molnet förutsatt att de verkar på en virtuell maskin som stöds CloneCloud. Systemet jobbar på trådnivå, det vill säga enskilda trådar kan avlastas medan övriga trådar fortsätter att exekveras på den mobila enheten. När tråden kört klart i molnet returneras den tillsammans med resultatet och sammansluts med den ursprungliga processen. Baserat på data från Dynamic Profiler (nätverksinformation, processorhastighet och energikonsumtion) kan avlastningen optimeras för energibesparing eller exekveringshastighet [18]. I kapitel 3.3.3 till 3.3.7 kommer jag mer ingående att gå igenom dessa processer och jämföra dem med motsvarande processer i MAUI.



Figur 4 CloneCloud Partition Analyzer

Figur 4 visar Partition Analyzer som finns i molnet och bestämmer vilka metoder som skall exekveras på enheten och vilka som skall exekveras i molnet. Partitionsmekanismen består av tre olika delar. Static Analyzer analyserar koden och bestämmer vilka delar som är tillåtna att migrera och vilka som måste köras på enheten. Dynamic Profiler analyserar kostnaden för att köra applikationen

under olika förhållanden. Kostnadsmåtten som används i prototypen är exekveringstid och energiförbrukningen i den mobila enheten. Slutligen använder Optimization Solver data från Static Analyzer och Dynamic Profiler för att välja den bästa exekveringsmetoden. Detta resultat används då programmet körs för att välja vilka metoder som skall migreras till molnet och vilka som skall exekveras lokalt [18].

3.3.3 Partitionering

Den stora skillnaden mellan partitioneringen i de två olika systemen är att medan partitioneringen i CloneCloud helt sköts utan programmerarens inverkan måste programmeraren i MAUI markera de metoder som får migreras. I båda fallen är det dock systemet som bestämmer huruvida en metod i praktiken migreras eller körs lokalt på enheten. Båda systemen tillåter också endast att hela metoder avlastas [16][18].

I MAUI är det programmeraren som markerar de metoder som systemet skall överväga att migrera och gör det genom att markera dessa metoder *remoteable*. Programmeraren tar dock inte ställning till huruvida det lönar sig att avlasta metoderna, utan systemet tar hand om det. Metoder skall inte markeras *remoteable* i tre fall. För det första skall metoder som implementerar applikationens användargränssnitt alltid köras på den lokala enheten. Även sådana metoder som använder sig av enhetens hårdvara, till exempel kamera eller sensorer, skall köras lokalt. Slutligen får sådan kod inte migreras som samverkar med externa komponenter där slutresultatet påverkas ifall koden måste exekveras på nytt. Ett exempel på det sistnämnda är metoder som via en nätverksanslutning utför e-handelstransaktioner [16].

Kriterierna för när kod får avlastas i CloneCloud är till stor del de samma som för MAUI. Liksom i MAUI får endast hela metoder avlastas. Dessutom kan endast applikationsmetoder avlastas, inte enhetens kärnmetoder. Likaså får endast metoder ur applikationer som kör på den virtuella maskinen migreras. Förutom ovan nämnda krav finns det tre begränsningar som Static Analyzer måste ta

ställning till. För det första måste metoder som använder sig av enhetens hårdvara köras på den mobila enheten. För det andra måste metoder som använder tillstånd utanför den virtuella maskinen köras på den maskin där tillstånden finns, eftersom dessa tillstånd inte kan migreras. Slutligen får det inte finnas nästlade migrationer, det vill säga en migrationspunkt i programmet måste efterföljas av en återintegreringspunkt [18].

3.3.4 Profilering

Systemens profilerares uppgift är att samla information som sedan används för att göra avlastningsbeslut. Partitioneringen tillsammans med profilerarnas resultat används av optimerare för att bestämma ifall en metod skall avlastas eller köras lokalt [16] [18].

I MAUI görs avlastningsbeslut baserat på tre olika faktorer: Den mobila enhetens energikonsumtion, programmets egenskaper och nätverkets egenskaper. MAUI Profiler undersöker enhetens egenskaper endast en gång då programmet startar medan programmets och nätverkets egenskaper undersöks regelbundet eftersom dessa ofta ändras under exekveringens gång[16]. I CloneCloud är det Dynamic Profiler som samlar in data för avlastningsbeslut. Även CloneCloud använder sig av tre faktorer: Metodens exekveringstid, enhetens energianvändning samt nätverkets egenskaper [18].

Smarttelefoner erbjuder inte detaljerad information om energiförbrukningen för en applikation. Detta har i MAUI lösts genom att, baserat på experimentella resultat, bygga en modell för energikonsumtionen i den mobila enhet som använts i experimentet [16]. Energikonsumtionen i CloneCloud beräknas utifrån tre parametrar: Processoraktivitet, skärmtillstånd och nätverkstillstånd. För alla parametrar gäller att de antingen är påslagna eller avslagna. Då en metod körs på den mobila enheten antas att processorn kör, skärmen är påslagen och nätverket är i vilotillstånd. Då en metod kör i molnet antas att processorn är i vilotillstånd, skärmen är påslagen och nätverket är i vilotillstånd. Slutligen då en process håller

på att avlastas antas att processorn kör, skärmen är påslagen och nätverket är aktiverat [18].

Programprofilering görs på olika sätt i MAUI och CloneCloud. I CloneCloud körs applikationen flera gånger med slumpmässiga indata. Varje exekvering görs både på den mobila enheten och på servern. För varje exekvering skapar Dynamic Profiler två profilträd, ett för enheten och en för servern. Noderna i träden representerar metodanrop och har som värde kostnaden för att köra dem, det vill säga exekveringstiden. Varje kant har som värde storleken på tillståndet som skulle behöva migreras. Resultatet av profileringen är ett set av exekveringar med olika indata och för varje exekvering finns det två profilträd [18]. MAUI Profiler mäter för varje metod i programmet exekveringstiden samt antalet klockcykler som krävs. Dessutom mäts storleken på det data som måste överföras för att metoden skall kunna köras i molnet. Liksom i CloneCloud kan resultatet beskrivas med en graf där noderna representerar metoder och kanterna representerar överföring av tillstånd. I stället för att alltid flytta över hela tillstånd, migrerar MAUI endast de data som ändrats sedan den senaste avlastningen. En uppskattning över hur mycket energi som krävs då en metod körs på enheten görs baserat på antalet klockcykler och exekveringstiden. Anrop till metoder kan skilja sig från gång till gång vilket betyder att också olika mängd energi används. MAUI använder data från tidigare körningar för sina uppskattningar, vilket enligt författarna ger ett bra resultat [16].

MAUI profilerar nätverksförbindelsen genom att skicka ut 10KB data till servern och mäta latens och bandbredd. Varje gång MAUI migrerar en metod mäts dessa på nytt och ett medeltal räknas ut [16]. I artikeln om CloneCloud beskrivs det inte hur systemet profilerar nätverksförbindelsen, dock tas förbindelsen i beaktande när en partitionering görs [18].

3.3.5 Optimering

MAUI Solvers uppgift är att lösa ett optimeringsproblem som bestämmer vilka metoder som skall köras lokalt och vilka som skall migreras. Målet är att hitta en

lösning som minimerar enhetens energiförbrukning. Som invariabler används data som MAUI profiler har samlat. Beslutet om vilka metoder som skall migreras är svårt eftersom lösningen måste vara optimerad för programmet som helhet, inte bara för en viss metod. Medan programmet exekverar körs Solver regelbundet på nytt, delvis för att kunna anpassa sig till förändrade omständigheter men också för att lära sig om programmets beteende.

Programmets exekvering kan beskrivas som en graf där noderna representerar metoder och kanterna representerar överföring av tillstånd. MAUI Solver löser linjärprogrammeringsproblemet så att den förbrukade energin minimeras. Systemet har dock en standardbegränsning som kräver att latensen för hela körningen inte får vara mer än 5 % högre än den skulle vara ifall hela programmet kördes lokalt. Denna begränsning kan modifieras av programmeraren för att bättre lämpa sig för ett visst program [16].

Optimeringsproblemet i CloneCloud löses av Optimaization Solver, vars uppgift det är att bestämma vilka metoder som skall migreras och vilka som skall köras på enheten. Optimization Solver använder de träd som skapats av Dynamic Profiler. Intuitivt kan man se optimeringsproblemet som att byta ut noder i trädet för den lokala exekveringen med noder ur träder för fjärrexekveringen. Resultatet skall minimera trädets totala kostnad. Alla körningar som Dynamic Profiler gjort ses vara lika sannolika, men detta kan modifieras för att bättre passa ett specifikt scenario. Då ett beslut har gjorts om huruvida avlastning skall ske eller inte, kommer alla anrop till metoden att följa detta beslut [18].

3.3.6 Avlastning

Tack vare att alla CLR-applikationer kompileras till CIL-instruktioner behöver MAUI inte bry sig om arkitekturskillnader mellan den mobila enheten och servern. För att kunna avlasta en applikation måste serven ha en kopia av CIL-instruktionerna. Dessa kan antingen kopieras rakt från enheten eller så kan enheten skicka instruktionernas signatur, varefter servern kan ladda ner dem från en molntjänst. MAUI modifierar de instruktioner som har markerats *remoteable*

så att en extra invariabel och ett extra returvärde läggs till. Dessa är till för att flytta över programtillstånd. Då en metod skall avlastas måste tillstånd som används av metoden serialiseras. XML-baserad serialisering används och det som serialiseras är metodparametrar, objektets variabler, statiska klasser och statiska variabler. För att minimera mängden data som migreras överförs endast det som ändrats sedan den senaste avlastningen. Vid exekvering skapar MAUI två proxys, en på enheten och en på servern. Dessa verkställer de beslut som görs av MAUI Solver, och tar hand om kontrollen och dataöverföringen som krävs. Felhantering sköts med en enkel timeout-mekanism. Ifall enheten tappar kontakten med servern medan en metod exekveras på servern, flyttas kontrollen tillbaka till enheten. Enheten kan sedan välja att antingen köra metoden lokalt eller försöka hitta en annan MAUI-server och sedan migrera metoden till den servern [16].

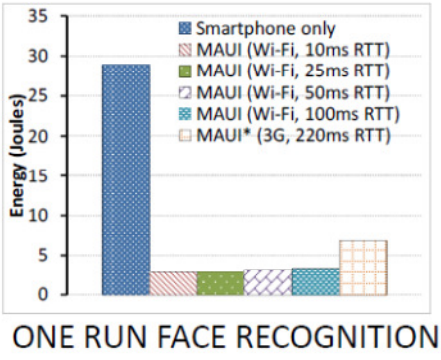
CloneCloud har tre komponenter som sköter om migrering så en applikation körs: Thread Migrator som hjälper en process att packa och packa upp trådar som skall migreras respektive reintegrera, Node Manager som sköter om kommunikation och en enkel databas som bestämmer vilken av de färdiga partitionerna som skall användas. När en användare försöker starta en partitionerad applikation undersöks rådande förhållanden. Baserat på dessa väljs en av de färdiga partitionerna och i koden läggs in migrerings- och reintegreringspunkter. Då en process når en migreringspunkt samlar Thread Migrator alla tillstånd som måste skickas och överlåter dem åt Node Manager. Övriga trådar fortsätter att köra förutsatt att de inte behöver information från metoden som avlastats. På servern placeras all information som anlänt i minnet och processen fortsätter. Då en reintegreringspunkt nås flyttas tillstånden tillbaka till enheten och sammanfogas där till den ursprungliga processen [18].

3.3.7 Försök och resultat

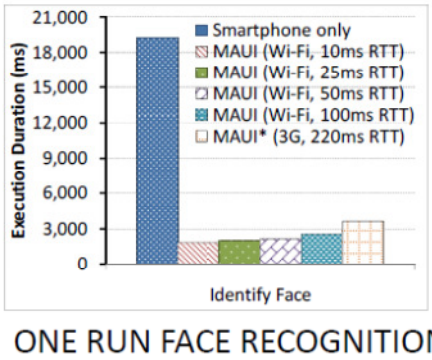
För att undersöka systemens funktionalitet har experiment utförts på bägge system. MAUI testas genom att undersöka prestanda- och energiförbrukningsskillnader mellan att köra fyra olika applikationer med och utan avlastning. Applikationerna som testas är en ansiktsigenkänningsapplikation, ett interaktivt videospel, ett schackspel och en röstbaserad översättare [16]. CloneCloud å andra sidan testas genom att undersöka skillnaderna i tre olika

applikationer: en virusskanner, en ansiktsigenkänningsapplikation och en applikation för riktad marknadsföring [18]. Eftersom båda systemen har testats med ansiktsigenkänningsapplikationer kommer jag att fokusera på dessa och jämföra resultaten från de båda experimenten.

Hårdvaran som används i de båda experimenten är prestandamässigt jämförbara med varandra. MAUI använder som mobil enhet en HTC Fuze med Windows Mobile 6.5 och en server med en 3.0 GHz processor med två kärnor. CloneCloud använder en HTC G1 med Android 1.5. Processorn i HTC G1 är den samma som i HTC Fuze. Även CloneClouds server har en 3.0 GHz processor, dock meddelas inte huruvida den har en eller två kärnor. CloneCloud använder sig i experimentet av en Wi-Fi-förbindelse med en bandbredd på 6,6Mbit/s och en latens på 69ms samt en 3G-förbindelse med en bandbredd på 0,4Mbit/s och en latens på 680ms. MAUI testas med fyra olika Wi-Fi-förbindelser med latenser mellan 10ms och 100ms samt med en 3G-förbindelse med en latens på 220ms. Nätverksprestandan är således lite olika i de båda försöken men båda uppvisar trots det liknande resultat [16] [18].

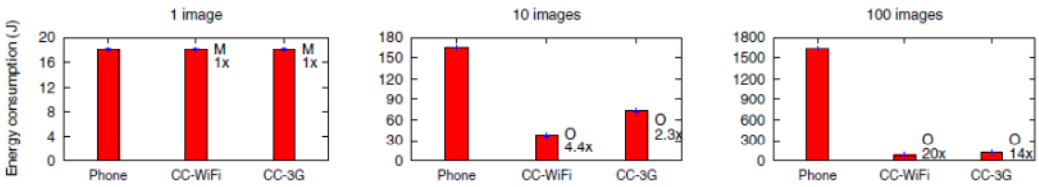


Figur 5 MAUI energiförbrukning [16]

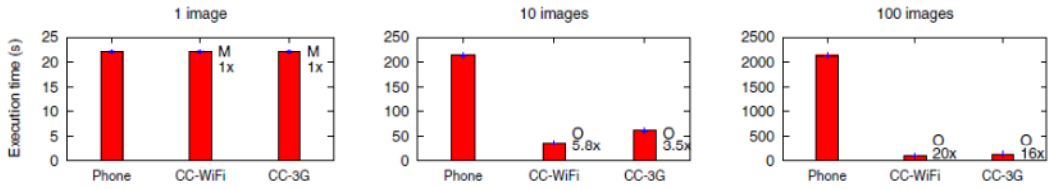


Figur 6 MAUI Exekveringstid [16]

I figur 5 ser vi en jämförelse av enhetens energiförbrukning då ansiktsigenkänningsapplikationen endast körs lokalt på och då MAUI optimerar avlastningen vid olika nätverkslatenser. Ansiktsigenkänningsapplikationen har kring tio metoder som potentiellt kan avlastas och det är programmeraren som har märkt dem *remoteable*. Som vi kan se i figuren är energivinningen betydande. Över Wi-Fi med en latens på 100ms är energiförbrukningen nästan tio gånger högre då applikationen endast körs lokalt. Över en 3G-förbindelse med en latens på 220ms är energiförbrukningen fyra gånger så hög då koden endast körs lokalt. Prestandaförbättringen är likaså betydande då ansiktsigenkänningsapplikationen avlastas med MAUI, vilket vi kan se ur figur 6. Utan avlastning tar exekveringen ungefär 19 sekunder. Då metoder avlastas över Wi-Fi med latensen 100ms är exekveringstiden kring 2500ms och minskar således med ungefär 85 %. Då metoder avlastas över en 3G-förbindelse är exekveringstiden ungefär 4000ms och således då också endast en femtedel av exekveringstiden lokalt [16].



Figur 7 CloneCloud energiförbrukning [18]



Figur 8 CloneCloud exekveringstid [18]

Figur 7 visar resultatet då ansiktsigenkänningsapplikationen kör med 1 bild, 10 bilder och 100 bilder. Varje diagram visar energiförbrukningen för applikationen då den körs lokalt på maskinen, då avlastning sker över en Wi-Fi-förbindelse och då avlastning sker över en 3G-förbindelse. Latensen över Wi-Fi-anslutningen är 69ms och över 3G-anslutningen 680ms. Då endast en igenkänning skall göras bedömer CloneCloud att det inte lönar sig att avlasta metoder utan exekveringen sker på enheten. Energikonsumtionen är i detta fall ungefär 18J. Då igenkänning skall göras på 10 och 100 bilder ser vi dock en stor energibesparing tack vare avlastning. Emedan energiförbrukningen stiger linjärt med antalet bilder då programmet kör lokalt, ökar den betydligt långsammare ifall avlastning sker. För 10 bilder är förbrukningen mindre än en femtedel så stor då avlastning sker över Wi-Fi och över hälften mindre då det sker över 3G. För 100 bilder är skillnaderna ännu större. Besparingarna är upp till 95 % då avlastning sker över Wi-Fi och över 90 % då avlastning sker över 3G. Vi ser liknande resultat när det kommer till förbättring av prestanda, vilket vi kan se i figur 8. Även i detta fall lönar det sig inte att avlasta då endast en igenkänning görs utan exekveringen sker på enheten. Exekveringstiden är då ungefär 22 sekunder. Liksom energiförbrukningen stiger exekveringstiden på enheten linjärt med antalet bilder. För 10 bilder är exekveringstiden mindre än en femtedel då avlastning görs över Wi-Fi och nästan en fjärdedel då den sker över 3G. Då igenkänning sker på 100 bilder är exekveringstiden endast 5 % och 6 % för avlastning över Wi-Fi respektive 3G jämfört med om exekveringen sker på enheten [18].

I de båda försöken undersöktes även hur länge det tar för optimeringsramverken att producera rätt partitioner för applikationen i en viss situation. För de olika applikationer som testades, tog den statiska analysen i CloneCloud i medeltal 23

sekunder att utföra, dock är det en beräkning som endast måste göras en gång per applikation. Lösningen av optimeringsproblemet tog under en sekund [18]. Optimeringen i MAUI tog i medeltal 18ms för schackapplikationen och 46ms för videospelet. Resultat för lösningstiden för ansiktsigenkänningsapplikationen meddelas inte men kan antas vara i samma storleksklass [16].

4. Slutsatser

5. Litteraturförteckning

[1] http://www.independent.co.uk/life-style/gadgets-and-tech/news/there-are-officially-more-mobile-devices-than-people-in-the-world-9780518.html (18.2.2015)

[2] https://gsmaintelligence.com/ (18.2.2015)

[3] http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536 (18.2.2015)

[4] http://www.statista.com/chart/1138/tablet-and-pc-shipment-forecast/ (18.2.2015)

[5] http://techcrunch.com/2014/08/21/majority-of-digital-media-consumption-now-takes-place-in-mobile-apps/ (18.2.2015)

[6] källa saknas.

[7] http://www.digitaltrends.com/computing/why-your-smartphone-wont-be-your-next-pc/ (24.2.2015) + http://www.techadvisory.org/2013/12/processors-computer-vs-mobile/

[8] The NIST Definition of Cloud Computing

[9] Developing Cloud Software Algorithms Applications and Tools

[10] See what cloud can do for you. Uncomplicated cloud business. Tieto insight.

[11] Above the Clouds: A Berkley View of Cloud Computing

[12] Understanding the Cloud Computing Stack – SaaS PaaS IaaS

[13] Overview of the Backend as a Service

[14] A survey of mobile cloud computing: achitecture, applications and approaches
[15] Mobile cloud computing, A survey
[16] MAUI – Making Smartphones Last Longer with Code Offload
[17] Mobile Computation Offloading – Factors Affecting Technology Evolution
[18] CloneCloud: Elastic Execution between Mobile Device and Cloud
[19] Mobile Cloud Computing