Åbo Akademi

fakulteten för
naturvetenskaper och teknik

Kandidatavhandling

Parallella och distribuerade beräkningar med Techila Grid

Skribent:
Kim Paulin, 29767

2016

Abstrakt

Behovet att hantera och analysera stora mängder data fortsätter öka inom områden som forskning och inom affärsvärlden. Beräkningskraftiga system för att lösa sådana problem kan ändå konstrueras relativt billigt och enkelt med Java-baserad middleware hos slutanvändare och på underutnyttjade datorer. Ett inhemskt system för distribuerade parallella beräkningar, Techila Grid, kan med mindre ändringar i färdiga program sprida beräkningsbördan till molnet eller företagets övriga datorer. Denna avhandling behandlar konceptet med parallella beräkningar på ett allmänt plan samt granskar Techila Grid systemets lösning.

Innehåll

Kapitel 1

Inledning

Informationsteknik fyller en växande roll inom många olika grenar av vetenskap där den används för hantering och analys av data. Nya fall där forskning skulle ha nytta av att använda stora mängder beräkningskapacitet hittas kontinuerligt ju kraftigare datorer blir och desto mer applicerbara algoritmer som hittas för olika slags problemområden. Även inom företagsvärlden fortsatter mängden insamlad data växa och kan stå som grund för smartare beslut med fotfäste i fakta om produkter, gränssnitt och kundbeteende istället för att grunda sig på subjektiva åsikter. Detta fenomen är så stort att det har myntat termen och IT-området '*big data*' där man använder sig av enorma mängder beräknings- och lagringskapacitet [1].

Inom detta område stöter man ofta på datorers begränsade kapacitet på ett helt annat vis än kontoristen på sin arbetsmaskin som lungt kör sitt textbehandlingsprogram. Processorn arbetar till sin fulla kapacitet och beräkningar kan ta dagar istället för millisekunder. Då börjar man intressera sig för hur man kan effektivera programmet och möjligtvis utnyttja flera kärnor eller processorer samtidigt för att snabbare få fram resultat och man börjar komma in på området för högeffektiva datorberäkningar, användningar av superdatorer. Parallellisering av problemet gör att man kan distribuera beräkningsbara delar till flera datorer som samarbetar i form av ett kluster där maskiner är enhetliga och nära sammankopplade eller som ett mer heterogent och utspritt parallellt .

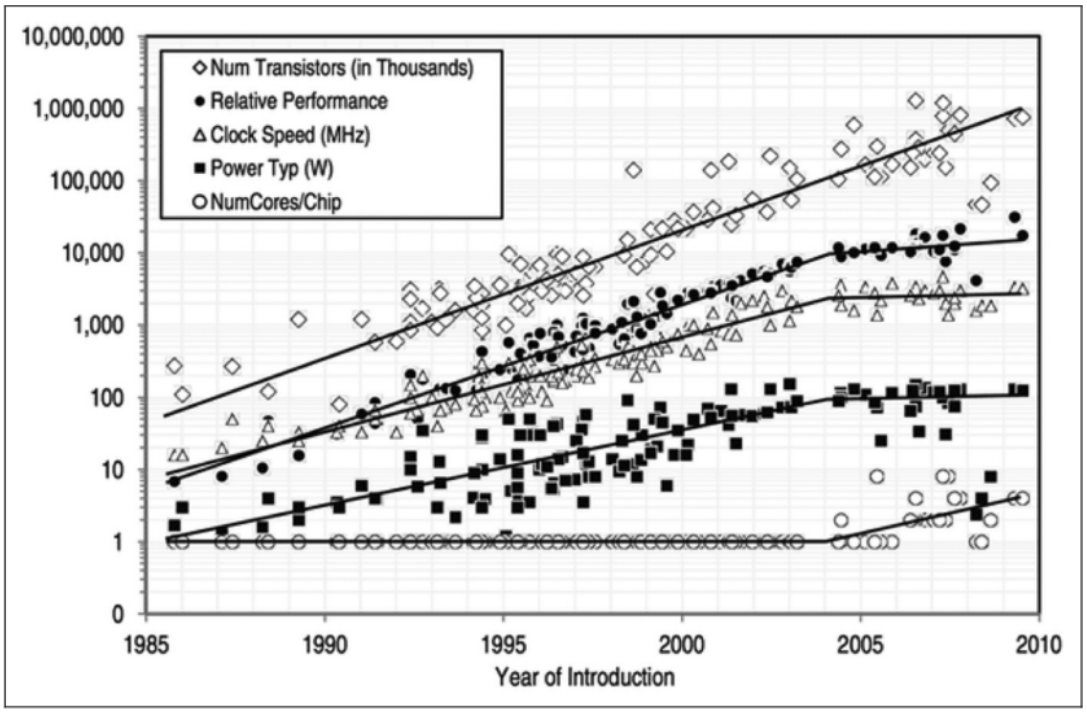
I denna avhandling analyseras konceptuellt hur produkten Techila Grid kan användas för att gå över från att lokalt köra stora beräkningar till att sprida på bördan och utnyttja beräkningskapacitet utanför den egna datorn eller servern,

antingen med organisationen lokala resurser i form av dedikerade eller delvis tillägnade datorer eller t.o.m. tillfällig beräkningskapacitet i form av molntjänster. Avhandlingen undersöker också var Techila Grid systemets möjligheter och begränsningar ligger och försöker definiera hurdana beräkningar som har störst potential till snabbare parallell exekvering.

Kapitel 2

Distribuerade beräkningar

IT har trängt sig in som ett verktyg i de flesta informations- och kunskapsfokuserade branscher. Mängden insamlad data och än mer behovet av processering av data ökar i enorm takt vilket har ställt IT-branschen inför nya utmaningar. En dator räcker inte till utan flera kopplas samman via ett nätverk till en parallell dator, en *multidator* (eng. multicomputer) för att samarbeta mot att lösa ett gemensamt problem. [2]



Figur 2.1: Utvecklingen av processorers beräkningskapacitet. [3]

Man kan påstå att Moores lag som år 1965 förutspådde att antalet transistorer, och därmed beräkningskapaciteten, i mikroprocessorer fördubblas vartannat år, har stagnerat kring 2005 (se figur 2.1) och pga fysiska begränsningar kan man inte längre förlita sig på den snabba utvecklingen av hårdvarans prestanda (klockfrekvens, exekveringsoptimering) skulle ge en gratis uppsnabbning av långsamma algoritmer och som skulle göra dem mer användbara. Sedan åren kring 2005 har man mer ökat antal kärnor per chip och förlitar sig istället på att utveckla parallellisering av program som kan dra nytta av hyperthreading och flerkärniga processorer. [4]

Det kan finnas flera orsaker till att vela övergå till parallella beräkningar på flera processorer eller datorer. I huvudsak möjliggör det mer arbete per tidsenhet. Ett givet problem kan lösas snabbare då arbetet sprids ut på flera enheter. Parallellisering kan istället ge möjligheten att lösa större problem, också sådana som p.g.a. deras storlek tidigare inte varit möjliga att lösa inom rimlig tid. [3]

2.1 Vad är distribuerade parallella beräkningar?

Man kan fråga sig när man behöver övergå till distribuerade beräkningar istället för att bearbeta data på den lokala datorn? För exekvering lokalt är det enklast att utveckla sekventiell kod utan att dra nytta av parallellism. Det är entydigt i vilken ordning kodrader körs och det är enklare att följa och förstå programmets beteende.

För tyngre beräkningar uppnås lätt full utnyttjandegrad av en processeringsenhet och man skulle ha nytta av en flerkärnig processor eller då man vill utföra operationer som tar länge asynkront från det övriga programmet kan man använda parallell exekvering med trådar. Trådar karakteriseras av ett delat minne men om denna egenskap inte används och trådar mer liknar egna isolerade processer passar de bra för s.k. trivialt parallella problem [5]. Trådar kör ändå i icke-deterministiskt ordning och användning av gemensamma resurser hos olika trådar kan leda till komplikationer som gridlock [6] och omskrivning av program för att använda sig av trådar är inte helt enkel.

Då kärnorna i en dators processor inte mer räcker till då det kommer till beräkningskapacitet kan man, med vissa begränsningar och bieffekter, dela upp exekveringen av parallella delar av programmet för att köras på skilda datorer

och därmed dra nytta av större mängder processeringskapacitet. Detta är vad som menas med distribuerade beräkningar. På detta sett har man ett större arbetsminne och lagringskapacitet att tillgå som inte skulle vara möjlig på en lokal dator. Men koordineringen av delprocesserna tillför en viss overhead vars storlek bör beaktas och delproblemen ska vara tillräckligt stora för att distribueringen ska vara effektiv och leda till en snabbare lösning av problemet.

Enheter som exklusivt ingår i ett homogent lokalt system bildar ett *kluster*. Klustrets enheter har god kommunikation och arbetar för att lösa ett delat problem. Även mer heterogena och geografiskt utspridda datorer kan semantiskt fungera som ett enhetligt parallellt system men kalls då ofta för ett *grid*. Detta system kan hantera delar av programmet parallellt och slutligen sammanställa resultaten - möjligast transparent för programmet som exekveras.

2.2 Lämplighetsanalys

Ett allmänt beräkningsbart problemet behöver till en början analyseras för att identifiera andel sekventiell kod och iterativa delar i koden. En parallell eller distribuerad version av programmet kan vara motiverad då stora delar av programmets körtid spenderas i en loop-struktur medan endast data byts ut eller flera komplexa analyser ska göras över samma data. Dessa två egenskaper kan karakterisera programmet antingen som dataintensivt eller beräkningsintensivt.

Parallelliserbara problem grupperas även m.a.p. hur självständigt dess delproblem kan lösas. Då problemets karaktär möjliggör att t.ex. en loops iterationer inte är rekursiva eller har andra beroendeförhållanden utan kan lösas totalt isolerade och självständigt kallas detta för ett “trivialt parallellt problem” (eng. embarrassingly parallel problem). Om delproblemen däremot behöver kommunicera sinsemellan för att kunna nå ett resultat så kallas det för ett parallellt problem (eng. parallel problem)

För att bedöma effektiviteten av en eventuell parallellisering kan man använda sig av Amdahls lag. En vidareutveckling av lagen som beaktar även massivt parallella system har gjorts av John L. Gustafson och Edvin H. Barsis.

2.2.1 Amdahls lag

Uppsnabbningen till följd av att parallellisera ett program formaliseras i Amdahls lag. Funktionen *T(p)* beskriver exekveringstiden för ett program med en seriell del s och den kvarstående parallelliserbara delen (1-s). Som parameter anges de parallellt tillgängliga och sinsemellan likvärdiga exekveringsenheterna p. Hela programmets exekveringstid beräknas som summan av den seriella delen som bara kan köra på en enhet och den resterande parallelliserbara delen (1-s) som multipliceras med exekveringstiden T(1) men som kan uppdelas på p exekveringsenheter.

$$T(p) = sT(1) + (1 - s)\frac{T(1)}{p}$$

För att kalkylera uppsnabbningen av ett parallelliserat vs ett seriellt program kan man beräkna *S(p)* enligt

$$S(p) = \frac{T(1)}{T(p)} = \frac{1}{s + (1 - s)/p}$$

dvs tiden för exekvering av programmet på en enhet delar exekveringstiden för samma program på p enheter. T(p), på engelska kallat *wall-clock time*, anger summan av den tid som använts för beräkning av problemet över alla använda processorer. Här kan man observera att då p ökar går S(p) mot 1/s vilket leder till att ifall den seriella delen av ett program utgör mer än 1% kan programmet aldrig försnabbas mer än 100-faldigt oberoende hur många parallella enheter som tillförs. [3]

Effektiviteten anges av *E(p)* enligt

$$E(p) = \frac{T(1)}{pT(p)} = \frac{1}{sp + (1 - s)}$$

vilket ställer exekveringstiden för programmet på p enheter i förhållande till seriell exekvering, dvs på en enhet.

2.2.2 Gustafson-Barsis lag

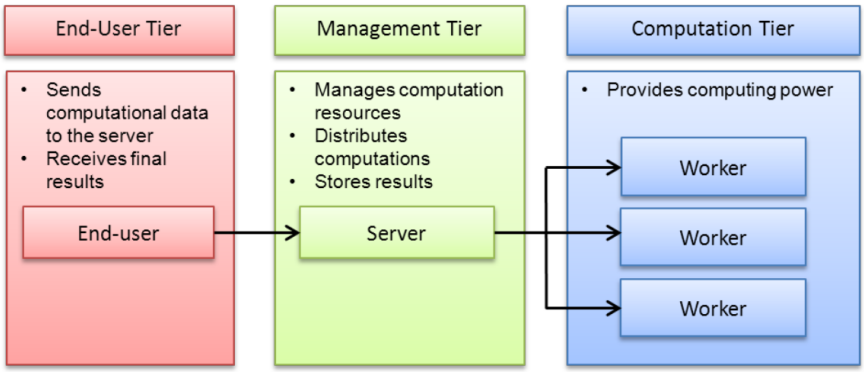
Gustafson-Barsis lag är en utveckling av Amdahls lag för massivt parallella system. Lagen tar i beaktande att om ny hårdvara blir tillgänglig, t.ex. med snabbare nätförbindelse noder emellan eller effektivare processering, kan ett större problem processeras under samma tid. Amdahls lag antar att problemets storlek som beräknas alltid förblir densamma. I verkligheten är oftast fallet det att problemstorleken kan ökas för att bättre dra nytta av hårdvaran och då är det endast den parallelliserbara delen som ökar medan den sekventiella kodens storlek bibehålls. Därmed minskar den andelen processeringstid som sekventiell kod upptar vilket i Amdahls lag var den del som begränsat effektiviteten då antalet noder i systemet ökar.[7]

$$Sp(N) = (1 - P) + P \cdot N$$

Kapitel 3

Systemet Techila Grid

Techila Technologies Ltd är ett finskt företag med säte i Tammerfors som utvecklat Techila Grid — ett system för distribuerat beräkningar för vilket man med mindre ändringar kan ta ett existerande lokalt exekverande program och omforma det till att utnyttja redan tillgänglig, oanvänd datorkapacitet i organisationen och använda dem för högeffektiva beräkningar eller köra högeffektiva beräkningar m.hj.a molntjänster utan investeringskostnader för hårdvara.



Figur 3.1: Systemet består av tre delar; SDKn för slutanvändaren, mjukvara för systemets administration och slutligen beräkningsnoderna. [8]

Techila Grid är en helhet som består av tre logiska delar eller skikt (se fig. 3.1). För slutanvändaren har man använt mellanprogramvara (eng. middleware) som tar över t.ex. MATLAB programmets exekvering på slutanvändarens dator för att sprida ut tunga beräkningar på flera maskiner kopplade till systemet. Slutanvändaren använder här ett lämpligt Techila SDK avsett för ifrågavarande programmiljö som förhållandevis lätt kan integreras i existerande program. SDKs

finns att tillgå för många populära tillämpningsspecifika programpaket som R skript och MATLAB men också för högnivå programmeringsspråk som Python, C/C++, C# /.NET, Perl och Fortran, m.fl.

För att sköta beräkningarna kan olika typer av hårdvara och molntjänster anslutas som enheter till beräkningsskiktet. Detta kan vara en blandning av olika plattformer med diverse operativsystem, varierande hårdvara i form av allt från servers till laptopar och även inköpt kapacitet i form av molntjänster. Om organisationen har en datorsal med datorer som används sporadiskt kan de vid outnyttjade tillfällen stå till förfogande som beräkningsnoder i Techila Grid. [8]

Det som orkestrerar allt detta är Techila Server som tar emot och hanterar kön med kvarstående jobb, vilket projekt de tillhör, vilka beräkningsnoder som finns tillgängliga i systemet och deras status samt distribueringen av jobb till beräkningsnoder för exekvering. Även resultatet från beräkningar samlas in på Techila Server och sammanställs innan det skickas till slutanvändaren.

3.1 Kännetecknande egenskaper

I många fall är kravet på programmeringskunskap och -teknik för hög för att utnyttja möjligheten att distribuera beräkningar. Även investering i hårdvara kan vara en tröskel. För slutanvändaren möjliggör Techila Grid en lätt transition från lokal exekvering till distribuerade beräkningar på egen hårdvara till tillfällig beräkningskapacitet i form av molntjänster. Då det kommer till slutanvändarens kod krävs det i bästa fall endast en lätt installation av och introduktion till Techila SDK och en ändring av en for-loop till en cloudfor-loop i ett existerande MATLAB eller R script program.

3.2 Techila för slutanvändaren

SDKn innehåller funktioner som underlättar användandet av Techila APIn och kan laddas ner från Techila Technologies' Extranet efter gratis registrering och godkännande av företaget. Där finns också beskrivande guider för slutanvändare som behandlar de vanligaste programmeringsspråken.

Slutanvändaren har den egentliga koden för de tunga beräkningarna i projektet. Sedan används funktioner från SDKn för att definiera beräkningsenheter,

deras parametrar och olika stödpaket (bundles). Ev. vill man också göra någon slags efterprocessering av resultat från de returnerade resultaten från beräkningarna. Detta kan definieras som en valfri parameter till *peach*-funktionsanropet (se nedan) och én *callback*-funktion kan anropas då delresultat från en beräkningsenhet returneras från Techila Grid till den lokalt exekverande koden hos slutanvändaren. Allt detta definieras i kontrollkod som i dokumentationen kallas *Local Control Code*. [8]

Särskilt programpaketet *MATLAB* och skriptspråket *R* utmärker sig som de mest integrerade med Techila i.o.m att deras SDKn innehåller den avancerade *cloudfor*-funktionen som automatiserar användningen av peach-funktionen. Det ända man kan behöva se efter är hur många loop iterationer en beräkningsenhet ska inkludera. Innebär en beräkningsenhet för lite jobb på noden går det proportionellt åt för mycket tid till för- och efterarbete samt överföring för att systemet ska fungera effektivt. Denna designfas kan kallas för *agglomeration*. [2]

Om cloudfor-funktionen inte ingår i plattformens SDK finns ändå oftast peach-funktionen att tillgå. Den är mer krävande för slutanvändaren att förstå och använda men tillåter en större möjlighet till konfiguration. Man kan också kalla på API funktioner direkt i Techila APIn skriven i Java. Här följer ett exempel på *techila.peach*-funktionen från Python SDKn.

```
techila.peach(funcname = 'example_function', # Function executed on Workers
              params = [var1,var2],          # Input arguments to the function
              files = ['codefile.py'],       # Evaluated at the start of a Job
              datafiles = ['file1','file2'], # Files transferred to Workers
              peachvector = [2,4,6,8,10]     # Peachvector definition
              )
```

Figur 3.2: Exempel på användning av peach-funktionen i Techila SDK för Python [9]

Parametern *funcname* anger den lokala funktion i koden som beskriver de tunga beräkningar som ska köras hos en beräkningsnod. Den förutnämnda funktionen kan då behöva förses med vissa parametrar för att fungera korrekt och anges då som en lista ([var1, var2]).

Vill man förse sin funktion med varierande input parametrar för att på så sätt kunna beräkna olika delresultat kan params-listan innehålla elementet ’<param>’ som signalerar Techila att dynamiskt ersätta denna parameter med

ett element från listan som utgör den sista peach-parametern <paramvector>. Även '<vecidx>' kan användas och ersätts då med det aktuella elementets index i paramvector-listan (elementets ordinaltal som startar från 0). Längden på paramvector-listan anger också antalet beräkningsenheter som projektet genererar.

files-parametern beskriver de filer som python behöver ladda (importera) före beräkningarna kan utföras. Slutligen kan man ange en lista med filer i parametern *datafiles* som peach-funktionen genererar s.k. *'data bundles'* och som bifogas projektet, lagras i Techila Servern och skickas till beräkningsnoderna enligt behov. Detta är naturligtvis nödvändigt ifall ett dataintensivt problem ska beräknas.

3.3 Krav på datormiljön

Techila Grid SDKn kör på Linux, OSX och Windows. Eftersom Techila SDKn bygger på Techila APIn som i sin tur är Java baserad behöver man ha en Java miljö installerad (Java JDK eller JRE ver. 6 eller nyare)[10]. Det finns också dynamiskt länkade bibliotek (.dll) för Windows miljöer, motsvarande delade objekt (.so-filer) i linux-miljö och .dylib-bibliotek för OS X. .NET-ramverket har en egen nativ Techila API och kräver inte Java om t.ex. C# används. [8]

Tilläggskrav hos slutanvändaren beror kraven på vilken plattform man använder för sitt projekt. T.ex. för Python importeras bara paketet *techila* som innehåller alla de nödvändiga moduler som behövs medan man med R skript förutom det specifika *techila* paketet kräver installering av två paket från standard CRAN paketarkivet för R.

Genererade beräkningsenheter som ska exekvera på beräkningsnoder ute i Techila systemet kan kräva en viss miljö, tillhörande data att behandla, olika stödande kodbibliotek (eng. libraries) som koden stöder sig på och ev. någon form av en tolk (python) som kan köra beräkningskoden på målplattformen. För MATLAB kan programmet m.hj.a. *MATLAB Compiler Toolbox* kompileras till en distribuerbar binärfil hos slutanvändaren så att inte MATLAB miljön behöver vara installerad hos beräkningsnoder [8].

Om Techila Grid körs i molnet behöver man endast ett konto på techila.fi för att kunna ladda ner SDKn och generera ett Techila projekt hos slutanvändaren. Ett konto hos molnberäkningstjänsten behövs med faktureringsuppgifter för att

kunna starta en virtuell maskin för Techila Server och beräkningsnoder.

Ett typiskt Techila-system som använder Google Compute Engine med en virtuell maskin med 4 kärnor och 64GB hårdskiveutrymme kostar 104,76 USD / månad i uppehåll. Den kan köra Techila Servern. Detta är utan beräkningsnoder. Dessa kräver mjukvara från Techila som företaget hyr ut via Google för 0,10 USD / kärna / timme eller 73 USD / kärna / månad. [11]

3.4 Avancerade funktioner

Organisationer som har existerande hårdvara med låg utnyttjningsgrad, exempelvis en datorsal vid ett universitet eller en ingenjörsfirmas arbetsstationer kunde nattetid använda sin samlade beräkningskraft för att kalkylera stresstest eller bidra till forskning vid universitetet. som annars skulle kräva investeringar i hårdvara eller köpt datorkapacitet i form av molntjänster.

Techila Grid har stöd för sk. *snapshotting* där en större beräkningsenhet under exekvering på en nod regelbundet kan spara mellanresultat och skicka dem till Techila Servern. Då genereras mer nätverkstrafik men resultatet från långa beräkningar går inte förlorade trots att beräkningsnoden blir upptagen eller bortfaller. Techila Servern kan dela ut den halvfärdiga enheten till en annan nod som slutför jobbet. En begränsning för att en snapshot ska skapas är ändå att minst en minut av beräkningar ska ha utförts.

Normalt skickas resultat från beräkningsenheter till Techila Server för att alla samlat skickas till slutanvändarens lokala kod för ev. efterbehandling. Vill man se resultat vartefter delresultat blir färdiga från beräkningar hos noderna i Techila Grid kan man använda *streaming* som genast skickar resultatet vidare till den lokala koden hos slutanvändaren och i callback-funktionen kan t.ex. en graf uppdateras eller tyngre efterbehandling av data kan köras vartefter delresultat strömmar in. Dessutom sprids ansträngningar på nätverket mellan Techila Server och slutanvändaren ut på en längre tid ifall bandbredden är begränsad. [8]

Arbetar man i ett Techila Grid med beräkningsnoder som kör på olika operativsystem kan fjärrkompilering vara av intresse. Då distribueras inte den förkompilerade binären från slutanvändaren utan den beräkningsbara koden innehåller instruktioner för en viss plattform att själv kompilera programmet före exekvering på noden. Den producerade binären kan vid behov överföras tillbaka till

slutanvändaren för att skapa ett multi-plattforms projekt där binärer för flera plattformer eller en plattform olik slutanvändarens kan distribueras. [8]

3.5 Kommunikation mellan beräkningsnoder

Parallella beräkningar utförs där beräkningsenheter inte är fristående och oberoende av varandra utan kräver kommunikation sinsemellan för att kunna beräkna sitt delproblem och nå ett resultat. Om adminstratören av Techila Grid systemet skapat grupper av beräkningsenheter som har möjlighet att skapa detta tillfälliga nätverk mellan sig kan slutanvändaren använda sig av systemet för att lösa parallella problem.

Eftersom beräkningsenheter behöver information av varandra för att slutföras måste tillräckligt av dessa speciella beräkningsnoder finnas tillgängliga samtidigt i Techila Grid. SDKn innehåller kommandon för att initialisera nätverket, skicka och ta emot meddelanden enskilda noder emellan samt skicka ut ett meddelande till alla noder i nätverket.

Kapitel 4

Sammanfattning och diskussion

Parallellism och distribuerade beräkningar fortsätter vinna mark i ett informationssamhälle där aktiehandeln långt styrs av algoritmer och där en av mänskligheten stora utmaningar just nu är kartläggning och en ökad förståelse av organismers komplexa biokemiska system. Hantering och analys av data är något som man kontinuerligt behöver mer av. En kommersiell finsk produkt aktuell inom detta växande område har granskats i denna avhandling.

Techila Grid verkar vara en användarvänlig och lättimplementerad lösning för att öka produktiviteten i många dataintensiva organisationer. Kan användaren skriva egen kod för sitt projekt behöves knappast lång skolning för att kunna utnyttja ett förinstallerat Techila Grid system. Arbetstagares värdefulla tid kan m.hj.a. Techila Grid användas till annat än att vänta på beräkningsresultat och en snabbare utvecklingsprocess är möjlig.

Med en fullt molnbaserad plattform och skalbar beräkningskraft öppnas en möjlighet för helt nya grupper av användare att dra nytta av högeffektiva beräkningar och tröskeln att prova på systemet är låg i.o.m. att man kan betala per timme av användning. Samtidigt kan större organisationer med kraftiga arbetsstationer öka utnyttjningsgraden på sin existerande hårdvaruinvestering. I tider av dataläckor och -intrång lämpar produkten sig väl för företag med känslig data som man vill hålla intern om systemet körs på endast egen hårdvara.

Den tekniska lösningen man valt med att basera APIn på Java och kräva att en Java-miljö för exekvering finns installerad på beräkningsnoderna är resonlig. Man kan på så vis satsa på utveckling av en produkt som oberoende plattform kör som förväntat. Användaren kan vid behov ändå köra nativ kod på beräk-

ningsnoden och således lider inte prestandan. Man har också SDKn för ett flertal populära språk och plattformer vilket gör användningen naturlig för användaren.

Priset för användningen av produkten Techila Grid är i flera scenarion oklar. Prisuppgifter hittades utsatt endast för Google Cloud Engine integrationen medan prissättningen för mjukvara ämnad för intern användning på egen hårdvara förblev okänd. Antagligen uppgörs kontrakt kundvis.

Litteraturförteckning

[1] E. C. Hayden, "Genome researchers raise alarm over big data," *Nature News*, 2015.

[2] I. Foster, *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[3] M. T. Heath, "A tale of two laws," *International Journal of High Performance Computing Applications*, vol. 29, no. 3, pp. 320–330, 2015.

[4] H. Sutter, "The free lunch is over: A fundamental turn toward concurrency in software," *Dr. Dobb's Journal*, pp. 1–9, 2005.

[5] E. Lee, "The Problem with Threads," *Computer*, vol. 39, pp. 33–42, may 2006.

[6] A. S. Tanenbaum, *Modern Operating Systems*. Upper Saddle River, NJ, USA: Prentice Hall Press, 3rd ed., 2007.

[7] J. L. Gustafson, "Reevaluating Amdahl's Law," *Commun. ACM*, vol. 31, pp. 532–533, may 1988.

[8] Techila Technologies Ltd, *Techila Fundamentals*. No. November, 2015.

[9] Techila Technologies Ltd, *Techila with Python*. No. November, 2015.

[10] Techila Technologies Ltd, *Techila in Google Cloud Launcher Guide*. No. April, 2016.

[11] Google Inc., "Cloud Launcher - Techila Distributed Computing."