LUNDS UNIVERSITET

Examensarbeten i datavetenskap 29 augusti 2014

INSTITUTIONEN FÖR DATAVETENSKAP

STUDENTER:

Dennis Andersson
Carl Fagerlin
Henrik Gyllensvärd
Linnea Johansson
Erik Karlsson
Jesper Lekland
Asmir Sabanovic
Astrid Stenholm
Oguz Taskin
Marie Versland
Gustaf Waldemarson
Magnus Walter
Niklas Welander

Program 29 augusti 2014

	LOKAL: E:2116	LOKAL: E:2405
Kl 09.15	**Varianthantering och gemensam kodbas för webbaserad apputveckling** **Författare:** Henrik Gyllensvärd, Niklas Welander **Handledare:** Thomas Hermansson (ÅF) **Examinator:** Lars Bendix (LTH)	**Schemaläggning i manycore realtidsystem** **Författare:** Erik Karlsson **Handledare:** Aneta Vulgarakis (ABB) **Examinator:** Flavius Gruian (LTH)
Kl 10.15	**Kompression av ytdata på dynamiska topologier** **Författare:** Oguz Taskin **Handledare:** Jan Schmid (DICE) **Examinator:** Michael Doggett (LTH)	**User Independent Gym Exercise Recognition Using a Single Wrist Worn Accelerometer** **Författare:** Jesper Lekland, Asmir Sabanovic **Handledare:** Håkan Jonsson (Sony Mobile/LTH) **Examinator:** Pierre Nugues (LTH)
Kl 11.15	**Ray-packet-tracing med ARM:s NEON-arkitektur** **Författare:** Gustaf Waldemarson **Handledare:** Johan Grönqvist (ARM) **Examinator:** Michael Doggett (LTH)	**Analyzing large data streams** **Författare:** Carl Fagerlin **Handledare:** Jacek Malec (LTH) **Examinator:** Pierre Nugues (LTH)
PAUS		
Kl 13.15	**Implementation of a condition-based maintenance tool for critical components** **Författare:** Dennis Andersson, Astrid Stenholm **Handledare:** Niklas Svanberg **Examinator:** Klas Nilsson (LTH)	
Kl 14.15	**Regressionstestning i ett konkret mjukvaru-projekt** **Författare:** Linnea Johansson, Magnus Walter **Handledare:** Magnus C Ohlsson (System Verification) **Examinator:** Emelie Engström (LTH)	
Kl 15.15	**Bedömningsmodell för systemintegrations-lösningar** **Författare:** Marie Versland **Handledare:** Elizabeth Bjarnason (LTH) **Examinator:** Per Runeson (LTH)	

Texterna i detta program är preliminära och ännu inte formellt godkända av examinatorerna.

Varianthantering och gemensam kodbas för webbaserad apputveckling

POPULÄRVETENSKAPLIG SAMMANFATTNING
HENRIK GYLLENSVÄRD, NIKLAS WELANDER

Handledare: Thomas Hermansson (ÅF)
Examinator: Lars Bendix (LTH)

En inblick i applikationer

När man bygger ihop mobila applikationer idag så är det vanligt att man gör det till flera olika system och telefoner, idag är Android och iPhone de mest kända exemplen. Man vill ofta bygga samma applikation, den ska alltså både se ut och bete sig likadant. Problemet för utvecklarna är att plattformarna har sina helt egna språk och strukturer. Tänk dig att en applikation är som styrdonet på ett fordon och du tillverkar dessa. Problemet är att mobilplattformerna nu är så olika att de kan ses som helt olika fordon: Android - flygplan och iPhone - motorcykel. Uppenbarligen kan vi inte använda samma ”ratt” på alla fordon.

Problem med dagens utveckling

Tillverkningen av styrdon skiljer sig så mycket att tillverkningen tvingas delas upp i helt skilda grupper spritt över världen och dessa grupper har ingen anledning till att kontakta varandra, de vet hur ”applikationen” ska se ut. För ditt företg gäller då att göra investeringar i specialistkunskaper och grupper världen runt och hålla kolla på dessa. Och om en ändring ska göras på ”applikationen” så måste alla grupperna planera och införa den seperat från varandra.

Nytt fordon, flera modeller

Just nu introduceras nya mobilplattformar på mobilmarknaden som stödjer appar skrivna i web-format. I och med att det finns flera plattformar som stödjer detta så kan vi se dessa som olika bilsorter och applikationen blir en bilratt. Man vill dessutom ofta när man utvecklar appar komma åt hårdvaran på mobilen, vi kan se detta som knapparna på ratten (blinkers, torkare, stereo etc.). Även om nu alla har möjlighet att göra bilrattar på exakt samma sätt uppkommer ett snarlikt problem från tidigare. Olika bilmärken gör olika fästanordningar för rattarna så vi har fortfarande problem om vi vill göra en ratt till alla bilmärken.

Exakt samma ratt, med varianter

Det vi vill undersöka är möjligheterna att kombinera ihop byggandet av rattarna. Kan vi lyckas komma runt problemet med att ha olika lag som jobbar på samma sak? Skillnaderna i fästningsanordningen kommer kvarstå, men man kan kanske bygga en lösning för själva ratten och bygga ihop olika fästen på den? Man kommer fortfarande ha flera modeller för fästena, men man har ett gemensamt sätt att bygga övriga delar av ratten på. På så sätt kan man minimera det till att ha en grupp som utvecklar ratten för flera modeller samtidigt.

Kommunicera med hårdvaran

Att slutanvändaren, bilisten, ska ha en behaglig användarupplevelse har varit givet under en längre tid. I ratt-tillverkningen har vi velat underlätta interaktionen, men inte med bilisten. I varje bilmodell finns en fästningsanordning olik den nästa och det finns ett kluster med sladdar och kontakter som skiljer sig i varenda en. Som utvecklare behöver man nu, beroende på ratten man ska tillverka, lära sig vart varje sladd ska gå och hur varje kontakt ska kopplas in. På mobilsidan motsvarar detta ett bibliotek av anrop till hårdvaran. Även om biblioteken är skrivna på samma språk så är de så annorlunda att utvecklarna måste lägga tid på att lära sig alla (om man inte vill dela upp tillverkningen, och det ville vi ju inte). Så vad vi har gjort är att skapa ett bibliotek som länkar ihop de olika plattformarnas bibliotek. På bilen motsvarar detta en adapter som passar alla bilar men har samma utseende för ratt-tillverkaren.

Målsättning

Lösningen kan tyckas banal för exemplet med ratten. Tyvärr är de mobila plattformarna betydligt mer komplexa och det finns många faktorer att ta hänsyn till. Poängen är däremot densamma. Det vi alltså vill undersöka är om det går att kombinera ihop utvecklingen till de olika plattformarna. Detta skulle då lösa problemet med det dubbla underhållet. Dessvärre är problemet fortfarande att man behöver ha olika versioner för varje plattform, fästet skiljer sig ju fortfarande! Det man åstadkommit är att underlätta det dubbla underhållet. Däremot introducerar man olika varianter av ratten. De skillnader som finns behöver man hantera på ett bra sätt för att minimera mängden extra arbete. Så vi kommer även undersöka vilka problem som finns angående hanteringen av dessa varianter.

Slutsats

I slutändan lyckades vi att minimera det dubbla underhållet när man skapar applikationer. Resultatet blev ett verktyg som utvecklare kan använda sig av för att uppnå detta. Det är fortfarande bara en prototyp och det finns fortfarande vissa problem kvar, men vi har visat att konceptet fungerar och det går att jobba vidare med. Sammanfattningsvis kan man säga att man nu bara behöver en grupp som löser konstruktionen av styrdonet till flera olika typer av fordon, långt mycket bättre än det vi började med!

Schemaläggning i manycore realtidsystem

PRELIMINÄR SAMMANFATTNING
ERIK KARLSSON

Handledare: Aneta Vulgarakis (ABB)
Examinator: Flavius Gruian (LTH)

Hastigheten på datorer ökar otroligt snabbt, i enlighet med den så kallade "moores lag", som säger att datorers hastighet kommer att dubblas var18nde månad. En dators hastighet bestäms till stor del av hastigheten på dess processor, beräkningsenheten. Ett sätt att öka hastigheten på processorer har varit att öka klockfrekvensen, vilket innebär att fler beräkningar kan göras per sekund. Men ökningen av klockfrekvens har nått en gräns där det inte längre är praktiskt att öka den för att öka processorns hastighet. För att fortsätta öka hastigheten har en ny arkitektur skapats, så kallad "multicore". I en multicore-processor har det lagts till flera kärnor i en processor, där varje kärna är som en egen processor, detta innebär att man kan göra flera beräkningar samtidigt. Förenklat sagt så kan man (i fallet av två kärnor) säga att man delar upp ett jobb i hälften, och ger en halva till var kärna, då blir jobbet klart på halva tiden.

Network on Chip

Nu har vi nått en gräns där det även börjar bli problem med att lägga till fler kärnor på en och samma processor. Alla kärnor i en processor måste prata med varandra för att koordinera sina beräkningar, t.ex. meddela svaret på en ekvation så ett en annan kärna kan ta över och fortsätta beräkningarna. I en multicore processor kan bara ett meddelande skickas på en gång och medans det skickas är "bussen" som meddelandet skickades på upptagen, om antalet kärnor är högt så kan väntetiden på "bussen" bli väldigt lång.

En lösning på det här problemet är arkitekturen "Network on Chip" (NoC), där meddelanden skickas över ett nätverk istället för över en "buss". Nätverket ser ut som ett rutmönster. Vid varje ställe där två linjer korsas sitter en router, och varje router har en av processorns kärnor kopplad till sig. När ett paket skickas börjar det först vid routern som är kopplad till kärnan som skickar paketet, sedan åker det iväg till en närliggande router i riktning mot sin destination. På detta vis blir enbart en länk i taget mellan två routrar upptagen av ett paket när det skickas, men alla övriga länkar i nätverket är lediga. Med denna arkitektur kan man ytterligare öka antalet kärnor i en processor för att öka dessa totala hastighet.

Schemaläggning

Schemaläggning innebär att man tilldelar vilka program som ska köras på vilken kärna i en processor. Detta försvåras av att programmen kan ha olika beroenden, ett program kan till exempel vara beroende av att ett annat program har körts färdigt innan det kan börja. Schemaläggningen som behandlas i detta arbete är statisk schemaläggning, vilket innebär att ett schema skapas som sedan följs exakt som det är skrivet, och upprepas i oändlighet. När man skapar ett statiskt schema måste man veta hur lång tid ett meddelande tar att skicka, så att man vet när man kan starta programmet som ska behandla det. Det är beräkningen av denna nätverkstrafik som är den största skillnaden mellan att schemalägga för multicore- och NoC-processorer.

För multicore processorer, de som bara har en "buss" att prata över, finns det relativt välutvecklade metoder för att schemalägga olika program. Däremot finns det inte lika mycket arbete gjort kring schemaläggning av program på NoC-baserade processorer.

Vårt arbete går ut på att utveckla och undersöka schemaläggning för en NoC-processor. Schemaläggning för NoC kan delas upp i två delar, en del är metoden (eller ekvationen) för att räkna ut den fördröjning som ett meddelande kan orsaka, och den andra delen är själva algoritmen som används för att schemalägga. Vår schemaläggare riktas mot realtids-system, och måste därför räkna med det värsta tänkbara fallet när vi räknar ut fördröjningen av ett meddelande. Det är många nya variabler med i ett NoC-system jämfört med en multicore, framför allt måste man räkna med att ett meddelande kan hindras och behöva vänta på att annan trafik skickas färdigt för att en länk är upptagen. Detta kan lätt få en påbyggande effekt som snabbt växer till väldigt långa tider (ett meddelande väntar på ett som väntar på ett annat o.s.v.). Själva algoritmen som gör schemaläggningen kan hämtas eller inspireras av redan etablerade multicore-algoritmer, men måste anpassas för att hålla den dynamiska meddelande-tiden i åtanke, och försöka schemalägga så att minsta möjliga nätverkskollision sker.

Kompression av ytdata på dynamiska topologier

POPULÄRVETENSKAPLIG SAMMANFATTNING
OGUZ TASKIN

Handledare: Jan Schmid (DICE)
Examinator: Michael Doggett (LTH)

Realtidsgrafik som nästan exklusivt används för spel och andra interaktiva 3D-applikationer erbjuder idag hög realism och naturtrogenhet. En av teknikerna som bidrar till detta är texturering av 3D-modeller som får de enkla ytorna att se detaljerade ut. Traditionella metoder har dock problem som sömmar och diskontinuiteter vid gränser mellan trianglar som kan ibland uppstå. Mesh Colors är en textureringmetod som undviker dessa problem genom att associera data direkt med enskilda trianglar.

Komprimering av texturer är av intresse då det sparar både lagringsutrymme och resulterar i bl.a. snabbare laddningstider. Traditionella texturer består av vanliga bilder och kan komprimeras med existerande kodekar. Texturer i mesh colors-format är dock inte kompatibla med existerande kodekar och kan ej komprimeras med dessa. Detta examensarbete fokuserar på att implementera en JPEG-liknande kodek för komprimering av texturer i mesh colors-format.

Mesh colors

Mesh colors tilldelar varje hörn, kant och yta av trianglar ett antal element som bestäms av triangelns upplösning. Denna teknik stöder även MIP-mapping där rekursivt nedsamplade versioner av texturer också lagras för att användas då modellen befinner sig på ett visst avstånd från kameran. Versioner av texturen sägs ha MIP-nivå ”x” där nivåer med lägre upplösning har ett högre värde för ”x”. Till skillnad från traditionella texturer lagras dock mesh colors-data endimensionellt.

JPEG

JPEG är ett välkänt och ett ofta använt format för kompression av bilder. Formatet är en standard för destruktiv komprimering där den dekomprimerade bilden har en försämrad bildkvalitet. Detta görs dock i enighet med det mänskliga synsystemets egenskaper vilket resulterar i, beroende på kompressionsgrad, återskapade bilder som uppfattas ha minimal förlorad kvalitet.

En av de egenskaper som utnyttjas är den ökade känsligheten för ljusnivåer än den för färginformation. Detta utnyttjas genom att transformera data från de tre färgkomponenter de är indelade i (röd, grön och blå) till tre komponenter där den första beskriver ljusnivåer och de andra färgtoner i bilden. De två komponenterna med färginformation ersätts sedan av lågupplösta versioner som vid dekomprimering skalas upp.

Efter detta partitioneras de tre komponenterna till block av data som transformeras till koefficienter som representerar frekvenser. Transformerad data kvantiseras, dvs. lagras med mindre precision, där koefficienter som representerar lägre frekvenser lagras med högre precision. Det mänskliga synsystemet är mindre känsligt för förvrängningar av högfrekventa egenskaper i bilder än lågfrekventa vilket gör att detta steg inte får den återskapade bilden att se särskilt förvrängd ut. Detta förfall i precision resulterar i att snarlika värden antar lika värden vilket påverkar de statistiska egenskaperna i datan och ökar graden av kompression som kan uppnås med en entropikodek.

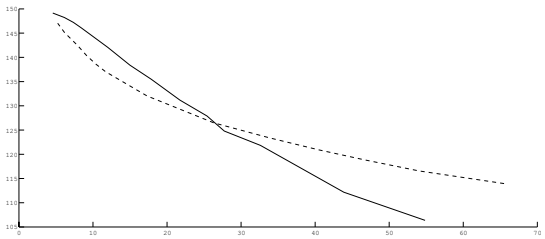
Utförande

Den utvecklade mesh colors-kodeken är lik JPEG-standarden, dock med vissa avvikelser. Mesh colors-texturer komprimeras med tillhörande MIP-nivåer där lågupplösta versioner komprimeras/dekomprimeras först. Detta utnyttjas för att uppnå en effektivare kompression av de komponenter som beskriver färgtonerna i datan. I stället för att ersätta dessa komponenter med lågupplösta versioner vid komprimering används tillgänglig MIP-data som per definition är versioner med lägre upplösning vid dekomprimering. Detta resulterar i att nedsamplade versioner av komponenter som beskriver färgtoner inte behöver lagras.

Element i mesh colors-textureringsalgoritmen som associeras med hörn är en del av 3D-modellen och behöver inte lagras med resterande data. De element som positioneras på kanter och ytor komprimeras i två olika pass där kantelement bearbetas först. Dessa partitioneras in till längder av element och transformeras endimensionellt innan kvantisering. Element som positioneras på ytor partitioneras in till tvådimensionella block och transformeras tvådimensionellt för att sedan kvantiseras. På grund av den triangulära formen av ytor resulterar partitionering av associerade element i block där diagonaler av element eventuellt kan saknas. Erhållen data kodas slutligen med en entropikodek.

Slutsats

Den implementerade mesh colors-kodeken uppnår en grad av kompression jämförbar med JPEG. Figuren nedan jämför den uppmätta kvalitén för textur i mesh colors-format komprimerad med den utvecklade mesh colors-kodeken och motsvarande textur i vanligt bildformat komprimerad med JPEG. Det kan observeras att mesh colors kodeken-presterar marginellt bättre upp till en viss kompressionsgrad och även att kvaliteten för mesh colors-data minskar snabbare än JPEG-filer för ökande kompressionsgrader. Dessa resultat visar att det går att uppnå JPEG-grad av kompression för mesh colors-data och att dess otraditionella struktur inte är ett hinder för att nå goda kompressionsresultat.



Kvalitét för komprimerad textur i mesh colors format (heldragen) och vanlig bildformat (streckad). Uppmätt kvalitét (y-axel) är plottad som en funktion av kompressionsgrad (x-axel).

User Independent Gym Exercise Recognition Using a Single Wrist Worn Accelerometer

POPULÄRVETENSKAPLIG SAMMANFATTNING
JESPER LEKLAND, ASMIR SABANOVIC

Handledare: Håkan Jonsson (Sony Mobile Communications/LTH)
Examinator: Pierre Nugues (LTH)

Our thesis takes gesture recognition, i.e interpreting human gestures via combination of hardware and software, and applies it to the ever growing world of personal fitness. We try to automatically predict what muscle groups a user is engaging at the gym, this totally without user interaction. This has been experimented on numerous times before, but none have shown very promising results. Trying to differentiate between exercises is no easy task, none has found a good way of telling when a user is actually performing an exercise and when they are resting (walking around, preparing, drinking water etc). To combat these problems, work similar to ours have used an array of several different sensors. Combinations of several accelerometers, gyroscopes, RFID, and heart rate sensors, have all been used but is considered a rather clunky set up and will be difficult to pitch to a real world user. We instead use a single, commercially available, wrist worn triaxial accelerometer and custom algorithms to recognize what muscle groups a user is engaging at the gym. With this information, we create an automated training diary in the form of an Android application. The Android application communicates with the accelerometer over Bluetooth 4.0 and is fed live accelerometer data, which it forwards to a server doing all the calculations. Extending this application even further, the user could possibly annotate their sessions, review their progress, plan feature exercise regimes and compare their workouts with friends.

To understand our presentation a little bit better, a few approaches and terms must be familiar. What is an accelerometer and what does it do? What is a classifier and how is gesture recognition performed?

A triaxial accelerometer measures acceleration in three different axes and is a fairly basic sensor, giving no additional information than straight acceleration. With a wrist worn accelerometer we recorded gym sessions off a total of 18 different subjects, performing a total of 54 different exercises during the course of one month. The subjects were followed around the gym while performing their training routines as usual. We tried to intervene as little as possible, keeping the data as true to life as possible. When a user performed an exercise, we recorded and annotated it with the correct name (label) of the exercise. Data collection is normally something so cumbersome that it is considered its own project. It often spans weeks, months or even years of work. In this thesis we both collect and analyze all of the data.

Once the data had been collected, we removed incorrect measurements, formated it into something useful (merging data from different files) and calculated what is known as ‘features‘. A feature is a single value, representing a specific attribute of the data, e.g the standard deviation, skewness, mean and similar mathematical transforms.

Once the feature were calculated, we turned to training the machine learning algorithms, also known as ‘classifiers’. These algorithms take the supplied data (our features) and tries to find connections between the values and the label (name of the exercise) of each observation. Once trained, the classifier can be used to predict what exercises future unknown data is.

Training the classifiers is much more than just providing it with the recorded data. Experiments had to be conducted to find the optimal mix of features, exercises, sample duration and classifier specific tuning parameters. This is by far the biggest challenge of machine learning, seeing as how the testing of one specific set up could take several days to complete its calculations.

The classifier for predicting exercises was the support vector machine, with a polynomial kernel. It gave an exercise classification accuracy of approximately 84% on the initial experiments, this without having to automatically detect when a user is resting or exercising. Once the rest/exercise detection were introduced the classification accuracy dropped to about 30%, in line with similar work. We then used custom algorithms and combinations of differently tuned classifiers to improve the model and finally reached a classification accuracy of 70%. These results are comparable to similar work, but excel by automatically detecting when an exercise is being performed. We also used far less sensors (only one) than similar experiments. We also glanced at removing the ‘user-independent’ aspect of the thesis, relying on the specific subject already being in the data to train the classifier. With this set up we reached even better results, having a 100% classification accuracy for 12 out of 14 sessions for the specific subject.

This thesis was done in collaboration with Sony Mobile Communications AB and the accelerometer used to capture the data was the Sony Smartband.

Ray-packet-tracing med ARM:s NEON-arkitektur

POPULÄRVETENSKAPLIG SAMMANFATTNING
GUSTAF WALDERMARSON

Handledare: Johan Grönqvist (ARM)
Examinator: Michael Doggett (LTH)

Introduktion

På senare år har datorgrafiken utvecklats enormt, framförallt på mobila enheter som smartphones och tablets. Förut kunde man knappt surfa på dem, men numera klarar dessa enheter mycket grafiskt krävande spel.

Processorn som finns i dessa enheter, ARM-processorn, kan göra betydligt mer. Den har funktioner som till exempel vektorprocessering vilket gör det möjligt att accelerera mer generella applikationer såsom ray tracing.

Just ray tracing används flitigt för att skapa realistiska bilder, men denna metod tar ofta mycket lång tid. Detta gör att varje förbättring som påskyndar metoden är mycket värdeful. I detta examensarbetet har jag utvecklat en enkel men väloptimerad ray tracing-applikation för några ARM-processorer och analyserat dess prestanda med hänsyn till både renderingstid och strömförbrukning.

Ray tracing

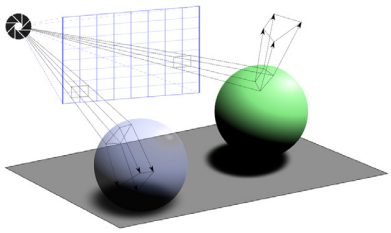
Ray tracing har länge varit den metod man använt för att skapa näst intill fotorealistiska bilder. Dessa bilder skapas genom att man följer fiktiva strålar från kamerans pixlar ut i scenen. Strålarna registrerar vilket objekt de träffat och väljer sedan antingen att använda objektets färg eller skapa en ny stråle. Den nya strålen kan till exempel ta den reflekterande riktningen för att också använda färgen från objektet därifrån. Denna metod är dessvärre oftast mycket långsam och lämpar sig ännu inte för realtids-applikationer. Trots det vill man självklart att applikationen ska vara så snabb som möjligt.

SIMD och NEON

Beräkningstunga applikationer kan ofta gå mycket snabbare om man utför samma operation på all data samtidigt. Av den anledningen utvecklades tidigt så kallade Single Instruction Multiple Data-instruktioner. Med dessa kan man samla ihop data i så kallade SIMD-vektorer, på vilka man sedan kan utföra samma operation på alla element samtidigt. I dagsläget är det vanligast att arbeta med vektorer som har fyra vektorelement. Processortillverkare kan tillhandahålla olika SIMD-teknologier med olika vektorlängder, operationer och datatyper, men addition och multiplikation på flyttals-vektorer finns i princip alltid tillgängliga. I ARM-processorer kallas dessa för Neon-instruktioner och de har tillgång till de flesta vanliga datatyper och operationer.

Ray-packet-tracing

Strålar från fyra närliggande pixlar går i många fall åt ungefär samma håll. Ofta går de samma väg genom scenen och kanske till och med träffar samma objekt. En effektiv optimering är då att paketera dessa strålar tillsammans och sedan använda SIMD-instruktioner för att göra beräkningar på alla strålar samtidigt. Just detta är tanken med så kallade ray-packet-tracers. I denna typ av ray tracer skapar man paket av strålar som är lika stora som vektorlängden för SIMD-teknologin, och följer således alla strålar i paketet samtidigt. Detta kräver i vissa fall att enstaka strålar i ett paket måste specialbehandlas, men generellt kommer de flesta strålar i paketet att träffa samma objekt vilket gör att prestandan ofta flerdubblas.

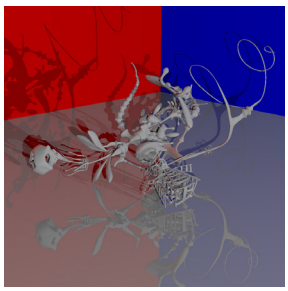


Exempel på hur ray-packet-tracing går till

Utvärdering och resultat

För att bedöma hur väl vår packet-tracer presterar har vi testat den på ett par olika ARM-plattformar och på ett urval av scener. En av scenerna vi testade vår optimerade packet-tracer på kan ses i figur 1. Renderingstiden och strömförbrukningen för denna scen på en Samsung Chromebook (XE303C12) kan ses i tabellen nedan.

	Referens	Packet-tracer
Renderingstid	32.6 s	12.6 s
Strömförbrukning	225 J	100 J



Figur 1 YeahRight-skulpturen, skapad av Keenan Crane

Slutsats

Resultat på denna scen visar att ray-packet-tracern ger en förbättring på ungefär 250% för renderingstiden jämfört med vår referens-ray-tracer, vilket i många fall mer än väl gör skäl för den större komplexiteten i en packet-tracer.

Den procentuella strömförbrukningen är något sämre än renderingtiderna men den är fortfarande ganska bra, med en förbättring på mellan 150-200% i genomsnitt i de scener vi testade.

Analyzing large data streams

POPULÄRVETENSKAPLIG SAMMANFATTNING
CARL FAGERLIN

Handledare: Jacek Malec (LTH)
Examinator: Pierre Nugues (LTH)

With modern and freely available services such as Facebook, Youtube, Twitter, Tumblr etc. we are approaching a point where we are creating data at a faster rate than we can store long-term. This creates many interesting problems: if we can't store everything, what do we store? And how do we make sense of the data we chose to store? These perpetual data streams present a problem for traditional models of evaluation. Many databases are designed with storage in mind more so than retrieval. Retrieving data can be costly in terms of resources and time. A seemingly simple database statement can take hours to complete. Even then, the produced output can be overwhelmingly vast for a human to process. Finding relevant information can be a huge challenge in itself – it becomes a proverbial search for a needle in a haystack. To take a real-world example: picture we are interested in discovering news using the Twitter data stream (messages with less than 160 characters). The data stream is staggering: on average, there are 500 million "tweets" per day, or roughly 5,700 tweets per second. We can use several different approaches:

Human intuition. Experts and experienced people in their respective field have intrinsic knowledge about what's important and what's not. An expert could identify of tweets generated by a real-world event carries significance, as well as geographical location and hashtags.

Machine learning. Using the expert knowledge of human operators a computer can be taught to look for certain patters. It still requires a human being to judge what is right or wrong; the machine will never be better than the training set used to "teach" the machine. An example of training set could teach the machine that only tweets which have been retweeted a certain amount of times are relevant.

Data mining An approach to discover previously unknown properties of the data, as was as helping bringing structure to data. Visualization is an important tool for helping people understand data; it helps us understand and makes it easier to see patterns. For example, clustering tweets and their frequency to geographical location on top of a map.

The degree project report describes two approaches to analyzing large data streams, discusses various algorithms, presents findings and discusses potential future work. After-the-fact processing can be used to process very large bulks of data quickly. For example, it can be used to filter messages according to criterion or evaluation against patterns. It's very suitable when needing to work on historical data. When real-time evaluation is not needed, cost (time and money) can be a smaller issue for after-the-fact processing. For example, Google's eigenvector calculations – the origins for the successful search engine – used to run only a few times a week (these days it's different!). For the occasions when near real-time performance is needed, a sliding window can be employed. Not the entire dataset is taken into account but instead only "windows" of different dimensions (units, time, and occurrences) residing in fast computer RAM is processed, in contrary to a traditional database which needs to retrieve everything from long-term storage. A sliding window is very suitable to filter very large data streams because of its use of pattern matching. To reiterate the Twitter example: once we known common hashtags for significant events and which types of people who make these kinds of tweets, we can adjust our window to look for the patterns we identified during our after-the-fact processing.

Implementation of a condition-based maintenance tool for critical components

POPULÄRVETENSKAPLIG SAMMANFATTNING
DENNIS ANDERSSON, ASTRID STENHOLM

Handledare: Niklas Svanberg
Examinator: Klas Nilsson (LTH)

Introduction

In all industrial machines most components eventually have to be replaced. Some of these components are more critical than others and if they fail during machine production it can lead to long downtimes and high costs. To avoid these expensive breakdowns some of these components are being replaced on fixed intervals. Thus, this can lead to fully functional components being discarded despite having several operating hours left. In between running a component until it fails, called corrective maintenance, and replacing it on fixed intervals, called preventive maintenance, is condition-based maintenance (CBM). The goal of CBM is to collect information from the machine and make decisions regarding maintenance based on this information. The optimal goal of CBM is to be able to predict failures and schedule maintenance in time to avoid breakdowns and unnecessary discarding of components. The difference of these methods can be seen in Figure 1.

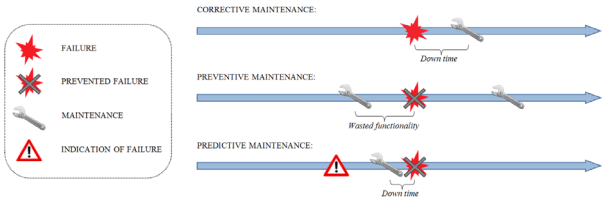


Figure 1 Schematic of different maintenance methods.

This thesis was performed at a company which is interested in implementing CBM. The thesis should serve as a prestudy with the goal of implementing customized CBM solutions for a couple of specific components. The thesis should cover the whole process and define what is needed in order to be successful.

Condition-based maintenance

A CBM tool consists of three main steps which are data acquisition, data processing and decision making. In the data acquisition step, all necessary data from the machine is collected. The data can be everything from a temperature being measured to information regarding the last repair of the machine. This data is then processed in the data processing step which turns the collected data into useful information. In the data processing step the data is first cleaned from bad samples, for example caused by a malfunctioning sensor, and then analyzed with a suitable method depending on what kind of data it is. When the data has been processed a decision should be made about if or when maintenance is needed. Decision making is divided into two subgroups; diagnostics and prognostics. In diagnostics the goal is to find if the component is in a state of failure and in that case what caused the failure. Prognostics, on the other hand, have the goal of estimating time until failure in order to schedule maintenance in time. This time estimate can be given as remaining useful life (RUL) or estimated time to failure (ETTF).

Method

In this thesis the focus has been on four different components which were selected after discussions with personnel at the company. These components all suffered from some sort of critical breakdown or were being replaced frequently. The goal was to find out if these components were suitable for CBM and what was needed to implement it. Data had already been collected from most of these components and therefore data acquisition was left out of this thesis. Different approaches were taken for the components in the data processing and decision making steps. Two of the components were fairly straightforward with only a few variables and were therefore investigated manually to find correlations between machine breakdowns and patterns in the data. For the other two components no clear failure cases existed making the analysis difficult. The third component was performing a repeating movement and therefore FFT was used to capture a complete data series in one point to be analyzed. Regarding the last component a lot of data had been collected for a large number of variables but since the failure cases were unknown a more complex method was needed. Therefore artificial intelligence was used to learn how the machine operated and automatically process the data.

Results

The goal of this thesis was to perform a prestudy of what was needed to implement a CBM tool for a couple of selected machine components. The first two components, which were analyzed manually, did not show sufficient signs of correlation between the data and failure, and therefore these were not suitable for CBM. On the component where the FFT was used clear differences could be seen between an optimal and a degenerated component even though the failure case was not established. In the case where the artificial intelligence was used it was able to automatically determine the current health status of the component. Both the FFT and artificial intelligence approach showed promising results in distinguishing between healthy and degraded components which corresponds to the diagnostic part of CBM. Since no historical data for the components were at hand it was not possible to estimate the time until failure. Thus, the prognostic part of the CBM tool was unsuccessful and requires further analysis. However, the analysis performed in this thesis, as well as the information regarding what is needed, can serve as a useful basis in the continuous work of implementing CBM.

Regressionstestning i ett konkret mjukvaruprojekt

POPULÄRVETENSKAPLIG SAMMANFATTNING
LINNEA JOHANSSON, MAGNUS WALTER

Handledare: Magnus C Ohlsson (System Verification)
Examinator: Emelie Engström (LTH)

I dagens samhälle finns mjukvara i mycket av den teknik vi använder. Det finns i bilar, hemsidor, industriella produktionssystem med mera. Här är testning ett av de viktigaste instrumenten för att säkerställa att en mjukvara verkligen gör det den är avsedd för. När en mjukvara släpps på marknaden slutar den ofta inte att utvecklas. Underhåll fortgår för att rätta till problem och nya funktioner kan komma att läggas till. Det är vid dessa tillfällen viktigt att säkerställa att den gamla funktionaliteten fortfarande fungerar som den ska. Detta kallas för *regressionstestning*. Forskning inom detta område har i stor utsträckning fokuserat på att minimera antalet testfall som behövs för att testa igenom en mjukvara. Antagandet har varit att detta är ett prioriterat problem då det kan vara tidskrävande att köra testfall upprepade gånger om dessa är många till antalet. I praktiken upplever organisationer även en rad andra problem som belysts mycket begränsat i forskningen. Dessa innefattar bland annat problem med hur testvänlig en mjukvara är, hur testfall bör designas och vilka testfall som är lämpliga att automatisera. Att studera regressionstestning ur ett bredare perspektiv kan bidra till en större förståelse för vilka sorters problem som kan uppstå i industrin.

Målet med detta examensarbete har därför varit att ta reda på vilka regressionsproblem ett konkret mjukvaruprojekt upplever. Detta har gjorts genom en fallstudie som studerat hur problem berör olika roller i en testorganisation och mellan olika typer av testning, manuell och automatiserad. Manuell testning innebär att alla steg i ett testfall utförs för hand. Automatiserad innebär att alla steg är skrivna som scriptkod som gör att en dator kan läsa och utföra testfallen själv. De studerade rollerna var testledare, designer, exekverare och automatiserare. Testledaren har ansvaret för att testningen genomförs. Designer och exekverare jobbar med manuell testning. Studien grundade sig på intervjuer där alla roller fanns representerade. Detta material har även kompletterats med enkäter från övriga anställda på företaget där intervjuerna genomfördes.

Resultat

Resultaten visade på att det finns problem med att ha en uppdelning mellan den manuella och automatiserade testningen. Detta tillskrivs den fysiska uppdelningen av de två teamen, där testledaren jobbar med det manuella teamet och därför har mindre kontakt med automatiserarna. Detta leder till att de automatiserade testarna får otillräcklig information om projektet och systemet. Risken är också att planering påverkas negativt och att problem som lyfts enbart av automatiserarna, utan stöd från testledaren, inte får samma gehör som annars.

Det framkom även att en omogen teststrategi är det problem som uppfattas som allvarligast och har störst inverkan på andra problem. En teststrategi beskriver på ett övergripande sätt hur man ska genomföra testningen. Att ha en omogen sådan innebär att de olika delarna av testningen inte är tillräckligt detaljerade eller anpassade till ett specifikt projekt kontext. Exempel på hur detta problem kan visa sig är avsaknad av ett teststrategidokument, design av testfall är inte tillräckligt specificerade, det saknas information om vilka funktioner i mjukvaran som är viktiga att testa med mera. Att åtgärda detta problem skulle ha en stor förmildrande effekt på övriga problem som identifierats i studien.

Regressiontestningen är även i stor utsträckning påverkad av mjukvarans arkitektur och hur olika komponenter är uppbyggda. Detta är ett dock mycket svårt att åtgärda eftersom mjukvaran i studien är för stor och komplex för att kunna omarbetas. Testmiljön där testningen sker har också en stark påverkan då den är instabil och delas av båda testteamen. Information kring stillestånd och underhållsaktiviteter kommuniceras sällan från dem som förvaltar miljön vilket bidrar till ytterligare tidsförluster. Detta skulle vara möjligt att åtgärda genom att samla denna information på en central websida som uppdateras löpande. Däremot är grundproblemen i testmiljön svåra att åtgärda på grund av att den ligger utanför testorganisationens inflytande.

Denna studie har visat att det finns en mängd problem som påverkar regressionstestningen kvalitet och tillförlitlighet. Även om det fanns problem med att minimera antalet testfall så visade resultaten inte att detta var det allvarligaste problemet eller det mest lämpade för åtgärder. Samtidigt var alla allvarliga problem inte aktuella för åtgärd på grund av ett lågt kostnad-nytta-värde. Problemen som identifierats i denna studie är kontextberoende; däremot är resultaten för hur ofta problem inträffar och hur allvarliga de är till stor del möjliga att generalisera till andra projekt.

Bedömningsmodell för systemintegrationslösningar

POPULÄRVETENSKAPLIG SAMMANFATTNING
MARIE VERSLAND

Handledare: Elizabeth Bjarnazon (LTH)
Examinator: Per Runeson (LTH)

Idag finns det i hela vårt samhälle datasystem som behöver integrera och kommunicera med varandra. Dagens lösningar är inte alltid optimala och vissa skulle behövas modifieras. Många företag har idag problem med sina integrationslösningar eftersom de bara har låtit dem växa utan att göra några direkta åtgärder. Det här arbetet har genomförts på ett konsultföretag som bl.a. går ut på olika företag och gör bedömningar av integrationslösningar för att sedan ge en rekommendation för hur företaget ska modifiera sin lösning. I arbetet har en modell designats som konsulterna kan använda och följa vid en sådan bedömning och på det visset göra bedömningen mer korrekt och mindre personberoende. Modellen har applicerats på en integrationslösning för att verifiera att den fungerar och för att finna förbättringar till modellen.

Systemintegration

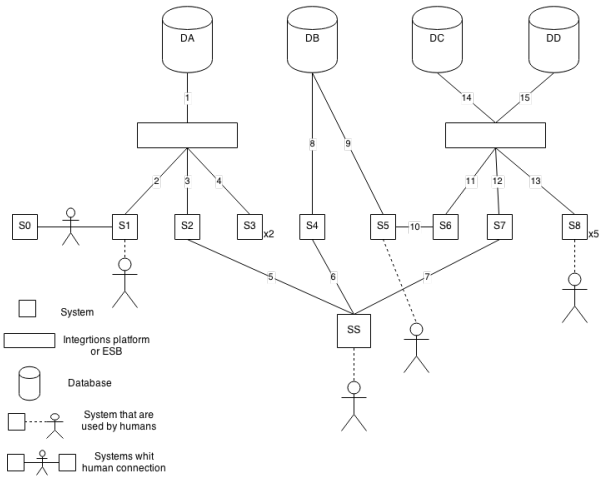
Vid systemintegration är det åtskilliga system och databaser som kommunicerar och delar data sinsemellan. Det finns åtskilliga metoder för att koppla ihop detta på. Det enklaste och snabbaste är genom point-to-point kommunikation där system är direkt kopplade till varandra. Detta ger dock ett systemlandskap som inte är flexibelt och kan vara svårt att modifiera, dessutom blir det ofta svårt att få en bra överblick när det är många system. En annan integrationsmetod är att använda en centraliserad lösning där alla systemen kommunicerar med varandra via en gemensam plattform. Detta ger ofta flexibilitet och funktionaliteten kan lättare återanvändas i de olika systemen.

Utmaningar

Det finns flera utmaningar med systemintegration och en av dem är hur data sparas. Om samma data sparas på ett flertal olika servrar blir den svår att uppdatera och det finns risk för att den blir inkonsekvent. Dataformaten är en annan utmaning, samma data har ofta olika format i olika system vilket orsakar problem när den ska sändas mellan olika system. En tredje utmaning är att många företag inte undersöker vad som redan existerar inom företaget utan bygger/skaffar nya system när de behöver något. Detta leder till att företaget kan få åtskilliga system som nästan utför samma uppgifter. Få har en total översikt av systemlandskapet inom ett företag vilket gör det svårt att underhålla och uppdatera landskapet.

Områden

Arbetet har visat att systemintegration kan fördelas i nio olika områden; översikt, arkitektur, organisation, dokumentation, data, teknologi, kvalitet, effektivitet och kostnad. Varje område kan sedan delas upp i mindre delområden. Alla de här områdena bör ingå i en bedömning. En central del när man gör en bedömning är översikten men det är oftast den som saknas på ett företag. I översikten tar man reda på vilka system som finns och hur de kommunicerar med varandra, hur systemen är kopplade och vilka verksamhetsprocesser som finns på företaget. Data är ett stort område som också är ett av det viktigaste inom systemintegration. Vilken data finns, var är den lagrad, vilket format har den, vilken storlek har datan och hur sänds den, är frågor som bör besvaras.



Överblicksbild över ett systemlandskap

Modellen

Den designade modellen består av tre överblicksbilder, alla tre beskriver olika aspekter av ett systemlandskap, de tre bilderna är; en över systemlandskapet, en över de olika dataformaten och en över de olika avdelningarna som äger systemen inom företaget. Utöver bilderna består modellen av en lista med frågor för vart och ett av de upptäckta områdena inom systemintegration. Dessa är sedan uppdelade i frågor för hela integrationslösningen och frågor för varje system. Modellen har applicerats på en integrationslösning som visar att modellen fungerar och kan generera ett relevant resultat som ger en beskrivning av systemintegrationslösningen.

Produktion: Camilla.Lekebjer@cs.lth.se • Tryck: E-husets tryckeri, Lunds universitet

LUNDS UNIVERSITET
Lunds Tekniska Högskola

cs.lth.se/examensarbete

LUNDS UNIVERSITET
Lunds Tekniska Högskola
Datavetenskap
Box 118, 221 00 LUND
Tel 046-222 80 40
cs.lth.se