KTH Computer Science
and Communication

Empirisk undersökning av programspråket Go med avseende på snabb utveckling

ALEXANDER HJALMARSSON
890216-7116
Döbelnsgatan 85
113 52 Stockholm
073-50 24 169
alehja@kth.se

JAKOB HUSS
890402-0396
Hästbäcksvägen 6a
14137 Huddinge
070-439 61 61
jakobhu@kth.se

2011-04-14
Examensarbete inom datalogi, grundnivå, DD143x
Handledare: Lars Kjelldahl
Examinator: Mads Dam

Referat

Google har nyligen lanserat språket Go som ska vara en modernt alternativ till äldre mer etablerade programmeringsspråk. På vissa punkter hävdar Google att Go är speciellt konkurrenskraftigt, exempelvis inom områden som utvecklingstid och parallella beräkningar. Dessa påståenden analyseras i rapporten tillsammans med Go som helhet. Enligt undersökningen håller Google vad de lovar och Go ser ut att vara ett lovande språk för framtiden.

Abstract

Empirical study of the Go programming language for rapid development

Google has recently introduced the Go programming language which is said to be a modern alternative to elder, more established programming languages. Google argues that Go is especially competitive in certain areas such as development time and in concurrency. Both these claims and Go in general are analysed in the report. The research concludes that Google's claims seems to be true and the future of Go looks promising.

Innehåll

Kapitel 1

Inledning

Under år 2009 lanserade Google programmeringsspråket Go. Språket designades av Robert Griesemer, Rob Pike och Ken Thompson och i slutet av år 2009 så släpptes första implementeringarna av kompilatorer officiellt till Linux och Mac OS X. Syftet med språket Go, från skaparnas sida, skall vara att täcka dagens behov av snabbt utvecklad programvara. Google har dessutom haft som mål att Go ska vara ett språk som kan utnyttja dagens hårdvarutrender och då främst underlätta vid parallell programmering på flerkärniga system.

Enligt skaparna så kombinerar språket utvecklingstiden som dynamiska språk som Python erbjuder samtidigt som det ger säkerheten hos kompilerade språk som C eller C++[1]. De hävdar även att språket är ett bra språk för systemprogrammering med ett gott stöd för parallella beräkningar.

Då språket är så nytt har det ännu inte blivit ett allmänt erkänt språk. Därför är det högaktuellt att utföra en opartisk undersökning för att se om språkets huvudsakliga mål uppfylls.

Kapitel 2

Problemformulering

Go sägs vara ett försök på att kombinera effektiv kompilering, effektiv exekvering samt enkel programmering. Google hävdar att det inte fanns något språk som täckte alla tre behov och således föddes idén till Go[2]. Rapporten kommer således att fokusera på huruvida Google har lyckats uppnå sina mål inom dessa tre områden.

För att få en bättre bedömning av hur språket Go är att utveckla i samt studera delar av dess funktionalitet har en mindre webbapplikation utvecklats. Parallellt med en applikationen i Go utvecklades en motsvarande i Python, detta för att i rapporten kunna jämföra de båda språkens inlärningströskel.

Kapitel 3

Bedömning av programmeringsspråk

Vad som är ett bra programmeringsspråk kan anses vara en personlig fråga men det finns oftast en underliggande grundtanke om vad som programmeraren uppskattar hos språket. Uday Khedker, professor i datalogi på IIT Bombay[6], har undersökt frågan och kommit fram till ett antal grundstenar som bör tänkas på för att skapa ett bra programmeringsspråk i rapporten ”What makes a good programming language?” [8].

Skriv- och läsbarhet Hur enkelt är det att läsa och förstå kod som är skriven i språket och hur enkelt är det att skriva den?

Enkelhet Hur lätt är ett att lära sig språkets funktioner och komma ihåg dem?

Syftesdefiniering Finns det ett tydligt syfte med språket eller är det tänkt att tillämpas till allt?

Ortogonalitet Syftar till att språket ska vara baserat på ett fåtal, enkla funktioner som sedan kan kombineras och resultatet ska gå att förutse. Inga specialfall bör existera.

Uttryckbarhet Hur bra är stödet för modularisering av program? Finns det stöd för att abstrahera data?

3.1 Inlärningströskel

För att mäta någonting subjektivt som en inlärningströskel krävs det att man försöker åstadkomma någonting med verktyget i fråga. För att kunna ge ett mer nyanserat svar på frågan ”Är det lätt att lära sig Go?” krävs det dessutom någonting att jämföra med. För att undersöka detta moment har författarna valt att implementera en enklare typ av dynamisk webbtjänst. Implementationen kommer att göras i språken Go samt Python. Valet Python som jämförelse är lämpligt då Go ska, enligt Google[1], vara en värdig motståndare till Python när det gäller utvecklingshastighet. För att undersöka inlärningströskeln objektivt krävs dessutom att

inga förkunskaper finns i språken. Valet Python passar således utmärkt då ingen av rapportförfattarna har tidigare erfarenheter i Go eller Python.

3.2 Enkel programmering

Med enkel programmering menas att det ska vara lätt och snabbt att utveckla kod. Detta är någonting som är subjektivt och svårt att bedöma teoretiskt. Enkelhet i ett språk har i mångt och mycket med läsbarheten att göra[9]. Programmerare som måste använda ett stort programmeringsspråk tenderar enligt Sebesta[9] att använda en delmängd av språket och ignorera resterande funktionalitet. Det finns en del nya koncept i språket Go som inte nödvändigtvis kommer att beröras under utvecklingen av webbtjänsten. Ett av de större områdena är hur man underlättar parallellprogrammering med så kallade Goroutines. Även andra funktioner i Go, som ska effektivisera programmeringen, kommer att undersökas.

3.3 Effektiv kompilering

Som tidigare nämnt är en effektiv kompilering någonting som Google har eftersträvat med programmeringsspråket Go. Undersökningen av detta kommer främst att bestå i att teoretiskt gå igenom de lösningar Google har tagit fram för att adressera problem med långvariga kompileringstider.

3.4 Effektiv exekvering

Att program ska exekveras så snabbt och så minneseffektivt som möjligt är naturligtvis någonting att eftersträva. Google påstår att Go ska sträva efter att nå samma prestanda som exempelvis C eller C++[1]. För att undersöka exekveringen bör tester inom olika områden jämföras med andra implementationer i flertalet språk. För att få ett brett spektrum på prestandatester kommer rapporten analysera tidigare etablerade resultat[4]. Detta kommer att ge en god överblick över huruvida Go exekveras snabbt och minneseffektivt.

Kapitel 4

Analys av språket

Analysen av språket består av två delar. En teoretisk där språkets hörnstenar beskrivs och vilka delar som språket vill utmärka sig med. Den praktiska delen består av att göra en enklare typ av webbtjänst för att utvärdera om språkets funktioner uppfyller de funktioner som söks hos en utvecklare när kort utvecklingstid önskas.

4.1 Teoretisk analys

Go är ett relativt litet programspråk med ett tydligt fokus på att få programvaruutveckling att gå snabbare. Språket använder skräpsamling och har som mål att vara snabbt med ett bra stöd för parallella beräkningar. Enligt skaparna av språket så finns det några huvudsakliga anledningar till att de ansåg att ett nytt språk behövdes på marknaden[3]:

- Datorer har blivit snabbare men programvaruutveckling har inte blivit det.
- Dynamiska språk som till exempel Python och Javascript har stigit i popularitet gäntemot typsäkra programspråk som Java och C++.
- Skräpsamling och samtidiga beräkningar stöds dåligt av populära systemspråk.

4.1.1 Kompilering

Go är ett språk som kräver kompilering innan körning. För att få programspråket mer ämnat för snabb programvaruutveckling har utvecklarna lagt relativt stort fokus till att göra en snabb kompilator. Kompilatorerna finns för arkitekturerna amd64 och 386 samt i skrivande stund begränsat stöd för arm.

4.1.2 Syntax

Go är ett språk som kräver kompilering innan körning. För att få programspråket mer ämnat för snabb programvaruutveckling har utvecklarna lagt relativt stort fokus

till att göra en snabb kompilator. I en demonstration av Rob Pike[10] visar han hur hela Gos källkod kan kompileras på strax över åtta sekunder. Kompilatorerna finns för arkitekturerna amd64 och 386 samt i skrivande stund begränsat stöd för arm.

Att Go är lätt att analysera innebär att det finns vissa krav på syntaxen. Dessa krav bidrar i sin tur till snabbare kompileringstider då det finns färre sätt att uttrycka samma sak på. En konsekvens av detta blir att grammatiken har blivit anpassad och får således ett speciellt utseende. Google hävdar själva att detta speciella utseende kommer även få den positiva effekten att alla program skrivna i Go kommer få väldigt snarlika utseenden[7]. Det finns med andra ord väldigt lite utrymme för programmerare att applicera sina egna preferenser på koden.

Syntaxen i Go går att känna igen ifrån C men det finns undantag som är värda att nämna. Variabeltyper definieras efter variabelnamn vilket ska göra den viktigaste information mer lättläst vid läsning från vänster till höger. En till skillnad jämfört med C är skillnaden mellan dessa två uttryck där det i C definieras:

```
int* a,b;
```

där a är en pekare men inte b. Go har ändrat på detta och har det lite mer intuitiva beteendet där:

```
var a,b *int;
```

betyder att både a och b pekar på typen int. Semikolon är inte obligatorisk i syntaxen vilket gör att ovanstående även kan skrivas som:

```
var a,b *int
```

4.1.3 Parallella beräkningar

För att hantera parallella beräkningar så har Go introducerat ett nytt begrepp kallat ”Goroutines”[12]. Valet att introducera ett nytt begrepp har gjorts då termer som tråd eller process har olika betydelser i olika kontext. Dessa rutiner gör att parallella beräkningar flyttas till separata operativsystemtrådar för att inte blockera varandra. Rutinerna är anpassade för att uppta så lite resurser som möjligt och göra det lätt att programmera effektivt på moderna datorer.

Goroutines har blivit konstruerad på ett sådant sätt att de ska vara mycket lätta att använda. Målet är att man som programmerare inte ska tveka inför att använda Goroutines. Ett exempel, finns nedan, på hur enkelt man startar en Goroutine hämtat från Googles webbsida om effective Go[11].

```
go list.Sort()
```

I exemplet ovan kommer listan sorteras i sin egna Goroutine. När sorteringen är färdigställd kommer Goroutinen avslutas automatiskt. Paralleller för hur rutinen körs och avlutas dras i effective Go[11] till hur tecknet ”&” används i Unixterminalen.

Kanaler

Go har infört ett system för kommunikation mellan trådar mycket likt pipelines i Unixsystem. Med hjälp av dessa går det på ett snabbt och effektivt sätt att programmera exempelvis parallella beräkningar. När en beräkning är färdig kan den via kanalen rapportera det och på så vis slutföra beräkningen eller fortsätta, beroende på vad som önskas. I exemplet nedan så tas en vektor av heltal och delar upp beräkningen till 8 stycken kärnor. När beräkningarna skickats med hjälp av Gorutinen så väntar programmet på att samtliga kanaler ska bli färdiga.

Kod 4.1: Exempel på parallell programmering i Go

```
type IntVector []int
// Beräkna från v[i], v[i+1] till v[n-1]
func (v IntVector) PartCalc(i, n int, c chan int) {
    for ; i < n; i++ {
        v[i] = calc(v[i])
    }
    c <- 1    // signal that this piece is done
}
const CPUCOUNT = 8  // Antal CPU-kärnor
func FullCalc(u IntVector) {
    c := make(chan int, CPUCOUNT)  // Ska en kanal med storlek 8.
    for i := 0; i < CPUCOUNT; i++ {
        go u.PartCalc(i*len(u)/CPUCOUNT, (i+1)*len(u)/CPUCOUNT, c)
    }
    for i := 0; i < CPUCOUNT; i++ {
        <-c    // Väntar på att en PartCalc ska att bli klar
    }
    // Allt är beräknat.
}
```

Tekniken gör att det snabbt, enkelt och med minimal ansträngning går att skapa beräkningar som utförs parallellt.

4.1.4 Gränssnitt

Grundarna till Go har implementerat gränssnitt på ett aningen annorlunda sätt jämfört med exempelvis Java. Gränssnitten appliceras implicit, vilket i praktiken betyder att om du skapar ett gränssnitt:

Kod 4.2: Interface i Go

```
type Absolute interface {
Abs() float
}
```

Då kommer detta ”Absolute”-gränssnitt att automatiskt implementeras av alla strukturer som har en metod ”Abs()” som returnerar ett flyttal. Det betyder att programmeraren inte behöver lägga tid på att lägga till något till de existerande strukturerna såvida de har metoden.

4.2 Praktisk analys och implementation

För att få en sann inblick i något programmeringsspråk står det klart att språket måste användas i praktiken. Målet med att implementera någonting är främst att få en uppfattning om hur inlärningingströskeln ser ut, det vill säga hur lätt eller svårt det är att komma igång med språket i fråga.

Valet att implementera en webbtjänst grundar sig på att Go sägs vara praktiskt då utvecklingen av produkten skall ske snabbt[1]. Det passar utmärkt för webben då denna är i ständig förändring och därmed krävs det ett programmeringsspråk som kan hålla jämna steg i denna snabba utveckling.

Webbtjänsten som implementeras är en förenklad klon av en social bokmärkningstjänst, likt reddit[5], där användare skickar in egna nyheter. Om man finner en nyhet speciellt intressant eller överflödig kan man som användare rösta på den för att placera nyheten högre upp respektive längre ned på förstasidan. På så sätt hamnar de mest, för tillfället, intressanta nyheterna alltid högst upp och ger andra användare lätt åtkomst till de viktigaste nyhetsuppdateringarna.

Implementationerna som gjorts resulterar inte i en färdig produkt utan syftar till att ge en fingervisning av funktionalitet och potentiella problem med språket. För att kunna ha någon referens till Go-implementationen så har samma tjänst programmerats i Python. Koden som skrivits är inte på något sätt optimerad för prestanda.

4.2.1 Implementation i Go

Go är ett litet språk som erbjuder spännande möjligheter för utvecklaren. Språket har en annorlunda syn på objektorientering och hierarkier av typer existerar inte. Go utnyttjar en typ av gränssnitt som låter kompilatorn granska om en aktuell typ implementerar de aktuella gränssnitten. På så vis behöver en struktur inte definiera att den implementerar några specifika gränssnitt, utan det räcker för strukturen att implementera ha de funktioner som krävs.

I Go är det möjligt att returnera flera värden vilket gör att oväntade resultat kan upptäckas på ett elegant och praktiskt sätt. Samma funktionalitet finns i Python men saknas till exempel i C, där det är vanligt att fel visas med ett heltalet -1.

Kod 4.3: Exempel på returnering i Go

```
// Öppnar en fil och returnerar hanteraren f.
// Om ett feluppstått är den andra variabeln,
// err, skiljd från nil, varpå en tom sträng
// skickas vidare tillsammans med variabeln
// err
f, err := os.Open(filename, os.O_RDONLY, 0)
if err != nil {
    return "", err
}
```

Figur 4.1: Den sociala bokmärkningstjänsten implementerad i Go och visad i Firefox.

Resultatet av implementationen i Go syns i figur 4.1. Den jämförande Python-implementationen ser i stort sett identisk ut.

Utvecklingsverktyg

Då Go är ett ungt språk så finns det ännu inte några välutvecklade integrerade utvecklingsmiljöer. Utvecklingsteamet för GoEclipse[13] har tillverkat en plugin för Eclipse som är ett gott steg framåt men utgåvan som testades till rapporten (alpha 0.0.19) var aningen ostabil. Det finns en rad med syntaxmarkerare som förenklar programmeringen i vanliga textredigare, något som räcker till för många utvecklare.

Standardbibliotek

Standardbiblioteket i Go inkluderar praktiska och användbara typer och metoder för webbprogrammering. Istället för att blanda html tillsammans med programkod så går det att inkludera mallpaketet för att separera kod och design till egna filer. På så sätt blir koden mer överskådlig och designändringarna lättare att göra.

Biblioteket erbjuder även en webbserver som gör det enkelt att lokalt testa och skriva applikationer.

4.2.2 Implementation i Python

Python är till skillnad från Go ett interpreterat språk, vilket tar bort kompileringstiden och ger intrycket av att vara snabbare att komma igång med.

Då Python är ett etablerat språk finns en uppsjö av olika ramverk för att förenkla utvecklandet. Det ramverk som använts i implementationen är webpy[14]. Webpy erbjuder en webbserver såväl som mallklasser vilket motsvarar Gos standardbibliotek.

Pythonprogrammerare har ett flertal integrerade utvecklingsmiljöer att välja mellan. Plugins finns till både NetBeans[15] och Eclipse[16] som stöder båda refaktorisering och kodkomplettering.

Det finns mycket dokumentation för Pythonprogrammering och språket används mycket till just webbapplikationer.

4.2.3 Prestanda

För att analysera ett programmeringsspråks prestanda krävs det ett flertal tester. Faktum är att utföra den stora mängd jämförelser med andra språk som skulle krävas för ett pålitligt prestandatest är utanför omfattningen av denna rapport. Valet att analysera redan en etablerad källa för prestandatester ter då sig som ett utmärkt alternativ.

The Computer Language Benchmarks Game[4] är en webbplats som opartiskt jämför prestanda mellan olika programmeringsspråk. Hemsidan fungerar på så sätt att den som vill, kan skicka in en implementation på något av de programmeringsproblem som finns definierade på hemsidan. Om det visar sig att den nya implementationen är effektivare än den tidigare rekordhållarens lösning kommer denna att publiceras som den nya ledaren. Detta tävlingsmoment bör ge en god uppskattning av prestanda då källkoden är öppen och tävlingen ger intresse för att hitta nya snabbare lösningar. Tack vare denna hemsida finns möjligheten att jämföra ett flertal programmeringsspråk med varandra i flera olika intressanta tester.

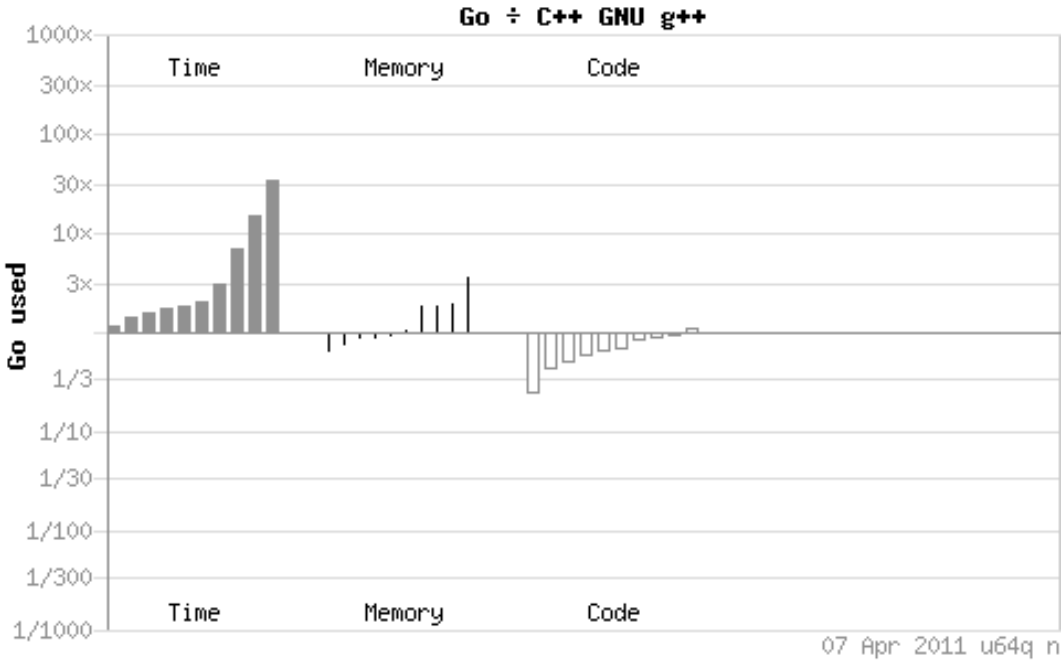
Material

Valet av vilken maskin som prestandatesterna ska ha utförts på är tämligen enkelt. Go ska, som tidigare nämnts, vara smidigt vid parallella beräkningar således kommer prestandatester studeras på en maskin med flera processor kärnor. Nämligen en maskin med x64 Ubuntu, Intel® Q6600® quad-core. Jämförelsen i prestanda kommer ske mellan Go, C++, Java samt Python av det skälet att Go kan främst vara ett alternativ till dessa språk. De tester[4] som har genomförts är Fannkuch-redux, Fasta, N-body, K-nucleotide, Spectral-norm, Pidigits, Reverse-complement, Mandelbrot, Binary-trees, Regex-dna.

Förklaring av diagram: Varje test som har utförts har genererat en stapel i diagrammen. En negativ stapel (riktad nedåt) innebär att Go var bättre på den punkten. De kriterier som jämförs är tidsåtgång, minneskonsumtion samt hur mycket kod programmet bestod av. För numerisk jämförelse se bilaga C.

Go jämfört med C++

Som synes förbrukar C++ mindre tid i alla tester men det är i få fall en markant skillnad. Minneskonsumtions delen av diagrammet visar att Go är i somliga fall bättre och i andra fall sämre, skillnaden är inte speciellt stor i något fall. Intressant att notera är att mängden kod som krävdes för att skriva programmen är i alla utom ett fall mindre i Go.



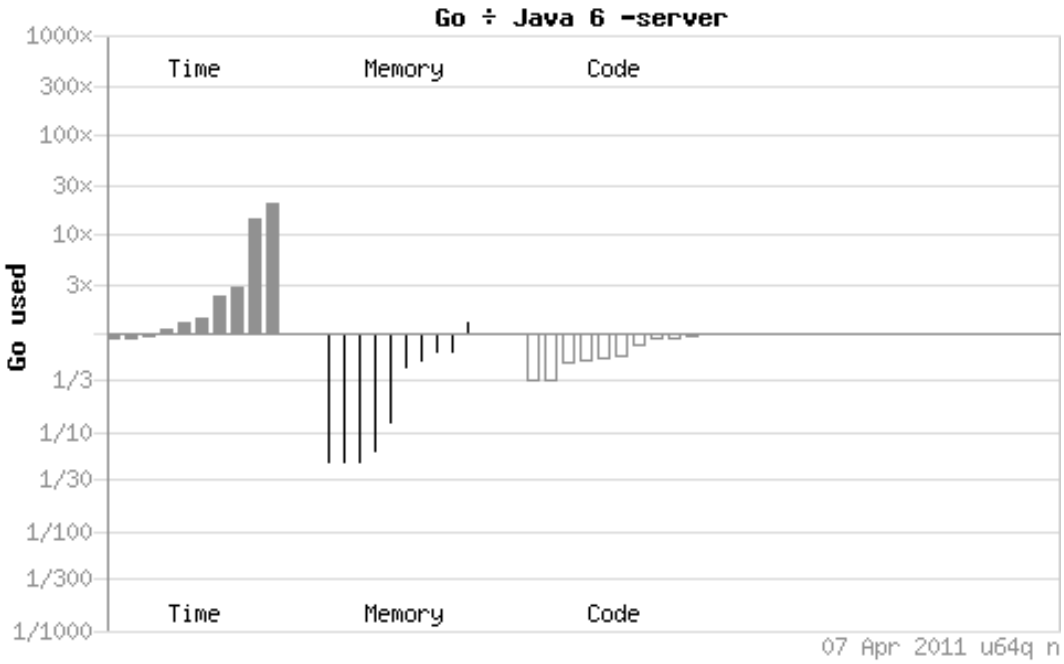
Figur 4.2: Diagram av jämförelse mellan Go och C++

Go jämfört med Java

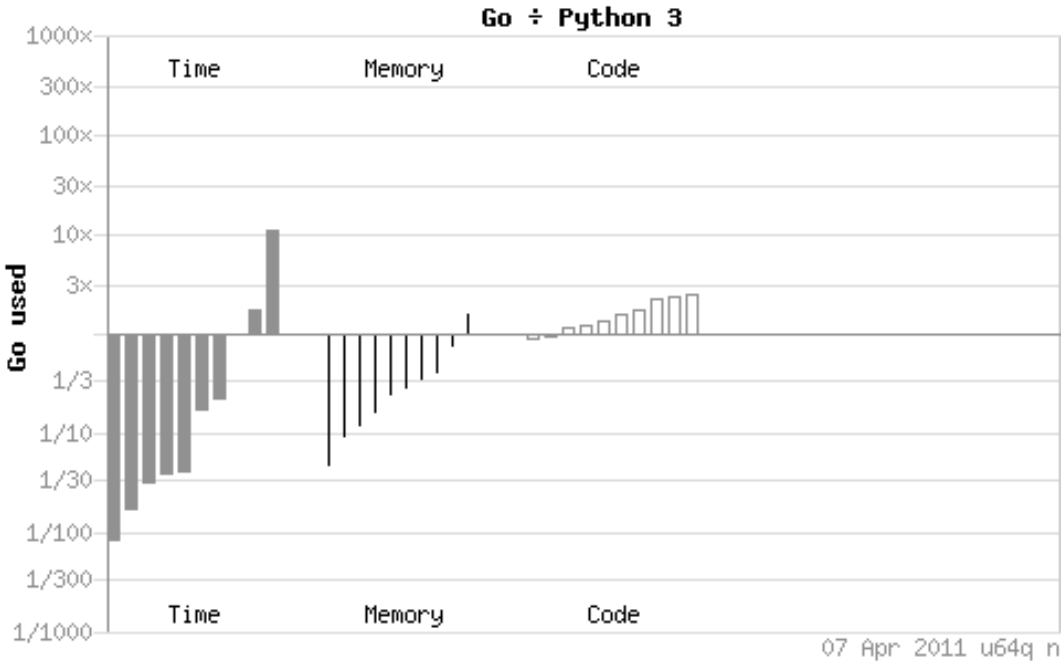
När det gäller exekveringstiden har Java i flera av testerna ett försprång. Men i de flesta fall är skillnaden inte speciellt markant. Minnesåtgången är däremot intressantare, här uppvisar Go mycket bra resultat. I somliga fall kan Go använda så lite som nästan 1/30 av den mängd minne som Java använder sig av i samma test. Precis som i jämförelsen med C++ krävs det färre rader kod för att utföra samma uppgift. Så lite som en tredjedel av kodmassa krävdes i somliga fall.

Go jämfört med Python

I detta test kan man se att det är ombytta roller jämfört med de andra språken. Go förbrukar oerhört mycket mindre tid och minne i så gott som alla tester. Men däremot krävdes det fler rader kod för att implementera testet.



Figur 4.3: Diagram av jämförelse mellan Go och Java



Figur 4.4: Diagram av jämförelse mellan Go och Python

Kapitel 5

Diskussion

5.0.4 Inlärningströskel

När Go skapades hade det som syfte att ersätta delar av programmering i C för att snabba upp utvecklingsprocessen i systemprogrammering. Det visade sig senare att programmet var lämpligt för andra tillämpningar också, vilket medförde att bland annat webbutvecklare öppnade ögonen för språket[17]. Det är i sig inte ett problem att språket visat sig vara mer användbart än väntat men då en av de egenskaperna som gör ett bra programmeringsspråk[8] är hur väl syftet med språket är definierat. Det kan resultera i att språket inte får något specifikt användningsområde där det är användbart.

Då Go är så pass litet och har få nyckelord jämfört med de flesta konkurrenterna[18] gör det att det går relativt snabbt att lära sig dessa nyckelord och det resulterar i detta fall att språket blir enklare att lära sig. Den aningen annorlunda implementationen av gränssnitt bidrar även det till att förenklande och gränssnitten gör det onödigt att explicit deklarera vilka gränssnitt som implementeras. På så vis så slipper utvecklare skriva ”onödig” kod och tiden kan istället läggas på annan funktionalitet.

Genom utvecklingen av webbapplikationen så kan slutsatsen dras att Go är ett språk som det går snabbt att komma igång med och har egenskaper som gör det praktiskt att programmera med. Om språket har tillräcklig funktionalitet och prestanda för att göra mera komplexa applikationer och system återstår att se i framtiden då inget av detta kan besvaras utifrån undersökningen.

Jämfört med Python och med bidragande faktorer som dokumentation, guider, exempelprogram och liknande så ligger Python steget före vad gäller inlärningskurva och snabbhet att komma igång. För en utvecklare som kommer ifrån C eller C++, där syntaxen är lik och kompilering av kod redan är av vana kan dock språket ses som en mer naturlig övergång än till Python.

5.0.5 Enkel programmering

Go ett språk som har en tydlig inriktning på att det ska vara enkelt att programmera. Det syns genom användningen av få nyckelord, enkla rutiner för parallella beräkningar och en förenklad syn på objektorienting med gränssnitt. Risken med att göra ett allt för simpelt programmeringsspråk är, som Sebesta beskriver i "Concepts of programming languages"[9], att läsbarheten i språket kan gå förlorat. Ett exempel är Assembly, där språket är språket i sig är enkelt men det tenderar att vara svårare att läsa än mer komplexa språk. Utvecklarna till stor del lyckats med att behålla god läsbarhet genom goda syntaktiska egenskaper men bibehåller ändå möjligheterna för programmerarna att göra det som önskas.

För att ett språk ska vara snabbt att utveckla i kan ett av önskemålen vara att mindre rader kod ska användas för att göra någon specifik uppgift. I de prestandatester som refererats till i rapporten ses det generellt att Python kräver aningen mindre kod för att utföra samma uppgifter medan C++ och Java kräver mer.

5.0.6 Effektiv kompilering

Google har lagt ned mycket energi på att få kompileringen att gå snabbt. Onekligen har de lyckats mycket väl med detta. Att kompilera alla Gos bibliotek tar enbart några få sekunder[10]. Detta innebär naturligtvis att man som programmerare inte behöva spendera speciell mycket tid väntande på att koden kompileras. En programmerare som inte behöver vänta på kompilering kommer att få mer tid över till att faktiskt programmera. Dessutom kommer personen inte tappa fokus på vad denne håller på med, något som är lätt hänt vid längre avbrott.

Snabb kompilering ger också den fördelen att koden inte nödvändigtvis behöver kompileras på externa kraftfulla servrar. Alltså kan den snabba kompileringen spara in pengar då man kan i vissa fall undvika att investera i dedikerade kompileringsservrar.

5.0.7 Effektiv exekvering

Ett av målen utstakade av Google var att Go skulle prestera i klass med andra stora kompilerade språk. Tar man sig en närmare titt på prestandajämförelsen när det gäller tidsåtgång inses man att Go är något långsammare än C++ och Java. Skillnaden är inte stor men den finns där. Varför Go inte har nått hela vägen fram till på det här området är flera. För det första ska man veta att implementationerna av testerna[4] nödvändigtvis inte är de bästa. Vid närmare granskning verkar de vara "direktöversatta" från Java-implementationerna och möjligen utnyttjar inte Gos fulla potential. Man kan också tillägga att Gos kompilator är, i skrivande stund, enbart två år gammal. Det finns utrymme för förbättringar[19] i kompilatorn som kan resultera i bättre prestanda. Jämför man däremot med det interpreterade språket Python står Go sig mycket starkt.

På det hela taget är Go ett snabbt programmeringsspråk och när språket och när dess bibliotek har hunnit mogna lite till bör det kunna komma ytterligare prestan-

daökningar. Att språket är så pass snabbt som det är gör att man inte behöver dra sig för att använda det som ersättare för C++ eller Java med tanke på prestanda. Tvärtom kan man studera andra skillnader mellan språken innan man väljer vilket språk man skriver i. I jämförelsen mellan Go och Python visar sig det sig att Go ofta presterar många gånger bättre. Alltså bör Go vara ett attraktivt alternativ till Python i projekt där prestanda är viktig. Python programmerare har anammat Go i större utsträckning än Google hade förutspått[17]. Detta beror säkerligen på den ökade prestanda som Go erbjuder tidigare Python programmerare.

Kapitel 6

Slutsats

Rapporten undersökning kommer fram till att Go är ett språk som lyckats väl med de mål som varit definierade. Kompilatorn är snabb, språket är litet och stödet för parallella beräkningar har förenklats jämfört med konkurrerande språk som C, C++ och Python. I mångt och mycket är Go, trots sin unga ålder, ett bra språk för snabb systemprogrammering och till viss del webbutveckling då prestanda är en viktig faktor.

Baserat på undersökningen i rapporten så kan slutsatsen dras att sig att Go är ett språk där utvecklingstiden kan förkortas jämfört med C och C++. Beroende på vilken mjukvara som ska utvecklas kan det också vara en attraktiv konkurrent till språk som exempelvis Python, vars fokus redan ligger på utvecklingstid och enkelhet snarare än prestanda.

Att Google använder språket internt spelar en viktig roll i språkets framtid. På så vis finns det ett utvecklande företag bakom produkten vilket förhoppningsvis kommer bidra till framtida resurser vad gäller utveckling av språket.

Litteraturförteckning

[1] Google Open source blog, 2011-03-08, http://google-opensource.blogspot.com/2009/11/hey-ho-lets-go.html

[2] Golang FAQ - Creating a new language, 2011-03-08, http://golang.org/doc/go_faq.html#creating_a_new_language

[3] Golang FAQ - What is the purpose of the project, 2011-03-08, http://golang.org/doc/go_faq.html#What_is_the_purpose_of_the_project

[4] Computer Benchmark Game, 2011-03-08, http://shootout.alioth.debian.org/

[5] Reddit 2011-03-08, http://www.reddit.com

[6] Bio-Data, 2011-04-14, http://www.cse.iitb.ac.in/~uday/

[7] Golang FAQ - Semicolons, 2011-04-14, http://golang.org/doc/go_faq.html#semicolons

[8] Uday Khedker, 1997, "What Makes A Good Programming Language?", http://www.cse.iitb.ac.in/~uday/soft-copies/ppl.ps, Okt 1997, 2011-04-14

[9] Robert W. Sebesta, 2001, "Concepts of Programming Languages", Addison Wesley Publishing Company, 5e utgåvan

[10] Rob Pike, "The go programming languages", http://www.youtube.com/watch?v=rKnDgT73v8s, 2009-11-10, 2011-04-14

[11] Effective Go, http://golang.org/doc/effective_go.html#concurrency, 2011-04-14

[12] Russ Cox, "Parallel Programming Talk #60 - Russ Cox about the Google Go programming language", http://software.intel.com/en-us/blogs/2010/01/21/parallel-programming-talk-60-russ-cox-about-the-google-go-programming-language/, 2010-01-21, 2011-04-14

[13] goclipse - Eclipse based IDE for Google Go, http://code.google.com/p/goclipse/, 2011-04-14

[14] WebPy, http://webpy.org/, 2011-04-14

[15] Python plugin for NetBeans, http://wiki.netbeans.org/Python, 2011-04-14

[16] PyDev, http://pydev.org/

[17] Google Programming Language Blog, ”Go: One year ago today”, http://blog.golang.org/2010/11/go-one-year-ago-today.html, 2010-11-10, 2011-04-14

[18] Rob Pike, ”Expressiveness of Go”, http://golang.org/doc/ExpressivenessOfGo.pdf, 2010-10-5, 2011-04-14

[19] Golang FAQ - Performance, http://golang.org/doc/go_faq.html#Why_does_Go_perform_badly_on_benchmark_x, 2011-04-14

Bilaga A

Källkod från Go-applikation

Kod A.1: Källkoden till Go-applikationen

```
package main

import (
        "http"
        "template"
        "strconv"
        "sort"
        "time"
        "fmt"
)
var LinkedAr = new(LinkArray)
var UID_count = 2;
var startTime = time.Seconds()

const MainPage = "main"

type Link struct {
        Title string
        Url string
        Score int
        UID int
        TimeAgo string
        Time int64
}
func (l *Link) save() {
        LinkedAr.data = append(LinkedAr.data,*l)
}
type MainHolder struct {
        Title string
        Content string
        Posts []Link
        PostsDate []Link
}

type LinkArray struct {
```

```
        data []Link
        SortType string
}

func (p LinkArray) Len() int            { return len(p.data) }
func (p LinkArray) Less(i, j int) bool  {
        if(p.SortType != "time") {
                return p.data[i].Score > p.data[j].Score
        }
        return p.data[i].Time > p.data[j].Time

}
func (p LinkArray) Swap(i, j int)       { p.data[i], p.data[j] = p
    .data[j], p.data[i] }

func updateTimeAgo() {
        for i:= range LinkedAr.data {
                ti := time.Seconds() - LinkedAr.data[i].Time
                if(ti <= 60) {
                        LinkedAr.data[i].TimeAgo = strconv.Itoa64(
                            ti) + " second(s) ago"
                } else if(ti > 60 && ti <= 3600) {
                        LinkedAr.data[i].TimeAgo = strconv.Itoa64(
                            ti/60) + " minute(s) ago"
                } else {
                        LinkedAr.data[i].TimeAgo = "> 1 hour ago"
                }
        }
}
func getCurrentTime() int64 {
        return time.Seconds()
}
func newHandler(w http.ResponseWriter, r *http.Request) {
        url := r.FormValue("Url")
        title := r.FormValue("Title")
        p := &Link{Title: title, Url: url, UID: UID_count,Time:
            getCurrentTime()}
        UID_count++;
        if(url == "" || title == "") {
                mainHandler(w,r,"Empty fields in form")
                return
        }
        p.save()
        http.Redirect(w, r, "/"+MainPage+"/", http.StatusFound)
}
func vote(uid, inc int) {
        for i := range LinkedAr.data {
            if(LinkedAr.data[i].UID == uid) {
                if(!(LinkedAr.data[i].Score <= 0 && inc == -1)) {
                        LinkedAr.data[i].Score += inc;
                }

            break;
            }
```

```
        }
        sort.Sort(LinkedAr)
}
var templates = make(map[string]*template.Template)

func init() {
        for _, tmpl := range []string{MainPage} {
                templates[tmpl] = template.MustParseFile(tmpl+".
                    html", nil)
        }
}
func mainHandler(w http.ResponseWriter, r *http.Request, content
    string) {
        updateTimeAgo()
        LinkedAr.SortType = "time"
        sort.Sort(*LinkedAr)
        newSlice := make([]Link,len(LinkedAr.data))
        copy(newSlice,LinkedAr.data)
        LinkedAr.SortType = "score"
        sort.Sort(*LinkedAr)
  pa := &MainHolder{Title: "Simple social bookmarking implemented
      in Go", Content:content, Posts: LinkedAr.data, PostsDate:
      newSlice}
        err := templates[MainPage].Execute(w, pa)
        if err != nil {
                http.Error(w, err.String(), http.
                    StatusInternalServerError)
        }
}
func voteHandler(w http.ResponseWriter, r *http.Request) {
        qmap, err := http.ParseQuery(r.URL.RawQuery)
        val, err2 := strconv.Atoi(qmap["val"][0])
        uid, err3 := strconv.Atoi(qmap["uid"][0]);
        if(err != nil || err2 != nil || err3 != nil) {
              http.Redirect(w, r, "/"+MainPage+"/", http.
                  StatusFound)
        }
        if(val == 1) {
                vote(uid,1)
        } else if(val == -1){
                vote(uid,-1)
        }
        http.Redirect(w, r, "/"+MainPage+"/", http.StatusFound)
}
func makeMainHandler(fn func(http.ResponseWriter, *http.Request,
    string)) http.HandlerFunc {
        return func(w http.ResponseWriter, r *http.Request) {
                content := ""
                fn(w, r, content)
        }
}

const lenPathStatic = len("/static/")
func staticFilesHandler(w http.ResponseWriter,r *http.Request) {
```

```
            file := r.URL.Path[lenPathStatic:]
            fmt.Printf(file)
            http.ServeFile(w, r, "static/"+file)
    }
    func main() {
            http.HandleFunc("/main/", makeMainHandler(mainHandler))
            http.HandleFunc("/vote/", voteHandler)
            http.HandleFunc("/new/", newHandler)
            http.HandleFunc("/static/", staticFilesHandler)
            http.ListenAndServe(":8080", nil)
    }
```

Kod A.2: HTML-mall för Go-applikationen

```
<head>
<link rel="stylesheet" type="text/css" href="/static/style.css" />
</head>
<body>
<div class="wh">
<h1>{Title}</h1>

<div style="margin-left:25px;">
<form action="/new/" method="POST">
<span>
<p>{Content}</p>
Add a new entry:<br/>
Title: <input name="Title" value="" type="text" />
Url: <input name="Url" value="" type="text" />
<input type="submit" value="Post">
</span>
</form>
</div>
<div class="cont">
<h2>Highest ranked links</h2>
<ol>
{.repeated section Posts}

    <li><a href="{Url}">{Title}</a><span style="float:right"><a
        href="/vote/?uid={UID}&val=1"><img src="/static/thumb-up.
        png"></img></a>  <a href="/vote/?uid={UID}&val=-1"><img src
        ="/static/thumb-down.png"></img></a> (+{Score})  </span><br
         />Added {TimeAgo}
    </li>
{.end}
</ol>
</div>
<div class="cont">
<h2>Most recently added</h2>
<ol>
{.repeated section PostsDate}
    <li class="li2"><a href="{Url}">{Title}</a><span style="float:
        right"><a href="/vote/?uid={UID}&val=1"><img src="/static/
        thumb-up.png"></img></a>  <a href="/vote/?uid={UID}&val=-1"
        ><img src="/static/thumb-down.png"></img></a> (+{Score})  <
        /span><br />Added {TimeAgo}
    </li>
{.end}
</ol>
</div>
<br style="clear:both" />
</div>
```

Bilaga B

Källkod från Python-applikation

Kod B.1: Källkoden till Python-applikationen

```
import web
from web import form
import datetime
import time
urls = ('/dyn/uid=(.*)&inc=(.*)','vote','/dyn', 'dyn')
app = web.application(urls,globals())
render = web.template.render('templates/')
class Link:
  def __init__(self,title,url,score,uid):
    global starttime
    self.title = title;
    self.url = url;
    self.score = score;
    self.uid = uid;
    self.time = time.mktime(datetime.datetime.fromtimestamp(time.
        time()).timetuple())
    self.timeAgo = 0;
def curTime():
  return time.mktime(datetime.datetime.fromtimestamp(time.time()).
      timetuple())
def updateTimeAgo(linkarray):
  for i in linkarray:
    i.timeAgo = curTime() - i.time
    if(i.timeAgo <= 60):
      i.timeAgo = str(int(i.timeAgo)) + " second(s) ago"
    elif(i.timeAgo > 60 and i.timeAgo < 3600):
      i.timeAgo = str(int(i.timeAgo / 60)) + " minute(s) ago"
    else:
      i.timeAgo = "> 1 hour ago"
  return
a = Link("title1","url1", 0,0)
b = Link("title2","url2", 0,1)
count = 2;
linkarray = []
linkarray.append(a)
```

```
    linkarray.append(b)

    addLinkForm = form.Form(
      form.Textbox('title',
                    form.notnull, description='Enter title:'
                    ),
        form.Textbox('url',
                    form.notnull, description='Enter URL:'
                    )
    )
    class dyn:
      def GET(self):
        updateTimeAgo(linkarray)
        sortedByDate = sorted(linkarray, key=lambda link: link.time,
            reverse=True)
        return render.dyn("title",addLinkForm(),"content",linkarray,
            sortedByDate)
      def POST(self):
        lForm = addLinkForm()
        if not lForm.validates():
          updateTimeAgo(linkarray)
          sortedByDate = sorted(linkarray, key=lambda link: link.time,
              reverse=True)
          return render.dyn("title",lForm(),"content",linkarray,
              sortedByDate)
        else:
          title = lForm['title'].value
          url = lForm['url'].value
          global count
          linkarray.append(Link(title,url,0,count))
          count+=1
          updateTimeAgo(linkarray)
          sortedByDate = sorted(linkarray, key=lambda link: link.time,
              reverse=True)
          return render.dyn("title",addLinkForm(),"content",linkarray,
              sortedByDate)
    class vote:
      def GET(self,uid,inc):
        for i in linkarray:
          if(i.uid == int(uid)):
            if(inc == "true"):
              i.score +=1
            elif(inc == "false"):
              if not i.score <= 0:
                i.score -=1
            break

        linkarray.sort(key=lambda link: link.score,reverse=True)
        raise web.seeother("/dyn")
    if __name__ == "__main__":
      app.run()
```

Kod B.2: HTML-mall för Python-applikationen

```
$def with (title,form,content,Links,dateLinks)
<head>
<link rel="stylesheet" type="text/css" href="/static/style.css" />
</head>
<body>
<div class="wh">
<h1>$title</h1>

<div style="margin-left:25px;">
<form method="POST">
<span>
<p>$content</p>
Add a new entry:<br/>
$if not form.valid: <p>Input is invalid.</p>
$:form.render()
<input type="submit">
</span>
</form>
</div>
<div class="cont">
<h2>Highest ranked links</h2>
<ol>
$for link in Links:
    <li><a href="$link.url">$link.title</a><span style="float:
        right"><a href="/dyn/uid=$link.uid&inc=true"><img src="/
        static/thumb-up.png"></img></a>  <a href="/dyn/uid=$link.
        uid&inc=false"><img src="/static/thumb-down.png"></img></a>
         (+$link.score)  </span><br />Added $link.timeAgo
    </li>

</ol>
</div>
<div class="cont">
<h2>Most recently added</h2>
<ol>
$for link in dateLinks:
    <li class="li2"><a href="$link.url">$link.title</a><span style
        ="float:right"><a href="/dyn/uid=$link.uid&inc=true"><img
        src="/static/thumb-up.png"></img></a>  <a href="/dyn/uid=
        $link.uid&inc=false"><img src="/static/thumb-down.png"></
        img></a> (+$link.score)  </span><br />Added $link.timeAgo
    </li>

</ol>
</div>
<br style="clear:both" />
</div>
```

Bilaga C

Detaljerad statistik över prestandajämförelse

Program Source Code	CPU secs	Elapsed secs	Memory KB	Code B	≈ CPU Load
fasta					
Go	2.10	2.11	756	1236	2% 0% 0% 100%
C++ GNU g++	1.88	1.88	984	1474	0% 0% 0% 100%
n-body					
Go	31.45	31.46	756	1310	0% 0% 0% 100%
C++ GNU g++	23.06	23.07	728	1428	0% 0% 100% 0%
spectral-norm					
Go	15.68	3.94	1,900	536	98% 98% 99% 99%
C++ GNU g++	10.02	2.52	1,064	1044	100% 100% 100% 100%
fannkuch-redux					
Go	85.19	21.38	1,012	951	100% 100% 100% 100%
C++ GNU g++	49.42	12.63	1,128	1439	100% 99% 94% 98%
pidigits					
Go	4.09	4.09	2,884	607	0% 0% 0% 100%
C++ GNU g++	2.29	2.30	1,576	682	2% 0% 1% 100%
reverse-complement					
Go	1.35	1.36	160,672	547	0% 3% 1% 100%
C++ GNU g++	1.00	0.69	245,728	2275	46% 21% 37% 49%
mandelbrot					
Go	74.44	18.63	30,444	700	100% 100% 100% 100%
C++ GNU g++	24.33	6.12	32,216	1017	100% 100% 100% 100%
k-nucleotide					
Go	55.43	22.07	260,820	1447	84% 96% 26% 44%
C++ GNU g++	11.57	3.32	138,528	3415	81% 86% 84% 98%
binary-trees					
Go	123.55	123.65	321,372	516	0% 0% 0% 100%
C++ GNU g++	26.99	8.40	358,832	892	87% 61% 99% 76%
regex-dna					
Go	337.92	130.02	752,008	733	51% 51% 82% 75%
C++ GNU g++	5.76	3.87	212,736	695	8% 38% 3% 100%

Figur C.1: Detaljerad statistik av jämförelse mellan Go och C++

Program Source Code	CPU secs	Elapsed secs	Memory KB	Code B	≈ CPU Load
fasta					
Go	2.10	**2.11**	756	1236	2% 0% 0% 100%
Java 6 -server	2.44	2.42	14,952	1436	0% 1% 100% 1%
spectral-norm					
Go	15.68	**3.94**	1,900	536	98% 98% 99% 99%
Java 6 -server	17.35	4.43	15,340	950	97% 97% 99% 98%
pidigits					
Go	4.09	**4.09**	2,884	607	0% 0% 0% 100%
Java 6 -server	11.90	4.44	58,656	1816	65% 57% 56% 63%
reverse-complement					
Go	1.35	1.36	160,672	547	0% 3% 1% 100%
Java 6 -server	2.52	1.24	311,496	1661	87% 62% 79% 59%
fannkuch-redux					
Go	85.19	21.38	1,012	951	100% 100% 100% 100%
Java 6 -server	68.43	17.33	15,280	1282	99% 98% 100% 98%
n-body					
Go	31.45	31.46	756	1310	0% 0% 0% 100%
Java 6 -server	22.48	22.48	15,080	1424	0% 0% 0% 100%
mandelbrot					
Go	74.44	18.63	30,444	700	100% 100% 100% 100%
Java 6 -server	31.44	7.98	68,232	802	98% 98% 98% 99%
k-nucleotide					
Go	55.43	22.07	260,820	1447	84% 96% 26% 44%
Java 6 -server	27.65	7.57	397,356	2431	90% 94% 89% 91%
binary-trees					
Go	123.55	123.65	321,372	516	0% 0% 0% 100%
Java 6 -server	18.56	8.86	496,572	1007	71% 61% 31% 46%
regex-dna					
Go	337.92	130.02	752,008	733	51% 51% 82% 75%
Java 6 -server	23.03	6.64	606,432	1410	84% 83% 86% 93%

Figur C.2: Detaljerad statistik av jämförelse mellan Go och Java

Program Source Code	CPU secs	Elapsed secs	Memory KB	Code B	≈ CPU Load
fannkuch-redux					
Go	85.19	**21.38**	1,012	951	100% 100% 100% 100%
Python 3	2,590.30	2,590.84	6,316	385	0% 0% 0% 100%
spectral-norm					
Go	15.68	**3.94**	1,900	536	98% 98% 99% 99%
Python 3	911.83	235.57	39,256	437	97% 96% 97% 96%
n-body					
Go	31.45	**31.46**	756	1310	0% 0% 0% 100%
Python 3	974.10	974.32	6,400	1181	0% 0% 100% 0%
fasta					
Go	2.10	**2.11**	756	1236	2% 0% 0% 100%
Python 3	54.45	54.47	8,124	922	0% 0% 100% 0%
mandelbrot					
Go	74.44	**18.63**	30,444	700	100% 100% 100% 100%
Python 3	1,807.24	454.14	41,360	777	99% 99% 100% 100%
k-nucleotide					
Go	55.43	**22.07**	260,820	1447	84% 96% 26% 44%
Python 3	390.27	129.11	910,192	647	97% 65% 81% 59%
reverse-complement					
Go	1.35	**1.36**	160,672	547	0% 3% 1% 100%
Python 3	6.12	6.13	653,672	325	0% 0% 0% 100%
binary-trees					
Go	123.55	**123.65**	321,372	516	0% 0% 0% 100%
Python 3	470.24	125.97	933,964	561	94% 91% 95% 92%
pidigits					
Go	4.09	4.09	2,884	607	0% 0% 0% 100%
Python 3	2.40	2.41	7,004	255	1% 0% 1% 100%
regex-dna					
Go	337.92	130.02	752,008	733	51% 51% 82% 75%
Python 3	23.65	12.21	491,780	478	28% 27% 99% 39%

Figur C.3: Detaljerad tatistik av jämförelse mellan Go och Python