SDMAPeG

- att utöka processorkraft via nätverk

Niklas Alexandersson
Oskar Nyberg
Victor Nilsson
William Dahlberg

Inledning

En dator är i regel begränsad av sin hårdvara, och är därmed begränsad i sin förmåga att utföra tyngre beräkningar. Att vi uppdaterar våra persondatorers kapacitet allt eftersom tekniken och mjukvaran blir mer avancerad är i dag en självklarhet. Syftet med detta projekt är att utforska en annan lösning; möjligheten att distribuera många krävande arbetsuppgifter över ett nätverk av datorer.

Syfte

Projektets syfte är att kunna förkorta beräkningstiden av ett godtyckligt antal prestandakrävande uppgifter, exempelvis primtalsfaktorisering, genom distribution av dessa över flera datorer. Målet är att dessa beräkningar ska kunna utföras korrekt utan ökad tidsåtgång - kanske till och med på kortare tid, förutsatt distribution av två eller fler beräkningar samtidigt.

Teori

Hur snabbt en enstaka dator kan utföra beräkningar beror på vilken hastighet dess processor (CPU) har*. En uträkning som kräver mycket tid för en processor är till exempel primtalsfaktorisering av stora tal, då det inte finns någon algoritm för att snabbt beräkna detta* - det handlar helt enkelt om att testa sig fram. Att göra detta på en enstaka dator binder snabbt upp en stor del, om inte alla, resurser som finns att tillgå. Vid beräkning av primtalsfaktorer av många tal kan det alltså vara rimligt att distribuera beräkningarna för varje ursprungstal över flera datorer, då dessa beräkningar kan ske parallellt i stället för sekvensiellt.

Applikationen BOINC är ett exempel på teorin bakom systemet (BOINC, 2013). BOINC är implementerat i större skala och står som modell för vårt projekt. Principen bakom är inte allt för komplicerad, se fig. 1.

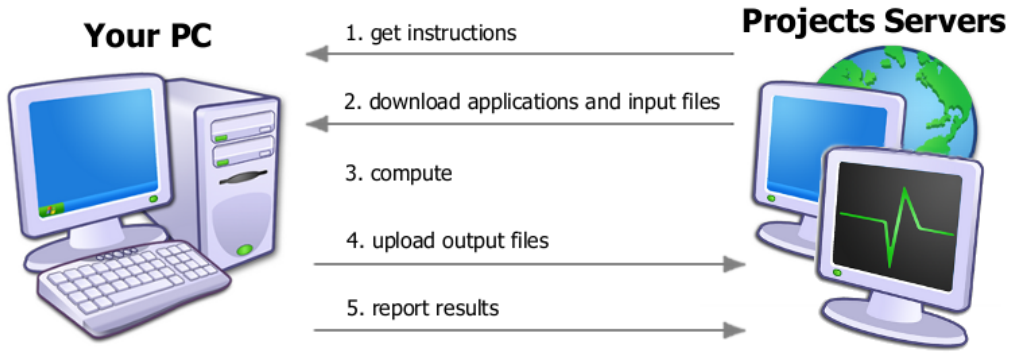

Fig 1: Modell av applikationen BOINC
(http://boinc.berkeley.edu/w/images/1/14/Comm_simple3.png)

Metod

Arkitektur

För ett projekt som sträcker sig över tre separata applikationer krävs gedigen planering. Under första veckan diskuterades enbart hur systemet skulle byggas upp, vilka uppgifter som varje modul skulle utföra, och hur de skulle kommunicera med varandra. Planeringstiden ägnades också åt att definera gränssnitt för större delen av systemet och deras gemensamma klasser. Arkitekturen planerades givetvis gemensamt då det är grunden för hela projektet, och det är av yttersta vikt att samtliga medarbetare var medvetna om vad som skulle göras.

Implementation

När väl samtliga gränssnitt definierats påbörjades implementation av dessa. Strukturen under implementationsfasen var något flytande då komplexiteten hos de olika klasserna är väldigt varierande; vissa gränssnitt implementerades enbart genom att skriva enstaka rader kod i varje metod.

Resultat

Funktionalitet

Projektet har resulterat i ett fullt fungerande system som utgörs av tre olika applikationer: klient, server och arbetare. Dessa tre är tänkta att starta på lika många olika datorer för att systemet ska fylla någon form av funktion. Klientapplikationen definierar en uppgift som ska utföras, exempelvis den tidigare nämnda primtalsfaktoriseringen. Denna uppgift skickas i sin tur till servermodulen, som hanterar uppkopplingar mellan klientapplikationer och arbetarapplikationer. Arbetarapplikationerna är, som namnet implicerar, anslutningen till de datorer som utför de faktiska beräkningarna. Systemet är uppbyggt för att tillåta ett godtyckligt antal klient- och arbetarapplikationer per server. De faktiska uppgifterna kan en användare definiera i kod via Python, eller välja bland ett antal förbestämda.

De grafiska gränssnitten i worker- och serverapplikationerna fyller ingen viktig funktion, och används snarare för att visualisera statistik. Klientens användargränssnitt har ett viktigare syfte, då en uppgift måste kunna definieras på ett överskådligt sätt.

Prestanda

För att kunna fungera krävs att nätverket som används för att låta de olika datorerna i systemet tillåter de portar som används - framför allt måste uppkopplingen till serverapplikationen vara tillgänglig för arbetaren och klienten. Om servern är uppkopplad till Internet via en router krävs därför att de portar som systemet använder görs tillgängliga. I övrigt kan systemet hantera de flesta tänkbara fel som kan tänkas uppstå med uppkopplingar, applikationer som stängs av under pågående beräkning, och så vidare. Programmet är förfinat nog för att aktivt försöka utjämna arbetsbördan mellan anslutna arbetarapplikationer - om någon av de anslutna datorerna har tillgänglig processorkraft kan systemet omdistribuera arbetet vid behov

Avslut

TODO

Källor:
http://docs.oracle.com/javase/tutorial/ (För mycket av koden överlag)
Göetz, Peierls, Bloch, Bowbeer, Holmes, Lea: *Java Concurrency in Practice* (Stilguide)