Obligatorisk oppgave 8: Regneklynge

IN1000

Frist: mandag 13. november 2017 kl. 12:00

Versjon 1.0 Siri Moe Jensen. Basert på oppgave i INF1010 skrevet av Kristian Hustad.

Innledning

I denne oppgaven skal du lage et program for å holde oversikt over alle komponentene i en *regneklynge* (eng: computer cluster). En slik regneklynge kan brukes til å fordele tunge beregninger på mange maskiner slik at de kan jobbe i parallell. På den måten kan en simulering som ville tatt en måned å kjøre på en vanlig maskin, kjøres på regneklyngen på noen timer i stedet. Det finnes flere regneklynger på UiO, hvorav Abel er den største.

Regneklyngens bestanddeler

En regneklynge består av ett eller flere *rack* (et kabinett med skinner) hvor mange *noder* kan monteres over hverandre. En *node* er en selvstendig maskin med et hovedkort med én eller flere prosessorer og minne, i tillegg til en del andre ting. I denne oppgaven skal vi bare se på antall prosessorer (maks 2) og størrelsen på minnet. Du kan derfor anta at en node har en eller to prosessorer og et heltallig antall GB med minne.

Programdesign

Programmet skal designes med tre klasser som representerer noder, racks og regneklynge. Et objekt av klassen regneklynge skal kunne referere til ett eller flere rack-objekter, der hvert rack-objekt igjen refererer til en eller flere node-objekter. Noen flere krav og tips til design av den enkelte klassen finner du videre i oppgaven.

Alternativ I: Mer ferdig design og kode

Under overskriften **Oppgaver og levering** nedenfor finner du beskrivelsen av hva programmet ditt skal kunne utføre. Dersom du ønsker mer hjelp på veien kan du ta utgangspunkt i vedlagte fil som viser det *offentlige grensesnittet* for hver klasse. Her finner du en dokumentert signatur (metode-hode) for de metodene klassene skal tilby utad, og som du skal implementere (skrive ferdig). Det kan i tillegg være nyttig å implementere hjelpemetoder (private metoder) som kun brukes innenfor den enkelte klasse (angis i Python med tegnet '_' foran metodenavnet).

Dersom du ønsker å arbeide mer selvstendig med oppgaven og lage dine egne grensesnitt for klassene er det flott – så lenge programmet omfatter disse tre klassene og har funksjonalitet som beskrevet i c) under "Oppgaver og levering" nedenfor.

Alternativ II: Mer selvstendig design

Klassen Node

Klassen skal kunne initiere nye objekter med ønsket minnestørrelse og prosessorantall, og for øvrig tilby tjenester (metoder) som trengs i andre deler av programmet.

Klassen Rack

Klassen `Rack` skal lagre `Node`-objektene som hører til et rack i en liste. Vi skal kunne legge til noder i racket hvis det er færre enn maks antall noder der fra før. For enkelhets skyld skal vi anta at hvert rack i regneklyngen har plass til like mange noder. Andre instansvariable og metoder etter behov.

Klassen Regneklynge

Klassen `Regneklynge` skal holde rede på en liste med racks, og må tilby en metode som tar imot et nodeobjekt og plasserer det i et rack med ledig plass. Hvis alle rackene er fulle, skal det lages et nytt `Rack`-objekt som legges inn i listen, og noden plasseres i det nye racket.

Tips. Det kan være lurt å ta inn antall noder per rack i konstruktøren til `Regneklynge`.

Oppgaver og levering

Du skal levere programmet i Devilry med en fil per klasse og hovedprogram, samt tegning fra oppgave a). Om du løser d) er det nok å levere siste versjon av programmet.

a) Datastrukturtegning

Merk. Denne deloppgaven skal være besvart - men tegningen vil ikke kunne trekke ned i vurdering av oppgaven, og du velger selv hvordan du synes det er nyttig å illustrere. Leveres som en egen fil i .png, .pdf eller lignende format.

Forslag. Variable tegnes som navngitte bokser med verdien inni. Objekter tegnes som firkanter med instansvariable inni. Klassenavn kan evt skrives over objektet med «:» foran. Referanser tegnes som piler fra variabelen de ligger i til objektet de refererer til.

Tegn datastrukturen slik den ville sett ut etter at vi har satt inn følgende noder i et nytt objekt av `Regneklynge`. La max antall noder per rack være 2.

* Node 1: 16 GB minne, 1 prosessor

* Node 2: 16 GB minne, 1 prosessor

* Node 3: 128 GB minne, 2 prosessorer

b) Antall prosessorer og minnekrav

Lag en metode `antProsessorer(self)` i `Regneklynge` som returnerer det totale antall prosessorer i regneklyngen.

Noen programmer trenger mye minne, typisk et gitt antall GB med minne på hver node vi bruker. Vi er derfor interessert i å vite hvor mange noder som har nok minne til at vi kan bruke dem. Lag en metode `noderMedNokMinne(self, paakrevdMinne)` i `Regneklynge` som returnerer antall noder med minst `paakrevdMinne` GB minne.

Utvid klassene Node og Rack slik at de støtter implementeringen av disse metodene.

c) Hovedprogram

Lag et hovedprogram for å teste at klassene virker som de skal. Lag en regneklynge, `abel`, og la det være plass til 12 noder i hvert rack. Legg inn 650 noder med 64 GB minne og en prosessor hver. Legg også inn 16 noder med 1024 GB minne og to prosessorer.

Sjekk hvor mange noder som har minst 32 GB, 64 GB og 128 GB minne. Finn totalt antall prosessorer, og sjekk hvor mange rack som brukes. Skriv ut svarene i terminalen. Utskriften kan f.eks. se slik ut:

```
Noder med minst 32 GB: 666
Noder med minst 64 GB: 666
Noder med minst 128 GB: 16

Antall prosessorer: 682
Antall rack: 56
```

d) Lese fra fil (frivillig ekstraoppgave – anbefales!)

Skriv en ny konstruktør til klassen regneklynge. Denne skal ha et filnavn som parameter (i stedet for max antall noder per rack), og lese alle data om regneklyngen fra fil. Filformatet er slik:

Frivillig tillegg til alternativ I eller II ovenfor

```
# Max noder per rack
# AntallNoder MinnePerNode AntallProsessorerPerNode
# AntallNoder MinnePerNode AntallProsessorerPerNode
# ...
```

`For regneklyngen med bestanddeler som` i oppgave c) blir innholdet i filen slik:

```
12
650 64 1
16 1024 2
```

Endre hovedprogrammet slik at det oppretter en regneklynge med data fra filen "regneklynge.txt" (som du lager selv etter mønster fra eksempelet over) før det gjør beregningene beskrevet i oppgave c). Test med flere variasjoner av data i filen.