Documentatie Project NetCentric

Gilles Lesire

Het cloud computing project bestaat uit 4 Java projecten, de loadbalancer, de scaler, de workers en de client.

# LoadBalancer project

De load balancer gaat zorgen voor de taakverdeling over de workers. Abstract genomen bestaat de load balancer uit 2 queues en 1 hashmap.

* 1 queue met jobs (runnable jar files)
* 1 queue met subscribed workers
* 1 hashmap met results

Ik maak gebruik van 2 queues om op deze manier de communicatie tussen de workers en de load balancer zwaar te verminderen. In plaats van de workers constant te laten pollen naar de load balancer of er nieuwe jobs ter beschikking zijn (wat een zware belasting is op de load balancer) schrijven de workers zich in bij de load balancer. De load balancer gaat dan zelf pas terug contact maken met de worker als hij een taak voor deze worker heeft.

Als een client een job wilt uitvoeren in de cloud dan gaat hij deze doorsturen naar de load balancer.

Wanneer de load balancer een nieuwe taak ontvangt gaat hij kijken in zijn workers queue om te zien of er workers ingeschreven zijn. Indien ja, dan stuurt hij de taak meteen door naar de eerstvolgende worker in de rij, indien niet, dan gaat hij de taak bijhouden in zijn queue met jobs. Er wordt voor elke taak een “resultkey” gegenereerd, deze wordt teruggegeven aan de client. De client moet deze key gebruiken voor zijn resultaat op te vragen. Deze key wordt ook meegegeven naar de worker samen met de taak.

Op een gelijkaardige manier gaat de load balancer, als een worker zichzelf “inschrijft”, kijken of er jobs in de queue staan, indien ja stuurt hij meteen een job naar de worker, indien nee dan gaat hij deze worker bijhouden in zijn worker queue en zal hij deze worker aanspreken wanneer een nieuwe taak binnenkomt.

Als een worker zijn taak afgerond heeft gaat hij zijn resultaat samen met bijhorende key terugsturen naar de load balancer. Die deze dan in zijn hashmap met results plaatst.

De client zal regelmatig pollen naar de load balancer met zijn gegeven key of zijn resultaat al berekend is.

De load balancer draait op een AWS machine en wordt automatisch opgestart door de scaler. De load balancer algoritme zal automatisch starten via een startlistener.

# Scaler

De scaler gaat automatisch bij startup een instantie van de load balancer starten.

Vanaf dat moment is de scaler een eeuwige lus die elke 5 seconden wordt doornomen en hier gaat hij nakijken hoeveel taken er nog in de job queue van de load balancer staan, hoeveel machines er aan staan, wat de gemiddelde work load is. Op basis hiervan gaat hij beslissen of er nieuwe machines gestart moeten worden of geterminate moeten worden. Er moet altijd minstens 1 worker draaien omdat anders een nieuwe taak veel te lang moet wachten als de rij voordien leeg was.

Telkens als een nieuwe worker wordt aangemaakt zal de scaler aan de worker het public DNS adres van de load balancer meegeven.

De scaler draait lokaal en moet manueel gestart worden als een Java Applicatie.

# Worker

De worker gaat als zijn workload onder de 40% is zich inschrijven voor nieuwe taken bij de load balancer. In dit testgeval heb ik ook ingesteld dat de worker maximaal 3 taken tegelijk kan draaien omdat de voorbeeldtaken niet genoeg rekenkracht nodig hebben om de workload hoog te krijgen.

Bij startup start een thread (LoadThread) die om de 5 seconden de work load op de worker gaat controleren en op basis daarvan zich gaat inschrijven bij de load balancer.

Als de worker een nieuwe taak krijgt dan gaat hij deze taak toevoegen aan zijn singelton lijst met current running tasks, de worker gaat dus de objecten bijhouden die hij op dat moment aan het berekenen is. Als een taak afgerond is wordt deze taak uit de lijst verwijderd en wordt het resultaat teruggestuurd naar de load balancer.

Waarom hou ik bij welke taken er draaiende zijn? Om zo het aantal draaiende taken te limiteren tot maximaal 3, en ook indien de worker wordt geterminate door de load balancer, dan worden de onafgewerkte taken teruggestuurd en vanvoor terug toegevoegd in de task queue van de load balancer.

De worker draait op een AWS instance en wordt automatisch opgestart door de load balancer.

# Client

De client gaat de taken doorsturen naar de load balancer. Zodra hij een taak gestuurd heeft krijgt hij een result key terug van de load balancer. De client zal dan elke 10 seconden pollen bij de load balancer met zijn resultkey om te kijken of zijn resultaat al klaar is.

Bij het doorsturen van de taak stuurt de client een serializatie van zijn taak mee samen met de link naar de jarfile die van het javaobject van de taak. In dit geval is de link een weblink <http://businessgame.be/netcentric/classes.jar>.

De client draait lokaal en moet manueel gestart worden als een Java Applicatie.