Afstudeerscriptie

Ontwikkelen van distributed software environment in de cloud voor 3Di-Waterbeheer

Opdrachtnemer:
Hugo Meijer
Student HHS
08066493

Originele opdrachtgever:
Dr. Ir. Gerwin de Haan
Technische Universiteit Delft

Bedrijfsmentor:
Dr. Ir. Tim Tutenel
Technische Universiteit Delft

Begeleidend examinator:
Henk van den Bosch
Haagse Hogeschool

25 maart 2013

Referaat

Dit verslag beschrijft het proces dat is doorlopen bij het uitvoeren van de afstudeeropdracht "Ontwikkelen van distributed software environment in de cloud voor 3Di-Waterbeheer-bij de TUdelft.

Afstudeerder	Hugo Meijer
Onderwijsinstelling	Haagse Hogeschool
Opleiding	Technische Informatica
Organisatie	TUDelft
Bedrijfsmentor	Dr. Ir. Tim Tutenel
Opdrachtgever	Dr. Ir. Gerwin de Haan

Descriptoren	Python Celery HPC Cloud Cloud Computing

Voorwoord

Mijn naam is Hugo Meijer en ik studeer Technische Informatica op de Haagse Hogeschool te Delft. Momenteel ben ik bezig met het afstuderen bij de TU-Delft. Dit verslag is bedoeld om duidelijkheid te creëren over de werkzaamheden die ik tijdens deze periode verricht heb.

Mijn dank gaat uit naar Gerwin de Haan, Tim Tutenel en Christian Kehl voor het begeleiden van mijn opdracht. Ook gaat mijn dank uit naar mijn twee begeleiders vanuit school, Henk van den Bosch en John Visser, voor de goede feedback.

Samenvatting

Dit project gaat over het ontwikkelen van een software omgeving voor software distributie in de cloud. Dit project is uitgevoerd in opdracht van 3Di-waterbeheer en de werkzaamheden zijn op de TUDelft verricht.

3Di waterbeheer werkt aan het verbeteren van overstromingssimulaties. Hiervoor hebben ze een zeer gedetailleerde 3D kaart van Nederland in gebruik genomen, de AHN2 kaart. Om specialisten te helpen bij het visualiseren van de data is de TUDelft bezig met het maken van een visualisatie programma. Om de data te kunnen visualiseren is het echter wel nodig om een aantal bewerkingen op de AHN2[1] kaart te doen.

Deze bewerkingen zijn zeer tijdsintensief en het is daarom lastig om deze op de eigen hardware van de TUDelft uit te voeren omdat dit de mensen hier hindert. Hierom is het de bedoeling dat dit naar de cloud verplaatst wordt. De cloud die hiervoor gebruikt is, is de HPC Cloud[2] van SARA[3].

Er is een uitgebreid vooronderzoek gedaan naar de werking van de cloud en de randsystemen en ook is er een analyse gemaakt van het originele systeem. Het originele systeem bleek erg inefficiënt te zijn indien er meerdere processoren gebruikt werden. Dit kwam door de constructie van deze code. Er zijn een vijftal ontwikkel incrementen gemaakt om alles werkend te krijgen. Als eerst zijn de virtuele machines aangemaakt die op de cloud draaien om met de cloud te kunnen werken.

De cloud is aan te sturen van binnenuit met XML-RPC[4] en van buitenaf met OCCI[5]. Hiermee is het mogelijk om VMs[6] te starten en te stoppen in een script. Celery[7] wordt gebruikt voor het distribueren van de taken over de verschillende VMs op de cloud. Nadat de eerste stap van het distribueren gezet was bleek dat er een grote verbetering gehaald was in de bezetting van de gebruikte processoren.

Ook is er gewerkt aan het afsluiten van de VMs zodat deze op een tijdig

[1] Actuele Hoogtekaart Nederland 2: www.ANN.nl
[2] High Performance Computing Cloud: www.cloud.sara.nl
[3] Stichting Academisch Rekencentrum Amsterdam: www.surfsara.nl
[4] XML-Remote Procedure Call
[5] Open Cloud Computing Interface: occi-wg.org
[6] Virtual Machine
[7] Asynchroon taken framework: www.celeryproject.org

moment stoppen en niet onnodig lang aan blijven staan.

Het uiteindelijke product is met succes gebruikt voor het voorbewerken van data op de HPC Cloud van SARA. Om het echt in de praktijk te kunnen gebruiken zal er echter een nieuw project gestart moeten worden om dit te verplaatsen naar een definitieve oplossing zoals de cloud van Amazon.

Inhoudsopgave

1 Inleiding **10**

2 Organisatie **12**
2.1 TUDelft 12
2.2 3Di waterbeheer 14
2.3 Deltares 14
2.4 Nelen & Schuurmans 14
2.5 TUDelft 14
2.6 Onderverdeling 15

3 Opdrachtdefinitie **16**
3.1 Totstandkoming Project 16
3.2 Probleemstelling 17
3.3 Doelstelling 17
3.4 De opdracht 18
3.5 Op te leveren producten 19
3.6 Projectgrenzen 20
3.7 Randvoorwaarden 20

4 Aanpak **21**
4.1 Project management 21
4.2 Werkzaamheden 22
4.3 Incrementenplan 22
4.3.1 Het distribueren van de voorbewerking 23
4.3.2 Het creëren van een software omgeving in de cloud . . 23
4.3.3 Het aansturen van de cloud 24
4.3.4 Het werken met niet locale data 24
4.3.5 Het onmiddellijk afsluiten van virtual machines die klaar zijn met hun taken 25
4.4 Standaarden 25

5 Werking van de Cloud **26**
5.1 SARA 26
5.2 Netwerk 27

5.2.1 TUDelft . . . 27
5.2.2 SARA . . . 27
5.3 VIRDIR . . . 28
5.4 HPC Cloud . . . 28
5.4.1 Aanstuurmethoden . . . 29

6 Werking van preprocessing **33**
6.1 AHN2 . . . 33
6.1.1 LiDAR . . . 34
6.2 De voorbewerkingen . . . 36
6.2.1 txt2las . . . 36
6.2.2 las2las . . . 37
6.2.3 las2las2 . . . 37
6.2.4 las2osg . . . 38
6.3 Preprocessing . . . 39
6.3.1 Python script . . . 39

7 Distributie software **41**
7.1 Celery . . . 41
7.1.1 Message Broker . . . 42
7.1.2 Backend Database . . . 42

8 Increment 1: Creatie van de omgeving **43**
8.1 Analyse . . . 43
8.1.1 Requirements . . . 43
8.2 Ontwerp . . . 43
8.2.1 Server VM . . . 44
8.2.2 Client VM . . . 44
8.2.3 Deployment . . . 44
8.2.4 Flow Chart . . . 45
8.2.5 Class Diagram . . . 46
8.3 Implementatie . . . 47
8.4 Bouw . . . 47
8.4.1 Server . . . 47
8.4.2 Client . . . 47
8.4.3 Celery code . . . 48
8.5 Test . . . 49
8.5.1 Celery Test . . . 49
8.5.2 Processor test . . . 50

9 Increment 2: Remote opstarten van cloud virtual machines **52**
9.1 Analyse . . . 52
9.2 Ontwerp . . . 53
9.2.1 XML-RPC . . . 53

9.2.2 OCCI 55
9.3 Implementatie 57
9.3.1 XML-RPC 57
9.3.2 OCCI 57
9.4 Test 59
9.4.1 XML-RPC 59
9.4.2 OCCI 59

10 Increment 3: Preprocessing omzetten in Celery task 60
10.1 Analyse 60
10.2 Ontwerp 61
10.2.1 Deployment 61
10.2.2 Flow charts 61
10.3 Implementatie 62
10.3.1 Task initiator 62
10.3.2 Tile Preprocessor 63
10.4 Test 63

11 Increment 4: Taskspawner 64
11.1 Analyse 64
11.2 Ontwerp 64
11.3 Implementatie 65
11.3.1 Serialisatie van configuratie data 65

12 Increment 5: Dynamisch stoppen van VMs 67
12.1 Analyse 67
12.2 Ontwerp 67
12.3 Implementatie 69
12.3.1 Change counter 69
12.3.2 OCCI 70
12.3.3 XML-RPC 70

13 Verder onderzoek 71

14 Conclusie 72

15 Nawoord 73

16 Begrippenlijst 74

17 Bronnen 76

Lijst van figuren

2.1 EEMCS gebouw 13
2.2 Organogram 13

3.1 Simpel overzicht netwerk. Uitgebreider te zien in Figuur 5.2 16

4.1 Incremental Development 22

5.1 Logo SARA 26
5.2 Netwerk topologie 27
5.3 Web based UI voor HPC Cloud 29
5.4 Menu van de website 30
5.5 Template tab 30
5.6 Voorbeeld van een image in de Images tab 31
5.7 Meter die laat zien hoeveel procent van de tijd gebruikt is . . 31

6.1 Huidige methode tegenover AHN2 34
6.2 Schematische weergave van vliegtuigen met LiDAR scanners . 35
6.3 Schouwen-Duiveland ingedeeld in tiles 35
6.4 De prototype toolchain van het hele process. Alleen de pre-processing toolchain wordt in dit project uitgevoerd 36
6.5 Een luchtfoto die gemaakt is tijdens werkzaamheden en een LiDAR scan die hierna gemaakt is. Het resultaat is bomen met de kleur van zand 37
6.6 3D rendering van Terneuzen, Zeeuws-Vlaanderen. De witte vlek komt van de wolk van een koeltoren. 38
6.7 Indeling van dynamisch level of detail. Ieder vakje is een file. 39

7.1 Schematische weergave Celery 42

8.1 Deployment diagram increment 1 45
8.2 Flow chart test creatie van omgeving 46
8.3 Processor use 51

9.1 Use Case Diagram 53
9.2 Deployment Diagram 54
9.3 flow chart Increment 2: XML-RPC 54

9.4 Sequence start VM Increment 2: XML-RPC 55
9.5 Sequence stop VM Increment 2: XML-RPC 55
9.6 Deployment Diagram OCCI 56
9.7 Flow chart OCCI 56

10.1 Use Case Increment 3 60
10.2 Deployment Increment 2 61
10.3 Flow Chart Preprocessing Initiator Increment 3 61
10.4 State Machine Preprocess Tile Increment 3 62
10.5 Processor use 63

11.1 Use Case Increment 4 64
11.2 Flow chart increment 4 65
11.3 Klasse diagram Configuration 66

12.1 Flow chart van dynamisch afsluiten VMs 69

Hoofdstuk 1

Inleiding

Dit verslag is geschreven in het kader van de afstudeeropdracht die uitgevoerd is in opdracht van de Haagse Hogeschool, Technische Informatica. De opdracht is uitgevoerd voor de groep Interactive Visualization and Virtual Reality van de TUDelft. Hier wordt voor het 3Di waterbeheer project gewerkt aan de visualisatie van overstromingssimulatie. Het voorbereiden van de data voor deze simulatie is echter zeer tijdsintensief. Dit proces wordt daarom gedistribueerd en naar externe cloud computing faciliteiten verplaatst. De uitvoering van deze opdracht wordt in dit verslag beschreven.

Dit verslag is opgebouwd uit de volgende onderdelen:

Organisatie In dit hoofdstuk word een duidelijke beschrijving gegeven van de organisatie waar binnen dit project is uitgevoerd. Hierna worden belangrijke partner bedrijven genoemd die bijdragen aan het overkoepelende project.

Opdrachtdefinitie Dit hoofdstuk bevat een korte samenvatting van de opdracht voor dit project en hoe deze binnen de organisatie tot stand is gekomen.

Aanpak Dit hoofdstuk creëert duidelijkheid over de aanpak die gekozen is voor dit project. Ook worden de werkzaamheden die bij deze aanpak horen hier beschreven.

Werking huidige software In dit onderdeel wordt beschreven hoe de huidige situatie is en hoe hiermee gewerkt moet worden. Dit onderdeel is in 3 delen opgedeeld: Het netwerk, het huidige systeem en de frameworks die gebruikt moeten worden voor de aanpassing.

Kern In dit onderdeel van het verslag wordt duidelijk welke werkzaamheden er verricht zijn. Het onderdeel begint met de creatie van de omgeving waarin gewerkt is. Hierna worden de incrementen stuk voor stuk doorlopen.

Begrippenlijst In dit hoofdstuk worden alle afkortingen en definities die gebruikt worden in dit verslag uitgelegd. Als een definitie of afkorting gebruikt wordt in het verslag zal er in de voetnoot een korte omschrijving staan en een verwijzing naar dit onderdeel.

Bijlagen Dit onderdeel bevat de relevante bijlages voor dit verslag.

Hoofdstuk 2

Organisatie

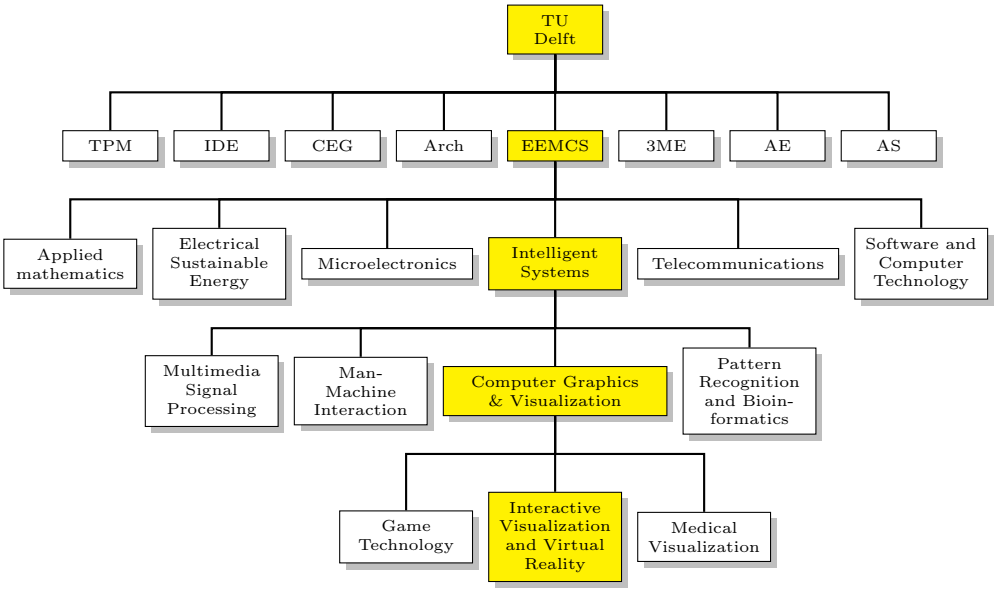
2.1 TUDelft

De Technische Universiteit Delft is opgericht op 8 Januari 1842 als een koninklijke academie voor technische en handelsgerichte vakken. De koninklijke academie werd in 1864 ontbonden en veranderd in een polytechnische school. In 1905 werd de school verheven tot "Technische Hogeschool van Delft" en had een officiële aanduiding van het academische niveau. Doordat een wet in 1985 bepaalde dat de term hogeschool alleen nog maar voor hoger beroeps onderwijs instellingen mocht gelden, ontstond op 1 september 1986 de Technische Universiteit Delft. Tegenwoordig omvat de Technische Universiteit Delft ongeveer 16.000 studenten. Deze studenten zijn verdeelt over verschillende faculteiten.

Dit onderzoek wordt gedaan op de EEMCS (Electrical Engineering, Mathematics and Computer Science) faculteit.

Figuur 2.1: EEMCS gebouw

EEMCS heeft zes afdelingen waaronder de Intelligent Systems afdeling. Deze afdeling, samen met de afdeling Software and Computer Technology, houd zich bezig met het bevorderen van de wetenschap en kennis in het veld van autonome systemen en informatie analyse en interactie. Binnen deze afdeling zit de Computer graphics en visulalisation group waar deze opdracht uitgevoerd wordt.



Figuur 2.2: Organogram

De werkzaamheden die beschreven staan binnen dit verslag zijn uitge-

voerd binnen de Interactive visualization and Virtual Reality group. Ook zijn de wekzaamheden onderdeel van het 3Di waterbeheer project.

2.2 3Di waterbeheer

3Di waterbeheer[1] is een vierjarig innovatieproject gestart in 2009, waarbinnen verschillende (ICT) producten worden ontwikkeld voor waterbeheerders, ruimtelijke ontwikkelaars en calamiteitenorganisaties. Deze producten maken het mogelijk wateroverlast nauwkeuriger en sneller te voorspellen. 3Di Waterbeheer biedt gedetailleerde informatie over wateroverlast als gevolg van hevige neerslag en overstromingen, direct inzicht in de effecten van maatregelen, realtime informatie ontsloten via een interactief webportaal en een duidelijk inzicht in wateroverlast door middel van 3D-animaties.

De initiatiefnemers en ontwikkelaars van 3Di Waterbeheer zijn de TU Delft, Deltares[2] en Nelen & Schuurmans[3].

2.3 Deltares

Deltares is een onafhankelijk toegepast kennisinstituut op het gebied van water, ondergrond en infrastructuur. Wereldwijd werken ze aan slimme innovaties, oplossingen en toepassingen voor mens, milieu en maatschappij. Ze richten zich voornamelijk op delta's, kustregio's en riviergebieden. Deltares heeft ruim 800 medewerkers en is gevestigd in Delft en Utrecht.

2.4 Nelen & Schuurmans

Nelen & Schuurmans is een adviesbureau op het gebied van water management dat de publieke en private sector ondersteunt bij strategische en operationele watervraagstukken. De groep bestaat uit hoog opgeleide adviseurs in diverse disciplines, zoals hydrologie, waterbeheer, waterkwaliteit, ecologie, fysische geografie, wiskunde en informatietechnologie.

2.5 TUDelft

De groep die zich binnen de Interactive visualization and Virtual Reality group met 3Di Waterbeheer bezig houdt bestaat uit vier personen. Tim Tutenel die alles begeleidt; Christian Kehl, een PHD student die bezig is met

[1] www.3di.nu
[2] www.deltares.nl
[3] www.nelen-schuurmans.nl

het renderen van data; Lars Wijtemans die in opdracht van Nelen & Schuurmans bezig is en ondergetekende, die de software omgeving voor distributed cloud computing maakt.

Ook is de afdeling Civiele Techniek en Geowetenschappen binnen de TUDelft bezig met het berekenen van de simulatie van overstromingen voor het 3Di project.

2.6 Onderverdeling

Binnen 3Di Waterbeheer zijn de taken verdeeld over de verschillende organisaties. Nelen & Schuurmans houdt zich onder andere bezig met het maken van de webinterface en werkt samen met Interactive Visualisation en Virtual Reality aan de 3D visualisatie van het landschap. Deltares is voornamelijk bezig met de overstromingssimulatie zelf. Hiervoor moeten extra factoren zoals het grondwatersysteem en rioleringen mee gerekend worden.

Hoofdstuk 3

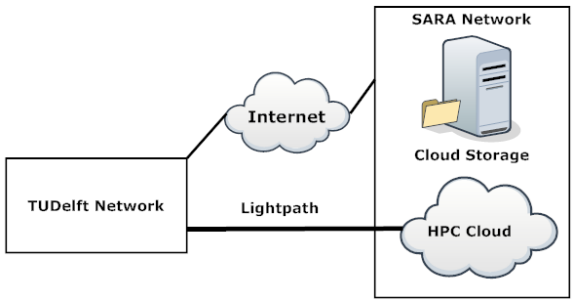
Opdrachtdefinitie

3.1 Totstandkoming Project

in 2011 is er door Gerwin de Haan van de Computer Graphics and Visualisation Groep meegedaan aan de Enlighten Your Research (EYR) wedstrijd van SURFnet. De winnaars van deze wedstrijd kregen toegang tot zelf bepaalde infrastructuren bij SARA[1] en de verbindingen die hiervoor nodig waren van SURFnet.

Het project had als doel om de TUDelft en de RijksUniversiteit Groningen beter te verbinden en het mogelijk te maken om interactieve sessies op het gebied van overstromings simulaties gezamenlijk uit te kunnen voeren.

Dit idee heeft een gedeelde derde prijs gekregen en heeft hiermee de benodigde infrastructuur tot zijn beschikking gekregen. Er is een lichtpad[2] vanaf SARA naar de TUDelft gekomen en een lichtpad vanaf SARA naar de RijksUniversiteit Groningen. Ook is er toegang gegeven voor 2000 core uren op de HPC Cloud[3] van SARA. Hierbij is 2 TB cloud opslag beschikbaar gesteld bij SARA.



Figuur 3.1: Simpel overzicht netwerk. Uitgebreider te zien in Figuur 5.2

[1] Stichting Academisch Rekencentrum Amsterdam: www.surfsara.nl
[2] rechtstreekse glasvezel verbinding naar SARA
[3] High Performance Computing Cloud: www.cloud.sara.nl

Voordat er echter visuele rendering gedaan kan worden voor de overstromings simulaties, moeten de brondata eerst verwerkt worden. Dit gaat om veel brondata in de vorm van een puntenwolk van heel Nederland. Deze berekeningen werden allemaal gedaan op de lokale computer van Christian Kehl en dit hinderde hem in zijn werk.

Als oplossing zou het verwerken van brondata eerst in de cloud gedaan worden. Dit betekent echter wel dat alle software die eerst op het interne netwerk van de TUDelft draaide verplaatst moest worden naar deze nieuwe infrastructuur.

3.2 Probleemstelling

Met behulp van de Actuele Hoogte Kaart Nederland (AHN2) is het mogelijk naukeurige overstromings-simulaties te maken. De kaart, gemaakt door Rijkswaterstaat en locale waterschappen, bestaat uit een puntenwolk van heel Nederland. Deze is zeer nauwkeurig en is dan ook enkele terabytes groot. Om de simulatie resultaten op een duidelijke manier aan experts te tonen moeten deze data gevisualiseerd worden. Om de kaart te kunnen visualiseren moeten er een aantal voorbewerkings-stappen gemaakt worden op de kaart. Vanwege de hoge nauwkeurigheid en de grootte van AHN2 duurt het voorbereiden voor de visualisatie al snel een uur per vierkante kilometer.

De voorbewerking van deze kaart zou ook op de HPC cloud kunnen gebeuren. Dit gaat over een grote hoeveelheid brondata van de totale kaart die sporadisch binnen komt en dan zo snel mogelijk beschikbaar gemaakt moet worden voor simulatie. Op dit moment gebeurt de voorbewerking met behulp van processor intensieve scripts op één workstation van de TUDelft. De scripts worden, als er een nieuwe experiment bedacht wordt, aangepast aan het experiment.

Ook wordt er op dit moment met interne opslag gewerkt. Het plan is om dit aan te passen en te veranderen naar de cloud storage van SARA. SARA's VIRDIR[4] is een storage medium dat gebruikt kan worden door Nederlandse onderzoeksinstellingen waaronder de TUDelft.

Het huidige systeem werkt nog niet met de nieuwe beschikbare infrastructuur. Hiervoor wordt een project gestart om de software hier wel op te laten werken.

3.3 Doelstelling

De doelstelling van dit project is om gebruik te maken van de nieuwe infrastructuur en om te onderzoeken welke mogelijkheden deze heeft. Dit wordt gedaan door de preprocessing pipeline naar de HPC cloud te verplaatsen. Ook wordt de brondata en de resultaten naar de cloud verplaatst.

[4]VIRtual DIRectory

De preprocessing pipeline moet aangepast worden om om te gaan met de veranderingen.

3.4 De opdracht

Om met de HPC cloud te kunnen werken moet er een virtuele machine aangemaakt worden. Deze virtuele machine zal draaien op de HPC cloud en werkt dan als een externe computer. Deze virtuele machine zal de benodigde software voor het preprocessen moeten bevatten om hier goed mee te kunnen werken.

Ook moeten de netwerken van SARA en de TUDelft onderzocht en geanalyseerd worden. Dit is nodig om de communicatie tussen de verschillende instanties goed af te kunnen handelen. Dit zal onderdeel zijn van het vooronderzoek.

De nieuwe pipeline wordt de nieuwe standaard manier van preprocessen binnen het 3Di project. De brondata moeten opgehaald worden van externe opslag. Met deze pipeline moet het ook mogelijk zijn het hele proces te starten, stoppen en monitoren.

Hiervoor moet de bestaande software aangepast worden zodat deze goed op de HPC cloud werkt. De bestaande code is voornamelijk in Python geschreven maar er zijn ook stukken C++. Ook moet er een virtuele machine gemaakt worden die zich als server gedraagt voor de virtuele machine die op de HPC cloud draait.

De nieuwe omgeving moet aan de volgende eisen voldoen. Deze zijn opgesteld met de MoSCoW methode. MoSCoW staat voor Must, Should, Could en Won't en dit zijn de catagorien waarin de eisen ingedeeld zijn.

Virtual machine server		
Nummer	Eis	Prioriteit
s1	De server moet binnen de TUDelft draaien	M
s2	De server moet onafhankelijk zijn van de gebruikte cloud omgeving	M
s3	Ubuntu operating systeem	M

Virtual machine HPC cloud		
Nummer	Eis	Prioriteit
p1	De Virtual machine moet op afstand opgestart kunnen worden	M
p2	De virtual machine moet op afstand afgesloten kunnen worden	M
p3	De virtual machine moet automatisch afsluiten als een preprocessing taak afgelopen is	C
p4	De virtual machine moet met een simpel script te updaten zijn naar de meest recente versie van de code	C
p5	Ubuntu operating systeem	M
p6	De virtual machine moet voor visualisatie doeleindes gebruikt kunnen worden	W

Preprocessing code		
Nummer	Eis	Prioriteit
c1	De code moet van commentaar voorzien zijn	M
c2	De code moet modulair ontworpen worden zodat deze makkelijk aangepast kan worden voor nieuwe experimenten	M
c3	De nieuwe omgeving moet op de zelfde manier gestart kunnen worden als de bestaande oplossing	M

Er zijn geen snelheids-eisen omdat er nog niet bekend is wat haalbaar is met de HPC cloud.

3.5 Op te leveren producten

Er dienen producten opgeleverd te worden bij het project. Deze zijn:

- Plan van Aanpak
- Eisendocument
- Ontwerp
- Source code
- Code documentatie

- Eindverslag

3.6 Projectgrenzen

Dit project zal zich alleen bezig houden met het preprocessen van de AHN2 kaart en zal niet ingaan op het renderen van beelden of het simuleren van overstromingen.

3.7 Randvoorwaarden

De randvoorwaarden van het project zijn: Er moet gewerkt worden met een Linux distributie vanwege de benodigde packages. Vanwege afhankelijkheden van verschillende libraries wordt Python 2.7 gebruikt.

Hoofdstuk 4

Aanpak

Het definiëren van de Aanpak is de eerste stap die gemaakt is. Er wordt hier bepaald hoe het project zal verlopen en welke werkzaamheden gedaan worden.

4.1 Project management

Project management is de discipline van het plannen, organiseren, beheren en controleren van een project. De ontwikkel methode die hiervoor gekozen wordt is dus ook bepalend voor de hele loop van het project. Er is keus uit veel verschillende ontwikkelmethodes en de juiste keuze hangt af van de sterke en zwakke punten van een methode.

Aan het begin van de opdracht is het eindproduct bekend maar de manier waarop dit te realiseren valt nog niet. Gedurende de ontwikkeling moeten er functionele versies zijn die gebruikt kunnen worden voor het voorbewerken. Dit komt omdat er een aantal deadlines zijn zoals een demo die op de tentoonstelling van het 60 jarig jubileum van de watersnoodramp getoond gaat worden. Er is dus een methode nodig die tussentijds al bruikbare resultaten oplevert. Dit sluit strategieën als acceptatie en waterval uit en laat incrementeel en iteratief over.

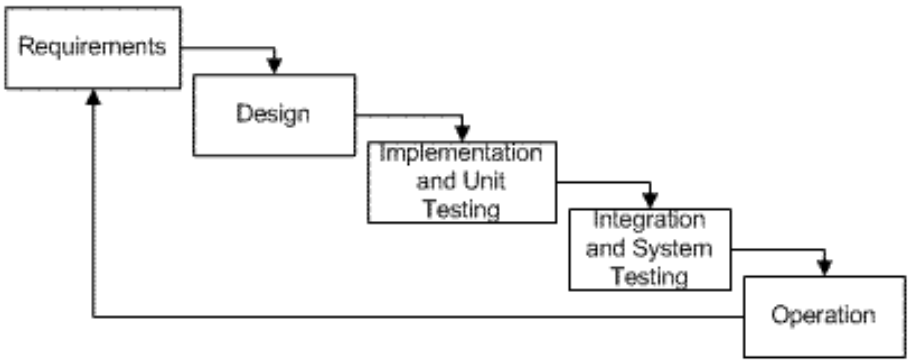
Het verschil tussen iteratief en incrementeel is in hoeverre het complete systeem aanwezig moet zijn in de tussenfasen. Bij een iteratieve methode wordt het hele systeem opnieuw gebouwd en worden alle fases van de ontwikkeling opnieuw doorlopen. Dit is zeer veilig en handig in het geval van een laag inzicht in het eindresultaat omdat er per iteratie makkelijk aanpassingen gemaakt kunnen worden en alles weer anders gedaan kan worden. Dit is echter niet het geval in dit project aangezien er al een duidelijk beeld is van het eindproduct.

Bij de TUDelft wordt veel gebruik gemaakt van Rapid Prototyping[1] en

[1] en.wikipedia.org/wiki/Rapid_prototyping

Rapid Application Development[2]. Dit zijn echter beiden iteratieve manieren van software ontwikkeling. Ondanks dat het een half jaar duurt is het project relatief kort. Omdat er veel tijd besteed wordt aan het onderzoeken van de werking van de cloud en het bouwen van een omgeving hiervoor is het lastig om veel iteraties uit te voeren.

De incrementele methode doorloopt ook alle fasen meerdere malen en alhoewel ieder increment een werkbaar resultaat oplevert wordt het eindproduct pas bereikt als alle incrementen samengevoegd worden. Het laat de gewenste flexibiliteit toe en er is snel een werkbare versie van het eindresultaat.



Figuur 4.1: Incremental Development

4.2 Werkzaamheden

Aan het begin van het project is er een analyse fase. Hierin zal ingelezen worden in het bestaande systeem en de te gebruiken randsystemen. Er zal een eisendocument opgesteld worden en de verzamelde kennis van deze fase zal als materiaal gebruikt worden voor het afstudeer dossier.

Hierna zal er tijd besteed worden aan het creëren van de omgeving. Hier zullen de taken die niet onder software ontwikkeling vallen uitgevoerd worden.

Hierna begint het incrementele systeem. Er zijn een viertal incrementen besloten en deze zijn uitgewerkt. Ieder increment bestaat uit een analyse, ontwerp, implementatie en uiteindelijk zullen er tests uitgevoerd worden.

4.3 Incrementenplan

Er zijn een viertal incrementen bedacht voor dit project. Om bij het eindproduct, een distributed software omgeving in de cloud te komen moeten aan de volgende punten voldaan worden

[2]en.wikipedia.org/wiki/Rapid_application_development

- Het distribueren van de voorbewerking.
- Het creëren van een software omgeving in de cloud.
- Het aansturen van de cloud.
- Het werken met niet locale data.
- Het onmiddellijk afsluiten van virtual machines die klaar zijn met hun taken.

4.3.1 Het distribueren van de voorbewerking

Relevantie

Dit increment moet aan alle eisen van c1, c2 en c3 voldoen.

Volgorde

Dit increment wordt als derde afgerond. Dit komt omdat dit de kern van alle functionaliteit is maar relatief gezien langer duurt dan het aansturen van virtuele machines in de cloud. Hierdoor is het handiger als dit increment als derde gemaakt wordt omdat er zo sneller met de cloud gewerkt kan worden.

Risico

Dit increment kan fout lopen als zou blijken dat het onmogelijk is de huidige voorbewerking in de cloud uit te voeren met de distributie software die gebruikt wordt.

Loopduur

De loopduur van dit increment is 3 weken.

4.3.2 Het creëren van een software omgeving in de cloud

Relevantie

Dit increment is nodig om alle P categorie eisen op te lossen. Zonder dit increment is het niet mogelijk om met de cloud te werken. Aan het eind van dit increment is er al een werkbare versie van de preprocessing code die gebruikt kan worden in de cloud. Dit is dan ook het belangrijkste increment

Volgorde

Dit increment wordt als eerste uitgevoerd. Dit komt omdat dit de opbouw van het systeem is. Zonder dit increment is het hele project niet werkbaar

Risico

Dit increment kan in gevaar komen als er problemen zouden blijken te zijn met de cloud computer van SARA

Loopduur

De geplande loopduur van dit increment is 3 weken.

4.3.3 Het aansturen van de cloud

Relevantie

Dit increment is bedacht om eisen p1 en p2 op te lossen. Het is een belangrijk increment dat ervoor moet zorgen dat het werken met de cloud sneller kan dan normaal.

Volgorde

Dit increment wordt als tweede uitgevoerd. Dit is omdat met behulp van dit increment er het snelst met de cloud gewerkt kan worden. Door een virtuele machine op de cloud op te starten kan hier handmatig het voorbewerken op gestart worden.

Risico

De goede afronding van dit increment kan in gevaar komen als er problemen zijn met het netwerk en het lichtpad naar SARA.

Loopduur

De geplande loopduur voor dit increment is 2 weken.

4.3.4 Het werken met niet locale data

Relevantie

Dit increment lost s1 en s2 op door de data en taken op de cloud zelf aan te maken in plaats van op de virtual machine server. Dit zorgt ervoor dat de server geen kennis hoeft te hebben van de cloud en dat deze ook intern kan draaien. Ook moet alles aan c1, c2 en c3 blijven voldoen

Volgorde

Dit increment wordt als derde uitgevoerd. Hier is van belang dat eerst de voorbewerking volledig gedistribueerd is dus moet dit increment daarna uitgevoerd worden.

Risico

Als de voorgaande incrementen werken is er vrijwel geen risico met dit increment.

Loopduur

De loopduur van dit increment is 2 weken.

4.3.5 Het onmiddellijk afsluiten van virtual machines die klaar zijn met hun taken

Relevantie

Dit increment is bedacht om de eis p3 op te lossen. Dit is een increment dat van belang is voor besparing van cloud uren. Aangezien dit geen probleem is bij SARA heeft dit een lage prioriteit.

Volgorde

Dit increment wordt als laatste uitgevoerd. Dit is omdat er al een andere methode wordt ontwikkeld in het eerste increment waarmee virtual machines op afstand gesloten kunnen worden.

Risico

Dit increment kan mis gaan als er geen goed ontwerp gemaakt kan worden voor dit systeem. Dit systeem is namelijk afhankelijk van wat er mogelijk is met de distributie software.

Loopduur

De originele loopduur van dit increment was 3 weken. Dit is verlengd naar 6 weken wegens implementatie problemen.

4.4 Standaarden

Benodigde software zal geschreven worden in Python tenzij anders afgesproken wordt. De code zal ook voorzien moeten worden van commentaar over de werking hiervan.

Hoofdstuk 5

Werking van de Cloud

5.1 SARA

In 1971 is SARA opgericht onder de naam Stichting Academisch Rekencentrum Amsterdam. De oprichters waren de Universiteit van Amsterdam (UvA), de Vrije Universiteit Amsterdam (VU) en de stichting Mathematisch Centrum (het huidige Centrum Wiskunde & Informatica) in Amsterdam. Aanvankelijk bestonden de werkzaamheden van SARA uit activiteiten gericht op dataverwerking voor de drie stichters.

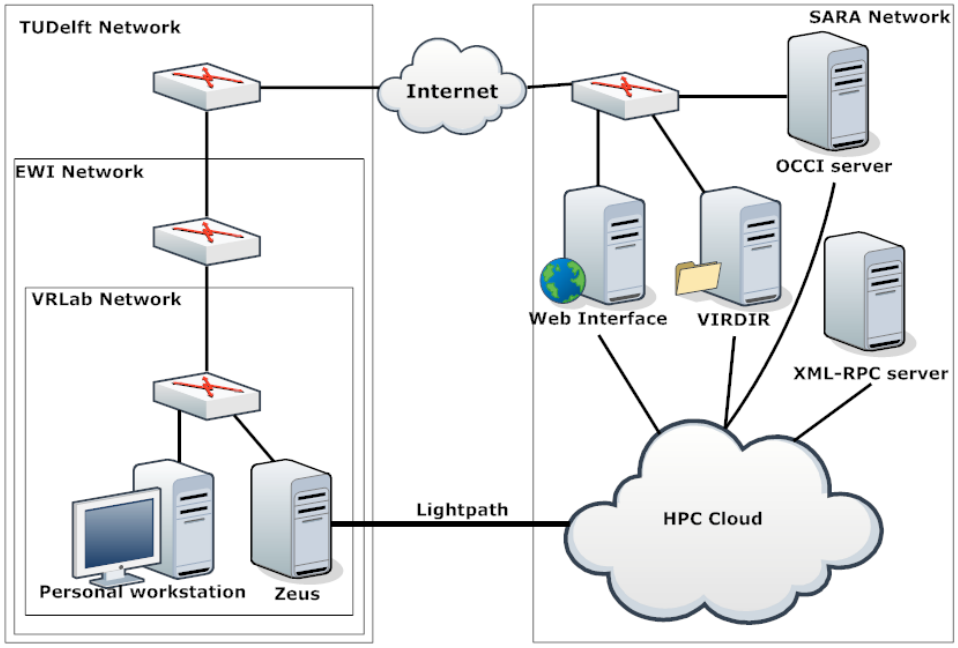
Na verloop van tijd werd de dienstverlening uitgebreid en gingen ook andere universiteiten en onderzoeksinstituten gebruik maken van de voorzieningen bij SARA. SARA's dienstverlening beperkte zich niet langer tot uitsluitend dataverwerking, maar omvatte ook rekentijd op supercomputers, dataopslag, netwerken en visualisatie.

Sinds 1984 vervult SARA de rol van nationaal supercomputercentrum voor het wetenschappelijk onderzoek en onderwijs in Nederland. Vanaf dat moment nam de supercomputerdienstverlening een grote vlucht. In de loop der jaren is de rekenkracht van supercomputers sterk toegenomen en breidden de activiteiten zich uit naar steeds meer wetenschapsgebieden. In 1995 kreeg SARA een zelfstandige status.

Figuur 5.1: Logo SARA

5.2 Netwerk

Vanwege de prijs van de Enlighten Your Research 3 wedstrijd is er een speciale dataverbinding getrokken van de TUDelft naar SARA. Dit betekent dat er twee manieren zijn om SARA te benaderen, via het lichtpad en via het internet. De opstelling ziet er ongeveer uit zoals in figuur 5.2.



Figuur 5.2: Netwerk topologie

5.2.1 TUDelft

Bij de TUDelft wordt er gewerkt in het VRLab. Dit is een netwerk waar de servers van Interactive Visualization and Virtual Reality in staan. Ook worden workstations van de mensen die dit netwerk nodig hebben hier op aangesloten. Er zijn twee servers beschikbaar, Zeus en Hera. Zeus dient als de default gateway voor internet verkeer en bevat ook de opslag van alle gegevens. Hera wordt in dit verslag niet gebruikt.

5.2.2 SARA

Bij SARA staan verschillende services en devices waar mee verbonden kan worden. Ieder project is in een apart VLAN[1] gezet en zo kan iedereen binnen zijn eigen project werken zonder een gevaar te worden voor andere projecten. Het lichtpad is ook onderdeel van het netwerk binnen een VLAN

[1] Virtual Local Area Network

om het onmogelijk te maken om andermans projecten zo te benaderen. Om gebruik te maken van het lichtpad moet dit dus aan het desbetreffende project gekoppeld worden bij SARA. Zo is het mogelijk om via dat lichtpad met interne services te werken. Voor dit project is er alleen gebruik gemaakt van de VIRDIR en HPC Cloud.

Het lichtpad dat dit project ter beschikking had werd met een ander project gedeeld. Hiervoor moeten er twee verschillende VLANs over dit lichtpad verzonden worden. De opstelling kan hier echter nog niet mee overweg en er is behoorlijk wat werk voor nodig om dit te laten werken. Om het te laten werken moet de veiligheid van andere projecten gegarandeerd kunnen worden. Er zijn verscheidene verzoeken gedaan om dit op te lossen maar hier wordt mee gewacht. Dit komt omdat het andere project bezig is met een verandering van de IP range die gebruikt wordt en als hier een besluit over genomen is worden alle aanpassingen gemaakt. Tot die tijd heeft dit andere project wel het lichtpad gereserveerd en hierdoor is het niet mogelijk geweest om het lichtpad te gebruiken voor dit project. De ontwerpen houden er rekening mee dat dit wel gaat gebeuren.

5.3 VIRDIR

VIRDIR staat voor Virtual Directory en is het file storage systeem dat bij de SARA HPC cloud hoort. De VIRDIR is ingedeeld in twee onderdelen, één deel bestaat als opslagruimte en de rest is gereserveerd als ruimte die ingenomen kan worden door draaiende Virtual Machines. Het opslag gedeelte kan op een Virtual Machine gemount worden indien deze op de HPC cloud draait. De opslag ruimte bevat twee folders, de Backup folder waar door SARA een backup van gemaakt wordt en de Scratch folder waarvan de data niet herstelbaar zijn indien deze verloren gaan. De VIRDIR is in totaal 400 TB waarvan dit project initiëel 2 TB beschikbaar had. Deze 2 TB was opgedeeld in 1 TB voor opslag en 1 TB als diskspace voor VM's. Dit was erg weinig en op verzoek is het totaal naar 5 TB vergroot waarvan 2 TB voor data en 3 TB voor disk images.

5.4 HPC Cloud

De HPC cloud is een hardware computer service die aangeboden word door SARA. Het bestaat uit 19 nodes. Iedere node heeft 32 Intel 2.13 GHz cores en 256 Gbyte RAM geheugen. Dit zorgt voor een totaal systeem van 608 cores en 4.75 TB RAM geheugen. Om met de HPC cloud te werken moet er een account verkregen worden en een toezegging van processor tijd. De initiële hoeveelheid tijd die gevraagd was was 2000 core uren maar dit is in het verloop van het project verhoogd naar 5000 (het normale basispakket).

Op de cloud kunnen virtual machines gestart worden. Hier kan dan als een normale computer mee gewerkt worden. Bij het starten van een virtual machine moeten een aantal instellingen gemaakt worden. Zo moeten de netwerk interfaces aangegeven worden (intern netwerk en extern netwerk) en ook welk disk image gebruikt wordt.

De cloud wordt gemanaged met OpenNebula[2]. dit is het systeem dat de urenregistratie doet van de users, templates opslaat, VNC clients kan starten op de VMs en nog meer. Het is het framework waarmee gewerkt wordt om met de cloud te werken.

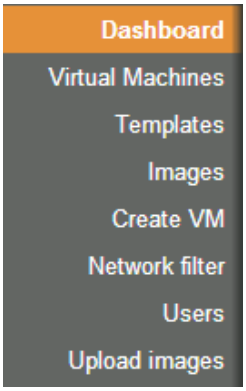
5.4.1 Aanstuurmethoden

Er zijn een aantal manieren om met de cloud te werken. Ten eerste kan de webinterface[3] gebruikt worden. Ook is het mogelijk om de communicatie zonder deze interface te doen en in plaats daarvan gebruik te maken van XML-RPC[4] of OCCI[5].

Webinterface

Figuur 5.3: Web based UI voor HPC Cloud

[2]Framework wat de HPC Cloud aanstuurt
[3]https://ui.cloud.sara.nl/
[4]XML-Remote Procedure Call
[5]Open Cloud Computing Interface: occi-wg.org



Figuur 5.4: Menu van de website

De webinterface, te zien in figuur 5.3, opent na inloggen op het dashboard. Hier is te zien hoeveel de HPC cloud gebruikt wordt en ook is er wat informatie te zien over het gebruik van de cloud door de gebruiker. Aan de linker kant zitten verschillende tabs die gebruikt kunnen worden bij het managen van de cloud.

De Virtual Machines tab geeft informatie over de VMs die actief zijn waaronder het interne en externe IP adres en de status van de VM. Ook is het mogelijk om een VNC window te starten vanuit deze tab.

Via de Templates tab kunnen nieuwe templates gemaakt en gestart worden. Templates bevatten alle randgegevens die nodig zijn om een VM te starten. Hier staat in hoeveel cores gebruikt worden, welke disk image gebruikt wordt en welke netwerk interfaces gestart worden.

All	ID	Owner	Group	Name	Registration time
	3117	eyr3_gh-tud-meijerh	eyr3_gh-tud	3Di Preprocessor 16	15:03:16 11/20/2012
	3118	eyr3_gh-tud-meijerh	eyr3_gh-tud	3Di Preprocessor 1	15:03:28 11/20/2012
	3222	eyr3_gh-tud-meijerh	eyr3_gh-tud	3Di Preprocessor 8	11:35:19 11/27/2012
	3292	eyr3_gh-tud-meijerh	eyr3_gh-tud	3Di Preprocessor 4	16:35:15 12/04/2012

Figuur 5.5: Template tab

Images bevat informatie over welke disk images er zijn. Hier kan ook gekozen worden of een image persistent is of niet. Als een image persistent is worden alle wijzigingen opgeslagen maar de disk image kan maar één keer actief zijn.

	1866	eyr3_gh-tud-meijerh	eyr3_gh-tud	3Di Preprocessor	10240	OS	15:00:43 11/20/2012		READY	0

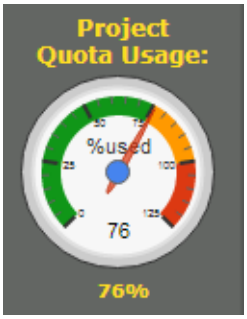
Figuur 5.6: Voorbeeld van een image in de Images tab

De Create VM tab bevat een handige wizard voor gebruikers om zelf images aan te maken. In deze wizard kan een standaard Windows of Ubuntu installatie gemaakt worden.

Network filters bevat firewall instellingen die gekozen kunnen worden in een template.

Users is een tab voor project administrators waarin ze rechten van users aan kunnen passen.

Upload Images heeft een upload tool waarmee .img bestanden naar de cloud geüpload kunnen worden.



Figuur 5.7: Meter die laat zien hoeveel procent van de tijd gebruikt is

In figuur 5.7 is een extra meter te zien. Deze meter geeft aan hoeveel procent van de beschikbare tijd gebruikt is. Indien er over de 100% heen gegaan wordt gebeurt er niets met de VMs, het is alleen een waarschuwing voor de gebruiker. Na de 125% wordt er door SARA contact opgenomen met de gebruiker en is het mogelijk om meer tijd te verkrijgen voor het project.

XML-RPC

XML-RPC is het protocol dat gebruikt kan worden van binnen het netwerk van SARA. Dit betekent dat er via het lichtpad gewerkt moet worden of via een VM bij SARA zelf. Met XML-RPC is het mogelijk om VMs te starten van binnen het netwerk zonder de webinterface te gebruiken. Dit is gewenst om automatisch VMs te starten.

Door het package opennebula-tools te installeren op ubuntu kan er met de commando's in dit pakket gewerkt worden. Zo kunnen er templates geïnstantieerd worden via server.one.template.instantiate waar server een open connectie is.

Aangezien de templates die gestart kunnen worden dezelfde templates zijn als die van de webinterface is dit zeer gebruiksvriendelijk. Zonder kennis van het systeem kan een template gemaakt worden op de website en het enige dat gebruikt hoeft te worden in het XML-RPC script is de template ID.

OCCI

OCCI staat voor Open Cloud Computing Interface. Dit lijkt op XML-RPC aangezien het ook in een script gezet kan worden maar heeft het voordeel dat dit via internet werkt. Het probleem is echter dat OCCI totaal om het OpenNebula framework heen gaat. Dit betekent dat de templates die

aangemaakt kunnen worden in de webinterface niet gebruikt kunnen worden. Er moet door de gebruiker een XML bestand aangemaakt worden met de gegevens van de VM die gestart moet worden. Deze XML bestanden worden met behulp van een HTTP actie naar de OCCI interface gezonden. HTTP GET vraagt informatie op, HTTP POST start nieuwe VM's op, HTTP PUT past bestaande VM's aan en HTTP DELETE kan VM's verwijderen of uitzetten.

Het grote nadeel hiervan is dat indien er een nieuwe testopstelling gemaakt moet worden er een nieuwe XML gemaakt moet worden voor de VMs die gestart gaan worden. Ook is het mogelijk om fouten te maken in de XML aangezien het hier mogelijk is om een variabel geheugen te reserveren, iets dat niet mogelijk is in de webinterface.

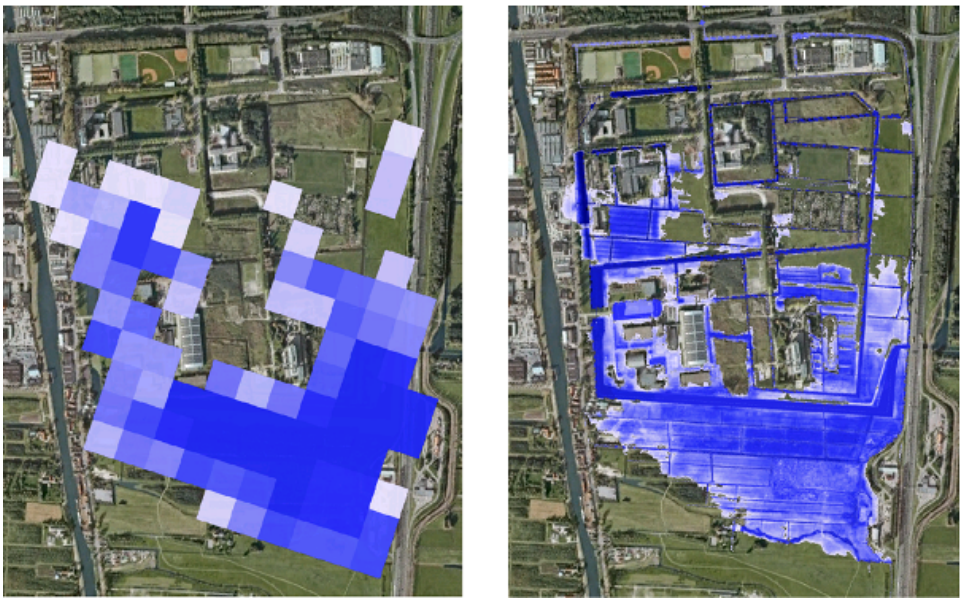
Hoofdstuk 6

Werking van preprocessing

6.1 AHN2

Voor het 3Di Waterbeheer project wordt er aan een overstromingssimulatie gewerkt. Hiervoor moet er een goede basis kaart zijn van de hoogte van het land. Huidige overstromingsprogramma's werken met een lage resolutie hoogte kaart van nederland. Het resultaat hiervan is te zien links op figuur 6.1. Het doel van 3Di Waterbeheer is om gedetailleerde gegevens aan te bieden en daarom is er een hogere resolutie nodig.

De AHN2 kaart is hier een goede uitkomst voor. AHN2 staat voor Actueel Hoogtebestand Nederland 2 en is een grote puntenwolk van heel Nederland. Deze puntenwolk bevat ongeveer tien punten per vierkante meter. Dit zorgt voor een hoge naukeurigheid in simulaties zoals te zien is rechts in figuur 6.1.

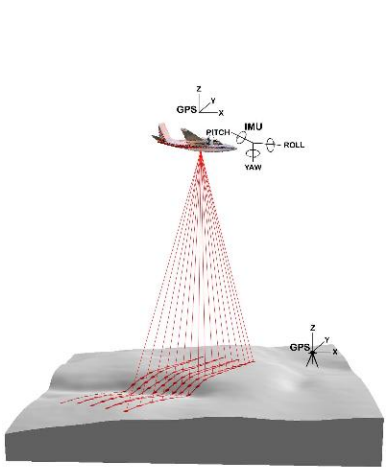


Figuur 6.1: Huidige methode tegenover AHN2

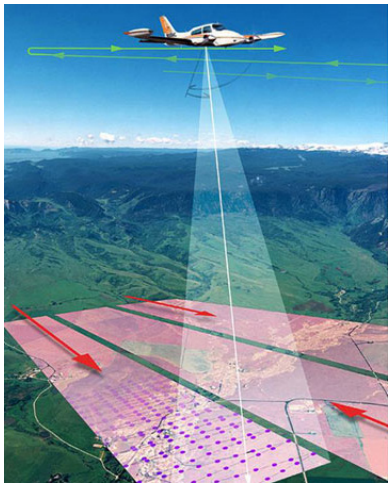
6.1.1 LiDAR

AHN2 word opgebouwd door middel van LiDAR[1]. LiDAR staat voor Light Detection And Ranging en is een meet methode die gebruikt word voor het genereren van een 3D aanzicht van bijvoorbeeld gebouwen. Dit kan ook gebruikt worden voor het maken van een kaart zoals te zien is in figuur 6.2b. Voor het maken van de AHN2 kaart zijn er scanners op vliegtuigen en helicopters gemonteerd.

[1]LIght Detection and Ranging: http://en.wikipedia.org/wiki/LIDAR



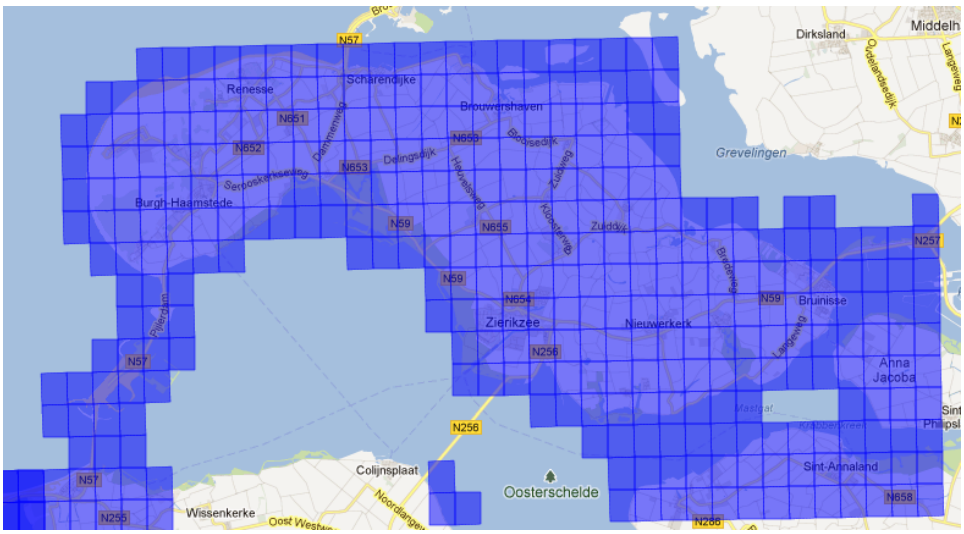
(a) Door gegevens van het vliegtuig te meten is het mogelijk de punten op de correcte cordinaten te geven



(b) Het LiDAR scan proces in een vliegtuig. De data word in lange stroken verzameld.

Figuur 6.2: Schematische weergave van vliegtuigen met LiDAR scanners

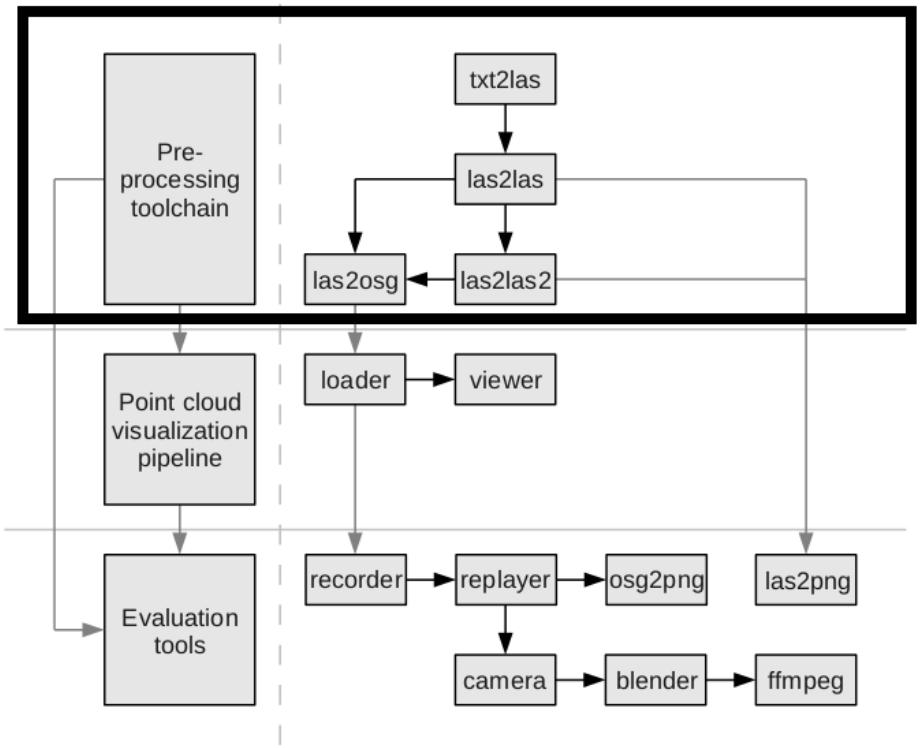
De vliegtuigen en helicopters meten informatie over hun eigen positie en voegen deze samen met de data die ingelezen worden uit het LIDAR apparaat. Hiervan wordt een nieuwe set gemaakt waarbij de punten die gemeten zijn gecorrigeerd zijn en dus hun werkelijke locatie hebben. Hier wordt dan een selectie uit geknipt van 1000 meter bij 1250 meter en dit vormt een tile in de kaart. Een layout van tiles is te zien in figuur 6.3.



Figuur 6.3: Schouwen-Duiveland ingedeeld in tiles

6.2 De voorbewerkingen

De AHN2 kaart wordt aangeleverd bij de TUDelft op harddisks die ieder een groot aantal tiles van de kaart bevatten. Deze tiles moeten verwerkt worden om deze goed te kunnen visualiseren. Het originele formaat waarin deze tiles binnen komen is een tekstbestand met alle punten, ook wel XYZ file genoemd. Dit moet dan door de preprocessing toolchain[2], te zien in Figuur 6.4, gaan om uiteindelijk goed gevisualiseerd te kunnen worden.



Figuur 6.4: De prototype toolchain van het hele process. Alleen de preprocessing toolchain wordt in dit project uitgevoerd

6.2.1 txt2las

De preprocessing toolchain begint met de txt2las functie. Deze functie zet een XYZ bestand om in een LAS bestand. Dit is de eerste stap voor de AHN2 kaart aangezien de brondata van deze kaart binnen komen in deze XYZ bestanden. Door het om te zetten wordt verzekerd dat de software ook met andere, gestandaardiseerde, opties overweg kan.

[2]Visualization on a Budget for Massive LiDAR Point Clouds, Author Berend Wouda

XYZ bestand

De XYZ file is het bestand waar alle punten in staan. Deze staan op de volgorde waarop deze ingescand zijn en niet makkelijk georganiseerd voor visualisatie. Het tekstbestand staat vol met regels van X, Y en Z coördinaten. Er is ook geen vast gedefinieerde manier waarop de cordinaten aangeleverd worden. Het is mogelijk dat waardes met een komma gescheiden staan, maar dit hoeft niet het geval te zijn. Dit betekent dat er een nieuw script gemaakt moet worden indien er een nieuwe kaart gebruikt wordt.

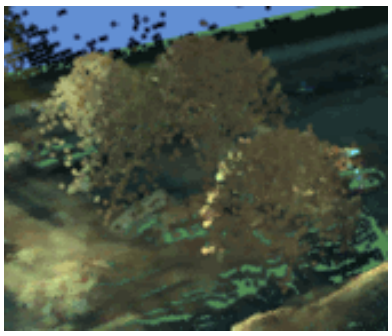
```
Voorbeeld van XYZ bestand inhoud.

57990.24,407334.75,-1.39
57990.75,407335.33,-1.09
57994.24,407339.47,-0.75
57985.63,407328.37,2.03
57986.27,407329.08,2.57
```

LAS bestand

Het LAS (LASer File Format) formaat is gemaakt als vervanging voor de ASCII files die origineel gebruikt werden om data van puntenwolken over te zetten. Het is voornamelijk voor puntenwolken die gemaakt zijn met LiDAR scanners maar werkt voor iedere vorm van puntenwolk. Omdat dit formaat vaker gebruikt wordt is dit een aantrekkelijker formaat om te gebruiken in de voorbewerking. Hierdoor wordt, zoals te zien is op Figuur 6.4, de XYZ file omgezet naar een LAS bestand (txt2las).

6.2.2 las2las

Het is ook mogelijk om in een LAS bestand RGB waardes toe te voegen aan een punt. Hierdoor is het mogelijk om bijvoorbeeld een luchtfoto uit te lezen en de X en Y waardes van de punten te matchen met punten uit het LAS bestand. Hier wordt een nieuw LAS bestand aangemaakt waar ieder punt een extra kleur waarde heeft. Dit proces gebeurt in de las2las functie van de preprocessing toolchain.

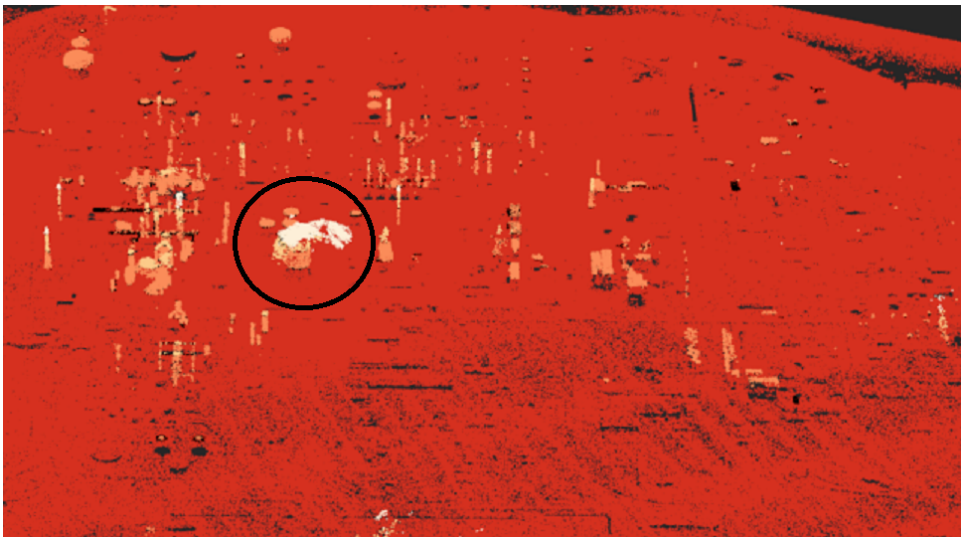


Figuur 6.5: Een luchtfoto die gemaakt is tijdens werkzaamheden en een LiDAR scan die hierna gemaakt is. Het resultaat is bomen met de kleur van zand

6.2.3 las2las2

De luchtfotos zijn echter niet op hetzelfde moment gemaakt als de LiDAR scan. Hierdoor ontstaan er inconsistenties

zoals op figuur 6.5 te zien is. Om dit probleem te verhelpen zijn er meer gegevens die toegevoegd kunnen worden die helpen bij het identificeren van de data. Zo kan het handig zijn om de hoogte van punten te zien in relatie tot andere punten. Een voorbeeld hiervan was te zien in het industrie terrein van Terneuzen in figuur 6.6. Hier was de wolk van de koeltoren niet te zien met een normale luchtfoto omdat de wolk de kleur kreeg van het onderliggende parkeer terrein. Door een hoogte schema toe te voegen is dit probleem opgelost. Dit gebeurt in de las2las2 functie



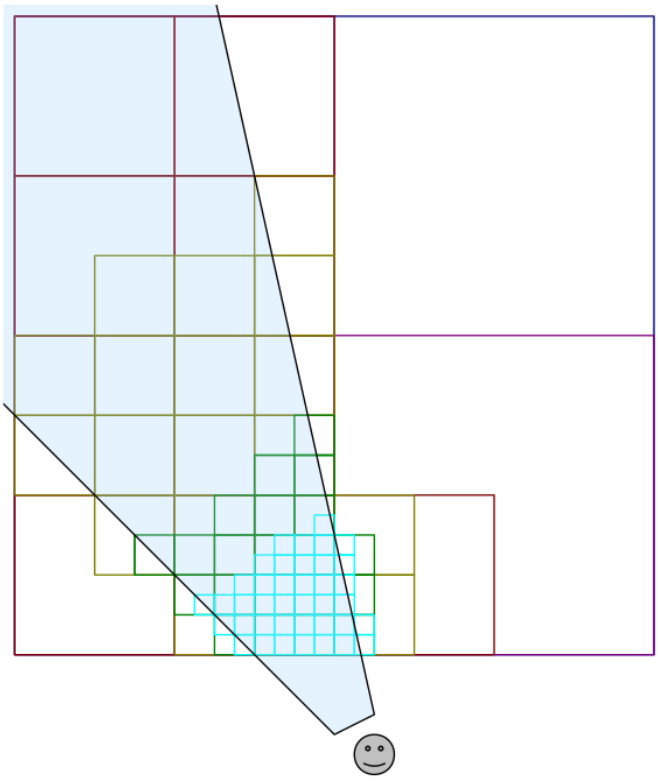
Figuur 6.6: 3D rendering van Terneuzen, Zeeuws-Vlaanderen. De witte vlek komt van de wolk van een koeltoren.

6.2.4 las2osg

De laatste stap van het preprocessen is om het gewenste LAS bestand om te zetten naar een OpenSceneGraph bestand. Dit gebeurt in de las2osg functie. OpenSceneGraph is het programma dat gebruikt wordt om de data te visualiseren en werkt via OpenGL. Het visualiseren is echter heel erg intensief voor de computer als alle duizenden punten gevisualiseerd moeten worden. Hiervoor wordt een Level of Detail file structuur gebruikt.

Level of Detail file structuur

Level of Detail is een manier die gebruikt wordt bij het visualiseren van grote data bestanden. Het heeft geen zin om meer punten in een datatype te hebben dan er op het scherm getoond kunnen worden. Hiervoor wordt er dus een lagere resolutie versie gemaakt van een tile. Ook wordt de tile zelf onderverdeeld in kleinere handelbare files. Op figuur 6.7 is te zien hoe dit werkt indien er schuin over een landschap gekeken wordt.



Figuur 6.7: Indeling van dynamisch level of detail. Ieder vakje is een file.

6.3 Preprocessing

De preprocessing toolchain die beschreven staat in Figuur 6.4 heeft een python implementatie op één enkele computer. Het proces dat beschreven staat in hoofdstuk 6.2 kan gestart worden door een python script te starten. Dit script leest een XML configuratie bestand uit waarin aangegeven staat welke bestanden gebruikt worden.

6.3.1 Python script

Het python script, genaamd batch_mp.py, zorgt voor de hele preprocessing toolchain. Zodra deze opgestart wordt begint deze met het parsen van de meegegeven opties en parameters. Hieruit worden globale variabelen ingesteld welke aanpassingen doen op het preprocessing proces. Zo kan er met -R gespecificeerd worden dat alle tussenstappen verwijderd worden zodra deze niet meer gebruikt hoeven te worden. Ook is het mogelijk om een configuratie bestand mee te geven. Dit zorgt ervoor dat het makkelijker is om het commando aan te roepen.

Na het parsen van de opties zal, indien deze is meegegeven, de configuratie XML geopend worden. Deze bevat de bronlocatie van de tiles, bronlocatie van de luchtfotos en de locatie waar de uiteindelijke files moeten komen. Ook staat hier een reguliere expressie van welke bronbestanden gebruikt moeten worden. Hierna worden de folders aangemaakt op de locatie waar de resultaten moeten komen.

De volgende stap is dat er voor ieder bronbestand een XML file aangemaakt wordt. Deze XML file bevat informatie over het gebruikte bronbestand zoals minimum en maximum X, Y en Z coördinaat welke gebruikt worden om de goede luchtfoto te gebruiken. Hierna wordt voor ieder bronbestand de prepocessing toolchain uit hoofdstuk 6.2 doorlopen. Dit duurt ongeveer dertig tot veertig minuten.

Hoofdstuk 7

Distributie software

In dit hoofdstuk wordt duidelijkheid gecreëerd over het framework dat gebruikt is voor de distributie met een uitleg van de werking hiervan.

7.1 Celery

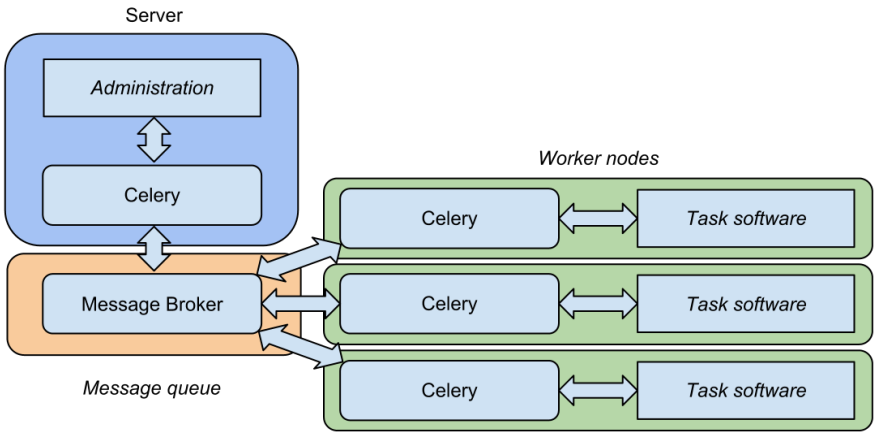
Celery is een framework dat gebruikt kan worden in Python en zorgt voor een makkelijke asynchrone distributie van taken over verschillende computers. Een functie kan makkelijk in een Celery task omgezet worden door een task decorator. Ook is Celery al sporadisch gebruikt binnen de Interactive Visualisation and Virtual Reality groep en is het gewenst om hier meer van te weten. Hierdoor werd aangeraden om Celery te gebruiken voor dit project.

Celery werkt door op een computer te draaien en op deze computer taken uit te voeren. Deze computer is een Celery worker node. De worker node start op met een vooraf bepaalde folder. Deze folder bevat de Celery tasks die uitgevoert kunnen worden. Om een stuk code in een Celery task te veranderen moet er een task discriptor toegevoegd worden aan de taak. De taak is dan met een stuk Celery code aan te roepen en zal hierna door Celery afgehandeld worden.

Celery heeft een extern programma nodig om messages te versturen en te ontvangen. Dit wordt een message broker genoemd. De message broker is de kern van Celery en zoals in figuur 7.1 te zien is zitten alle Celery instanties, zowel code als workers, aan een message broker gekoppeld. De message broker geeft door welke taken gestart worden en Celery workers moeten hier taken uit halen. Zonder een message broker zal een Celery worker node niets doen.

Zodra een Celery worker node draait gaat deze taken aanvragen van de message queue van de message broker. Het aantal taken dat tegelijk gestart kan worden is gelijk aan de concurrentie waarmee de Celery node opgestart wordt. Dit is standaard gelijk aan het aantal processor cores in de computer

of, in het geval van een VM, gesimuleerd wordt.



Figuur 7.1: Schematische weergave Celery

7.1.1 Message Broker

De message broker die gebruikt is, is RabbitMQ. RabbitMQ is een opensource message service. Dit is een middelware programma dat gebruikt wordt om messages tussen services te sturen. Het implementeert het AMQP (Advanced Messaging Queuing Protocol) en is geschreven in Erlang.

RabbitMQ is failure complete. Dit betekend dat deze overweg kan met het plotseling stoppen van workers zonder dat er data verloren gaat of dat deze taken niet uitgevoerd worden.

7.1.2 Backend Database

De Backend Database die gebruikt is voor resultaten is Redis. Redis is een key-value dictionary systeem. Redis is net als RabbitMQ failure complete. Echter is de hele Redis dictionary in memory opgeslagen. Dit kan voor problemen zorgen bij plotselinge terminatie van het programma.

RabbitMQ kan ook als backend Database gebruikt worden echter slaat deze geen resultaten op. Een resultaat wordt als message behandeld en kan dus maar een keer opgehaald worden.

Hoofdstuk 8

Increment 1: Creatie van de omgeving

Dit hoofdstuk beschrijft de creatie van de omgeving. Hierin zijn verscheidene virtual machines gemaakt voor het uiteindelijke systeem en om de Celery setup te testen.

8.1 Analyse

8.1.1 Requirements

Om goed met het systeem te kunnen werken zijn er een aantal VMs nodig. Zo is er een server nodig en ook een client waar de volledige preprocessing functionaliteit op staat.

De server moet een RabbitMQ en Redis distributie draaien. Om te voldoen aan eis s3 is dit een Ubuntu operating systeem.

De client moet de volledige preprocessing toolchain uit kunnen voeren. Ook is het handig als dit een minimalistische installatie is omdat deze VM op de cloud gaat draaien. Ieder programma dat niet opgestart hoeft te worden bespaart namelijk hier weer tijd. Dit is ook weer een Ubuntu installatie conform met eis p5.

Het eindresultaat zijn twee functionele VMs. Om de functionaliteit hiervan te testen wordt er een simpel Celery test programma gebruikt. Ook wordt de preprocessing toolchain uitgevoerd op de client.

8.2 Ontwerp

Het ontwerp omvat het ontwikkelen van een server VM en een client VM. Deze worden een server welke de taken zal distribueren en een client welke de taak uit zal voeren.

8.2.1 Server VM

De server VM bevat een standaard Ubuntu installatie. Hier moet Celery, RabbitMQ en Redis opgezet worden. RabbitMQ en Redis moeten alle twee gestart worden bij het opstarten van de VM.

8.2.2 Client VM

De client VM is gebaseerd op de laatste versie van Ubuntu server om extra functionaliteit zo veel mogelijk te vermijden. Hier moet Celery op geïnstalleerd worden en ook alle benodigde packages voor de preprocessing toolchain.

8.2.3 Deployment

De initiele test deployment wordt compleet op Zeus gesimuleerd.

Server

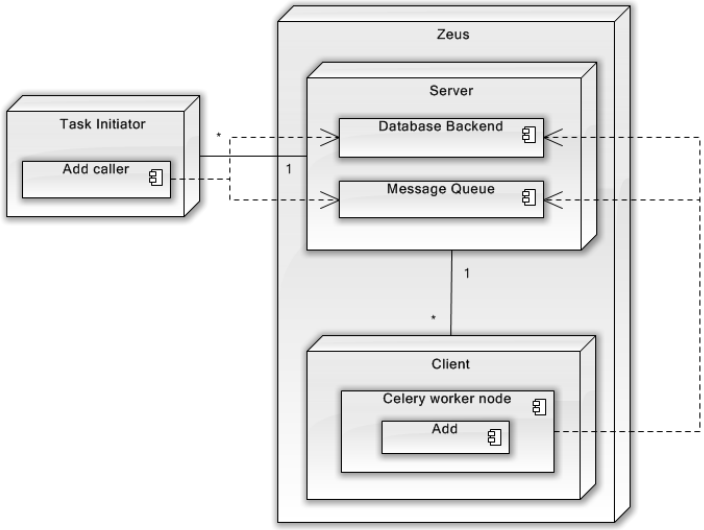
De server moet de twee backend benodigdheden van Celery draaien maar hoeft zelf niets te weten van Celery of de taak die uitgevoerd wordt.

Client

De client moet de taak hebben en een Celery node hebben om deze taak uit te voeren. De Celery node koppelt bij het opstarten aan de twee databases die draaien op de server.

Task initiator

Door te koppelen aan de twee databases die op de server draaien met een Celery object is het mogelijk om taken op te starten en resultaten uit te lezen.



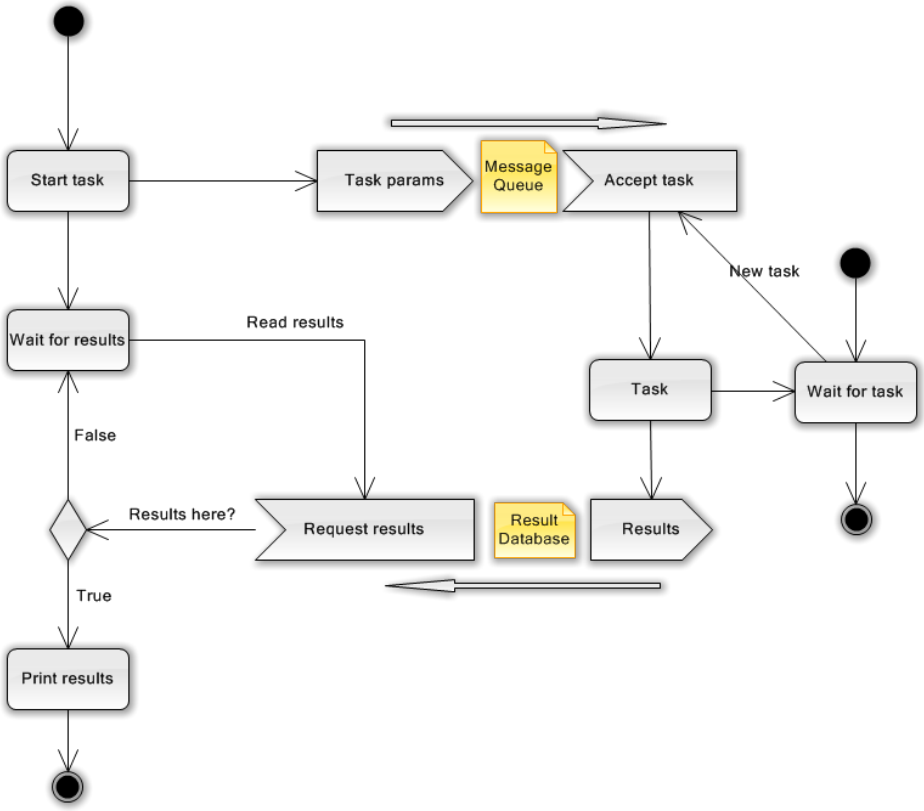
Figuur 8.1: Deployment diagram increment 1

8.2.4 Flow Chart

In de flow chart van figuur 8.2 staat de Add caller en de interactie met de Celery worker node beschreven. De Celery worker node staat aan de rechterkant en de Add caller staat aan de linkerkant.

Als de Add caller gestart wordt, begint deze met het aanroepen van de "start task" functie. Deze functie stuurt een message naar de message queue van Celery. Hier wacht deze op een node die de task op kan starten. De Add caller begint hierna met het wachten op resultaten in "Wait for results". Deze functie vraagt aan de result database of het resultaat al binnen is. Zodra dit binnen is wordt het antwoord geprint en stopt de functie.

De Celery worker node staat aan de rechterkant van figuur 8.2. Zodra deze wordt gestart gaat hij wachten op een taak van de Celery message queue. Zodra er een taak binnen komt zal deze node de desbetreffende taak uit gaan voeren. De resultaten worden in de result database gezet en de worker node gaat weer verder met het wachten op nieuwe taken.



Figuur 8.2: Flow chart test creatie van omgeving

8.2.5 Class Diagram

Alhoewel deze zeer beperkt is is het wel handig om te weten wat Celery doet om een taak te maken. Neem bijvoorbeeld de code van een simpele Add functie.

```
@celery.task
def add(x, y):
    return x + y
```

Zodra een functie de task decorator (@celery.task) krijgt, wordt hier door Celery een nieuwe klasse van gemaakt. Deze nieuwe klasse bevat dan een groot aantal functies van de Celery.task klasse en de functie zelf word in de run() gezet. De naam van deze klasse kan zelf gekozen worden maar wordt, indien dit niet gedaan wordt, automatisch aangemaakt. Deze naam moet uniek zijn omdat er anders problemen ontstaan.

De add functie is gebruikt als task voor figuur 8.2.

8.3 Implementatie

Er zijn een tweetal VMs gemaakt voor dit increment, een server waar RabbitMQ en Redis op draaien en een client die een Celery worker node draait. Deze VMs draaien met behulp van QemuKVM op Zeus.

8.4 Bouw

8.4.1 Server

Als basis is een standaard Ubuntu installatie genomen. Hier is RabbitMQ, Redis en Celery op gezet. Deze server draait op Zeus met behulp van QemuKVM.

Portforwarding

Om gebruik te kunnen maken van de Server moeten er een aantal poorten open staan. Dit zijn de poorten voor RabbitMQ en Redis. Er was echter een probleem met het forwarden van poorten voor deze server. Het probleem met het forwarden van poorten was dat QemuKVM standaard geen netwerk aanmaakt op Zeus. Dit betekent dat alhoewel het lijkt dat er een netwerk is met Zeus, indien er van binnen de VM gekeken wordt, dit niet het geval is. Als er vanuit Zeus gekeken wordt is er namelijk geen netwerk te zien, het enige dat zichtbaar is, is een applicatie. Door deze structuur is het dus nodig om een speciale manier van portforwarding te maken.

Het maken van een nieuwe netwerk bridge kan voor problemen zorgen. Indien er een fout gemaakt wordt met het koppelen van netwerken kan de server zijn eigen routering wissen. Dit zorgt ervoor dat Zeus niet meer beschikbaar is voor de buitenwereld. Alhoewel deze optie wel het beste was werd deze afgeraden door de systeem administrator aangezien er dan erg veel ingesteld moest worden wegens IP ranges.

Na dit uitgezocht te hebben bleek er ook een port forwarding commando in QemuKVM te zitten. Door de

```
redir
```

parameter te gebruiken is het mogelijk om poorten van Zeus te forwarden naar de server.

8.4.2 Client

Als basis is een Ubuntu server installatie genomen. Dit is een minimale versie van Ubuntu waar alleen de kern op staat. Hier zijn de benodigde packets op geïnstalleerd zoals een compiler en python. Ook is er voor gebruiksgemak een UI op geïnstalleerd. Er zijn twee accounts, één voor Celery en één account voor de user (genaamd user).

Voor de preprocessing pipeline zijn er een groot aantal libraries bijgezet waaronder OpenSceneGraph, OSGSwig, GDal en GeoTiff. Deze libraries zijn nodig voor het maken van de OSG bestanden die gebruikt worden bij het visualiseren.

Bij het opstarten van de VM wordt er automatisch een Celery node opgestart die evenveel taken tegelijk aan kan hebben als er processoren zijn. Deze Celery node verbindt automatisch met de server VM van het vorige increment.

Ook wordt de VIRDIR van SARA automatisch gemount bij het opstarten van een VM. Deze komt in de /data partitie te staan en hier kunnen brondata in gezet worden.

Als laatste worden er een aantal environment variabelen goed gezet bij het opstarten. Deze zijn nodig om een preprocessing taak op te kunnen starten.

8.4.3 Celery code

Om een verbinding te maken met Celery moet er een Celery object aangemaakt worden. Dit object is nodig bij het aanmaken van de taak zelf. Er zijn drie parameters nodig. De eerste is een soort van namespace voor de taken die aangemaakt worden. Hierna wordt de backend connectie ingevuld en de broker connectie ingevuld.

```
from Celery import Celery

celery = Celery('tasks', backend='redis://zeus:1235',
                         broker='amqp://zeus:1234')
```

De Celery taak die gebruikt is was de volgende.

```
@celery.task
def add(x, y):
    return x + y
```

Door de taak te starten op de initiator wordt er een simpele optelsom uitgevoerd op een van de clients. De optelsom is te starten nadat een connectie met de server gemaakt is met het Celery object. Door de add som uit te voeren met een asynchroon commando uit de task discriptor wordt deze naar de broker gestuurd. Hierna gaat deze task naar een client die naar de broker luistert en die ook een task heeft met dezelfde naam.

Door

```
result = add.delay(3,5)
```

aan te roepen zal de optelsom 3+5 op een andere client ook uitgevoerd worden. Het antwoord wordt niet direct getoond aangezien het een asynchrone

taak is. Om te kijken of het antwoord al bekend is kan result.ready() aangeroepen worden en deze functie geeft een boolean terug. Ook is het mogelijk om te wachten op het resultaat met een timeout door

```
result.get(timeout=4)
```

maar dit verandert de taak in een synchrone taak aangezien de initiator een periode zal wachten op het antwoord.

8.5 Test

De tests die binnen dit increment uitgevoerd zijn, dienen om duidelijkheid te creëeren over de werking van Celery en de werking van de initiële preprocessing toolchain.

8.5.1 Celery Test

Om te testen of de Celery setup werkt is de volgende test uitgevoerd. De Celery setup is belangrijk voor het functioneren van het complete systeem en is hierdoor van groot belang. De server VM is opgestart met Redis en RabbitMQ. Er is een tweede VM opgestart die als Celery worker node dient. Beide VMs beschikken over de Add task.

Op de server VM wordt de result = add.delay(2,2) functie uitgevoerd. De verwachting is dat het resultaat dat hieruit komt 4 is. Er wordt meteen een bericht gestuurd als de message bij de message broker aangekomen is.

```
>>> result = add.delay(2,2)
```

Ondertussen is er een worker node aan het luisteren. Deze heeft door dat er een taak gestart wordt en begint met het uitrekenen van deze taak.

```
Got task from broker: tasks.add[69a648b2-21c2-43b2-a708-bf7c08
1107ba]
Task tasks.add[69a648b2-21c2-43b2-a708-bf7c081107ba] succeeded
in 0.00395798683167s: 4
```

De functie is uitgevoert op de worker en de taak is succesvol geweest, ook is het antwoord 4 zoals verwacht. Hierna is het resultaat te zien in de server door dit op te vragen met result.get().

```
>>> result.get()
4
```

Zoals te zien is komt hier inderdaad het verwachte resultaat uit.

Om meer te leren over de werking van Celery is er een tweede Celery worker opgestart. Deze tweede Celery worker heeft echter een andere add functie. Deze vermenigvuldigt in plaats van dat deze optelt. Door 2 + 2

meerdere malen uit te voeren kunnen we niet weten dat één van de twee workers corrupt is. Door een andere optel som zoals 3 + 3 door te sturen is dit echter wel te meten.

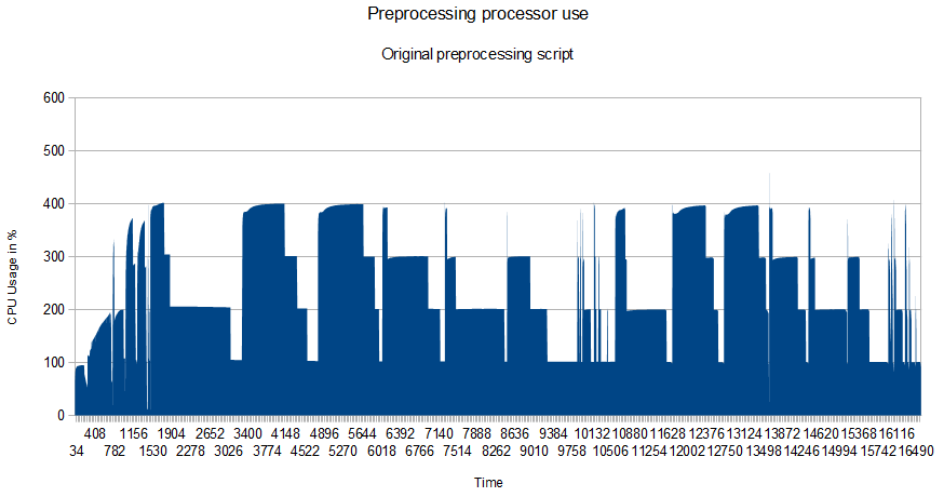
```
Task tasks.add[ca2edc17-27a9-40f2-bba2-198171c52fd6] succeeded
in 0.000809192657471s: 9
```

De reden dat de corrupte functie ook de berekening maakt is omdat deze dezelfde naam heeft. Dit is de reden dat de namen uniek moeten zijn. Dit zorgt er echter wel voor dat er goed gekeken moet worden naar versies van de software die draait op de clients. Als dit niet gedaan wordt kunnen de foute resultaten berekend worden. Helaas zijn er geen bestaande unit tests voor het totale preprocessing systeem en dit kan hier ook voor problemen zorgen. Om dit op te lossen zal eis P4 geïmplementeerd moeten worden.

8.5.2 Processor test

Als test van de preprocessing toolchain is er een sectie van de AHN2 kaart gepreprocessed. Door de brondata van een sectie van de kaart en de luchtfotos op de VIRDIR te zetten kunnen deze direct gebruikt worden in de cloud. Het resultaat word ook weer op de VIRDIR gezet en kan hiervan bekeken worden.

Om inzicht te krijgen over het huidige systeem is er een log gemaakt van de processoractiviteit op de VM. Dit log bevat de processor activiteit van de top 10 programma's die eens in de 5 seconde weggezet is in een file. De VM heeft met 4 cores gedraaid dus is 400% belasting het maximale wat behaald kan worden. Wegens de werking van de initiële code wordt er verwacht een trap te zien. De code start namelijk vier taken tegelijk en als deze alle vier klaar zijn worden pas nieuwe taken opgestart.



Figuur 8.3: Processor use

Uit de grafiek in Figuur 8.3 is af te lezen dat er gemiddeld maar 2 en een halve core actief is. Dit is een efficiëntie van 62.5%. Er kan hier al een grote verbetering gemaakt worden.

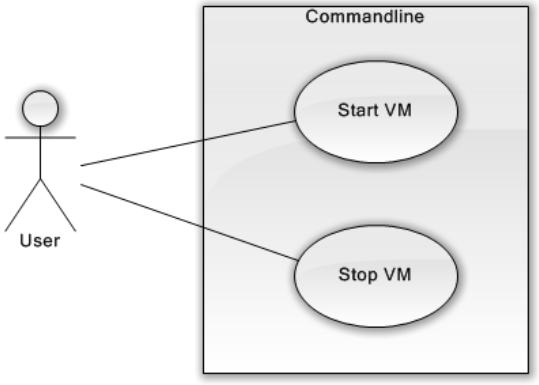
Hoofdstuk 9

Increment 2: Remote opstarten van cloud virtual machines

Dit hoofdstuk beschrijft het tweede increment. Hierin is zijn twee implementaties gemaakt voor het starten van VMs op de cloud. Het eindresultaat van dit increment is een methode om VMs op de cloud op te starten waar hierna het preprocessing proces is op te starten.

9.1 Analyse

Om goed te kunnen werken met de cloud is er een manier nodig om VMs op afstand op te starten. Dit is namelijk niet via de webinterface te scripten. De makkelijkste manier is XML-RPC en deze werkt van binnen uit het SARA netwerk. Wegens langdurige problemen met het lichtpad leek dat steeds minder een optie. Hierdoor werd het gewenst om ook via OCCI VMs op te kunnen starten. Beide implementaties zijn ontworpen en geïmplementeerd.



Figuur 9.1: Use Case Diagram

De twee taken die in Figuur 9.1 te zien zijn, zijn nodig om de cloud aan te sturen. Deze twee taken zullen geïmplementeerd worden met de XML-RPC methode en ook met de OCCI Methode.

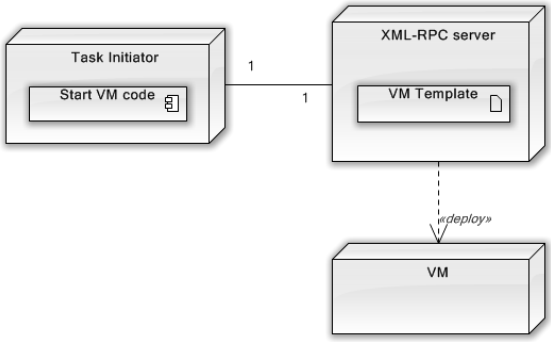
9.2 Ontwerp

9.2.1 XML-RPC

Om VMs op te starten in de cloud zijn er twee manieren. De eerste manier is door middel van XML-RPC. Zoals eerder beschreven is werkt XML-RPC via het interne netwerk van SARA. Ook is het mogelijk om het lichtpad te gebruiken. XML-RPC kan gebruik maken van templates gemaakt binnen de webinterface.

deployment

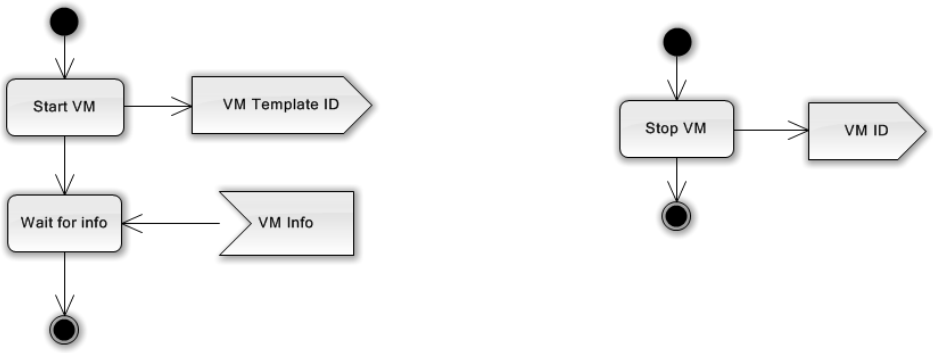
Zoals te zien is in figuur 9.2 zijn er 3 nodes van belang in dit increment. Als eerste is er weer een node nodig waarvandaan alle taken gestart kunnen worden, de initiator. Hiernaast is er ook nog de XML-RPC server die bij SARA draait. Deze is van groot belang voor het kunnen opstarten van VMs. Als laatste is er de VM node die opgestart wordt door de XML-RPC server.



Figuur 9.2: Deployment Diagram

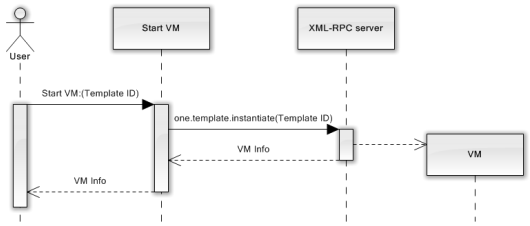
flow chart en sequence

De twee taken die er zijn, zijn uitgewerkt in deze state machines en sequence diagrammen. De linker taak van Figuur 9.3 is het starten van een VM en de rechter taak is het stoppen van een VM.



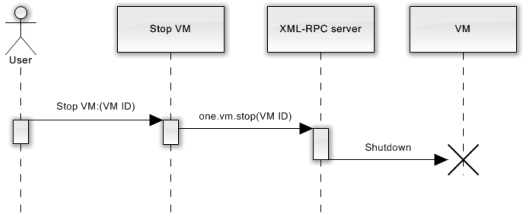
Figuur 9.3: flow chart Increment 2: XML-RPC

Bij het starten van een VM wordt eerst het start commando weg gestuurd. Dit commando bevat het ID van een template die eerder gemaakt is. Hierna wacht de functie en print deze de resultaten uit die de XML-RPC server heeft terug gestuurd. Hier staat onder andere het ID van de VM die opgestart is in Figuur 9.4.



Figuur 9.4: Sequence start VM Increment 2: XML-RPC

Bij het stoppen van een VM word alleen maar het stop commando weg gestuurd naar de XML-RPC server. Hierna terminate de XML-RPC server de VM (figuur 9.5).

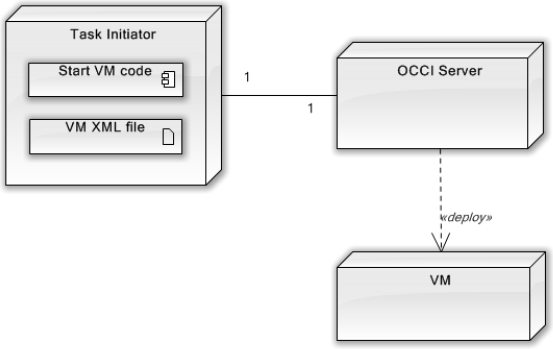


Figuur 9.5: Sequence stop VM Increment 2: XML-RPC

9.2.2 OCCI

Deployment

Het deployment diagram is wel anders als dat van XML-RPC. Zo word er niet meer gewerkt met de XML-RPC service maar nu met de OCCI server. De OCCI server heeft geen kennis van de templates die gebruikt worden binnen OpenNebula en kan hier dus niets mee doen. Dit heeft tot gevolg dat de Task Initiator nu zelf beschikking moet hebben over de standaard informatie van de VM. Deze staat in de VM xml file.

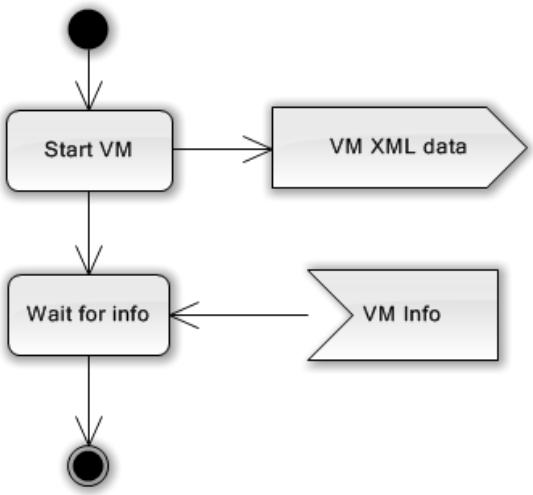


Figuur 9.6: Deployment Diagram OCCI

State

Het starten van een VM en het stoppen zien er weer redelijk hetzelfde uit aan states. Het grote verschil hier zit in de manier van verbinden met de server en de data die hierheen gestuurd wordt. De stappen zelf zijn hetzelfde als de stappen die gedaan worden in het XML-RPC gedeelte.

In Figuur 9.7 is te zien wat er moet gebeuren om een verandering door te geven via OCCI. De XML file wordt naar de server gestuurd en hier wordt ook in een XML antwoord op gegeven. Dit sturen gaat volgens een HTTP request.



Figuur 9.7: Flow chart OCCI

9.3 Implementatie

9.3.1 XML-RPC

Om een VM met XML-RPC op te starten is het nodig om eerst een verbinding op te zetten met de XML-RPC server van SARA. Deze is te bereiken via een proxy die in het VLAN beschikbaar is. Om door die proxy heen te komen is er een speciale klasse nodig die deze proxy connectie af kan handelen.

```
class ProxiedTransport(xmlrpclib.Transport):
    def set_proxy(self, proxy):
        self.proxy = proxy
    def make_connection(self, host):
        self.realhost = host
        h = httplib.HTTPConnection(self.proxy)
        return h
    def send_request(self, connection, handler, request_body):
        connection.putrequest("POST", 'http://%s%s' % (self.realhost,
                                                       handler))
    def send_host(self, connection, host):
        connection.putheader('Host', self.realhost)

def openConnection():
    p = ProxiedTransport()
    p.set_proxy('10.0.133.3:3128')
    server_url = 'http://one-xmlrpc.calligo.sara.nl:2633/RPC2';
    return xmlrpclib.Server(server_url, transport=p);
```

openConnection() opent een verbinding met de XML-RPC van SARA.

Met deze open connectie is het mogelijk om gebruik te maken van verscheidene standaard functies. Met deze functies kan gebruik gemaakt worden van de zelfgemaakte templates op de HPC Cloud. Door aan one.template.instantiate een template ID en een naam mee te geven wordt er een VM opgestart met de parameters die al eerder ingesteld zijn op de webinterface. one.vm.action kan verschillende acties uitvoeren op VMs waaronder "shutdown". Hiervoor moet deze actie mee gegeven worden en een ID van de VM waar deze actie op moet worden uitgevoerd. Beide commando's hebben een username en wachtwoord combinatie nodig.

9.3.2 OCCI

Om VMs te starten met OCCI is er wat meer werk nodig dan bij XML-RPC. De communicatie opzetten gaat echter wel een stuk makkelijker wegens een package in Ubuntu. Dit package is speciaal gemaakt om makkelijk met OCCI te kunnen werken. Met het commando

```
occi-compute --url https://occi.cloud.sara.nl --username
USER --password PASS create 3di-node.xml
```

wordt een VM opgestart met de eigenschappen die vermeld staan in 3di-node.xml. In deze xml moet informatie staan over de netwerken die gewenst zijn, het ID van het diskimage wat gebruikt moet worden en ook wat voor instance er opgestart moet worden. De informatie die hierin staat is vrijwel hetzelfde als een template wat gemaakt kan worden op de webinterface. De webinterface is echter makkelijker te gebruiken en hier zijn een aantal interne gegevens al bekend.

3di-node.xml

```
<COMPUTE>
        <NAME>3Di-ppOCCI</NAME>
        <INSTANCE_TYPE>small</INSTANCE_TYPE>
        <DISK>
                <STORAGE href="https://occi.cloud.sara.nl:443//storage/1866" />
        </DISK>
        <NIC>
                <NETWORK href="https://occi.cloud.sara.nl:443//network/0">
                        <ID>0</ID>
                        <NAME>internet</NAME>
                        <NETWORK_FILTER>229</NETWORK_FILTER>
                </NETWORK>
        </NIC>
        <NIC>
                <NETWORK href="https://occi.cloud.sara.nl:443//network/36">
                        <ID>36</ID>
                        <NAME>eyr3_gh-tud</NAME>
                </NETWORK>
        </NIC>
        <CONTEXT>
                <DATA>DATA1</DATA>
                <HOSTNAME>MAINHOST</HOSTNAME>
        </CONTEXT>
</COMPUTE>
```

Het opstarten van een VM met OCCI geeft een xml terug die de eigenschappen van de opgestarte VM bevat. Hier staat in op welke node de VM draait en welk ID deze heeft. Hetzelfde commando kan enkele keren herhaald worden om de gewenste hoeveelheid VMs op te starten.

Het afsluiten van VMs met OCCI gaat op ongeveer dezelfde manier. Door shutdown.xml aan te roepen en deze met, in plaats van create, de

update parameter in het commando te gebruiken zal de VM met in dit geval id 16432 zichzelf afsluiten.

shutdown.xml

```
<COMPUTE href='https://occi.cloud.sara.nl:443//compute/16432'>
<ID>16432</ID>
<STATE>SHUTDOWN</STATE>
</COMPUTE>
```

Om alle VMs af te sluiten op hetzelfde moment moet het shutdown commando voor ieder ID uitgevoerd worden. Alle IDs zijn met het list commando op te halen. Na wat parsen is het mogelijk hier alle VM IDs uit te halen.

```
Command = "occi-compute " + self.Arguments + " list "
id = []
xml = ET.fromstring(subprocess.check_output(Command, shell=True))
for instance in xml.findall('COMPUTE'):
    href = instance.get('href')
    id.append(href.split("/compute/")[1])
return id
```

9.4 Test

9.4.1 XML-RPC

Het testen van de volledige implementatie is lastig wegens problemen met de aansluiting van het lichtpad. Er zijn echter wel tests uitgevoerd vanuit de cloud zelf waarbij de initiator een VM in de cloud was. Hieruit blijkt dat de XML-RPC methode van VMs opstarten en aanpassen werkt zoals verwacht. De normale shutdown is echter een probleem. Deze wordt niet altijd goed uitgevoerd omdat de interne VM niets van het commando hoort. De enige oplossing is dan het Deleten van de desbetreffende VMs. Dit is echter niet mogelijk met XML-RPC.

9.4.2 OCCI

Het testen van OCCI was makkelijker omdat dit niet vanaf de cloud zelf gedaan hoefde te worden. VMs werden opgestart zoals het hoorde en er kon mee gewerkt worden. Er waren echter dezelfde problemen met het afsluiten als de problemen die bij XML-RPC voorkwamen. Hiervoor is hier een extra Delete implementatie gemaakt om de VMs te forceren dat deze afsluiten indien ze te lang doorgaan.

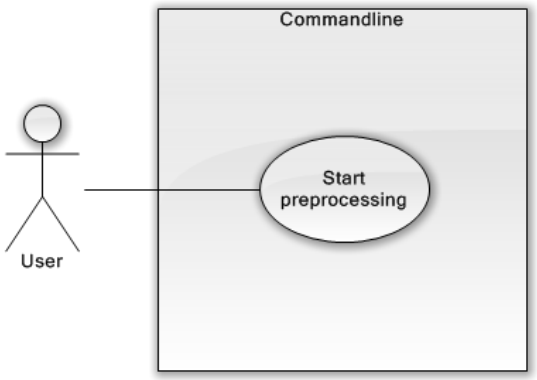
Hoofdstuk 10

Increment 3: Preprocessing omzetten in Celery task

In dit increment is de Preprocessing omgezet in een Celery task. Het doel hiervan is om de preprocessing op een goed gedistribueerde manier te laten werken.

10.1 Analyse

De preprocessing moet op dezelfde manier blijven werken. Er moet echter wel een onderscheid komen tussen tiles en hier moet de preprocessing op worden opgedeeld in plaats van de stappen waar op dit moment de code mee werkt. Dit moet allemaal automatisch gebeuren aangezien de enige actie die de gebruiker hoeft te ondernemen het starten van het preprocessing is.

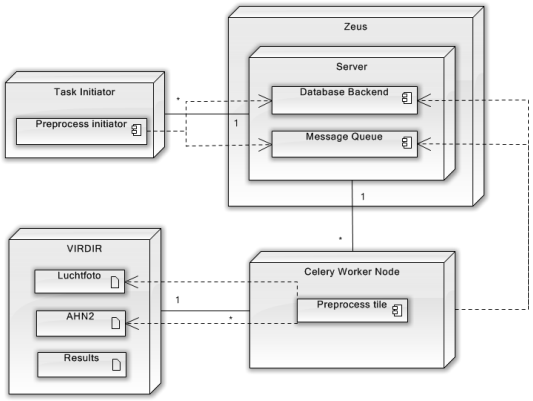


Figuur 10.1: Use Case Increment 3

10.2 Ontwerp

10.2.1 Deployment

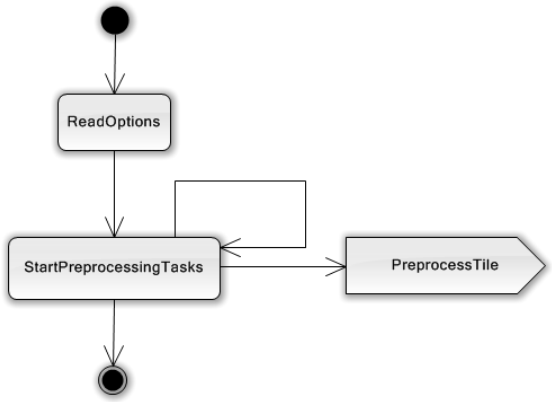
De volledige deployment, te zien in Figuur 10.2, bevat 4 nodes. De task initiator met de start preprocessing taak, de server welke draait op Zeus, de Celery worker node met de preprocessing Tile functie, en de VIRDIR waar de brondata op staan. De VIRDIR staat gemount op de Celery worker node.



Figuur 10.2: Deployment Increment 2

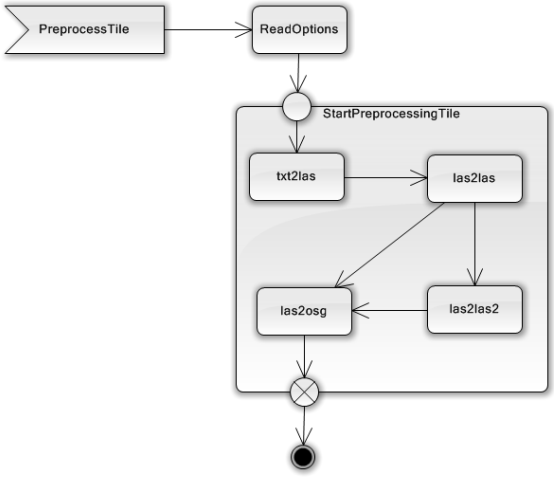
10.2.2 Flow charts

In Figuur 10.3 staat het verloop van de preprocessing initiator die gedraaid wordt op de Task initiator machine. Deze begint met het uitlezen en parsen van de opties en de configuratie xml. Hierna worden de files die geprepro-cessed moeten worden stuk voor stuk in een nieuwe taak omgezet. Deze nieuwe taak wordt gestart en als alle taken gestart zijn stopt de initiator.



Figuur 10.3: Flow Chart Preprocessing Initiator Increment 3

De preprocess Tile taak, te zien in Figuur 10.4 gaat van start op een Celery worker node zodra deze een taak ontvangt. Hier wordt begonnen met het uitlezen van de meegegeven parameters en opties die relevant zijn voor het preprocessen zelf. Hierna wordt er voor de desbetreffende tile alle stappen van de preprocessing Toolchain uitgevoerd. Het eindresultaat wordt in files weggeschreven op de VIRDIR.



Figuur 10.4: State Machine Preprocess Tile Increment 3

10.3 Implementatie

De bestaande preprocessing taak is ontleed in twee onderdelen. Als eerste de Task initiator, deze leest de opties en configuratie file uit en start aan de hand hiervan nieuwe taken. Als tweede is er de Tile preprocessor. Dit is de preprocessing taak voor een enkele tile. Deze moet werken als Celery task.

10.3.1 Task initiator

De task initiator is voor wat betreft code bijna hetzelfde als het originele preprocessing script. Het grote verschil zit hem in het feit dat deze, in plaats van een lijst van commando's af te gaan voor iedere stap van het preprocessen, hij alleen nog maar een lijst maakt met tiles. Deze lijst met tiles wordt dan afgewerkt en hiervan worden Celery tasks gestart. Deze Celery tasks hebben de opties en de desbetreffende tile die gepreprocessed gaat worden nodig als parameter. Als alle tiles gestart zijn is de task initiator klaar en kan deze afgesloten worden. De resultaten zullen op de VIRDIR komen te staan als het preprocessen klaar is.

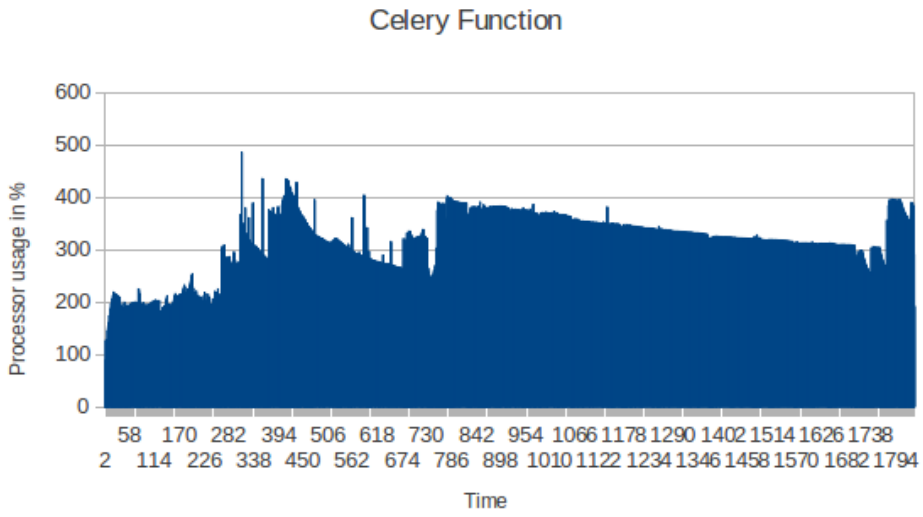
10.3.2 Tile Preprocessor

De tile preprocessor zal alle stappen doorlopen voor één enkele tile. Deze zal voor deze enkele tile de stappen uitvoeren die in het originele preprocessing script voor iedere tile uitgevoed werd. In het originele script werd ook eerst alles van een stap afgewerkt, dus werd van alle tiles eerst de las file gemaakt voordat er verder gewerkt werd. In de Tile Preprocessor zullen alle stappen voor één enkele tile doorlopen worden voordat de volgende taak aan de volgende tile kan starten. Dit zorgt ervoor dat er eerder onderdelen van de kaart gevisualiseerd kunnen worden.

Omdat er enkele taken tegelijk zullen draaien en omdat direct zodra een taak klaar is de volgende gestart kan worden zal de efficientie omhoog gaan ten opzichte van het originele systeem. Als gevolg zal ook de tijd die het preprocessen nodig heeft naar beneden gaan. Het resultaat is een efficiënter en sneller systeem wat minder tijd in beslag neemt op de cloud.

10.4 Test

Hier is een herhaling van de test van 8.5.2 uitgevoerd echter dit keer met Celery. Hiervoor is weer een enkele machine met 4 cores aangezet.



Figuur 10.5: Processor use

Te zien is dat dit veel beter gebruik maakt van de processor en zelfs inclusief de opstart periode worden de processoren gemiddeld voor bijna 80% benut.

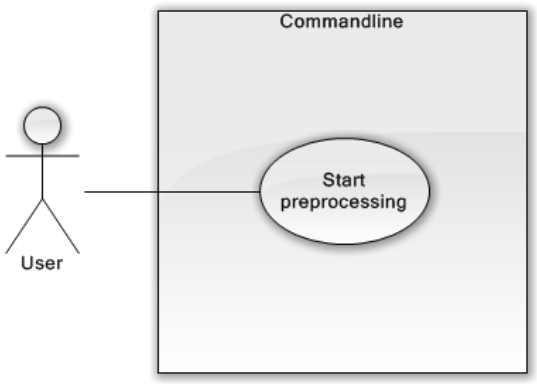
Hoofdstuk 11

Increment 4: Taskspawner

In dit increment wordt beschreven hoe de taskspawner werkt. Deze is gemaakt omdat het uitlezen van de brongegevens niet altijd locaal kan.

11.1 Analyse

De manier van het opstarten van het preprocessen moet hetzelfde blijven als de originele code. Er moet echter een opdeling gemaakt worden tussen het uitlezen van het configuratie bestand en het inlezen van de bronbestanden. Hierdoor is de usecase voor de gebruiker hetzelfde als de usecase van increment 2.



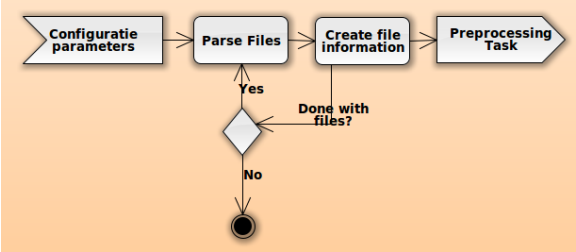
Figuur 11.1: Use Case Increment 4

11.2 Ontwerp

Door het originele script alleen nog maar de configuratie file uit te laten lezen en de resultaten hiervan door te sturen naar een fileparser is het systeem niet meer afhankelijk van het eigen file systeem. Hiermee is het mogelijk om

op een ander filesysteem, waarvan de opbouw niet bekend is voor de start functie, te werken.

De nieuwe functie die gebouwd is werkt zoals omschreven in Figuur . Er komt een configuratie bestand binnen. Deze word verwerkt volgens de originele functionaliteit en er worden taken weg gestuurd. Dit in combinatie met het vorige increment zorgt voor een volledig geheel.



Figuur 11.2: Flow chart increment 4

11.3 Implementatie

Er is een nieuwe functie aangemaakt. Deze functie loopt eerst alle gegevens door die deze binnen krijgt van de start task. Deze gegevens worden uit de configuratie file gehaald op de start computer. De gegevens moeten wel geserialiseerd worden om verzonden te kunnen worden door Celery.

11.3.1 Serialisatie van configuratie data

Om de configuratie gegevens door te sturen moet het dataformaat geserializeerd worden. Dit word automatisch gedaan door Celery en is in te stellen in de configuratie file van Celery.

JSON

JSON is de standaard serializer van Celery en werkt voor veel programmeertalen waaronder Python. Hierdoor is JSON goed voor het versturen van cross-language data. Een nadeel van JSON is echter wel dat de serializeerbare datatypes beperkt zijn.

Pickle

Pickle is een serializer speciaal gemaakt voor Python. Pickle ondersteunt dan ook ieder Python datatype. Pickle heeft echter niet de cross-language capaciteit van JSON omdat Pickle alleen maar voor Python werkt. Pickle is ook iets sneller dan JSON.

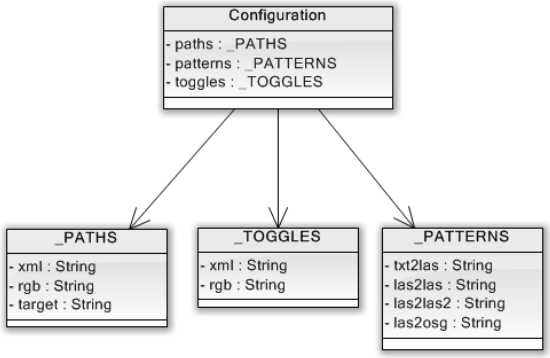
YAML

YAML lijkt erg veel op JSON maar heeft als voordeel dat er meer datatypes ondersteund worden . Een groot nadeel voor dit project is echter dat de YAML libraries voor python trager zijn dan de JSON libraries.

Keuze

Serializer analyse			
Eis	JSON	Pickle	YAML
Snelheid	goed	uitstekend	slecht
Hoeveelheid Datatypes	slecht	uitstekend	goed
Ondersteuning programmeertalen	uitstekend	slecht	uitstekend

Alhoewel Pickle er heel slecht uitkomt in de "Ondersteuning programmeertalen" categorie is dit niet erg omdat dit project in Python gemaakt wordt. Hierdoor is Pickle in alle relevante gebieden de beste keuze.



Figuur 11.3: Klasse diagram Configuration

Ondanks dat Pickle alle Python basis types aan kan was de Configuration klasse niet door Pickle te serialiseren. Dit komt door de subklasses die gebruikt worden zoals te zien is in Figuur 11.3. Om dit op te lossen is er een nieuwe klasse aangemaakt die als container dient voor de configuratie.

Hoofdstuk 12

Increment 5: Dynamisch stoppen van VMs

12.1 Analyse

Dit increment is van belang om geld te besparen op niet gebruikte cloud resources. Door een VM te stoppen zodra deze klaar is in plaats van het wachten op de rest van de VMs is het mogelijk om een aantal uren per VM aan core tijd te besparen. Dit is vooral aantrekkelijk voor het 3Di waterbeheer project omdat dit uit een groep bedrijven. Ook is het belangrijk als het project ooit overgezet wordt naar een cloud van bijvoorbeeld Amazon.

Om VMs te stoppen moet er een commando naar het OpenNebula framework gestuurd worden. Dit commando moet op het goede moment gestuurd worden. het goede moment om een VM te stoppen is het moment dat de laatste taak op een VM afgerond is. Indien de VMs eerder afgesloten worden ontstaat er datacorruptie en er wordt onnodig core uren verspild.

Het probleem is echter dat het niet mogelijk is om van binnen in een VM deze af te sluiten met gebruikelijke shutdown commando's. Indien dit geprobeerd wordt zal OpenNebula de VM opnieuw opstarten aangezien het framework de reservering nog steeds kent. Het commando van afsluiten moet dus direct naar OpenNebula gestuurd worden.

12.2 Ontwerp

Aan het einde van de preprocessing toolchain worden een even groot aantal shutdown commando's naar de message broker gestuurd als dat er Celery threads zijn. Dit aantal wordt bepaald door het aantal VMs en het aantal processoren van deze VMs.

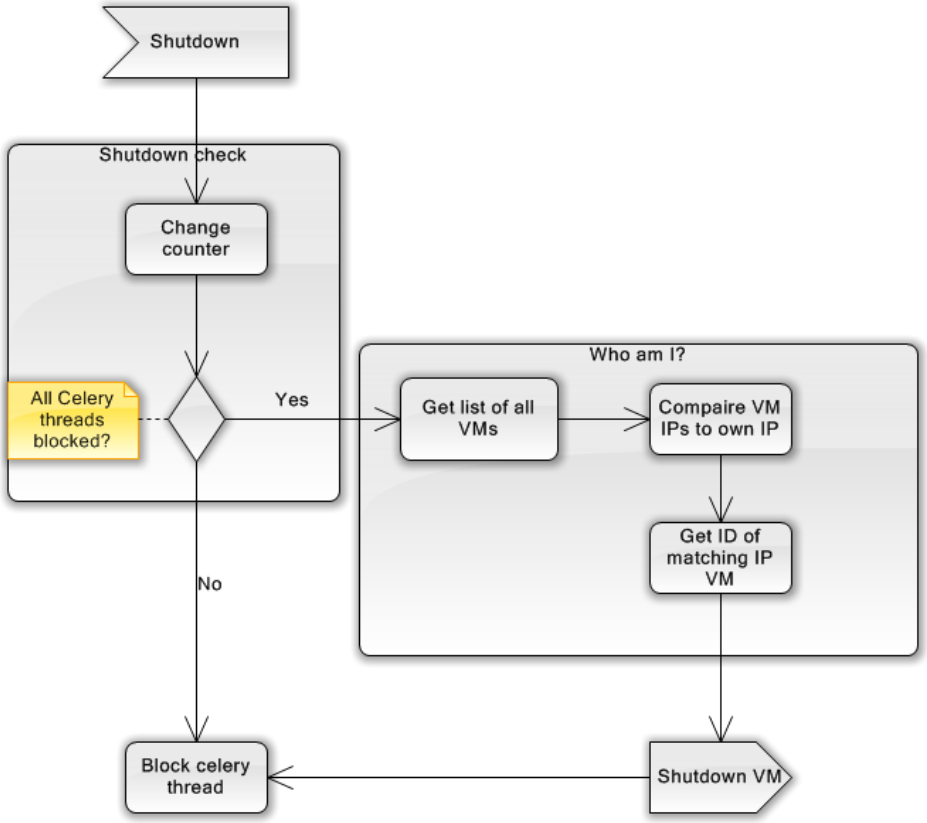
Het shutdown commando moet een aantal dingen doen. Het moet als eerste ervoor zorgen dat de desbetreffende celery thread geen nieuwe taken meer gaat accepteren. Als dit niet zou gebeuren zou de eerste thread die

klaar was alle shutdown commando's afhandelen en dit zorgt voor problemen bij de andere VMs. Door de taak een oneindige loop aan het einde te geven is ervoor te zorgen dat deze taak nooit klaar zal zijn en afgehandeld kan worden door Celery.

In Figuur 12.1 is de flow van de functie te zien. Als eerste wordt de teller bijgewerkt welke bijhoud hoeveel threads er al klaar zijn. Hierna wordt gekeken of alle andere threads geblokkeerd zijn en zo niet zal de functie zichzelf blokkeren.

Als het echter de laatste thread is die afgesloten wordt, gaat de VM kijken naar wie hij is. De identiteit van een VM is het ID dat OpenNebula aan hem hangt, echter wordt dit nooit doorgegeven aan de VM zelf. De VM is zich totaal niet bewust van het OpenNebula framework waar deze onder draait. Om te weten welk ID van de VM zelf is wordt dit vergeleken op iets wat de VM en OpenNebula beide weten. Dit is het IP adres van het interne netwerk of het externe netwerk. Dit is bekend bij OpenNebula omdat dit de netwerk interfaces verzorgd. De VM kent dit omdat dat zijn eigen netwerken zijn. Het is hier wel nodig om het eigen IP met de IPs van alle VMs in OpenNebula te vergelijken. Door deze vergelijking kan het ID gevonden worden.

Met dit ID is het mogelijk om een shutdown commando uit te voeren of een delete commando indien het shutdown commando niet werkt.



Figuur 12.1: Flow chart van dynamisch afsluiten VMs

12.3 Implementatie

12.3.1 Change counter

De counter waar bijgehouden wordt hoeveel threads er al afgesloten zijn is een real time kritisch stuk. Het is namelijk mogelijk dat twee shutdown commando's tegelijkertijd uitgevoerd worden. Dit zou betekenen dat beide shutdowns de counter op hetzelfde moment proberen te verhogen met als resultaat dat dit maar een keer gebeurt. Om dit te voorkomen is er een mutex gebruikt welke de file met de counter blokkeert indien deze in gebruik is. Op deze manier is dit race probleem vermeden.

Omdat de Celery threads niets met elkaar te maken hebben gebeurt dit met het fcntl.lockf commando. fcntl is een Python module voor Linux en deze bevat functies voor het locken van files.

Met de code:

```
fcntl.lockf(file.fileno(),fcntl.LOCK_EX)
```

wordt een file afgesloten en kan maar één programma hier gebruik van maken.

12.3.2 OCCI

De initiële gedachte was om met behulp van OCCI dit increment te bouwen. Na lang zoeken bleek echter dat het niet mogelijk was om een koppeling te maken tussen de OCCI data en een VM. OCCI is namelijk zeer afhankelijk van een ID en een lijst van alle VMs die opgestart zijn geeft alleen maar de IDs terug. Aangezien een VM niet weet wat zijn eigen ID is heeft deze hier niets aan. Na lang puzzelen en denken bleek echter dat XML-RPC wel perfect is voor deze taak.

12.3.3 XML-RPC

XML-RPC is in het begin niet overwogen omdat hier niet mee gewerkt kon worden vanwege het lichtpad. Dit was echter een foute gedachte aangezien de VMs zelf het afsluit commando geven. VMs draaien altijd op de cloud dus is het ook mogelijk dat deze altijd door middel van XML-RPC kunnen werken.

met het commando server.one.vmpool.info is het mogelijk om een zeer gedetailleerde lijst van alle VMs op te vragen in XML vorm. Deze XML is hierna te parsen op de desbetreffende netwerk IPs en hier is een vergelijking mee te maken.

Hoofdstuk 13

Verder onderzoek

Een belangrijk onderdeel van verder onderzoek is hoe dit systeem makkelijk over gezet kan worden op een andere cloud. Dit is belangrijk omdat het SARA project binnenkort zal stoppen en niet meer hiervoor gebruikt kan worden.

Ook is een module waarmee de software makkelijker up to date gehouden kan worden gewenst wegens het constante aanpassen van de huidige functies binnen de preprocessing toolchain.

Hoofdstuk 14

Conclusie

In dit project is aangetoond dat het preprocessen van de AHN2 kaart op de cloud mogelijk is. Dit preprocessen kan op grote schaal gebeuren en is hierdoor goed te gebruiken voor projecten als 3DI waterbeheer. Behalve tijdsbesparing geeft het ook het gemak van hardware op afstand. Dit betekent veel processing kracht voor een lage prijs. Het nadeel van hardware op afstand is dat dit lastiger is om te beheren en te controleren. Dit komt tot uiting in het onvermogen om software makkelijk automatisch te updaten naar de meest recente versie. Ook betekent hardware op afstand niet dat er totaal geen kennis hoeft te zijn over het systeem. De kennis van bijvoorbeeld het netwerk is nodig om dit goed te laten werken.

De belangrijkste beperking van dit project is dat het specefiek voor de SARA cloud is. Er is geprobeerd met universele standaarden te werken maar dit zijn wel aankoppelingen op het OpenNebula framework en op dit moment is het onbekend welke methodes op andere clouds werken.

Hoofdstuk 15

Nawoord

Persoonlijk heb ik veel geleerd van dit project. Niet alleen over Python, Celery en cloud computing, maar ook over mijzelf. Ik vind het jammer dat in dit project zo'n sterke nadruk op onderzoek lag. Daardoor was er relatief weinig tijd voor het schrijven van code en deze is dan ook minder uitgebreid en complex dan ik vooraf verwacht had.

Verder ben ik van plan om verder te gaan studeren aan de TUDelft zodra ik deze studie heb afgerond. Ik vind namelijk nog dat ik veel te leren heb.

Ik wil nogmaals mijn begeleiders aan de TUDelft bedanken, Tim Tutenel en Christian Kehl. De nuttige en goede feedback die ik gekregen heb over het schrijven van verslagen, welke ik niet allemaal heb toe kunnen passen in dit verslag, zal ik zeker bewaren voor in de toekomst. Ook was de sfeer binnen de TUDelft zeer goed om binnen te werken.

Als laatste wil ik mijn ouders bedanken voor de constante steun die ze mij geboden hebben tijdens dit project.

Hoofdstuk 16

Begrippenlijst

AHN2 Puntenwolk van Nederland. Zeer groot en zeer nauwkeurig.

HPC Cloud High Preformance Computing Cloud. Hier wordt al het werk op verricht.

SARA Stichting Academisch Rekencentrum Amsterdam. Werkt samen met wetenschappelijke organisaties en bied computer functionaliteiten aan aan deze groepen.

Celery Framework wat asynchroon taken kan distribueren over meerdere computers.

OCCI Staat voor Open Cloud Computing Interface. Wordt gebruikt om de HPC Cloud te besturen.

XML-RPC Een Remote Procedure Call module die gebruikt kan worden om de HPC cloud van binnen uit te besturen.

VM Afkorting voor Virtual Machine

VIRDIR Virtual Directory. De cloud storage solution van SARA.

Zeus de server op de TUDelft

VLAN Virtual Local Area Network. Het interne netwerk van SARA werkt hiermee om projectgroepen apart te houden.

LiDAR LIght Detection and Ranging. Is gebruikt om bijvoorbeeld de AHN2 kaart op te stellen.

EEMCS Electrical Engineering, Mathematics and Computer Science

EYR Enlighten Your Research, wedstrijd voor wetenschappers met als prijs cloud computing mogelijkheden

lichtpad Glasvezel verbinding tussen de TUDelft en SARA.

Hoofdstuk 17

Bronnen

Visualization on a Budget for Massive LiDAR Point Clouds, auteur Berend Wouda, Msc. Thesis
http://www.3di.nu/
www.ANN.nl
http://www.deltares.nl/en
http://www.nelen-schuurmans.nl/NL/home
http://en.wikipedia.org/wiki/Rapid_application_development
http://en.wikipedia.org/wiki/Rapid_prototyping
http://www.celeryproject.org/
www.surfsara.nl
https://ui.cloud.sara.nl/
http://en.wikipedia.org/wiki/LIDAR
www.occi-wg.org
http://docs.python.org/2.7
linux.about.com
http://stackoverflow.com/
http://opennebula.org/documentation:rel3.8:occiug
https://www.cloud.sara.nl/projects/hpc-cloud-documentation/wiki/XML-RPC
http://opennebula.org/documentation:archives:rel3.0:api
pypi.python.org
http://en.wikibooks.org/wiki/LaTeX/