RAPPORT LEERTAAK 2

Datum : 02-06-2014

Auteurs : Tom Broenink

Ward Holthof

Yuri Hoogeweg

School : Hanzehogeschool Groningen

Versiebeheer

VERSIENUMMER	DATUM	DOOR	COMMENTAAR
0.6	02-06-2014	WARD HOLTHOF	AFWERKING
0.5	02-06-2014	TOM BROENINK	UITWERKING VRAAGSTELLING
0.4	02-06-2014	WARD HOLTHOF	UITWERKING TESTRESULTATEN
0.3	01-06-2014	WARD HOLTHOF	TOEVOEGING TESTRESULTATEN
0.2	31-05-2014	WARD HOLTHOF	EERSTE AANVULLING
0.1	30-05-2014	TOM BROENINK	EERSTE OPZET

Inleiding

In dit rapport wordt Leertaak 2 beschreven. Dit rapport is geschreven door Tom Broenink, Ward Holthof en Yuri Hoogeweg.

De taakomschrijving voor Leertaak 2 is als volgt:

In leertaak 2 wordt er een oplossing voor het opslaan van gegevens uit meerdere bronnen tegelijkertijd gemaakt, zonder dat deze elkaar beïnvloeden. De data wordt nu niet in een RDBMS opgeslagen, zoals in Leertaak 1, maar in een ander opslagsysteem waarbij er rekening wordt gehouden met zaken als concurrency. Er wordt tevens opnieuw een stresstest gedaan om vast te stellen welke belasting de proefversie wel en niet kan verwerken.

De applicatie bevat de volgende functies:

- Correctie van de weergegevens zoals beschreven in de casustekst;
- Opslag van de weergegevens (op een schaalbare manier);
- Eén voorbeeld naar eigen inzicht van een selectiequery op de weergegevens zoals beschreven is in de casustekst.

In dit rapport worden de volgende onderdelen beschreven:

- Een uitleg van de programmaonderdelen die de gevraagde functies vormgeven;
- De resultaten van de stresstest inclusief een verklaring voor de grenzen van het systeem.
- Hierbij wordt aangegeven:
    - een overzicht van de door ons gebruikte systemen en infrastructuur (besturingssysteem, CPU, geheugen, netwerkoverzicht)
    - wat de gehaalde verwerkingssnelheid (aantal verwerkte berichten per seconde) is
    - welke resource de bottleneck vormt en door welk proces dit wordt veroorzaakt
    - een onderbouwing van onze uitleg met behulp van gegevens van de taskmanager (of een door jouw gebruikte OS-variant ervan), inclusief screenshots
    - welke vorm(structuur) van opslag we hebben gekozen
    - de schaalbaarheid van ons systeem: hoe zou het gaan als daadwerkelijk 8000 individuele stations tegelijkertijd en een jaar lang hun gegevens aanbieden.

Inhoudsopgave

Probleemstelling

Het UNWDMI heeft toegang tot ruim 8000 weerstations, die wereldwijd zijn opgesteld. Deze weerstations sturen 24 uur per dag en 7 dagen per week elke seconde een aantal gegevens naar Groningen.

Tot deze gegevens behoren:

- Een identifcatiecode van het station
- De lokale datum en tijd
- Weergegevens:
    - Temperatuur
    - Dauwpunt
    - Luchtdruk
    - Zichtbaarheid
    - Neerslag
    - Sneeuwdiepte
    - Bewolking
    - Windrichting
    - Windsnelheid
    - Gebeurtenissen

Op te treden fouten

Bij het meten en verzenden van de meetwaarden gaat er nog wel eens wat mis. Zeker in afgelegen gebieden op de wereld kunnen er storingen in de weerstations optreden, die niet zo snel verholpen kunnen worden. Als gevolg daarvan kunnen meetwaarden soms irreëel zijn of zelfs ontbreken. Daarom vindt in de applicatie controleslag plaats voordat de meetgegevens opgeslagen worden in de centrale database. De applicatie gaat daarbij als volgt te werk:

1. Indien één of meer meetwaarden ontbreken, worden ze door het systeem berekend door middel van extrapolatie van de dertig voorafgaande metingen. Dit komt ongeveer in 1% van alle gevallen voor.
2. Een meetwaarde voor de temperatuur wordt als irreëel beschouwd indien ze 20% of meer groter is of kleiner is dan wat men kan verwachten op basis van extrapolatie van de dertig voorafgaande temperatuurmetingen. In dat geval wordt de geëxtrapoleerde waarde ± 20% voor de temperatuur opgeslagen. Voor de andere meetwaarden wordt deze handelswijze niet toegepast.

De hier beschreven handelswijze is volkomen geaccepteerd in de wereld van de meteorologie. Alle instellingen die weergegevens opslaan passen dit systeem toe en alle gebruikers van weerinformatie weten dat dit systeem toegepast wordt en nemen daar genoegen mee. Dat geldt niet alleen voor landelijke weerdiensten, maar ook voor commerciële weeradviesbureaus en onderzoeksinstituten.

Verwerking

De applicatie moet de weergegevens van 8000 weerstations per seconde zo nodig verbeteren, verwerken en opslaan. Vervolgens moet er gemakkelijk op deze opslag aanvragen gedaan kunnen worden zoals het opvragen van de gemiddelde temperatuur in een bepaalde regio in de afgelopen maand.

In dit rapport worden de test resultaten van een stresstest op de applicatie en database gemeten en verklaard.

Verklaring programmaonderdelen

De applicatie bevat vijf classes die functionaliteit bevatten. Verder word de applicatie vanuit de 'main' class gestart. De main class zullen we in dit onderdeel verder niet bespreken omdat dat een simpele class is met één regel code. (Het enige wat deze class doet is een Receiver object maken en starten).

Classes:

Message

Alle messages die van clusters worden ontvangen worden gedefinieerd als een Message object. Dit object bevat alle informatie van een message in een hashmap.

Receiver

Luistert naar een poort voor nieuwe verbindingen van clusters. Wijst clusters toe aan een thread (ClusterConnection) in een threadpool. Aantal threads per pool, maximale aantal threadpools en de poort worden in de main method in de main class gespecifieerd.

ClusterConnection

Wordt per cluster aangemaakt, kijkt continu of er nieuwe messages worden ontvangen. Zodra deze ontvangen zijn worden ze doorgestuurd naar het DatabaseConnection object. De ClusterConnection class implementeert 'Runnable' en wordt dus ook als Thread gerund.

DatabaseConnection

Bevat alle informatie voor de verbinding naar de database, connection string, username/password etc. Zodra dit object een message binnen krijgt zal hij bij de MessageCorrector checken of deze message compleet en correct is.

MessageCorrector

Per station is er één MessageCorrector, deze houdt de afgelopen 30 messages bij, en kan hier waarden uit extrapoleren om te kijken of een nieuwe message wel compleet en accuraat is. Vervolgens kan de MessageCorrector aanpassingen maken voordat de message gereturned wordt naar de DatabaseConnection om verzonden te worden.

Stresstest

Overzicht gebruikte systemen en infrastructuur

Voor het uitvoeren van de stresstest hebben we een desktop gebruikt met de volgende specificaties:

- *Besturingssysteem*: Windows 8.1 x64
- *CPU*: Intel Core i7-2600K 3.4GHz
- *Geheugen*: 16GB DDR3 RAM

De database draait op een virtuele machine. Er wordt een connectie met de database gemaakt vanaf de desktop naar de server via het interne netwerk. Hierbij is er een latency van 0.500 ms.

De UNWDMI Generator draait op dezelfde machine als onze applicatie.

Specificaties van de server:

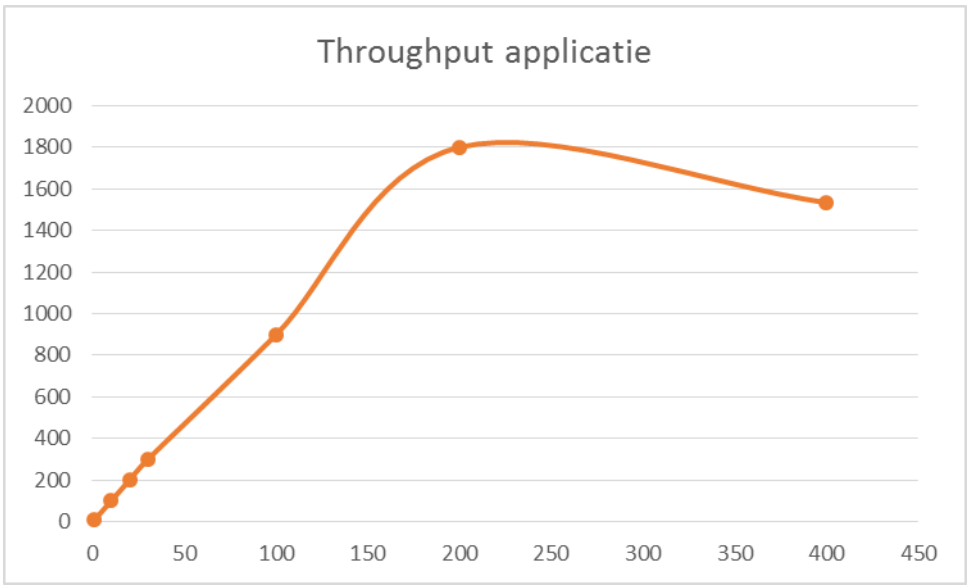
- *Besturingssysteem*: Ubuntu 14.04 x64
- *CPU*: Intel Core i7-2600K 3.4GHz
- *Geheugen*: 8GB DDR3 RAM
- *Hardeschijf*: 15GB SSD Disk
- *Database* MongoDB

Gehaalde verwerkingssnelheid

Na het optimaliseren van onze applicaties is deze een stuk sneller geworden.

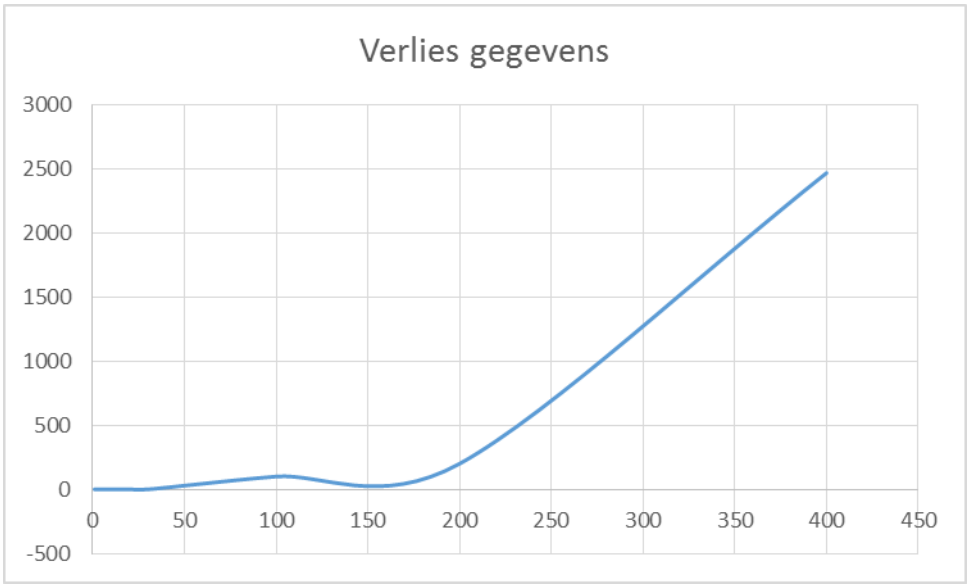
Uit tests is gebleken dat waar we voorheen een maximale throughput hadden van 6.7 berichten per seconde dit nu is gestegen naar een maximale 1800 berichten per seconde.

Uit de tests blijkt echter wel dat wanneer we meer data genereren deze throughput weer terug begint te lopen zoals te zien in onderstaande grafiek:



Dit probleem komt weg bij de checks die wij doen op de data om deze vervolgens in de database te kunnen plaatsen. Dit vertraagd onze throughput en op een bepaald moment beginnen we zelf data te verliezen. Dit is dan hoogstwaarschijnlijk ook de voornaamste reden dat we nog niet de maximale load (van 8000 berichten per seconden) aankunnen.

In onderstaande grafiek is duidelijk te zien dat data wordt verloren wanneer we meer berichten proberen te verwerken:

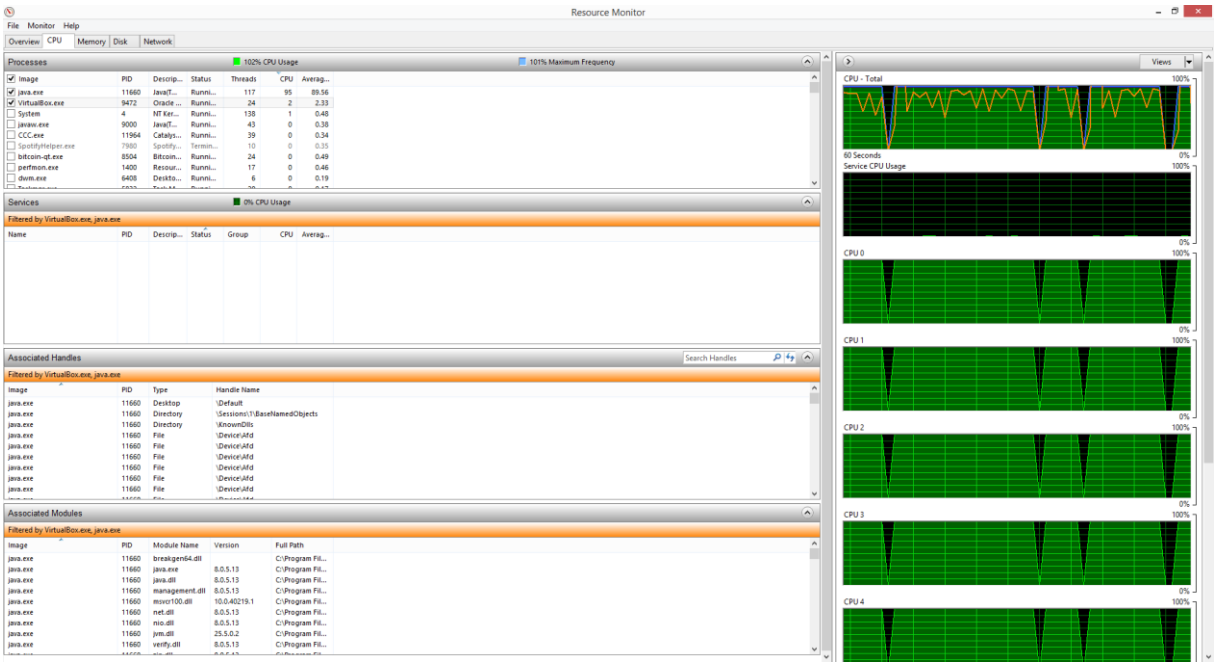


Nadat onze applicatie klaar is met draaien en alle berichten verwerkt zijn lijkt er een bug te ontstaan waardoor onze applicatie opeens veel geheugen en processorkracht opeist.

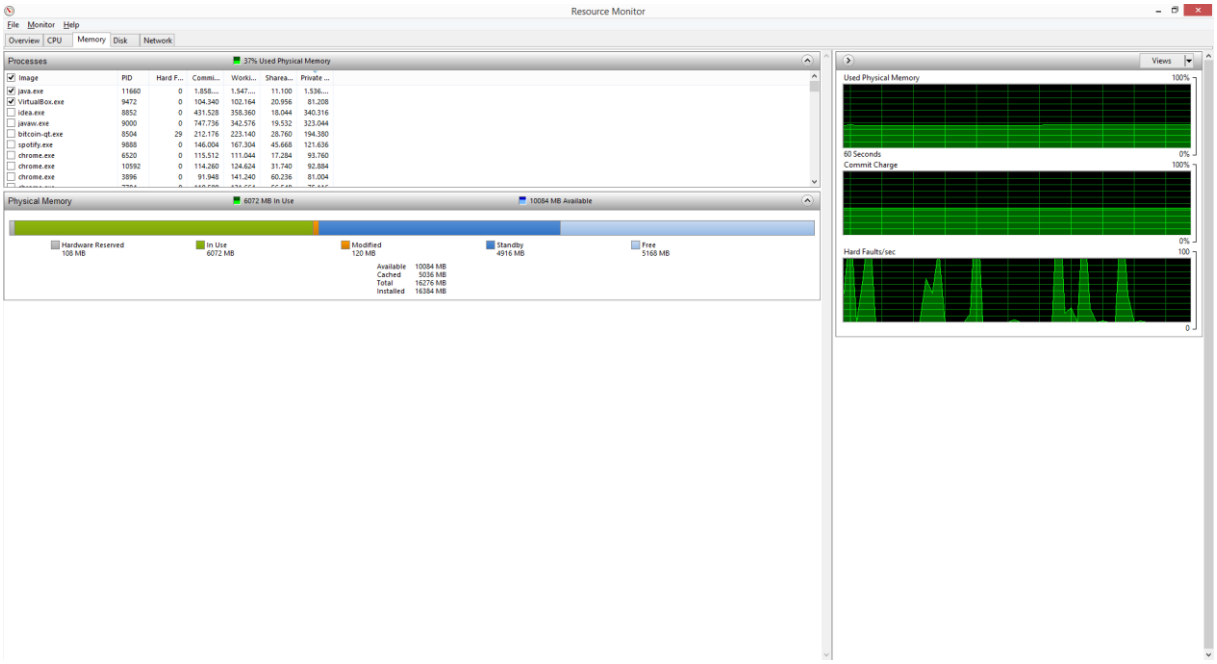
Tijdens het draaien van de applicatie is er echter geen probleem en is het cpu en geheugen gebruik relatief laag.

Deze bug zou kunnen komen doordat we nog veel open sockets hebben en losse threads hebben die opgeruimd moeten worden.

CPU Verbruik door Java



Geheugen gebruik door Java



Conclusie

Onze applicatie moet nog door een aantal verbeter stappen voordat er voldaan kan worden aan de eis van 8000 berichten per seconde. Wel is er al een aanzienlijke verbetering geboekt en moet het mogelijk zijn om met niet al te veel veranderingen de 8000 berichten per seconden te gaan halen.

Ook is er een grote verbetering geboekt in het aantal fouten dat er op treed. Waar voorheen nog een groot gedeelte van de data verdween is dit nu amper nog het geval. Alleen bij een zeer hoge throughput waarop onze applicatie eigenlijk niet is voorbereid kan dit nog optreden.