Leertaak 2 Testrapport

Groep: *1a*
Auteurs: *André Nanninga & Maurits van Mastrigt*
Datum: *30 mei 2014*

Inhoud

Inleiding

Dit rapport beschrijft de conclusies en bevindingen van André Nanninga en Maurits van Mastrigt tijdens het uitwerken van Leertaak 2.

De rapportage is opgedeeld in drie delen: inleiding, resultaten, en conclusie. Waarbij in de inleiding de tweede leertaak, de probleemstelling en de programmaonderdelen zullen worden toegelicht. Hierop volgend worden de stresstest resultaten gespresenteerd en toegelicht. En afsluitend worden de conclusies en bevindingen beschreven.

Het uitgevoerde onderzoek is een vervolg op de uitwerking van de eerste leertaak.

Definitie leertaak twee

In de tweede leertaak wordt voortborduurt op de eerste leertaak, waarbij er een oplossing moet worden gevonden voor het tegelijkertijd opslaan van gegevens uit meerdere bronnen, zonder dat deze elkaar beïnvloeden. Tevens mag de data niet worden opgeslagen in een relationele database en moet het mogelijk zijn tegelijkertijd weerdata uit te lezen.

Zoals in de leertaak beschreven, bevat dit rapport de volgende onderdelen:

- Een verklaring van de programmaonderdelen die de gevraagde functies vormgeven;
- De resultaten van de stresstest inclusief verklaring voor de maximale snelheid van de gegevenswerking, waarbij het volgende zal worden aangegeven:
    - Een overzicht van de gebruikte systemen en infrastructuur;
    - De gehaalde verwerkingssnelheid (aantal verwerkte berichten per seconde);
    - Welke resource de bottleneck vormt en door welk proces dit wordt veroorzaakt;
- Een onderbouwing met behulp van de verzamelde gegevens;
- Een verklaring voor de gekozen opslagmethode;
- Een uitleg over de schaalbaarheid van het systeem.

De in dit rapport beschreven conclusies en bevindingen zullen worden meegenomen in het uitwerken van leertaak vijf.

Probleemstelling

Het UNWDMI heeft toegang tot ruim 8000 weerstations verspreid over de hele wereld. Elk van deze weerstations sturen 24 uur per dag en 7 dagen per week elke seconde een aantal gegevens zoals:

- Identificatiecode van het station
- lokale datum en tijd
- weergegevens:
- temperatuur
- dauwpunt
- luchtdruk
- zichtbaarheid
- neerslag
- sneeuwdiepte
- bewolking
- windrichting
- windsnelheid
- gebeurtenissen

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.

- 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.
- 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.

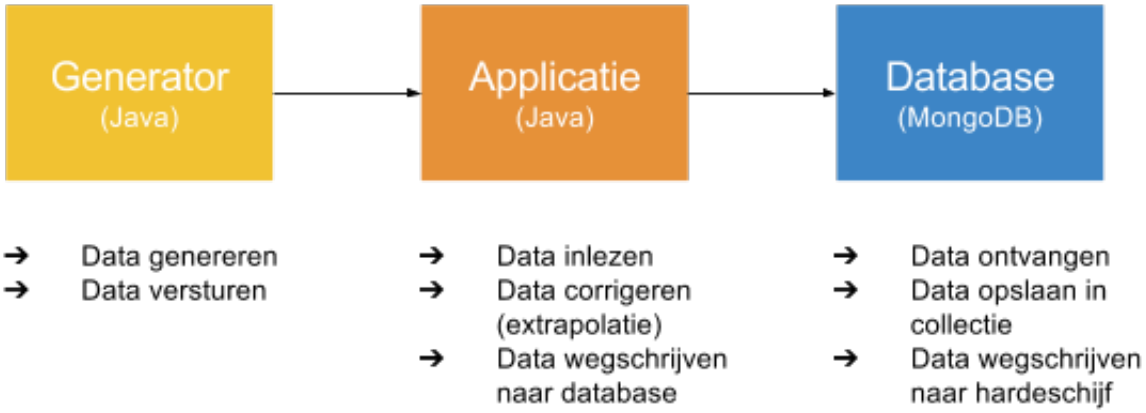
De applicatie moet de weergegevens van 8000 weerstations per seconde zo nodig verbeteren, verwerken en opslaan. Vervolgens moet er gemakkelijk op deze opslag aanvragen gedaan kunnen worden zoals het opvragen van de gemiddelde temperatuur in een bepaalde regio in de afgelopen maand.

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

Verklaring programmaonderdelen

Onderstaand een toelichting op de verschillende programmaonderdelen. Met als eerst een beschrijving van de infrastructuur en daarop volgend meer over de gebouwde applicatie.

Infrastructuur



In het bovenstaande figuur is te zien waar elk onderdeel verantwoordelijk voor is. Zowel de generator, applicatie, en de database werden gedraaid op één en dezelfde desktop PC. Deze PC heeft de volgende specificaties:

> *Besturingssysteem:* *Windows 8.1 Pro 64-bit*
> *CPU:* *AMD Athlon(tm) X4 740 Quad Core Processor*
> *Geheugen:* *8.00GB Dual-Channel DDR3 @ 665MHz (9–9–9–24)*
> *Moederboord:* *ASRock FM2A75 Pro4-M (CPUSocket)*
> *Opslag:* *RAID 0 (3x 232GB Seagate ST3250318AS ATA Device)*

De **generator** is door de Hanzehogeschool als uitvoerbaar `.jar`-bestand aangeleverd. Deze Java applicatie genereert (semi)willekeurge weerdata aan de hand van een aantal instellingen. Zo kon het aantal workers worden ingesteld, waarmee de server applicatie eenvoudig te stresstesten was.

De bovengenoemde **applicatie** is volledig zelf ontwikkeld. Hier lagen wel een aantal vereisten aan ten grondslag. Samengevat moest er een multithreaded Java applicatie worden gebouwd, die door middel van sockets een XML stream uitleest. De ingelezen gegevens moeten vervolgens worden omgezet naar een werkbaar data formaat. Ontbrekende data moest worden gecorrigeerd en de data moest worden opgeslagen in non-relationale database. De specifieke uitwerking van de applicatie zal nader worden toegelicht in paragraaf *Applicatie*.

Voor opslag van de gegevens is er gekozen voor de meest bekende non-relationele database: **MongoDB**. Deze database is eenvoudig in opzet, kan zeer hoge lees- en schrijfsnelheden behalen, en schaalt enorm goed.

Database

De opdracht verbiedt het gebruik van een relationele database. Hierdoor is er gekeken naar de mogelijke alternatieven. De twee voornaamste alternatieven zijn non-relationele databases en flat-file databases.

Flat-file databases lastig schalen, omdat deze logica zelf ingebouwd moet worden. Bijvoorbeeld als een bestand een bepaalde grootte (bijvoorbeeld 4 gigabyte) overschrijdt, kan er worden gekozen om in een nieuw bestand te schrijven. Omdat hierbij zelf de functionaliteit voor schaling van de database moet worden voorzien, is een non-relationele database in dit geval een betere optie.

Het meest bekende en doorontwikkelde non-relationele database is MongoDB. Dit is een zogeheten "document based" database, wat inhoudt dat objecten in een document (collectie) worden opgeslagen. Dit in tegenstelling tot een relationele database, die met tabellen, kolommen, en rijen werkt. MongoDB is eenvoudig op te zetten, makkelijk in gebruik, en wordt goed ondersteund (in de vorm van driver beschikbaarheid). Dit maakt het vervangen van de MySQL database een stuk eenvoudiger.

Deze database is ontwikkeld met "Big Data" in gedachte, waardoor schaalbaarheid door middel van clusters heel eenvoudig op te zetten is. Ook dit aspect is meegenomen in de keuze voor MongoDB. Omdat deze database gemaakt is voor het verwerken van veel data, sluit hij goed aan op de wens is daarmee een zeer geschikte vervanger voor MySQL.

Applicatie

In deze paragraaf zullen de verschillende klassen van de applicatie in volgorde van verloop worden toegelicht.

Runner

De zogeheten "Runner" van de applicatie dient als startpunt van de applicatie. Hierin wordt de applicatie geconfigureerd en worden de initiële onderdelen, zoals de Server, opgezet. Tevens dient de *Runner* klasse voor het meten van de resultaten, waarbij de server een X aantal seconden wordt gedraaid - terwijl ondertussen metingen worden verricht als:

- Ingevoerde configuratiewaarden
- Uiteindelijke looptijd (inclusief afronden,)
- Geheugengebruik
- Aantal workers (inkomende connecties)
- Aantal database queries
- Verwachtte aantal records
- Uiteindelijke aantal records
- Efficiëntie

Met deze meeteenheden kan applicatiebrede efficiëntie worden gemeten, wat enorm heeft geholpen bij het verbeteren van de applicatie code.

Server

De *Server* klasse beheert de database connectie en accepteert continue inkomende connecties. Elke inkomende connectie wordt overgezet naar een aparte *Worker*, welke vervolgens de data inleest en verwerkt. Deze connecties komen voor nu nog vanaf de generator.

Database

De *Database* klasse zorgt voor zowel het tot stand brengen van een verbinding met de Mongo database, als eenvoudige communicatie met deze database. Deze laag van abstractie heeft het vervangen van de MySQL database (gebruikt in de eerste leertaak) met een non-relationele database, in dit geval MongoDB, zeer eenvoudig gemaakt.

Ten opzichte van de eerste leertaak is er geen gebruik gemaakt van een *Database.Executor* (of *RecordBuffer*), omdat de non-relationele database het inschieten van de data objecten prima aan kan.

Worker

De *Worker* klasse staat centraal aan de applicatie. Deze moet zo snel mogelijk de ingelezen gegevens verwerken tot bruikbare weerdata, de data corrigeren (met behulp van een *Corrector*), en inschieten in de database (in dit geval zonder behulp van een *RecordBuffer*). Dit laatste punt kost weinig tijd door het gebruik van een non-relationele database, die zeer hoge schrijfsnelheiden behaald ten opzichte van een relationele database (zoals gebruikt in de eerste leertaak).

Corrector

Elke *Worker* heeft een eigen instantie van de *Corrector*, welke - zoals de naam aanduidt - de missende waarden corrigeert. Met behulp van extrapolatie wordt een schatting gedaan van de ontbrekende waarde. Tevens wordt er gekeken of de temperatuur niet meer dan 20% afwijkt van de vorige waarde. Is dit wel het geval, dan wordt ook deze gecorrigeerd (naar het maximale percentage).

Ook de *Corrector* moet snel handelen, omdat deze draait in dezelfde thread als de bijbehorende *Worker*. Dit is mogelijk, omdat de corrector zeer efficiënt omgaat met de gegevens. Dit is behaald door het vinden van de juiste record buffer grootte (het aantal records waar extrapolatie op wordt toegepast) en extreme code optimalatie.

Record

De *Record* klasse dient voornamelijk als hulpmiddel bij gebruik van een record object. De ingelezen weerdata wordt namelijk niet omgezet naar een klasseinstantie, maar wordt in een Object array gezet. Dit is zeer lichtgewicht, waardoor er enkel een hulpmiddel nodig is voor het defineren van de indexen van de array (welke sleutel welke waarde representeert).

Tevens biedt deze klasse de mogelijkheid de missende waarde van een record object te bepalen en een record object om te zetten naar een database object. Op deze manier wordt alle logica intern gehouden, waardoor de applicatie code netjes blijft en andere klassen geen kennis hoeven te hebben van het record object. Met uitzondering van het ophalen van een waarde (bijvoorbeeld `record[ Record.WNDDIR ]`) en het instellen van een waarde (bijvoorbeeld `record[ Record.WNDDIR ] = value;`).

Stresstest resultaten

De stresstest is meerdere malen uitgevoerd met een doorloop tijd van 30 seconden. De resultaten hiervan zijn als volgt:

Stresstest resultaten met MongoDB

Clusters	Geheugen	Mutaties	Aantal records	Verwacht aantal records	Efficiëntie
800	318.00 MB	247160	246360	240000	102.65%
800	318.00 MB	246230	245430	240000	102.25%
800	515.50 MB	246040	245240	240000	102.18%
800	518.00 MB	248000	247200	240000	102.99%
800	515.50 MB	247460	246660	240000	102.78%
Gemiddeld					
800	436.90 MB	246978	246178	240000	102.57%

Stresstest resultaten met MySQL

Clusters	Geheugen	Mutaties	Aantal records	Verwacht aantal records	Efficiëntie
800	499.50 MB	872	246480	240000	102.70%
800	557.00 MB	872	247910	240000	103.30%
800	507.00 MB	879	248000	240000	103.33%
800	509.00 MB	881	248000	240000	103.33%
800	499.50 MB	888	247480	240000	103.12%
Gemiddeld					
800	514.40 MB	878	247574	240000	103.16%

De reden dat de efficiëntie boven 100% is omdat de workers niet direct worden gestopt en zo dus nog een klein beetje data kunnen ontvangen.

Omdat de applicatie met MySQL al 100% efficiëntie kon bereiken is daar geen verschil in te zien. Het enige verschil is een kleine daling in geheugen verbruikt (van 514.40 MB naar 436.90 MB). Deze daling is ontstaan door het verwijderen van de RecordBuffer, deze bufferde de Records tot een bepaald punt waarna het werd wegschreven naar de database. Deze buffer zorgde voor extra geheugenverbruik.

Hier staat tegen over dat het aantal mutaties wel flink is opgelopen (van 878 gemiddeld naar 246978 gemiddeld). Met de voormalige RecordBuffer werden veel minder mutaties gedaan maar bestonde mutaties wel uit meerdere Records. Met het verwijderen van de RecordBuffer wordt elke Record direct weggeschreven wat zorgt voor een groter aantal mutaties. MongoDB is zelf erg geomptimaliseerd voor het snel wegschrijven van veel data dus deze wijzigen heeft niet gezorgd voor een daling in efficiëntie.

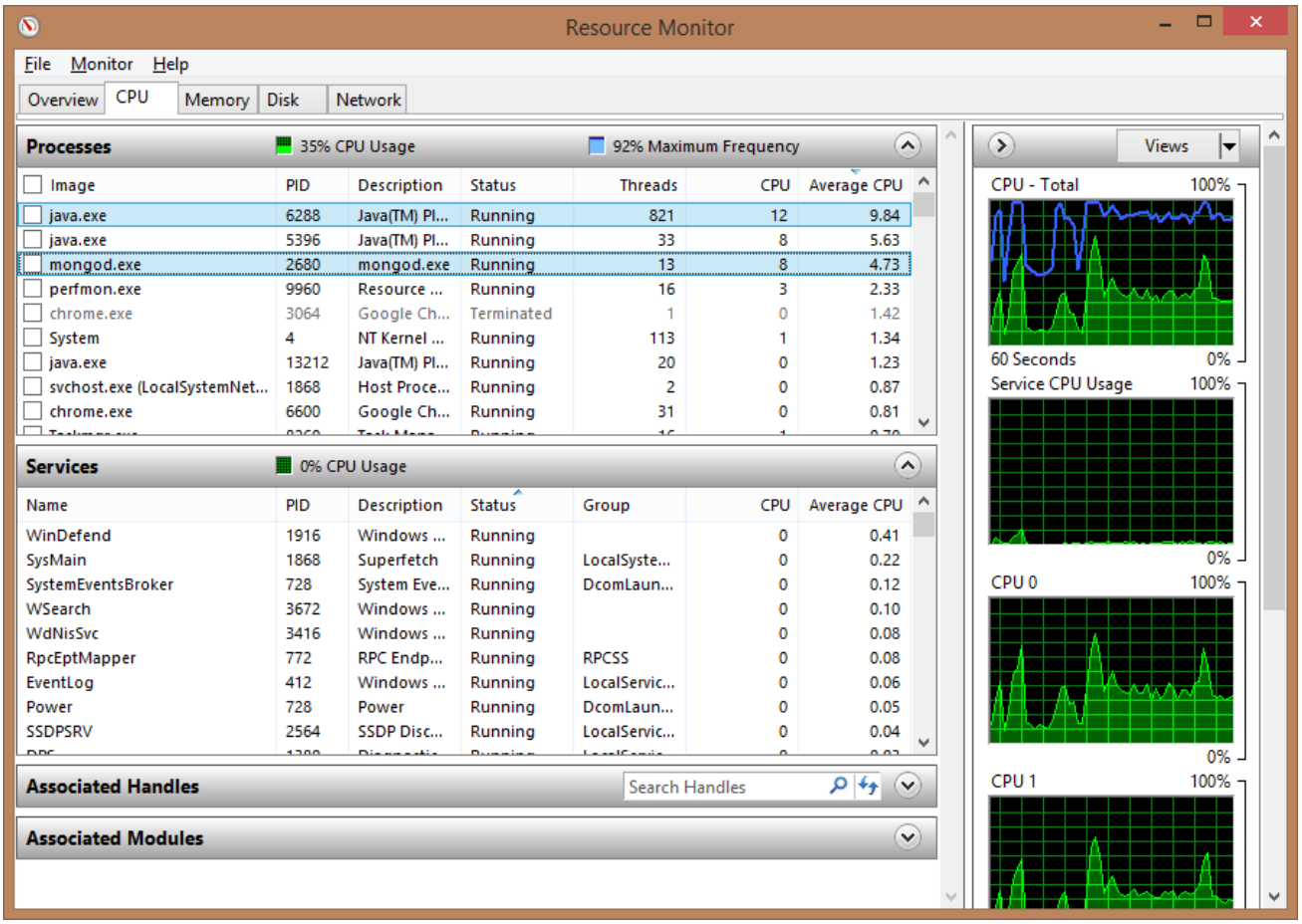
Langdurende stresstests

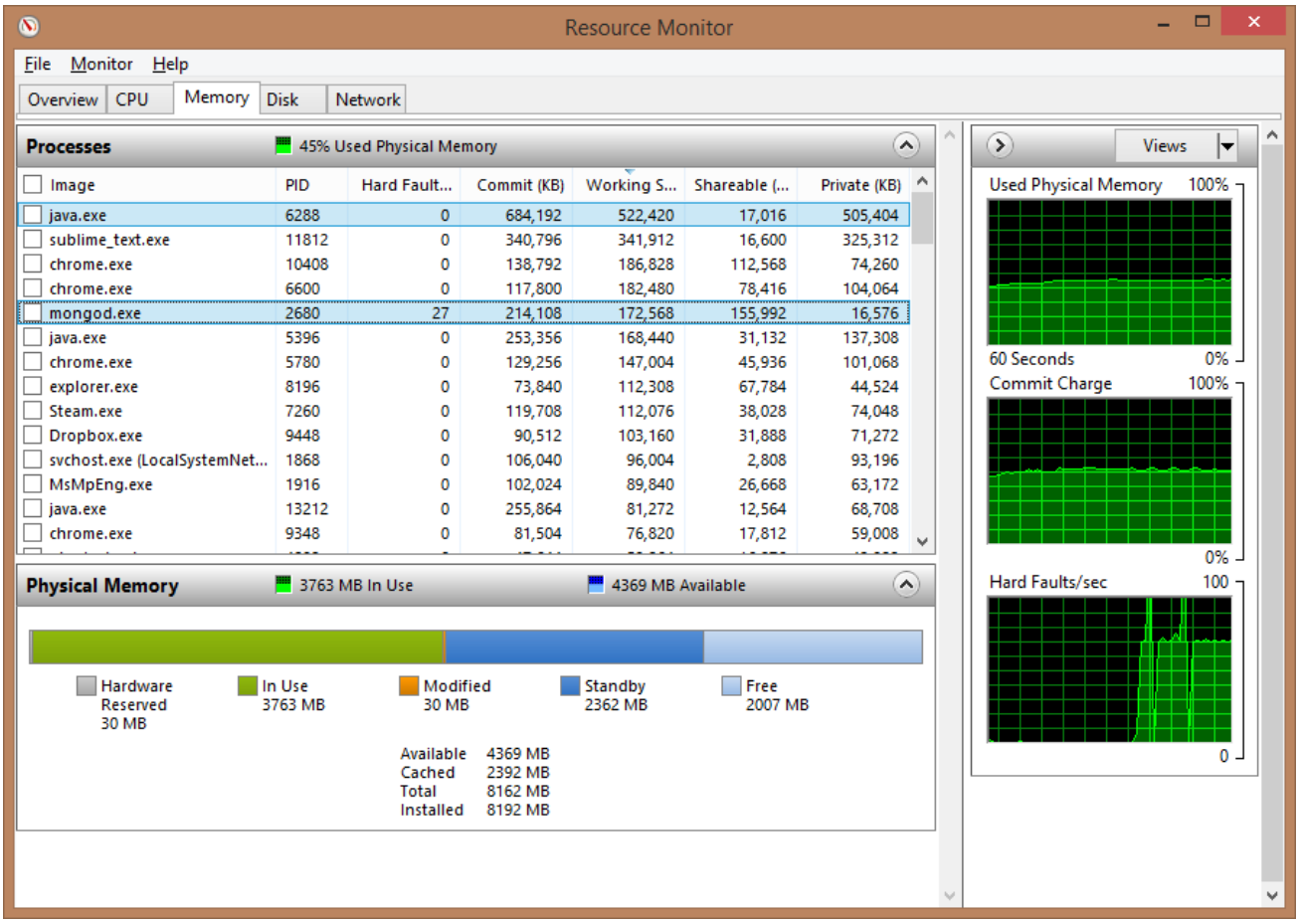
Duur	Clusters	Geheugen	Mutaties	Aantal records	Verwacht aantal records	Efficiëntie
15 minuten	800	153.50 MB	7207550	7206750	7200000	100.09%
30 minuten	800	149.00 MB	14176000	14472520	14400000	100.05%
60 minuten	800	148.50 MB	28805760	28804960	28800000	100.02%

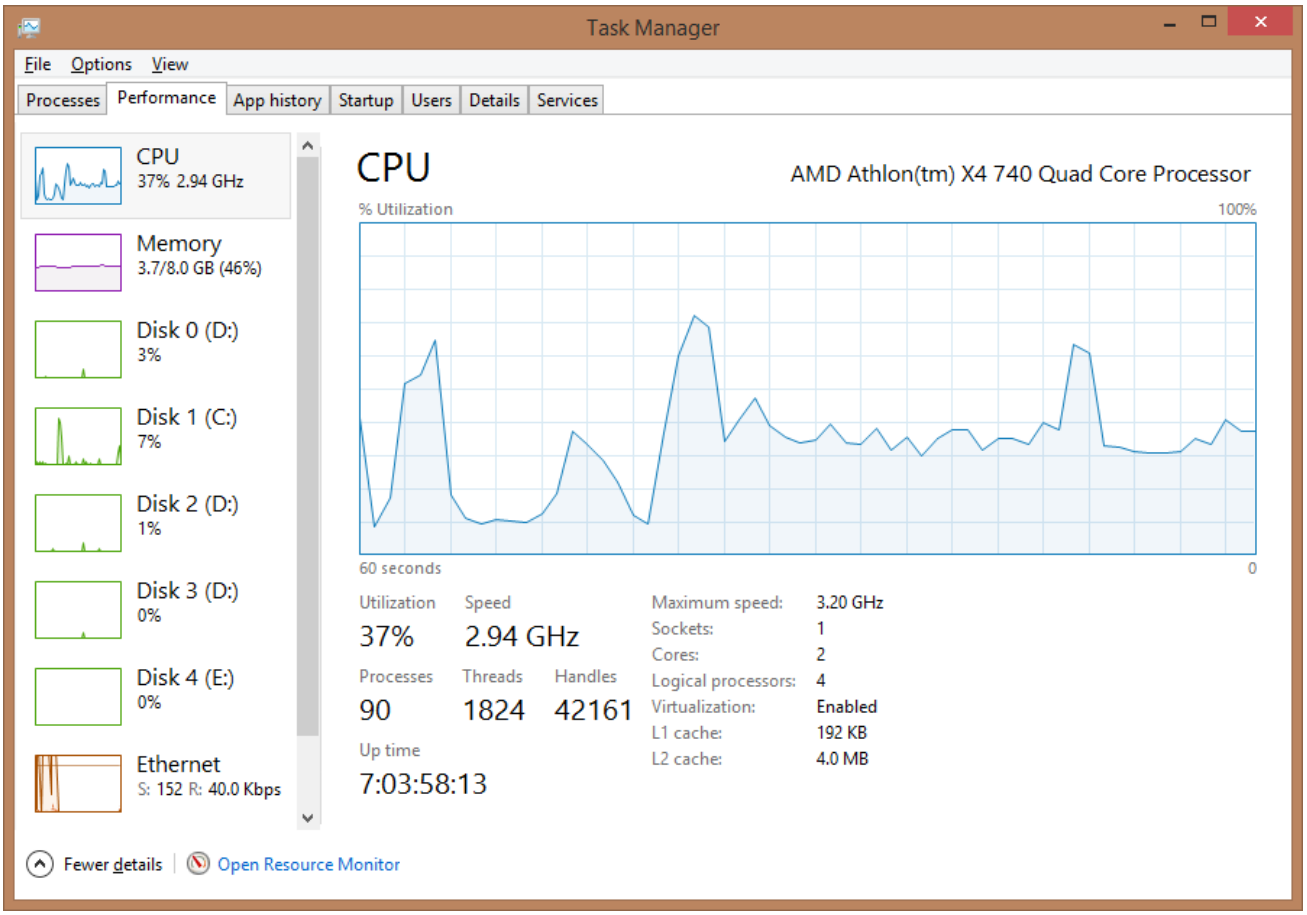
Na het uitvoeren van een aantal langere stresstests ontstaan nog steeds geen problemen. De efficiëntie blijft 100% en het gebruikte geheugen stijgt ook niet. De grootte van de database na de stresstest van 15 minuten was *1.73 GB*. Na de stresstest van 60 minuten was de grootte *6.91 GB* oftewel (bijna) precies 4 keer zo groot als bij de stresstest van 15 minuten. De stresstest van 30 minuten toont een database grootte van *3.46 GB* oftewel twee keer zo groot als de grootte na een kwartier en twee keer zo klein als de grootte na een uur. De groei van de database lijkt dus lineair te zijn.

Machine gebruik tijdens stresstesting

Onderstaand schermafdrukken van respectievelijk het CPU-, geheugen-, en hardeschijfgebruik.




Resource Monitor
File Monitor Help
Overview CPU Memory Disk Network
Processes 45% Used Physical Memory
Image PID Hard Fault... Commit (KB) Working S... Shareable (... Private (KB)
java.exe 6288 0 684,192 522,420 17,016 505,404
sublime_text.exe 11812 0 340,796 341,912 16,600 325,312
chrome.exe 10408 0 138,792 186,828 112,568 74,260
chrome.exe 6600 0 117,800 182,480 78,416 104,064
mongod.exe 2680 27 214,108 172,568 155,992 16,576
java.exe 5396 0 253,356 168,440 31,132 137,308
chrome.exe 5780 0 129,256 147,004 45,936 101,068
explorer.exe 8196 0 73,840 112,308 67,784 44,524
Steam.exe 7260 0 119,708 112,076 38,028 74,048
Dropbox.exe 9448 0 90,512 103,160 31,888 71,272
svchost.exe (LocalSystemNet... 1868 0 106,040 96,004 2,808 93,196
MsMpEng.exe 1916 0 102,024 89,840 26,668 63,172
java.exe 13212 0 255,864 81,272 12,564 68,708
chrome.exe 9348 0 81,504 76,820 17,812 59,008
Physical Memory 3763 MB In Use 4369 MB Available
Hardware Reserved 30 MB
In Use 3763 MB
Modified 30 MB
Standby 2362 MB
Free 2007 MB
Available 4369 MB
Cached 2392 MB
Total 8162 MB
Installed 8192 MB
Views
Used Physical Memory 100%
60 Seconds 0%
Commit Charge 100%
0%
Hard Faults/sec 100
0


Task Manager
File Options View
Processes Performance App history Startup Users Details Services
CPU 37% 2.94 GHz
Memory 3.7/8.0 GB (46%)
Disk 0 (D:) 3%
Disk 1 (C:) 7%
Disk 2 (D:) 1%
Disk 3 (D:) 0%
Disk 4 (E:) 0%
Ethernet S: 152 R: 40.0 Kbps
CPU
AMD Athlon(tm) X4 740 Quad Core Processor
% Utilization 100%
60 seconds 0
Utilization 37%
Speed 2.94 GHz
Processes 90
Threads 1824
Handles 42161
Up time 7:03:58:13
Maximum speed: 3.20 GHz
Sockets: 1
Cores: 2
Logical processors: 4
Virtualization: Enabled
L1 cache: 192 KB
L2 cache: 4.0 MB
Fewer details
Open Resource Monitor

Conclusies & bevindingen

Onderstaand worden de conclusies en bevindingen toegelicht.

Het doel van de opdracht is het bepalen en oplossen van bottlenecks in de applicatie, waardoor er uiteindelijk 8000+ weerstations ondersteund kunnen worden. Elk weerstation levert elke seconde een meting van bepaalde waarden (bijv. temperatoor, windrichting, etc.). Belangrijk hierbij is dat de applicatie schaalbaar is en in de toekomst een grote hoeveelheid weerdata (bijv. weerdata van het afgelopen jaar), maar ook meer weerstations, aan moet kunnen.

De uitwerking in dit rapport, naar aanleiding van de tweede leertaak, borduurt verder op de resultaten uit de eerste leertaken. Bij de eerste leertaak is een relationele database (MySQL) gebruikt, die de bottleneck voor grote hoeveelheden weerdata bleek te vormen. Dit probleem is in dit rapport opgelost door het gebruik van een non-relationele database (MongoDB).

Conclusie

Met het gebruik van MongoDB als database kan het doel van het verwerken van de meetgegevens van 8000+ weerstations makkelijk gehaald worden. Omdat de vorige versie van de applicatie al de 8000+ weerstations kon verwerken was er minimaal werk nodig om deze geschikt te maken voor gebruik van MongoDB. Ook de database zelf schaalt goed mee; zoals uit de stresstests is gebleken schaalt de datbase lineair mee over langere periode van verwerken van meetgegevens.

Na een stresstest van een uur was ongeveer 8 GB aan data opgeslagen in de database. Als we uit gaan van de lineaire stijging in database opslag zoals geconstateerd bij de stresstests dan betekent dit dat elk jaar bijna 60 TB aan data wordt opgeslagen. Het opslaan van de meetgegevens van 8000+ weerstations per seconde is mogelijk excessief. Gekeken zou kunnen worden naar het samenvoegen van data na een bepaalde periode of het opslaan van de meetgegevens in een lagere frequentie.

Bevindingen

Onderstaand de bevindingen tijdens het uitwerken van de tweede leertaak.

Eliminatie RecordBuffer

Tevens zijn er optimalisaties in de applicatie toegepast. Zo wordt er geen gebruik meer gemaakt van een *RecordBuffer* die de weerdata objecten even vasthoudt om deze vervolgens in een batch in te schieten. Door de grote doorvoersnelheid van MongoDB kan elke inkomende record direct in de database worden geschoten.

Voorheen had de applicatie bij het afsluiten een aantal seconden (oplopend tot meerdere minuten) nodig om de deels gevulde buffers te verwerken. Door eliminatie van deze buffers is dit niet meer nodig, waardoor bij een mogelijk vastlopen van de applicatie geen weerdata verloren zal gaan.

Schaalbaarheid

Omdat een non-relationele database als Mongo DB gemaakt is voor grote datasets, is het schalen van deze database dan ook zeer eenvoudig. Wanneer de data blijft groeien, kan het voorkomen dat een enkele machine niet meer voldoende is voor het verwerken van de data. Dit wordt opgelost met een techniek die **Sharding** (of 'horizontaal schalen') heet. Hiermee worden de werklast en datasets verdeeld over meerdere machines ('shards'), waardoor de database horizontaal schaalt.