E4All supply en demand matcher

Introductie

In het kader van het E4All project, met als beoogd doel een smartgrid te ontwikkelen waarbij slimme huishoudens energieposities met elkaar kunnen uitwisselen, is er een matching tool ontwikkeld. Deze matching tool is de eerste stap in het proces om tot een digitale marktplaats te komen waar energieposities kunnen worden verhandeld.

Ter voorbereiding op de software ontwikkeling is er een kort onderzoek gedaan naar bestaande projecten op het gebied van slimme energie oplossingen. Hieruit is het Flexines project naar voren gekomen. Op basis van de inzichten die gedurende dit project zijn opgedaan zijn gebruikt tijdens het ontwikkelen van de eerste versie van de matching software. Met name de eisen die gesteld worden aan een digitale marktplaats voor energieposities zijn hierbij in ogenschouw genomen.

Het doel van de eerste versie van de energie matching software is om het mogelijk te maken gesimuleerde energieposities uit te wisselen. Studenten van de Hanzehogeschool Groningen (HG) zullen vanaf September 2018 simulaties ontwikkelen om energie vraag en aanbod data te creëren. Hieruit zullen, naar verwachting, nieuwe eisen voor de matching software voortkomen die gebruikt kunnen worden in het ontwikkeltraject om tot een digitale marktplaats te komen.

Methode

In deze sectie word er ingegaan op de matching software en de manier waarop de software is getest. De software architectuur, matchingstrategieën en testopstelling zullen hierbij aan bod komen. Informatie over de mappenstructuur en de ondersteunde REST-api calls zijn te vinden in de bijlagen van dit document. Deze informatie is van belang in het geval er gebruik gemaakt dient te worden van de matching software, maar word niet verder behandeld in dit onderdeel van het document.

Software Architectuur

Voor de ontwikkeling van de matching POC is er een software stack ontwikkeld bestaande uit een Flask-based REST api en een MongoDB database. De REST api ontvangt HTTP requests die worden gebruikt om nieuwe gegevens op te slaan of bestaande gegevens op te vragen.

Nieuwe energie vraag of aanbod wordt direct vastgelegd in de database. De matcher wordt in dit geval gebruikt om de nieuwe energiepositie te matchen met bekende energieposities. De uitkomst van deze matching resulteert in transacties die tevens in de database worden opgenomen.

Vastgelegde data wordt zonder tussenkomst van de matcher direct vanuit de database geserveert aan de client. Voor overzichtspagina’s zoals de dashboard en index pagina’s worden tevens afgeleide gegevens runtime berekend met behulp van statistiekmodules. Deze modules gebruiken de gegevenscollecties om statistieken zoals gemiddelden en verhoudingen te berekenen. Hiermee word meer inzicht verworven in de performance van de matching software.

Matchingstrategiën

De matcher maakt gebruik van allocatiestrategiën die gebaseerd zijn op bekende algoritmen uit de memory management. Dit zijn de first fit, best fit en worst fit allocatie-algoritmen. Bekende en nieuwe vraag en aanbod word tegen elkaar afgezet en op basis van deze allocatiemethoden aan elkaar gekoppeld.

Voor de POC is ervoor gekozen om vraag en aanbod uitsluitend te matchen wanneer het aanbod de vraag volledig kan voorzien. Deze benadering vergemakkelijkt het matching process en is tevens als oplossing terug te vinden in Bijlage 17 van het Flexines project eindrapport *(http://flexines.org/publicaties/eindrapport/BIJLAGE17.pdf)*.

Gedurende het testen wordt er gekeken naar de impact van de matchingstrategiën op de energieuitwisseling. Hierbij zijn gegevens als gemiddeld gealloceerde vraag en aanbod, totaal gealloceerde vraag en aanbod, en totale energie allocatie indicatief.

First Fit

First fit is een strategie waarbij de eerst mogelijke set van vraag en aanbod gematched wordt wanneer het aanbod de vraag volledig kan voorzien. Deze naïve manier van matching is gemakkelijk te implementeren, maar hoeft niet in alle gevallen tot de beste verdeling van vraag en aanbod te leiden.

Best Fit

Best fit matching zoekt naar een set van vraag en aanbod, waarbij het verschil tussen vraag en aanbod minimaal is. Met deze strategie wordt de best passende combinatie van vraag en aanbod gevonden.

Worst fit

Worst fit is een matching strategie waarbij gezocht word naar een set van vraag en aanbod, waarbij het verschil tussen vraag en aanbod maximaal is. Met deze strategie wordt de grootst passende combinatie van vraag en aanbod gevonden.

Testopstelling

De matcher is getest met behulp van Apache JMeter. Deze software maakt het mogelijk met meerdere threads gelijktijdige aanvragen uit te voeren op de REST api. Vraag en aanbod van energie word gelijktijdig door 100 threads aangeboden. In totaal worden 5000 energievragen en 5000 energieaanbiedingen naar de matcher gestuurd. De hoeveelheid energie die gevraagd of aangeboden word, is willekeurig gemaakt door middel van een pseudorandomizer. Belangrijke gegevens die tijdens het testen worden verzameld zijn:

- *responsetijden*
- *hoeveelheden geslaagde en gefaalde requests*

Daarnaast is het van belang om erop toe te zien dat de data op een juiste wijze word verwerkt. De validiteit van de applicatie wordt getoetst door de totale vraag, het totale aanbod en de gecreëerde transacties in ogenschouw te nemen. Concreet betekend dit dat transacties moeten voldoen aan de volgende voorwaarden:

- *Per transactie mag het totaal aan verhandelde energie nooit groter zijn dan de aangeboden en gevraagde hoeveelheid energie.*
- *Voor elke valide set van vraag en aanbod moet een transactie bestaan*.

Resultaten

In dit onderdeel word ingegaan op de resultaten na het uitvoeren van de stresstest. Per allocatiestrategie worden de performance gegevens gepresenteerd.

De totale supply is 502208, de totale demand is 501512. Hiermee is de supply/demand ratio vastgesteld op 100.14%. Het aanbod van energie is groter dan de gevraagde hoeveelheid energie.

First Fit

Matching resultaten:

Transacties	Totale energie-allocatie	Gem. energietransactie	Gem. vraagallocatie
4827	470436	97.46	96.54% (σ = 18.28)

Gem. aanbodallocatie	Gealloceerde vraag	Gealloceerde aanbod
93.67% (σ = 33.77)	93.80%	93.67%

Alle mogelijke combinaties van vraag en aanbod zijn uitgeput en voor iedere match is een transactie aangemaakt.

Demand request resultaten:

Gem. responsetijd	Min. responsetijd	Max. responsetijd	Errors	Throughput
527ms (σ = 120.6)	11ms	1015ms	0	20.39 Kb/sec.

Supply request resultaten:

Gem. responsetijd	Min. responsetijd	Max. responsetijd	Errors	Throughput
529ms (σ = 121.8)	17ms	1037ms	0	19.59 Kb/sec.

Best Fit

Matching resultaten:

Transacties	Totale energie-allocatie	Gem. energietransactie	Gem. vraagallocatie
4844	479903	99.07	96.88% (σ = 17.39)

Gem. aanbodallocatie	Gealloceerde vraag	Gealloceerde aanbod
83.83% (σ = 31.25)	95.69%	95.56%

Alle mogelijke combinaties van vraag en aanbod zijn uitgeput en voor iedere match is een transactie aangemaakt.

Demand request resultaten:

Gem. responsetijd	Min. responsetijd	Max. responsetijd	Errors	Throughput
524ms (σ = 151.7)	283ms	992ms	0	18.87 Kb/sec.

Supply request resultaten:

Gem. responsetijd	Min. responsetijd	Max. responsetijd	Errors	Throughput
527ms (σ = 151.9)	282ms	991ms	0	21.46 Kb/sec.

Worst Fit

Matching resultaten:

Transacties	Totale energie-allocatie	Gem. energietransactie	Gem. vraagallocatie
4778	458154	95.89	95.56% (σ = 20.60)

Gem. aanbodallocatie	Gealloceerde vraag	Gealloceerde aanbod
84.19% (σ = 35.11)	91.35%	91.23%

Alle mogelijke combinaties van vraag en aanbod zijn uitgeput en voor iedere match is een transactie aangemaakt.

Demand request resultaten:

Gem. responsetijd	Min. responsetijd	Max. responsetijd	Errors	Throughput
523ms (σ = 165.8)	274ms	1170ms	0	21.79 Kb/sec.

Supply request resultaten:

Gem. responsetijd	Min. responsetijd	Max. responsetijd	Errors	Throughput
527ms (σ = 166.9)	276ms	1162ms	0	18.53 Kb/sec.

Conclusie

Alle matchingstrategieën voldoen aan de voorwaarden die gesteld zijn aan transacties. Hiermee is duidelijk dat vraag en aanbod zo volledig mogelijk kan worden gealloceerd

Op basis van de behaalde resultaten is naar voren gekomen dat de best fit matchingstrategie, gegeven de testdata, tot de beste uitwisseling van energieposities leidt. Om deze reden is ervoor gekozen de best fit allocatiealgoritme als standaard matchingstrategie te gebruiken.

Discussie

De huidige POC is in staat om energieposities te matchen op basis van energie aanbod en energie vraag. Het totaal aan enegie vraag en aanbod wordt gemiddeld voor 96% geallocceerd.

API

De POC maakt gebruik van Flask waarvan in de documentatie staat aangegeven dat de server niet direct geschikt is voor gebruik in een productie omgeving *(http://flask.pocoo.org/docs/1.0/deploying/)*. Voor een volwaardig eindproduct dient daarom gezocht te worden naar een REST-api framework of deployment techniek die wel geschikt is voor productie.

Matching

De huidige oplossing is in staat om energie vraag en aanbod van huishoudens te ontvangen en te vertalen naar transacties. Hierbij wordt gekeken naar de gevraagde en aangeboden hoeveelheden energie.

Vraag en aanbod van energie wordt direct gematched met in de database aanwezige energieposities. Dit in een poging om de aanvrager zo snel mogelijk van een reactie te voorzien. Het is onduidelijk in hoeverre het wachten op meer data met betrekking tot vraag en aanbod tot betere uitwisseling van energieposities leidt.

Vraag en aanbod worden pas gematched wanneer de aanbod volledig kan voorzien in de vraag. Dit leidt in veel gevallen tot situaties waarbij niet alle aangeboden energie daadwerkelijk word benut. Dit geldt ook in gevallen waarbij de vraag naar energie groter is dan het aanbod van energie. Een aanpassing op het matching algoritme is daarom nodig. Om energieposities beter uit te wisselen moet er per energiepositie gezocht worden naar de kleinste set van vraag en aanbod wat leidt tot een volledige voorziening van vraag of benutting van aanbod.

De data die tijdens de stresstest is gebruikt om de performance van de matchingstrategieën tegen elkaar af te zetten hoeft niet representatief te zijn voor werkelijke energieposities. Om deze reden is ervoor gekozen om alle allocatiemethoden instelbaar te maken in de E4All matcher. De volgens de test meest geschikte allocatiemethode is hierbij als standaard ingesteld.

Database

Concurrency control wordt op het moment niet afgevangen door de DBMS. Het voorkomen van race conditions is opgelost door middel van een mutex lock op de matching methode. Het migreren van de concurrency control verantwoordelijkheid naar de persistence laag is wenselijk. Hierbij moet gedacht worden aan het locken van de database op record niveau. Dit niveau van locking maakt het mogelijk om gemakkelijker software te ontwikkelen die om kan gaan met grote hoeveelheden gelijktijdige aanvragen zonder dat de consistentie van de data afneemt.

Tevens is het belangrijk om database transacties te introduceren. Hierdoor kunnen energietransacties atomair worden vastgelegd. MongoDB biedt hiervoor onvoldoende ondersteuning gezien locking uitsluitend op document niveau word gefaciliteerd *(https://docs.mongodb.com/manual/core/write-operations-atomicity/)*. Hierdoor worden er momenteel geen rollbacks van energietransacties uitgevoerd wanneer er fouten optreden in het systeem.

Statistieken

Afgeleide gegevens, zoals gemiddelde vraag en aanbod waarden worden runtime berekend bij een aanvraag. De matcher is dusdanig opgebouwd dat er sprake is van een *O(n)* probleem. Dit houdt in dat met het groeien van de hoeveelheid data in de database de reactietijd van de applicatie afneemt. Een potentieële oplossing hiervoor is om afgeleide gegevens vast te leggen en te updaten bij nieuwe data. Deze gegevens kunnen dan direct aangeboden worden vanaf de database. Deze vorm van pre-processing kan de afnemende reactietijd tegengaan.

Schaalbaarheid

De schaalbaarheid van de huidige applicatie is gelimiteerd. Concreet houdt dit in dat de applicatie in zijn geheel opgeschaald moet worden in het geval van een performance bottleneck. Een flexibeler oplossing is het verdelen van de matcher in kleinere componenten. Door de REST-api, het matching algoritme en de database laag in verschillende containers te hosten kunnen specifieke onderdelen van de matcher worden opgeschaald. Voor de POC is ervoor gekozen een simpeler architectuur in te zetten gezien de noodzaak van containers, en de daarbij horende extra complexiteit, nu nog niet bestaat.

Prioriteitsmatching

Een strategie die niet is uitgewerkt in de huidige POC is het matchen van vraag en aanbod op basis van prioriteit. Hierbij krijgt een energie vragende of aanbiedende partij prioriteit over andere partijen. Matching op basis van prioriteit kan worden gecombineerd met first fit en best fit matching. Op dit moment is het niet duidelijk of matching op basis van prioriteit wenselijk is.

Timeslots

Er wordt geen onderscheid gemaakt tussen timeslots bij het matchen van vraag en aanbod van energie. Dit is in de toekomst wel noodzakelijk om elementen zoals day-ahead matching te faciliteren.

Prijzbepalingen

Energieposities zijn momenteel niet voorzien van een prijs. Hierdoor is het niet mogelijk om biedingen te plaatsen voor energieposities. Een toekomstige versie van de matcher zal in staat moeten zijn om vraag en aanbod te matchen op basis van de gevraagde en aangeboden prijs.

Vervolg

Het is raadzaam de uitkomsten van de simulaties af te wachten en in te spelen op de wensen en eisen die hieruit voortkomen. Op deze wijze kan de E4All matcher worden verbeterd en uitgebreid. De uiteindelijke vervolgstap dient uiteraard in overleg vastgesteld te worden alvorens er tot ontwikkeling overgegaan word.

Modules

Onderstaande overzicht geeft de mappenstructuur van de matching software weer.

E4All/			*Root Directory*
|----------	models/		
|	|----------	models	*E4All modellen*
|			
|----------	static/		
|	|----------	css/	*Style sheets*
|			
|----------	templates/		
|	|----------	dashboard.html	*Dashboard template*
|	|----------	demand.html	*Demand template*
|	|----------	demand_index.html	*Demand index template*
|	|----------	layout.html	*Layout template*
|	|----------	supply.html	*Supply template*
|	|----------	supply_index.html	*Supply index template*
|	|----------	transaction.html	*Transaction template*
|	|----------	transaction_index.html	*Transactions index template*
|			
|----------	util/		
|	|----------	matcher.py	*Vraag en aanbod matcher*
|	|----------	persistence.py	*MongoDB client*
|	|----------	statistics.py	*Statistieken modules*
|			
|----------	.gitignore		*Gitignore bestand*
|----------	api.py		*Flask-based REST api*
|----------	readme.md		*Readme bestand*
|----------	requirements.txt		*E4All matcher dependencies*

REST API

In dit onderdeel worden de ondersteunde REST api calls behandeld. Wanneer de Flask applicatie is gestart (`$ FLASK_APP=api.py flask run`) kunnen onderstaande calls worden gestuurd naar *localhost:5000*. Verder zijn er URL's die bedoeld zijn om met behulp van een browser te benaderen. Te weten:

http://localhost:5000/	Dashboard pagina
http://localhost:5000/demand	Demand index pagina
http://localhost:5000/supply	Supply index pagina
http://localhost:5000/transactions	Transacties index pagina

Create Demand	
Route	/demand
Methode	POST
URL Parameters	-

Data Parameters	**Example**
	{
	"timeslot_id": 1,
	"amount": 48,
	"energy_type": "buffer",
	"start": "02-03-2018 13:45",
	"residence_id": 1,
	"duration": 100
	}

Succses Response	**Example**
	Code: 200
	Content:
	{
	"_id": "5b717240260d2a726f09cc95",
	"amount": 48,
	"duration": 100,
	"energy_type": "buffer",
	"remaining": 0,
	"residence_id": 1,
	"satisfied": 48,
	"start": "02-03-2018 13:45",
	"timeslot_id": 1,
	"transactions": [
	{
	"_id": "5b717240260d2a726f09ccb0",
	"amount": 48,
	"demand_id": "5b717240260d2a726f09cc95",
	"supply_id": "5b717240260d2a726f09ccaf"
	}
	]
	}

Error Response	**Example** **Code:** 400 **Content:** ValueError: time data '02-02-20' does not match format '%d-%m-%Y %H:%M'
Omschrijving	Maakt een nieuwe energie demand aanvraag aan.

Get Demand	
Route	/demand/:demand_id
Methode	GET
URL Parameters	**Required:** demand_id=[alphanumeric]
Data Parameters	-

Succses Response	**Example**
	Code: 200
	Content:
	{
	"_id": "5b717240260d2a726f09cc95",
	"amount": 48,
	"duration": 100,
	"energy_type": "buffer",
	"remaining": 0,
	"residence_id": 1,
	"satisfied": 48,
	"start": "20-02-2030 13:45",
	"timeslot_id": 1,
	"transactions": [
	{
	"_id": "5b717240260d2a726f09ccb0",
	"amount": 48,
	"demand_id": "5b717240260d2a726f09cc95",
	"supply_id": "5b717240260d2a726f09ccaf"
	}
	]
	}

Error Response	**Example** **Code:** 404 **Content:** Demand with id: 5b717240260d2a726f09cc95 not found.
Omschrijving	Vraagt informatie over een bestaand demand request op.

Create Supply	
Route	/supply
Methode	POST
URL Parameters	-

Data Parameters	**Example**
	{
	"timeslot_id": 1,
	"amount": 48,
	"energy_type": "buffer",
	"start": "02-03-2018 13:45",
	"residence_id": 1,
	"duration": 100,
	"price": 12.5
	}

Succses Response	**Example**
	Code: 200
	Content:
	{
	"_id": "5b717240260d2a726f09cca8",
	"amount": 48,
	"duration": 100,
	"energy_type": "buffer",
	"price": 12.5,
	"remaining": 0,
	"residence_id": 1,
	"start": "02-03-2018 13:45",
	"timeslot_id": 1,
	"transactions": [
	{
	"_id": "5b717240260d2a726f09cca9",
	"amount": 48,
	"demand_id": "5b717240260d2a726f09cca2",
	"supply_id": "5b717240260d2a726f09cca8"
	}
	]
	}

Error Response	**Example** **Code:** 400 **Content:** ValueError: time data '02-02-20' does not match format '%d-%m-%Y %H:%M'
Omschrijving	Maakt een nieuwe energie supply aanvraag aan.

Get Supply	
Route	/supply/:supply_id
Methode	GET
URL Parameters	**Required:** supply_id=[alphanumeric]
Data Parameters	-
Succses Response	**Example** **Code:** 200 **Content:**

	{
	"_id": "5b717240260d2a726f09cca8",
	"amount": 48,
	"duration": 100,
	"energy_type": "buffer",
	"price": 12.5,
	"remaining": 0,
	"residence_id": 1,
	"start": "02-03-2018 13:45",
	"timeslot_id": 1,
	"transactions": [
	{
	"_id": "5b717240260d2a726f09cca9",
	"amount": 48,
	"demand_id": "5b717240260d2a726f09cca2",
	"supply_id": "5b717240260d2a726f09cca8"
	}
	]
	}

Error Response	**Example** **Code:** 404 **Content:** Supply with id: 5b717240260d2a726f09cca8 not found.
Omschrijving	Vraagt informatie over een bestaand supply request op.

Get Transaction	
Route	/transactions/:transaction_id
Methode	GET
URL Parameters	**Required:** transaction_id=[alphanumeric]
Data Parameters	-

Succses Response	**Example**
	Code: 200
	Content:
	{
	"_id": "5c166515260d2a3b7fd60cf4",
	"amount": 200.0,
	"demand_id": "5c166515260d2a3b7fd60cf3",
	"supply_id": "5c166425260d2a3ade3db29f",
	"timeslot_id": 1
	}

Error Response	**Example** **Code:** 404 **Content:** Transaction with id: 5b717240260d2a726f09cc95 not found.
Omschrijving	Vraagt informatie over een bestaande transaction op.