Inhoudsopgave

Lijst van figuren

Lijst van tabellen

Lijst van listings

1

Inleiding

1.1 Probleemstelling

De samenleving wordt alsmaar meer afhankelijk van elektronisch communicatie. Op elk gegeven moment en locatie kan een elektronisch apparaat verbinding trachten te maken met een groter netwerk. Steeds meer apparaten dienen voorzien te worden van draadloze communicatie zoals IOT-apparaten beginnend van kleine sensoren tot zelfrijdende wagens.

Eens te meer wordt duidelijk waarom we aan de vooravond staan van de volgende generatie draadloze communicatie genaamd 5G. Deze nieuwe technologie moet in staat zijn om meer mensen te verbinden in grootheden van een miljoen connecties per vierkante meter terwijl een vertraging van enkele milliseconden wordt gehanteerd en snelheden tot wel 10 Gbps kunden oplopen. [**?**]

Ook in uitzonderlijke en mogelijks levensgevaarlijke situaties wordt er beroep gedaan op het draadloze netwerk. Tijdens de aanslagen in zaventem zagen telecomoperatoren een drastische stijging in communicatie tot op het punt waarbij het netwerk zo goed als verzadigd werd waarbij sommige operatoren genoodzaakt waren de stralingsnorm tijdelijk te overschreiden. [1]

Bloodstelling aan elektromagnetische straling kan echter niet licht opgenomen worden. Uit onderzoek blijkt dat bloodstelling aan straling kan leiden tot diverse aandoeningen zoals ... [2]

1.2 Doelstelling

todo

2

Cloud resourcebeheer, resource-allocatie en OpenStack

- reeds uitgevoerd onderzoek naar exposure - manets -> exosure combineren

In dit hoofdstuk wordt er dieper ingegaan op enkele belangrijke termen binnen cloud computing. Deze termen geven een beter idee over de werking van diverse cloud-systemen en de mogelijkheden die er zijn voor zowel de cloud-provider als de cloud-gebruiker, beiden besproken in Hoofdstuk **??** om hun cloud te beheren.

2.1 Cloud resourcebeheer

Om ervoor te zorgen dat een cloud provider en een cloud-gebruiker steeds voldoen aan hun SLA, moeten beide partijen *resource management* toepassen. Resource management heeft als doel de beschikbare bronnen zo efficiënt mogelijk te benutten, en precies voldoende bronnen te voorzien om *underload* en/of *overload* te vermijden.

2.1.1 Resourcebeheer: de mogelijkheden

Het doel van een cloud provider is het efficiënt beheren van de infrastructuur van het datacenter. Onderdelen hiervan zijn bijvoorbeeld *balanced load* waarbij resources op een welbepaalde manier

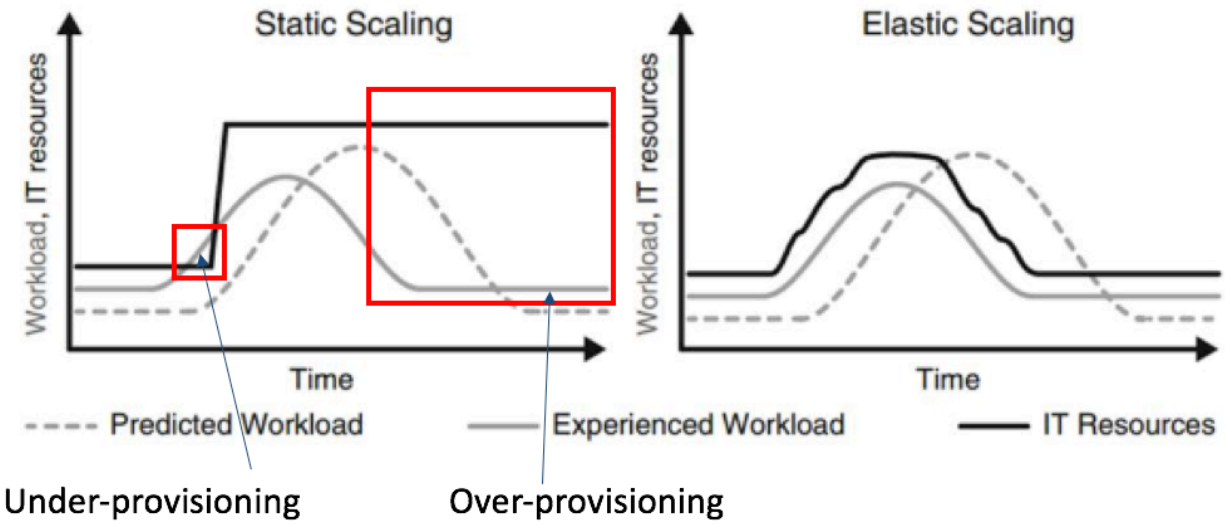
worden toegekend zodat het gebruik gebalanceerd is over alle resources van dat type, *fault tolerance* waarbij de resources worden toegekend zodat de impact van het falen van een onderdeel minimaal blijft, of *energy use minimization* waarbij de plaatsing van taken of jobs over resources zo gebeurt dat de energiekosten minimaal zijn. Een eventuele combinatie van bovenstaande onderdelen aan de hand van prioriteiten is mogelijk en de cloud provider kan ook bepaalde onderdelen koppelen aan verschillende operationele condities. Een voorbeeld van het laatste is het minimaliseren van het energieverbruik tijdens perioden met lage belasting, maar bij hogere belasting overschakelen naar balanced load in plaats van energy use minimization.

Een cloud-gebruiker heeft meestal een SLA met zijn eigen eindgebruikers met andere management doelen dan de cloud provider ten opzichte van zijn cloud-gebruikers. Hij maakt gebruik van de elasticiteit van cloud-omgevingen om zo extra resources te reserveren tijdens perioden met een hogere werklast en resources vrij te geven in perioden met minder werklast.

2.1.2 Schalen

Een cloud-gebruiker kan in twee probleemsituaties terecht komen indien hij verkeerd gebruikmaakt van de elasticiteit van cloud-omgevingen: *over-provisioning* en *under-provisioning*. Over-provisioning betekent dat de cloud-gebruiker meer resources ter beschikking heeft dan nodig met een hogere kost als gevolg. Doordat cloud computing ook een "business" is, zorgt dit voor nadelige effecten bij zowel de cloud-gebruiker als eventuele eindgebruikers. Anderzijds betekent under-provisioning dat de cloud-gebruiker te weinig resources ter beschikking heeft, waardoor hij bijvoorbeeld performantiecriteria in de SLA met zijn eindgebruiker niet kan nakomen. Ook dit is een zeer nadelig effect voor beide partijen aangezien een eindgebruiker dan langer moet wachten op een antwoord (langere antwoordtijd) of geen antwoord krijgt door het falen van componenten.

Schalen van de beschikbare resources (of gebruik maken van de elasticiteit van een cloud-omgeving) voorkomt grotendeels de twee bovenstaande nadelige effecten. Hierin worden twee grote categorieën onderscheiden, statisch of periodiek en horizontaal of verticaal schalen. Bij statisch schalen worden er meer fysieke servers voorzien maar de tijd om deze te ordenen, te configureren en op te starten zal niet reactief genoeg zijn om een snelle stijging van de werklast te kunnen verwerken. Het gevolg hiervan is eerst een under-provisioning en nadien een over-provisioning omdat de opstart te lang duurt en het verlagen van het aantal resources pas na een bepaalde tijd van mindere activiteit zal worden uitgevoerd. Bijgevolg zorgt dit voor een slechtere performantie in het begin, door de under-provisioning, en een te hoge kostprijs achteraf door de over-provisioning. Elastisch schalen daarentegen voorziet een stijging in resources met kleine stappen wat het systeem veel reactiever en kost-efficiënter maakt. Een belangrijke opmerking hierbij is dat het systeem zo ontworpen moet zijn dat het kan schalen, of met andere woorden,

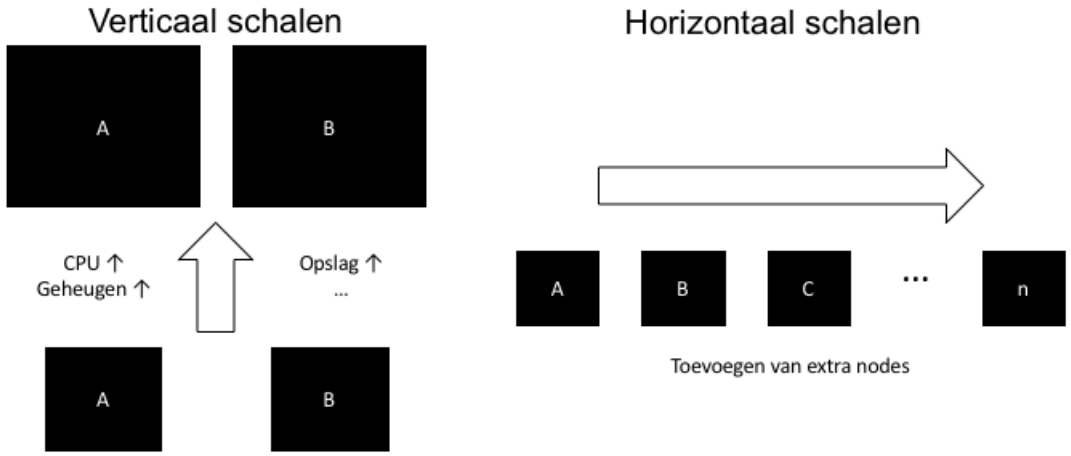


Figuur 2.1: Statische vs. elastische schaling [**?**].

nuttig gebruik kan maken van de extra verkregen resources. Een SQL-databank is een voorbeeld van een minder schaalbare toepassing [**?**] omdat het gebruikmaakt van transacties die het gehele gebeuren blokkeren tot er een *commit* wordt uitgevoerd. Het toekennen van extra resources heeft hier dus weinig nut omdat de toegang tot de gegevens geblokkeerd blijft.

In Figuur 2.1 worden twee grafieken weergegeven die de toegekende resources weergeven bij statisch of elastisch schalen. Zoals te zien is, hangt statisch schalen sterk af van eventuele voorspellingen van de werklast waardoor het minder performant is bij pieken. Ook de nadelige effecten van under- en over-provisioning worden duidelijk weergegeven in de linkergrafiek. De rechtergrafiek weerspiegelt de kost-efficiëntie en performantie bij elastisch schalen en voor dit scenario is het duidelijk de aangewezen werkwijze om te schalen.

Naast statisch of elastisch schalen kan er ook verticaal of horizontaal geschaald worden. Verticaal schalen wordt omschreven als het toevoegen van extra hardware zoals meer CPU-kracht, extra geheugen en opslag, etc. De voordelen zijn onder andere dat elke applicatie hier gebruik van kan maken omdat dit geen speciale architectuur vereist. Daarnaast omvat het een weinig risicovolle en minder complexe operatie dan horizontaal schalen en ten slotte is de kost ook redelijk in vergelijking met algoritmische verbeteringen. De nadelen wegen echter iets meer door dan de voordelen. Zo is er mogelijks een *downtime* als er fysieke hardware wordt toegevoegd en is het niet zeker dat de applicatie gebruik kan maken van de nieuwe hardware. Een voorbeeld van dat laatste is een *single-threaded* applicatie waarbij het toevoegen van extra CPU-kernen de applicatie niet zal versnellen en bijgevolg weinig nut heeft. Het is ook veel moeilijker om te *downgraden* bij een dalende werklast en bovendien is er een absolute bovenlimiet doordat de onderliggende hardware van de fysieke node beperkt is.



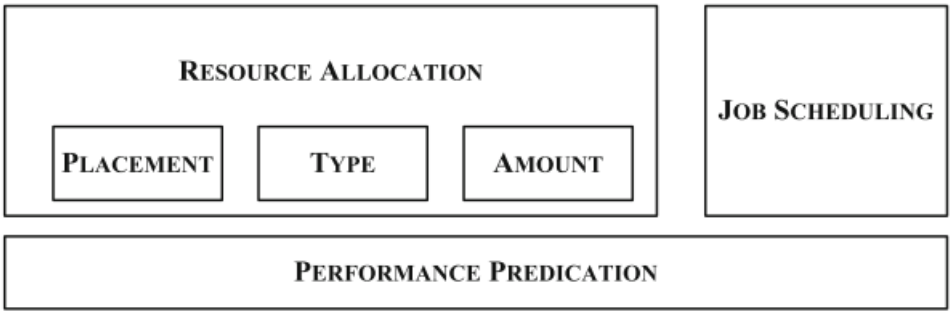
Figuur 2.2: Verticaal vs. horizontaal schalen van cloud resources

Horizontaal schalen voegt extra fysieke- of applicatienodes toe in plaats van deze te upgraden waardoor een veel hogere capaciteit mogelijk is dan de grootste beschikbare enkelvoudige node. Een belangrijk voordeel is de hoge kost-efficiëntie omdat het elastisch schalen toelaat. De enige vereiste is dat de applicatie of toepassing die erop draait een gedistribueerde architectuur moet bevatten zodat verschillende nodes dezelfde functionaliteit van een applicatie kunnen uitvoeren. Meestal zijn zulke nodes volledig identiek geconfigureerd –zelfde hardware resources, zelfde besturingssysteem, zelfde software– waardoor ze homogeen zijn. De homogeniteit van bepaalde nodes is een zeer belangrijke eigenschap bij horizontaal schalen omdat hierdoor *round-robin load balancing* goed werkt en het maakt capaciteitsplanning en auto-schalen veel eenvoudiger dan bij heterogene nodes. Figuur 2.2 geeft een overzicht van verticaal en horizontaal schalen.

2.2 Resource-allocatie

Resource allocation is een onderdeel van een overlappend concept samen met *resource provisioning* en *resource scheduling* [**?**]. Resource provisioning en resource-allocatie zorgen voor het toekennen van resources van een cloud provider aan een cloud-gebruiker in een concurrerende omgeving van groepen programma's of gebruikers. Resource scheduling geeft een overzicht weer van welke resources er beschikbaar zijn en welke er gedeeld worden op bepaalde tijdstippen om zo zware rekentaken te plannen. Samengevat beschrijven deze drie termen het bepalen van wanneer rekenactiviteit moet gestart of gestopt worden, gebaseerd op voorgaande activiteiten en relaties alsook op de *gealloceerde resources* en de duur van de activiteit.

In het algemeen bepaalt een cloud-gebruiker de hoeveelheid en het type van de resources die hij nodig heeft en zal de cloud provider deze aangevraagde resoures toewijzen in hun datacenters. Bij het bepalen van de hoeveelheid en het type van de resources moet er rekening gehouden worden



Figuur 2.3: Basisoverzicht van het resource-allocatie mechanisme.

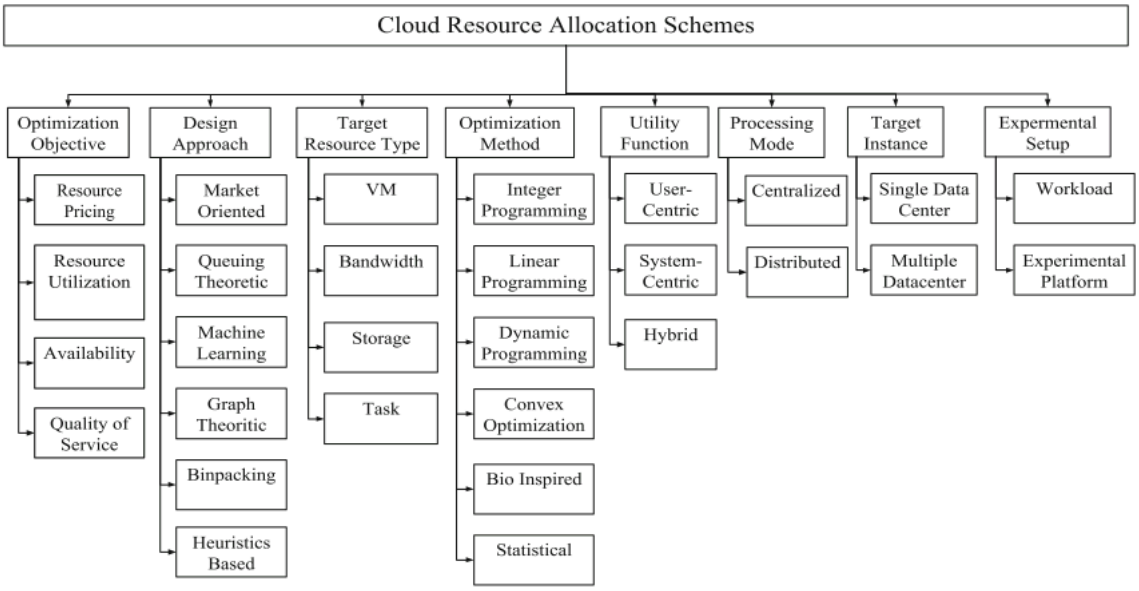
met eventuele vereisten zoals bijvoorbeeld de maximale duur van een taak. Zoals te zien in Figuur 2.3 beschrijft het cloud resource-allocatie mechanisme de beslissing in hoeveel, welke, waar en wanneer de beschikbare resources moeten toegekend worden aan de cloud-gebruikers. Dankzij de elasticiteit van cloud omgevingen kan een cloud-gebruiker resources dynamisch aanvragen en vrijgeven.

Cloud providers en cloud-gebruikers proberen uiteraard zoveel mogelijk winst te maken. Een cloud provider poogt dit door zoveel mogelijk virtuele machine's (VM's) uit te rollen op elke fysieke machine zodat de inkomsten hoog zijn en de investeringen laag. Een logisch gevolg hiervan is dat te veel VM's uitrollen op een fysieke machine kan leiden tot performantiedegradatie waardoor de cloud-gebruikers minder tevreden zijn. Cloud users daarentegen willen hun kost minimaal houden voor een maximale performantie, wat betekent dat ze efficiënt de nodige resources zullen reserveren voor hun applicatie.

Doordat de datacenters van cloud providers heterogeen toenemen en dus bestaan uit verschillende generaties van hardware, zullen verschillende cloud-gebruikers heterogene resources van de cloud provider gebruiken (en delen met andere cloud-gebruikers) zonder veel inzicht in de infrastructuur met als nadelig gevolg dat het hele systeem onvoorspelbaar wordt. Het doel van resource-allocatie is zorgen voor efficiënte cloud-services, waarbij efficiënt staat voor het toekennen van de juiste resources aan de juiste applicatie op het juiste tijdstip. Hierdoor kunnen applicaties deze resources nuttig gebruiken.

2.2.1 Indeling van cloud resource-allocatie schema's

Zoals weergegeven in Figuur 2.4 kunnen de cloud resource-allocatie schema's onderverdeeld worden in acht categorieën [**?**]. Deze worden hieronder per categorie kort besproken.



Figuur 2.4: Indeling van resource-allocatie schema's voor cloud computing [**?**].

Optimalisatiedoel (*Optimization objective*)

De huidige resource-allocatie schema's pogen vier optimalisatiedoelen te volbrengen: *Resource pricing* formuleert prijsmodellen om het algemeen verbruik van cloud resources te verbeteren waardoor cloud resources gekocht, verkocht en geruild kunnen worden met andere cloud providers en/of cloud-gebruikers. *Resource utilization* maximaliseert het gebruik van de virtuele en fysieke resources voor een betere performantie en is zeer gerelateerd met energieconsumptie. *Availability* verzekert een bepaalde beschikbaarheid tijdens een contractuele periode door het dupliceren van taakuitvoering op resources of het dupliceren van data op verschillende geografische locaties, om zo om te kunnen gaan met x-voudige *failures*. *Quality of Service* zorgt voor een snelle responsetijd, weinig latency, etc.

Ontwerpbenadering (*Design approach*)

De ontwerpbenadering representeert het onderliggende model van de resource-allocatie en komt voor in vijf verschillende types. Het marktgeoriënteerd model (*market-oriented*) benadert de functies van het vraag en aanbod systeem. Het *queuing theoretic model* voorspelt verschillende performantiemetrieken en bijgevolg de nodige resources om de werklast te kunnen verwerken. *Machine learning models* bekijken de resource-allocatie als een voorspellingsfunctie waarbij resultaten worden verwerkt met statistische technieken om zo verschillende patronen te ontdekken van bijvoorbeeld aanvragen, onzekerheidsfactoren, etc. *Graph theoretic models* abstraheren cloud-systemen en zijn componenten voor het opstellen van modellen alsook voor het optimali-

seren van de resource-allocatie aan de hand van grafen waarin de taken en de *workflow* worden weergeven. Het *bin packing model* plaatst verschillende items in een minimaal aantal *bins* en dat wordt hier gebruikt om VM's te combineren op de fysieke nodes en om vrijgegeven resources consequent samen te nemen met als voordeel een efficiënter energieverbruik. Samengevat bevindt het cloud resource-allocatie probleem zich in de NP, NPC en NP-hard complexe klassen. Oplossingen baseren zich daarom op heuristische methoden, die minder complex en sneller zijn, om dit probleem op te lossen.

Type van doel-resource (*Target resource type*)

Het *target resource type* bepaalt hoe de extra resources worden afgeleverd aan de cloud-gebruiker of eindgebruiker. Afhankelijk van de vraag kan er bijvoorbeeld een extra VM ter beschikking worden gesteld, extra opslagruimte of bandbreedte voorzien worden, etc.

Optimalisatiemethode (*Optimization method*)

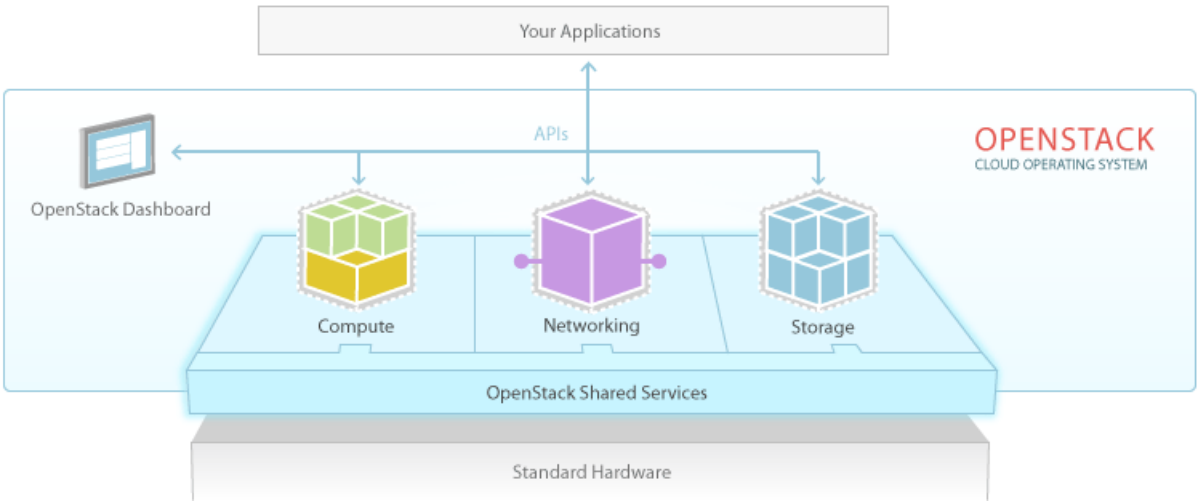
De optimalisatiemethode beschrijft de mathematische formulering voor het resource-allocatie probleem om zo een optimale of bijna-optimale oplossing te bepalen. Doordat de vraag en budgetten van gebruikers steeds variëren zijn polynomiale tijdsalgoritmen ongeschikt voor het oplossen van dit probleem en worden er daarom verschillende optimalisatiemethoden gebruikt om bijna-optimale oplossingen te creëeren zoals *integer programming*, genetische algoritmen, etc.

Nutsfunctie (*Utility function*)

De nutsfunctie bepaalt of het resource-allocatie schema een *system-centric*, *user-centric* of *hybrid* nut volgt. Deze drie types beschrijven hoe de cloud zal reageren indien het geen nieuwe resources meer kan alloceren. Bij system-centric staat de cloud provider zijn performantie centraal, bij user-centric staan de cloud-gebruikers centraal en het hybride-type bevindt zich tussen beiden.

Verwerkingsmodus (*Processing mode*)

Het doel van de nutsfunctie kan behaald worden door een optimalisatiemethode te gebruiken in een oftewel gecentraliseerde processing modus of een gedecentraliseerde modus. De gedecentraliseerde modus heeft als voordeel dat het geen *single-point of failure* is maar het kan, in vergelijking met de gecentraliseerde modus, geen globale informatiestatus bevatten.



Figuur 2.5: Architectuur van OpenStack [**?**].

Doel-instantie (*Target instance*)

De doel-instantie bepaalt of het resource-allocatie schema moet werken in een *single data center (SDC)* of een *multiple data center (MDC)* cloud omgeving.

Experimentele setup (*Experimental setup*)

Elk nieuw resource-allocatie schema wordt eerst getest in een experimentele omgeving waarbij telkens het platform en de werklast bepaald moeten worden.

2.3 OpenStack: een open-source softwareplatform voor cloud computing

OpenStack [**?**] is open-source cloud computing software voor het beheren en onderhouden van clouds. Het OpenStack project is gestart in 2010 door Rackspace en NASA die verantwoordelijk zijn voor de eerste versie van de code. Ondertussen is OpenStack al uitgegroeid tot een ruime community met ondersteuning van meer dan 150 bedrijven [**?**].

Figuur 2.5 geeft de basisarchitectuur weer van OpenStack, waarbij duidelijk wordt dat het bestaat uit verschillende onafhankelijke componenten met een welgevormde *Application Programming Interface* (API). OpenStack *Compute* –Nova– zorgt voor het beheren van VM's. *Storage* zorgt dan weer voor het beheren van de opslag en *Networking* zorgt voor de netwerkresources. Het OpenStack *Dashboard* is een webinterface waarmee de 3 bovenstaande blokken van com-

ponenten beheerd worden, zoals het starten of stoppen van VM's, virtuele netwerktopologieën maken, etc.

Naast de vier aanwezige basiscomponenten in Figuur 2.5, bestaan er nog verschillende andere componenten die deel uitmaken van het OpenStack project. Zo zijn er de *Orchestration*-services Heat [**?**], de Telemetry-Services [**?**] *Ceilometer* en *Aodh*, etc. Bovendien bestaan er naast componenten ook andere toepassingen die ontwikkeld zijn voor en door OpenStack. Zo is er bijvoorbeeld het *Heat Orchestration Template* (HOT) [**?**] formaat, ontwikkeld door community van OpenStack voor het aanmaken en configureren van verschillende onderdelen in Heat, Ceilometer, Nova, Neutron, etc.

Op dit ogenblik bevindt OpenStack zich in een vijftiende release, Ocata genaamd.[1] Deze release is de opvolger van Newton en brengt enkele verbeteringen aan zoals betere stabiliteit en een versterkte *core*-infrastructuur maar ook enkele nieuwe functies zoals de integratie van containers [**?**].

[1] Februari 2017

3

Gerelateerd werk

Aangezien resource allocatieschema's van clouds nog steeds worden onderzocht, bestaan er enkele werken die gerelateerd kunnen worden aan deze masterproef. In Sectie 3.2 wordt nadien een overzicht gegeven van reeds ontwikkelde cloud resource allocatieschema's.

3.1 Relevant onderzoek

Een zeer gerelateerd werk van Maenhaut et al. [**?**] beschrijft een Raspberry Pi cluster voor het valideren van cloud management strategieën. Dankzij een eenvoudige webinterface kan in deze demo alles eenvoudig geconfigureerd en getest worden. De Node.js-agent beschreven in Sectie **??** heeft dan ook inspiratie gehaald uit deze demo mits enkele aanpassingen en verbeteringen omdat hier gebruik wordt gemaakt van fysieke servers in plaats van Raspberry Pi clusters.

RT-OpenStack [**?**] plugt een CPU-resource management systeem in op een OpenStack omgeving. Het is een systeem bestaande uit 3 onderdelen, namelijk de integratie van een real-time hypervisor, een real-time scheduler en een VM-to-host mapping strategie wat dus vooral de nadruk legt op real-time VM's. Een belangrijke onderdeel van dat werk is de manier waarop RT-OpenStack werkt in combinatie met een normale OpenStack cloud-omgeving maar in deze thesis wordt er geen gebruik gemaakt van een aangepast management systeem noch van aangepaste hypervisors

voor de evaluatie van allocatieschema's.

Een ander voorbeeld van een inplugbaar framework in OpenStack is OpenStack Neat [**?**]. OpenStack Neat is eenvoudig inplugbaar in een bestaande OpenStack cloud-omgeving en gaat zorgen voor dynamische en energie-efficiënte algoritmen om VM's te consolideren. Daarbovenop biedt het ook de mogelijkheid om nieuwe VM consolidering algoritmen te evalueren en te vergelijken. De structuur van het framework en de werking ervan zijn zeer relevant doch ligt in deze thesis de focus meer op de correcte werking van het algoritme in plaats van de energie-efficiëntie.

Daarnaast bestaan er ook verschillende simulators om nieuwe schema's te testen zoals bijvoorbeeld OpenStackEmu [**?**]. Hierbij wordt een OpenStack omgeving gesimuleerd samen met een *Software Defined Networking (SDN)* controller en een large-scale network emulator CORE (*Common Open Research Emulator*). Dergelijke oplossing benadert al iets meer een reëele cloud-omgeving en de gebruikte methode is gerelateerd aan de werking van deze thesis. Enkel wordt er hier getracht om een echte cloud-omgeving te gebruiken in plaats van te benaderen of te simuleren.

Een ander voorbeeld van een simulatietool wordt beschreven door Maenhaut et al. [**?**] waarbij verschillende nieuwe schema's worden getest met behulp van verschillende invoerparameters. Deze resultaten geven een goed oog op de mogelijk performantie maar opnieuw gaat het om een simulator waarbij mogelijke constanten van een echte cloud-omgeving verborgen blijven. Deze thesis probeert daarom deze verborgen constanten zichtbaar te maken door de evaluatie uit te voeren op een reëel cloud-testbed.

Ten slotte bestaan er ook enkele interfaces om eenvoudig met OpenStack te communiceren zoals Apache Libcloud [**?**], een python bibliotheek, of pkgcloud [**?**], een node.js-pakket. Via een eenvoudige interface bieden ze een mogelijkheid aan om met verschillende cloud-besturingssystemen, zoals onder andere OpenStack, Google Cloud Platform, CloudFlare, etc te communiceren en hierop commando's uit te voeren. Indien deze interfaces gebruikt kunnen worden bij bijvoorbeeld de evaluatie van zo'n nieuw schema kan dit de overstap naar een andere cloud-omgeving zeer eenvoudig maken aangezien de commando's dezelfde blijven en de interface deze automatisch zal omvormen naar de commando's typisch voor het gebruikte cloud-systeem. Deze bibliotheken bieden dus de mogelijkheid om gebruikte cloud-omgeving te besturen en niet om evaluaties uit te voeren.

3.2 Beknopt overzicht van cloud resource allocatieschema's

Globale planning van gevirtualiseerde resources beschrijft het systeemwijde perspectief van de allocatie van fysieke en virtuele resources in een cloud omgeving. De meeste van deze methoden

zijn gecentraliseerd zodat ze volledige controle hebben over de allocatie van een verkregen set van resources. De hiervoor gebruikte technieken worden onderverdeeld in 4 groepen namelijk de initiële plaatsing van de VM's, dynamische plaatsing van VM's, VM-plaatsing rekening houdend met de netwerkbronnen en technieken om het energieverbuik te minderen.

De initiële plaatsing van VM's op fysieke machine's is gerelateerd aan het *vector bin packing* probleem, wat NP-hard is waardoor enkel heuristische methoden in aanmerking komen [**?**]. Enkele voorbeelden hiervan zijn de *First Fit Decreasing (FFD)* en *Best Fit Decreasing* heuristieken die gebaseerd zijn op gulzige algoritmen [**?**] of het *Reordering Grouping Algorithm (RGGA)* gebaseerd op genetische algoritmen [**?**]. Daarnaast zijn er ook algoritmen om dynamisch een pool van resources te alloceren aan een set van concurrerende VM's en algoritmen die rekening houden met beperkingen opgelegd door de cloud-gebruikers.
Live migration [**?**] of dynamische plaatsing van virtuele machines is een zeer handige techniek waarbij een draaiende VM wordt gepauzeerd, geserialiseerd en verplaatst naar een andere fysieke machine. Gebruikte heuristieken hiervoor zijn bijvoorbeeld de *first-fit* heuristiek [**?**] dat de verschillende VM's dynamisch hermapt op de fysieke machine's met als resultaat een minimaal gebruik van fysieke machines, of *King-fisher* [**?**], een set van technieken voor VM-schaling, replicatie en migratie waarbij de problemen worden geformuleerd als ILP-model. Een ander voorbeeld is het live migration proces voorstellen als een roman zodat het probleem herleid wordt tot een *Stable Marriage* probleem [**?**].

Er zijn ook algoritmen die rekening houden met de netwerktopologie vooraleer een VM wordt geplaatst op een fysieke machine. Een voorbeeld hiervan zijn heuristieken die VM's die data-intensief met elkaar moeten communiceren dicht bij elkaar trachten te plaatsen.

Technieken om het energieverbuik te minderen zijn gebaseerd op *right-sizing* van datacenters.[1] [**?**] Nieuwere methoden maken een combinatie van deze technieken om het energieverbuik te minderen samen met de mogelijkheid tot het aanpassen van de processorsnelheid van fysieke nodes met behulp van *Dynamic Voltage Scaling* (DVS). Dit zorgt ervoor dat de kloksnelheid van een processor kan dalen (minder stroomverbruik) en een processor kan werken in een omgeving met hogere temperaturen (minder koeling) wat in beide gevallen zorgt voor een lagere energieconsumptie.

Naast de globale planning van gevirtualiseerde resources moet er ook lokale planning van de gevirtualiseerde resources gebeuren. Lokaal resource management bepaalt hoe de fysieke resources worden gedeeld tussen de gevirtualiseerde resources die erop draaien. Recente onderzoeken proberen hiervoor een volledige automatische oplossing te vinden die constant de VM's monitort waarbij dynamisch extra resources voorzien worden, rekening houdend met de afgesloten SLA's. Technieken die hiervoor gebruikt worden zijn onder andere *fuzzy-logic* [**?**] en *reinforcement*

[1] Het dynamisch herschalen van actieve fysieke nodes bij een veranderende werklast.

learning [**?**]. Een voorbeeld van een systeem dat het delen van lokale fysieke resources tussen virtuele resources vergemakkelijkt is *DeepDive* [**?**], een systeem ontwikkeld om interferentie tussen verschillende VM's te identificeren en te verzachten.

Tot slot zijn er nog tal van andere technieken om resources te alloceren rekening houdend met bijvoorbeeld het profiel van de vraag, de schaalbaarheid van de applicatie, etc. maar hier wordt niet dieper op ingegaan.

In Tabel 3.1 wordt een overzicht weergegeven van een aantal ontwikkelde resource allocatieschema's tussen 2012 en 2015. De verschillende kolommen beschrijven respectievelijk de naam van het algoritme of de naam van de ontwikkelaar, het jaartal van de publicatie, het type waarop het schema is gebaseerd, of het een algoritme, framework of protocol is, wat de invoerparameters zijn, wat de uitvoerparameters zijn en op welke manier het getest is.

Tabel 3.1: Overzicht resource-allocatieschema's
A=Algoritme, P=Protocol, F=Framework S=Simulator, C=Cloud, ILP=Integer Linair Programming, GH = Greedy Heuristic, SBP=Stochastic Bin Packing, MINLP=Particle swarm, RR=Round-Robin, SA=Simulated Annealing, SMT=Satisfiabilty Module Theory, FFD=First Fit Decreasing, MI(N)=Mixed Integer (Non-), SPLE=Software Product Line Engineering, L=Lijst, G=Graaf, B=Boom, FN=Fysieke node, Mig=Migraties, (?)=Niet vermeld

Naam	Jaar	Type	A | F | P	Invoer	Uitvoer	Getest
Alicherry et al. [**?**]	2012	k-sneden	A	G	par{G}	S
MCRVMP [**?**]	2012	ILP & GH	A	B{netwerk}	VM-plaatsing	C
Breitgand et al. [**?**]	2012	SBP	A	L{VM}	L{VM}/Bin	S
Snooze [**?**]	2012	Framework	F	-	-	C
Sequence planning [**?**]	2012	Heuristiek	A	Net + S{Mig}	L{VM/Mig}	S
Giurgu et al. [**?**]	2012	Beam seach	A	-	VNI-plaatsing	S
Seagull [**?**]	2012	Framework	F	-	-	C
v-Bundle [**?**]	2012	Novel	A	B{Node}	VM-plaatsing	S + C
VMPR [**?**]	2012	Markov-ketens	A	L{Jobs}	Job-afhandeling	S (?)
TROPIC [**?**]	2012	Framework	F	-	-	C
Konstanteli et al. [**?**]	2012	MINLP	A	-	Gealloceerde resources	C
CloudMap [**?**]	2012	RR	A	Resource-verbruik	Servercluster	C
VirtualKnotter [**?**]	2012	SA	A	Verkeer	VM-plaatsing	S
P* [**?**]	2012	Gossip protocol	P	-	-	S
Scattered [**?**]	2012	Black- en Gray-box	A	L{hosts}	VM-plaatsing	C
VMM-Planner [**?**]	2013	SMT	A	VM-plaatsing + doel	L{Mig}	C
Shi et al. [**?**]	2013	ILP	A	L{FN}	VM-plaatsing	C
Shi et al. [**?**]	2013	FFD	A	L{FN}	VM-plaatsing	C
Omega [**?**]	2013	Large scale	F	-	-	S
PACMan [**?**]	2013	-	A	VM interferentie	VM-plaatsing	C
Max-BRU [**?**]	2014	-	A	Datacenter	VM-allocatie	C
RT-OpenStack [**?**]	2015	-	A	Budget VCPU's	CPU-resources	C
J.Bi et al. [**?**]	2015	SA, MINLP	A	Datacenter	VM-allocatie	S
F. Fakhfakh et al. [**?**]	2015	MILP	A	L{Taken, deadl.}	VM-allocatie	S
L. Li et al. [**?**]	2015	Framework	F	-	-	(C)
A. Ruiz-Alvarez et al. [**?**]	2015	IPL	A	Max. duur	Optimale kost	C
Dohko [**?**]	2015	SPLE	A	Vereisten	Resource selectie	S
A.aral et al. [**?**]	2015	LAD	A	Gewenste topologie	Topologie	S
K. Metwally et al. [**?**]	2015	MILP	A	Datacenter	VRP + resource allocatie	C

Bibliografie

[1] D. standaard, "Base overschreed stralingsnormen na aanslagen," 2016. [Online]. Available: https://www.standaard.be/cnt/dmf20160324_02200340

[2] L. Hardell and C. Sage, "Biological effects from electromagnetic field exposure and public exposure standards," *Biomedicine and Pharmacotherapy*, vol. 62, no. 2, pp. 104 – 109, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0753332207002909