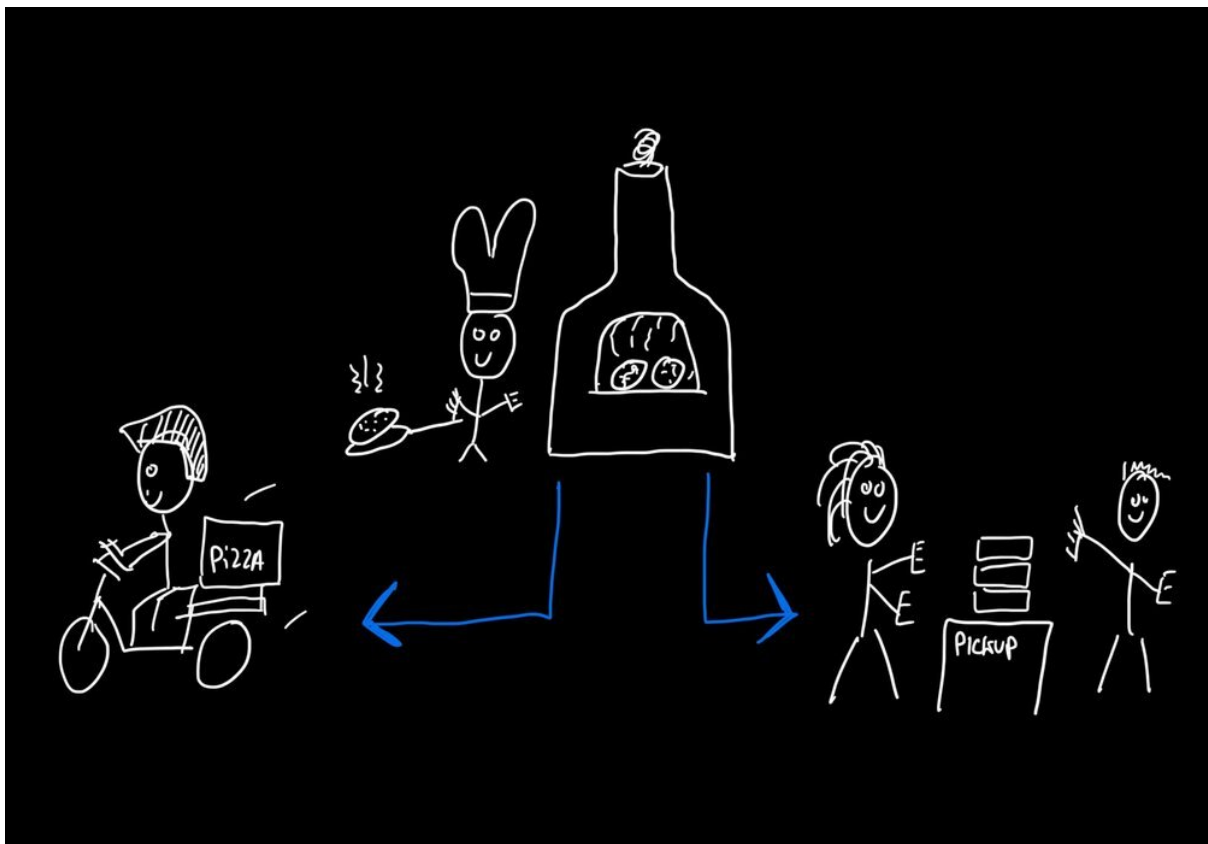


Het ontwerpen van een multidimensionale capaciteitswachtrij

Beheren van keuken-, bezorg- en afhaalcapaciteit

Willem L. Middelkoop

2 dec. 2020



Deze maand moest ik extra dimensies toevoegen aan een capaciteitswachtrijmechanisme. De voedselbestel-app die ik heb gemaakt, moest de capaciteit kunnen beperken op basis van het aantal bestellingen, de inhoud van individuele bestellingen en het verzendtype (afhalen/bezorgen). Lees verder om te ontdekken hoe ik een Lambda Architectuur hiervoor heb gebruikt.

Capaciteit beperken

Hoewel je je dit misschien niet realiseert, is een van de krachtigste functies van de [food ordering app that I created](#) juist het kunnen beperken van het aantal bestellingen. Dit klinkt contra-intuïtief, maar door het aantal bestellingen dat geplaatst kan worden te beperken,

kunnen mijn klanten (de restaurants en winkeleigenaren die mijn app gebruiken) ervoor zorgen dat bestellingen op een goede manier worden voorbereid en bezorgd.

Drie pizza's complexiteit

Stel, je wilt drie pizza's bestellen. De food-app moet dan weten:

- Wil je ze **afhalen** of laten **bezorgen**?
- **Hoe laat?** Wanneer moeten ze de oven in?
- Is er **genoeg ruimte beschikbaar** voor *drie pizza's in de oven op dat tijdstip*?
- Als je hebt gekozen voor afhalen: **kunnen we je veilig ontvangen** op dat tijdstip?
- Als je hebt gekozen voor bezorgen: **is er een bezorger beschikbaar** om je pizza's te brengen?
- **Ja op al deze vragen?** Dan **gelieve te betalen** terwijl wij je **pizza's ingepland houden** in de oven en bezorging of afhaling. Betaling mislukt? Toch bedacht? Dan "maken we de planning ongedaan" en wachten we tot iemand anders pizza's bestelt!

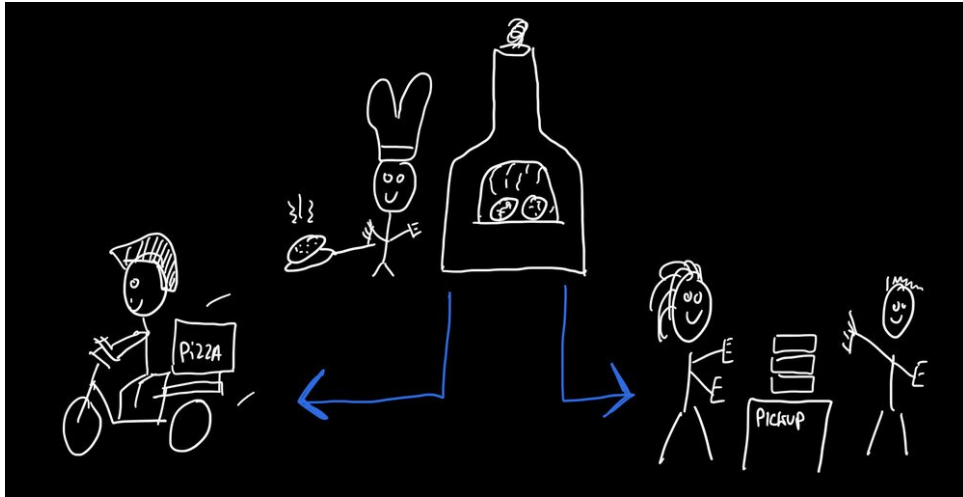


Pizza eten tijdens de ontwikkeling is hoe je "het algoritme leeft"

Drie capaciteitsdimensies

Vooraf rond "etenstijd" bestaat het risico dat te veel mensen tegelijk eten bestellen. Hoewel de app het extra verkeer aan kan dankzij de [highly scalable architecture](#), blijven fysieke beperkingen bestaan, zoals:

- **keukencapaciteit:** het aantal gerechten dat je tegelijk kunt bereiden
- **afhaalcapaciteit:** het aantal mensen dat je tegelijk veilig kunt ontvangen met inachtneming van de COVID-19-beperkingen
- **bezorgcapaciteit:** het aantal bestellingen dat je kunt bezorgen, rekening houdend met verkeer, afstand en routeplanning



Drie capaciteitsbeperkingen: keuken / bezorging / afhalen

Onderscheiden maar onderling afhankelijk

Deze drie verschillende soorten beperkingen zijn onderling afhankelijk. Het is bijvoorbeeld mogelijk dat één grote afhaalbestelling alle keukencapaciteit in beslag neemt, waardoor er ook geen bezorgcapaciteit meer beschikbaar is. Kleinere bestellingen kunnen nog steeds worden geaccepteerd om de keukencapaciteit op te vullen, terwijl er mogelijk niet genoeg capaciteit meer beschikbaar is voor grote bestellingen. Dit betekent dat je een mechanisme nodig hebt dat naar *alle* statistieken tegelijk kijkt om de beschikbaarheid voor nieuwe bestellingen te bepalen, inclusief de inhoud van een niet-geplaatste bestelling (bijv. wat er in je winkelmandje zit).

Verschillen in tijdspanne granulariteit

Het is gebruikelijk om een fysieke beperking te definiëren met behulp van een tijdspanne, bijvoorbeeld "de keuken kan 10 gerechten per 15 minuten produceren". Het wordt een uitdaging wanneer je andere fysieke beperkingen definieert met behulp van verschillende tijdspannes. Terwijl de keuken per 15 minuten kan worden beperkt, kan de bezorgcapaciteit een langere tijdspanne gebruiken (bijv. 3 bezorgingen per 30 minuten). Het mechanisme moet op de een of andere manier met deze verschillen in tijdspanne granulariteit kunnen omgaan.

Vooraf bestellen

Een extra uitdaging is de mogelijkheid voor mensen om hun bestellingen vooraf te plaatsen. Mensen kunnen een tijdslot in de toekomst selecteren en de benodigde capaciteit laten reserveren. Soms zie ik mensen hun bestelling dagen van tevoren plaatsen om er zeker van te zijn dat ze hun eten op een specifieke dag en tijd bezorgd krijgen. Dit betekent dat er (minstens) twee tijdstippen aan elke bestelling zijn gekoppeld: besteltijd en bezorgtijd.

Capaciteit reserveren en vrijgeven

De meeste bedrijfseigenaren vereisen online betalingen voor online bestellingen, dit is om ervoor te zorgen dat alle bestellingen legitiem zijn en er geen capaciteit (of voedsel)

wordt verspild aan no-shows of spookbestellingen. Het [probleem met online betalingen](#) is dat niet alle transacties succesvol zijn. Soms wordt een creditcard- of betaalpasbetaling afgewezen (bijv. vanwege onvoldoende saldo). Soms sluiten mensen gewoon de app voordat ze de bestelling hebben voltooid. Het systeem moet de benodigde capaciteit voor een bepaalde bestelling reserveren, terwijl het mensen in staat stelt de online betaling uit te voeren, en het moet de gereserveerde capaciteit vrijgeven als de bestelling wordt geannuleerd of wanneer de betaling mislukt.

Hoog volume, hoge snelheid

Tot slot moet het mechanisme betrouwbaar en performant zijn. De beschikbaarheid van bestellingen moet binnen enkele milliseconden worden bepaald, terwijl er tegelijkertijd veel verschillende clients verbonden zijn met de food-bestel-app tijdens spitsuren. Wanneer iemand een bestelling plaatst, heeft dit invloed op de beschikbaarheid voor alle andere (potentiële) bestellingen. Niet iedereen heeft een betrouwbare verbinding, omdat sommige mensen de app gebruiken op telefoons met slechte ontvangst of wifi. Het systeem moet goed werken onder wisselende omstandigheden met veel verschillende verbonden personen.

Lambda-architectuur

De [Lambda Architecture](#) is een manier om grote hoeveelheden data te verwerken door gebruik te maken van zowel stream processing als batch processing methoden. Een lambda-architectuur is afhankelijk van een datamodel met een append-only, immutable datasource, vaak bestaande uit events met tijdstempels.

Onveranderlijke data

Dit is een zeer belangrijke basis voor het mechanisme: alle data heeft een tijdstempel en verandert *nooit*. In plaats van bestaande records te muteren, worden er gewoon nieuwe records toegevoegd om de bestaande records te overschrijven. Dit maakt hoge concurrentie en gedistribueerde algoritmen mogelijk, omdat synchronisatie eenvoudiger wordt omdat men alleen hoeft te zoeken naar nieuwe data (in plaats van zowel nieuwe als bijgewerkte!).

Realtime / speed layer

De stream processing, of realtime/speed layer, in een lambda-architectuur is ontworpen om supersnelle antwoorden te geven op realtime databehoeften. Het onderhoudt views en statistieken op basis van realtime verwerking van events.

Precomputation layer

De batch processing, of precomputation layer, biedt een complete en nauwkeurige basis van alle data in het systeem. Omdat het verwerken van grote hoeveelheden data tijd kan kosten, is de batch processing niet snel genoeg om alles direct te doen. Daarom werken de twee lagen samen in een lambda-architectuur.

Implementatie

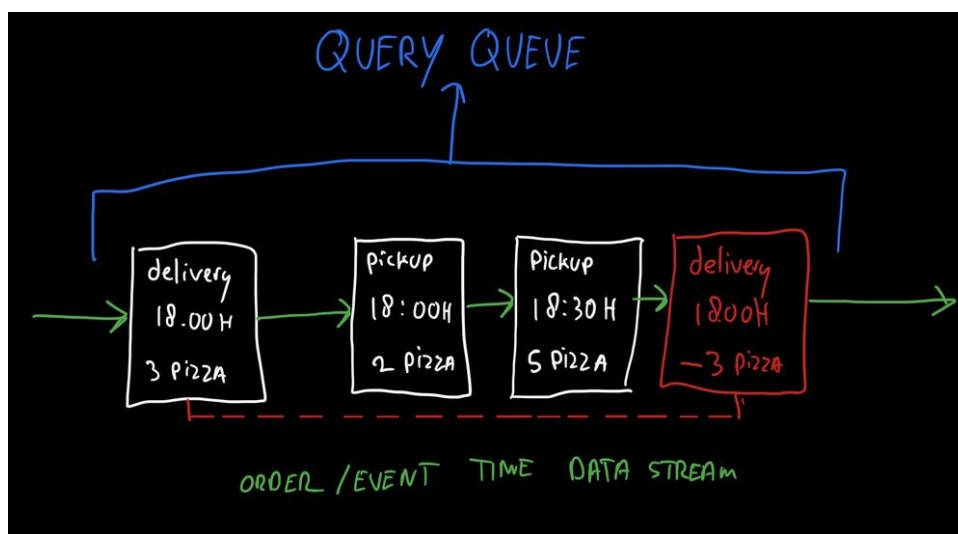
De backend servers die food-bestellingen van de app verwerken werken met een datastroom van events met tijdstempels. In wezen: elke bestelling is een event in het systeem. Elke bestelling wordt opgeslagen met de bestel-datum/tijd, verzend-datum/tijd (geplande bezorg- of afhaaltijd), klantgegevens en bestel-inhoud (aantal gerechten, enz.).

Capaciteit reserveren en vrijgeven

Als een betaling mislukt of als een bestelling wordt geannuleerd, wordt een "negatieve" bestelling toegevoegd aan de datastroom. Het neutraliseert elk effect op capaciteitsbeperkingen door de negatieve statistieken. Bijvoorbeeld: om een bestelling van drie pizza's te annuleren, wordt een negatieve bestelling van "min drie pizza's" aangemaakt in de datastroom met dezelfde verzendtijdstempel. Dit is een elegante manier om ervoor te zorgen dat capaciteit zowel gereserveerd als vrijgegeven wordt.

Multidimensionale realtime statistieken

In plaats van capaciteit te abstraheren naar tellers, gebruikt de backend server de *echte* stream data om queries voor realtime statistieken te beantwoorden. Dit stelt het systeem in staat om om te gaan met verschillen in tijdspanne granulariteit.



Een event stream van bestellingen is de kern van het mechanisme, queries worden beantwoord door naar deze stream te kijken: hoeveel pizza's zijn er klaar om 18:00 uur?

De eerste keer dat een query wordt ontvangen voor een bepaalde statistiek, laadt het systeem historische data via een batch mechanisme om het model te initialiseren dat het vervolgens realtime kan bijwerken. Deze statistiek wordt bijgehouden door te kijken naar alle nieuwe bestellingen die het systeem binnenkomen zolang er queries voor een bepaalde statistiek komen. Zodra de data niet meer wordt opgevraagd, wordt het model verwijderd. Als het opnieuw nodig is, herhaalt het proces zich.

Het mooie van deze aanpak is dat de meeste queries binnen enkele milliseconden worden beantwoord omdat ze direct uit het RAM-geheugen worden gelezen zonder database- of schijfinteractie. Dit maakt ultrahoge prestaties mogelijk op een "need to have"-basis. Dit werkt goed met de food-bestel-app, omdat niet *alle* tijdspannes even populair zijn.

De food-bestel-app vraagt herhaaldelijk een centrale capaciteitsserver die de wachtrijstatistieken bijhoudt. Vervolgens filtert het beschikbare tijdsloten door te kijken naar de daadwerkelijke items die een persoon selecteert voor zijn of haar bestelling. Het selecteren van een (waarschijnlijk) beschikbaar tijdslot gebeurt daarom client-side, waardoor de belasting van de backend wordt verminderd. Zodra een bestelling is geplaatst, wordt de capaciteit door de backend gereserveerd door de centrale wachtrijstatistieken bij te werken, waardoor anderen worden geblokkeerd om dezelfde plek in de wachtrij te claimen.

Conclusie

Wachtrijmechanismen kunnen een uitdaging zijn wanneer je te maken hebt met meerdere statistieken die de snelheid bepalen. Je moet echt de tijd nemen om het goed te modelleren, denk aan het soort data dat je *echt* nodig hebt om wachtrijgerelateerde controles uit te voeren.

Nu weet je dat het bestellen van drie pizza's niet zo eenvoudig is! Garanderen dat ze warm en op tijd gereserveerd kunnen worden is een grote technische uitdaging - naast het zijn van een goede kok. Gelukkig voor jou dat *ik* me niet op dat laatste richt, ha!