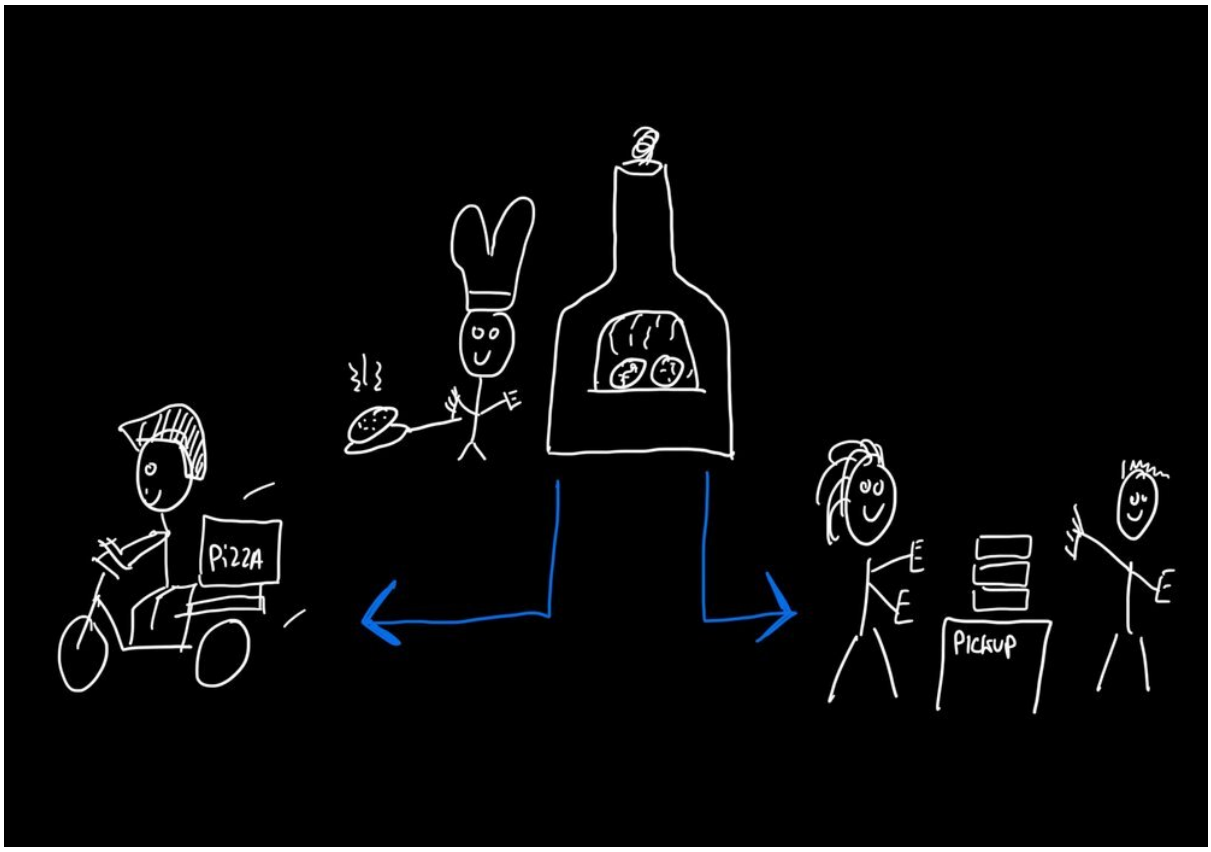
Entwurf einer mehrdimensionalen Kapazitätswarteschlange

Management der Küchen-, Liefer- und Abholkapazität

Willem L. Middelkoop
Dec. 2, 2020



Diesen Monat musste ich zusätzliche Dimensionen zu einem Kapazitätswarteschlangen-Mechanismus hinzufügen. Die von mir entwickelte App für Essensbestellungen musste in der Lage sein, die Kapazität basierend auf der Anzahl der Bestellungen, dem Inhalt einzelner Bestellungen und der Versandart (Abholung/Lieferung) zu beschränken. Lesen Sie weiter, um herauszufinden, wie ich dafür eine Lambda-Architektur verwendet habe.

Kapazitätsbeschränkung

Obwohl Sie es vielleicht nicht bemerken, ist eine der mächtigsten Funktionen der food ordering app that I created die Möglichkeit, die Anzahl der Bestellungen zu begrenzen. Das klingt widersprüchlich, aber indem sie die Anzahl der Bestellungen begrenzen, können

meine Kunden (die Restaurants und Ladenbesitzer, die meine App nutzen) sicherstellen, dass die Bestellungen gut zubereitet und geliefert werden.

Drei-Pizza-Komplexität

Angenommen, Sie möchten drei Pizzen bestellen. Die Food-App muss dann Folgendes wissen:

- Möchten Sie die Pizzen **abholen** oder **liefern lassen**?
- Zu **welcher Uhrzeit?** Wann müssen sie in den Ofen?
- Ist *zu diesem Zeitpunkt* **genügend Platz für** *drei* Pizzen **im Ofen** vorhanden?
- Wenn Sie Abholung gewählt haben: **Können wir Sie zu diesem Zeitpunkt sicher empfangen**?
- Wenn Sie Lieferung gewählt haben: **Steht ein Lieferbote zur Verfügung**, um Ihnen Ihre Pizzen zu bringen?
- **Auf alles Ja?** Dann **bitte bezahlen**, während wir **Ihre Pizzen im Ofen und für die Lieferung oder Abholung einplanen**. Zahlung fehlgeschlagen? Zweifel? Dann "machen wir" die gesamte Planung rückgängig und warten darauf, dass jemand anderes Pizzen bestellt!

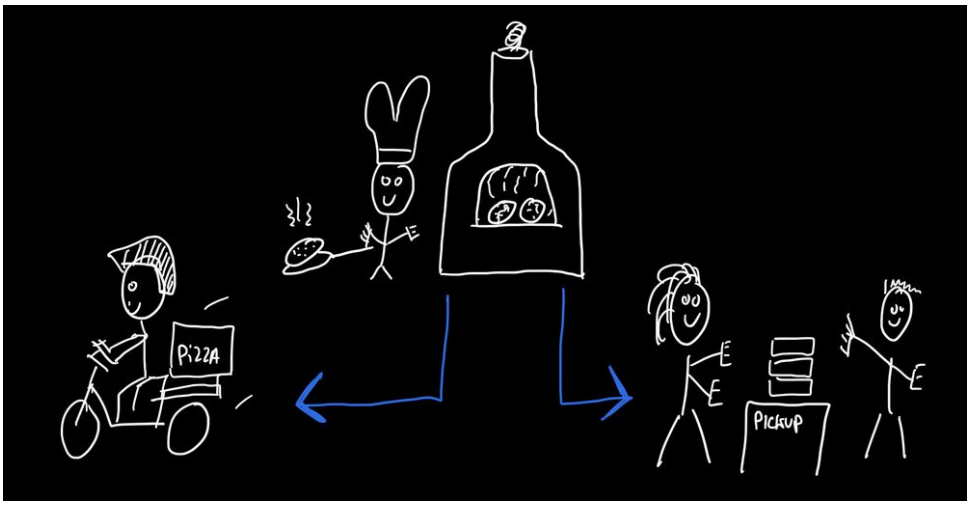
Pizza essen während der Entwicklung bedeutet, den Algorithmus zu "leben".

Drei Kapazitätsdimensionen

Besonders zur "Dinner-Zeit" besteht die Gefahr, dass zu viele Leute gleichzeitig Essen bestellen. Während die App den zusätzlichen Traffic dank ihrer highly scalable architecture bewältigen kann, bleiben physische Einschränkungen bestehen, wie z. B.:

- **Küchenkapazität**: die Anzahl der Gerichte, die Sie gleichzeitig zubereiten können

- **Abholkapazität**: die Anzahl der Personen, die Sie unter Berücksichtigung der COVID-19-Beschränkungen gleichzeitig sicher empfangen können
- **Lieferkapazität**: die Anzahl der Bestellungen, die Sie unter Berücksichtigung von Verkehr, Entfernung und Routenführung ausliefern können



Drei Kapazitätsbeschränkungen: Küche / Lieferung / Abholung

Unterschiedlich, aber voneinander abhängig

Diese drei verschiedenen Arten von Einschränkungen sind voneinander abhängig. Es ist zum Beispiel möglich, dass eine große Abholbestellung die gesamte Küchenkapazität beansprucht, was auch zu einer Nichtverfügbarkeit für Lieferungen führt. Kleinere Bestellungen könnten noch angenommen werden, um die Küchenkapazität auszufüllen, während für große Bestellungen möglicherweise nicht mehr genügend Kapazität verfügbar ist. Das bedeutet, dass Sie einen Mechanismus benötigen, der *alle* Kennzahlen kombiniert betrachtet, um die Verfügbarkeit für neue Bestellungen zu bestimmen, einschließlich des Inhalts einer nicht aufgegebenen Bestellung (z. B. was sich in Ihrem Warenkorb befindet).

Unterschiede in der Granularität der Zeitspanne

Es ist üblich, eine physische Beschränkung mithilfe einer Zeitspanne zu definieren, z. B. *"die Küche kann 10 Gerichte pro 15 Minuten zubereiten"*. Es wird schwierig, wenn Sie andere physische Beschränkungen mit unterschiedlichen Zeitspannen definieren. Während die Küche auf 15 Minuten begrenzt sein könnte, könnte die Lieferkapazität eine längere Zeitspanne verwenden (z. B. 3 Lieferungen pro 30 Minuten). Der Mechanismus sollte in der Lage sein, mit diesen Unterschieden in der Granularität der Zeitspanne umzugehen.

Vorbestellung

Die Möglichkeit, Bestellungen im Voraus aufzugeben, stellt eine zusätzliche Herausforderung dar. Personen können ein Zeitfenster in der Zukunft auswählen und die erforderliche Kapazität reservieren lassen. Manchmal sehe ich Leute, die ihre Bestellung Tage im Voraus aufgeben, um sicherzustellen, dass ihr Essen an einem bestimmten Tag und zu einer bestimmten Uhrzeit geliefert wird. Das bedeutet, dass mit jeder Bestellung (mindestens) zwei Zeiten verbunden sind: Bestellzeitpunkt und Lieferzeitpunkt.

Reservierung und Freigabe von Kapazität

Die meisten Geschäftsinhaber verlangen Online-Zahlungen für Online-Bestellungen, um sicherzustellen, dass alle Bestellungen legitim sind und keine Kapazität (oder Lebensmittel) durch Nichterscheinen oder Geisterbestellungen verschwendet wird. Das problem with online payments ist, dass nicht alle Transaktionen erfolgreich sind. Manchmal wird eine Kredit- oder Debitkartenzahlung abgelehnt (z. B. wegen unzureichender Deckung). Manchmal schließen die Leute die App einfach, bevor sie die Bestellung abschließen. Das System sollte die erforderliche Kapazität für eine bestimmte Bestellung reservieren, während es den Leuten ermöglicht, die Online-Zahlung durchzuführen, und es sollte die reservierte Kapazität freigeben, falls die Bestellung abgebrochen wird oder die Zahlung fehlschlägt.

Hohes Volumen, hohe Geschwindigkeit

Zu guter Letzt sollte der Mechanismus zuverlässig und performant sein. Die Verfügbarkeit von Bestellungen sollte innerhalb von Millisekunden ermittelt werden, während gleichzeitig eine hohe Parallelität bewältigt wird, da während der Stoßzeiten viele verschiedene Clients mit der Food-Ordering-App verbunden sind. Wenn jemand eine Bestellung aufgibt, wirkt sich dies auf die Verfügbarkeit für alle anderen (potenziellen) Bestellungen aus. Nicht jeder hat eine zuverlässige Verbindung, da manche Leute die App auf Handys mit schlechtem Empfang oder WLAN verwenden. Das System muss unter verschiedenen Bedingungen mit vielen verschiedenen verbundenen Personen gut funktionieren.

Lambda-Architektur

Die Lambda Architecture ist eine Möglichkeit, große Datenmengen zu verarbeiten, indem sowohl Stream- als auch Batch-Verarbeitungsmethoden verwendet werden. Eine Lambda-Architektur basiert auf einem Datenmodell mit einer nur anfügbaren, unveränderlichen Datenquelle, die oft aus zeitgestempelten Ereignissen besteht.

Unveränderliche Daten

Dies ist eine sehr wichtige Grundlage für den Mechanismus: Alle Daten sind mit einem Zeitstempel versehen und ändern sich *nie*. Anstatt bestehende Datensätze zu verändern, werden einfach neue Datensätze hinzugefügt, um die bestehenden Datensätze zu überschreiben. Dies ermöglicht eine hohe Parallelität und verteilte Algorithmen, da die Synchronisierung einfacher wird, da man nur nach neuen Daten suchen muss (statt nach neuen und aktualisierten!).

Echtzeit-/Geschwindigkeits-Schicht

Die Stream-Verarbeitung oder Echtzeit-/Geschwindigkeits-Schicht in einer Lambda-Architektur ist darauf ausgelegt, superschnelle Antworten auf Echtzeit-Datenbedürfnisse zu liefern. Sie verwaltet Ansichten und Metriken basierend auf der Echtzeitverarbeitung von Ereignissen.

Vorberechnungsschicht

Die Batch-Verarbeitung oder Vorberechnungsschicht bietet eine vollständige und genaue Grundlage für alle Daten im System. Da die Verarbeitung großer Datenmengen zeitaufwendig sein kann, ist die Batch-Verarbeitung nicht schnell genug, um alles sofort zu erledigen. Deshalb arbeiten die beiden Schichten in einer Lambda-Architektur zusammen.

Implementierung

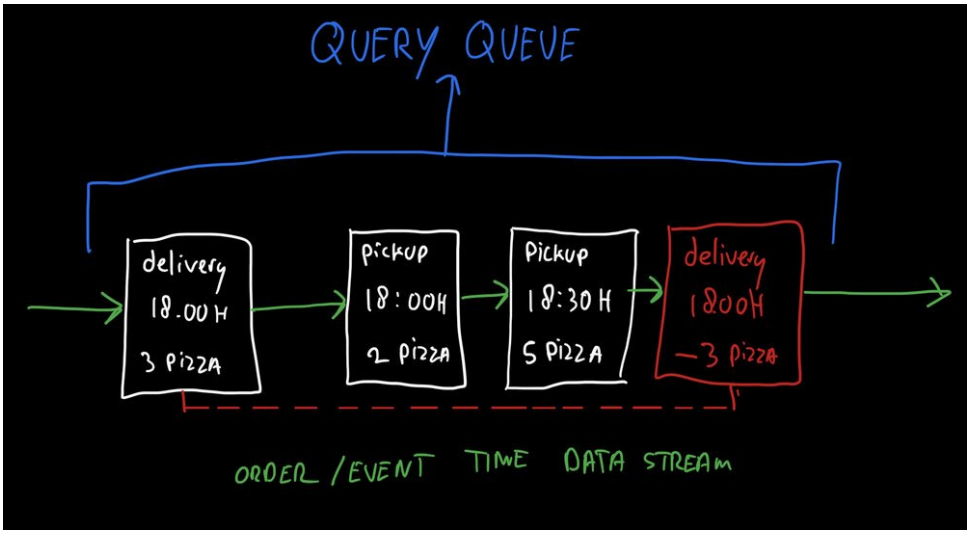
Die Backend-Server, die Essensbestellungen von der App verarbeiten, arbeiten mit einem Datenstrom von zeitgestempelten Ereignissen. Im Wesentlichen: Jede Bestellung ist ein Ereignis im System. Jede Bestellung wird mit Bestelldatum/-zeit, Versanddatum/-zeit (geplante Liefer- oder Abholzeit), Kundendaten und Bestellinhalt (Anzahl der Gerichte usw.) gespeichert.

Reservierung und Freigabe von Kapazität

Wenn eine Zahlung fehlschlägt oder eine Bestellung abgebrochen wird, wird dem Datenstrom eine "negative" Bestellung hinzugefügt. Sie neutralisiert jegliche Auswirkungen auf Kapazitätsbeschränkungen durch ihre negativen Kennzahlen. Beispiel: Um eine Bestellung von drei Pizzen zu stornieren, wird im Datenstrom eine negative Bestellung von "minus drei Pizzen" mit demselben Versandzeitstempel erstellt. Dies ist eine elegante Möglichkeit, um sicherzustellen, dass Kapazität sowohl reserviert als auch freigegeben wird.

Mehrdimensionale Echtzeit-Metriken

Anstatt die Kapazität in Zähler zu abstrahieren, verwendet der Backend-Server die *echten* Stream-Daten, um Abfragen nach Echtzeit-Metriken zu beantworten. Dies ermöglicht es dem System, mit Unterschieden in der Granularität der Zeitspanne umzugehen.



Ein Ereignisstrom von Bestellungen ist das Herzstück des Mechanismus. Abfragen werden durch Betrachtung dieses Stroms beantwortet: Wie viele Pizzen sind um 18:00 Uhr fertig?

Wenn zum ersten Mal eine Abfrage für eine bestimmte Metrik eingeht, lädt das System historische Daten über einen Batch-Mechanismus, um das Modell zu initialisieren, das es dann in Echtzeit aktualisieren kann. Diese Metrik wird durch die Betrachtung aller neuen Bestellungen, die in das System gelangen, so lange aktualisiert, wie Abfragen für eine bestimmte Metrik eingehen. Sobald die Daten nicht mehr benötigt werden, wird das Modell entladen. Wenn es wieder benötigt wird, wiederholt sich der Vorgang.

Das Tolle an diesem Ansatz ist, dass die meisten Abfragen innerhalb von Millisekunden beantwortet werden, da sie direkt aus dem RAM-Speicher gelesen werden, ohne dass eine Datenbank- oder Festplatteninteraktion erforderlich ist. Dies ermöglicht eine extrem hohe Leistung auf einer "Need-to-have"-Basis. Dies funktioniert gut mit der Food-Ordering-App, da nicht *alle* Zeitspannen gleich beliebt sind.

Die Food-Ordering-App fragt wiederholt einen zentralen Kapazitätsserver ab, der die Warteschlangenmetriken verwaltet. Sie filtert dann verfügbare Zeitfenster heraus, indem sie die tatsächlichen Dinge betrachtet, die eine Person für ihre Bestellung auswählt. Die Auswahl eines (wahrscheinlich) verfügbaren Zeitfensters erfolgt daher clientseitig, wodurch die Last auf dem Backend reduziert wird. Sobald eine Bestellung aufgegeben wird, wird die Kapazität vom Backend reserviert, indem die zentralen Warteschlangenmetriken aktualisiert werden, wodurch andere Personen daran gehindert werden, denselben Platz in der Warteschlange zu beanspruchen.

Fazit

Warteschlangenmechanismen können eine Herausforderung sein, wenn Sie es mit mehreren Metriken zu tun haben, die die Geschwindigkeit steuern. Sie sollten sich wirklich die Zeit nehmen, es richtig zu modellieren und über die Art von Daten nachzudenken, die Sie *wirklich* benötigen, um warteschlangenbezogene Prüfungen durchzuführen.

Jetzt wissen Sie, dass die Bestellung von drei Pizzen nicht ganz so einfach ist! Zu garantieren, dass sie heiß und pünktlich serviert werden können, ist eine große technische Herausforderung - zusätzlich dazu, ein guter Koch zu sein. Zum Glück für Sie konzentriere *ich* mich nicht auf Letzteres, ha!