Verteilte Systeme

May 12, 2015

1 ØMQ

1.1 Verwendung

1. Context: Es muss für ØMQ ein Kontext erstellt werden (das Umfeld)

2. Socket: Es muss ein Socket in diesem Kontext erstellt werden

3. Sockettyp: Was für eine Art Socket wird benötigt (SUB/PUB etc.)

4. Binding: Socket an eine Adresse und einen Port binden (Remote, oder Lokale für listening)

5.1 Receive Receive (blockierend), besonderer Messagetyp

5.2 Send Send, gegebener Messagetyp ist zu füllen

ØMQ Buffert alle Nachrichten bis zu einem gewissen Grad. So werden auch langsamere Reader oder später registrierte Subscriber bedient, jedoch ohne versicherung, dass sie alle Nachrichten empfangen.

1.2 Patterns

Request-Reply Synchrone Request-Reply Kommunikation

Push-Pull Sender erwartet keine Rückantwort (Push), Empfänger benutzen Pull, um Nachricht zu empfangen. Kann benutzt werden, um Lasten zu verteilen, weil eine Nachricht nur höchstens eimal empfangen werden kann.

Pub/Sub Ein Publisher sendet Nachrichten mit bestimmten Tags, alle Subscriber empfangen alle Nachrichten, die den vorgegebenen Tags des Subscribers entsprechen.

1.3 Transportvarianten

ipc: Interprozess, FIFO etc.

inproc: Socketkommunikation innerhalb eines Prozesses.

mMulticast: ØMQ unterstützt Multicast, wenn das Netzwerk es unterstützt.

1.4 PlayØMQ, Broker

Sources generieren Zufallszahlen und geben sie an den Broker.
Worker nehmen Zahlen aus dem Broker und prüfen auf Primzahl.
Sinks hören auf Nummern/Primzahlen. Broker: Sendet einkommende Zahlen per Push-Socket an Worker, sendet Zahlen und Primzahlen per PUB-Socket Source: Request Reply auf Broker.
Worker: Push/Pull → Parallelisierung der Berechnung
Sink: Subscriber, die auf den Broker "Subscriben", mit dem entsprechenden Tag (number/primzahl)

1.4.1 Nachrichten

Senden und Empfangen von Daten muss besonders behandelt werden, da Nachrichten erstellt werden müssen. Nachrichten bestehen aus Tag (identifier Char) und Wert (long)

1.4.2 Source

Benötigt URL von Broker
Erstelle REQ (Request) Socket, der mit dem Broker verbindet.
Source generiert Zahlen, sendet sie an Broker und wartet, bis dieser den Empfang bestätigt hat.

1.4.3 Broker

Broker hat Sockets für REP (Reply) für Source, PUB (Publisher) für Sinks, PUSH für Worker.
Im Beispiel ist Endpoint eine Methode, die die Befehle zum Erstellen eines einfachen Sockets zusammenfasst.
Broker empfängt Zahl von Source, bestätigt dem Empfang, publiziert die Nummer, sendet sie an die Worker, wartet auf die Antwort und publiziert u.U. die Primzahl.

1.4.4 Worker

Verbindet per PULL und REQ (Request) Socket auf den Broker.
Empfängt vom Broker per PULL Zahlen, prüft diese und sendet sie u.U. per REQ an Broker. Wenn die Primzahl gesendet wird, muss der Worker wieder auf die Antwort (REPly) des Broker warten.

1.4.5 Sink

Erstellt SUB (Subscriber) Socket, über den er vom Broker Daten empfängt.
Beim Erstellen wird festgelegt, auf welche Tags der Subscriber hört.

2 Verteilung von Last

Kreative Lastverteilung: Verteilte Programmierung.
Mechanische Lastverteilung: Verteilung auf Rechner, gleiche Auslastung der verwendeten Rechner

Durch verteilte Rechner ist die "aktuelle Last" verzögert, um die Latenz zwischen A und B. A schickt also Last an B, auch wenn B inzwischen eine hohe Last erreicht hat.
Lösung: A kann anhand der verteilten Last merken, wie viel Last ungefähr an B vergeben wurde (wenn B Last nicht weitergibt).

Paketweitergabe:

- Wenn der Code lokal vorhanden ist, muss er kopiert werden (achte auf identische Pfadnamen etc.)
  Bei Netzwerklaufwerken geht Zeit verloren, da das Netzwerklaufwerk limitierende Bandbreite hat.
- Verteiler schickt Programmcode mit (Große Datenmenge, Achtung Viren)

2.1 Architektur

Erstellen einer Lastmetrik: Welche Last hat ein Rechner, welche Last erzeugt ein Paket?
Verteilung der Last der beteiligten Rechner: Entscheidungsgundlage für Verteiler.
Verteilung des Lastpaketes auf gewählte Rechner: Welche Informationen müssen übertragen werden.

Lastmetriken

- Prozessorauslastung (Anzahl Prozesse, CPU-Zeit)
- Speicherauslastung (Bedarf etc.)
- Kommunikationslast (IO, Netzwerk)

Pull

Zieht Lastwerten an. (Unterforderte Rechner suchen Last)
Lastwert wird bei der Erzeugung eines Paketes ermittelt.
Varianten:

n allen Rechnern: Verteiler sendet Broadcast und fragt nach Last, alle Rechner antworten mit eignener Last.
Verbesserung: Antowrt wird verzögert, je nach eigener Last. Damit antwortet der am wenigsten ausgelastete als erstes.

Feste Teilmenge: Erreriche bestimmte Worker

fällige Teilmenge: Frage zufällige Worker an

Push

Lastwerte verteilen. (Überforderte Rechner verteilen Last)

2.2 Verteilungsverfahren

Last sollte so verteilt werden, dass Rechen- und Speicherlast gleich verteilt ist und die Kommunikation möglichst lokal stattfindet.

atisches Verfahren Optimale Verteilung vor dem Start der Anwendung ermitteln.
Alle Bestandteile der Lastmetrik sind für alle Prozesse bekannt.
Findet häufig Anwendung, Last ist vorher meistens bekannt (Meterologie etc.)

misches Verfahren Ausführungsort wird für jeden Prozess ermittelt.

Mit Migration Migration heißt, die Adressräume und die Prozessstati zu übertragen. Es muss gewartet werden, bis alle Threads keine OS-Ressourcen mehr benutzen.
Problem: Adressräume sind u.U. sehr groß.
Lösung: Copy-on-Reference: Kopiere Adressen, wenn sie benötigt werden. (Reduziert Speicherlast auf Quellknoten nicht).

Ohne Migration Hoher Aufwand die Lastverteilung zu bestimmen.

Durchgesetzt hat sich Initial Placement
Bei Paketentstehung Zielknoten bestimmen, im Extremfall zufällige Wahl.
Ist durchaus das schnellste Verfahren.

2.3 Zeit bei Verteilung

2 Möglichkeiten:

- 1. Synchronisieren der Uhren der Systeme
- 2. Erstellung eines neuen Zeitmaßes

2.3.1 Synchronisation der Uhren

Es wird angenommen, dass jede Uhr ungenau ist und eine lineare Abweichung besitzen.
Ziel: Zeit innerhlab eines Rahmen halten und Abweichungen durch Synchronisation verhindern.
Ein System hat eine genaue Zeit, die an andere Systeme verteilt wird:
Die nachgestellten Uhren hinken immer ein wenig hinterher.
DCF77 erste Zeitsender in Offenbach, senden Zeitsignal über Langwelle. Pulst jede Sekunde, Zusatzdaten auf das Sekundensignal moduliert.
Vergleich nach F. Christian:
Client muss herausfinden, was die Laufzeit des Singals ist. Er speichert seine Zeit, wenn das Signal losgeschickt wird und wenn die Antwort empfangen wird. Der Client berechnet die Mitte aus Empfangszeit-Sendezeit, dieser Zeitpunkt wird auf die Zeit des Signals (Servers) gesetzt.
Lokale Uhr kann schneller/langsamer sein. Zurückstellen der Zeit ist unangenehm, deswegen werden Anti-Schaltmillisekunden eingeführt, die die Uhr bei Bedarf leicht abbremsen. Auch beim Vorstellen der Uhr werden meist Schaltmillisekunden verwendet.
NTP:
Server werden in Genauigkeitsstufen unterteilt, Stratum1 ist genau. Stratum 3 oder mehr ist inzwischen selten. Lokale NTP Server ermöglichen eine sehr genaue lokale Zeit im Netzwerk. Die Uhr ist nicht unbedingt genau die Globalzeit, alle lokalen Systeme laufen aber synchron.

2.3.2 Lamportzeit

Bei jedem lokalen Ereignis "tickt" die logische Uhr. Lokale Ereignisse können Instruktionen oder spezielle Ereignisse sein. Wenn ein Ereignis eintritt, wird diesem ein Zeitstempel zugewiesen. Ein späteres Ereignis tritt auch wirklich später ein und KANN kausal Abhängig von denn vorherigen Ereignissen des selben Systems sein. Kausale Abhängigkeiten zwischen verschiedenen Systemen kann nur durch Datenaustausch ausgelöst werden. Wenn zwei Ereignisse kausal abhängig sind, muss das abhängige Ereignis einen neueren Zeitstempel besitzen als die Ursache.
Lamportzeit:
Die logische Uhr tickt bei jedem Ereignis hoch. Der Sender schickt seinen Zeitstempel beim Senden mit. Der Empfänger stellt seine Uhr auf den größeren der beiden Zeitstempel (sein eigener und der vom Sender).
Man kann von der Kausalität auf die Zeitstempel schließen, z.B. zum korrigieren. Die Zeitstempel liefern jedoch keine Informationen über die Kausalität, frühere Ereignisse müssen nicht abhängig sein.
Problem: Zähler erreicht Maximum des Speichers und läuft über. Praktisch selten erreichbar, 64Bit Zeitstempel reicht LANGE.

2.3.3 Vektorzeit

Bei n Rechnern, ein Vektor der Dimension n.
Bei lokalen Ereignissen erhöht das System den eigenen Wert im Vektor, die anderen bleiben gleich. Beim Senden, wird der lokale Eintrag erhöht und der gesamte Vektor mitgeschickt. Beim Empfangen wird komponentenweise das Maximum in den eigenen Vektor übernommen, vorher wird auch der eigenen Wert um eins erhöht, da das Empfangsereignis ein lokales Ereignis ist.
Wenn ein Vektor komplett komponentenweise größer gleich ist als ein anderer, so ist das Ereignis kausal abhängig. Dabei MUSS mindestens eine Komponente echt größer sein. In allen anderen Fällen sind die Ereignisse unabhängig.