Aktorový model

Róbert Novotný

(UINF/KOPR, 10. december 2013)

Základný problém vláknovej konkurentnosti

Ako spravovať prístup k zdieľanej pamäti?

- mašinéria zámkov, monitorov…
- ťažko sa ladí
  - ťažké reprodukovať problémy
  - správanie je často nepredvídateľné
  - vývojár musí uvažovať na mnohých frontoch

Škálovateľnosť má hranice

Škálovateľnosť má hranice

- vlákna od istého momentu neškálujú
  - 10000 vlákien anyone?
- potrebujeme škálovať v oboch rovinách:
  - horizontálne: pridávame uzly / jadrá / CPU…
  - vertikálne: zvyšujeme frekvenciu, dodávame RAM…

Potreba iných modelov!

Všetkému je na vine...

- zrušme prístup k zdieľaným dátam a bude dobre!
  - kde nie sú zdieľané dáta, nie sú deadlocky, prepisy, ...
- žiadne zdieľané inštančné premenné...
- ako potom programovať?

Potreba iných modelov!

Aktor = výpočtová entita

since 1973

- vie prijímať a odosielať správy
- po prijatí správy môže:
  - odoslať konečný počet správ iným aktorom
  - vytvoriť konečný počet nových aktorov
  - zmeniť svoj stav
  - určiť správanie pre správy prijaté v budúcnosti
- možnosti sa môžu diať v ľubovoľnom poradí
  - a dokonca paralelne

Správa

- ľubovoľný **nemenný** objekt posielaný medzi aktormi
- jediný spôsob výmeny informácií / zdieľania dát medzi aktormi
- medzi aktormi neexistujú žiadne zdieľané dáta!

```
public class Správa {
   private final String data;
   public Správa(String data)
   {
      this.data = data;
   }
   public String getData()
   {
      return this.data;
   }
}
```

Asynchronicita

- správy sú odosielané **asynchrónne**
- nečaká sa na prijatie
  - **fire-and-forget**
- nezáleží na poradí prijatých správ
  - inšpirácia z packet-oriented systems (UDP?)
- v praxi: podporuje sa aj synchronicita

Buffrovanie správ: áno alebo nie?

- jeden názor: správy sa nebufferujú: všetko sa deje naraz
- kompromisný: „správy sa bufferujú v éteri"
- prakticky [Akka]: správy sa radia do buffera / frontu

Implementácie aktorového modelu

- Erlang
  - funkcionálny jazyk
  - namiesto zdieľaného stavu odovzdávanie správ
- Akka
  - open source framework pre aktorov
  - implementácie: Java / Scala

Aktory v Akke = akktory

- objekty so stavom a správaním
  - deja vu z OOP
- stav i správanie sa menia len na základe prijatej správy
- aktor v Akke: implementovaný ako vlákno
  - to nás nemá čo zaujímať ;-)

```
import akka.actor.UntypedActor;
import akka.event.*;
public class ExampleActor extends UntypedActor {
  private LoggingAdapter log
          = Logging.getLogger(getContext().system(), this);

  public void onReceive(Object message) throws Exception {
    if (message instanceof String)
      log.info("Received String message: {}", message);
    else
      unhandled(message);
    }
  }
}
```

Hello World akktor

```
public static void main(String[] args) {
   ActorSystem system = ActorSystem.create();
   ActorRef actor = system.actorOf(
                        new Props(ExampleActor.class));

   for (int i = 0; i < 5; i++) {
      actor.tell("Hello at " + new Date());
   }
}
```

Thread-safety aktorových tried

- akktor spracuje správu pred spracovaním ďalšej správy
  - v súlade s "happens-before" zásadou z JVM
  - inštančné premenné akktora nemusia byť volatile a pod.
- implementácia cez ideu **mailboxu**

Mailboxy a buffrovanie

- akktor má mailbox v ktorom sa kopia správy
  - každý akktor má vlastný
  - nedajú sa zdieľať
- správy defaultne radené podľa času odoslania
- možno využiť prioritný mailbox:
  - radenie podľa priority
- mailbox je front!
  - aktor vidí **len** jeho začiatok

Dôležité rysy aktora

- aktor spracováva v danom čase **len jednu správu**
  - ostatné sa kopia vo fronte
  - dlhotrvajúce spracovanie správy znamená dlhšie čakanie!
- správy z aktora A do aktora B **garantujú poradie** doručenia
  - ak nemáme špeciálny typ mailboxu
  - nespoliehajme sa na poradie doručenia správ z rozličných aktorov!

Typické nasadenie

- rozsiahle transakčné operácie
  - bankové systémy
- SOAP/REST webservices
  - tony volaní, ktoré nepotrebujú stav
- dávkové spracovanie
  - map/reduce, grid..
- integrácia systémov
  - mediácia, transformácia správ

Ďalšie črty akktorov

- hierarchia akktorov
  - vhodné pri rozdeľovaní práce na podúlohy
- jednoznačná identifikácia
  - pomocou URI (hostname, port, cesta)
- a iné
  - voliteľná synchronicita posielania správ
  - podpora transakcií (model STM)

DEMO> POČÍTANIE FREKVENCIÍ SLOV

Frekvencia slov v dokumentoch

Napadlo mu: zlom dobro zlom.
V krajine Mu napadlo mnoho snehu.

napadlo:2, mu:2, zlom:2, dobro:1, v:1,
krajine:1, mnoho:1, snehu:1

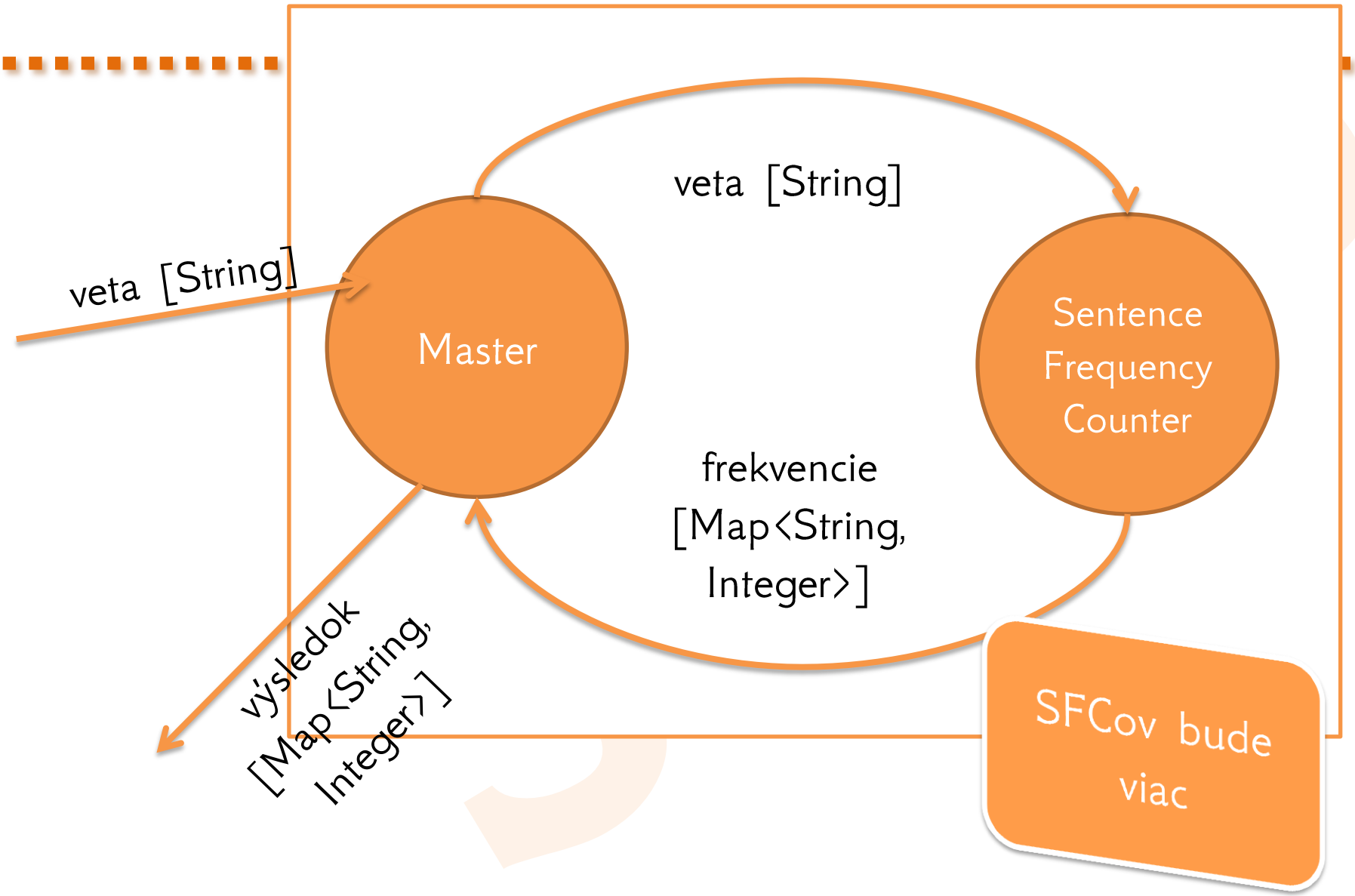
Ako na deľbu práce?

- Jednotlivé vety rozhadzujeme na **uzly**.
- **Uzol** zráta frekvencie slov v jednej vete.
- Samostatný **uzol** berie čiastkové výsledky a kumuluje ich.

uzol = aktor

Návrh v Akke

- potrebujeme navrhnúť správy tečúce medzi aktormi
- správa je identifikovaná svojim dátovým typom
  - trieda v Jave
- **odporúčanie:** z rozličných aktorov musia prichádzať správy rozličných typov


veta [String]
Master
veta [String]
Sentence Frequency Counter
frekvencie [Map<String, Integer>]
výsledok [Map<String, Integer>]
SFCov bude viac

```
public class Master extends UntypedActor {
   private Map<String, Integer> allFreqs
         = new HashMap<String, Integer>();
   public void onReceive(Object message) throws Exception {
      if (message instanceof String) {
         sentenceFrequencyCounters.tell(message, getSelf());
      } else if (message instanceof Map<?, ?>) {
         String sentenceFreqs = (Map<String, Integer>) message;
         aggregate(allFreqs, sentenceFreqs);
      } else {
         unhandled(message);
      }
   }
}
```

zaslanie vety do workerov

združenie frekvencií vety s globálnymi frekvenciami

```
public class Master extends UntypedActor {

   private Map<String, Integer> allFrequencies
       = new HashMap<String, Integer>();

   private ActorRef sentenceFrequencyCounters
     = getContext().actorOf(
         Props.create(SentenceCountActor.class).withRouter(
           new RoundRobinRouter(3)));

   ....
}
```

globálne frekvencie

vytvorený router s troma workermi, robota sa odovzdáva na striedačku

```
public class SentenceCountActor extends UntypedActor {
   @Override
   public void onReceive(Object message) throws Exception {
      if(message instanceof String) {
         String sentence = (String) message;
         Map<String, Integer> freqs = calculateFrequencies(sentence);
         getSender().tell(freqs, getSelf());
      } else {
         unhandled(message);
      }
   }
   …
}
```

odpoveď odosielateľovi (masterovi)

Zlé demo!

[vysvetlí sa neskôr]

Dve základné otázky

1. kde vidím výstup?
2. kedy systém skončí?

2. Kedy systém skončí?

Mýtus: keď sa vyprázdnia mailboxy

- čo keď príde o minútu ďalšia správa?

2. Kedy systém skončí?

- master nemôže čakať na dobehnutie workerov
- blokoval by príchod ďalších správ
- **blokovanie** v aktorovom modeli je **ZLO**

Akka neháda, kedy naša aplikácia skončí, musíme jej to povedať.

Riešenie č. 1: ak viem, koľko správ pošlem do systému

- viem, koľko správ pošlem do systému
  - koľko viet, toľko správ
- mastera vytvorí s počítadlom
- s každou došlou správou od workera znížim počítadlo
- ak počítadlo klesne na nulu:
  - **vypisujem výsledok**
  - **zatváram systém**

konštruktor

```
public Master(int numberOfSentences) {
        this.numberOfSentences = numberOfSentences;
}
```

poslané do konštruktora

```
public class Runner {
   public static void main(String[] args) {
      ActorSystem system = ActorSystem.create();
      ActorRef master
          = system.actorOf(Props.create(Master.class, 3));
```

Master#onHandle()

```
else if (message instanceof Map<?, ?>) {
    String sentenceFreqs = (Map<String, Integer>) message;
    aggregate(allFreqs, sentenceFreqs);
    numberOfSentences--;
    if(numberOfSentences == 0) {
        logger.info(allFrequencies.toString());
        getContext().system().shutdown();
    }
}
```

alternatívne: pošlem správu do tretieho aktora, vypisovača

Výsledný projekt

https://github.com/novotnyr/akka-wordfrequencies

Nabudúce:

Ako zavrieť aplikáciu, keď nevieme dopredu počet správ?