Obrada podataka u realnom vremenu

Osnovni koncepti, Kafka, Storm, Spark Streaming

1

Sadržaj

- Osnovi obrade podataka u realnom vremenu
- Sistemi za razmenu poruka
- Apache Kafka
- Obrada tokova podataka
- Apache Storm
- Apache Spark (obrada u realnom vremenu)

2

Osnovi obrade podataka u realnom vremenu

3

Osnovi obrade podataka u realnom vremenu

- Paketna obrada podataka i tokovi podataka
  - obrađuje se velika količina podataka
    - rezultat izvedeni podaci koji se prikazuju ili koriste za *ad-hoc* upite
  - izvedeni podaci se uvek mogu dobiti ponovnim izvršenjem iste funkcije nad ulaznim podacima
  - poznata veličina glavnog skupa podataka
    - pa je moguće obezbediti završetak izvršavanja algoritma paketne obrade
  - **problem**: u većini sistema podaci neprestano pristižu
    - i potrebno ih je što pre obraditi
    - u slučaju paketne obrade, potrebno je ograničiti se na deo svih podataka i nad njima izvršiti analizu
      - svi podaci pristigli u međuvremenu će biti deo sledeće paketne obrade

4

4

Osnovi obrade podataka u realnom vremenu

- Obrada podataka u realnom vremenu
  - rešenje problema primene paketne obrade podataka s tokovima podataka
    - dugo čekanje na rezultat obrade
    - obrada neograničenog toka podataka
  - vrste obrade podataka u realnom vremenu
    - **obrada podataka u realnom vremenu**
      - engl. *realtime processing*
    - **obrada podataka približno u realnom vremenu**
      - engl. *near-realtime / soft realtime processing*
    - kategorizacija ne zavisi od implementacione tehnike već samo od garantovanja vremenskog okvira izvršenja
      - moguće imati obradu nad tokovima podataka ili obradu putem specijalizovanog hardvera

5

5

Osnovi obrade podataka u realnom vremenu

- Vrste obrade podataka u realnom vremenu
  - **obrada podataka u realnom vremenu**
    - obrada podataka u kojoj postoje **jasni vremenski okviri u kojima je rezultat zagarantovan**
      - vremenski okviri su izuzetno kratki, mere se u milisekundima i sekundima
      - npr. algoritam trgovine deonicama na berzi mora da garantuje rezultat obrade podataka u okviru 10ms
    - obrada koja je izuzetno teška za implementaciju
      - sa softverske tačke gledišta, potrebni jezici višeg nivoa apstrakcije
      - zadaci obrade podataka obično nisu izolovani
        - već zavise od prethodno sračunatih vrednosti
        - sistem mora izvršavati prioritetnije zadatke pa može doći do kašnjenja izvršenja nekog zadatka

6

6

Osnovi obrade podataka u realnom vremenu

- Vrste obrade podataka u realnom vremenu
  - **obrada podataka u približno realnom vremenu**
    - obrada podataka u **kojoj postoje vremenski okviri u kojima je rezultat zagarantovan ali sistem ne garantuje da će taj okvir biti uvek i ispoštovan**
      - npr. u 99% slučajeva će vremenski okvir biti ispoštovan
      - obično postoji značajna degradacija performansi ukoliko se propusti nekoliko vremenskih okvira
    - češće susretana vrsta obrade podataka

7

7

Osnovi obrade podataka u realnom vremenu

- Obrada tokova podataka
  - engl. *stream processing*
  - odnosi se na metodologiju implementacije obrade u kojoj se obrađuju neprekidni tokovi podataka dobijeni od izvora
  - ne moraju postojati definisani vremenski okviri u kojima se garantuje obrada
    - podaci se obrađuju istom brzinom kojom i pristižu
      - ali je moguća i situacija u kojoj ti podaci i čekaju na obradu
    - obično se povezuje sa obradom u (približno) realnom vremenu
  - obično se postavljaju dva zahteva pred sistem za obradu tokova podataka
    - **da je učestanost generisanja rezultata obrade podataka veća ili makar jednaka sa učestanošću prikupljanja podataka**
      - inače nastaju problemi skladištenja podataka koji čekaju na obradu
    - **da sistem za obradu poseduje dovoljno memorije da skladišti sve ulazne podatke koji pristižu u trenucima kada je sistem zauzet obradom kompleksnog tekućeg podatka**

8

8

Osnovi obrade podataka u realnom vremenu

- Obrada tokova podataka
  - u opštem slučaju, tok podataka se odnosi na podatke **koji se neprestano obuhvataju iz nekog izvora podataka**
    - npr. *stdin*, *stdout*, *FileInputStream*, TCP *connection*, VOIP
  - svodi se na obradu tokova **događaja**
    - engl. *event stream processing*
    - događaj je atomički, nepromenljiv podatak koji opisuje događaj iz realnog sveta
      - nad kojim se izvršava obrada podataka
      - analogan torci u sistemima paketne obrade datoteka
      - npr. akcija korisnika u sistemu, merenje sa senzora, linija u logu aplikativnog servera itd.

9

9

Osnovi obrade podataka u realnom vremenu

- Obrada tokova podataka
  - događaj je generisan od strane jednog **generatora**
    - engl. *producer, publisher, sender*
  - događaj može biti prihvaćen od strane više **primaoca**
    - engl. *consumers, subscribers, recipients*
  - srodni događaji mogu biti grupisani u **teme** ili **tokove podataka**
    - engl. *topic, stream*
  - primaoci poruke su obavešteni kada postoji novi događaj koji je potrebno obraditi (engl. *push*)
    - efikasnije rešenje nego u slučaju kada primaoci poruke pitaju bazu podataka da li postoji izmena usled veće iskorišćenosti komunikacionog kanala (engl. *pull*)
      - kao što je slučaj kod paketne obrade podataka

10

10

Sistemi za razmenu poruka

11

Sistemi za razmenu poruka

- Sistemi za razmenu poruka
  - komunikacioni sistemi specijalizovani za razmenu velikog broja poruka
  - uobičajen način za obaveštavanje primaoca poruke da postoji nova poruka/događaj za obradu
    - generator poruke šalje poruku sistemu za razmenu poruka koji je prosleđuje primaocima
      - najjednostavnija implementacija - **direktna komunikacija**
        - UNIX pipe ili TCP konekcija između generatora i primaoca poruke
        - mana je što se svakim direktnim komunikacionim kanalom povezuju samo jedan generator i jedan primalac
      - češća implementacija - **posredna komunikacija**
        - generator objavljuje (engl. *publish*) događaje u okviru tema
        - primaoci poruka se pretplaćuju (engl. *subscribe*) na teme putem kojih dobijaju podatke
        - ovakav model komunikacije se naziva model objave i pretplate (engl. *publish-subscribe*) ili skraćeno model komunikacije ***pub-sub***

12

12

Sistemi za razmenu poruka

- Sistemi za razmenu poruka - modeli razmene poruka
  - odabir modela razmene poruka zavisi od vrste komunikacije koju je potrebno ostvariti
  - prvi kriterijum odabira je broj učesnika u komunikaciji
    - **jedan-na-jedan**
      - poruka koju šalje jedan klijent je opslužena od strane jednog servisa
    - **jedan-na-više**
      - poruka koju šalje jedan klijent je opslužena od strane više instanci servisa
  - drugi kriterijum odabira je vremenski sled elemenata interakcije
    - **sinhrona**
      - klijent čeka na odgovor na poruku od servisa
    - **asinhrona**
      - klijent ne blokira izvršenje dok čeka na odgovor od servisa kojem je poslao poruku
        - odgovor može biti i opcion

13

13

Sistemi za razmenu poruka

- Sistemi za razmenu poruka - modeli razmene poruka

	jedan-na-jedan	**jedan-na-više**
sinhrona	zahtev/odgovor	-
asinhrona	notifikacije	objava/pretplata
	zahtev/asinhroni odgovor	objava/asinhroni odgovor

14

14

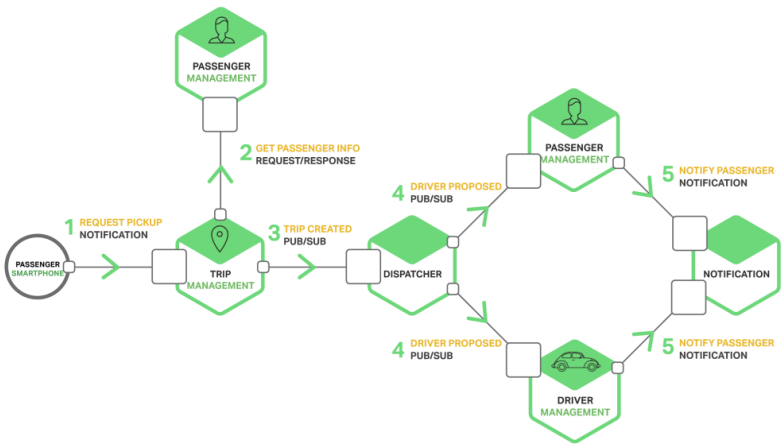
Sistemi za razmenu poruka

- Sistemi za razmenu poruka
  - jedan-na-jedan
    - **zahtev/odgovor** (engl. *request/response*)
      - klijent šalje zahtev ka servisu i čeka na odgovor
      - klijent je blokiran dok čeka na odgovor
    - **notifikacije** (engl. *notification*)
      - klijent šalje poruku servisu i ne očekuje odgovor
    - **zahtev/asinhroni odgovor** (engl. *request/async response*)
      - klijent šalje zahtev ka servisu i očekuje odgovor
      - klijent nije blokiran dok čeka na odgovor
  - jedan-na-više
    - **objava/pretplata** (engl. *publish/subscribe*)
      - klijent objavljuje poruku za koju može biti zainteresovano nula ili više servisa
    - **objava/asinhroni odgovor** (engl. *publish/async response*)
      - klijent objavljuje poruku i čeka predefinisani period vremena na odgovore od zainteresovanih servisa

15

15

Sistemi za razmenu poruka



izvor: https://dzone.com/articles/building-microservices-inter-process-communication

16

16

Sistemi za razmenu poruka

- Model komunikacije *pub-sub*
  - postoji više načina implementacije ovog modela
    - nijedan nije namenjen za sve moguće situacije
  - dva pitanja od kojih zavisi način implementacije *pub-sub* sistema
    - **1. šta se događa ako generator šalje poruke većom brzinom od one kojom je primalac poruke obrađuje?**
      - sistem može da **odbaci poruke** koje nisu obrađene
      - sistem može da **smešta poruke u red čekanja** na obradu
        - potrebno znati šta se događa u slučaju da se red čekanja prepuni
          - da li sistem otkaže ili se preliva na disk i kako to utiče na performanse?
      - sistem može da **blokira generator** dok se tekuće poruke ne obrade
        - engl. *backpressure*, *flow control*
        - postoji mali *buffer* koji kada se popuni generator je blokiran dok se memorija ne oslobodi

17

17

Sistemi za razmenu poruka

- Model komunikacije *pub-sub*
  - dva pitanja od kojih zavisi način implementacije *pub-sub* sistema
    - **2. šta se događa ukoliko deo sistema za razmenu poruka otkaže? da li se gube poruke u slučaju otkaza?**
      - moguća je kombinacija replikacije i pisanja poruka na disk
      - ukoliko nije potrebno garantovati isporuku svake poruke
        - moguće je postići visoku propusnost i manje vreme čekanja na obradu
        - za neke primene ovo jednostavno nije dopustivo

18

18

Sistemi za razmenu poruka

- Model komunikacije *pub-sub*
  - **direktna komunikacija generatora i primaoca poruka**
    - UDP *multicast*, osnovne ZeroMQ implementacije, komunikacija sa servisima putem RPC/HTTP protokola (*webhooks*)
    - mane:
      - aplikacije moraju voditi računa o gubitku poruka
      - podrazumevaju da su generatori i primaoci neprestano pokrenuti

19

19

Sistemi za razmenu poruka

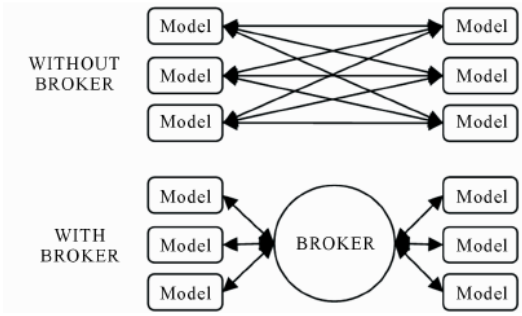
- Model komunikacije *pub-sub*
  - **posredna komunikacija generatora i primaoca poruka**
    - postoji posrednik u komunikaciji
      - naziva se **broker** ili **red čekanja poruka** (engl. *message broker, message queue*)
      - koji predstavlja skladište podataka optimizovano za rad sa tokovima podataka
    - broker predstavlja server u klijent-server arhitekturi sistema
      - generatori i primaoci poruka ostvaruju konekciju sa brokerom
      - generatori objavljuju svoje poruke na brokeru
        - raspoređene po temama
      - primaoci poruka se pretplaćuju na poruke od brokera
        - pretplaćuju se na teme od interesa

20

20

Sistemi za razmenu poruka

- Model komunikacije *pub-sub*



izvor: *Daniel Salas, Xu Liang, Yao Liang, A Systematic Approach for Hydrological Model Couplings*

21

21

Sistemi za razmenu poruka

- Model komunikacije *pub-sub*
  - posredna komunikacija generatora i primaoca poruka
    - postojanje brokera omogućava **dinamičko priključivanje i isključivanje** klijentskih procesa
    - pitanje trajanja podataka koji se šalju je u nadležnosti brokera
      - umesto da aplikacije vode računa o tome
      - neki brokeri čuvaju poruke samo u memoriji, dok drugi smeštaju poruke i na disk
    - obično omogućavaju prikupljanje poruka u redu čekanja
      - posledica je **asinhrona komunikacija generatora i primaoca**
        - generator samo čeka da broker potvrdi prijem poruke

22

22

Sistemi za razmenu poruka

- Model komunikacije *pub-sub*
  - brokeri i baze podataka u komunikaciji i integraciji sistema
    - sličnosti
      - čuvaju podatke o događajima duži vremenski period / trajno
      - mogu da implementiraju dvofazni protokol potvrđivanja transakcije
      - poseduju mehanizme grupisanja/organizovanja podataka i pretrage po grupama
    - razlike
      - baze podataka čuvaju podatke dok se oni eksplicitno ne obrišu, brokeri najčešće brišu podatke nakon uspešnog dostavljanja primaocu poruke
      - zbog čestog brisanja poruka, memorija koji broker zauzima je manja
      - baze podataka podržavaju indeksne strukture

23

23

Sistemi za razmenu poruka

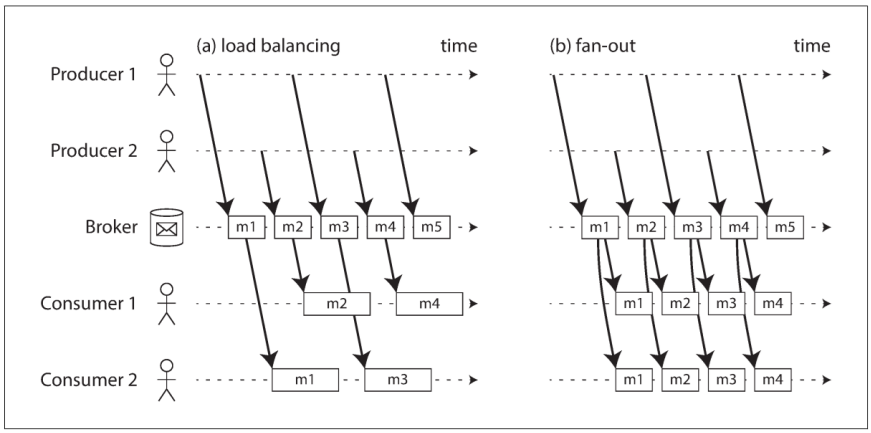
- Model komunikacije *pub-sub*
  - brokeri i više primaoca poruka
    - načini slanja poruke primaocima
      - **izjednačavanje opterećenja** (engl. *load balancing*)
        - svaka poruka je dostavljena tačno jednom primaocu
        - moguć je slučajan odabir primaoca poruka
        - šablon je koristan **kada se poruke dugotrajno obrađuju** ili se želi postići lako skaliranje i paralelizacija obrade prostim dodavanjem novih primaoca
      - **slanje svim primaocima** (engl. *fan-out*)
        - svaka poruka se dostavlja svim primaocima
        - šablon dozvoljava da se primaoci povežu na broker bez da utiču na druge primaoce
      - **kombinovani pristup**
        - selektivan odabir primaoca, za različite teme različiti pristupi

24

24

Sistemi za razmenu poruka

- Model komunikacije *pub-sub*



izvor: Martin Kleppmann: *Designing data-intensive applications, O'Reilly*

25

25

Sistemi za razmenu poruka

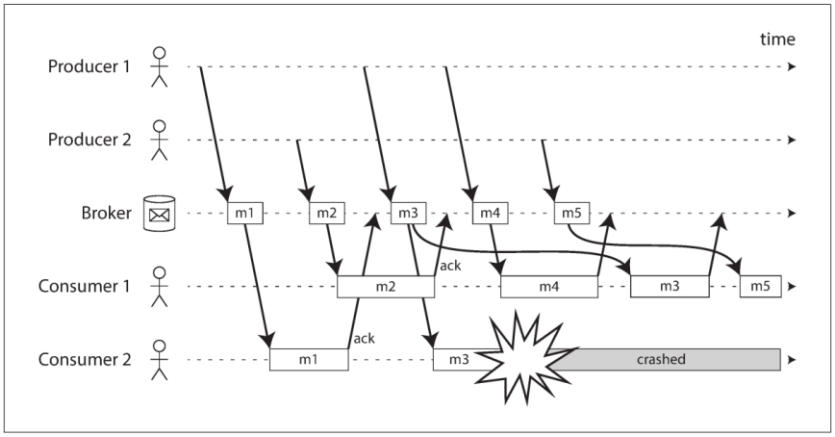
- Model komunikacije *pub-sub*
  - potvrda prijema i ponovno slanje poruka
    - brokeri zahtevaju potvrdu da je primalac obradio poruku kako bi je mogli ukloniti iz reda čekanja
      - **u ovom slučaju čitanje poruke je destruktivna operacija**
    - u slučaju izostanka potvrde prijema broker poruku šalje ponovo
      - istom ili drugom primaocu poruke
    - mehanizam potvrde u kombinaciji sa izjednačavanjem opterećenja dovodi do neredosledne obrade poruka
      - potencijalni problem u nekim sistemima

26

26

Sistemi za razmenu poruka

- Model komunikacije *pub-sub*



izvor: Martin Kleppmann: *Designing data-intensive applications, O'Reilly*

27

27

Sistemi za razmenu poruka

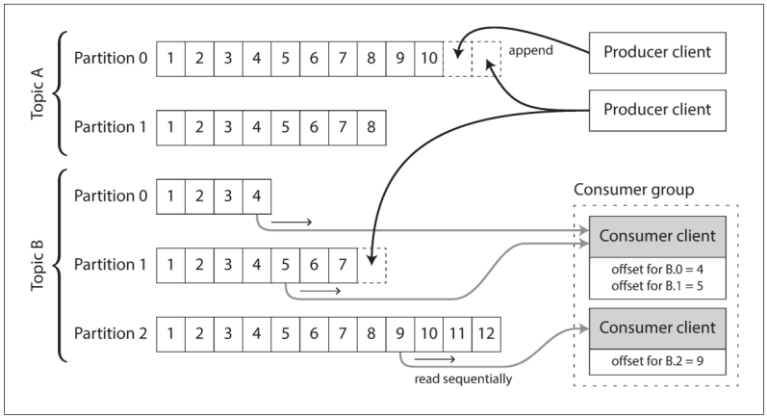
- Model komunikacije *pub-sub*
  - u tradicionalnim brokerskim arhitekturama, dodavanje novih primaoca poruke u sistem omogućava primaocu samo da primi novopridošle poruke
    - sve stare poruke se ne čuvaju, tj. obrisane su nakon uspešnog slanja i potvrde prijema
  - **brokeri zasnovani na logovima**
    - logovi su struktura podataka na disku koja dozvoljava dodavanje zapisa na kraj
      - podatak koji je generator poslao, broker smešta u log
      - primalac prima poruke sekvencijalnim čitanjem iz loga
    - radi poboljšanja performansi, **log može biti particionisan**
      - različite particije mogu biti raspoređene na različite računare
      - teme mogu obuhvatati više particija koje sadrže poruke istog tipa
    - u okviru svake particije broker dodaje **redni broj zapisa**
      - radi praćenja obrađenih podataka
    - jedan od najpoznatijih predstavnika je ***Apache Kafka***

28

28

Sistemi za razmenu poruka

- Model komunikacije *pub-sub*



izvor: Martin Kleppmann*: Designing data-intensive applications, O'Reilly*

Sistemi za razmenu poruka

- Model komunikacije *pub-sub*
  - brokeri zasnovani na logovima
    - podrazumevana **podrška slanju poruka svim primaocima**
      - usled činjenice da primaoci mogu nezavisno da čitaju poruke iz particija logova
        - bez uticaja na druge primaoce jer čitanje ne briše poruku iz loga
    - moguća **podrška za izjednačavanje opterećenja**
      - dodelom celih particija primaocima poruka
        - svaki primalac čita isključivo svoje particije
      - mane ovog pristupa
        - broj primaoca je ograničen brojem particija
        - spora obrada poruke blokira obrade narednih poruke iste particije

Sistemi za razmenu poruka

- Model komunikacije *pub-sub*
  - brokeri zasnovani na logovima
    - mogu da prate dokle je primalac poruke stigao sa čitanjem
      - ***offset* primaoca** (engl. *consumer offset*)
      - sve poruke čiji je redni broj manji od *offset*-a su pročitane
        - veći ili jednak za nepročitane poruke
      - broker ne ažurira *offset* poruke za svaku pročitanu poruku
        - već periodično, tako smanjujući saobraćaj i povećavajući propusnu moć sistema zasnovanog na logovima

31

31

Sistemi za razmenu poruka

- Model komunikacije *pub-sub*
  - brokeri zasnovani na logovima
    - upravljanje prostorom na disku u vidu kružnih bafera (engl. *circular buffer, ring buffer*)
      - baferi se nalaze na disku pa mogu biti prilično veliki
      - stari segmenti se periodično brišu ili arhiviraju
        - **kompakcija logova**
    - čitanje poruke ne izaziva uklanjanje poruke iz loga
      - **čitanje poruke nije destruktivna operacija**
        - jedini bočni efekat čitanja je uvećavanje *offset*-a primaoca
      - dozvoljava se ponovno čitanje poruke od proizvoljnog trenutka u vremenu

32

32

Apache Kafka

33

Kafka

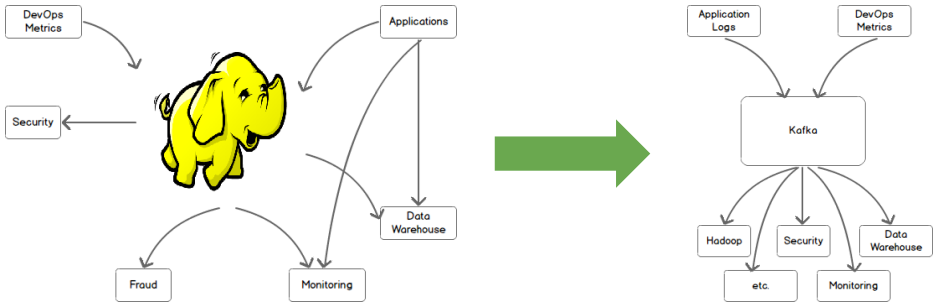
- Kafka
  - distribuirani sistem za razmenu poruka
    - zasnovan na *pub-sub* mehanizmu
    - poseduje broker zasnovan na logovima
  - glavne osobine
    - brza razmena poruka
    - velika mogućnost skaliranja
    - mogućnost uvođenja redundanse kroz replikaciju poruka

34

34

Kafka

- Kafka
  - motivacija: smanjenje kompleksnosti razmene poruka između velikog broja komponenti u sistemu velikih skupova podataka



izvor: Kevin Sookocheff, *Kafka in a Nutshell:* https://sookocheff.com/post/kafka/kafka-in-a-nutshell/

35

35

Kafka

- Kafka - osnovni elementi
  - tema (engl. *topic*)
    - skup poruka koje su bliske po značenju ili svrsi
    - sve poruke se upisuju u teme i čitaju iz tema
  - generator poruka (engl. *producer*)
    - predstavlja izvor poruke koju upisuje u odgovarajuću temu
  - primalac poruka (engl. *subscriber*)
    - obrađivač poruke koju je prethodno pročitao iz željene teme
  - broker (engl. *broker*)
    - svaki čvor u distribuiranom Kafka klasteru
    - sadrži teme sa porukama

36

36

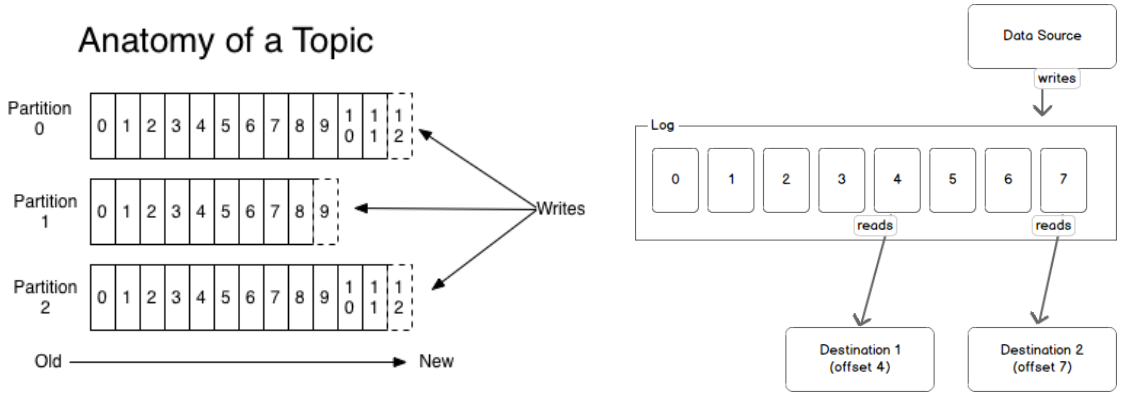
Kafka

- Kafka - teme
  - tema je podeljena u **particije**
    - **distribuirane** su u klasteru
      - tako da različiti brokeri dobijaju različite particije
    - omogućavaju **paralelizaciju rada** sa temama
      - više primaoca poruke može da čita poruke istovremeno
      - jedan primalac može da čita poruke iz više brokera
    - svaka poruka ima svoj identifikator koji se zove ***offset***
      - poruke su uvek sortirane u rastućem redosledu *offset*-a
      - *offset*-ima upravlja alat Kafka
      - omogućava primaocima poruke da prate dokle su stigli sa čitanjem
        - **Kafka ne vodi računa do kojeg *offset*-a je sa čitanjem primalac stigao**
  - poruke se skladište predefinisani period vremena (engl. *retention period*)
    - nakon isteka tog vremena, poruke se brišu iz sistema
    - na primaocima je da vode računa o isteku poruka

37

Kafka

- Tema

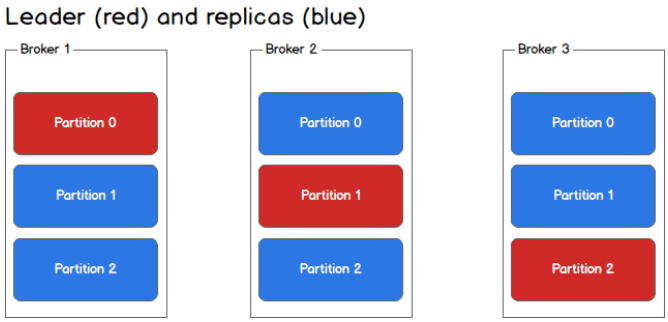


izvor: Kevin Sookocheff, *Kafka in a Nutshell:* https://sookocheff.com/post/kafka/kafka-in-a-nutshell/

38

Kafka

- Kafka - brokeri i particije
  - brokeri sadrže više particija
    - svaka particija može biti **vodeća particija** ili **kopija (replika) particije**
    - svako pisanje nove poruke je koordinisano preko vodeće particije
    - kada vodeća particija postane nedostupna jedna od kopija postaje vodeća

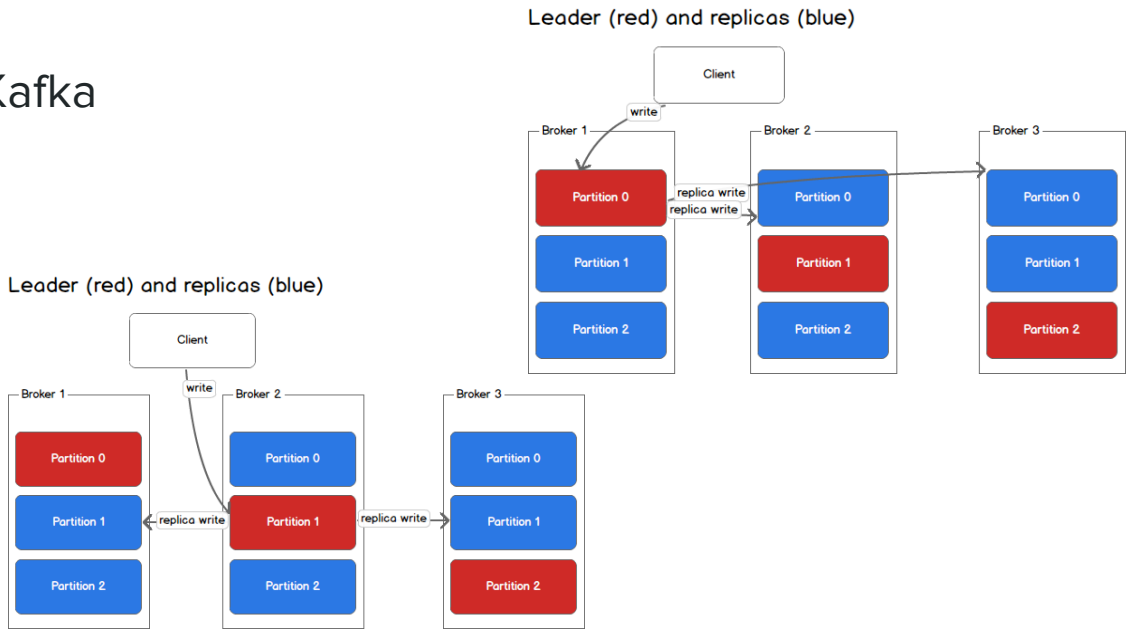


39

izvor: Kevin Sookocheff, *Kafka in a Nutshell:* https://sookocheff.com/post/kafka/kafka-in-a-nutshell/

39

Kafka



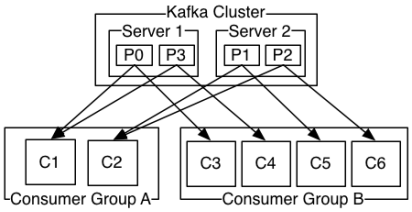
40

izvor: Kevin Sookocheff, *Kafka in a Nutshell:* https://sookocheff.com/post/kafka/kafka-in-a-nutshell/

40

Kafka

- Kafka - primaoci poruka i grupe primaoca
  - primaoci poruka mogu da budu podešeni da čitaju iz bilo koje particije
    - što omogućava veću propusnost sistema
  - primaoci mogu biti organizovani u grupe koje čitaju određenu temu
    - svaki primalac poruke unutar grupe je zadužen za odgovarajući broj particija u temi
      - ukoliko postoji više primaoca od particija neki primaoci neće raditi ništa
      - ukoliko postoji više particija nago primaoca, neki primaoci će čitati iz više particija



izvor: Kevin Sookocheff, *Kafka in a Nutshell:* https://sookocheff.com/post/kafka/kafka-in-a-nutshell/

41

41

Kafka

- Kafka - konzistentnost i dostupnost brokera
  - Kafka pruža sledeće garancije pod uslovom da postoje **najviše jedan generator i najviše jedan primalac** poruke po particiji
    - poruke će biti dodate u particiju u redosledu u kojem su poslate
    - jedna instanca primaoca vidi poruke u redosledu u kojem su smeštene u particiju
      - redosled dodavanja i čitanja na nivou teme nije zagarantovan
    - poruka je u stanju *commited* tek kada je snimljena u sve ažurne replike
    - nijedna poruka u stanju *commited* neće biti izgubljena dokle god postoji makar jedna ažurna replika
  - **ažurna replika** (engl. *in sync replica*)
    - replika koja poseduje poslednju sliku stanja particije

42

42

Kafka

- Kafka - **rukovanje otkazima brokera**
  - u slučaju otkaza replike particije
    - upis poruka u sistem neće dolaziti do replike u otkazu
      - ona će zaostajati za originalom
  - u slučaju otkaza vodeće particije
    - druga ažurna replika postaje vodeća
  - u slučaju otkaza svih replika uključujući i vodeću particiju
    - vodeća particija se smatra za jedinu ažurnu kopiju jer nakon njenog otkaza ne postoji više kopija u koje je moguće upisati podatke
    - moguća rešenja
      - čekati na oporavak vodeće particije i minimizovati gubitak podataka
      - čekati na oporavak bilo kojeg brokera sa replikom i nju proglasiti za vodeću
        - minimizuje se vreme čekanja na oporavak nauštrb gubitka nekih podataka jer ta replika možda nije bila ažurna u trenutku otkaza

43

43

Kafka

- Kafka - konzistentnost generatora i primalaca poruka
  - **konzistentnost upisa poruke** (generatora) može biti podešena odabirom sledećih tipova čekanja na upis poruke na brokeru:
    - čekati da sve ažurne replike potvrde upis poruke
    - čekati da vodeća kopija potvrdi upis poruke
    - ne čekati uopšte na potvrdu prijema poruke
  - **konzistentnost čitanja poruka** (primaoca) može imati sledeće pojavne oblike:
    - **čitanje poruke najviše jednom**
    - **čitanje poruke najmanje jednom**
    - **čitanje poruke tačno jednom**

44

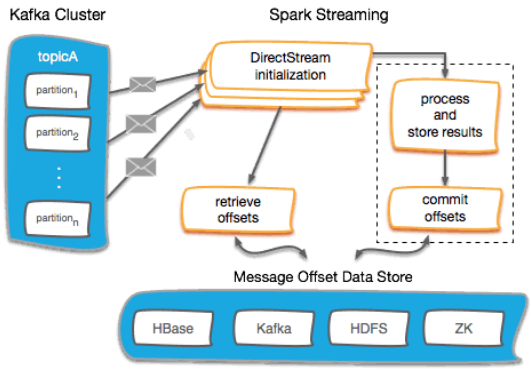
44

Kafka

- Kafka - primaoci poruka
  - **čitanje poruke najviše jednom**
    - primalac nakon preuzimanja poruke odmah snima *offset* i onda obrađuje poruku
    - u slučaju otkaza, čitanje se nastavlja od sledeće poruke u particiji
  - **čitanje poruke najmanje jednom**
    - primalac obrađuje poruku nakon preuzimanja i tek nakon obrade snima *offset*
    - moguć otkaz između obrade i slanja *offset*-a
      - višestruka obrada poruke
      - bez gubitka poruke
    - **najčešći model** čitanja poruke
  - **čitanje poruke tačno jednom**
    - primalac snima *offset* i rezultat izvršenja poruke kao deo jedne transakcije
    - nema višestruke obrade poruka niti gubitka poruka
    - loše performanse sistema usled postojanja transakcija

Kafka

- Kafka - primaoci poruka
  - upravljanje *offset*-om
    - koristeći eksterno skladište
      - HBase, Zookeeper ...
    - koristeći Kafku
      - na posebnu temu



izvor: *Kafka offset management:* https://blog.cloudera.com/offset-management-for-apache-kafka-with-apache-spark-streaming/

Kafka

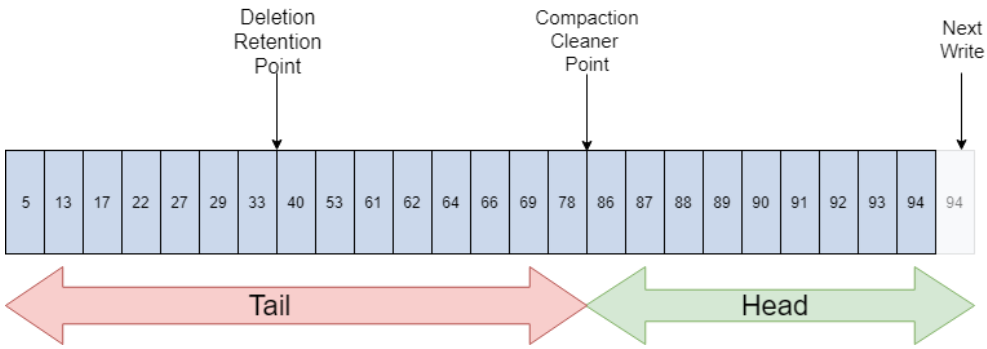
- Kafka - **kompakcija logova**
  - engl. *log compaction*
  - omogućava uštedu prostora **čuvanjem samo poslednje vrednosti torke za neki ključ u okviru jedne particije**
    - čuva se samo poslednja izmena za svaku torku u nekoj temi
    - postoji ključ po kojem se mogu prepoznati torke koje predstavljaju isti događaj
  - omogućava lak oporavak od otkaza
  - kompaktni log poseduje dve oznake
    - **glavu** (engl. *head*)
      - predstavlja poslednju torku u logu nakon koje se dodaju nove torke
    - **rep** (engl. *tail*)
      - predstavlja kraj loga na kojem se vrši kompakcija

47

47

Kafka

- Kafka - kompakcija logova



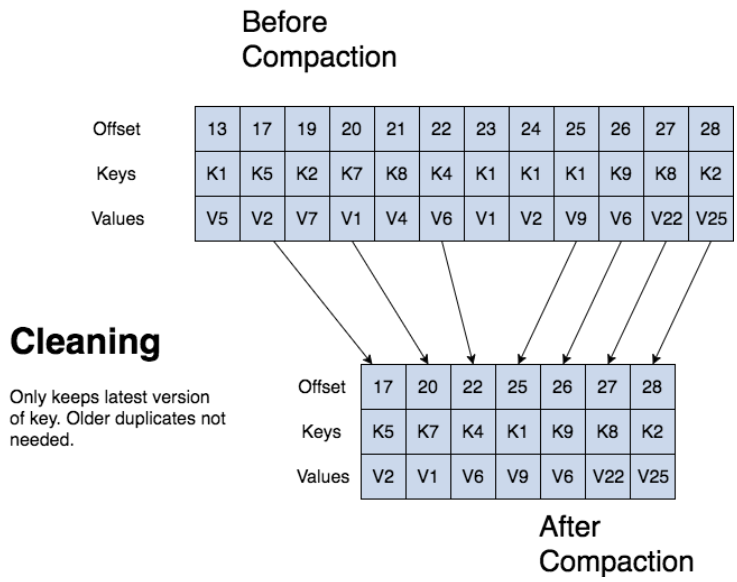
48

izvor: Cloudurable, *Kafka Architecture: Log Compaction*: http://cloudurable.com/blog/kafka-architecture-log-compaction/index.html

48

Kafka

- Kafka - kompakcija logova



49

izvor: Cloudurable, *Kafka Architecture: Log Compaction:* http://cloudurable.com/blog/kafka-architecture-log-compaction/index.html

49

Obrada tokova podataka

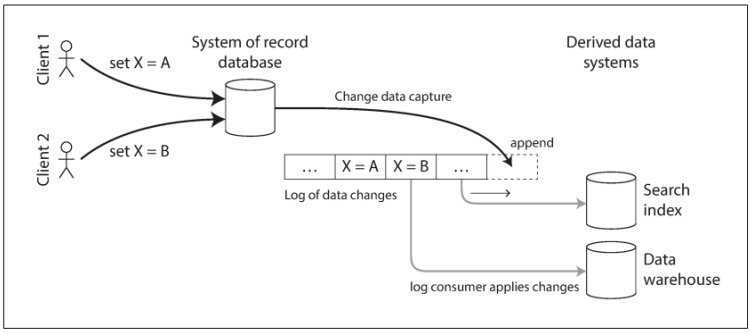
50

Obrada tokova podataka

- Obrada tokova podataka
  - postoje tri vrste obrade i moguća ciljna odredišta za tokove podataka
    - **1. podatke iz događaja snimiti u sva potrebna skladišta podataka**
      - baze podataka, indekse za pretragu, *cache* memorije
      - čuvamo skladišta podataka ažurnim i tako dozvoljavamo različite vrste upita
      - primalac poruke je skladište podataka
    - **2. podatke iz događaja direktno proslediti korisnicima**
      - slanjem email-a, notifikacije, vizualizacijom sadržaja toka podataka na kontrolnom panelu aplikacije
      - primalac poruke je krajnji korisnik
    - **3. događaje iz jednog toka podataka obraditi i kreirati novi tok podataka**
      - arhitektura sistema za obradu obuhvata višestruke tokove podataka i programe za obradu tokova

Obrada tokova podataka

- Obrada tokova podataka
  - tri opcije za obradu tokova podataka
    - **1. podatke iz događaja snimiti u sva potrebna skladišta podataka**



izvor: Martin Kleppmann: *Designing data-intensive applications, O'Reilly*

Obrada tokova podataka

- Obrada tokova podataka
  - **3. događaje iz jednog toka podataka obraditi i kreirati novi tok podataka**
    - kôd za obradu podataka u ovakvoj arhitekturi nosi naziv **operator** ili **posao**
      - engl. *operator, job*
    - slično principima paketne obrade u MapReduce programima
      - obrađivač čita ulazne podatke/događaje i rezultat snima na drugu lokaciju tako što dodaje torke na kraj izlazne datoteke ili skladišta podataka
      - važe slični principi paralelizacije i particionisanja
    - za razliku od paketne obrade
      - **tokovi podataka nemaju kraj**
        - ne postoji predefinisana količina podataka nad kojom je potrebno pokrenuti obradu što ima mnogobrojne posledice na način obrade
          - npr. sortiranje i *sort-merge* spoj takođe nemaju smisla

53

53

Obrada tokova podataka

- Obrada tokova podataka - slučajevi korišćenja
  - tradicionalni slučajevi korišćenja
    - detekcija prevare u bankarskim sistemima
      - da li korišćenje kartice u nekoj situaciji odgovara šablonu ponašanja korisnika?
    - sistemi za berzansko poslovanje
      - obrađuju kretanja na tržištu i shodno tome obavljaju transakcije
    - sistemi za upravljanje proizvodnjom
      - treba da reaguju na pojedine događaje u postrojenju
    - vojne i špijunske akcije
      - praćenje neprijateljskih kretanja i identifikacija potencijalnih opasnosti

54

54

Obrada tokova podataka

- Obrada tokova podataka - slučajevi korišćenja
  - noviji slučajevi korišćenja
    - obrada složenih događaja (engl. *complex event processing*, CEP)
      - **pristup analizi tokova podataka radi pronalaženja uzoraka koji odgovaraju traženim šablonima**
        - inverzan pristup u odnosu na baze podataka
        - čuva se upit a podaci su tranzijentni
      - koristi se deklarativni jezik visokog nivoa apstrakcije ili grafički korisnički interfejs
        - za specifikaciju traženih šablona
      - obrađivač prima specificirani šablon i tok podataka
        - i održava unutrašnju **mašinu stanja** (engl. *state machine*)
        - ako se pronađe deo toka koji odgovara šablonu obrađivač generiše složeni događaj koji obuhvata sve događaje koji odgovaraju šablonu
      - npr. alat *Samza*
        - omogućava deklarativne upite nad tokovima podataka

55

55

Obrada tokova podataka

- Obrada tokova podataka - slučajevi korišćenja
  - noviji slučajevi korišćenja
    - analitika nad tokovima podataka
      - obuhvata primenu statističkih metoda i agregacija nad velikim brojem događaja
        - i to je razlikuje od CEP-a koji je više okrenut ka pronalaženju šablona
        - obično se izračunavaju u fiksnim vremenskim intervalima (engl. *window*)
        - mogu da koriste i probabilističke agregatore (npr. *Bloom filter*, *HyperLogLog*)
        - npr.
          - merenje učestanosti nekog tipa događaja
          - izračunavanje prosečne vrednosti u nekom periodu vremena
          - poređenje trenutno izračunatih vrednosti sa vrednostima izračunatim u nekom od prethodnih perioda vremena
            - detekcija trendova
      - npr. alati *Apache Storm, Spark Streaming, Flink, Concord, Samza* i *Kafka Streams*

56

56

Obrada tokova podataka

- Obrada tokova podataka - slučajevi korišćenja
  - noviji slučajevi korišćenja
    - održavanje materijalizovanih pogleda
      - za razliku od prethodnih primena, zahteva analizu svih ili velikog broja događaja
        - ne isključivo podskupa ograničenog fiksnim vremenskim intervalom
      - npr. *Samza* i *Kafka Streams*
        - **omogućeno Kafka mehanizmom za kompakciju logova**

57

57

Obrada tokova podataka

- Obrada tokova podataka - slučajevi korišćenja
  - noviji slučajevi korišćenja
    - pretraga u tokovima podataka
      - potrebno pronaći konkretan događaj od interesa
        - oslanjajući se na složene obrasce pretrage
        - za razliku od CEP-a koji pronalazi skupove elemenata po šablonu
      - snima se upit i primenjuje nad svakim događajem u toku podataka
        - za razliku od klasičnih sistema za pretragu
          - kod kojih se indeksiraju svi dokumenti pre pretrage

58

58

Obrada tokova podataka

- Obrada tokova podataka - vreme
  - paketna obrada podataka i vreme
    - obrađuje velike količine **istorijskih podataka** relativno brzo (u odnosu da celokupan vremenski interval)
    - vreme se posmatra isključivo preko vremenske oznake podatka
      - omogućava **determinističku obradu**
    - sistemsko vreme ne igra nikakvu ulogu u obradi podataka
      - jer nije relevantno za događaje koji su se već odigrali u nekom trenutku u vremenu
  - obrada tokova podataka i vreme
    - često se koristi sistemsko vreme kako bi se odredio fiksni vremenski interval obrade
      - jednostavno upravljanje obradom
      - ima smisla samo ukoliko je vreme između generisanja i obrade događaja zanemarljivo kratko
        - **kašnjenje između generisanja i obrade događaja dovodi do loše obrade**

59

59

Obrada tokova podataka

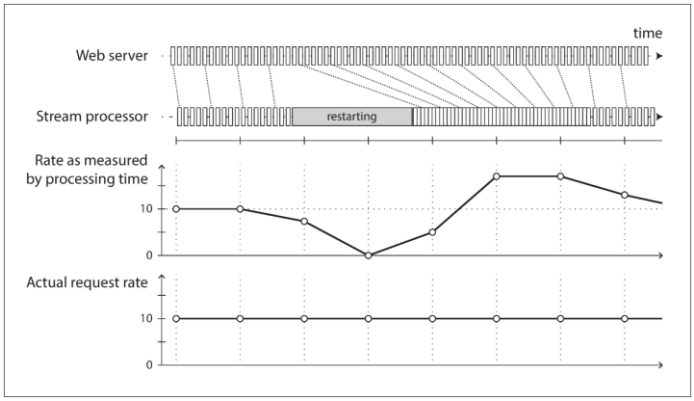
- Obrada tokova podataka - vreme
  - **vreme događaja** - vremenski trenutak generisanja događaja
  - **vreme obrade** - vremenski trenutak kada je događaj obrađen
  - **kašnjenje obrade** - vreme proteklo između generisanja događaja i njegove obrade
    - uzroci mogu biti i hardverske i softverske prirode
    - može dovesti do **obrade pogrešnih podataka i obrade događaja u pogrešnom redosledu**

60

60

Obrada tokova podataka

- Obrada tokova podataka - vreme
  - problem kada se u obzir uzima **vreme obrade** (primer: broj zahteva ka web serveru)



61

izvor: Martin Kleppmann: *Designing data-intensive applications, O'Reilly*

61

Obrada tokova podataka

- Obrada tokova podataka - vreme
  - problem kada se u obzir uzima **vreme događaja**
    - kako da budemo sigurni da smo obradili sve događaje u nekom fiksnom vremenskom intervalu
      - moguće je da su događaju ostali u nekom *buffer*-u
    - dve mogućnosti
      - ignorisati događaje koji pristignu nakon što je interval proglašen kompletnim
        - moguće je pratiti broj ignorisanih događaja i dodati ih kao relevantnu metriku
      - ponoviti obradu i uzeti u razmatranje naknadno pristigle događaje
        - objaviti ispravku rezultata i poništiti prethodni rezultat
  - problem različitih satova kada se u obzir uzima vreme događaja
    - različiti generatori mogu generisati događaje označene po različitim satovima
      - mogućnost namerne ili slučajne greške u podešavanju vremena

62

62

Obrada tokova podataka

- Obrada tokova podataka - vreme
  - vreme događaja je relevantnije sa stanovišta označavanja događaja
    - događaji mogu biti označeni baš u trenutku nastajanja
      - npr. korisničke interakcije sa mobilnim uređajem
  - problem različitih satova
    - serversko označavanje događaja po prispeću
      - tačnije sa stanovišta podešavanja sata
      - manje relevantno jer je moguće kašnjenje između nastanka događaja i njegovog obuhvata od strane sistema
        - npr. mobilni uređaj bio *offline* kada se interakcija sa korisnikom dogodila

63

63

Obrada tokova podataka

- Obrada tokova podataka - vreme
  - problem različitih satova
    - nije uočljiv samo kod obrade u realnom vremenu već i kod paketne obrade
      - samo je u obradi u realnom vremenu bitniji usled velike važnosti vremenske dimenzije
    - rešenje je **označavanje događaja sa tri vremenske odrednice**
      - **vreme kada se događaj odigrao** (po satu uređaja)
      - **vreme kada je događaj poslat serveru** (po satu uređaja)
      - **vreme kada je događaj primljen od strane servera** (po satu servera/brokera)
    - ***offset*** između sata uređaja i servera može biti sračunat oduzimanjem druge vremenske odrednice od treće
      - pod uslovom da je mrežno kašnjenje zanemarljivo kao i da je sat na uređaju ostao nepromenjen između dva računanja vremena

64

64

Obrada tokova podataka

- Obrada tokova podataka - vreme
  - određivanje vremenskog okvira
    - potrebnog za agregaciju podataka i izračunavanje
    - moguće nakon što je vremenska odrednica događaja poznata
    - postoji nekoliko tipova vremenskih okvira
      - **nepreklapajući vremenski okvir fiksne dužine** (engl. *tumbling window*)
      - **preklapajući vremenski okvir fiksne dužine** (engl. *hopping window*)
      - **relativni vremenski okvir** (engl. *sliding window*)
      - **vremenski okvir na nivou sesije** (engl. *session window*)

65

65

Obrada tokova podataka

- Obrada tokova podataka - vreme
  - određivanje vremenskog okvira
    - **nepreklapajući vremenski okvir fiksne dužine** (engl. *tumbling window*)
      - svaki događaj pripada tačno jednom okviru
      - npr. okvir trajanja 1 minut
        - npr. svi događaji od 10:03:00 do 10:03:59
        - pripadnost okviru se može dobiti zaokruživanjem vremenske odrednice događaja na prvi manji celi minut
    - **preklapajući vremenski okvir fiksne dužine** (engl. *hopping window*)
      - fiksne dužine ali sa preklapanjima
      - npr. okvir od 5 minuta sa dozvoljenim preklapanjem od 4 minuta
        - prozor 1: svi događaji od 10:03:00 do 10:07:59
        - prozor 2: svi događaji od 10:04:00 do 10:08:59
        - moguće izračunati nepreklapajuće okvire na nivou minute i potom agregirati susedne okvire

66

66

Obrada tokova podataka

- Obrada tokova podataka - vreme
  - određivanje vremenskog okvira
    - **relativni vremenski okvir** (engl. *sliding window*)
      - sadrži sve događaje koji su se dogodili u nekom vremenskom razmaku
      - moguće je implementirati ga pomoću *buffer*-a
      - npr. okvir u trajanju od 5 minuta
        - obuhvatio bi sve događaje od 10:03:39 do 10:08:39
    - **vremenski okvir na nivou sesije** (engl. *session window*)
      - nema predefinisano ili fiksno trajanje
      - služi za obradu događaja nekog korisnika koji su se dogodili u bliskim vremenskim trenucima
      - okvir se zatvara kada je korisnik neaktivan predefinisani period vremena

67

67

Obrada tokova podataka

- Obrada tokova podataka - spoj tokova podataka
  - postoji potreba za spajanjem tokova podataka kako bi se dobili potpuniji i prošireni podaci
  - problem predstavlja neograničenost tokova podataka
    - kod koje nisu svi događaji poznati u trenutku spajanja
  - tipovi spajanja tokova podataka
    - **spojevi tokova podataka** (engl. *stream-stream joins*)
    - **spojevi tokova podataka i tabela** (engl. *stream-table joins*)

68

68

Obrada tokova podataka

- Obrada tokova podataka - spoj tokova podataka
  - **spojevi tokova podataka** (engl. *stream-stream joins*)
    - obrađivač poseduje stanje
      - npr. indeksna struktura događaja u predefinisanom vremenskom intervalu
    - za svaki događaj u jednom obrađivaču
      - događaj se dodaje u indeks tog obrađivača
      - vrši se provera u indeksu drugog obrađivača za događajima koji imaju istu vrednost obeležja po kojem se vrši spajanje
      - ukoliko postoji poklapanje, generiše se novi događaj koji objedinjuje dva osnovna događaja
      - u slučaju da za događaj posle predefinisanog vremena ne postoji odgovarajući događaj u drugom toku sa kojim je moguće spojiti događaj se briše iz indeksa
    - primer: praćenje linkova koji predstavljaju rezultat pretrage (engl. *click-through rate*)
      - potrebno obraditi i akcije pretrage i akcije odabira linka u rezultatu pretrage kako bi se održao kvalitet analize

69

69

Obrada tokova podataka

- Obrada tokova podataka - spoj tokova podataka
  - **spojevi tokova podataka i tabela** (engl. *stream-table joins*)
    - za svaki događaj pronalazi se torka u odgovarajućoj tabeli baze podataka
      - koja sadrži detalje u vezi sa nekim obeležjem samog događaja
        - npr. za ID korisnika pronalazimo detalje događaja
      - problem: upiti nad udaljenom bazom podataka mogu usporiti obradu toka
        - rešenje: **kopirati bazu podataka na čvor u klasteru na kojem se nalazi obrađivač toka podataka**
          - koju je potrebno održavati kako se original menja
          - makar kreirati repliku za čitanje u tom čvoru (engl. *read replica*)
    - primer: obogaćivanje događaja statičkim podacima iz baza podataka
      - npr. za svaki događaj u kojem imamo samo ID korisnika, dodaju se svi podaci o korisniku

70

70

Obrada tokova podataka

- Obrada tokova podataka - otpornost na greške
  - pojava grešaka kod paketne obrade se rešava ponovnim izvršenjem zadatka
    - moguće usled **konačnosti i nepromenljivosti** skupa podataka
  - komplikovano obezbediti otpornost na greške prilikom obrade tokova podataka
    - usled beskonačnosti tokova
  - jedno rešenje: podeliti tok podataka u manje blokove i obradu svakog bloka tretirati kao paketnu obradu
    - **mikropaketna obrada** (engl. *microbatching*)
    - paket tipično obuhvata podatke u okviru od jedne sekunde
    - implicitno pruža nepreklapajući vremenski okvir fiksne dužine od jedne sekunde
  - drugo rešenje: periodično snimiti rezultate obrade u trajno skladište
    - **obrada sa snimanjem stanja** (engl. *checkpointing*)
    - ne definiše vremenski okvir za obradu tokova fiksne dužine
      - uslovljena je granicama u samim podacima a ne predefinisanim trajanjem okvira
  - ova dva rešenja nisu dovoljna za obezbeđenje **obrade tačno jednom**

71

71

Obrada tokova podataka

- Obrada tokova podataka - otpornost na greške
  - obezbediti **obradu** svakog događaja **tačno jednom**
    - potrebno uvesti pojam **transakcije**
      - generisanje i obrada novog događaja isključivo ako je obrada izvornog događaja bila uspešna
      - nije podržana od strane alata Kafka
    - potrebno da operacije budu **idempotentne**
      - višestruko izvršavanje operacija mora da ima isti efekat na sistem kao kada izvršimo operaciju samo jednom
        - većinu neidempotentnih operacija je moguće načiniti idempotentnima dodavanjem pojedinih meta-podataka
    - potrebna **ponovna izgradnja stanja** kod obrađivača koji održavaju stanje
      - moguća replikacija ukoliko postoji skladište podataka sa stanjem
      - moguće ponovo izgraditi stanje iz ulaznih tokova
        - ukoliko su okviri obrade kratki i događaji se čuvaju duži vremenski period

72

72

Apache Storm

73

Apache Storm

- Apache Storm
  - distribuirani sistem koji omogućava obradu velikih skupova podataka u realnom vremenu
    - optimizovan za obradu tokova podataka
    - moguće ga je horizontalno skalirati
    - ne čuva stanje između dve obrade
  - često se koristi u kombinaciji sa Apache ZooKeeper-om
  - garantuje da će svaki podatak biti **obrađen bar jednom**

74

74

Apache Storm

Apache Storm	Apache Hadoop / Map Reduce
obrada u realnom vremenu	paketna obrada
čvorovi ne čuvaju stanje (*engl. stateless*)	čvorovi čuvaju stanje (*engl. stateful*)
arhitektura u kojoj postoje glavni (*nimbus*) i podređeni čvorovi (*supervizori*), organizacija pomoću *Zookeeper*-a	arhitektura u kojoj postoje glavni (*job tracker*) i podređeni čvorovi (*task tracker*)
obrada tokova podataka u sekundama	obrada podataka u minutima i satima
program se izvršava dok ne bude zaustavljen od strane korisnika ili ne dođe do otkaza	programi sa konačnim vremenom izvršavanja
distribuirani sistemi koji tolerišu parcijalne otkaze	
otkaz čvorova zahteva restart sistema koji dozvoljava nastavak obrade	otkaz glavnog čvora zahteva ponovno pokretanje obrade od početka

75

75

Apache Storm

- Apache Storm - prednosti
  - otvorenog kôda, robustan i lak za korišćenje
  - otporan na greške, fleksibilan, pouzdan
  - podržava veliki broj programskih jezika
  - visok nivo skaliranja
  - garantuje procesiranje poruke čak i u slučaju otkaza pojedinih čvorova u klasteru

76

76

Apache Storm

- Apache Storm - osnovni koncepti
  - **torka** (engl. *tuple*)
  - **tok podataka** (engl. *stream*)
  - **izvor toka podataka** (engl. *spout*)
  - **obrađivač podataka** (engl. *bolt*)
  - **topologija** (engl. *topology*)
  - **zadatak** (engl. *task*)
  - **grupisanje tokova podataka** (engl. *stream grouping*)

77

77

Apache Storm

- Apache Storm - osnovni koncepti
  - **torka** (engl. *tuple*)
    - osnovna struktura podataka u Storm-u
    - uređena kolekcija vrednosti
      - dinamički tipizovana
        - ugrađena podrška za veliki broj tipova
        - moguće implementirati proizvoljan serijalizator
      - uobičajeno kolekcija vrednosti odvojenih zapetom
  - **tok podataka** (engl. *stream*)
    - neuređena i neograničena sekvenca torki

78

78

Apache Storm

- Apache Storm - osnovni koncepti
  - **izvor toka podataka** (engl. *spout*)
    - omogućava obuhvat podataka iz različitih izvora
      - strukturiranih i nestrukturiranih
      - npr. *Twitter* API, *Apache Kafka*, baza podataka, ručno napisan generator torki itd.
    - radi praćenja torki, izvor može da doda ID svakoj poruci
  - **obrađivač podataka** (engl. *bolt*)
    - predstavljaju logičke jedinice obrade tokova podataka
    - mogu izvršiti operacije filtriranja, agregacije, spajanja i transformacije tokova podataka
    - prima podatke i generiše nove podatke u vidu novog toka podataka
      - poseduje **ulazni i izlazni tok podataka**

79

79

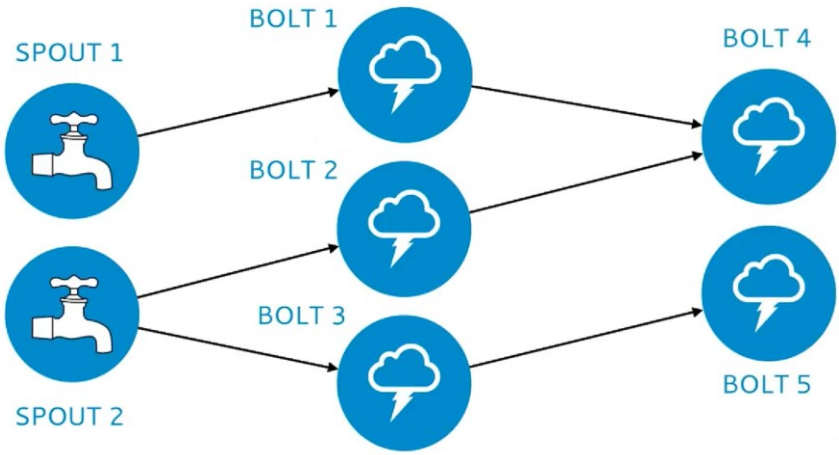
Apache Storm

- Apache Storm - osnovni koncepti
  - **topologija** (engl. *topology*)
    - sekvenca izvora i obrađivača, povezanih zajedno u cilju analize i obrade podataka u nekom domenu
      - analogno *MapReduce* poslu
    - predstavlja **usmereni aciklični graf** (engl. *directed acyclic graph*, DAG)
      - kod kojeg čvorovi predstavljaju obrade a ivice tokove podataka
    - *Apache Storm* neprekidno izvršava topologiju dok korisnik ne prekine njeno izvršavanje ili ne dođe do greške u izvršavanju od koje se nije moguće automatski oporaviti
      - moguće je da u okviru *Apache Storm* klastera postoji više pokrenutih topologija

80

80

Apache Storm

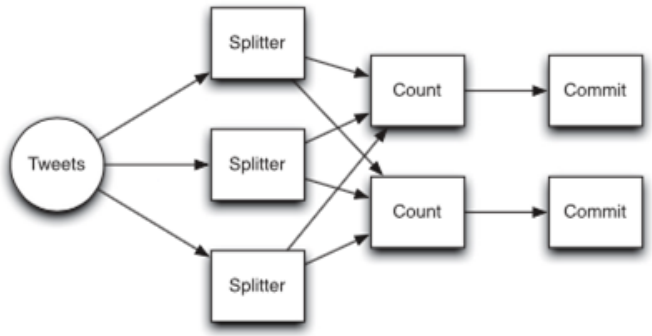


81

81

Apache Storm

- Fizička arhitektura topologije za brojanje reči u tvitovima



82

izvor: Apache Storm, Tutorial Point: https://www.tutorialspoint.com/apache_storm/apache_storm_tutorial.pdf

82

Apache Storm

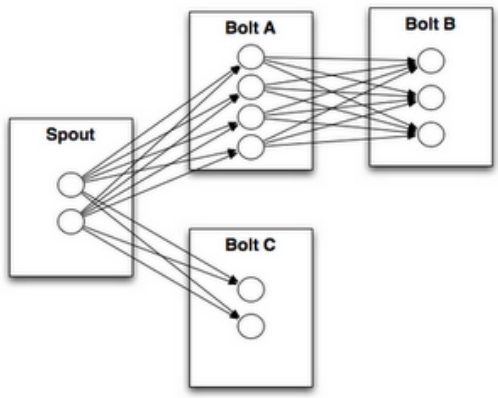
- Apache Storm - osnovni koncepti
  - **zadatak** (engl. *task*)
    - instanca izvora ili obrađivača podataka
      - koja se izvršava u okviru pokrenute topologije *Apache Storm*-a
    - u okviru pokrenute topologije, moguće je imati više zadataka istog elementa topologije
      - u cilju što bolje i efikasnije obrade i obuhvata podataka
    - zadaci se izvršavaju u okviru radnih procesa (engl. *worker process*) u *Storm* klasteru
      - svaki proces predstavlja posebnu JVM i izvršava zadatke u topologiji
      - *Apache Storm* raspoređuje zadatke ravnomerno po radnim procesima
      - svaki proces ima zadatak da pokrene i zaustavi izvršavanje zadataka po zahtevu *Apache Storm*-a

83

83

Apache Storm

- Apache Storm - osnovni koncepti
  - **grupisanje tokova podataka** (engl. *stream grouping*)
    - određuje koji tokovi podataka su obrađeni od strane kojih obrađivača
    - specificira rutiranje torki u okviru topologije
      - jedna od najbitnijih stvari za odrediti prilikom projektovanja
    - zadaci u okviru obrađivača mogu raditi nad podskupovima torki unutar toka podataka



84

izvor: *Apache Storm stream grouping:* http://www.corejavaguru.com/bigdata/storm/stream-groupings

84

Apache Storm

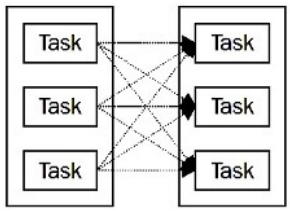
- Apache Storm - osnovni koncepti
  - grupisanje tokova podataka - vrste
    - **slučajno grupisanje** (engl. *shuffle grouping*)
    - **automatski određeno grupisanje** (engl. *none grouping*)
    - **grupisanje na osnovu vrednosti obeležja** (engl. *fields grouping*)
    - **balansirano grupisanje na osnovu vrednosti obeležja** (engl. *partial key grouping*)
    - **grupisanje celog toka** (engl. *global grouping*)
    - **multiplikacija toka** (engl. *all grouping*)
    - **direktno grupisanje** (engl. *direct grouping*)
    - **lokalno grupisanje** (engl. *local or shuffle grouping*)
    - **korisnički definisano grupisanje** (engl. *custom grouping*)

85

85

Apache Storm

- Apache Storm - osnovni koncepti
  - grupisanje tokova podataka - vrste
    - **slučajno grupisanje** (engl. *shuffle grouping*)
      - najčešće korišćeno grupisanje
      - uniformno raspoređuje torke na slučajan način po zadacima unutar čvorova za obradu
      - koristi se kada ne postoji potreba za particionisanjem koje je vođeno karakteristikama podataka
        - svi zadaci obrađuju jednak broj torki
    - **automatski određeno grupisanje** (engl. *none grouping*)
      - kada korisnika ne interesuje način grupisanja
      - trenutna implementacija se svodi na slučajno grupisanje
        - u budućnosti će zadaci sa ovim grupisanjem biti dinamički raspoređeni na optimalan način



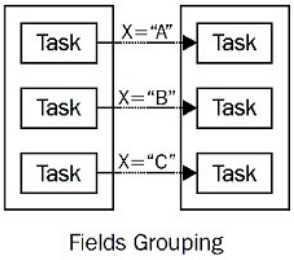
Shuffle Grouping

86

86

Apache Storm

- Apache Storm - osnovni koncepti
  - grupisanje tokova podataka - vrste
    - **grupisanje na osnovu vrednosti obeležja** (engl. *fields grouping*)
      - omogućava kontrolu slanja torki obrađivačima
      - garantuje da će torke koje imaju istu vrednost nekog obeležja biti distribuirane istom obrađivaču
      - npr. jedan obrađivač za sve tvitove istog korisnika, spajanje tokova podataka



Fields Grouping

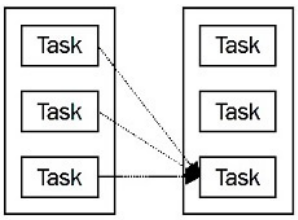
    - **balansirano grupisanje na osnovu vrednosti obeležja** (engl. *partial key grouping*)
      - slično kao prethodno grupisanje
      - torke se prosleđuju obrađivačima korišćenjem algoritama za izjednačavanje opterećenja

87

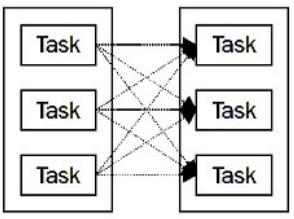
87

Apache Storm

- Apache Storm - osnovni koncepti
  - grupisanje tokova podataka - vrste
    - **grupisanje celog toka** (engl. *global grouping*)
      - sve torke proizvedene od strane svih zadataka izvornog obrađivača ili izvora prosleđuju se jednom zadatku ciljnog obrađivača
      - npr. kombinovanje rezultata obrade u topologiji
    - **multiplikacija toka** (engl. *all grouping*)
      - ne particioniše tokove podataka već šalje kopiju svake torke svim zadacima obrađivača
      - najčešće se koristi za slanje signala ili parametara obrađivačima



Global Grouping



All Grouping

88

88

Apache Storm

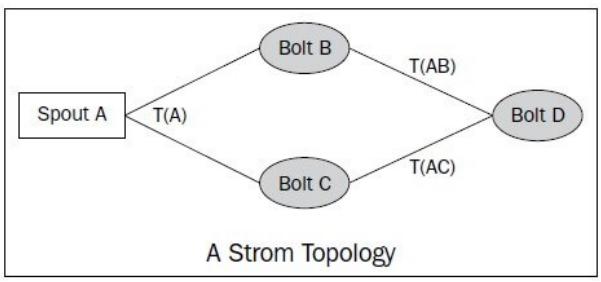
- Apache Storm - osnovni koncepti
  - grupisanje tokova podataka - vrste
    - **direktno grupisanje** (engl. *direct grouping*)
      - pošiljalac torke određuje kojem obrađivaču je prosleđuje
    - **lokalno grupisanje** (engl. *local or shuffle grouping*)
      - grupiše torku prema fizičkoj udaljenosti izvora toka i odredišta
      - rutira torke unutar istog čvora unutar hijerarhije
        - ukoliko je odredište na istom fizičkom računaru
    - **korisnički definisano grupisanje** (engl. *custom grouping*)
      - grupisanje implementirano od strane korisnika

89

89

Apache Storm

- Apache Storm - osnovni koncepti
  - **garancija obrade torke**
    - *Apache Storm* garantuje da će svaka torka koja je preuzeta sa izvora biti u potpunosti obrađena
      - **potpuno obrađena torka** - je ona torka za koju su obrađene ona i sve torke nastale na osnovu nje u svim obrađivačima topologije (npr. T(A), T(AB), T(AC))



90

izvor: *Apache Storm stream grouping:* http://www.corejavaguru.com/bigdata/storm/stream-groupings

90

Apache Storm

- Apache Storm - osnovni koncepti
  - **garancija obrade torke**
    - životni ciklus torke
      - izvor torke dodeljuje svakoj torci ID i pušta je kroz jedan od tokova podataka
      - torka se šalje odgovarajućim obrađivačima
      - Apache Storm prati status svake poruke (stablo poruka)
        - ukoliko je torka u potpunosti obrađena, Storm šalje **`ack`** odgovarajućem izvoru zajedno sa ID-jem poruke
        - ukoliko je došlo do greške u obradi, Storm šalje **`fail`** odgovarajućem izvoru zajedno sa ID-jem poruke

91

91

Apache Storm

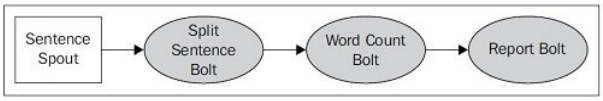
- Apache Storm - osnovni koncepti
  - **garancija obrade torke**
    - neophodni preduslovi za garanciju obrade torke
      - svaka torka koja dolazi iz nekog izvora mora imati jedinstveni identifikator
      - prilikom generisanja novih torki kao rezultat obrade neke torke, potrebno je link ka originalnoj torci poslati dalje zajedno sa novom torkom (engl. *anchoring*)
      - nakon uspešne ili neuspešne obrade obavezno je poslati signal o uspešnosti operacije
        - **ukoliko je usmereni aciklični graf torke (sa izvedenim torkama) kompletiran**, Apache Storm šalje potvrdu izvoru podataka
        - postoji pozadinski proces u Apache Storm-u koji nadgleda graf torke

92

92

Apache Storm

- Apache Storm - primer brojanja reči
  - problem: potrebno je u neprekidnom toku podataka brojati koliko se puta neka reč pojavljuje
  - rešenje: elementi topologije
    - jedan izvor podataka - generiše rečenice
    - tri obrađivača podataka
      - obrađivač za izdvajanje reči iz rečenice
      - obrađivač za brojanje
      - obrađivač za izveštavanje




```
# torka iz izvora podataka
{ "sentence":"my dog has fleas" }

# torke koje predstavljaju izlaz iz obrađivača 1
{ "word" : "my" }
{ "word" : "dog" }
{ "word" : "has" }
{ "word" : "fleas" }

# torka koja predstavlja izlaz iz obrađivača 2
{ "word" : "dog", "count" : 5 }
```

93

93

Apache Storm

```
public class SentenceSpout extends BaseRichSpout {
  private SpoutOutputCollector collector;
  private String[] sentences = {
      "my dog has fleas",
      "i like cold beverages",
      "the dog ate my homework",
      "don't have a cow man",
      "i don't think i like fleas"
  };
  private int index = 0;
  public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declare(new Fields("sentence"));
  }
  public void open(Map config, TopologyContext context,
      SpoutOutputCollector collector) {
    this.collector = collector;
  }
  public void nextTuple() {
    this.collector.emit(new Values(sentences[index]));
    index++;
    if (index >= sentences.length) {
      index = 0;
    }
    Utils.waitForMillis(1);
  }
}
```

```
public class SplitSentenceBolt extends BaseRichBolt{
  private OutputCollector collector;
  public void prepare(Map config, TopologyContext context,
      OutputCollector collector) {
    this.collector = collector;
  }
  public void execute(Tuple tuple) {
    String sentence = tuple.getStringByField("sentence");
    String[] words = sentence.split(" ");
    for(String word : words){
      this.collector.emit(new Values(word));
    }
  }
  public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declare(new Fields("word"));
  }
}
```

94

Apache Storm

```
public class WordCountBolt extends BaseRichBolt{
  private OutputCollector collector;
  private HashMap<String, Long> counts = null;

  public void prepare(Map config, TopologyContext context,
      OutputCollector collector) {
    this.collector = collector;
    this.counts = new HashMap<String, Long>();
  }

  public void execute(Tuple tuple) {
    String word = tuple.getStringByField("word");
    Long count = this.counts.get(word);
    if(count == null){
      count = 0L;
    }
    count++;
    this.counts.put(word, count);
    this.collector.emit(new Values(word, count));
  }

  public void declareOutputFields(OutputFieldsDeclarer declarer) {
    declarer.declare(new Fields("word", "count"));
  }
}
```

```
public class ReportBolt extends BaseRichBolt {
  private HashMap<String, Long> counts = null;
  public void prepare(Map config, TopologyContext context,
      OutputCollector collector) {
    this.counts = new HashMap<String, Long>();
  }
  public void execute(Tuple tuple) {
    String word = tuple.getStringByField("word");
    Long count = tuple.getLongByField("count");
    this.counts.put(word, count);
  }

  public void declareOutputFields(OutputFieldsDeclarer declarer) {
    // this bolt does not emit anything
  }
  public void cleanup() {
    System.out.println("--- FINAL COUNTS ---");
    List<String> keys = new ArrayList<String>();
    keys.addAll(this.counts.keySet());
    Collections.sort(keys);
    for (String key : keys) {
      System.out.println(key + " : " + this.counts.get(key));
    }
    System.out.println("--------------");
  }
}
```

95

Apache Storm

```
--- FINAL COUNTS ---
a : 1426
ate : 1426
beverages : 1426
cold : 1426
cow : 1426
dog : 2852
don't : 2851
fleas : 2851
has : 1426
have : 1426
homework : 1426
i : 4276
like : 2851
man : 1426
my : 2852
the : 1426
think : 1425
--------------
```

```
public class WordCountTopology {
  private static final String SENTENCE_SPOUT_ID = "sentence-spout";
  private static final String SPLIT_BOLT_ID = "split-bolt";
  private static final String COUNT_BOLT_ID = "count-bolt";
  private static final String REPORT_BOLT_ID = "report-bolt";
  private static final String TOPOLOGY_NAME = "word-count-topology";
  public static void main(String[] args) throws Exception {
    SentenceSpout spout = new SentenceSpout();
    SplitSentenceBolt splitBolt = new SplitSentenceBolt();
    WordCountBolt countBolt = new WordCountBolt();
    ReportBolt reportBolt = new ReportBolt();

    TopologyBuilder builder = new TopologyBuilder();
      builder.setSpout(SENTENCE_SPOUT_ID, spout);
      // SentenceSpout --> SplitSentenceBolt
      builder.setBolt(SPLIT_BOLT_ID, splitBolt)
        .shuffleGrouping(SENTENCE_SPOUT_ID);
      // SplitSentenceBolt --> WordCountBolt
      builder.setBolt(COUNT_BOLT_ID, countBolt)
        .fieldsGrouping(SPLIT_BOLT_ID, new Fields("word"));
      // WordCountBolt --> ReportBolt
      builder.setBolt(REPORT_BOLT_ID, reportBolt)
        .globalGrouping(COUNT_BOLT_ID);

    Config config = new Config();
    LocalCluster cluster = new LocalCluster();
    cluster.submitTopology(TOPOLOGY_NAME, config, builder.
        createTopology());

   waitForSeconds(10);
    cluster.killTopology(TOPOLOGY_NAME);
    cluster.shutdown();
  }
}
```

96

96

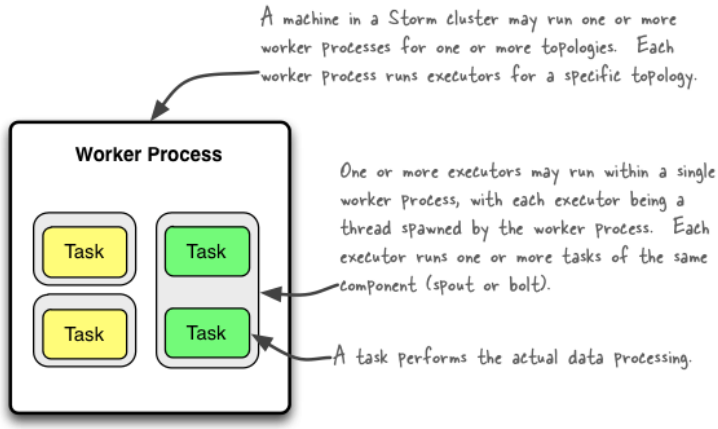
Apache Storm

- Apache Storm - elementi klastera
  - propusna moć topologije je srazmerna broju instanci (zadataka) svakog od elemenata topologije koji se izvršavaju paralelno
  - osnovne komponente izvršenja topologije
    - **radni procesi** (engl. *worker processes*)
      - JVM koje se izvršavaju na čvorovima u klasteru
      - svaki čvor je konfigurisan tako da omogući izvršavanje jednog ili više radnih procesa Storm topologije
    - **izvršna nit** (engl. *executor threads*)
      - Java niti koje se izvršavaju unutar radnih procesa
      - moguća dodela više zadataka istoj niti
        - podrazumevano jedna nit -> jedan zadatak
    - **zadatak** (engl. *task*)
      - instance izvora ili obrađivača podataka

97

97

Apache Storm



98

izvor: *Apache Storm topology parallelism :* http://www.corejavaguru.com/bigdata/storm/topology-parallelism

98

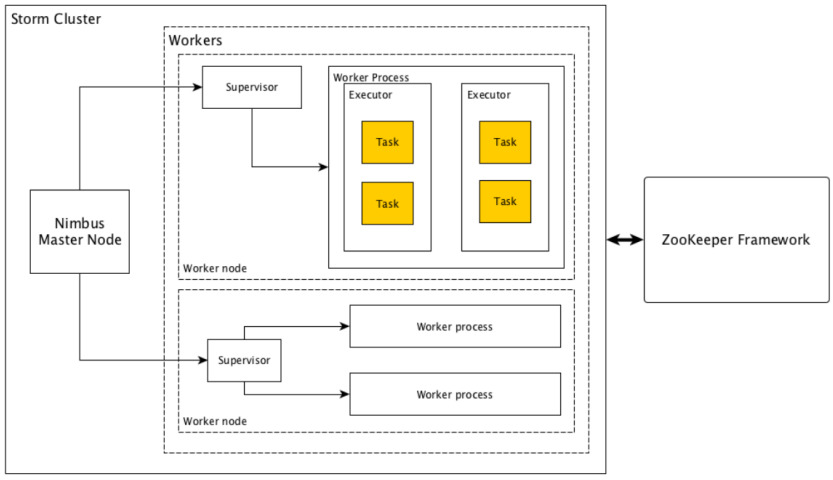
Apache Storm

- Apache Storm - elementi klastera
  - **Nimbus**
    - glavni čvor i centralna komponenta Storm klastera
      - svi ostali čvorovi se zovu radni čvorovi
    - osnovni zadatak je da izvršava topologiju
      - odgovoran je za distribuciju podataka radnim čvorovima
      - analizira topologiju, izdvaja zadatke koje je potrebno izvršiti i šalje ih supervizoru
        - Apache Storm koristi interni distribuirani sistem za razmenu poruka za ovu komunikaciju
      - detektuje greške u obradi podataka i otkaze čvorova u klasteru
  - **Supervizor**
    - nadgleda jedan ili više radnih procesa kojima delegira zadatke na izvršenje
      - radni procesi će kreirati koliko god je potrebno izvršnih niti

99

99

Apache Storm



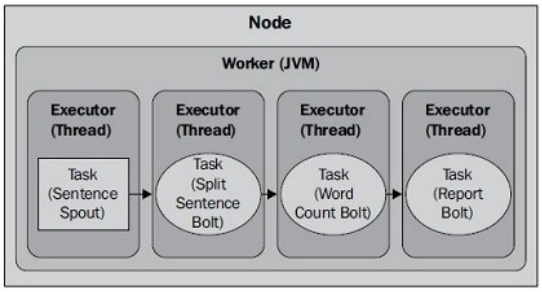
100

izvor: Apache Storm, Tutorial Point: https://www.tutorialspoint.com/apache_storm/apache_storm_tutorial.pdf

100

Apache Storm

- Apache Storm - paralelizam
  - podrazumevana vrednost za većinu konfiguracionih parametara je 1
    - jedini paralelizam je na nivou izvršnih niti



101

izvor: *Apache Storm topology parallelism* : http://www.corejavaguru.com/bigdata/storm/topology-parallelism

101

Apache Storm

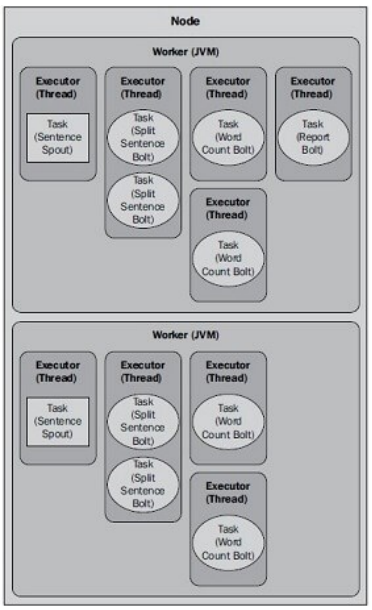
- Apache Storm - paralelizam
  - **konfiguracija**
    - Storm dozvoljava podešavanje globalnih parametara paralelizacije u okviru konfiguracionih parametara
    - elementi topologije se podešavaju prilikom kreiranja

```
Config config = new Config();
config.setNumWorkers(2);

builder.setSpout(SENTENCE_SPOUT_ID, spout, 2);

builder.setBolt(SPLIT_BOLT_ID, splitBolt, 2)
.setNumTasks(4)
.shuffleGrouping(SENTENCE_SPOUT_ID);

builder.setBolt(COUNT_BOLT_ID, countBolt, 4)
.fieldsGrouping(SPLIT_BOLT_ID, new Fields("word"));
```



102

izvor: *Apache Storm topology parallelism* : http://www.corejavaguru.com/bigdata/storm/topology-parallelism

102

102

Apache Storm

- Apache Storm - paralelizam
  - **rebalansiranje**
    - Storm dozvoljava da se parametri paralelizacije menjaju u toku izvršavanja topologije
    - kroz komandnu liniju

```
## Reconfigure the topology "mytopology" to use 5 worker processes,
## the spout "blue-spout" to use 3 executors and
## the bolt "yellow-bolt" to use 10 executors.

$ storm rebalance mytopology -n 5 -e blue-spout=3 -e yellow-bolt=10
```

103

103

Apache Storm

- Apache Storm - topologija Trident
  - apstrakcija nad Apache Storm primitivama
  - omogućava **obradu tačno jednom**
  - uvodi
    - **pojam paketne obrade torki u tokovima podataka**
    - **pojam stanja**
    - **pojam transakcione obrade**
  - usložnjava osnovnu arhitekturu
    - slabije performanse u odnosu na ručno pisan kôd Storm arhitekture
      - usled generisanja neoptimalnog kôda

104

104

Apache Storm

- Apache Storm - topologija Trident
  - pojam paketne obrade torki u tokovima podataka
    - dolazne torke se grupišu u paketima i na taj način obrađuju
    - paketi mogu imati hiljade i milione torki
      - u zavisnosti od količine podataka koja stiže u sistem
    - dozvoljavaju se agregacije nad i između paketa
      - stanje se čuva u memoriji
    - omogućava bolje performanse sistema kada se konačno izračunato stanje čuva u bazama podataka
      - umesto pojedinačnih ažuriranja, ulazni podaci se posmatraju u paketima
        - smanjuje broj ažuriranja baze podataka

105

105

Apache Storm

- Apache Storm - topologija Trident
  - pojam stanja
    - obrada podataka bez čuvanja stanja (Storm model)
      - ne čuva se podatak o obradi neke torke (jedinstveno identifikovane ID-jem poruke)
    - obrada podataka s čuvanjem stanja (Trident model)
      - čuva se zapis o svakoj obrađenoj torci
      - Trident automatski podržava čuvanje stanja o obradi poruka
        - u memoriji ili u skladištu podataka

106

106

Apache Storm

- Apache Storm - topologija Trident
  - pojam transakcione obrade
    - svakom paketu koji je osnovna jedinica obrade u Trident-u dodeljuje se jedinstveni identifikator transakcije
      - u slučaju ponovnog slanja paketa na obradu, on dobija isti ID transakcije
    - ažuriranje stanja nakon obrade paketa je isključivo moguće u redosledu paketa
    - ukoliko transakciji ne dodelimo ID i ne želimo da ga čuvamo u skladištu podataka Trindent obrada se svodi na model najmanje jednom

107

107

Apache Storm

- Apache Storm - topologija Trident
  - primer prebrojavanja pojava neke reči u ulaznom toku podataka
    - napraviti sistem koji u realnom vremenu broji reči u neograničenom toku podataka
    - napraviti upite koji vraćaju broj pojavljivanja reči za neku listu reči

```
#ulazni tok podataka
FixedBatchSpout spout = new FixedBatchSpout(new Fields("sentence"), 3,
  new Values("the cow jumped over the moon"),
  new Values("the man went to the store and bought some candy"),
  new Values("four score and seven years ago"),
  new Values("how many apples can you eat"));
spout.setCycle(true);

#rešenje zadataka
TridentTopology topology = new TridentTopology();
TridentState wordCounts = topology.newStream("spout1", spout)
    .each(new Fields("sentence"), new Split(), new Fields("word"))
    .groupBy(new Fields("word"))
    .persistentAggregate(new MemoryMapState.Factory(), new Count(), new Fields("count"))
    .parallelismHint(6);
```

108

108

Apache Storm

- Apache Storm - topologija Trident
  - primer prebrojavanja pojava neke reči u ulaznom toku podataka

```
# Split funkcija
public class Split extends BaseFunction {
   public void execute(TridentTuple tuple, TridentCollector collector) {
       String sentence = tuple.getString(0);
       for(String word: sentence.split(" ")) {
           collector.emit(new Values(word));
       }
   }
}

######
#moguće je smestiti stanje u nekoj bazi podataka
wordCounts.persistentAggregate(
       MemcachedState.transactional(serverLocations), new Count(), new Fields("count")
);
```

109

Apache Storm

- Apache Storm - topologija Trident
  - primer prebrojavanja pojava neke reči u ulaznom toku podataka

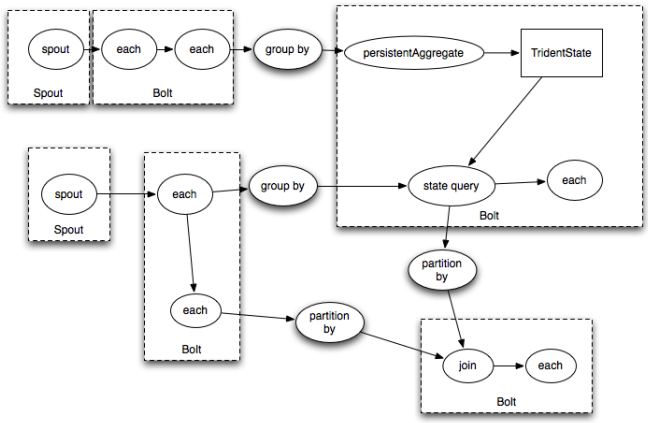
```
#implementacija distribuiranog upita nad sistemom
topology.newDRPCStream("words")
  .each(new Fields("args"), new Split(), new Fields("word"))
  .groupBy(new Fields("word"))
  .stateQuery(wordCounts, new Fields("word"), new MapGet(), new Fields("count"))
  .each(new Fields("count"), new FilterNull())
  .aggregate(new Fields("count"), new Sum(), new Fields("sum"));

#upit nad sistemom
DRPCClient client = new DRPCClient("drpc.server.location", 3772);
System.out.println(client.execute("words", "cat dog the man");
// prints the JSON-encoded result, e.g.: "[[5078]]"
```

110

Apache Storm

- Apache Storm - topologija Trident
  - izvršenje topologija Trident

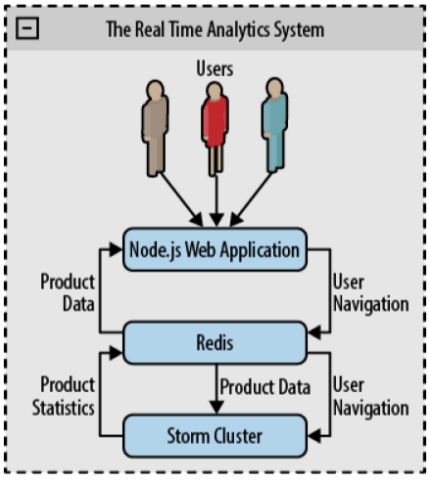


111

111

Apache Storm

- Apache Storm - primer analitike na web aplikaciji
  - problem: pratiti u realnom vremenu statistiku korišćenja web aplikacije
    - prateći korisničku navigaciju po aplikaciji
    - **koje sve kategorije je pogledao korisnik koji je pogledao neki proizvod**
  - jedno rešenje: implementirati Storm topologiju uz pomoć baze podataka Redis
    - koja će služiti i kao izvor podataka i kao skladište za međurezultate



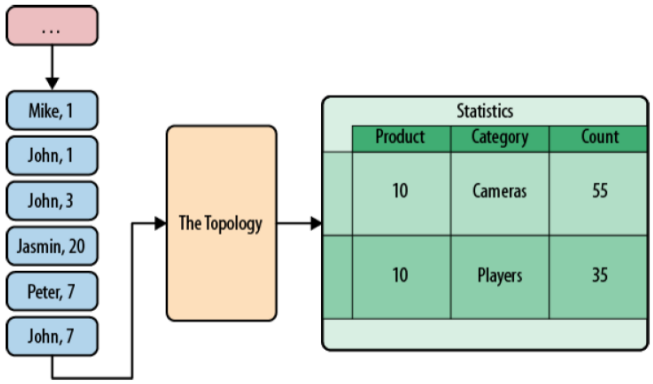
112

izvor: Apache Storm, Tutorial Point: https://www.tutorialspoint.com/apache_storm/apache_storm_tutorial.pdf

112

Apache Storm

- Apache Storm - primer analitike na web aplikaciji
  - ulazi i izlazi iz topologije



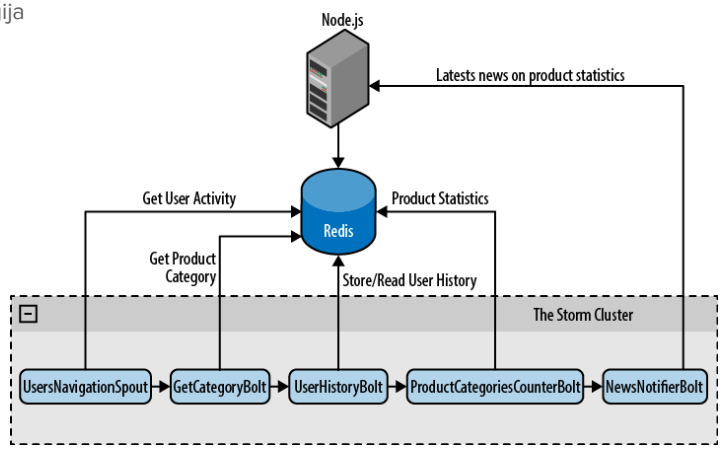
113

izvor: Apache Storm, Tutorial Point: https://www.tutorialspoint.com/apache_storm/apache_storm_tutorial.pdf

113

Apache Storm

- Apache Storm - primer analitike na web aplikaciji
  - topologija



114

izvor: Apache Storm, Tutorial Point: https://www.tutorialspoint.com/apache_storm/apache_storm_tutorial.pdf

114

Apache Storm

- Apache Storm - primer analitike na web aplikaciji
  - navigacija se skladišti u bazi podataka Redis
    - za svakog korisnika koji je link ispratio, kog tipa je ta stranica, koji se proizvod nalazi na tom linku, koja je kategorija proizvoda itd.
  - `UsersNavigationSpout`
    - preuzima podatke iz Redis-a
    - generiše torku sa sledećim vrednostima
      - korisnik koji je izvršio navigaciju
      - tip stranice na koju je korisnik izvršio navigaciju
      - mapa elemenata specifičnih za konkretan tip stranice
        - npr. za tip stranice PRODUCT, sadrži ID proizvoda koji je pretraživan

Apache Storm

- Apache Storm - primer analitike na web aplikaciji

```
redis 127.0.0.1:6379> llen navigation
(integer) 5
redis 127.0.0.1:6379> lrange navigation 0 4
1)"{\"user\":\"59c34159-0ecb-4ef3-a56b-
99150346f8d5\",\"product\":\"1\",\"type\":\"PRODUCT\
"}"
2)"{\"user\":\"59c34159-0ecb-4ef3-a56b-
99150346f8d5\",\"product\":\"1\",\"type\":\"PRODUCT\
"}"
3)"{\"user\":\"59c34159-0ecb-4ef3-a56b-
99150346f8d5\",\"product\":\"2\",\"type\":\"PRODUCT\
"}"
4)"{\"user\":\"59c34159-0ecb-4ef3-a56b-
99150346f8d5\",\"product\":\"3\",\"type\":\"PRODUCT\
"}"
5)"{\"user\":\"59c34159-0ecb-4ef3-a56b-
99150346f8d5\",\"product\":\"5\",\"type\":\"PRODUCT\
"}"
```

```
public void nextTuple() {
  String content = jedis.rpop("navigation");
  if (content == null || "nil".equals(content)) {
    try {
      Thread.sleep(300);
    } catch (InterruptedException e) {
    }
  } else {
    JSONObject obj = (JSONObject) JSONValue.parse(content);
    String user = obj.get("user").toString();
    String product = obj.get("product").toString();
    String type = obj.get("type").toString();
    HashMap<String, String> map = new HashMap<String, String>();
    map.put("product", product);
    NavigationEntry entry = new NavigationEntry(user, type, map);
    collector.emit(new Values(user, entry));
  }
}
```

Apache Storm

- Apache Storm - primer analitike na web aplikaciji
  - `GetCategoryBolt`
    - za svaku torku koja se tiče navigacije na stranicu sa proizvodom
      - iz Redis-a dobavlja podatke o proizvodu
    - generiše torku koja sadrži
      - korisnika
      - podatke o proizvodu
      - podatke o kategoriji proizvoda

117

117

Apache Storm

- Apache Storm - primer analitike na web aplikaciji

```
public void execute(Tuple input, BasicOutputCollector collector) {
  NavigationEntry entry = (NavigationEntry) input.getValue(1);
  if ("PRODUCT".equals(entry.getPageType())) {
    try {
      String product = (String) entry.getOtherData().get("product");
      // Call the items API to get item information
      Product itm = reader.readItem(product);
      if (itm == null)
        return;
      String categ = itm.getCategory();
      collector.emit(new Values(entry.getUserId(), product, categ));
    } catch (Exception ex) {
      System.err.println("Error processing PRODUCT tuple" + ex);
      ex.printStackTrace();
    }
  }
}
```

118

118

Apache Storm

- Apache Storm - primer analitike na web aplikaciji
  - `UserHistoryBolt`
    - **glavni obrađivač u sistemu**
    - **agregira podatke o svakom korisniku**
      - proizvode na koje je izvršio navigaciju
    - koristi Redis bazu kao skladište podataka
      - poseduje lokalni *cache* za svakog korisnika koji se prenosi u bazu u pozadinskom procesu i predstavlja bitnu tačku razmatranja kod paralelizacije
        - u slučaju paralelizacije posla bilo bi potrebno iskoristiti grupisanje po vrednosti obeležja ***korisnik*** za ovaj obrađivač
    - generiše torke
      - koje sadrže trenutni proizvod iz navigacije i sve kategorije koje je prethodno korisnik posetio
      - koje sadrže trenutnu kategoriju i sve prethodno posećene proizvode

119

119

Apache Storm

- Apache Storm - primer analitike na web aplikaciji
  - `UserHistoryBolt`

User	#	Category
John	0	Players
John	2	Players
John	17	TVs
John	21	Mounts

User	#	Category
John	8	Phones

#	Category
8	Players
8	Players
8	TVs
8	Mounts
0	Phones
2	Phones
17	Phones
21	Phones

120

izvor: Apache Storm, Tutorial Point: https://www.tutorialspoint.com/apache_storm/apache_storm_tutorial.pdf

120

Apache Storm

- Apache Storm - primer analitike na web aplikaciji

```
public void execute(Tuple input) {
  String user = input.getString(0);
  String prod1 = input.getString(1);
  String cat1 = input.getString(2);
  // Product key will have category information embedded.
  String prodKey = prod1 + ":" + cat1;
  Set<String> productsNavigated = getUserNavigationHistory(user);
  // If the user previously navigated this item -> ignore it
  if (!productsNavigated.contains(prodKey)) {
    // Otherwise update related items
    for (String other : productsNavigated) {
      String[] ot = other.split(":");
      String prod2 = ot[0];
      String cat2 = ot[1];
      collector.emit(new Values(prod1, cat2));
      collector.emit(new Values(prod2, cat1));
    }
    addProductToHistory(user, prodKey);
  }
}
```

121

121

Apache Storm

- Apache Storm - primer analitike na web aplikaciji
  - `ProductCategoriesCounterBolt`
    - prima parove proizvod - kategorija
    - generiše torke koje sadrže
      - proizvod
      - kategoriju
      - broj pojavljivanja konkretnog para proizvod - kategorija
    - koristi Redis bazu kao skladište podataka
      - poseduje lokalni *cache* za svakog korisnika koji se prenosi u bazu u pozadinskom procesu i predstavlja bitnu tačku razmatranja kod paralelizacije
        - u slučaju paralelizacije posla bilo bi potrebno iskoristiti grupisanje po vrednosti obeležja ***proizvod*** za ovaj obrađivač

122

122

Apache Storm

- Apache Storm - primer analitike na web aplikaciji

```
public void execute(Tuple input) {
  String product = input.getString(0);
  String categ = input.getString(1);
  int total = count(product, categ);
  collector.emit(new Values(product, categ, total));
}

private int count(String product, String categ) {
  int count = getProductCategoryCount(categ, product);
  count++;
  storeProductCategoryCount(categ, product, count);
  return count;
}
```

123

123

Apache Storm

- Apache Storm - primer analitike na web aplikaciji
  - `NewsNotifierBolt`
    - obaveštava web aplikaciju o promenama u obračunatim statističkim informacijama
      - šalje HTTP POST zahtev odgovarajućem servisu na web serveru

124

124

Apache Storm

- Apache Storm - primer analitike na web aplikaciji

```
public void execute(Tuple input) {
  String product = input.getString(0);
  String categ = input.getString(1);
  int visits = input.getInteger(2);
  String content = "{ \"product\": \"" + product + "\",
\"categ\":\"" + categ + "\",\"visits\":" + visits + " }";
  HttpPost post = new HttpPost(webserver);
  try {
    post.setEntity(new StringEntity(content));
    HttpResponse response = client.execute(post);
    org.apache.http.util.EntityUtils.consume(response.getEntity());
  } catch (Exception e) {
    e.printStackTrace();
    reconnect();
  }
}
```

125

125

Apache Storm

- Apache Storm - primer analitike na web aplikaciji
  - **Redis predstavlja usko grlo u arhitekturi**
    - moguće poboljšati stepen paralelizma uvođenjem horizontalnih replika (engl. *shards*)
    - moguće povećati otpornost na otkaze uvođenjem master-slave sistema za Redis
    - ove izmene bi zahtevale izmenu kôda i u topologiji i u web aplikaciji

126

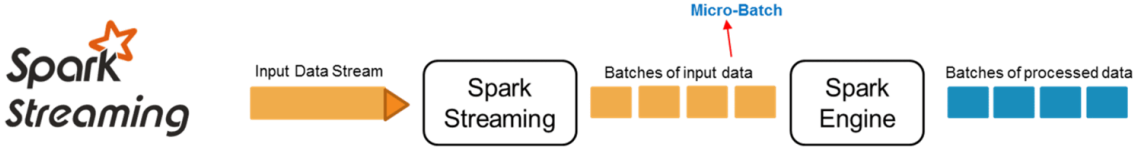
126

Apache Spark (obrada u realnom vremenu)

141

Apache Spark Streaming

- Apache Spark Streaming
  - modul alata Apache Spark koji omogućava obradu tokova podataka
    - iz velikog broja različitih izvora podataka (Kafka, Flume, HDFS, Kinesis, Twitter ...)
  - kontinuirano prihvata podatke sa ulaznog toka i **deli ih na pakete**
    - koji se dalje procesiraju radi generisanja izlaznog toka podataka
    - moguća obrada podataka upotrebom kompleksnih algoritama za obradu
      - iskazanih funkcijama visokog nivoa apstrakcije (*map, reduce, join, window*)

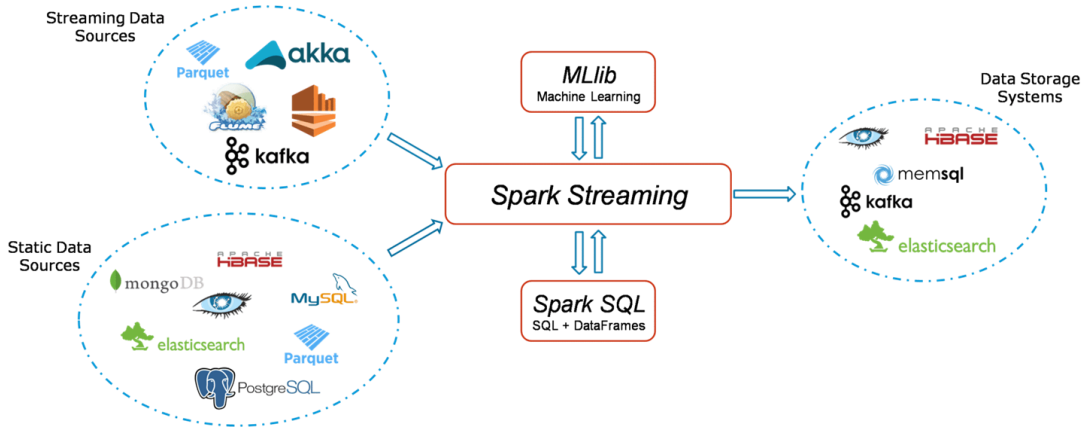


142

izvor:Sandeep Dayananda: https://www.edureka.co/blog/spark-streaming/

142

Apache Spark Streaming



izvor:Sandeep Dayananda: https://www.edureka.co/blog/spark-streaming/

143

143

Apache Spark Streaming

- Osobine
  - pogodan za skaliranje
    - lako skaliranje na stotine čvorova u klasteru
  - brzo izvršavanje
    - uslovljeno malim kašnjenjem i brzim izvršenjem obrade podataka
  - tolerantan na otkaze
    - postojanje mehanizama za oporavak od otkaza do na trenutak u vremenu
  - integracija sa ostatkom Spark ekosistema
    - integracija sa ostalim komponentama za paketnu i obradu u realnom vremenu
    - integracija sa komponentom za mašinsko učenje i analizu podataka putem SQL-a

144

144

Apache Spark Streaming - kontekst

- `StreamingContext`
  - obuhvata tok podataka nad kojim se vrši obrada
  - predstavlja ulaznu tačku programa
    - kreira konekciju sa izvorom podataka
    - kreira se na osnovu klase `SparkContext`
      - koja predstavlja vezu sa Spark klasterom
  - koristi se za kreiranje osnovnih jedinica obrade **`Dstream`**
    - moguće ga koristiti sa DataFrame *(structured streaming)*
  - omogućava fino podešavanje ponašanja obrađivača podataka
    - npr. podešava se veličina svakog paketa nad kojim se vrši obrada

Figure: *Spark Streaming Context*

Figure: *Default Implementation Sources*

izvor:Sandeep Dayananda: https://www.edureka.co/blog/spark-streaming/

145

145

Apache Spark Streaming - kontekst

- `StreamingContext`
  - postavljanje odgovarajućeg **intervala paketiranja**
    - na osnovu potrebe konkretne aplikacije, kao i na osnovu dostupnih resursa klastera
      - interval paketiranja može imati veliki uticaj na količinu podataka koju aplikacija može da obradi
    - sistem bi trebalo da obrađuje podatke onoliko brzo koliko se brzo sami podaci generišu
      - kako bi aplikacija bila stabilna
      - **vreme obrade bi trebalo da je manje od intervala paketiranja**
    - u zavisnosti od kompleksnosti operacija koje se izvršavaju

146

146

Apache Spark Streaming - DStream

- Diskretizovani tok podataka (engl. *Discretized Stream*, DStream)
  - predstavlja osnovnu apstrakciju koja pruža Apache Spark Streaming
  - kontinualni tok podataka
    - ulazni tok podataka sa izvora podataka
    - obrađeni tok podataka koji je nastao transformacijom ulaznog toka podataka
      - podržan veliki broj operacija kao i nad RDD-evima
  - interno predstavljen serijom RDD-jeva
    - svaki RDD u DStream-u sadrži podatke iz određenog intervala

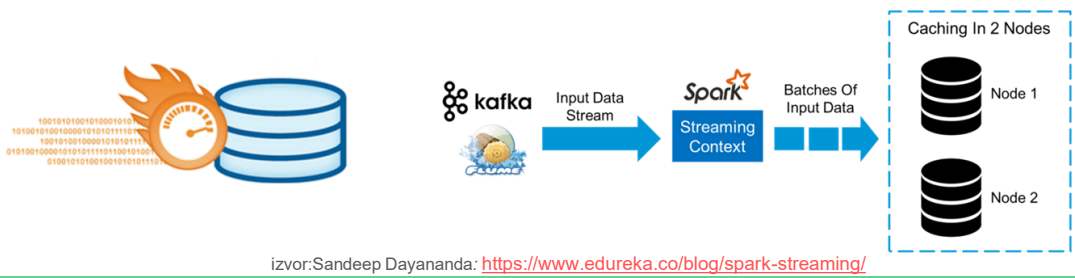
147

izvor:Apache Spark Streaming: https://spark.apache.org/docs/latest/streaming-programming-guide.html

147

Apache Spark Streaming - DStream

- Diskretizovani tok podataka
  - omogućava *cache*-iranje podataka u memoriji
    - radi lakšeg višestrukog izvršavanja operacija nad istim DStream-om
    - smešta svaki povezani RDD u memoriju
    - omogućeno upotrebom metode `persist()`
  - podrazumevana replikacija je na dva čvora u klasteru



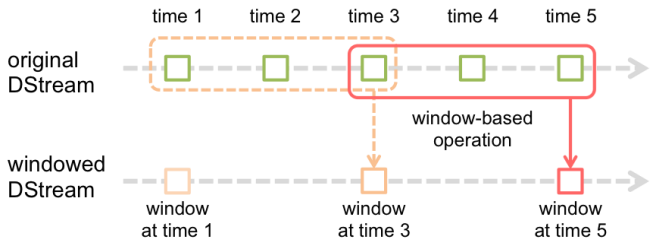
148

izvor:Sandeep Dayananda: https://www.edureka.co/blog/spark-streaming/

148

Apache Spark Streaming - transformacije

- Operacije nad podacima u vremenskom intervalu (engl. *window operations*)
  - omogućava izračunavanja nad kliznim vremenskim intervalom
  - svaki put kada se klizni interval pomeri nad izvornim DStream-om
    - izvorni RDD-evi se grupišu kako bi se kreirali RDD-evi intervalnog DStream-a.
    - npr. operacija se primenjuje na prethodne tri vremenske jedinice, a klizanje se vrši za dve vremenske jedinice



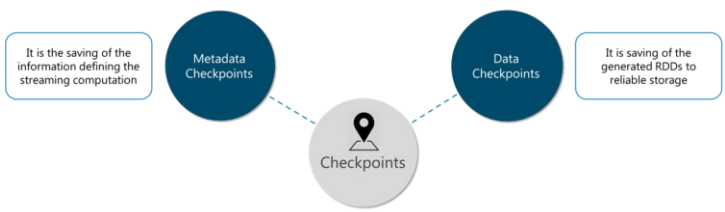
izvor:Apache Spark Streaming: https://spark.apache.org/docs/latest/streaming-programming-guide.html

151

151

Apache Spark Streaming - čuvanje stanja

- Čuvanje stanja (engl. *checkpointing*)
  - omogućava oporavak od grešaka i rad sistema 24/7
  - čuvanje stanja meta-podataka
    - čuva podešavanja, operacije nad DStream-ovima i polovično obrađene pakete u HDFS
  - čuvanje stanja podataka
    - čuva RDD-eve ukoliko su potrebni za izračunavanja u nekoliko paketa za obradu (*stateful* obrada)



izvor:Sandeep Dayananda: https://www.edureka.co/blog/spark-streaming/

153

153

Apache Spark Streaming - primer

- Primer brojanja reči u tekstu

```
// Create the context with a 1 second batch size
SparkConf sparkConf = new SparkConf().setAppName("JavaNetworkWordCount");
JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, Durations.seconds(1));

// Create a JavaReceiverInputDStream on target ip:port and count the
// words in input stream of \n delimited text (eg. generated by 'nc')
// Note that no duplication in storage level only for running locally.
// Replication necessary in distributed scenario for fault tolerance.
JavaReceiverInputDStream<String> lines = ssc.socketTextStream(
        args[0], Integer.parseInt(args[1]), StorageLevels.MEMORY_AND_DISK_SER);
JavaDStream<String> words = lines.flatMap(x -> Arrays.asList(SPACE.split(x)).iterator());
JavaPairDStream<String, Integer> wordCounts = words.mapToPair(s -> new Tuple2<>(s, 1))
    .reduceByKey((i1, i2) -> i1 + i2);

wordCounts.print();
ssc.start();
ssc.awaitTermination();
```

154

154

Literatura

- Martin Kleppmann: *Designing data-intensive applica*tions, O'Reilly
- Jonathan Leibiusky, Gabriel Eisbruch, Dario Simonassi: *Getting started with Storm*
- Tutorial Point, *Apache Storm*: https://www.tutorialspoint.com/apache_storm/apache_storm_tutorial.pdf
- Code Java Guru, *Apache Storm*: http://www.corejavaguru.com/bigdata/storm/introduction
- Online Documentation, *Apache Spark*, https://spark.apache.org/docs/latest/streaming-programming-guide.html

155

155