



Факултет техничких наука

Универзитет у Новом Саду

Рачунарски системи високих перформанси

Имплементација основних концепата Кафка алата

Аутор:
Сара Попарић

Индекс:
Е2 92/2022

15. јануар 2023.

Сажетак

Догађаји, у свету који одликују друштвене мреже, бројни сензори, као и *Internet of Things*, морају бити брзо и поуздано прихваћени, дистрибуирани и анализирани. Кафка омогућује корисницима да буду сигурни да су подаци прихваћени и прослеђени даље на сигуран начин. Она представља дистрибуирани, високо скалабилни брокер порука, направљен за размену велике количине порука између извора поруке и њеног циља. Са друге стране, *OpenMPI*, који представља отворену имплементацију *MPI (Message Passing Interface)* стандарда, који прописује комуникацију разменом порука на различитим паралелним архитектурама, погодан је за имплементацију различитих протокола за слање порука. Овај рад се бави могућношћу имплементације основних концепата Кафке уз помоћ *OpenMPI*-ја.

Садржај

1	Увод	1
2	Кафка	2
2.1	Основни концепти	2
2.2	Ток података	3
2.3	Предности	3
3	Имплементација основних концепата Кафке уз помоћ OpenMPI-ја	4
3.1	Топици	4
3.2	Партиције	5
3.3	Подаци у оквиру партиција	7
4	Закључак	10

Списак изворних кодова

1	Пример креирања глобалног комуникатора	5
2	Распоређивање партиција по топицима (комуникаторима)	6
3	Слање података	8

Списак слика

1	Последњи примљени податак по партицијама	9
---	--	---

Списак табела

1 Увод

У последњих десетак година дошло је до "експлозије" података, до чега је дошло услед тренда повећаног генерисања података, за чега су највише заслужне друштвене мреже и свеprisутнији паметни уређаји. Осим што сви ови подаци морају, углавном, да буду обрађени у што краћем временском периоду, они се неретко обрађују и чувају на местима која нису и места на којима се они генеришу.

За пренос података у виду порука постоје различите имплементације као што су: *Apache Kafka*, *RabbitMQ*, *ActiveMQ*, и други. У доста случајева се сматра да је Кафка један од најбољих брокера, али то који ће се *message broker* користити углавном зависи од случаја до случаја. Неке од главних предности Кафке су описане касније.

Овај рад се бави описом неких главних карактеристика Кафке као алата, током који подаци прођу кроз саму Кафку, као и неким предностима које Кафку издвајају у односу на друге системе за пренос порука. Касније је дато поређење концепата Кафке и концепата који постоје у *OpenMPI*-ју, као и како су ове сличности искоришћене за представљање Кафка система уз помоћ *OpenMPI*-ја.

2 Кафка

Apache Kafka је популарни *open-source streaming* систем који се користи за *stream processing*, *pipeline*-ове у реалном времену и интеграцију велике количине података. Првобитно је креирана за руковање постовима на *LinkedIn*-у, а брзо је еволуирала од реда порука (*message queue*-а) до платформе за стриминг догађаја (*event streaming platform*) која рукује милионом порука у секунди, односно трилионима порука у да-ну.

2.1 Основни концепти

Кафка представља дистрибуирану стриминг платформу која омогућује обраду и анализу података у реалном времену. Дизајнирана је тако да може да рукује са великом количином података, при том са малом латенцијом, што је чини веома погодном за широк спектар употребе као што је аналитика у реалном времену, *event sourcing*, и агрегација логова.

У основи, Кафка се заснива на *publish-subscribe* моделу. У овом моделу, произвођачи (*producer*-и) пишу податке у *topic*-е, а корисници (*consumer*-и) читају податке из *topic*-а. Ово омогућује да више *consumer*-а чита исте податке, што омогућује паралелну обраду и анализу података у реалном времену. Подаци се чувају у оквиру *topic*-а.

Једна од кључних карактеристика Кафке је способност руковања великим протоком података. Ово је могуће због њене дистрибуиране архитектуре. У Кафка кластеру постоји неколико сервера, који се називају брокери, и који чувају и управљају подацима. Брокери се могу груписати у кластере, који обезбеђују високу доступност и толеранцију на грешке (*fault tolerance*). Подаци се чувају на неколико брокера и преко њих се реплицирају. Репликацијом је омогућено да ако се један брокер "сруши" да су подаци доступни на другим брокерима. Такође, Кафка користи шему партиционисања за дистрибуцију података међу брокерима, омогућавајући на тај начин *load balancing* и хоризонтално скалирање.

Још једна кључна карактеристика Кафке је способност руковања токовима података у реалном времену. Као што је већ речено, подаци се чувају у *topic*-има, где их уписују *producer*-и, а одакле их читају *consumer*-и. Ово омогућава обраду и анализу података у реалном времену, док се генеришу, уместо да се чека да сви се подаци прикупе пре него што се крене са обрадом.

2.2 Ток података

Када *producer* упише податак у *topic*, податак се прво уписује у водећу партицију, која је одговорна за репликацију података у осталим партицијама. Водећа партиција се бира зависно од шеме партиционисања која се конфигурише за сваки *topic*, и може се одредити на основу кључа података или *round-robin* приступом.

Са друге стране, када *consumer* жели да чита податке, он се конектује на брокер и тада мође да чита податке са партиција, почевши од податка који је захтевао. *Consumer* такође може захтевати да буде обавештен о новим подацима који су уписани у *topic*.

2.3 Предности

Кафка се данас користи готово у свакој индустрији, за безброј различитих случајева коришћења. Представља де факто технологију која се користи за изградњу најновије генерације скалабилних апликација за стриминг података у реалном времену. Иако се све што Кафка нуди може постићи низом технологија које су доступне на тржишту у наставку су наведени неки од главних разлога зашто је Кафка толико популарна:

- Висока пропусност - могућност руковања великом фреквенцијом и великим обимом података. Кафка може да рукује милионима порука у секунди.
- Висока скалабилност - Кафка кластере је могуће скалирати до хиљаду брокера, трилиона порука по дану, петабајта података, стотина хиљада партиција. Нуди еластично и прошириво сладиште, као и обраду података.
- Мала латенција - Кафка може да испоручи велику количину података користећи кластере са кашњењем од само 2ms.
- Трајно складиште - могућност чувања токова података у дистрибуираном, издржљивом, поузданом кластеру отпорном на грешке.
- Висока доступност - могућност ефикасног проширивања кластера преко зона доступности или повезивања кластера из различитих географских региона, што чини Кафку високо доступном и толерантном на грешке без ризика од губитка података.

3 Имплементација основних концепата Кафке уз помоћ OpenMPI-ја

У оквиру овог поглавља описано је како су концепти које нуди *OpenMPI* искоришћени да би се представиле основне функционалности које Кафка нуди. Као илустрација за ток података у реалном времену коришћен је једноставан пролазак кроз низ од првих 100 природних бројева коришћењем *for* петље. При томе свака итерација представља примање податка који се одмах затим шаље на обраду. Пошто се подаци обрађују одмах при њиховом пристизању постиже се илустрација *real-time* обраде података, јер се не чека да сви подаци буду прикупљени, па да се тек затим обрађују.

3.1 Топици

Као што је описано у претходном поглављу, *producer*-и шаљу податке на топике, одакле се даље прослеђују на одговарајуће партиције. Да би се представио концепт *topic*-а коришћен је концепт комуникатора који нуди *OpenMPI*. Комуникатор у основи представља скуп процеса од којих сваки има свој ранг, односно идентификатор. Такође, комуникатор обезбеђује да порука која се шаље преко њега буде достављена искључиво процесима који се налазе унутар њега.

Пример имплементације је замишљен тако да систем садржи максимално 4 топика, али тај број може да буде мањи, зависно од потреба корисника. Такође, замисао је била да се подаци распоређују по топицима зависно од самих података:

- *Topic 0*: Прима податке (бројеве) дељиве бројем 3,
- *Topic 1*: Прима податке (бројеве) дељиве бројем 5,
- *Topic 2*: Прима просте бројеве,
- *Topic 3*: Прима све податке који нису прослеђени ни на један од претходно наведених топика.

Одабир броја топика се врши приликом покретања програма. Команда којом се сам програм је покреће је:

```
mpiexec python3 kafka.py.
```

У продужетку ове команде могуће је одредити који број топика ће се налазити у систему. За сваки топик се наводе "[]" како би се дефинисало да се жели коришћење тог топика. Уз то, унутар "[]" морају да се нађу минимално два елемента (разлог за то ће бити касније описан) како би топик био успешно формиран. На пример, ако бисмо

```
1 world_comm = MPI.COMM_WORLD
2 world_rank = world_comm.Get_rank()
3 world_size = world_comm.Get_size()
```

Изворни код 1: Пример креирања глобалног комуникатора

желели да имамо искључиво топики 0 и 2, начин покретања програма би изгледао овако:

```
mpirun -np 16 --oversubscribe python3 kafka.py [3,4] [] [1, 10, 12] []
```

3.2 Партиције

У Кафки брокери се састоје из партиција, а паралела томе у *OpenMPI*-ју је представљена уз помоћ процеса унутар комуникатора. На почетку, корисници имају могућност да бирају колико укупно партиција ће бити коришћено у систему, али и да заједно са дефинисањем броја топика одреде и које партиције ће се наћи унутар одређеног топика. Ако поново искористимо претходни пример за покретање програма:

```
mpirun -np 16 --oversubscribe python3 kafka.py [3,4] [] [1, 10, 12] []
```

први параметар који се користи је *-np 16* који дефинише да ће систем укупно поседовати 16 партиција. На основу овог броја се прави један глобални топик (комуникатор) који ће садржати све партиције. У оквиру Изворног кода 1 дат је пример креирања глобалног комуникатора.

Касније је неопходно распоредити партиције по топицима, што се ради учитавањем вектора аргумената прослеђених при покретању програма. Топику 0 одговара индекс 1 вектора аргумената, топику 1 индекс 2, итд. Бројеви који се прослеђују унутар "[]" представљају индексе партиција који ће се наћи унутар топика. Ови индекси морају да имају вредност између 0 и укупног броја партиција-1, како би та партиција из глобалног топика, била пребачена у одговарајући топик. Сами индекси, у свету *OpenMPI*-ја представљају рангове процеса, где ранг представља јединствени идентификатор за одговарајући процес. Уз помоћ њих могуће је издвојити низ рангова (које корисник дефинише покретањем програма) који ће бити "пребачени" из глобалног у новокреирани комуникатор, односно топик. Када се комуникатор креира, партиције унутар њега више немају ранг који су имале у оквиру глобалног топика, већ овде добијају рангове од 0 до укупног броја партиција унутар тог топика-1.

Водећа партиција је она која је одговорна да се порука пошаље свим осталим партицијама које се налазе у оквиру истог топика. У Кафки постоји више начина да

```
1 if len(sys.argv) > 1:
2     partitions0 = get_partitions(sys.argv[1], world_size)
3 if len(sys.argv) > 2:
4     partitions1 = get_partitions(sys.argv[2], world_size)
5 if len(sys.argv) > 3:
6     partitions2 = get_partitions(sys.argv[3], world_size)
7 if len(sys.argv) > 4:
8     partitions3 = get_partitions(sys.argv[4], world_size)
9
10
11 def create_comm(world_comm, partitions):
12
13     group = world_comm.group.Incl(ranks)
14     if len(partitions) > 1:
15         group = world_comm.group.Incl(partitions)
16     comm = world_comm.Create(group)
17
18     return comm
```

Изворни код 2: Распоређивање партиција по топицима (комуникаторима)

се изабере која партиција представља ону водећу, што је описано у претходном поглављу, а у оквиру ове имплементације одабрано је да увек партиција са најмањим рангом буде она која је водећа. из тог разлога, пре него што се креира комуникатор (топик) низ партиција се сортира, како би водећа партиција сваког топика имала ранг 0. Та водећа партиција је одговорна за прослеђивање порука свим осталим партицијама.

Концептом комуникатора и процеса, омогућено је да се илуструје концепт топика и партиција из Кафке, јер слањем поруке у оквиру комуникатора, ту поруку примају искључиво процеси који се налазе унутар тог комуникатора, што заправо обезбеђују топици у Кафки.

3.3 Подаци у оквиру партиција

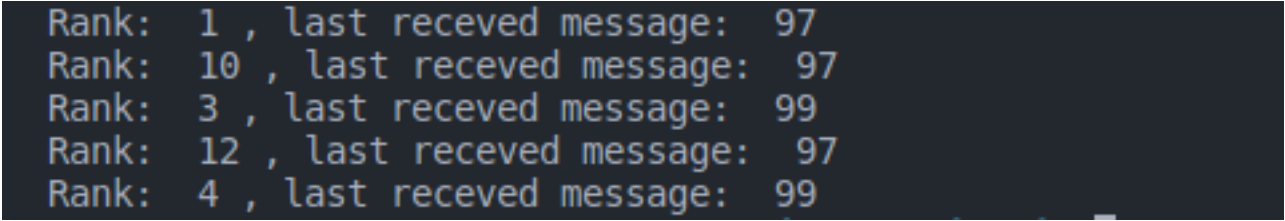
Подаци се од водеће партиције достављају свим осталим партицијама уз помоћ методе `bcast` одговарајућег комуникатора. Ова метода неће кренути са слањем следеће поруке, све док све партиције унутар топика (комуникатора) не приме претходну поруку. Овим је постигнуто да ниједна партиција (процес унутар комуникатора) не касни у односу на остале, и да су све партиције, унутар једног топика, увек усклађене.

Такође сами процеси унутар комуникатора имају дистрибуирану меморију, што значи да стање меморије једног процеса ни у ком случају не зависи од стања меморије других процеса. Ова карактеристика *OpenMPI*-ја нуди баш оно што Кафкине партиције имају, а то је независност, и самостално вођење рачуна о томе које су податке примиле.

Дистрибуираност меморије могуће је испратити уз помоћ променљиве *last_received*. Када се она дефинише, свака партиција добија своју приватну верзију ове променљиве, које је независна од стања те променљиве у осталим партицијама. Сваки пут када партиција прими податак, он се уписује у ову променљиву, а када се на крају програма провери њено стање (Слика 1), може се видети да све партиције у оквиру једног топика имају исту вредност ове променљиве, што је обезбеђено уз помоћ методе *bcast*.

```
1 for i in range(1, 101):
2     data = i
3     root = 0
4     sent = False
5
6     if (i % 3) == 0 and len(partitions0) > 1:
7         if rank0 != MPI.UNDEFINED:
8             comm0.bcast(data, root)
9             print('Rank: ', world_rank, ', message: ', data)
10            sent = True
11
12    if (i % 5) == 0 and len(partitions1) > 1:
13        if rank1 != MPI.UNDEFINED:
14            comm1.bcast(data, root)
15            print('Rank: ', world_rank, ', message: ', data)
16            sent = True
17
18    if is_prime(i) and len(partitions2) > 1:
19        if rank2 != MPI.UNDEFINED:
20            comm2.bcast(data, root)
21            print('Rank: ', world_rank, ', message: ', data)
22            sent = True
23
24    if sent == False and len(partitions3) > 1:
25        if rank3 != MPI.UNDEFINED:
26            comm3.bcast(data, root)
27            print('Rank: ', world_rank, ', message: ', data)
```

Изворни код 3: Слање података



```
Rank: 1 , last received message: 97  
Rank: 10 , last received message: 97  
Rank: 3 , last received message: 99  
Rank: 12 , last received message: 97  
Rank: 4 , last received message: 99
```

Слика 1: Последњи примљени податак по партицијама

4 Закључак

Један алат високог нивоа апстракције као што је Кафка нуди корисницима бројне могућности уз минимално утрошено време конфигурације. Са друге стране, алат ниског нивоа попут *OpenMPI*-ја за сваку функционалност, која је приближно слична ономе што нуди Кафка, захтева доста подешавања.

Као прво, неопходно је подесити коришћење процеса, односно логичких језгара рачунара, као партиција. Поред тога што *OpenMPI* има подразумевану дистрибуирану меморију сваког процеса, ова дистрибуираност представља највећу муку при имплементацији, јер дистрибуираност меморије није нешто што је свима блиско. Потребно је у сваком тренутку пратити стање меморије сваког процеса, како би било обезбеђено да у одређеним ситуацијама сви процеси имају исто стање одређених променљивих на глобалном нивоу, а у другим да сваки процес има своје приватно стање променљивих.

Код Кафке, корисници о овоме не морају да брину, када формирају топике и одреде број партиција унутар сваког топика, слање порука и стање меморије, нису више њихова брига.

Иако су у оквиру овог рада имплементирани неки од главних концепата Кафке, она такође нуди и могућност складиштења података у оквиру топика. Ова функционалност би представљала могуће проширење овог рада, с обзиром да је при тренутној имплементацији могуће искључиво чување последњег примљеног податка у оквиру партиције. Могуће је проширење се огледа у томе да сваки податак када буде примљен буде и сачуван у оквиру неке врсте текстуалног фајла, како би била могућа и визуализација свих примљених података.

Библиографија