AZƏRBAYCAN RESPUBLİKASI ELM VƏ TƏHSİL NAZİRLİYİ

AZƏRBAYCAN TEXNİKİ UNİVERSİTETİ

"Kibertəhlükəsizlik" kafedrası

Əlyazması hüququnda

Qasımov Elçin Yusif oğlu

İxtisas: 060632 – "İnformasiya texnologiyaları və sistemləri mühəndisliyi"

İxtisaslaşma: "İnformasiya mühafizəsi və təhlükəsizliyi"

Mövzu: Data-intensiv sistemlərdə replikasiya üsullarının müqayisəli tədqiqi və reallaşdırılması

MAGİSTRLİK DİSSERTASİYASI

Elmi rəhbər: **t.e.n. dos. A. F. Hüseynov**

Bakı-2023

MÜNDƏRİCAT

GİRİŞ

Mövzunun aktuallığı. Replikasiya üsulları data-intensiv sistemlərdə bu sistemlərin dizayn edilməsi qədər mühüm xarakter daşıyır, çünki onlar məlumatların əlçatanlığını, məlumatların davamlılığını və məlumat əldə etmək performansını yaxşılaşdırmaq üçün bir vasitə təmin edir, bu da müasir məlumatların idarə edilməsində, data sistemlərində etibarlılığın və miqyaslan bilirliyin təmin edilməsində xüsusi rol oynayır.

Hər sistemin fərqli tələb və gözləntilər olduğundan eyni üsul tətbiq edilə bilmir. Müxtəlif amillərin(yazma və oxuma əməliyyatların intensivliyi, izafilik - consistency) təsiri, eləcə də sistemin fəaliyyət göstərdiyi mühitdən(monolit vəya paylanmış mühit) asılı olaraq fərqli replikasiya üsulları mövcuddur. Digər tərəfdən sistemdə baş verən nasazlıqlar - şəbəkə sınması, aparat təminatı qəzaları bu üsulların həmin vəziyyəti necə idarəetməsi diqqət edilməli nüanslardan biridir.

Yuxarıda qeyd olunanlar əsasında deyə bilərik ki, tədqiqat işinin mövzusu kifayət qədər aktual və bu sahənin tədqiqi məsələyə kompleks yanaşmanın olmasını tələb edir.

Mövzunun öyrənilmə vəziyyəti. Data-intensiv sistemlərdə replikasiya üsullarının müqayisəli tədqiqi və inkişaf etdirilməsi bu sahənin mütəxəssisləri tərəfindən, o cümlədən də Martinn Klepman, Alex Petrov və digərləri tərəfindən əhatəli araşdırılmışdır.

Lakin təəsüflər olsun ki, bu sahə üzrə yerli mütəxəssislər tərəfindən tədqiqi mümkün olmamışdır. Bu isə data-intensiv sistemlərdə replikasiya üsullarının müqayisəli tədqiqi və araşdırılmasına dair kifayət qədər boşluqların olduğunu göstərir.

Tədqiqat işinin predmeti. Tədqiqat işinin predmetinin əsasında data-intensiv sistemlərdə replikasiya üsullarının müqayisəli tədqiqi dayanır.

Tədqiqat işinin obyekti. Tədqiqatın obyekti olaraq isə data-intensiv sistemlərdə replikasiya üsullarının müqayisəli tədqiqi və reallaşdırılması seçilmişdir.

Tədqiqat işinin metodu. Dissertasiya mövzusunun tədqiq olunmasında həm empirik, həm də nəzəri tədqiqat metodlarından istifadə olunmuşdur. Belə ki, data-intensiv sistemlərdə replikasiya üsullarının öyrənilməsində, müqayisəli analizində

müşahidəetmə, faktların toplanması və seçilməsi, onlar arasında qarşılıqlı əlaqənin yaradılması kimi empirik metodlardan istifadə olunmaqla yanaşı, eyni zamanda da tədqiqatın məqsədyönlülüyünü artırmaq məqsədilə müqayisə, ölçmə, eksperiment, analiz, sintez, induksiya və deduksiya kimi metodlardan da geniş istifadə olunmuşdur. Belə ki, analiz metodu kimi mövzu tam şəkildə götürülmüş və daha sonra fəsillərə bölünərək ayrı-ayrılıqda təhlil olunmuşdur. Daha sonra isə sintez metodu vasitəsilə bu fəsillər ümumi sistemdə birləşdirilmişdir. İnduksiya metodu vasitəsilə dissertasiya işi haqqında texniki faktlar toplanmış, sistemləşdirilmiş və araşdırılmışdır. Sonra isə deduksiya metodu vasitəsilə isə həmin toplanmış faktlar əsasında nəzəri nəticələr, ümumi prinsiplər, başqa sözlə desək, əməli fəaliyyət üçün lazım olan zəruri tövsiyyələr müəyyən edilmişdir.

Tədqiqat işinin məqsədi və vəzifələri. Tədqiqat işinin əsas məqsədi data-intensiv sistemlərdə replikasiya üsullarının müqayisəli tədqiqi və reallaşdırılmasıdır. Bu məqsədə çatmaq üçün isə aşağıdakı əsas vəzifələrin icra olunması məqsədə müvafiqdir:

1. Data-intensiv sistemlərin öyrənilməsi.
2. Bu sistemlərdə replikasiyanın həyata keçirilməsi zamanı istifadə edilən üsulların tədqiqi və müqayisəli analiz edilməsi.
3. Tək-liderli replikasiya üsulunun reallaşdırılması, nümunə proqram təminatının hazırlanması.

Tədqiqat işinin informasiya bazası. Tədqiqat işinin informasiya bazasının əsasında data-intensiv sistemlərdə replikasiya üsullarının tədqiqinə dair xarici ədəbiyyatlar, jurnallar, elmi məqalələr, o cümlədən də internet informasiya ehtiyatlarının resursları təşkil edir.

Tədqiqat işinin elmi yeniliyi. Tədqiqat işinin elmi yeniliyini aşağıdakılar təşkil edir:

1. Mövcud üsullarda bəzi çatışmazlıqların - müasir sistemlərdə olan tələblərin ödənilmədiyi halların, aradan qaldırılması istiqamətində həll yolları.
2. Möcud replikasiya üsullarının müsbət və mənfi tərəflərini müəyyən etmək və uyğun tətbiq yerlərinin müəyyən edilməsi.

Tədqiqat işinin strukturu və quruluşu. Tədqiqat işi girişdən, 3 fəsildən, nəticə və təkliflərdən ibarətdir. Tədqiqat işinin sonunda ədəbiyyat siyahısı verilmişdir.

Tədqiqat işinin giriş hissəsində mövzunun aktuallığı, tədqiqat metodları, mövzunun öyrənilmə səviyyəsi, tədqiqatın məqsədi və vəzifələri, eləcə də tədqiqat işinin predmeti və obyekti öz əksini tapmışdır.

Tədqiqat işinin birinci fəslində data-intensiv sistemlərin nə olduğu, hansı xüsusiyyətlərə malik olması və bu sistemlərdə olan xüsusi məsələlərə yer ayrılmışdır.

Tədqiqat işinin ikinci fəsli replikasiya və data-intensiv sistemlərdə replikasiyanın mahiyyəti, mövcud istifadə edilən üsullar və həmin üsulların istifadə edilmə yerləri araşdırılmış, müqayisəli təhlil edilmişdir.

Tədqiqat işinin üçüncü fəsli praktiki yönümlü xarakter daşıyır. Nümunə olaraq, RAFT replikasiya üsuluna əsaslanan proqram təminatı hazırlanmışdır.

Tədqiqat işinin nəticə hissəsində data-intensiv sistemlərdə replikasiya üsullarının təkmilləşdirilməsinə dair təkliflər, bu üsulların tədqiqi nəticəsində müəyyən edilən tövsiyyələr verilmişdir.

Dissertasiya işinin sonunda isə tədqiqatın istinad olunduğu ədəbiyyat siyahısı və eləcə də informasiya bazası kimi internet resursları qeyd olunmuşdur.

I FƏSİL. DATA-İNTENSİV SİSTEMLƏRİN TƏDQİQİ

1.1. Data-intensiv sistemlərin tədqiqi

Günümüzdəki müasir proqram təminatlarının vəya sistemlərin tələblərinə nəzər yetirsək, onların əksəriyyətinin hesablama intensiv yox, data intensiv olduğu görə bilərik. CPU gücü nadir hallarda bu proqramlar üçün məhdudlaşdırıcı amildir - daha böyük problemlər isə adətən məlumatların miqdarı, məlumatların mürəkkəbliyi və onların dəyişmə sürətidir [1].

Data-intensiv sistemlər bir neçə oxşar funksionallıqları paylaşırlar.

- Məlumatı elə saxlamaq lazımdır ki, bu və ya digər proqram onları daha sonra yenidən tapa bilməsi (verilənlər bazaları)
- Oxumaları sürətləndirmək üçün bahalı əməliyyatın nəticəsini xatırlaması(keşlər)
- İstifadəçilərə məlumatı açar sözə görə axtarmağa və ya müxtəlif üsullarla filter əməliyyatından keçirməyə icazə verilməsi (axtarış indeksləri)
- Asinxron şəkildə idarə olunmaq üçün başqa prosesə mesaj göndərilməsi (axın işlənilməsi)
- Böyük miqdarda yığılmış məlumatı vaxtaşırı parçalamaq (toplu emal)

Biz adətən verilənlər bazası, növbələr(*ing., queues*), keşlər və s. alətlərin çox müxtəlif kateqoriyaları olduğunu düşünürük. Verilənlər bazası və mesaj növbəsi(*ing., message queue*) bəzi səthi oxşarlığa malik olsa da

- hər ikisi müəyyən müddət ərzində məlumatları saxlayır
- onların çox fərqli formada verilənlərə müraciət üsulları var, bu fərqlilik öz növbəsində performansların da fərqlənməsinə gətirib çıxarır.

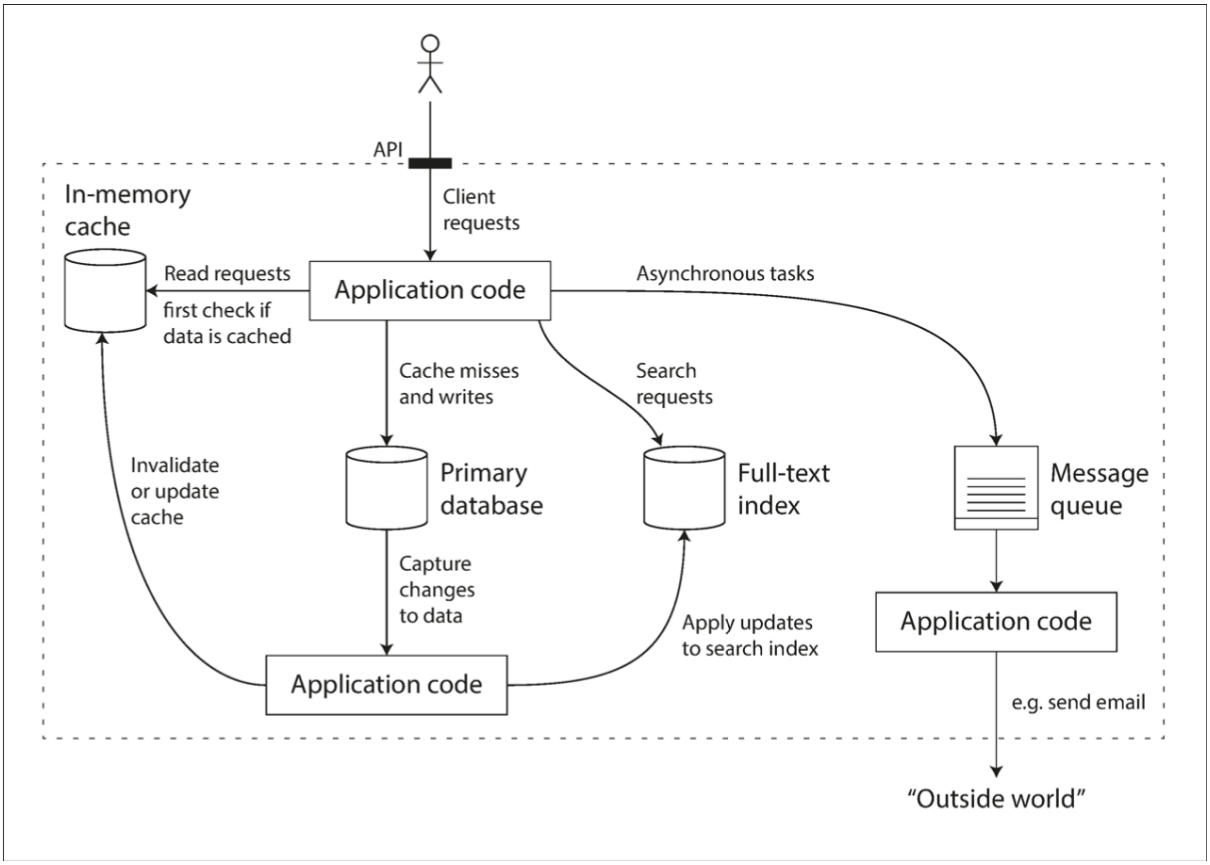
Sual yarana bilər ki, nəyə görə məhz yeni termin - data-intensiv sistemlər altında bu sistemləri qruplaşdırmaq ehtiyac duyulur?

Son illərdə məlumatların saxlanması və emalı üçün bir çox yeni proqram(və ya sistemlər) ortaya çıxdı. Onlar müxtəlif istifadə halları üçün optimallaşdırılmışdır və onlar artıq ənənəvi kateqoriyalara uyğun gəlmir [6]. Məsələn, mesaj növbələri(*ing., message queue*, Redis) kimi də istifadə olunan məlumat anbarları və verilənlər bazası

kimi davamlılıq zəmanəti olan mesaj növbələri var (*ing., message brokers, Apache Kafka, Apache Pulsar, RedPanda*).

Digər tərəfdən, müasir proqram təminatları üçün olan tələb və ehtiyaclar o qədər fərqli və unikal olur ki, tək vahid bir sistem istifadə edilərək, həmin tələblərin ödənilməsi hardasa qeyri-mümkün olur, olsa da effektiv olmur.

Həll yolu kimi, bir neçə komponentləri birlikdə istifadə edərək, ümumi tələblər ödənilir.



Şəkil 1.1. Sistem arxitekturası

Şəkil 1.1-də qeyd olunan sistem arxitekturasına nəzər yetirsək, sistemin bir neçə komponentdən ibarət olduğunu görə bilərik.

In-memory cache - tez-tez sorğulanan məlumatları daxili vəya operativ yaddaşında(RAM) saxlayaraq, keş rolunu oynayır.

Primary database - yazma əməliyyatları bu verilənlər bazasında icra olunur.

Full-text-index - tam mətn axtarış tipli sorğular bu bazada yönləndirilir.

Message queue - mesajların asinxron icra olunması üçün mesajlari müvəqqəti özündə saxlayan sistemdir.

Change Data Capture - bu texnologiya vasitəsilə verilənlər üzərində edilən dəyişiklər oxuyub, digər sistemlərə göndərmək mümkündür və beləliklə *In-memory-cache* və *Full-text index* sistemlərinin *Primary database* ilə consistent olmasını təmin edə bilərik [7].

Bu komponentlər bir-birləri ilə əvvəlcədən müəyyən olunmuş API(*ing,. Application Programming Interface*) vasitəsilə əlaqə qururlar.

1.2. Data-intensiv sistemlərin xüsusiyyətlərinin analizi

Bir məlumat sistemi və ya servisi dizayn edirsinizsə, bir çox suallar yaranır.

- Daxildə işlər səhv getsə belə, məlumatların düzgün(*ing., consistent*) və tam qalmasını(*ing., integrity*) necə təmin edilməlidir?
- Sisteminizin bəzi hissələri performası zəifləsə vəya sıradan çıxsa, müştərilərə davamlı olaraq yaxşı performansı necə təmin etmək olar?
- Artan yükü idarə etmək üçün sistemi necə miqyaslana bilən etmək olar?
- Servisin API necə dizayn edilməlidir ?

Məlumat sisteminin dizaynına təsir edə biləcək bir çox amillər var, o cümlədən dizaynda iştirak edecək insanların bacarıqları və təcrübəsi, köhnə sistem asılılıqları, çatdırılma üçün vaxt miqyası, təşkilatınızın müxtəlif risk növlərinə dözümlülüyü, tənzimləyici məhdudiyyətlər və s. Bu amillər vəziyyətdən çox asılıdır.

Bu bölmədə biz əksər proqram sistemlərində vacib olan üç məsələyə diqqət yetirəcəyik.

Etibarlılıq(*ing., Reliability*)

Sistem hətta baş verən çətinliklərə (aparat və ya proqram təminatı nasazlıqları və hətta insan səhvi) baxmayaraq düzgün işləməyə davam etməlidir (istənilən performans səviyyəsində düzgün funksiyanı yerinə yetirməklə) [7].

Miqyaslanma(i*ng., Scalability*)

Sistem böyüdükcə (məlumat həcmi, trafik həcmi və ya mürəkkəblik baxımından) bu artımla mübarizə aparmaq üçün ağlabatan yollar olmalıdır [8].

Xidmət qabiliyyəti(*ing., Maintainability*)

Zamanla sistem üzərində çoxlu müxtəlif insanlar işləyəcək (mühəndislik və əməliyyatlar, həm cari davranışı qorumaq, həm də sistemi yeni istifadə vəziyyətlərinə uyğunlaşdırmaq) və onların hamısı onun üzərində məhsuldar işləyə bilməlidirlər [2].

Etibarlılıq(*ing., Reliability*)

Hər kəs bir şeyin etibarlı və ya etibarsız olmasının nə demək olduğu barədə intuitiv təsəvvürə malikdir.

Proqram təminatı üçün tipik gözləntilərə aşağıdakılar daxildir:

- Tətbiq istifadəçinin gözlədiyi funksiyanı yerinə yetirir.
- O, istifadəçinin səhv etməsinə və ya proqram təminatından gözlənilməz şəkildə istifadə etməsinə dözə bilər.
- Onun performansı gözlənilən yük və məlumat həcmi altında tələb olunan istifadə halı üçün kifayət qədər yaxşıdır.
- Sistem hər hansı icazəsiz girişin və sui-istifadənin qarşısını alır.

Əgər bütün bu şeylər birlikdə "düzgün işləmək" mənasını verirsə, onda biz etibarlılığı, təqribən "işlər səhv getdikdə belə düzgün işləməyə davam etmək" mənasını başa düşə bilərik [14].

Səhv ola biləcək şeylərə nasazlıqlar(ing faults), nasazlıqları qabaqcadan görən və onların öhdəsindən gələ bilən sistemlərə isə xətaya dözümlü(fault tolerant) və ya möhkəm(resilient) deyilir [3]. Əvvəlki termin bir qədər yanıltıcıdır: bu, sistemi reallıqda mümkün olmayan hər cür nasazlığa dözümlü edə biləcəyimizi göstərir.

Qeyd edək ki, nasazlıq(fault) uğursuzluqla(failure) eyni deyil. Bir nasazlıq adətən sistemin bir komponentinin öz spesifikasiyasından kənara çıxması kimi müəyyən edilir, halbuki uğursuzluq sistemin bütövlükdə istifadəçiyə tələb olunan xidməti göstərməyi dayandırmasıdır.

Nasazlığın olma ehtimalını sıfıra endirmək mümkün deyil; buna görə də adətən ən yaxşısı xətaların uğursuzluğa səbəb olmasının qarşısını alan nasazlığa dözümlülük mexanizmlərinin layihələndirilməsidir [6].

Aparat təminatı nasazlıqları(ing., Hardware faults)

Sistem nasazlığının səbəblərini düşündüyümüz zaman ağlımıza tez bir zamanda aparat nasazlıqları gəlir. Sərt disklər sıradan çıxır, RAM sıradan çıxır, elektrik şəbəkəsi kəsilib, kimsə səhv şəbəkə kabelini ayırır. Böyük məlumat mərkəzləri ilə işləyən hər kəs sizə deyə bilər ki, çoxlu maşınlarınız olduqda bu cür hadisələr hər zaman baş verir. Sərt disklərin ortalama uğursuzluq müddəti (MTTF) təxminən 10-50 il olduğu bildirilir. Beləliklə, 10.000 diski olan bir saxlama klasterində gündə orta hesabla bir diskin sıradan çıxmasını gözləyə bilərik [6].

İlk cavabımız adətən sistemin nasazlıq dərəcəsini azaltmaq üçün fərdi aparat komponentlərinə ehtiyat əlavə etməkdir. Disklər RAID konfiqurasiyasında qurula bilər, serverlərdə ikili enerji təchizatı və isti dəyişdirilə bilən CPU ola bilər və məlumat mərkəzlərində ehtiyat enerji üçün batareyalar və dizel generatorları ola bilər.

Komponentlərdən biri öldükdə, qırılan komponent dəyişdirilərkən lazımsız komponent onun yerini tuta bilər. Bu yanaşma hardware problemlərinin nasazlıqlara səbəb olmasının qarşısını tamamilə ala bilməz, lakin o, yaxşı başa düşülür və tez-tez bir maşının illər boyu fasiləsiz işləməsini təmin edə bilər.

Proqram təminatı xətaları(ing., Software Errors)

Biz adətən hardware nasazlıqlarını təsadüfi və bir-birindən müstəqil hesab edirik: bir maşının diskinin nasazlığı başqa bir maşının diskinin sıradan çıxması demək deyil. Zəif korrelyasiya ola bilər (məsələn, ümumi səbəbə görə, məsələn, server çarxındakı temperatur), lakin əks halda eyni zamanda çoxlu sayda aparat komponentlərinin sıradan çıxması ehtimalı azdır.

Başqa bir nasazlıq sinfi sistem daxilində sistematik xətadır. Bu cür nasazlıqları təxmin etmək daha çətindir və onlar nodelar arasında korrelyasiya olunduğundan, korrelyasiya olunmayan aparat xətalarından daha çox sistem nasazlığına səbəb olur.

Məsələn:

- Bəzi paylaşılan resurs- CPU vaxtını, yaddaşı, disk sahəsini və ya şəbəkə bant genişliyini istifadə edən nəzarətdən altında olmayan proses.

- Kaskadlı nasazlıqlar, burada bir komponentdə kiçik bir nasazlıq digər komponentdə nasazlıq yaradır və bu da öz növbəsində əlavə nasazlıqlara səbəb olur.

Proqram təminatında sistematik nasazlıqlar probleminin tez həlli yoxdur. Çoxlu xırda şeylər kömək edə bilər:

- sistemdəki fərziyyələr və qarşılıqlı əlaqələr haqqında diqqətlə düşünmək; hərtərəfli sınaq;
- prosesin izolyasiyası; proseslərin çökməsinə və yenidən başlamasına imkan verir;
- İstehsalda sistemin davranışını ölçmək, izləmək və təhlil etmək.

Sistemin müəyyən bir zəmanət verməsi gözlənilirsə (məsələn, mesaj növbəsində, gələn mesajların sayı gedən mesajların sayına bərabərdir), o, işləyərkən daim özünü yoxlaya bilər və uyğunsuzluq olarsa, xəbərdarlıq verə bilər.

İnsan xətaları(ing., Human Errors)

İnsanlar proqram sistemlərini dizayn edir və qururlar və sistemləri işlək vəziyyətdə saxlayan operatorlar da insandır. Ən yaxşı niyyətləri olsa belə, insanların etibarsız olduğu bilinir.

Məsələn, böyük internet xidmətlərinin bir araşdırması göstərdi ki, operatorlar tərəfindən konfiqurasiya xətaları kəsintilərin əsas səbəbidir, halbuki avadanlıq nasazlıqları (serverlər və ya şəbəkə) kəsilmələrin yalnız 10-25%-də rol oynamışdır.

Etibarsız insanlara baxmayaraq, sistemlərimizi necə etibarlı edə bilərik?

Ən yaxşı sistemlər bir neçə yanaşmanı birləşdirir:

- Sistemləri səhv imkanlarını minimuma endirəcək şəkildə dizayn edin. Məsələn, yaxşı dizayn edilmiş abstraksiyalar, API-lər və admin interfeysləri “doğru olanı” asanlaşdırır və “yanlış iş”dən çəkindirir. Bununla belə, əgər interfeyslər çox məhdudlaşdırıcıdırsa, insanlar onların ətrafında işləyərək faydalarını inkar edəcəklər, buna görə də bu, düzgün əldə etmək çətin bir tarazlıqdır.

- İnsanların ən çox səhv etdikləri yerləri uğursuzluqlara səbəb ola biləcək yerlərdən ayırın. Xüsusilə, insanların real istifadəçilərə təsir etmədən real datadan istifadə edərək təhlükəsiz şəkildə kəşf edə və sınaqdan keçirə bildiyi tam xüsusiyyətli qeyri-istehsal sandbox mühitləri təmin edin.
- Vahid testlərindən tutmuş bütün sistem inteqrasiya testlərinə və əl testlərinə qədər bütün səviyyələrdə hərtərəfli sınaqdan keçirin. Avtomatlaşdırılmış sınaqdan geniş istifadə olunur, yaxşı başa düşülür və normal əməliyyatda nadir hallarda yaranan künc hallarını əhatə etmək üçün xüsusilə dəyərlidir.
- Uğursuzluq halında təsirləri minimuma endirmək üçün insan səhvlərindən tez və asan bərpa olunmağa icazə verin.

  Məsələn, konfiqurasiya dəyişikliklərini geri qaytarmağı sürətləndirin, tədricən yeni kodu təqdim edin (belə ki, hər hansı gözlənilməz səhvlər istifadəçilərin yalnız kiçik bir hissəsinə təsir etsin) və məlumatları yenidən hesablamaq üçün alətlər təqdim edin (əgər köhnə hesablamaların köhnə hesablamaların işləndiyi ortaya çıxsa).

Etibarlılıq nə qədər vacibdir?

Etibarlılıq təkcə atom elektrik stansiyaları və hava hərəkətinə nəzarət proqramları üçün deyil – daha sadə tətbiqlərin də etibarlı şəkildə işləməsi gözlənilir.

Biznes tətbiqlərindəki səhvlər məhsuldarlığın itirilməsinə səbəb olur (və rəqəmlər səhv bildirildikdə hüquqi risklər) və e-ticarət saytlarının kəsilməsi itirilmiş gəlir və nüfuza ziyan baxımından böyük xərclərə səbəb ola bilər.

Miqyaslanma(ing., Scalability)

Sistem bu gün etibarlı işləsə belə, bu o demək deyil ki, gələcəkdə mütləq etibarlı şəkildə işləyəcək.

Performans zəifləməsinin ümumi səbəblərindən biri artan yükdür: ola bilsin ki, sistem eyni vaxtda 10.000 istifadəçidən 100.000 paralel istifadəçiyə və ya 1 milyondan 10 milyona qədər artıb. Ola bilsin ki, o, əvvəlkindən daha böyük həcmdə verilənləri emal edir [6].

Miqyaslanma, sistemin artan yükün öhdəsindən gəlmək qabiliyyətini təsvir etmək üçün istifadə edilən termindir.

Bununla belə, nəzərə alın ki, bu, bir sistemə əlavə edə biləcəyimiz bir ölçülü etiket deyil: “X miqyaslana bilir” və ya “Y miqyaslı deyil” demək mənasızdır. Əksinə, miqyaslılığın müzakirəsi “Sistem müəyyən bir şəkildə böyüyərsə, böyümənin öhdəsindən gəlmək üçün seçimlərimiz nələrdir?” kimi sualları nəzərdən keçirmək deməkdir. və "Əlavə yükü idarə etmək üçün hesablama resurslarını necə əlavə edə bilərik?"

Artan yükün öhdəsindən gəlmək üçün yanaşmalar

Bir səviyyəli yükə uyğun olan arxitektura çətin ki, 10 dəfə yükün öhdəsindən gələ bilsin.

Sürətlə böyüyən bir xidmət üzərində işləyirsinizsə, buna görə də çox güman ki, arxitekturanızı hər bir yük artımı sifarişində yenidən nəzərdən keçirməli olacaqsınız - və ya bəlkə də daha tez-tez.

Geniş miqyasda işləyən sistemlərin arxitekturası adətən tətbiq üçün çox spesifikdir - ümumi, hər kəsə uyğun standart arxitektura kimi bir şey yoxdur. Arxitektura seçilərkən nəzərə alınmalıdır: oxumaların həcmi yazıların həcmi, saxlanacaq məlumatların həcmi, məlumatların mürəkkəbliyi, cavab müddəti tələbləri, giriş nümunələri və s.

Məsələn, hər biri 1 kB ölçüsündə saniyədə 100.000 sorğunu idarə etmək üçün nəzərdə tutulmuş bir sistem, hər iki sistem eyni olsa belə, dəqiqədə 3 sorğu üçün nəzərdə tutulmuş sistemdən çox fərqli görünür, hər biri 2 GB ölçüsündədir. məlumat ötürmə qabiliyyəti.

Xidmət qabiliyyəti(*ing., Maintainability*)

Məlumdur ki, proqram təminatının dəyərinin böyük hissəsi onun ilkin işlənib hazırlanmasında deyil, onun davamlı texniki xidmətində – səhvlərin aradan qaldırılması, sistemlərinin işlək vəziyyətdə saxlanılması, nasazlıqların araşdırılması,

onun yeni platformalara uyğunlaşdırılması, yeni istifadə halları üçün dəyişdirilməsidir. , texniki borcun ödənilməsi və yeni funksiyaların əlavə edilməsi.

Biz proqram təminatını elə dizayn edə bilərik və etməliyik ki, o, texniki xidmət zamanı çətinlikləri minimuma endirsin.

Bu məqsədlə proqram sistemləri üçün üç dizayn prinsipinə xüsusi diqqət yetirəcəyik:

- Əməliyyat qabiliyyəti: Əməliyyat qruplarına sistemin düzgün işləməsini asanlaşdırın.
- Sadəlik: Sistemdən mümkün qədər çox mürəkkəbliyi aradan qaldıraraq, yeni mühəndislərin sistemi başa düşməsini asanlaşdırın. (Qeyd edək ki, bu istifadəçi interfeysinin sadəliyi ilə eyni deyil.)
- Təkamül qabiliyyəti: Mühəndislərin gələcəkdə sistemdə dəyişikliklər etməsini asanlaşdırın, tələblər dəyişdikcə onu gözlənilməz istifadə hallarına uyğunlaşdırın. Genişlənmə, dəyişdirilə bilənlik və ya plastiklik kimi də tanınır.

Operativlik: Əməliyyatlar üçün həyatı asanlaşdırmaq

Əməliyyat qrupları proqram sisteminin düzgün işləməsini təmin etmək üçün çox vacibdir.

Yaxşı bir əməliyyat qrupu adətən aşağıdakılara və daha çoxuna cavabdehdir:

• Sistemin sağlamlığının monitorinqi və nasazlıq yaranarsa, xidmətin tez bərpa edilməsi

• Sistem nasazlıqları və ya aşağı performans kimi problemlərin səbəblərinin izlənməsi

• Proqram təminatının və platformaların, o cümlədən təhlükəsizlik yamalarının yenilənməsi

• Fərqli sistemlərin bir-birinə necə təsir etdiyini izləmək, beləliklə, problemli dəyişikliyin zədələnmədən əvvəl qarşısını almaq olar.

Gələcək problemləri təxmin etmək və onlar meydana gəlməzdən əvvəl onları həll etmək (məsələn, potensialın planlaşdırılması)

• Yerləşdirmə, konfiqurasiyanın idarə edilməsi və s. üçün yaxşı təcrübələrin və alətlərin yaradılması

• Tətbiqi bir platformadan digərinə köçürmək kimi mürəkkəb texniki xidmət tapşırıqlarının yerinə yetirilməsi

• Konfiqurasiya dəyişiklikləri edildikdə sistemin təhlükəsizliyinin təmin edilməsi

• Əməliyyatları proqnozlaşdırıla bilən və istehsal mühitini sabit saxlamağa kömək edən proseslərin müəyyən edilməsi

• Ayrı-ayrı insanlar gəlib getsə belə, təşkilatın sistem haqqında biliklərinin qorunması

Sadəlik: Mürəkkəbliyin idarə edilməsi

Kiçik proqram layihələri sadə və ifadəli koda malik ola bilər, lakin layihələr böyüdükcə onlar çox vaxt çox mürəkkəb və anlaşılması çətin olur. Bu mürəkkəblik sistem üzərində işləməli olan hər kəsi ləngidir və texniki xidmətin qiymətini daha da artırır.

Yüksək səviyyəli proqramlaşdırma dilləri maşın kodunu, CPU registrlərini və sistem zənglərini gizlədən abstraksiyalardır. SQL mürəkkəb diskdə və yaddaşdaxili məlumat strukturlarını, digər müştərilərin paralel sorğularını və qəzalardan sonra uyğunsuzluqları gizlədən abstraksiyadır. Əlbəttə ki, yüksək səviyyəli bir dildə proqramlaşdırma zamanı biz hələ də maşın kodundan istifadə edirik; biz ondan birbaşa istifadə etmirik, çünki proqramlaşdırma dili abstraksiya bizi bu barədə düşünmək məcburiyyətindən xilas edir.

Ancaq yaxşı abstraksiyalar tapmaq çox çətindir. Paylanmış sistemlər sahəsində, çoxlu yaxşı alqoritmlər olsa da, sistemin mürəkkəbliyini idarəolunan səviyyədə saxlamağa kömək edən onları abstraksiyalara necə yığmağımız daha az aydındır.

Təkamül qabiliyyəti: Dəyişikliyi asanlaşdırmaq

Sisteminizin tələblərinin əbədi olaraq dəyişməz qalması ehtimalı çox azdır. Onların daimi axınında olma ehtimalı daha yüksəkdir: siz yeni faktlar öyrənirsiniz, əvvəllər gözlənilməz istifadə halları yaranır, biznes prioritetləri dəyişir, istifadəçilər yeni funksiyalar tələb edir, yeni platformalar köhnə platformaları əvəz edir, hüquqi və ya tənzimləyici tələblər dəyişir, sistem qüvvələrinin artması memarlıq dəyişiklikləri və s.

Təşkilati proseslər baxımından, çevik iş nümunələri dəyişikliklərə uyğunlaşmaq üçün çərçivə təmin edir. Agile icması həmçinin tez-tez dəyişən mühitdə proqram

təminatının işlənib hazırlanması zamanı faydalı olan texniki alətlər və nümunələr işləyib hazırlamışdır, məsələn, testə əsaslanan inkişaf (TDD) və refaktorinq.

Məlumat sistemini dəyişdirmək və onu dəyişən tələblərə uyğunlaşdırmaq asanlığı onun sadəliyi və onun abstraksiyaları ilə sıx bağlıdır: sadə və asan başa düşülən sistemləri dəyişdirmək adətən mürəkkəb sistemlərdən daha asandır. Ancaq bu, çox vacib bir fikir olduğundan, məlumat sistemi səviyyəsində çevikliyə istinad etmək üçün fərqli bir söz istifadə edəcəyik: təkamül qabiliyyəti

II FƏSİL. REPLİKASİYANIN TƏDQİQİ

2.1. Replikasiyanın mahiyyəti və ilkin anlayışlar

Replikasiya eyni məlumatın bir nüsxəsini şəbəkə vasitəsilə birləşdirilən bir neçə maşında saxlamaq deməkdir. Məlumatları təkrarlamaq istəməyinizin bir neçə səbəbi var:

• Məlumatları coğrafi cəhətdən sistem istifadəçilərinizə yaxın saxlamaq (və beləliklə də məlumat ötürülməsi zamanı gecikməni azaltmaq)

• Bəzi hissələri(aparat təminatə, qurğular və s.) sıradan çıxsa belə sistemin işləməyə davam etməsinə icazə vermək (və beləliklə əlçatanlığı vəya əlyetərliliyi artırmaq)

• Oxunma sorğularına xidmət edə bilən maşınların(və ya serverlərin) sayını artırmaq (və beləliklə sistem istifadəçi sorğuları ilə çox yükləndikdə belə sistemin performansı zəifləməyəcək)

Əgər replika etdiyiniz məlumatlar zamanla - yəni statik məlumatlardırsa, dəyişmirsə, o zaman replikasiya asandır: sadəcə olaraq məlumatları hər qovşaqda bir dəfə kopyalamalısınız və işiniz bitdi. Replikasiyada bütün çətinliklər təkrarlanan məlumatlara edilən *dəyişikliklərin* idarə edilməsindədir. Biz nodelar arasında dəyişiklikləri təkrarlamaq üçün üç məşhur alqoritmi nəzərdən keçirəcəyik: *tək lider, çox lider və lidersiz replikasiya*.

Demək olar ki, bütün paylanmış verilənlər bazaları bu üç yanaşmadan birini istifadə edir. Onların hamısının müxtəlif müsbət və mənfi cəhətləri və istifadəsi nəzərdə tutulan hallar var, biz onları ətraflı araşdıracağıq.

Replikasiyada nəzərə alınmalı bir çox güzəştlər var: məsələn, sinxron və ya asinxron replikasiyadan istifadə etmək və uğursuz replikaları necə idarə etmək [10]. Bunlar çox vaxt verilənlər bazalarında konfiqurasiya seçimləridir və təfərrüatlar verilənlər bazasına görə fərqlənsə də, ümumi prinsiplər bir çox müxtəlif tətbiqlərdə oxşardır. Biz bu fəsildə bu cür seçimlərin nəticələrini müzakirə edəcəyik.

Verilənlər bazalarının replikasiyası köhnə bir mövzudur - prinsiplər 1970-ci illərdə öyrənildikdən sonra çox dəyişməmişdir, çünki şəbəkələrin əsas məhdudiyyətləri eyni qalmışdır. Bununla belə, tədqiqat xaricində bir çox tərtibatçı uzun müddət verilənlər bazasının yalnız bir qovşaqdan vəya nodedan ibarət olduğunu düşünməyə davam etdi. Paylanmış verilənlər bazalarının əsas istifadəsi daha yenidir. Bir çox proqram tərtibatçıları bu sahədə yeni olduğundan, son ardıcıllıq(*eventual consistency*) kimi məsələlər ətrafında çoxlu anlaşılmazlıqlar var.

Replikasiya üsullarini müzakirə etmədən öncə bir neçə anlayışları başa düşmək lazımdır.

Liderlər və İzləyicilər (ing. Leaders/Followers)

Verilənlər bir surətini saxlayan hər node replika adlanır. Çoxsaylı replikalar olduqda belə bir sual yaranır:

Bütün məlumatların bütün replikalarda eyni olmasından necə əmin ola bilərik?

Verilənlər bazasına hər yazma əməliyyatə hər replika tərəfindən işlənmə vəya proses edilməlidir; əks halda, replikalar artıq eyni məlumatları özündə ehtiva etməyəcək və inconsistency səbəb olacaq.

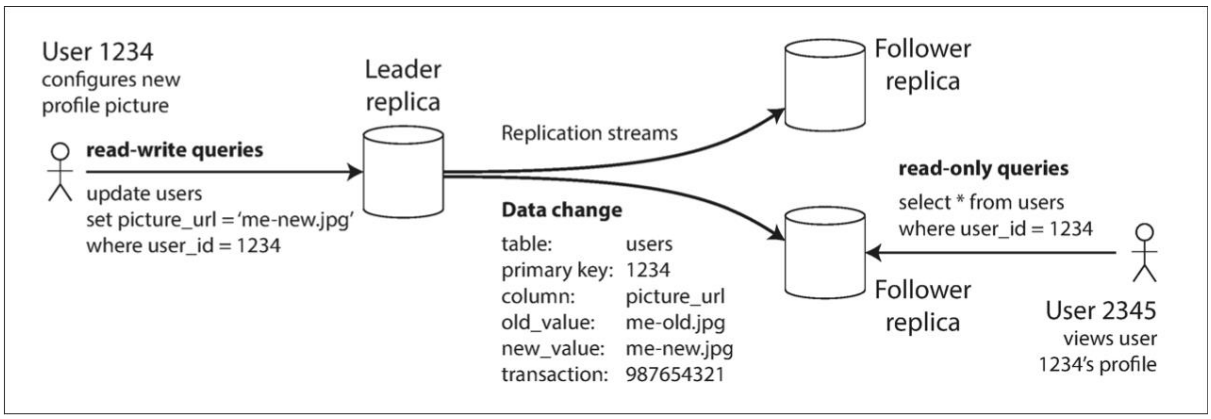
Bunun üçün ən ümumi həll liderə əsaslanan replikasiya adlanır (həmçinin aktiv/passiv və ya master-slave replikasiyası kimi də tanınır).

1. Replikalardan biri lider təyin edilir (həmçinin əsas və ya master kimi tanınır). Kliyentlər verilənlər bazasına məlumat yazmaq istədikdə, sorğularını liderə göndərməlidirlər, o, ilk növbədə yeni məlumatları lokal daimi yaddaşına yazır.

2. Digər replikalar izləyicilər kimi tanınır (oxuma replikaları, slave və ya ikincilər). Lider öz lokal daimi yaddaşına yeni məlumat yazdıqda, o, həmçinin replikasiyanın bir hissəsi kimi məlumat dəyişikliyini bütün izləyicilərinə göndərir. Hər bir izləyici liderdən jurnalı(log məlumatı) götürür və bütün rekordları liderdə işləndiyi ardıcıllıqla tətbiq edərək verilənlər bazasının öz lokaldakı surətini yeniləyir.

3. Klyient verilənlər bazasından oxumaq istədikdə ya liderə, ya da hər hansı bir izləyiciyə sorğu verə bilər.

Qeyd edək ki, kliyent yazma əməliyyatları lider mode üzərindən, oxuma əməliyyatları isə izləyici nodelar tərəfindən icra olunur.



Şəkil 2.1. Replikasiya axımı

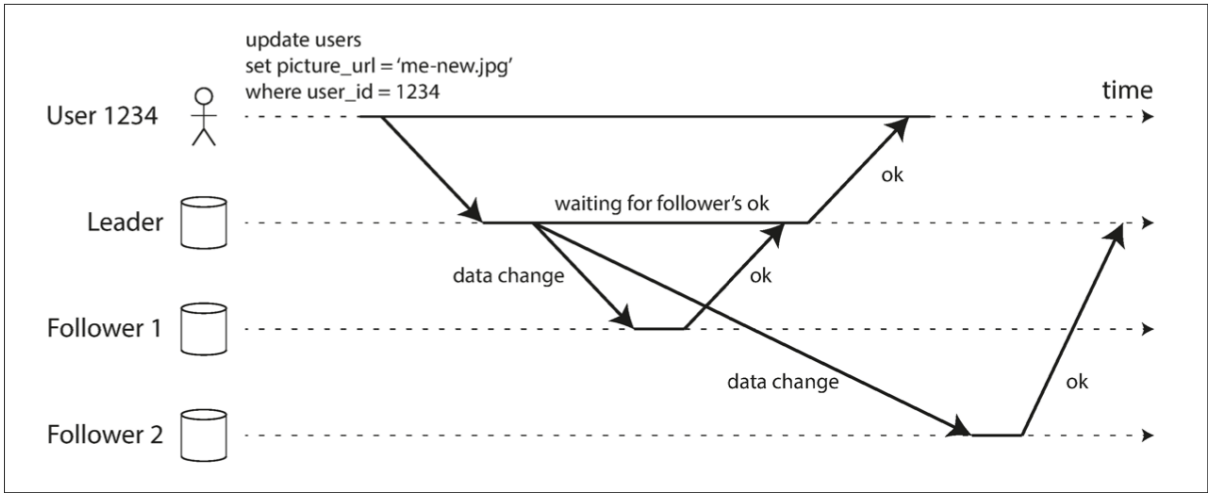
Bu replikasiya rejimi PostgreSQL (9.0 versiyasından bəri), MySQL, Oracle Data Guard və SQL Serverin AlwaysOn Availability Groups kimi bir çox əlaqəli verilənlər bazalarının daxili xüsusiyyətidir [11]. O, həmçinin MongoDB, RethinkDB və Espresso daxil olmaqla bəzi qeyri-relational verilənlər bazalarında istifadə olunur.

Nəhayət, liderə əsaslanan replikasiya yalnız verilənlər bazası ilə məhdudlaşmır: Kafka və RabbitMQ yüksək əlçatan növbələr kimi paylanmış mesaj brokerləri də ondan istifadə edir.

2.2. Replikasiya formalarının tədqiqi. Sinxron və Asinxron

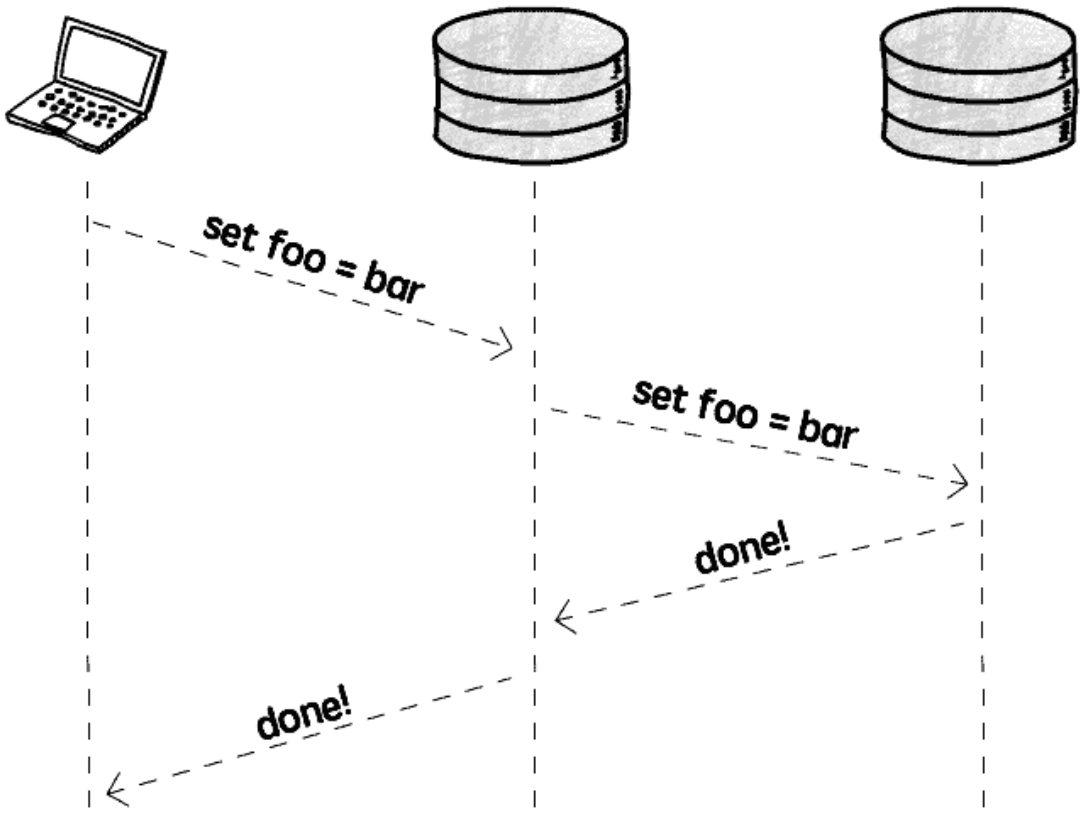
Replikasiya olunan sistemin mühüm detalı replikasiyanın sinxron və ya asinxron baş verməsidir. Relyasiyalı verilənlər bazalarında bu, çox vaxt konfiqurasiya edilə bilən bir seçimdir; digər sistemlər çox vaxt ya biri və ya digəri olmaq üçün kodlaşdırılır. Nümunə: Veb sayt istifadəçisinin profil şəklini yenilədiyi Şəkil 5-1-də nə baş verdiyini düşünün. Kliyent müəyyən vaxtda liderə yeniləmə sorğusu göndərir; bir müddət sonra lider tərəfindən qəbul edilir. Bir nöqtədə lider məlumat dəyişikliyini izləyicilərə ötürür. Nəhayət, lider müştəriyə yeniləmənin uğurlu olduğunu bildirir.

Şəkil-2.2-də sistemin müxtəlif komponentləri arasında əlaqəni göstərir: istifadəçinin müştərisi, lider və iki izləyicisi. Zaman soldan sağa axır. Sorğu və ya cavab mesajı qalın ox kimi göstərilir.



Şəkil 2.2. Bir sinxron və bir asinxron izləyici ilə lider əsaslı replikasiya

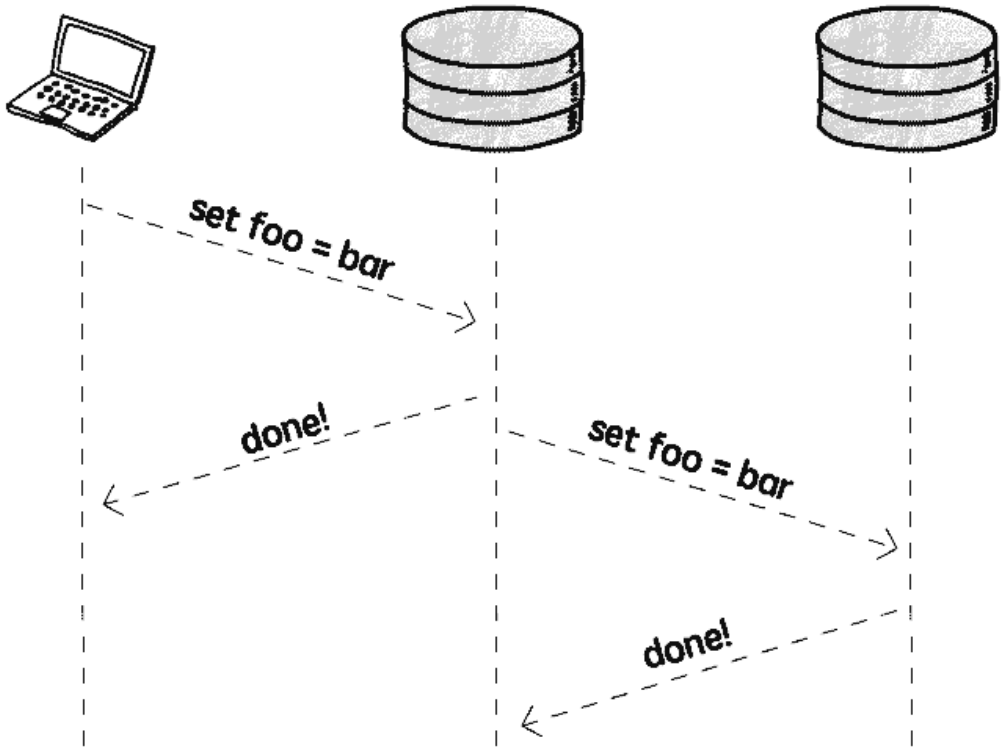
Şəkil 2.2 misalında, *izləyici 1-ə* replikasiya sinxrondur: lider istifadəçiyə uğur barədə məlumat verməzdən əvvəl və yazının digər müştərilərə görünməsi üçün *1-ci izləyicinin* yazını qəbul etdiyini təsdiqləyənə qədər gözləyir. *2-ci izləyiciyə* təkrarlama asinxrondur: lider mesaj göndərir, lakin izləyicidən cavab gözləmir. Diaqram göstərir ki, *izləyici 2* mesajı emal etməzdən əvvəl xeyli gecikmə var. Normalda replikasiya olduqca sürətlidir: əksər verilənlər bazası sistemləri dəyişiklikləri izləyicilərə bir saniyədən az müddətdə tətbiq edir. Ancaq bunun nə qədər davam edəcəyinə dair heç bir zəmanət yoxdur. İzləyicilərin liderdən bir neçə dəqiqə və ya daha çox geri qaldığı hallar var; məsələn, izləyici uğursuzluqdan sağalırsa, sistem maksimum tutuma yaxın işləyirsə və ya qovşaqlar arasında şəbəkə problemləri varsa.



Şəkil 2.2. Sinxron replikasiya nümunə

Sinxron replikasiyanın üstünlüyü ondan ibarətdir ki, izləyiciyə liderlə uyğun gələn məlumatların aktual surətinə malik olacağına zəmanət verilir. Lider birdən uğursuz olarsa, məlumatların hələ də izləyicidə mövcud olduğuna əmin ola bilərik. Dezavantaj odur ki, sinxron izləyici cavab vermirsə (çünki qəzaya uğrayıb və ya şəbəkə nasazlığı var və ya hər hansı başqa səbəbdən), yazma əməliyyatı icra edilə bilməz. Lider bütün yazıları bloklamalı və sinxron replika yenidən mövcud olana qədər gözləməlidir.

Çox vaxt liderə əsaslanan replikasiya tamamilə asinxron olmaq üçün konfiqurasiya edilir. Bu halda, əgər lider uğursuz olarsa və onu bərpa etmək mümkün deyilsə, izləyicilərə hələ replikasiya olmamış rekordlar itirilir. Bu o deməkdir ki, kliyent tərəfindən təsdiqlənmiş olsa belə, yazma əməliyyatının davamlı olacağına zəmanət verilmir. Bununla belə, tam asinxron konfiqurasiyanın üstünlüyü ondan ibarətdir ki, lider bütün izləyiciləri geridə qalsa belə, yazma əməliyyatlarını icra etməyə davam edə bilər.



Şəkil 2.3. Asinxron replikasiya nümunə

Asinxron şəkildə replikasiya olunan sistemlərdə lider uğursuz olarsa məlumatların itirilməsi ciddi problem ola bilər, buna görə də tədqiqatçılar məlumatları itirməyən, lakin yenə də yaxşı performans və əlçatanlıq təmin edən replikasiya üsullarını araşdırmağa davam etdilər.

Məsələn, zəncirvari replikasiya Microsoft Azure Storage kimi bir neçə sistemdə uğurla həyata keçirilmiş sinxron replikasiya variantıdır.

2.3. Replikasiyada olan xüsusi proseslərin tədqiqi

Yeni izləyicilərin(*ing,. follower*) qurulması

Zaman zaman yeni izləyicilər sazlanmasına ehtiyac yaranır - Bu ehtiyac ola bilsin ki, replikaların sayını artırmaq və ya uğursuz nodeları əvəz etmək üçün yaranır.

Və bu zaman bele bir sual yaranır: Yeni sazlanmış izləyicinin liderdəki məlumatlarının dəqiq surətinə malik olmasını necə təmin etmək olar?

Sadəcə məlumat fayllarını bir nodedan digərinə köçürmək adətən kifayət etmir: kliyentlər daim verilənlər bazasına yazır və məlumatlar həmişə axınındadır, buna görə də standart fayl nüsxəsi verilənlər bazasının müxtəlif hissələrini müxtəlif vaxtlarda görə bilər. Nəticənin heç bir mənası olmaya bilər.

Siz verilənlər bazasını kilidləməklə (onu yazmaq üçün əlçatmaz etməklə) diskdəki faylları ardıcıl edə bilərsiniz, lakin bu, yüksək əlçatanlıq məqsədimizə zidd olardı.

Yeni izləyicinin yaradılması və sazlanması adətən fasiləsiz həyata keçirilə bilər. Konseptual olaraq proses belə görünür:

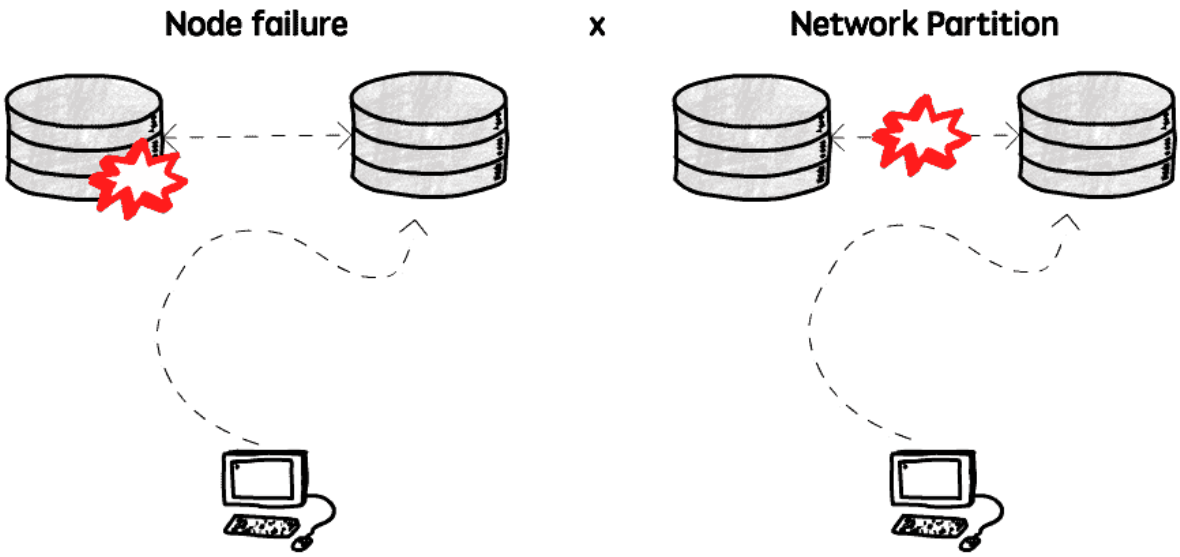
1. Mümkünsə, bütün verilənlər bazasına kilid vurmadan liderin məlumat bazasının snapşotunu çıxardın. Əksər verilənlər bazası bu xüsusiyyətə malikdir, çünki bu, ehtiyat nüsxələri üçün də tələb olunur. Bəzi hallarda üçüncü tərəf alətləri, məsələn, MySQL üçün innobackupex tələb olunur.
2. Snapşotu yeni izləyici qovşağına kopyalayın.
3. İzləyici liderə qoşulur və snapşot alındıqdan sonra baş vermiş bütün məlumat dəyişikliklərini tələb edir. Bu, snapshotın liderin təkrarlama jurnalında dəqiq mövqe ilə əlaqələndirilməsini tələb edir. Bu mövqenin müxtəlif adları var: məsələn, PostgreSQL onu log sıra nömrəsi, MySQL isə binlog koordinatları adlandırır.
4. İzləyici snapşotdan sonra məlumat dəyişikliklərinin geri planını emal etdikdə, onun yetişdiyini deyirik. O, indi liderin məlumat dəyişikliklərini baş verən kimi emal etməyə davam edə bilər.

İzləyicinin yaradılmasının praktiki addımları verilənlər bazasına görə əhəmiyyətli dərəcədə dəyişir. Bəzi sistemlərdə proses tam avtomatlaşdırılıb, digərlərində isə administrator tərəfindən əl ilə yerinə yetirilməli olan bir qədər gizli çox addımlı iş axını ola bilər.

Node nasazlıqların(ing., outages, failures) idarə olunması

Sistemdəki hər hansı bir node, gözlənilmədən nasazlıq səbəbindən, lakin planlaşdırılmış texniki xidmət (məsələn, kernel təhlükəsizlik yamasını quraşdırmaq üçün maşının yenidən işə salınması) səbəbindən fəaliyyət göstərməyə bilər.

Fərdi nodeları fasiləsiz yenidən işə sala bilmək əməliyyatlar və texniki xidmət üçün böyük üstünlükdür. Beləliklə, bizim məqsədimiz fərdi node nasazlıqlarına baxmayaraq sistemi bütövlükdə işlək vəziyyətdə saxlamaq və qovşaq kəsilməsinin təsirini mümkün qədər kiçik saxlamaqdır.



Şəkil 2.4. Node nasazlıqları və şəbəkə sınması

Lider əsaslı replikasiya ilə yüksək əlçatanlığa necə nail olmaq olar?

İzləyici uğursuzluğu: Yaxalanma bərpa metodu (ing., Catch-up recovery)

Yerli diskində hər bir izləyici liderdən aldığı məlumat dəyişikliklərinin jurnalını saxlayır. İzləyici qəzaya uğrayarsa və yenidən işə salınarsa və ya liderlə izləyici arasında şəbəkə müvəqqəti olaraq kəsilirsə, izləyici asanlıqla bərpa oluna bilər: onun jurnalından o, xəta baş verməmişdən əvvəl işlənmiş son əməliyyatı bilir. Beləliklə, izləyici liderə qoşula və izləyicinin əlaqəsi kəsildiyi müddətdə baş verən bütün məlumat dəyişikliklərini tələb edə bilər. Bu dəyişiklikləri tətbiq etdikdə o, liderə çatdı və əvvəlki kimi məlumat dəyişikliyi axını almağa davam edə bilər.

Lider uğursuzluğu: Failover

Liderin uğursuzluğunun öhdəsindən gəlmək daha çətindir: izləyicilərdən biri yeni lider olmaq üçün təklif edilməlidir, kliyentlər öz yazılarını yeni liderə göndərmək üçün yenidən konfiqurasiya edilməlidir və digər izləyicilər yeni liderdən məlumat dəyişikliklərini proses etməyə başlamalıdırlar. Bu proses *failover* adlanır.

Failover manual baş verə bilər (inzibatçıya liderin uğursuz olduğu barədə məlumat verilir və yeni lider təyin etmək üçün lazımi addımlar atır) və ya avtomatik. Avtomatik əvəzetmə prosesi adətən aşağıdakı addımlardan ibarətdir:

1. Liderin uğursuzluğunun müəyyən edilməsi. Potensial olaraq səhv gedə biləcək bir çox şey var: qəzalar, elektrik kəsilməsi, şəbəkə problemləri və s. Nəyin səhv getdiyini aşkar etməyin heç bir qüsursuz yolu yoxdur, ona görə də əksər sistemlər sadəcə olaraq vaxt aşımından istifadə edirlər: qovşaqlar tez-tez mesajları bir-birləri arasında irəli və geri qaytarır və əgər qovşaq müəyyən müddət ərzində cavab vermirsə, məsələn, 30 saniyə— öldüyü güman edilir. (Lider planlı təmir üçün bilərəkdən işdən çıxarılıbsa, bu tətbiq edilmir.)

2. Yeni liderin seçilməsi. Bu, seçki prosesi vasitəsilə həyata keçirilə bilər (burada lider qalan replikaların əksəriyyəti tərəfindən seçilir) və ya yeni lider əvvəllər seçilmiş nəzarətçi qovşağı tərəfindən təyin edilə bilər. Rəhbərliyə ən yaxşı namizəd adətən köhnə liderdən ən müasir məlumat dəyişiklikləri olan replikadır (hər hansı məlumat itkisini minimuma endirmək üçün). Bütün qovşaqların yeni lider üzərində razılığa gəlməs konsensus problemidir.

3. Yeni liderdən istifadə etmək üçün sistemin yenidən konfiqurasiyası. Kliyentlər indi öz yazma sorğularını yeni liderə göndərməlidirlər. Əgər köhnə lider geri qayıdarsa, o, hələ də onun lider olduğuna inana bilər, digər replikaların onu istefa verməyə məcbur etdiyini dərk etmir. Sistem köhnə liderin davamçı olmasını və yeni lideri tanımasını təmin etməlidir.

Failover zamanı səhv ola biləcək şeylərlə vardır:

- Asinxron replikasiya istifadə edilərsə, yeni lider uğursuzluqdan əvvəl köhnə liderdən bütün yazıları almaya bilər. Əgər keçmiş lider yeni lider seçildikdən sonra yenidən qrupa daxil olarsa, o yazıların aqibəti necə olmalıdır? Yeni lider bu arada ziddiyyətli yazılar almış ola bilər. Ən ümumi həll köhnə liderin təkrarlanmayan yazılarının sadəcə silinməsidir ki, bu da müştərilərin davamlılıq gözləntilərini poza bilər.
- Verilənlər bazasından kənarda olan digər saxlama sistemləri verilənlər bazası məzmunu ilə əlaqələndirilməlidirsə, yazıların ləğvi xüsusilə təhlükəlidir. Məsələn, GitHub-da bir hadisədə köhnəlmiş MySQL izləyicisi liderliyə yüksəldi. Verilənlər bazası yeni sətirlərə əsas açarları təyin etmək üçün avtomatik artırma sayğacından istifadə edirdi, lakin yeni liderin sayğacı köhnə

liderdən geri qaldığı üçün o, əvvəllər köhnə lider tərəfindən təyin edilmiş bəzi əsas açarlardan yenidən istifadə etdi. Bu əsas açarlar Redis mağazasında da istifadə olunurdu, buna görə də əsas açarların təkrar istifadəsi MySQL və Redis arasında uyğunsuzluqla nəticələndi və bu da bəzi şəxsi məlumatların yanlış istifadəçilərə açıqlanmasına səbəb oldu.

- Müəyyən nasazlıq ssenarilərində iki node-un hər ikisinin lider olduqlarına inanması baş verə bilər. Bu vəziyyət bölünmüş beyin adlanır və bu təhlükəlidir: əgər hər iki lider yazıları qəbul edərsə və konfliktləri həll etmək üçün heç bir proses yoxdursa, məlumatların itirilməsi və ya korlanmış. Təhlükəsizliyin qarşısını almaq üçün bəzi sistemlərdə iki lider aşkar edilərsə, bir nodu bağlamaq mexanizmi var.
- Liderin öldüyü elan edilməzdən əvvəl düzgün fasilə nədir? Daha uzun fasilə, liderin uğursuz olduğu halda bərpa üçün daha uzun müddət deməkdir. Lakin, vaxt aşımı çox qısa olarsa, lazımsız uğursuzluqlar ola bilər. Məsələn, müvəqqəti yük artımı qovşağın cavab vaxtının vaxt aşımından çox artmasına və ya şəbəkə nasazlığı paketlərin gecikməsinə səbəb ola bilər. Əgər sistem artıq yüksək yüklə və ya şəbəkə problemləri ilə mübarizə aparırsa, lazımsız uğursuzluq vəziyyəti yaxşılaşdırmağa yox, daha da pisləşdirə bilər.

Bu problemlər - node uğursuzluqları; etibarsız şəbəkələr; və replika ardıcıllığı, davamlılıq, əlçatanlıq və gecikmə ilə bağlı mübadilələr - əslində paylanmış sistemlərdə əsas problemlərdir.

Bu problemlərin asan həlli yoxdur. Bu səbəbdən, bəzi əməliyyat qrupları, proqram təminatının avtomatik işləməyi dəstəkləməsinə baxmayaraq, uğursuzluqları əl ilə yerinə yetirməyə üstünlük verirlər.

2.4. Replikasiya logların həyata keçirilmə üsullarının analizi

Başlıq altında liderə əsaslanan replikasiya necə işləyir? Təcrübədə bir neçə fərqli təkrarlama metodu istifadə olunur, ona görə də hər birinə qısaca nəzər salaq.

Statement əsaslı replikasiya(*ing., Statement-based replication*)

Ən sadə halda, lider icra etdiyi hər bir yazı sorğusunu (bəyanatını) qeyd edir və həmin bəyanat jurnalını öz izləyicilərinə göndərir. Əlaqəli verilənlər bazası üçün bu o deməkdir ki, hər INSERT, UPDATE və ya DELETE bəyanatı izləyicilərə yönləndirilir və hər bir izləyici həmin SQL ifadəsini sanki müştəridən alınmış kimi təhlil edir və icra edir.

Bu ağlabatan səslənsə də, replikasiyaya bu yanaşmanın pozulmasının müxtəlif yolları var:

- Cari tarix və vaxtı əldə etmək üçün NOW() və ya təsadüfi nömrə əldə etmək üçün RAND() kimi qeyri-müəyyən funksiyanı çağıran istənilən ifadə, çox güman ki, hər replikada fərqli dəyər yarada bilər.
- Əgər ifadələr avtomatik artım sütunundan istifadə edirsə və ya verilənlər bazasındakı mövcud verilənlərdən asılıdırsa (məsələn, UPDATE ... WHERE <condititions>), onlar hər bir replikada tam olaraq eyni ardıcıllıqla yerinə yetirilməlidir, əks halda onlar fərqli təsir göstərir. Birdən çox eyni vaxtda icra edilən əməliyyatlar olduqda bu məhdudlaşdırıla bilər.
- Yan təsirləri olan ifadələr (məsələn, triggerlər, saxlanılan prosedurlar, istifadəçi tərəfindən müəyyən edilmiş funksiyalar) yan təsirlər tamamilə deterministik olmadıqda, hər bir replikada fərqli yan təsirlərin baş verməsi ilə nəticələnə bilər.

Bu məsələlərin ətrafında işləmək mümkündür - məsələn, lider hər hansı qeyri-müəyyən funksiya çağırışlarını ifadə daxil olduqda sabit qaytarma dəyəri ilə əvəz edə bilər ki, izləyicilərin hamısı eyni dəyəri əldə etsin. Bununla belə, çox sayda kənar hallar olduğundan, indi başqa replikasiya üsullarına üstünlük verilir.

Statement əsaslı replikasiya 5.1 versiyasından əvvəl MySQL-də istifadə edilmişdir. O, kifayət qədər yığcam olduğu üçün bu gün də bəzən istifadə olunur, lakin ifadədə qeyri-determinizm varsa, defolt olaraq MySQL indi sətir əsaslı(row-based) replikasiyaya keçir (qısaca müzakirə olunacaq). VoltDB bəyanata(statement-based) əsaslanan replikasiyadan istifadə edir və əməliyyatların deterministik olmasını tələb etməklə onu təhlükəsiz edir [4].

Write-ahead log (WAL) shipping

Log verilənlər bazasına bütün yazıları ehtiva edən baytların yalnız əlavə olunan ardıcıllığıdır. Biz başqa bir node üzərində replika yaratmaq üçün eyni jurnaldan istifadə edə bilərik: jurnalı diskə yazmaqla yanaşı, lider onu şəbəkə vasitəsilə öz izləyicilərinə də göndərir.İzləyici bu jurnalı emal etdikdə, liderdə olduğu kimi eyni məlumat strukturlarının surətini yaradır.

Bu replikasiya üsulu digərləri arasında PostgreSQL və Oracle-da istifadə olunur. Əsas çatışmazlıq jurnalın məlumatları çox aşağı səviyyədə təsvir etməsidir: WAL hansı disk bloklarında hansı baytların dəyişdirildiyi barədə təfərrüatları ehtiva edir [9]. Bu, təkrarlamanı yaddaş mühərriki ilə sıx əlaqələndirir. Verilənlər bazası saxlama formatını bir versiyadan digərinə dəyişirsə, lider və izləyicilərdə verilənlər bazası proqram təminatının müxtəlif versiyalarını işə salmaq adətən mümkün deyil.

Bu kiçik bir icra detalı kimi görünə bilər, lakin böyük əməliyyat təsiri ola bilər.

Replikasiya protokolu izləyiciyə liderdən daha yeni proqram təminatı versiyasından istifadə etməyə imkan verirsə, siz əvvəlcə ardıcılları təkmilləşdirməklə və sonra təkmilləşdirilmiş qovşaqlardan birini yeni lider etmək üçün əvəzetmə yerinə yetirməklə verilənlər bazası proqram təminatının sıfır fasiləsiz təkmilləşdirməsini həyata keçirə bilərsiniz. Əgər təkrarlama protokolu bu versiyanın uyğunsuzluğuna imkan vermirsə, çox vaxt WAL göndərişində olduğu kimi, bu cür təkmilləşdirmələr dayanma müddəti tələb edir.

Məntiqi (sətir əsaslı) log replikasiyası

Alternativ replikasiya üçün və saxlama mühərriki üçün müxtəlif jurnal formatlarından istifadə etməkdir ki, bu da replikasiya jurnalını yaddaş mühərrikinin daxili hissələrindən ayırmağa imkan verir. Bu cür təkrarlama jurnalı onu yaddaş mühərrikinin (fiziki) məlumat təqdimatından fərqləndirmək üçün məntiqi jurnal adlanır.

Əlaqəli verilənlər bazası üçün məntiqi jurnal, adətən, verilənlər bazası cədvəllərinə bir sıra qranularlığında yazılanları təsvir edən qeydlər ardıcıllığıdır:

- Daxil edilmiş sətir üçün jurnal bütün sütunların yeni qiymətlərini ehtiva edir.

- Silinmiş sətir üçün jurnal silinmiş sıranı unikal şəkildə müəyyən etmək üçün kifayət qədər məlumatı ehtiva edir. Tipik olaraq bu, əsas açardır, lakin cədvəldə əsas açar yoxdursa, bütün sütunların köhnə dəyərləri qeyd edilməlidir.
- Yenilənmiş sətir üçün jurnal yenilənmiş sətiri və bütün sütunların yeni dəyərlərini (və ya ən azı dəyişmiş bütün sütunların yeni dəyərlərini) unikal şəkildə müəyyən etmək üçün kifayət qədər məlumatı ehtiva edir.

Bir neçə sıranı dəyişdirən əməliyyat bir neçə belə jurnal qeydini yaradır, ardınca əməliyyatın həyata keçirildiyini göstərən qeyd. MySQL-in binlogu (sətir əsaslı replikasiyadan istifadə etmək üçün konfiqurasiya edildikdə) bu yanaşmadan istifadə edir.

Məntiqi jurnal yaddaş mühərrikinin daxili hissələrindən ayrıldığından, o, daha asanlıqla geriyə uyğun saxlanıla bilər ki, bu da liderə və izləyiciyə verilənlər bazası proqram təminatının müxtəlif versiyalarını və ya hətta müxtəlif yaddaş mühərriklərini işə salmağa imkan verir [13].

Tətik əsaslı replikasiya(ing.,Trigger-based replication)

İndiyə qədər təsvir edilən təkrarlama yanaşmaları heç bir proqram kodu daxil edilmədən verilənlər bazası sistemi tərəfindən həyata keçirilir. Bir çox hallarda, istədiyiniz budur - lakin daha çox çevikliyin lazım olduğu bəzi hallar var. Məsələn, yalnız məlumatların alt çoxluğunu təkrarlamaq istəyirsinizsə və ya bir verilənlər bazasından digərinə təkrarlamaq istəyirsinizsə və ya konfliktin həlli məntiqinə ehtiyacınız varsa, onda sizə lazım ola bilər replikasiyanı tətbiq səviyyəsinə daşımaq üçün.

Bəzi alətlər, məsələn, Oracle GoldenGate verilənlər bazası jurnalını oxumaqla proqramda məlumat dəyişikliklərini əlçatan edə bilər. Alternativ olaraq, bir çox əlaqəli verilənlər bazasında mövcud olan funksiyalardan istifadə etmək olar: tetikleyiciler və saxlanılan prosedurlar.

Tətik, verilənlər bazası sistemində məlumat dəyişikliyi (yazma əməliyyatı) baş verdikdə avtomatik icra edilən xüsusi proqram kodunu qeydiyyatdan keçirməyə imkan

verir. Tətik bu dəyişikliyi xarici proses tərəfindən oxuna bilən ayrı bir cədvələ daxil etmək imkanına malikdir. Həmin xarici proses sonra hər hansı zəruri tətbiq məntiqini tətbiq edə və məlumat dəyişikliyini başqa sistemə təkrarlaya bilər. Məsələn, Oracle üçün Databus və Postgres üçün Bucardo belə işləyir.

Tətik əsaslı replikasiya adətən digər replikasiya üsullarına nisbətən daha çox məsrəflərə malikdir və verilənlər bazasının daxili replikasiyası ilə müqayisədə səhvlərə və məhdudiyyətlərə daha çox meyllidir. Bununla belə, elastikliyinə görə faydalı ola bilər.

2.5. Replikasiya üsullarının növləri

Tək lider replikasiya (ing., Single-Leader Replication)

Paylanmış hesablama sistemlərində məlumatların təkrarlanması(ing., data replication) məlumatların əlçatanlığını, etibarlılığını və miqyasını artırmaq üçün ümumi bir texnikadır. Tək lider replikasiyası elə məlumatların təkrarlanması növüdür ki, burada lider adlanan bir qovşaq bütün yazma sorğularını qəbul etmək və emal etmək üçün cavabdehdir, izləyicilər və ya replikalar adlanan digər qovşaqlar isə liderdən məlumatları köçürür və oxunmuş sorğulara xidmət edir. Bu bölmədə biz tək lider replikasiyasının necə işlədiyini təsvir edəcəyik və bu yanaşmadan istifadə edən müasir sistemlərdən bəzi nümunələr təqdim edəcəyik.

Tək Lider Replikasiyası necə işləyir?

Tək lider replikasiyası, lider qovşağının usta(*ing, master*), izləyici qovşaqlarının isə kölə(*ing, slave*) olduğu master-slave arxitekturasıdır.

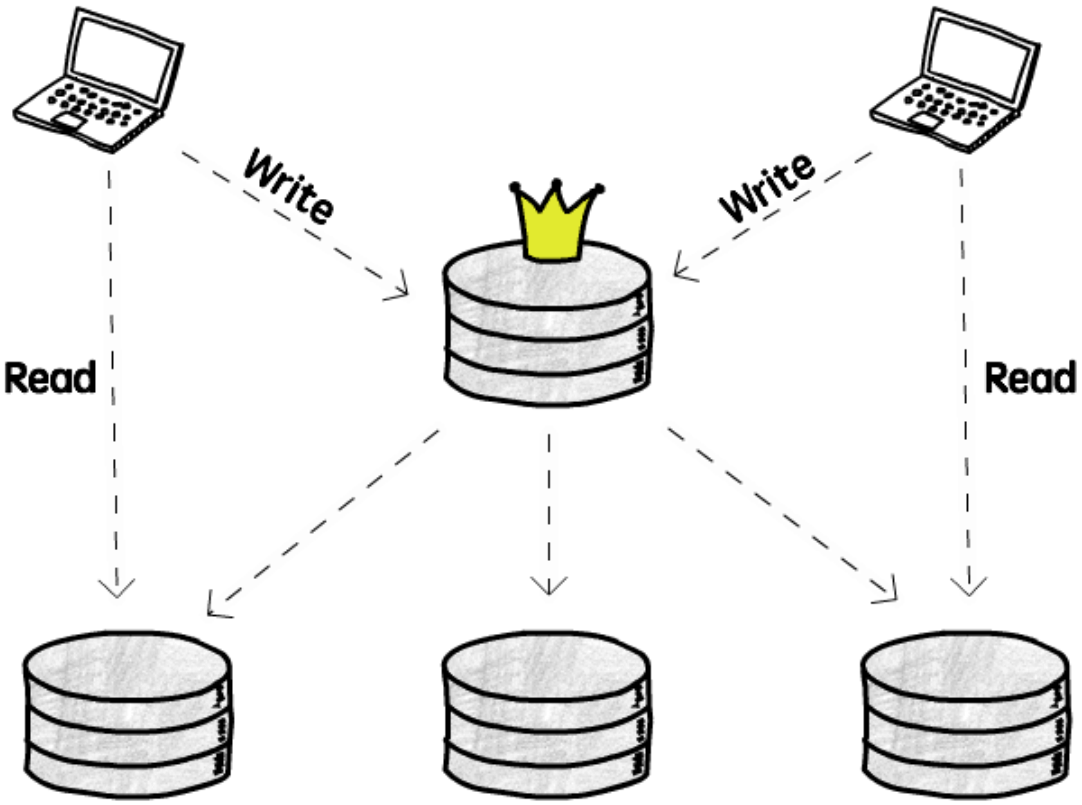
Nodelardan birinin lider (və ya master) seçilməsi lazımdır. Bu proses verilənlər bazası inzibatçısı(*ing., DBA*) tərəfindən əl ilə edilə bilər və ya bir çox hallarda bu, konsensus alqoritmləri vasitəsilə avtomatik olaraq edilə bilər.

Tək lider replikasiya ideyası ondan ibarətdir ki, kliyentdən gələn bütün yazma sorğular ilk növbədə liderin lokal davamlı yaddaşına yazılmalıdır [5].

Digər nodelar isə oxuma replikalar (və ya izləyicilər) kimi qurulur. Lider məlumatı lokal davamlı yaddaşına yazdıqdan sonra, replikasiya jurnalının bir hissəsi kimi

məlumat dəyişikliklərini bütün izləyicilərə göndərir. Hər bir izləyici daha sonra lokal daimi yaddaşın lokal nüsxəsini yeniləyir və liderdəki dəyişikliklərin ardıcıl olaraq özündə icrasının təmin edir.

Tək liderin təkrarlanması yüksək əlçatanlıq, xətaya dözümlülük və miqyaslılıq kimi bir sıra üstünlüklər təmin edir. Bu, bir və ya daha çox izləyici qovşağı uğursuz olsa belə, sistemin işləməyə davam etməsinə imkan verir və çoxlu sayda oxuma sorğularını idarə edə bilən genişlənə bilən bir arxitektura təmin edir [5].



Şəkil 2.3. Tək liderli replikasiya prosesi

Kliyent məlumatları oxumaq istədikdə ya liderə, ya da izləyicilərə sorğu verə bilər. Bu rejimi istifadə edən sistemlərə misal olaraq relyasiyalı verilənlər bazaları (MySQL, PostgreSQL, Oracle, SQL Server), bəzi qeyri-relational verilənlər bazaları (MongoDB) və mesaj brokerləri (Kafka, RabbitMQ) daxildir. Aşağıda həmin sistemləri nəzərdən keçirək.

- *Apache Kafka:* Kafka paylanmış pub-sub mesajlaşma sistemindən istifadə edir. Məlumatların hər bir bölməsi üçün bir liderdən istifadə edir. Bütün yazılar liderə

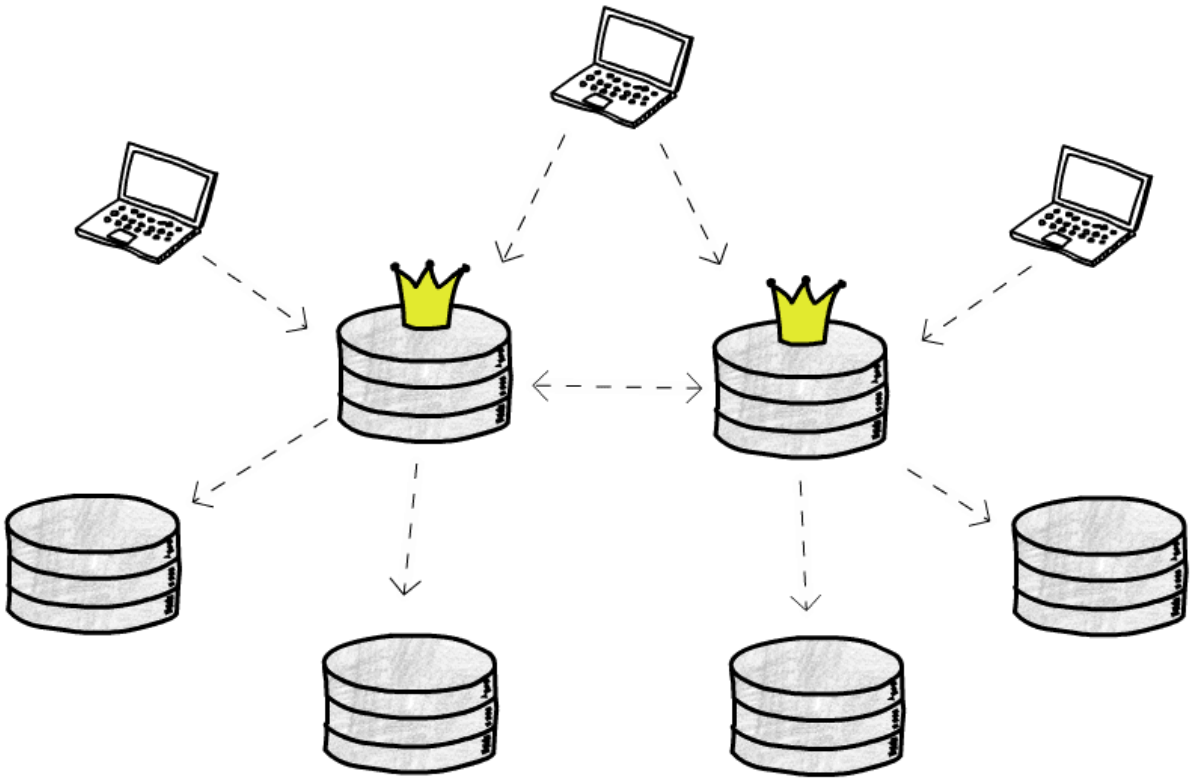
yönəldilir və izləyicilər liderin məlumatlarını təkrarlayır. Lider uğursuz olarsa, ardıcıllardan biri yeni lider olmağa yüksəlir [12].

- *MySQL və Postgres*: Həm MySQL, həm də Postgres yüksək əlçatanlıq və fəlakətin bərpası üçün tək lider replikasiyasından istifadə edir. MySQL-də buna master-slave replikasiyası, Postgres-də isə axın replikasiyası deyilir. Hər iki halda, əsas qovşaq bütün yazıları qəbul edir və izləyivi qovşaqlar masterdən məlumatları təkrarlayır. Usta uğursuzluq halında, izləyicilərdən biri yeni lider olmaq üçün yüksəldilir.
- *MongoDB:* MongoDB həmçinin yüksək əlçatanlıq və fəlakətin bərpası üçün tək lider replikasiyasından istifadə edir. O, birincil-ikincili təkrarlama modelindən istifadə edir, burada əsas qovşaq bütün yazıları qəbul edir, ikincili qovşaqlar isə məlumatları birincidən təkrarlayır. Birincili uğursuzluq halında, ikinci dərəcəlilərdən biri yeni birinciliyə yüksəlir.

Ümumilikdə, tək lider replikasiyası məlumatların ardıcıllığını və mövcudluğunu təmin etmək üçün müasir sistemlərdə geniş istifadə olunan bir texnikadır.

Çox lider replikasiya (ing., Multi-Leader Replication)

Multi-lider replikasiyası paylanmış sistemlərdə istifadə edilən bir texnikadır ki, bu da çoxlu qovşaqların eyni vaxtda yazı və yeniləmələri qəbul etməsinə imkan verir. Bu yanaşmada tək lider və ya master node yoxdur və hər bir qovşaq lider rolunu oynaya və yenilikləri digər qovşaqlara təkrarlaya bilər.



Şəkil 2.4. Çox liderli replikasiya prosesi

Çox Lider Replikasiya necə işləyir

Çox liderli təkrarlamada klasterdəki hər bir qovşaq öz məlumatlarına cavabdehdir və müstəqil olaraq yazıları və yeniləmələri qəbul edə bilər. Yazı sistemə təqdim edildikdə, klasterdəki bütün qovşaqlara təkrarlanır. Daha sonra hər bir qovşaq yazını məlumatların öz nüsxəsinə tətbiq edir və yeniləməni klasterdəki digər qovşaqlara təkrarlayır.

Ardıcıllığı təmin etmək üçün sistem ziddiyyətli yeniləmələri həll etmək üçün vektor saatları və ya vaxt ştampları kimi münaqişələrin həlli mexanizmlərindən istifadə edir. Bundan əlavə, çox-lider replikasiya tənzimlənə bilən ardıcıllıq səviyyələrini təmin edə bilər ki, bu da tətbiqlərə ehtiyaclarına ən yaxşı uyğun gələn ardıcıllıq səviyyəsini seçməyə imkan verir.

Çox Lider Replikasiyasının müsbət və mənfi tərəfləri

Çox liderli replikasiyanın əsas üstünlüklərindən biri yüksək əlçatanlıq və nasazlıqlara qarşı dözümlülük təmin etmək qabiliyyətidir. Tək bir uğursuzluq nöqtəsi

olmadığından, bəzi qovşaqlar uğursuz olsa və ya əlçatmaz olsa belə, sistem işləməyə davam edə bilər. Əlavə olaraq, tək lider qovşağının yaratdığı darboğaz olmadığı üçün çox liderli təkrarlama performansı və miqyaslılığı yaxşılaşdıra bilər.

Bununla belə, çox liderli replikasiyanın da bəzi çətinlikləri var. Əsas problemlərdən biri münaqişələri idarə etmək və məlumatların bütün qovşaqlarda ardıcıl olmasını təmin etməkdir. Bəzi hallarda, ziddiyyətli yeniləmələrin həlli üçün əl ilə müdaxilə tələb oluna bilər ki, bu da vaxt və resurs tələb edə bilər.

Çox lider replikasiyanın başqa bir problemi bütün qovşaqlarda sinxronizasiyanı idarə etməkdir. Hər bir qovşaq öz məlumatlarına cavabdeh olduğundan, məlumatların bütün qovşaqlarda ardıcıl olmasını təmin etmək üçün sistemdə mexanizmlər olmalıdır. Bu, xüsusilə yüksək yazma həcmi və tez-tez yeniləmələri olan sistemlərdə çətin ola bilər.

Çox Lider Replikasiyadan istifadə edən müasir sistemlər

Bir neçə müasir sistem yüksək əlçatanlıq və nasazlığa dözümlülüyü təmin etmək üçün çoxlu lider replikasiyasından istifadə edir. Budur bir neçə nümunə:

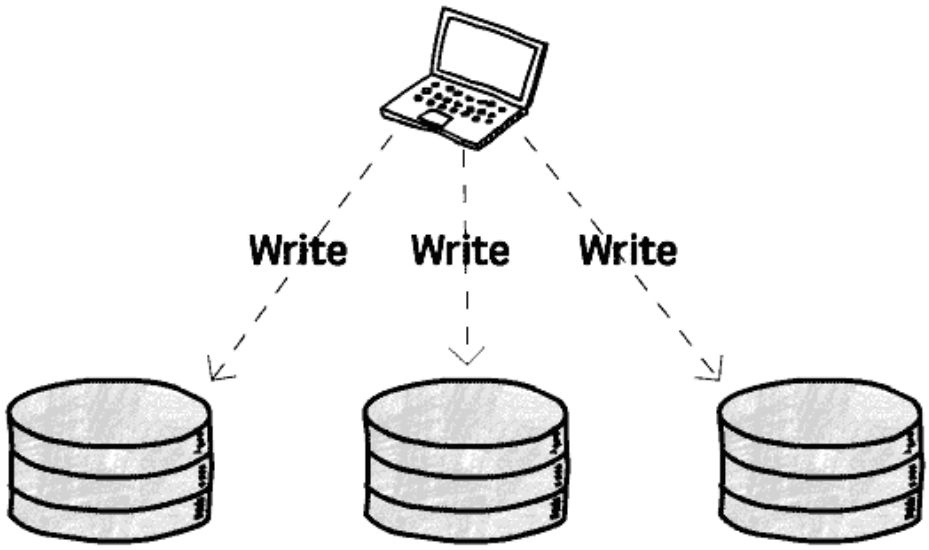
- *Apache CouchDB*: Apache CouchDB, çox lider replikasiyasını dəstəkləyən paylanmış sənəd yönümlü verilənlər bazası sistemidir. CouchDB-də çoxluqdakı hər bir qovşaq yazıları və yeniləmələri qəbul edə bilər və bu dəyişikliklər klasterdəki digər qovşaqlara asinxron şəkildə yayılır. Bu yanaşma yüksək yazma qabiliyyətinə və təkmil xətaya dözümlülüyünə imkan verir, çünki tək bir uğursuzluq nöqtəsi yoxdur. Bundan əlavə, sistem paralel yeniləmələr səbəbindən yarana biləcək münaqişələri idarə etmək üçün konfiqurasiya edilə bilər.
- *CockroachDB*: CockroachDB yüksək əlçatanlıq və miqyaslılığı təmin etmək üçün çox lider replikasiyasından istifadə edən paylanmış SQL verilənlər bazasıdır. CockroachDB-də klasterdəki hər bir qovşaq lider kimi çıxış edə və yazı sorğularını qəbul edə bilər. Bu dəyişikliklər daha sonra klasterdəki digər qovşaqlara asinxron şəkildə təkrarlanır. Sistem həm də müştərilərin hər zaman ən aktual məlumatları görməsini təmin edərək güclü ardıcıllıq zəmanəti verir.

- *Google Cloud Spanner*: Google Cloud Spanner yüksək əlçatanlıq və ardıcıllıq təmin etmək üçün çox lider replikasiyasından istifadə edən qlobal şəkildə paylanmış əlaqəli verilənlər bazası sistemidir. Spanner-də çoxlu qovşaqlar lider kimi çıxış edə və yazı sorğularını qəbul edə bilər. Bu dəyişikliklər daha sonra klasterdəki digər qovşaqlara asinxron şəkildə təkrarlanır. Spanner həmçinin güclü ardıcıllıq təminatları təqdim edərək, onu kritik missiya tətbiqləri yaratmaq üçün məşhur seçimə çevirir.

Ümumilikdə, çox liderli replikasiya paylanmış sistemlərdə yüksək əlçatanlığı, miqyaslılığı və nasazlığa dözümlülüyü təmin edə bilən güclü texnikadır. Münaqişələri idarə etmək və bütün qovşaqlarda ardıcıllığı təmin etmək kimi bəzi çətinlikləri olsa da, böyük həcmdə yazı və yeniləmələri idarə etməli olan müasir məlumat sistemlərinin qurulması üçün dəyərli bir vasitə ola bilər.

Lidersiz replikasiya (ing., Leaderless Replication)

Lidersiz replikasiya paylanmış sistemlərdə istifadə edilən, təyin edilmiş lider və ya master node ehtiyacını aradan qaldıran bir texnikadır. Bu yanaşmada, klasterdəki bütün qovşaqlar eyni vaxtda yazıları və yeniləmələri qəbul edə bilər və məlumatlar avtomatik olaraq bütün qovşaqlarda təkrarlanır.



Şəkil 2.5. Lidersiz replikasiya prosesi

Lidersiz Replikasiya necə işləyir

Lidersiz replikasiyada klasterdəki hər bir qovşaq öz məlumatlarına cavabdehdir və müstəqil olaraq yazıları və yeniləmələri qəbul edə bilər. Yazı sistemə təqdim edildikdə, o, bütün qovşaqlara yayımlanır və hər bir qovşaq yazını verilənlərin öz nüsxəsinə tətbiq edir. Ardıcıllığı təmin etmək üçün sistem ziddiyyətli yeniləmələri həll etmək üçün vektor saatları və ya vaxt ştampları kimi münaqişələrin həlli mexanizmlərindən istifadə edir.

Lidersiz Replikasiyanın müsbət və mənfi cəhətləri

Lidersiz replikasiyanın əsas üstünlüklərindən biri yüksək əlçatanlıq və nasazlıqlara qarşı dözümlülük təmin etmək qabiliyyətidir. Tək bir uğursuzluq nöqtəsi olmadığından, bəzi qovşaqlar uğursuz olsa və ya əlçatmaz olsa belə, sistem işləməyə davam edə bilər. Bundan əlavə, lidersiz replikasiya performansı və miqyaslılığı yaxşılaşdıra bilər, çünki tək lider node tərəfindən yaradılan heç bir maneə yoxdur.

Bununla belə, lidersiz təkrarlamanın bəzi çətinlikləri də var. Əsas problemlərdən biri sistemin mürəkkəbliyini idarə etməkdir. Hər bir qovşaq öz məlumatlarına cavabdeh olduğundan, məlumatların bütün qovşaqlarda ardıcıl olmasını təmin etmək üçün sistemdə mexanizmlər olmalıdır. Bu, xüsusilə yüksək yazma həcmi və tez-tez yeniləmələri olan sistemlərdə çətin ola bilər.

Lidersiz replikasiyanın digər problemi münaqişələri idarə etmək və məlumatların bütün qovşaqlarda ardıcıl olmasını təmin etməkdir. Bəzi hallarda, ziddiyyətli yeniləmələrin həlli üçün əl ilə müdaxilə tələb oluna bilər ki, bu da vaxt və resurs tələb edə bilər.

Lidersiz Replikasiyadan istifadə edən Müasir Sistemlər

Bir sıra müasir sistemlər yüksək əlçatanlıq və xətaya dözümlülük təmin etmək üçün lidersiz təkrarlamadan istifadə edir. Budur bir neçə nümunə:

- *Apache Cassandra*: Cassandra, paylanmış mühitlərdə yüksək əlçatanlığa və nasazlığa dözümlülüyünə imkan verən Dinamo tipli replikasiya adlı lidersiz replikasiya modelindən istifadə edir. Cassandra-da məlumatlar avtomatik olaraq bir çox qovşaqda təkrarlanır və ardıcıllığı təmin etmək üçün münaqişələrin həlli mexanizmlərindən istifadə olunur.
- *Riak KV*: Riak KV həmçinin yüksək əlçatanlıq və nasazlığa dözümlülük təmin etmək üçün lidersiz replikasiya modelindən istifadə edir. Riak KV-də məlumatlar bir neçə qovşaqda təkrarlanır və ardıcıllığı təmin etmək üçün münaqişələrin həlli mexanizmlərindən istifadə olunur.
- *Amazon DynamoDB*: DynamoDB yüksək əlçatanlıq və genişlənmə təmin etmək üçün lidersiz replikasiya modelindən istifadə edən idarə olunan NoSQL verilənlər bazası xidmətidir. DynamoDB-də məlumatlar bir neçə qovşaqda təkrarlanır və sistem ardıcıllığı təmin etmək üçün münaqişələrin həlli mexanizmlərindən istifadə edir.

Ümumilikdə, lidersiz replikasiya paylanmış sistemlərdə yüksək əlçatanlıq və nasazlıqlara qarşı dözümlülük təmin edə bilən güclü texnikadır. Münaqişələri idarə etmək və bütün qovşaqlarda ardıcıllığı təmin etmək kimi bəzi çətinlikləri olsa da, böyük həcmdə yazı və yeniləmələri idarə etməli olan müasir məlumat sistemlərinin qurulması üçün dəyərli bir vasitə ola bilər.

2.6. Replikasiya üsullarının müqayisəli tədqiqi

Replikasiya müasir paylanmış məlumat sistemlərində yüksək əlçatanlığı, miqayaslanma və xətaya dözümlülüyü təmin etmək üçün istifadə edilən vacib bir texnikadır. Müxtəlif replikasiya üsulları arasında tək lider replikasiyası, çoxlu lider replikasiyası və lidersiz təkrarlama ən çox istifadə olunur.

Tək Lider Replikasiyası

Tək liderin təkrarlanması, yazıları və yeniləmələri qəbul etmək üçün cavabdeh olan bir təyin edilmiş lider qovşağının olduğu bir təkrarlama üsuludur. Klasterdəki digər qovşaqlar replika rolunu oynayır və lider qovşağından yeniləmələri alır. Tək lider

replikasiyası MySQL, Oracle və PostgreSQL kimi verilənlər bazası sistemlərində geniş istifadə olunur.

Müsbət cəhətləri:

- Güclü ardıcıllıq(Strong consistency): Tək lider replikasiyası güclü ardıcıllıq təminatlarını təmin edir, çünki bütün yazma sorğularını əlaqələndirmək və bütün replikaların yenilənməsini təmin etmək üçün cavabdeh olan tək bir qovşaq var.
- İdarə etmək daha sadədir: Yalnız bir lider olduğundan, tək liderli təkrarlama sistemini idarə etmək çox vaxt daha sadədir və qovşaqlar arasında daha az koordinasiya tələb edir.
- Proqnozlaşdırıla bilən performans: Tək liderli təkrarlama sistemi ilə performans daha proqnozlaşdırıla bilər, çünki çoxsaylı qovşaqlar arasında münaqişələri həll etməyə ehtiyac yoxdur.

Mənfi cəhətləri/Çətinliklər:

- Tək uğursuzluq nöqtəsi: Tək liderli təkrarlamanın bir uğursuz nöqtəsi var, çünki lider node bütün yazma sorğularının əlaqələndirilməsinə cavabdehdir. Lider node uğursuz olarsa, sistem əlçatan ola və ya məlumatları itirə bilər.
- Məhdud yazma ötürmə qabiliyyəti: Yazma sorğularını qəbul edən tək bir qovşaqla, tək liderli təkrarlama sistemində əldə edilə bilən yazma qabiliyyətinə məhdudiyyət var.
- Artan gecikmə: Lider düyün klasterdəki digər qovşaqlardan uzaqda yerləşirsə, yazma sorğuları lider qovşaqına göndərilməli və təkrarlanmalı olduğu üçün artan gecikmə ilə qarşılaşa bilər.

Tək liderin təkrarlanması güclü ardıcıllıq zəmanətləri tələb edən və yüksək yazma qabiliyyəti tələbləri olmayan sistemlər üçün uyğundur. O, həmçinin məlumatların bütövlüyünün vacib olduğu və performansın kritik əhəmiyyət kəsb etmədiyi sistemlər üçün uyğundur.

Çox Lider Replikasiyası

Multi-lider replikasiya bir neçə qovşaqın eyni vaxtda yazıları və yeniləmələri qəbul etməsinə imkan verən təkrarlama üsuludur. Təyin edilmiş lider qovşağı yoxdur və hər bir qovşaq lider rolunu oynaya və yeniləmələri digər qovşaqlara təkrarlaya bilər. Çox liderli replikasiya Apache CouchDB, CockroachDB və Google Cloud Spanner kimi paylanmış məlumat sistemlərində istifadə olunur.

Müsbət cəhətləri:

- Yüksək yazma ötürmə qabiliyyəti: Çox lider replikasiyası çoxlu qovşaqlara yazma sorğularını qəbul etməyə imkan verir və yüksək yazma qabiliyyətinə imkan verir.
- Nasazlığa yüksək dözümlülük: Çox liderli təkrarlama artıqlığı təmin edərək sistemi uğursuzluqlara qarşı daha davamlı edir.
- Azaldılmış gecikmə: İş yükünü çoxlu qovşaqlar arasında paylamaqla, çox rəhbər replikasiya yazma sorğularının gecikməsini azalda bilər.

Mənfi cəhətləri/Çətinliklər:

- Artan mürəkkəblik: Çox liderli təkrarlama sistemə mürəkkəblik əlavə edir, diqqətli koordinasiya və münaqişələrin həlli mexanizmlərini tələb edir.
- Ardıcıllıqla bağlı problemlər: Çoxlu qovşaqlar yazma sorğularını qəbul edə bildiyi üçün, məlumatlarda uyğunsuzluğa səbəb ola biləcək ziddiyyətlərin baş verməsi mümkündür. Münaqişələrin həlli çətin ola bilər və vaxt möhürü sifarişi və ya münaqişənin həlli alqoritmləri kimi əlavə mexanizmlər tələb edə bilər.
- Əməliyyat xərcləri: Çox liderli təkrarlama sisteminin idarə edilməsi konfiqurasiya, monitorinq və texniki xidmət üçün əlavə resurslar tələb edən çətin ola bilər.

Çox liderli replikasiya yüksək əlçatanlıq, xətaya dözümlülük və miqyaslılıq tələb edən sistemlər üçün uyğundur. O, həmçinin yüksək yazma qabiliyyəti tələbləri olan və böyük həcmdə yeniləmələri idarə etməli olan sistemlər üçün uyğundur.

Lidersiz Replikasiya

Lidersiz replikasiya çoxluqdakı hər bir qovşağın müstəqil olaraq yazıları və yeniləmələri qəbul edə bildiyi və təyin edilmiş lider qovşağının olmadığı bir təkrarlama üsuludur. Sistem ardıcıllığı təmin etmək üçün münaqişələrin həlli mexanizmlərindən istifadə edir. Lidersiz təkrarlama, Cassandra və Riak kimi paylanmış məlumat sistemlərində istifadə olunur.

Müsbət cəhətləri:

- Yüksək yazma qabiliyyəti: Yazma sorğularını qəbul edən çoxsaylı qovşaqlarla lidersiz replikasiya yüksək yazma qabiliyyətini təmin edə bilər, xüsusən də çoxlu müştərisi olan sistemlərdə və yazma intensivliyi olan iş yükü.
- Nasazlığa yüksək dözümlülük: Lidersiz təkrarlama artıqlığı təmin edərək sistemi uğursuzluqlara qarşı daha davamlı edir. Bir node uğursuz olarsa, digər qovşaqlar yazma sorğularını emal etməyə davam edə bilər.
- Aşağı gecikmə: Yazma sorğularının koordinasiyasına cavabdeh olan tək bir qovşaq olmadığından, lidersiz replikasiya tək liderli təkrarlama ilə müqayisədə yazma sorğuları üçün daha az gecikmə təmin edə bilər.

Mənfi cəhətləri/Çətinliklər:

- Daha zəif ardıcıllıq(consistency) zəmanətləri: Lidersiz təkrarlama sistemində güclü ardıcıllıq zəmanətlərini saxlamaq çətin ola bilər. Bu, əlavə mexanizmlərdən istifadə etməklə həll edilməli olan ziddiyyətli yeniləmələr və ya məlumat uyğunsuzluğu ilə nəticələnə bilər.
- Daha yüksək mürəkkəblik: Lidersiz təkrarlama ardıcıllığı təmin etmək və münaqişələri həll etmək üçün əlavə mexanizmlər tələb edən daha mürəkkəb texnikadır.
- Qlobal sıralanmaya nail olmaqda çətinlik: Lidersiz replikasiyada yazma sorğularının əlaqələndirilməsinə cavabdeh olan tək bir qovşaq yoxdur və bu, yazıların qlobal sıralanmasına nail olmağı çətinləşdirir.

Lidersiz təkrarlama yüksək əlçatanlıq, nasazlığa dözümlülük və miqyaslılıq tələb edən sistemlər üçün uyğundur. O, həmçinin böyük həcmdə yeniləmələri idarə etməli olan və

son ardıcıllığa dözə bilən sistemlər üçün uyğundur. Bununla belə, güclü ardıcıllıq zəmanəti tələb edən sistemlər üçün uyğun olmaya bilər.

Xülasə, hər bir təkrarlama metodunun öz üstünlükləri, mənfi cəhətləri və müvafiq istifadə halları var. Tək liderin təkrarlanması güclü ardıcıllıq təminatları tələb edən sistemlər üçün uyğundur, çoxliderli və lidersiz replikasiya isə yüksək əlçatanlığa, nasazlığa dözümlülüyünə və genişlənmə qabiliyyətinə ehtiyacı olan sistemlər üçün uyğundur. Çox liderli replikasiya yüksək yazma qabiliyyəti tələbləri olan sistemlər üçün daha uyğun ola bilər, lidersiz replikasiya isə son ardıcıllığa dözə bilən sistemlər üçün daha uyğun ola bilər. Nəhayət, düzgün təkrarlama metodunun seçilməsi sistemin xüsusi tələblərindən və xüsusiyyətlərindən asılıdır.

III FƏSİL. Tək liderli replikasiya üsulunun reallaşdırılması

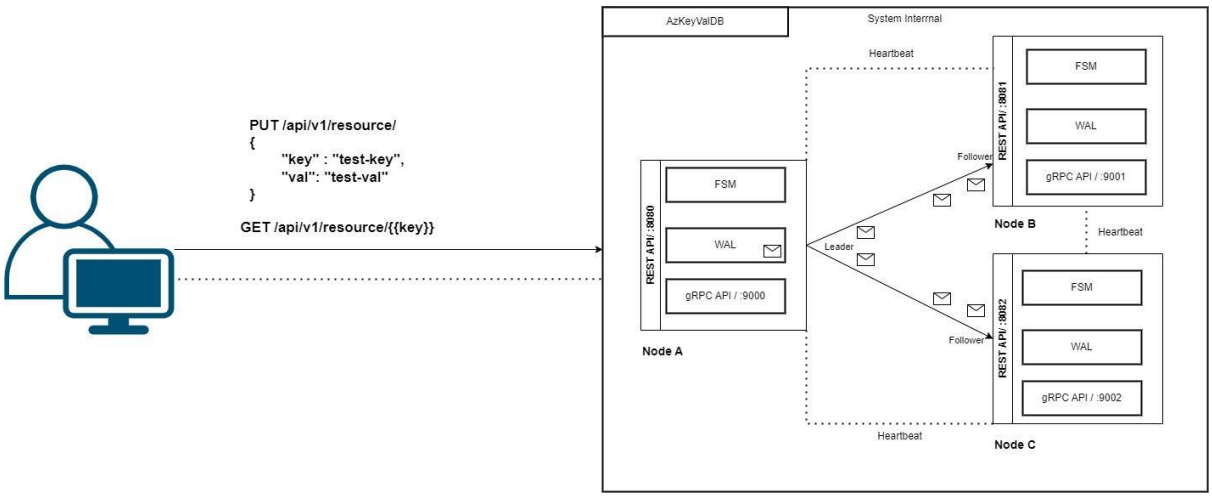
3.1 Proqram təminatı nümunə

Bu fəsildə praktiki əhəmiyyət daşıyan proqram təminatına baxış keçiriləcəkdir.

AzKeyValDB *nədir?*

AzKeyValDB verilənlərin saxlanması üçün açar-dəyər formatından istifadə edən və Go proqramlaşdırma dilindən istifadə edərək klyient-server arxitekturasına malik verilənlərin idarə edilməsi həllidir. Etibarlılığı təmin etmək üçün sistem vahid lider təkrarlama yanaşmasından istifadə edir.

Logların replikasiyası və lider qovşağın seçilməsi RAFT alqoritmindən istifadə etməklə idarə olunur.



Şəkil 3.1. Proqram arxitekturası

Şəkil 3.1-də göstərilən arxitekturaya nəzər yetirdikdə, sistemin 3 node-dan(Node A, B və C) ibarət olduğunu görə bilərik.

Node A: Lider vəya master node

Node B: Follower vəya slave node

Node C: Follower vəya slave node

3.2 Proqramın arxitekturası

Hər bir node 4 hissədən ibarətdir.

1-ci hissə: REST API

REST API və ya Representational State Transfer Application Programming Interface veb xidmətləri qurmaq üçün təlimatlar və prinsiplər toplusudur. Bu, HTTP-dən rabitə protokolu kimi istifadə edən veb-əsaslı xidmətlərin yaradılması üçün memarlıq üslubudur.

RESTful API-lər müxtəlif sistemlər və proqramlar arasında əlaqə yaratmağa imkan verir və məlumatların standart formatda mübadiləsinə imkan verir. RESTful API-lər ümumiyyətlə veb inkişafında istifadə olunur və genişlənən, çevik və təkrar istifadə edilə bilən xidmətlərin qurulmasına imkan verir.

OpenAPI (əvvəllər Swagger kimi tanınır) RESTful API sənədlərinin qurulması üçün spesifikasiyadır. O, son nöqtələr, əməliyyatlar, giriş/çıxış parametrləri, cavab kodları, autentifikasiya tələbləri və digər detallar haqqında məlumat daxil olmaqla, API-ləri təsvir etmək üçün standart formatı müəyyən edir.

OpenAPI müqaviləsi RESTful API-nin davranışını təsvir edən sənəddir. O, gözlənilən giriş və çıxış formatları, icazə verilən HTTP metodları və görülməli olan hər hansı autentifikasiya və ya təhlükəsizlik tədbirləri daxil olmaqla, API ilə qarşılıqlı əlaqə qaydaları və tələblərini müəyyən edir. OpenAPI müqavilələri çox vaxt çoxlu API tətbiqlərində ardıcıllıq və dəqiqliyi təmin etmək və tərtibatçılar və maraqlı tərəflər arasında əməkdaşlığı asanlaşdırmaq üçün istifadə olunur. Cari sistem üçün hazırlanmış OpenAPI nümunəsi Əlavə 1-də göstərilmişdir.

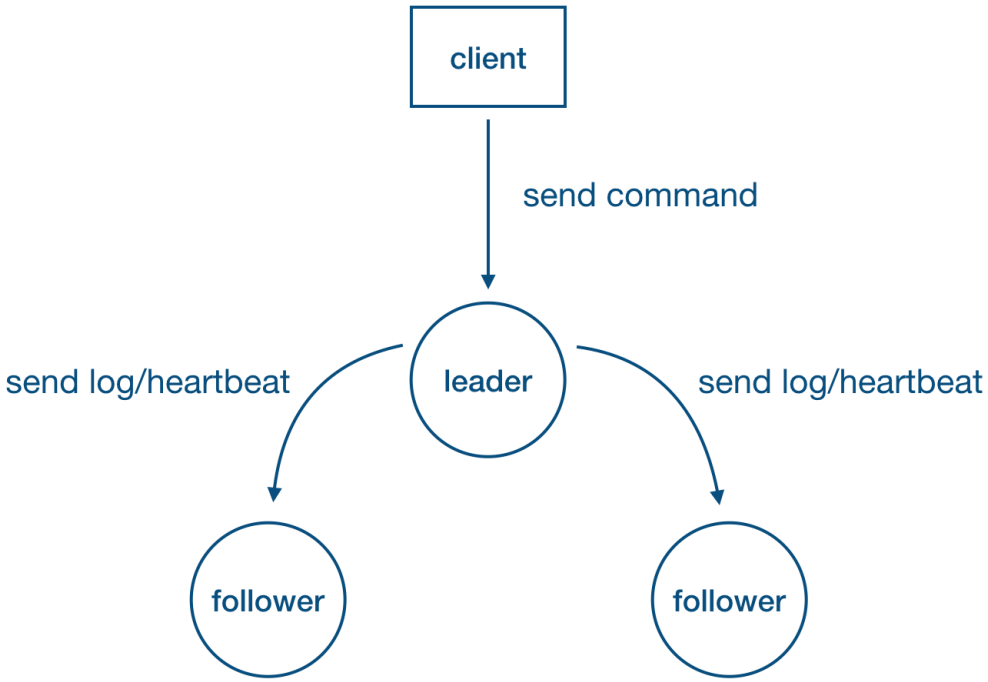
Klyient sorğuları http protokol üzərindən expose edilmiş REST API ilə handle olunur.

2-ci hissə: FSM

Raft konsensus alqoritmində replikasiya olunan state maşın həyata keçirilməsi üçün sonlu state maşını (FSM) istifadə olunur [12]. Təkrarlanan state maşını Raft alqoritminin əsas komponentidir və klasterdəki bütün serverlərin eyni vəziyyətə malik olmasını təmin etmək üçün məsuliyyət daşıyır.

Raftdakı FSM, bir sıra vəziyyətlərdən və bu statelər arasında keçidlərdən ibarət olan bir modeldir. Hər bir vəziyyət Raft alqoritmində fərqli bir mərhələni təmsil edir və hər bir keçid bir mərhələdən digərinə dəyişikliyi təmsil edir.

Raft-da FSM, klasterdəki bütün serverlərin eyni vəziyyətdə olmasını təmin etmək üçün istifadə olunur və onların hansı vəziyyətdə olduqları barədə razılığa gəlirlər. Müştəri Raft klasterinə əmr göndərdikdə, əmr ilk olaraq liderin təkrarlanan state maşınında icra olunur. Lider daha sonra həmin əmri öz təkrarlanan state maşınlarına tətbiq etməsi üçün followerlara göndərir.



Şəkil 3.2. Log və heartbeat işləmə prosesi

FSM, liderlərin və followerların hamısının state maşınına əmrlərin tətbiq olunma qaydası ilə razılaşmasını təmin etmək üçün istifadə olunur. Bu, bütün əmrləri alındıqları ardıcıllıqla qeyd etmək üçün jurnaldan istifadə etməklə və bütün serverlərin jurnalın məzmunu ilə razılaşdığını təmin etmək üçün konsensus alqoritmindən istifadə etməklə həyata keçirilir.

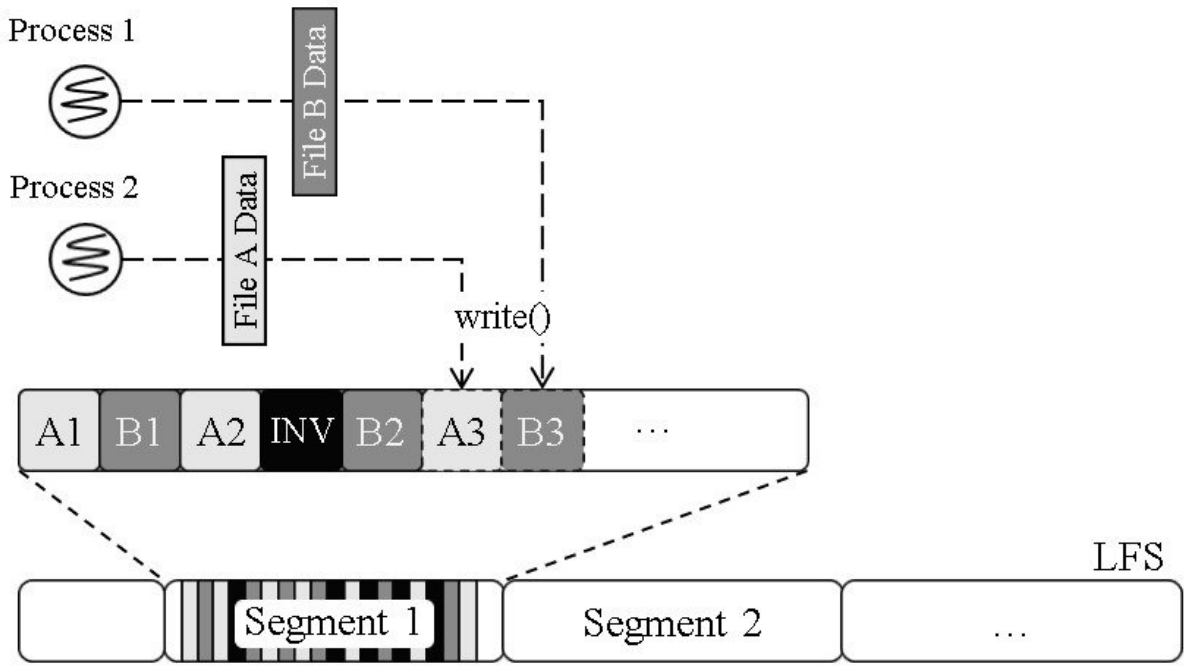
Raft-dakı FSM sadə və asan başa düşülən, alqoritmin davranışı haqqında düşünməyi asanlaşdırmaq və onun düzgün və səmərəli olmasını təmin etmək üçün nəzərdə tutulmuşdur. Əlavə 1-də proqramın FSM əlaqəli kodları verilmişdir.

3-cü hissə: WAL

Raft konsensus alqoritmində qabaqcadan yazma jurnalı (WAL) təkrarlanan state maşınının davamlılığını və ardıcıllığını təmin etmək üçün istifadə olunan kritik komponentdir. WAL, state maşınına bütün yeniləmələri qeyd etmək üçün istifadə edilən yalnız əlavə üçün jurnal faylıdır [9].

Müştəri Raft klasterinə əmr göndərdikdə, komanda əvvəlcə lider qovşağında WAL-a yazılır. Lider daha sonra əmri öz təkrarlanan state maşınlarına tətbiq edən və həmçinin WAL-larına yazan izləyicilərə göndərir.

WAL-dan istifadə etməklə, Raft alqoritmi state maşınındakı bütün yeniləmələrin tətbiq edilməzdən əvvəl qeydiyyata alınmasını təmin edir. Bu, server nasazlığı və ya şəbəkə bölməsi halında state maşınının bərpasına imkan verir. Server uğursuz olduqda və ya klasterlə əlaqəni kəsdikdə, WAL-dan jurnal qeydlərini təkrar oxutmaqla öz vəziyyətini bərpa edə bilər.



Şəkil 3.3. WAL işləmə prosesi

WAL yalnız əlavə oluna bilən jurnal faylı olmaq üçün nəzərdə tutulmuşdur, yəni yeni qeydlər yalnız jurnalın sonuna əlavə edilə bilər. Bu, jurnalın dəyişməz olmasını və əvvəlki qeydlərin dəyişdirilə və ya silinə bilməyəcəyini təmin edir. Bütün yeniləmələrin dəyişməz jurnalını saxlamaqla, Raft alqoritmi hətta uğursuzluqlar və ya şəbəkə bölmələri halında belə state maşınının həmişə ardıcıl olmasını təmin edə bilər.

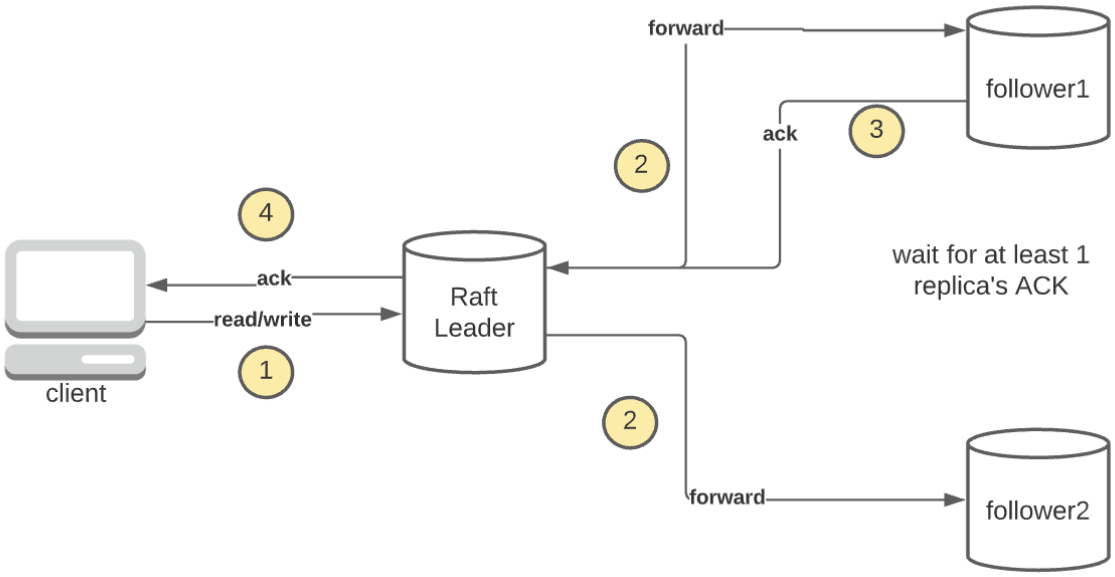
Xülasə, Raft-dakı qabaqcadan yazmaq jurnalı (WAL) təkrarlanan state maşınına bütün yeniləmələri qeyd etmək üçün istifadə edilən yalnız əlavə üçün jurnal faylıdır. State maşınının davamlılığını və ardıcıllığını təmin edən Raft alqoritminin kritik komponentidir.

4-cü hissə: gRPC API

Sistem daxilində nodelar bir-birləri ilə gPRC API əsasında kommunikasiya yaradır. Aşağıdakı hallarda nodelar arasında kommunikasiya baş verir.

- Replikasiya olunan logların Lider nodedan follower nodelara göndərilməsi zamanı
- Lider fail olduğu zaman, yeni liderin seçilməsi zamanı
- Heartbeat tipli mesajların göndərilməsi zamanı.

Raft konsensus alqoritmində liderin seçki prosesi cari lider uğursuz olduqda və ya klasterin qalan hissəsi ilə əlaqəsi kəsildikdə yeni lider seçmək üçün istifadə olunur. Raftda lider seçki prosesi sadə, səmərəli və yüksək əlçatan olmaq üçün hazırlanmışdır.



Şəkil 3.4. RAFT replikasiya prosesi

Raftda lider seçki prosesində iştirak edən addımlar bunlardır:

Bir izləyici qovşağı müəyyən bir müddət ərzində liderdən ürək döyüntülərini ala bilmədikdə (seçki vaxt aşımı kimi tanınır), o, yeni seçkiyə başlayır. Fərqli qovşaqların eyni vaxtda seçkilərə başlamamasını təmin etmək üçün seçki vaxtı müəyyən diapazonda təsadüfi olaraq seçilir.

İzləyici qovşağı namizəd vəziyyətinə keçir və klasterdəki bütün digər qovşaqlara RequestVote mesajı göndərir. RequestVote mesajında namizədin jurnalı haqqında məlumat, o cümlədən son jurnal girişinin indeksi və müddəti var. Namizəd digər qovşaqlardan səs çoxluğunu alarsa, o, yeni lider olur. Namizəd bütün digər qovşaqlara onun lider olduğunu bildirmək üçün ürək döyüntüsü mesajı göndərir. Namizəd səslərin əksəriyyətini ala bilmədikdə, ya seçkidə qalib gələnə, ya da yeni lider seçilənə qədər artan seçki vaxtı ilə yeni seçkilər keçirməyə davam edir.

Lider seçkisi zamanı qovşaqlar bir müddət ərzində yalnız bir namizədə səs verə bilər. Hər bir qovşaq verdiyi ən son səsin müddət nömrəsini qeyd edir, beləliklə, cari müddətdə artıq səs verib-vermədiyini müəyyən edə bilər. Bu, namizədin müəyyən müddətdə bir qovşaqdan birdən çox səs ala bilməməsini təmin edir.

Ümumilikdə, Raft-da lider seçki prosesi uğursuzluq və ya şəbəkənin bölünməsi halında yeni liderin tez seçilməsini təmin edərək yüksək əlçatan olmaq üçün nəzərdə tutulmuşdur.

3.3 Proqramın istifadə qaydası

Proqramdan istifadəni asanlaşdırmaq üçün Makefile istifadə edilmişdir. Makefile proqram layihəsinin necə qurulacağını müəyyən edən təlimatlar toplusunu ehtiva edən fayldır. O, adətən, hər biri mənbə kodundan müəyyən bir fayl və ya fayl dəstinin necə qurulacağını müəyyən edən bir sıra qaydalardan ibarətdir.

Əlavə 2-də göstərilən makefile tək node Raft konsensus alqoritmindən istifadə edərək *azkeyvaldb* adlı paylanmış açar-dəyər database proqramını qurmaq, hazırlamaq və işə salmaq üçün qaydalar toplusunu ehtiva edir.

1. *app.generate*: Bu qayda oapi-codegen istifadə edərək OpenAPI spesifikasiya faylı əsasında Go kodunu yaradır. O, modellər, server, idarəetmə serveri və sağlamlıq serveri üçün kod yaradır.
2. *app.prepare*: Bu qayda bütün asılılıqların güncəl olmasını təmin etmək üçün Go modul sistemindən istifadə edir və istifadə olunmamış asılılıqları aradan qaldırır.
3. *app.build*: Bu qayda cari kataloqdakı mənbə kodundan *azkeyvaldb* tətbiqini qurur. Bütün asılılıqların güncəl olmasını təmin etmək üçün bu, *app.prepare* qaydasından asılıdır.
4. *app.run.0*: Bu qayda 9090 portunda Raft konsensus alqoritmindən və 8080 portunda API serverindən istifadə edərək state direktoriya *$(TMPDIR_PATH)/0* olaraq təyin edilmiş *azkeyvaldb* proqramını işə salır.
5. *app.run.1*: Bu qayda state direktoriya *$(TMPDIR_PATH)/1* olaraq təyin edilmiş *azkeyvaldb* tətbiqinin ikinci nümunəsini 9091 portda Raft konsensus alqoritmindən, 8081 portunda API serverindən istifadə edərək və birinci instansiyaya qoşularaq işlədir. 9090 portunda.
6. *app.run.2*: Bu qayda state direktoriya *$(TMPDIR_PATH)/2* olaraq təyin edilmiş azkeyvaldb tətbiqinin üçüncü nümunəsini 9092 portunda Raft konsensus

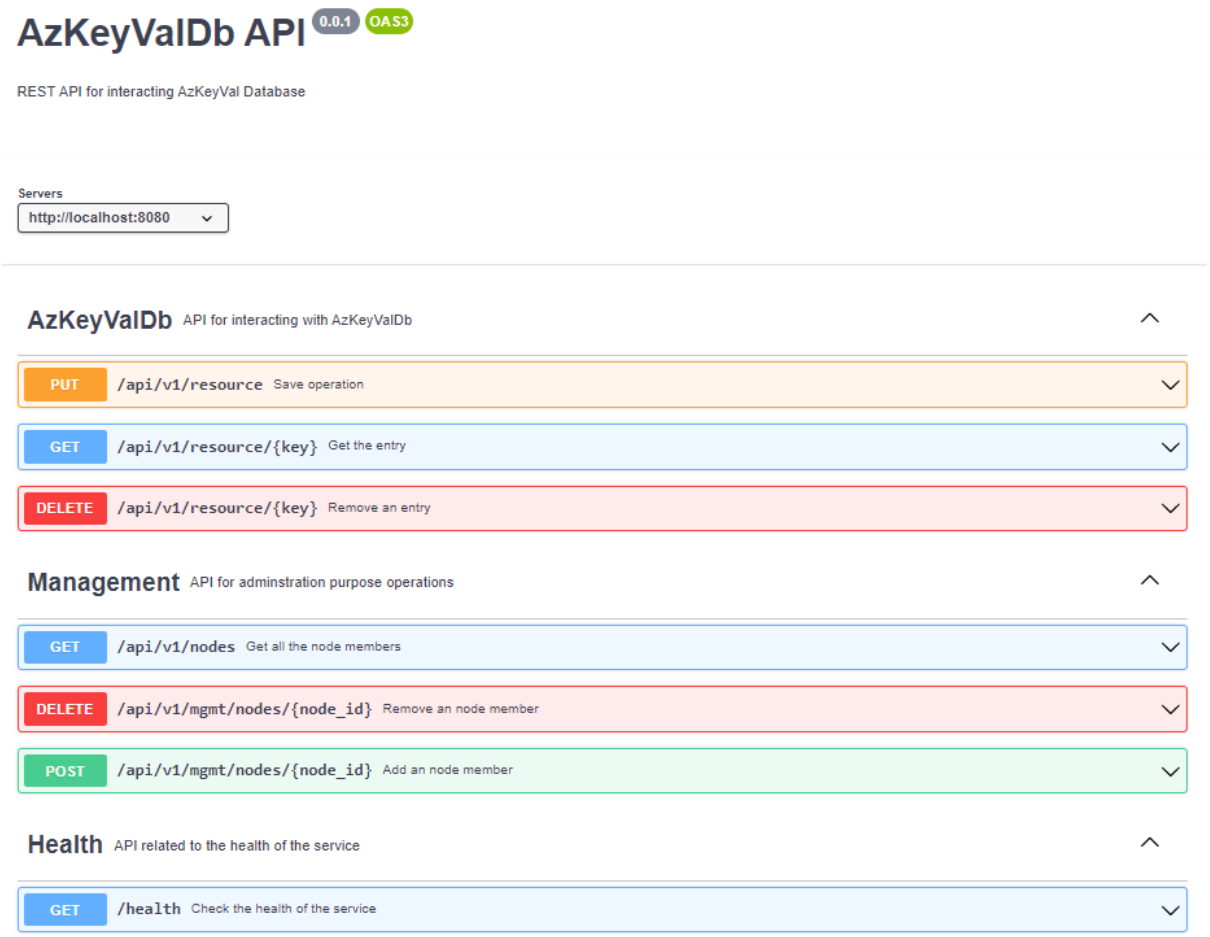
alqoritmini, 8082 portdakı API serverini istifadə edərək və birinci instansiyaya qoşularaq işlədir. 9090 portunda.

Ümumilikdə, bu qaydalar tək qovşaq Raft konsensus alqoritmindən istifadə edərək paylanmış açar-dəyər verilənlər bazası qurulması və işlədilməsi prosesini avtomatlaşdırır və tətbiqi idarə etməyi və yerləşdirməyi asanlaşdırır.

Makefileda qeyd olunan stagelər vasitəsilə 1, 2 vəya 3 nodedan ibarət cluster initializasiya etmək mümkündür.

Əlavə 4.1-də 1 nodedan ibarət klaster initializasiya etmək üçün zəruri əmrlər verilmişdir.

Əgər onu interaktiv şəkildə sınamaq istəyirsinizsə, alətlərlə open-api-spec.yaml kontraktından istifadə edə bilərsiniz: online swagger-ui, Postman və s.



Şəkil 3.5. Veb servvisin Open-API kontraktı

Şəkil 3.5-də göstərilən endpointlərin izahı:

- ***PUT /api/v1/resource:*** verilənlər bazasında açar-dəyər girişini saxlayır.
- ***GET /api/v1/resource/{key}:*** Xüsusi açarla əlaqəli dəyəri alır.
- ***DELETE /api/v1/resource/{key}:*** Xüsusi açarla əlaqəli dəyəri silir.
- ***GET /api/v1/nodes:*** Klasterdəki bütün qovşaqlar haqqında məlumat alır.
- ***DELETE /api/v1/mgmt/nodes/{node_id}:*** Xüsusi qovşağı klasterdən çıxarır.
- ***POST /api/v1/mgmt/nodes/{node_id}:*** Klasterə yeni qovşaq əlavə edir.
- ***GET /health:*** Xidmətin sağlam olduğunu göstərmək üçün 200 OK cavabını qaytarır.

Komponentlər bölməsi sorğu və cavab yükləri üçün sxemlər və HTTP status kodları üçün cavablar kimi təkrar istifadə edilə bilən komponentləri müəyyən edir. Məsələn, *KeyValEntry* sxemi iki tələb olunan xüsusiyyətlə müəyyən edilir:
açar(key) və dəyər(value). Bu sxem açar-dəyər girişləri ilə məşğul olan son nöqtələr üçün sorğu və cavab yüklərində istifadə olunur. Eynilə, *UnknownError* cavabı sistem daxilində naməlum xəta baş verdikdə qaytarılan səhv mesajının təsviri və sxemi ilə müəyyən edilir.

Mövcud HTTP klyient alətlər(Postman və s.) vəya cUrl terminal proqramı ilə sorğu göndərə bilərsiniz:
Əlavə 4.2-də cUrl kod nümunələri mövcuddur. Əlavə 4.3-də cUrl ilə açar(*ing., key*)əsasında dəyərin(*ing., value*) əldə olunması, Əlavə 4.4-də Klasterin initializasiyası üçün zəruri kodlar verilmişdir.

Klaster rejimde initializasiya etdikdən sonra klasterin istənilən üzvünə açar-dəyər cütü yazmaq və eyni şəkildə onu istənilən üzvdən almaq mümkündür.

Fault-Tolerant olmasının yoxlanılması

Klasterin baş verə biləcək xətalara qarşı necə tolerant olmasını yoxlamaq üçün *"foo"* açarına *"foo"* dəyərini yazın: (bax Əlavə 4.5)

Daha sonra, qovşağı (gRPC:9092) dayandırın və klasterin mövcudluğunu yoxlamaq üçün dəyəri "bar" ilə əvəz edin:(bax Əlavə 4.6)

Nəhayət, nodu(gRPC:9092) geri qaytarın və onun yenilənmiş “bar” dəyəri ilə bərpa olub-olmadığını yoxlayın: (bax Əlavə 4.7)

Nəticədən göründüyü kimi, məqsədyönlü şəkildə öncədən fail olunmuş node, start edilən zaman, catch-up recovery prosesi əsasında öz daxili stateni master node ile sync vəziyyətinə gətirmişdir.

Nəticə

Magistr dissertasiya mövzusu əlaqədar data-intensiv sistemlərin və replikasiyanın mahiyyətinin öyrənilməsi ilə başlamış, replikasiyada istifadə edilən üsulların tətbiq olunduğu sistemlərin arxitekturalarına nəzər yetirilmiş və hər üsulun realizasiyası zamanı yaranan çətinlikləri, o cümlədən müsbət və mənfi cəhətləri müyyən edilməklə aşağıdakı nəticələrlə yekunlaşmışdır. Praktiki tətbiq təkrarlama metodlarının məlumatların intensiv istifadəsi sistemlərinə necə uğurla daxil edilə biləcəyinin əyani nümayişi kimi xidmət etdi.

- Müəyyən edilmişdir ki, replikasiya geniş miqyaslı sistemlərdə məlumatların əlçatanlığının, nasazlığa dözümlülüyün və miqyaslılığın təmin edilməsində həlledici rol oynayır. Replikasiya sistem performansını yaxşılaşdırmağa, məlumatların davamlılığını artırmağa və qüsurlara qarşı davamlılığı artırmağa imkan verir. Bu amillər yüksək əlçatanlığın və etibarlılığın əsas olduğu məlumatların intensiv istifadəsi sistemlərində həyati əhəmiyyət kəsb edir.
- Replikasiya üsullarının müqayisəli tədqiqi hər bir metodun öz güclü və məhdudiyyətlərinə malik olduğunu ortaya qoydu. Tək liderin təkrarlanması güclü ardıcıllıq təmin edir və linearizasiyaya zəmanət verir, lakin performans darboğazları və tək uğursuzluq nöqtələri yarada bilər. Çox liderli təkrarlama artan performans və miqyaslılıq təklif edir, lakin replikalar arasında ardıcıllığı qorumaq çətin ola bilər. Lidersiz replikasiya yüksək əlçatanlıq və xətaya dözümlülük təmin etməklə yanaşı, replikalar arasında ziddiyyətlərin diqqətlə idarə olunmasını və koordinasiyanı tələb edir.
- Tək liderin təkrarlanmasının praktiki həyata keçirilməsi replikasiya üsullarının real dünya ssenarisində həyata keçirilməsinin mümkünlüyünü nümayiş etdirdi. Nümunə verilənlər bazası etibarlı və genişlənə bilən replikasiyaya nail olmaq üçün lazımi arxitektura, sinxronizasiya mexanizmləri və replikasiya quraşdırmasını nümayiş etdirdi. Bu praktiki tətbiq təkrarlama strategiyalarının effektiv şəkildə tətbiqi ilə bağlı mürəkkəbliklərə və mülahizələrə işıq salır.
- Data-intensiv sistemlərində replikasiya metodlarının müqayisəli tədqiqi və tətbiqinə dair tədqiqat məlumat sisteminin etibarlılığını və miqyasını təmin

etməkdə replikasiyanın əhəmiyyətini vurğuladı. Müqayisəli təhlil müxtəlif təkrarlama üsullarının güclü tərəfləri və məhdudiyyətləri haqqında dəyərli fikirlər verdi. Praktiki tətbiq təkrarlama metodlarının məlumatların intensiv istifadəsi sistemlərinə necə uğurla daxil edilə biləcəyinin əyani nümayişi kimi xidmət etdi.

İstifadə olunmuş ədəbiyyat siyahısı

1. Akidau, T., Chernyak, S., & Lax, R. (2018). Streaming Systems - The What, Where, When, and How of Large-Scale Data Processing.
2. Beg, C. E., & Connolly, T. M. (2004). A Practical Approach to Design, Implementation, and Management.
3. Burns, B. (2018). Designing Distributed Systems - Patterns and Paradigms for Scalable, Reliable Services.
4. Connolly, T. M., & Begg, C. E. (2014). DATABASE SYSTEMS A Practical Approach to Design, Implementation, and Management (6th ed.).
5. Elmasri, R., & Navathe, S. B. (2021). Fundamentals of Database Systems (7th ed.).
6. Kleppmann, M. (2017). Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems (1st ed.).
7. Kleppmann, M. (2020). Distributed Systems notes.
8. Marz, N., & Warren, J. (2015). Big Data: Principles and best practices of scalable realtime data systems (1st ed.).
9. Nassar, H. (2023). Article: "Why databases need Write-ahead log (WAL)— A deep dive".
10. Petrov, A. (2019). Database internals.
11. PostgreSQL Global Development Group. (2023). Documentation - PostgreSQL 15.2.
12. The Apache Software Foundation. (2023). Documentation - Apache Kafka 3.4.
13. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). Database System Concepts (7th ed.).
14. Tanenbaum, A. S., & Van Steen, M. (2007). Distributed Systems (2nd ed.).

ƏLAVƏLƏR

Əlavə 1

```
package core

import (
	"encoding/json"
	"io"
	"io/ioutil"
	"log"
	"strings"
	"sync"
)

type StateMachine struct {
	mu sync.Mutex
	kv map[string]string
}

func NewStateMachine() *StateMachine {
	return &StateMachine{
		kv: make(map[string]string),
	}
}

func (s *StateMachine) Apply(data []byte) {
	var e Entry
	if err := json.Unmarshal(data, &e); err != nil {
		log.Println("unable to Unmarshal entry", err)
		return
	}
	s.mu.Lock()
	defer s.mu.Unlock()
	s.kv[e.Key] = e.Value
}

func (s *StateMachine) Snapshot() (io.ReadCloser, error) {
	s.mu.Lock()
	defer s.mu.Unlock()
	buf, err := json.Marshal(&s.kv)
	if err != nil {
		return nil, err
	}
	return io.NopCloser(strings.NewReader(string(buf))), nil
}

func (s *StateMachine) Restore(r io.ReadCloser) error {
	s.mu.Lock()
	defer s.mu.Unlock()
```

```
        buf, err := ioutil.ReadAll(r)
        if err != nil {
        return err
        }

        err = json.Unmarshal(buf, &s.kv)
        if err != nil {
                return err
        }

        return r.Close()
}

func (s *StateMachine) Read(key string) string {
        s.mu.Lock()
        defer s.mu.Unlock()
        return s.kv[key]
}
```

Əlavə 2

```
openapi: 3.0.3

info:
  version: 0.0.1
  title: AzKeyValDb API
  description: |
    REST API for interacting AzKeyVal Database

servers:
  - url: http://localhost:8080
  - url: http://localhost:8081
  - url: http://localhost:8082

tags:
  - name: AzKeyValDb
    description: API for interacting with AzKeyValDb
  - name: Management
    description: API for adminstration purpose operations
  - name: Health
    description: API related to the health of the service

paths:
  /api/v1/resource:
    put:
      summary: Save operation
      description: Update the value with the associated key if exist, else insert
```

```
operationId: save
      parameters:
        - $ref: '#/components/parameters/x-correlation-id'
      tags:
        - AzKeyValDb

    requestBody:
        $ref: '#/components/requestBodies/UpdateKeyValEntryRequest'
      responses:
        200:
          $ref: '#/components/responses/KeyValEntryResponse'
        default:
          $ref: '#/components/responses/UnknownError'

  /api/v1/resource/{key}:
    get:
      summary: Get the entry
      description: Retrieve an entry based on the key
      operationId: find
      parameters:
        - $ref: '#/components/parameters/x-correlation-id'
        - $ref: '#/components/parameters/p_key'
      tags:
        - AzKeyValDb
      responses:
        200:
          $ref: '#/components/responses/KeyValEntryResponse'
        default:
          $ref: '#/components/responses/UnknownError'

    delete:
      summary: Remove an entry
      description: Remove an entry based on the key
      operationId: remove
      parameters:
        - $ref: '#/components/parameters/x-correlation-id'
        - $ref: '#/components/parameters/p_key'
      tags:
      - AzKeyValDb
      responses:
        204:
          $ref: '#/components/responses/NoContentResponse'
        default:
          $ref: '#/components/responses/UnknownError'


  /api/v1/nodes:
    get:
      summary: Get all the node members
      description: Retrieve all cluster members
```

```
operationId: getAllNodeMembers
      parameters:
        - $ref: '#/components/parameters/x-correlation-id'
      tags:
        - Management
      responses:
        200:
          $ref: '#/components/responses/NodeMembersResponse'
        default:
          $ref: '#/components/responses/UnknownError'

  /api/v1/mgmt/nodes/{node_id}:
    delete:
      summary: Remove an node member
      description: Remove a member from cluster
      operationId: removeNodeMember
      parameters:
        - $ref: '#/components/parameters/x-correlation-id'
        - $ref: '#/components/parameters/p_node_id'
      tags:

    - Management
      responses:
        204:
          $ref: '#/components/responses/NoContentResponse'
        default:
          $ref: '#/components/responses/UnknownError'
    post:
      summary: Add an node member
      description: Add a member from cluster
      operationId: addNodeMember
      parameters:
        - $ref: '#/components/parameters/x-correlation-id'
        - $ref: '#/components/parameters/p_node_id'
      tags:
        - Management
      responses:
        204:
          $ref: '#/components/responses/NoContentResponse'
        default:
          $ref: '#/components/responses/UnknownError'

  /health:
    get:
      summary: Check the health of the service
      description: Returns a 200 OK response if the service is healthy
      tags:
        - Health
      responses:
        200:
```

```
      description: The service is healthy

components:
  schemas:
    Error:
      type: object
      description: Representation of an Error that can appear using the application.
      required:
        - code
        - message
      properties:
        code:
          description: The code of an error that describes the Error.
          type: string
        message:
          description: The message of an error that describes the Error.
          type: string

    KeyValEntry:
      type: object
      description: Key-Value entry representation
      required:
        - key
        - value
      properties:
        key:
          description: The key of entry
          type: string
        value:
          description: The value of the entry
          type: string

    NodeMember:
      type: object
      description: Node Member representation
      required:
        - id
        - addr
      properties:
        id:
          description: ID of node member
          type: string
        addr:
          description: The network address of node where running on
          type: string

  responses:
    NoContentResponse:
      description: The request was successfully processed.
```

```
  UnknownError:
    description: The unknown error appeared. Check your payload or contact support.
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Error'

  KeyValEntryResponse:
    description: OK
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/KeyValEntry'

NodeMembersResponse:
 description: OK
    content:
      application/json:
        schema:
          type: array
          items:
            $ref: "#/components/schemas/NodeMember"


 requestBodies:
  UpdateKeyValEntryRequest:
    required: true
    content:
      application/json:
        schema:
          $ref: "#/components/schemas/KeyValEntry"

 parameters:
  x-correlation-id:
    in: header
    name: x-correlation-id
    required: false
    description: >
      The unique request identifier
    schema:
      type: string

  p_key:
    in: path
    name: key
    required: true
    description: >
      Represents key of the entry
    schema:
      type: string
```

```
    p_node_id:
      in: path
      name: node_id
      required: true
      description: >
        Represents ID of member node
      schema:
        type: string
```

Əlavə 3

```
###
TMPDIR_PATH?=./clustertempdata/

app.generate:
	cd ./restapi/api \
	&& oapi-codegen -config models.cfg.yaml ./open-api-spec.yaml \
	&& oapi-codegen -config azkeyvaldb-server.cfg.yaml ./open-api-spec.yaml \
	&& oapi-codegen -config management-server.cfg.yaml ./open-api-spec.yaml \
	&& oapi-codegen -config health-server.cfg.yaml ./open-api-spec.yaml

app.prepare:
	go mod tidy

app.build: app.prepare
	go build .

# Running single node raft
app.run.0: app.build
	./azkeyvaldb -state_dir=$(TMPDIR_PATH)/0 -raft :9090 -api :8080

app.run.1:
	./azkeyvaldb -state_dir=$(TMPDIR_PATH)/1 -raft :9091 -api :8081 -join :9090

app.run.2:
	./azkeyvaldb -state_dir=$(TMPDIR_PATH)/2 -raft :9092 -api :8082 -join :9090
```

Əlavə 4.1

```
 make app.run.0
```

Əlavə 4.2

```
 INPUT
 curl -X 'PUT' \
  'http://localhost:8080/api/v1/resource' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{ "key": "hello", "value": "medium" }'

 OUTPUT
 {
          "key": "hello",
          "value": "medium"
 }
```

cUrl ilə açar(*ing., key*)əsasında dəyərin(*ing., value*) əldə olunması

```
INPUT
curl -X 'GET' \
 'http://localhost:8080/api/v1/resource/hello \
 -H 'accept: application/json'

OUTPUT
{
          "key": "foo",
          "value": "bar"
}
```

Əlavə 4.3

```
INPUT
curl -X 'GET' \
 'http://localhost:8080/api/v1/resource/hello \
 -H 'accept: application/json'

OUTPUT
{
          "key": "foo",
          "value": "bar"
}
```

```
```

Əlavə 4.4

```
make app.run.0
make app.run.1
make app.run.2
```

Əlavə 4.5

```
***INPUT***
curl -X 'PUT' \
 'http://localhost:8080/api/v1/resource' \
 -H 'accept: application/json' \
 -H 'Content-Type: application/json' \
 -d '{ "key": "foo", "value": "foo" }'

***OUTPUT***
{
        "key": "foo",
        "value": "foo"
}
```

Əlavə 4.6

```
 ***INPUT***
 curl -X 'PUT' \
  'http://localhost:8080/api/v1/resource' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{ "key": "foo", "value": "bar" }'

 ***OUTPUT***
 {
         "key": "foo",
         "value": "bar"
 }


 ***INPUT***
 curl -X 'GET' \
  'http://localhost:8080/api/v1/resource/foo' \
  -H 'accept: application/json'

 ***OUTPUT***
 {
```

```
            "key": "foo",
            "value": "bar"
 }
```

```

### Əlavə 4.7

```
INPUT
curl -X 'GET' \
 'http://localhost:8082/api/v1/resource/foo' \
 -H 'accept: application/json'

OUTPUT
{
        "key": "foo",
        "value": "bar"
}
```
```

Data-intensiv sistemlərdə replikasiya üsullarının müqayisəli tədqiqi və reallaşdırılması

XÜLASƏ

Dissertasiya məlumat sistemlərində replikasiyanın əhəmiyyətinə diqqət yetirir və müasir məlumat tələb edən mühitlərdə geniş istifadə olunan müxtəlif replikasiya növlərini araşdırır. Dissertasiya replikasiyanın əhəmiyyətini və geniş miqyaslı sistemlərdə məlumatların əlçatanlığını, xətalara dözümlülüyünü və miqyasını təmin etməkdə onun rolunu vurğulamaqla başlayır.

Tədqiqat üç əsas replikasiya növünü araşdırır: tək lider replikasiyası, çox lider replikasiyası və lidersiz replikasiya. Hər bir təkrarlama metodu onların əsas prinsipləri, üstünlükləri və məhdudiyyətləri daxil olmaqla hərtərəfli araşdırılır. Dissertasiya bu təkrarlama üsullarının hərtərəfli müqayisəsini təqdim edir, onların müxtəlif istifadə halları və iş yükləri üçün uyğunluğunu təhlil edir.

Dissertasiyanın yekun hissəsində vahid liderin təkrarlanmasının həyata keçirilməsini nümayiş etdirmək üçün nümunə verilənlər bazası hazırlanmışdır. Bu praktik tətbiq, məlumatların intensiv istifadəsi sistemində vahid liderin təkrarlanması metodunun necə effektiv şəkildə istifadə oluna biləcəyinin praktiki nümunəsi kimi xidmət edir. Nümunə verilənlər bazası etibarlı və genişlənə bilən replikasiya sxeminə nail olmaq üçün tələb olunan replikasiya konfiqurasiyasını, sinxronizasiya mexanizmlərini və ümumi arxitekturanı nümayiş etdirir.

Replikasiya üsullarının müqayisəli təhlilini təqdim etməklə və praktiki tətbiq nümunəsini təqdim etməklə bu dissertasiya məlumatların intensivliyi tələb edən sistemlərdə replikasiya üsullarının başa düşülməsinə töhfə verir. Bu, onların xüsusi tələbləri və məqsədləri ilə uyğunlaşan səmərəli replikasiya strategiyalarını tərtib etmək və tətbiq etmək istəyən tədqiqatçılar, sistem memarları və praktikantlar üçün dəyərli fikirlər təqdim edir.

Açar sözlər: verilənlər, verilənlərin replikasiyası, replikasiya stratejiyaları, verilənlər bazası, uyğunluq, verilənlər-intensiv sistemlər, yüksək əlçatmazlıq.

Comparative study and implementation of replication methods in Data-Intensive Systems

SUMMARY

The dissertation focuses on the significance of replication in data systems and explores various replication types commonly employed in contemporary data-intensive environments. The dissertation begins by highlighting the importance of replication and its role in ensuring data availability, fault tolerance, and scalability in large-scale systems.

The study delves into the three main types of replication: single leader replication, multi-leader replication, and leaderless replication. Each replication method is thoroughly examined, including their underlying principles, advantages, and limitations. The dissertation provides a comprehensive comparison of these replication techniques, analyzing their suitability for different use cases and workloads.

In the final part of the dissertation, an example database is developed to demonstrate the implementation of single leader replication. This practical implementation serves as a hands-on illustration of how the single leader replication method can be effectively utilized in a data-intensive system. The example database showcases the replication setup, synchronization mechanisms, and the overall architecture required to achieve a reliable and scalable replication scheme.

By presenting a comparative analysis of replication methods and offering a practical implementation example, this dissertation contributes to the understanding of replication techniques in data-intensive systems. It provides valuable insights for researchers, system architects, and practitioners seeking to design and deploy efficient replication strategies that align with their specific requirements and objectives.

Keywords: data, data replication, replication strategies, database, consistency, data-intensive systems, high-availability.

Сравнительное исследование и реализация методов репликации в системах с интенсивным использованием данных

РЕЗЮМЕ

Диссертация посвящена важности репликации в системах данных и исследует различные типы репликации, обычно используемые в современных средах с интенсивным использованием данных. Диссертация начинается с подчеркивания важности репликации и ее роли в обеспечении доступности данных, отказоустойчивости и масштабируемости в крупномасштабных системах.

В исследовании рассматриваются три основных типа репликации: репликация с одним лидером, репликация с несколькими лидерами и репликация без лидера. Каждый метод репликации тщательно изучается, включая лежащие в его основе принципы, преимущества и ограничения. В диссертации представлено всестороннее сравнение этих методов репликации с анализом их пригодности для различных вариантов использования и рабочих нагрузок.

В заключительной части диссертации разрабатывается пример базы данных для демонстрации реализации репликации с одним лидером. Эта практическая реализация служит практической иллюстрацией того, как можно эффективно использовать метод репликации с одним лидером в системе с интенсивным использованием данных. Пример базы данных демонстрирует настройку репликации, механизмы синхронизации и общую архитектуру, необходимую для создания надежной и масштабируемой схемы репликации.

Представляя сравнительный анализ методов репликации и предлагая пример практической реализации, эта диссертация способствует пониманию методов репликации в системах с интенсивным использованием данных. Он предоставляет ценную информацию для исследователей, системных архитекторов и практиков, стремящихся разработать и внедрить эффективные стратегии репликации, соответствующие их конкретным требованиям и целям.

Ключевые слова: данные, репликация данных, стратегии репликации, база данных, непротиворечивость, системы с интенсивным использованием данных, высокая доступность.