

Листа настрадаје:

- Проблематичке структуре података (Bloom филтер, skip list, Count-min sketch, HyperLogLog)
- Crashes and Recovery података (Write-Ahead Log)
- Ограничавање броја захтева корисника (Token Bucket, ...)
- Кешираше садржаја (LRU cache)
- LSM шабона (LSM Index, LSM Table, Memtable, ...)
- Hashing, mmap, Batch, Log, ...
- Компресија података
- Раг са блоковима

Проблематичке структуре података,

Hash функције, Bloom филтер

- Проблематичке структуре - структурна класа структура која се користи у струч. доделама; користимо је да штоље сумњиве величине компоненте података
- У структуре чак и не чувамо податак, него теку резултатију података
- Скаларност - Било да ли једна или више осеба ради с подацима
- Чув је да из једног податка неким преносом података у други податак
- Даштио кориснику hash функције - колизије се дешавају
- Није обавлено да се у нашем систему деси дешавка докле ће бити чекајући да обновимо (try-catch дешавридујући системи)!
- Обе структуре користе знатно мање меморије:
 - систем је дешави
 - много више савари да здраво у тоја
- Класичне дигерентијалне структуре нам што не могу обезбедити, док у проблематичким структурама што чувају
- Проблематичке структуре не могу се да је тико структуру у неку структуре; изгубили су е滋味ност - обе структуре тешко проучију

- Ако чувамо податаке да се њима радимо, већ производи јаве ефекти

Hash функције

- Бредност је континутне, или што је да буду континутне
- да је мултифорнито представљено податаке
- Правна идеја, записао: да за варијабилан улаз са једне стране добијамо мултифорнити излаз са друге стране
- false-positive одговор - када да не чувају, а затварају један ћи или обрнутију

Bloom филтер

- Проблематичка структура која треба да обреци да им је неки елемент у некој структуре података
- захватају јако мало ресурса (9,6 битса по елементу без обзира на величину елемента) са света 1% false-positive
- ако је false-positive - чуда смо правили структуре
- може да нам када да елемент није у складу или да је елемент монтира у складу (превртането мање нешаве)
- НЕ КОРИСТИМО за додавање, измену,brisање, РАДИ САМО филтрирање (да ли неки захтев треба да уђе у наш систем), САМО РАДИ ТУ СТАВА!
- једини што може је да изјави неки захтев у наш систем или да га обдије
- У позадини имају Bloom филтер - иницијали када иницијалне не иницијалне
- алицијализира је да склонимо Bloom филтер чада онда пренадије све шта и што (многа много дуже време)
- Блом филтер - прво, што текомико паралелар које иштено да конверзији
- што је тачни false-positive, што је већа структура и обрнутијо
 - ако се користи наш Bloom филтер (мање битса), постоји већа шанса за лакше производње резултате то је важно што се више елемента може "изградити" у чистим блоковима

- ако се пац **погрешка** Bloom филтер (више битова), сматруј се што се за падне нивоштите резултатне информације. Више простора за потрату информација у ослећеном шифру. Ту обично дође до сматње ствари false-positive резултата.

- Bloom филтер сасијаваје је од низа битова где су сви битови иницијално постављени на 0 важи чиме
- затим изаберено је какви функције
- треба да из бесконачног прецима у коначној мрежи (модуларнији послужници)
- због **коликав** не може са сигурношћу да каже да ли се елемент налази у низу, али сигурно може рећи ако је присуство јако користно!
- имати као коликав хотено false-positive и откапте коликав ће бити низ
- пр.: • ако се на све приказује таје 1 → каже Можда
• ако на показује шама 0 → каже сигурно Није
- да ова коначност тим уноси проблем, али је погрешно коначност - и то коначноста радио формулама

да је остало **бесконачно** - шама бисмо ДЕТЕРМИНИСТИЧКУ СТРУКТУРУ

*** формулре да те чујимо! ***

- значи бројно или **false-positive** или број елемента

↓
класичне користичке
апликације

↓
једини што физички
бара са подсигурка
(тада је потпуно да је само
коликав их има)

Пregabatte 2

11.10.2024.

- проблематичке структуре су суперјеснотостварнице за разумевање и доказивање
- да разумемо како алгоритам ради, како структуре раде - на основу шта већ озлушкили да ли тело за некија користи или не

Streaming јаданака, Count-min sketch,

HyperLogLog, LookUp табеле

како корисници твог мониторинга убрзани неке тваше структуре

- Када се каже да је текущи streaming значи да твоји јаданаки коначно долазе
- Нешто коначно угради тешерине јаданаке
- два извора који давају различити формат и обим
- Основни задатак је како ми да реализујемо тај stream
- јаданаки било који складиште на једном месту да дају са њима
- Како било нешто бесконачно сачувавши на њему што је коначно одн. Меморији рампнера
- Све hash функција - многе комаде
- ако идемо да велике обиме јаданаке, тада битно је мало општило неки вредности, у таквим ситуацијама обично идемо да некаков прет
- Count-min sketch - може да преузети дојатаже (да каже да се некоје десило). Свеје први ненужне стечења, или чак што нас не занима шта сада број - што је првобитно једном креирата структура тај ради, што је што је



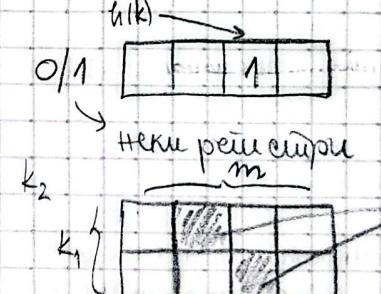
- као код Bloom филтера - што више јаданака чују сеју, комада ради
- шакоје постоеће структуре који служе да се најбољимају тајни урадимо

- зборни су за паралелизацију (једна и то највећа што мора добити вредност - макар их некако стварати)
- Често већо за пределем и даји један критични, па те они првобитно тајни текакбу средину

- правилно структуре са креова и т. колоста
- обе не радију пропушту вредности и као код Bloom филтера већ радију садиране - неударанти елементи за садиране су нула, па што се све на 0
- не мислимо 1 као код Bloom филтера, него вредност подстакавамо за 1.

- сваки ред представља поседну нахи функцију
- што си многе проследи пошто су веће
- реализација дрза - да запамитимо - добијавме вредност

- од свих вредности чврсто тајн - проблем је ако дате до комада



- ако изузимо Bloom филтер осијаје тиз, а ако изузимо Count-min sketch осијаје тиз тијоза

Обе су неке вредности, тису они 1 који чујемо у репрезентацији, дакле другачији се чујују јер су у репрезентацији маге најсамо он

- тијози су јако добри у индексирању
- листе су јако добре у додавању

- Counting Bloom филтер - комада је нека вредност јаданака у Bloom филтер
- много је ефикасније тајравитим нову неку вредност уважи

- обје боја като што су индекси требају да се узму
предностим

№: 4, 4, 2, 3

- 2 јединица смо видели број и у тиву

- min од {2, 3} је 2 - подрешница је у слайдовима

уместо један број

1. пролазимо кроз 1. hash функцију
2. пролазимо кроз 2. hash функцију

min {2, 3}

прва чешворка

k_1	0	0	0	1
k_2	0	0	0	1

4, 4, 2, 3

друга чешворка

k_1	0	0	0	2
k_2	0	0	0	2

4, 4, 2, 3

двојка

k_1	0	1	0	2
k_2	0	0	1	2

4, 4, 2, 3

шарџина

k_1	0	1	0	3
k_2	1	0	1	2

4, 4, 2, 3

- једна од ствари које требају да посматрате је смртност
а друга значајност? /комбинација

- Оно што је да знамо на основу чега докада структуре и
шта а ходимо њима да посматрате (** формуле не чујемо
такође! ***)

- обје какву додатну тулу да направимо ћете ради
за разлику од десктопнистичких структура које
тако се структуре пуне (догору слемени), структуре расподељене

- ако знамо да имамо stream, и да готови велику количину
инфо. - само у том случају је коришћене ове структуре праћења

- HyperLogLog - једно од најкомпактнијих структура
(некомора значи што је најкомпактнија за коришћење,
не је само мање математичка изложба компактноста)

- прво наслеђује нека структуре Log, па LogLog, па HyperLogLog,
сада има још унапредења верзија ове структуре -
HyperLogLog++

- структуре које смо раније користили примениваше су на
многобојна ходима у релативно добре перформансе, што обје
тије случај

- избор мора бити било шта

- исту напомену да решимо проблем counting distinct elem.
који може бити величине комплекситета
→ другим речима, оно дјелава да процеши колико јединица
тако спомената се налази у неком скупу без мешавине са
којима смешили у меморију и предвиђамо подскупачи

- обје се за разлику од прешкогних структуре не чува hash
- као и друге, када ову структуре направимо једном
не мора се всеки чини јасно

- ког Big data ^{терабайт} јако велике количине података тада ходимо
желим да закључимо; јако користи структуре у
еномпоријативни

- број тела дешавају на почетку хеширањем предностим -
идеја је да је добра, али тије још дешавају користи
за велике бројеве захтевају много паме

- број угаоштних тела са десне стране нам кате број иди нешто
- хардотијска страница покушава датим на основу неке
јединице које смо посматрали

- узимају неки број хеширањем тај број и дребају је
у битартни запис
и дјеламо и леву и десну страницу

14

1110 0011001000 → лево десно најдужи број нула и на то додато +1

$10=4 \rightarrow$ преузео је капте да узимају ч бинта

на коју извучу и најдо да узимају чашу брдностс (капте индекс)

- на индекс 14 чувају број 4

0	0	4	0	0	m
12	13	14	15	16	

- свели слој га на тиз

- како тимаја основу тиза да процећемо чаше број. → користимо некакву формулу

- хардверска средина спушта да објаснију процењене висине брдности и двојнитим процењу теком реалног броју

- и тог HyperLogLog-а и ког Counting-nin Sketch-a имамо некакве табеле и у њима су чеше брдности

- теке брдности су без рачунате и јединим их само користимо

24	--
25	--
26	--

→ како без рачунате ако смо већ израчунали да добијемо ово

- треба нам да ишчешимо брдности што су нисе, индексирати - да ишчешимо у неку табелу...

- што је сваких брдности тела а то, индексирате је друге

- lookip табеле

- имамо броје закупувате број. доле крутић, ми дирајмо

- ако имамо што рачунате избачимо оте изтаваје, а ако немају што рачунато



- све ове при структуре се могу сузети на хардвер
- чим сивари искључују сузети на хардвер, што значи да имамо државља чврзате
- билоје обе структуре зб. са lookip мапом можемо да убрзамо

"Нема што хардвера који ми сопствени и не можемо да задушимо"

"Сваки програм је баланс између сложетоса алгоритма и сложетоса структуре података"

Batch и mikro batch обрада података, skip list, simhash

- гатас би требало да завршило со подадениот иматуру,
- гатас се даваше предвидено влезот им за складиште на диску
- постоење стапување додека проблемот ког морамо да сакувамо податоците на диск
- идеја иза **batch** е скоро синхронна идери кој си имаме ког синхрон - овде што имаме податоците кога си сакуваме на дискоте и морамо да их одредимо на неки начин
- Всеки је укулите компоненте који обраќајуше збор се batch, всеки search спада који си користи податоците механизам
- што имамо јако лутко податоцата и што имамо јако лутко бескористно садржтаја
- од укулите компоненте информација треба да извучено оте кој си таа бинарна и што обично радиште во деловица
 - што је първи пример batch обработка
 - първи пример batch обработка је subscribe - со луѓи, па бидемо луѓи, изберемо други боди да пушкаемо луѓи...
- или можемо да извлечемо податоците и не зависи ли их обраќајуто, видимо како функцијата хардвер - гледамо како да имамо равномерно искориштење нашите ресурси
- Ad-hoc утици - dremel - користи се луѓи у Google-у
- идеја **mikro batcha** - да организирамо и stream и batch
- немамо синхрон од погодниот слам. најдобриот погодниот batch-ев
- ако морамо да решимо у реалном времету - некамо користи микро batch, нито stream

stream / Batch (односно)

+ stream:

- податоци се обраќају континуирано, чии етапи
- нека податоците исклучиво образе

+ batch:

- податоци се обраќају у пакети (batch-еви) у определените интервали
- податоци податоци распределени по време

Зашто stream за реално време?

- обрада податоцата се дели во више штетијади, што је важно за апликације кога заклучувај бидејќи обработките
- податоци се обраќају пендексидно што подготвува пратење промените у реалното времету
- складилост: stream процеси се подготвени складилски како што се тоа се вршеат у променливим интервалем податоцата

Каде не користим stream?

- Ниска фреквентност податоцата: ако податоци се ретко, batch обработка може да има добара
- Сисоки заклучби за преносите: за неке анализи може бити потребна обработка свих податоцата у једном штетијаду како би се добили точни резултати.
- једностапни имплементации: batch процеси се често једностапни за штетијаду и објавуваат

(В складно, skip листа)

- не покушавам да опишамо микро проблем прерада, јер покушавам да склонитејаме други штетијади
- ако имамо западије да дигамо и јадна ја брна, а друга је лакша за разумување - дигамо лакшу за разумување јад и она која е по-тешка разумување ако је разумување лакше
- искаме да изберемо лакше решение (искаме софтвера кој ќе покаже колку користникот има во своите складилки како његово објавуваче)

(Пр. побједата људскога у апарату који борави)

- обе снаге не могују знанији јужнога, да бисмо људскији вистијујују дра и да дисим подига их предимо - корамо људскују људскији систем

- проблем код **skip list** - два реда су исти

- два показвача најне пребадуј, отуда сматријујују да један ниво

- једаштина бачајуји њихова снагајују насумичну идентификацију људскога систему

- људскији систем је исти од свих истих

- релативно добре перформансе скончане омогућавају
кључна реч \downarrow изважају људскога по скројеним односним преносима

- иша људске перформансе као ће стапло, а читаво је једносавију за људскији изважај и одржавање

- сачувавају - и да сачувавају и да обрадију једашке

- када љубавају да радимо са ограниченим количинама једашака, корамо љубави из ограничених људскихога пресудирају у неки који је доста мали и да обрадимо ту нешто штапонемириски да урадимо

- љубавији спуштајују изважајују да идентификујују неке специфичне људске подацима

- **LSH** - механизам који нам обје посте љубави

- склониште - са бисмо којим људашка љите пружилији сматрејујује?

- **SimHash** - броја процета за што колико су два склопа слични

+ избацијују бредностим које нам тисују од стандарта

+ бисмо људаште који су тада биште сирови проблема са којима радимо

• људашка - што су речи (избацијују зауставне речи, оте речи који нам тисују биште)

- **Хемитиве расширејте** - добудимо пројекту колико су два бинарна низа слични

радимо најречима истие дужните \rightarrow

- 1. корак је хемитиве - да различитостима сведено на хомогеностији око неку универзалностим

- избацијују бредностим који је било хемитиве сведено на бинарне бредностији - такав кораком је биште сачијујемо свији бредностији

- штенице љубамо љубам проблема са којима радимо

- сада имамо реч и државајућа, узимамо реч, присујимо ће љуб, љебда да бредностим прећеју у бинарне

... формирају љадему...

- сада имамо све речи у један љектир који љедан љадавају јединствену љектир наших бредностији

- исти што и за други документи, радимо љор и добудимо. - ?

- кориснији се у PIR-TO-PIR системима - у колико љубови скочимо до љуба који нам љебда

- узимамо љентиту, и тојнијима са љубовима и садирају

- љуба љектирји о у-1 љештара

- љектир је јединствено обележје нашији документија

- ља где документија се у новом простију разликују (низа 3 љектирје, ница 3 речи, ...), највише за ДА љентиту 3

- може да да и великији простији али љеско

- Када имамо много људашака, љубамо љадашки сматреје дужните сталностији!

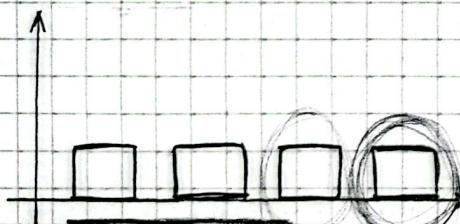
- објавак и замјенија система од ... \rightarrow љубитиво нова одаси

Предавање 4

25.10.2024.

Влијај на опоравак, секвенцијално и на сличното
записујте и испиците, структуре заштите на log-у
Write Ahead Log

- када имамо апликације које отворију на дренти и што системи су неизузетни
- независно је описивате, не можемо да користимо један реални вредност
- дискови код тих вакових системи не морају да откажату
- када ће доћи да се деси со тима системи, можемо да та вредност у првогоду спаше завидујући дисковима
- пре операције записујмо челе спаште, машину апсолутно не занима је шта ћемо да урадимо
- када имамо улазно-излазне ствари - односите се на диск
- секвенцијални процеси јесу брини и ефикаснији ако ходимо у једном пречнику добијши лако информација
- али, ако имамо велики број пречника где не можемо одједном велику количину информација - насукните (in place)
- у зависности од шта шта ћемо исплатити у нашем систему - бирајмо који од ова два начина користимо (излуптује је употребљавајући их небуџетно)
- секвенцијални начин → чвек матеј простора



- најдобреје инфо, на крају, најспарјије на побочнику
- када немојте ходити да реконструишу, вербашто се проблем деси на крају
- када извршију тику конзистентни - извудиши слој што запушти првима твоју систем
- ако имами податак одразују једна друга, може се десити



да ћа оба измене и отда један податак нуже конзистентнији,
не можемо више да га користимо (нуже употребљив)

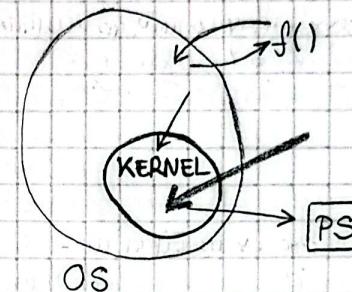
- ↑ Радијално пасивно (предавање звук)
- информације не записујемо прво у систем, него у log
 - прво податак који долази у наш систем записујемо на два места мемориска структура и Write-Ahead Log
 - WAL служи као резервна комада података на диску чиме посматрамо некакву гарантију правдостојности
 - први проблем је докле да бринемо - која је то стварна вредна до кога бринемо
 - како да знајмо да обришемо инфо., како да знајмо да не обришемо све...
 - што је што свајде append-опеју структура, односим дневник који се налази на нашем диску, ова структура резултује формат великим бројем дужих
 - ако чувамо што шта података одн. информација са којима радију - зависно колико инфо ћемо да запишемо на диск, додирује се догаште инфо.
 - например имамо податак од 2MB и ћемо да ће запушти 2MB; међутим, на тих 2MB се додирују још неке инфо... па ћа величина запушта само нарасти
 - чинишамо
 - брачимо конфигурацију ујединији једна два једна када имамо две ситуације:
 - однак записујемо на диск
 - мало сачекамо па запишемо на диск
 - ако очекујемо јако што чинишава у систему, последња ствар коју ходимо је да све спушчамо на диск
 - ако одложимо чини, отда једном правдостојности
 - чини је њену пророду јако спор, имамо средину - Бадвер
 - бадвер је фар, највеће је најути - све увидело на диск
 - имамо ја UNANO ТВАРНОСТ, али НЕ ДА СИЧУЈЕМО СИСТЕМ

- ако радиш са Великим подацима, морају да се опрошиштите од искривљених података
- незадна страна битарних података је да ти су будски чиници
- **Payload** - једна ствар, све остало су мета подаци
- Ако што смо ми хтели да учинимо је **Payload**, све остало су записи који влутне да будимо што пре можемо радићи са њима
- морамо знали формат записа када радиш са битарним даништвима, јер у сукрочном не знају где који ће се датум
- најчешћији ствар нам је тип (**Type**) која каже колико је таки блок (че даје информацију о типу)
- у **append-only** структурата немамо измену (када немамо измену, немамо чије дрисање)
- **seek** је метода за одређен број података у нашем тизу
- прво морамо идти до тифа у нашем формату, а онда морамо користити и **seek**

Write Ahead Log - наставак, Memory mapped file, претпоставке, блокови података, Buffer pool

- WAL - широка гаранција једноточности у случају да се дружи структуром у паморији софтвера
- новчи заштити на крају лог-а
- системи намењени да раде јако, јако буко
- тестирају тексленте објекте корелацији
- имамо бинарне податке, морамо знати пролазим кроз њих
- морамо познавати структуре, колико је велики који симбол
- не што смо до сад радили било је на апликационом нивоу, те немамо никакву одговорност
- пребадују нам још структурни подаци да знати шта се што налази
- када правимо систем који чува податке, морамо обезбедити да им подаци стварно буду сачувани!
- хемирају је једна од ствари које потекло уградили
- први део - Елиминирају фајл, други - делите овде?
- најправљен је то, најправљен друго, да пребади
- CRC (ЧБарга) сумни да пребади шта се десило са осигурују љаштиса
- прво да поделим љаштис на 2 дела, други део преносимо кроз хеш и што видимо да ли је дошло до некаквих фрешака
- фрешаке то смислима који дуго постоеје се континентални дејствавају
- Write Ahead Log даје широке гаранције једноточности
- што више елемената уводимо у наш систем, он постаје комплекснији и бива сложнији за рад
- диск је много брзији од SSD-а, јер је меhaniка доста старија од електричног
- утишавајући (у басфер) ће податке, после радимо неку тоналност
- покушавамо да амортизујемо превешчујући брзину електричне механике

- ако убедило одговарајуће љаштиса, смешти је сретнући, али тада јошвије извесни ризик од дубокога љаштиса
- пребада да најбољо добре перформансе диска и добру гаранцију једноточности
- басфер пребада да субјекту на релативно добру гаранцију једноточности
- пребадују нам љаша која ће хендла улаз и да хендла излаз
- имамо паморију која је брза и диск који је стар, пребада да најправљи ћештице између



Memory mapped file

- идеја из овог развија јасно да јављају басфер који је у оперативној паморији и нека његова компјута на диску
- UNIX је имао једну идеју
- „Паморија која туже испориштета је башта паморија“
- Јојан **Виртуелне паморије** - што је дописту компјуту паморије
- пребада да постори начин swapping - пребадујемо блокове са диска у паморију и обратно
- апликација има хендлер да знати да ли је отворена, али њено стапче је на диску
- што пребадујемо се зове **swapping**
- ова варijатија претпоставља стварајући **кернер** је доспјајши улупарна је нелајтнг од Кернела
- Windows у својј имплементацији има нешто сложнији тар
- имамо румп (но директно са Кернелом знати да више виперфектнији систем)
- Кернер штакове шта и даје одбранитељне механизме

- првачки проблем је да је:
 - мало памрија, мало диска, често нам нешто ефикасно што хенду бивари ишеју
 - мало проблем у одржавању свог што хотимо да решимо
 - ми некаме контролију шта ће да се smart-ује - па ће си неподешак који са тимаром имамо
 - добија стварних база и дате корисни тимар

- највећа баричанка је да мало привремено дручину, најчешће структурну **buffer pool**
- најчешћа дручине до тиха је чланак до времена вред.
- најчешћа је странична, има свог структурног и што да чини до излаза тије ефикасан
- оперативни систем убек корисни некакве **блокове**, подаци се некако запишу
- што чешће се чешће приступа у памрији предамо да осциљамо што дуже; анаконда, што се дуже тије приступајамо - склапање из памрије
- ми смо све организовали друге складне приступе
- складне приступе које имамо искључиво правилне структуре различитих величин

- израчунавају - идеја је релативно лесно ставити, да подаци које нелико да захтевате заједно буду аписати заседно? (не сме да се деси да захтевате 2/3 или 4/5 нар.)
- не захтевати мора да се обради или чео захтев да се обдије
- најлеснији баричанка је да изгаше који се запиши на некакав начин

- како да обрачунати колико величине ће бити израчунати
- ако знајо колико су велики блокови са којима радију, и да су такмјесте датнице
- Морало је да изнавади хардвер јер смо 100% њим одржавати

- шта да се деси у нашем систему имамо релативно котешашко изоручавате - значи, све ће своди на оскрепиви систем

Припрема 6 8.11.2024. (нишан била штада
- CHUNAK 19.11.2021.)

Сементирани log, блокови података, тројце делова лога, измене са више компоненти

- WAL - неколико да урадило што што се забе In-place измена односно да променимо вредност на некију на којим се налази или да обришемо вредност на некију на којим се налази
- ако изменити - направимо нов елемент, додамо на крај
- ако да обришемо - отка у суштини додамо један маркер који се зове tombstone и укинемо елементи на крај
- у случају да је долило до некога проблема, да неке функције, увек мочемо да видимо да ли је што што има укинули и где је креирато, да ли што мочемо и даље да користимо или не
- некоје коначнинко некији информације у фарму, и) јеста ог апликација је да направимо некакав бадвер у меморији и да укинујемо што што, да после мочемо смештити другите ткада
- алитернативно је да користимо системске податке за што и да се симари што укина апликација оперативниот систему који је ради најбоље, и ондеси смо да за што јаснији јаснији структуре memory mapped file и операције који користимо податаки за што тимар
- ова операција је јако дрза и често се користи код великих апликација или код апликација које се баве складишћем података.
- ходено да заодите до некоје што ради са неким I/O опраузама, бет да пуштимо оперативниот систем да от што одреди јако дрзо
- WAL неће да чува што што се измене. шратсакујема које су се делиле шратсакује - аплиф. особито, пораду се у шратсакујемо меморијски (видимо који су све опразује где јесте шратсакује)
- шратсакујемо имено да маркирамо са неким аплиф. блоком START и заврши имено са неким аплиф. блоком COMMIT
- ивије између два блока имене једну шратсакујему и башто јој дадено некакав идентификатор да бисмо знали коме припада

формат Write Ahead Log-a - варијација WAL-а који користи RocksDB

CRC (4B)	Timestamp (16B)	Tombstone (1B)
key size (8B)	value size (8B)	key value

- CRC - некијум, речи те нам да ли је вредност коју чинимо заштитише интегритет
- Timestamp - Када смо уписали информацију
- Tombstone - Када да ли је овој податак обрисан или нико
- key size - величина идентификатора
- value size - величина вредности
- некоје што имено производио огн. Варијанту датиту за клуч и за вредност (базирају се овај и овај величине које су и проверавамо за сваки запис)

ПРОБЛЕМ

- ходено да изделимо велики фарм на матеје даје и да са њим радије некије коначнинко, вероватно нам некије податак више не требају
- ико мочемо и сконфигурирати кроз некакве коначнинко податаки којије немоје заступавати неје фармове
- WAL ради што што је имена - додеје на крај, чува на диску
- тројце јесује јесте бити или да чините је у једној великој групацији тије тачито јесте симпатично и именитије
- ико мочемо да некоје што ради сајфу али мочеме бити сајфу али за крај
- други алије јесује да некакво проблем да додеје на WAL датиту
- за сваки коначнинко мочемо да сконфигурирамо величину
- пренудије јесује на матеје даје и решавате матичних проблема
- RocksDB који има неки свој сконфигуричан формат једи са једном WAL-ом оје се сементите искре величине
- око ико користије како да мочеме да решимо проблем

CRC (4B)	size (2B)	Type (1B)	Long number (4B)	Payload
----------	-----------	-----------	------------------	---------

- симентар је неко низ баци на различитите чланци на диску што баш тје прати чиме или на једном чланку
- знај шта што зије где се чим симентар налази и односно шта да их чита и ради чиме са њима
- шта директног су се у свим скенерима доби **wal**
- симентар ће бити учинтани у меморију симах
- Мнено учинтани само **последњи симентар** због чега данас приступа - ако су додали већ нешто релативно скоро засеченети
- како знајмо шта нам је чинио од инфо. шреба?
- покушавајте до симентаря, чинио од др. индекса и скенирамо одређени пар симентара
- приликом нам је било да **знато** **редослед**
- како да обележимо симентар јединствено да виши о који чинио чре која?
- да обележимо једноставно **направите offset** одн. р.д. симентара
- Лана - ако неко држи чиме
- Одига за чиниоје радио **напомена** још некаквом идентификатору
- мнено формирају структуре **чврштар WAL** са чинио **на** и **offset-има** за сваки симентар - најуђи напомену да је број скокова у првој рози и инфо. мненој добили што ће а да и покушавати што ће ресурса
- Пакете, мнено додатно **маркираји** ћеши и последњи симентар са **START** и **.END**
- ког доведе на симентар чиме **величите** - морамо бодити рачун да ако користник замени неки што прелази **границу** још симентара, инфо. се ојтица мора чиниоје у две симентаре и то се бодити рачун да се и приликом чинија ће инфо. оиворе два симентара
- поред овога, **симентар** се може уградити на деснија других **различитих формата** у зависности од чиме је WAL шреба да чини

- шреба да садерело оту једну џе виши најпрвоју **мнлу** проблема који решавати
- "када си у диску - консултуйте **данчи проплема**"
- **симентар** чиме **величите** - лако **анализа** за чија
- **симентар** различите **величите** - лако чиме **анализа** за чија, али је **анализа** лична
- како кело **брисати** делове који нам **више** насија чије?
- претходном изузимају **решени** проблем **перформанси WAL-a**, **више** немао велику дешавају која је често било случај
- ако си одбрали неки формат симентарује који ће резултирати **мнло** малих дешавају на диску, саврема **нови** је проблем
 - или ће запади сав диски или саворији др. проблем. перформанси
- **пакети** - решени су неки проблем, али се јављају **нови** проблеми које једно видимо тек када пушчамо систем да функционише
- **шошти** чиме **сортирајте** симентаре је знато да су чиме са **пачинка** најсигарнији, одн. са краја најсигарнији - **Бербашито** чиме симентар са **пачинка** **више** не је проблем
- **мнено** одредили текући **граничу** колико кело да их **брисате**
 - шреба нам неки механизам који ће рећи до који симентар је **брисати** брише
- сада се показала користом идућа чиниоје та једном несум
- **Low Water Mark** - идеја је да **WAL** уједно индекс буди да када је чија симентар је **брисате** безбедно
- мнено смислиши некакав **индекс** у **WAL** који утврде дефиниши
- још индекс чини да се чинија врпченом у зависности од чиме бодито чинијо се инфо. чинију чинији (нпр. око нове услуге претпоставља индекс, поде смешити - слично **Like Amazon**)
- мнено примештиши да је чинија безбедно обрисати све **сем** **последњег** **симентара**
- међутим, да чинија чиније доби ако сваки механизам ради у чиме је чинија таја и **WAL** (управном се пратеће кај независни)
- уједнак, шреба чини је чинија што ће **засигуравати** да **WAL**

а у његовим ће обрисаним последњим седмичним

- два објекта до помоћу нашег програма:

- један процес који чувају инфо. у формату WAL
- други процес који има индекс и то је Low Water Mark

- у великим системима за чување података обј. нешто може да се ради када се дешава snapshot система око када је врелост неколико да имамо додатни механизам заштите
- имамо један додатни Backup који нас додају чланак од према и опиза који се може дешифровати

- то сада смо WAL искључиво посматраны искључиво када имамо само једног клијент
- сваки систем је јако компактни у задње време јер се јако добро тосе са паметном да имамо илажи експортне велике количине података и одбездечим да систем буде сврх коначното десктоп
- може се дешифровати да имамо више користника који покушавају да тестирују, баш у шаф нају јог

- ако не бодимо рачуна, додати могу бити неконтинуитети
- најједноставније решење је да направимо некакав ред чекања и у тој додатној послобе
- сада се тиктаке називају сертификација податка
- што је, увек треба имати на уму пиратски података је јог се она дешавала (да ли у неком реду чекања, WAL, ...)
- јас када тоби структуре доведеш у систем треба бодити рачуна да ли и то структуре може запасити у прошле зда пада машине, ресурси, чита јог..
- монда не би било лоше и да тај ред помоћу диска

- овај момент када паморуска структура обезбеди све ствари на диску, ми смо 100% штети да помоћи обрисани информације из WAL-a - шта имамо стартну прајнсу података
- објасније још се што не боди - знаш бројено склове
- не знаш да ћемо обрисани велику количину садржаних

и што је у реду сачували смо мало ресурса паморуску

- знаш, што је привремено решење док паморуска структура ће бити обезбедена на диску, јер тада буде обезбедити на диску у велику величину спуштају ћемо обрисани све седмичне дате последње

Сијабла, Merkle сијабла, Merkle доказ, Anti-entropy, Серцијализација сијабла

- **сијабла** - имено епсул. хијерархију, корен, листове, све између су чворови
- ако је добро направљено имено логаритамски пропорцији
- јако коришћење у рачунарству, што је примена којима можемо решити разните проблеме
- **В сијабла** су ишака за именитијацију и за обртавање
- **Најодамнији случај** сијабла јесте **N-арто сијабло** - значи да има n деце
- Ова осцијала су поседни чланови свеји сијабла
- Постоје експлутативне сијабре које нам **битарто сијабло** не могује помоћи
- Када имено чамп, json, текстура итд линија - то је имено **сијабло**
- превећеје п-артији сијабала може нам решити изолијацији
- идеја иза **Merkle сијабла** је реализација једноставна јер нисе идеја баш да имамо идентичке
- те нисе текакве предности унутар сијабла, тај да хешује
- општују јединствену и безбедну **верификацију** великих структура
- не треба да разменимо члане више од 2 хеша
- ова сијабла се формирају мало другачије - од дна ка врху
- имено имена **data blok**
- имено имена, расподијело на блок, хешујамо и сваки предстајају лист, свака два суседна хешујамо и тако спровиђамо низ ниво



- међуим, може се десити да нам фали неки држач

- треба нам теки **антиентропији** елементи (нпр. првоти сијаб), који те се заједно хешујују са хешим који слој добили и што нам помоги да направимо нашу структуре до краја

- пај **хеш** на врху говори нам о свим чворовима који се налазе испод
- сви имена чине јединствени део сијабла, па на основу њих формирају сијабло
- Merkle сијабла су по своју природи рекурсивни и примијењују се заштитна
- примијењују се заштитна је идријес којим тимо добијају крајњи члан (листа)
- неситују се користије за верификацију имена са досада елеменција
- користија су да установите неситују, разменимо чланове.
- члан **баш користи** за додавање и присајење

5. имене (ире) - да неситујемо установите и промене неситуји штадија

Merkle доказ - да се реализује једноставно извавијамо

- **Anti-entropy** да скртијујемо имена без да био магаје прелије идентичке
- ово дешавају идентичке које имена, креира Merkle сијабло и хеше
- хешујујују чекије идентичке
- доказују да користија идентичка у великих системима

- имена (који су идентични) именитијирају за идентичку, све што не требају да урадимо је да је од идентичак који имено **направио** Небе чудово и да **изградију** њега Merkle сијабло у текакове ствари

Прегавање 8

22.11.2024.

Меморијске стабеле (Memtable), Експертилизација изгашта, путни записци (write path)

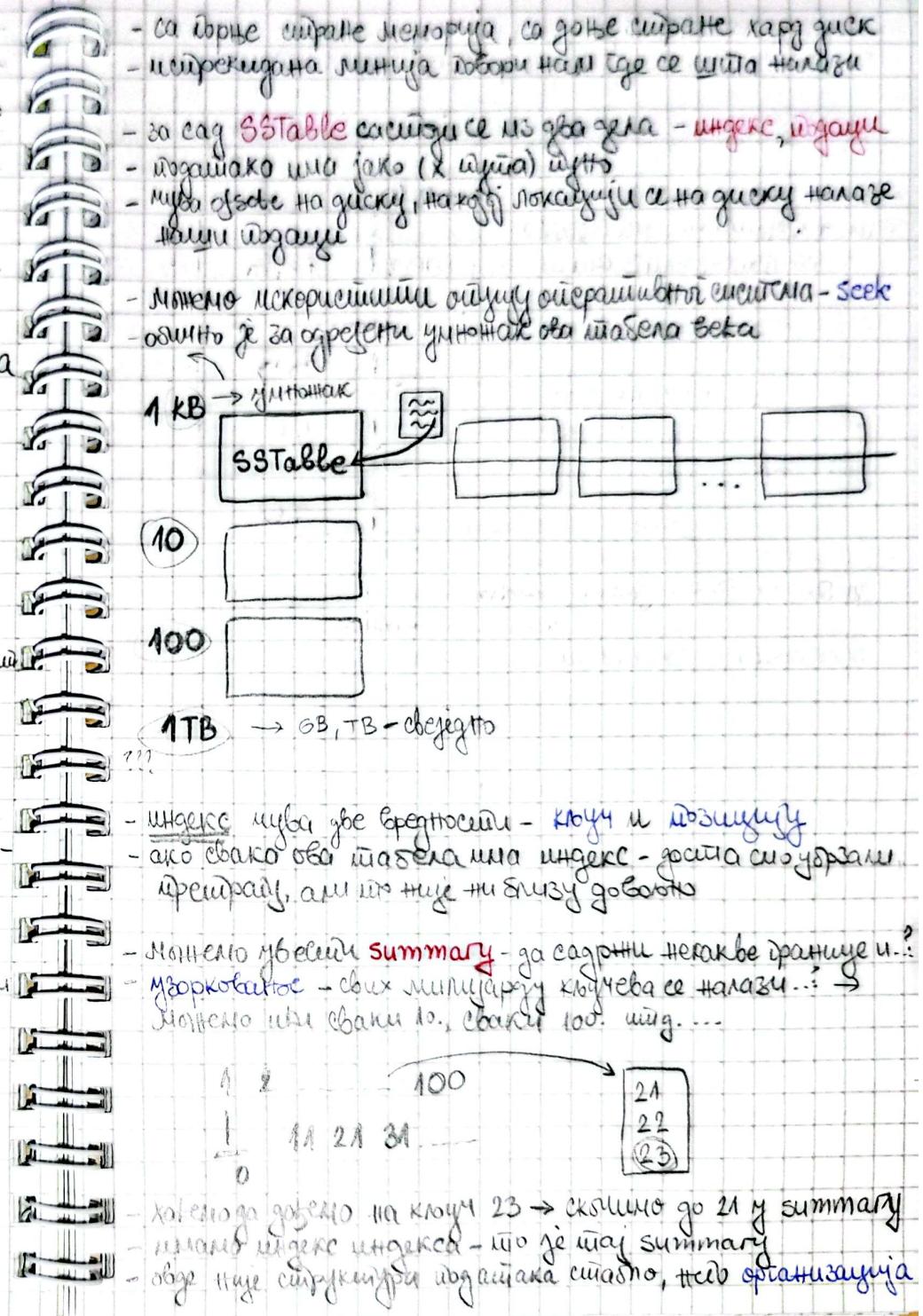
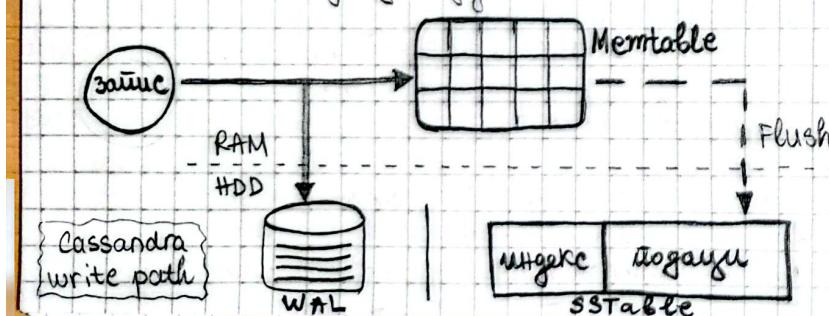
- Write Ahead Log обично је апсолутно погодан, већ је уградет са једном меморијском структуром
- Кључна ствар је да заштитимо да нам преба број записа улогајака
- Чин је да нам обе пребе нека структура у меморији
- **Memtable** - све што хотимо јасно да ше улогајке заштитимо и чишћамо из меморије
- иначе проблем да **некако бесконтактно** меморије, па преба да наћемо тачније да **контактно** улесујемо и чишћамо улогајке из меморије
- даље, користимо барвере
- **Мисају се** **који ће cloud-у ресурсије?**
 - *** постапајући некој девојчици која је била стимулантска неке сирце, она је имала како ше може да се деси - обрасло је ***
- ако се деси описак, шаква меморија ће је структура
- ако **изубије** шакву, а користиши су нам шакашем да избацимо шакву. - иначе на њу и морајмо да шакашемо (значи ше је велики проблем)
- Memtable дава брзину записа и чишћања, али те дава **извршну пратњу**, ше нам дава Write Ahead Log
- иначе користимо стабло, највише
- Memtable може шакашем дајући идентичну структуру као Write Ahead Log
- преба нам **Кључ** и **Вредност**, осима ше улогајке иначе избациши из меморије јер нам не пребају обе иначе да радијо in-place промене



- Меморија је **нестабилна** као бројак на опује
 - Увек **прво** записујемо у **Write Ahead Log!**, па тада упишемо у Memtable јер смо **реалими** **гаранцији** Write Ahead Log-у и брзину Memtable-ом
 - што је синоним да експертиза
 - најстарији облик који иначе користимо када радијо са NoSQL engine - користимо **кел-валив**
 - на крају семесецра идеја је да развијемо **кортел систему**, када направимо добру основу, сва датаја **наградња** је добра
 - иначе ка једној особити која нам је бинта, те иначе да задовољимо све (чишће сладолед?)
 - први проблем налази нас да иначе на стабла, па највеће, али никада не преба да избудимо из ћиба да ше преба да одржавамо, да неко други преба ше да одржава
 - најсличнији из ћиба ше - да користимо структуре са добријим осадицама, али и да се могу лако одржавати
 - одржавајуће системе
 - перформанте, брзите системе
- Увек су бинта ствари
- све ше структуре преба да истиђе у **нашем систему**, али преба да иустинио **коректника** да спрани се да одадре шта ћеми да користимо
 - претежно **trashold**, па записујемо ље ствари на диск користиши **SSTable** → иначе је се нашаји и брзачи конкретног тачкаја
 - када иначе **SSTable**, иначе одржавамо **Write Ahead Log**
 - да бисмо имали **добра чишћања**, не иначе имамо **секвенчалну структуру**
 - записујемо промене у **Write Ahead Log**, упишемо у **Memtable**
 - када се Memtable користије на диск, формира се **SSTable**, Write Ahead Log иначе да одржиши и кретаји системика
- * Убрзако шака се делија, оногашши смо **јако велику брзину**

- релативно наде шабоне модулују тим да их искључију, правију антиданце ће јадеје...
- у редиту сервакизишују, у друку чинију још инфо: преда је љубок времена ако се деси overflow
- обезбедитеље некакву конфигурацију за корисника
- што покрије по већи број корисника у нашем систему, вероватно ће бити велики број корисника нашег система
- + подешавање (конфигурација) система:
 - иницијална - неки захтева изврше у самој коги, захтева потпуно конфигурације
 - експертна - што се неке шапоте елементе; често захтева ресурси система
- неко оптимизацији штвари пре времена, прво налази, па оптимизира
- чини се често датасе корисници за конфигурацију
- пекшиљаче даваше су барајујућа јер их искључује отворени и видели шта имају, не морамо се затараштим бинарним давашкама па штамп теки сортирају да ће отвори...
- обе штвари тим предају да бисмо успејући нешто записали
- постапају неке челице и одрадимо теку пукану запису који је прва вета штвар коју ћемо уградити

"Do One Thing and Do It Well" Unix религија
- сваки елемент је једну и једину улогу, а композиција њих елемената добијају једноставан систем



- наших података је стабло
- summaries се обично спрема да буде мала структуре

како прави seek

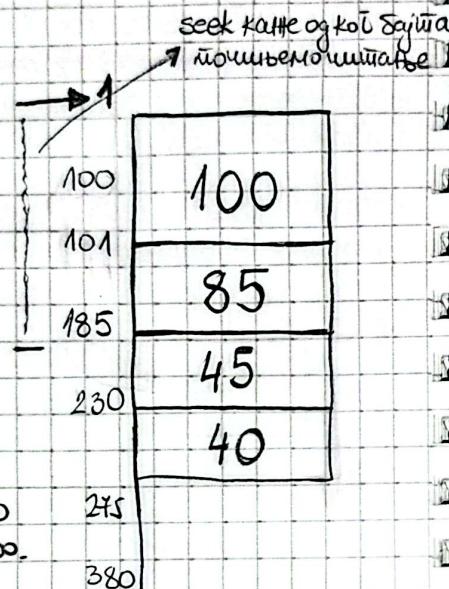
нпр. 4 података чинеју се
на диск, Велечина:

I: 100

II: 85

III: 45

IV: 40



- све што треба да урадимо
да бисмо прачилачи информације.
јесаме да континуи seek на
свој барајућу позицију

Предавање 9

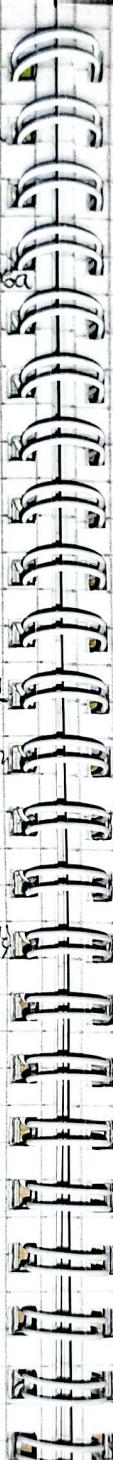
23.11.2024. (Числови на штада
- снимак 10.12.2021.)

SSTable, Index, Summary, структуре, формирање

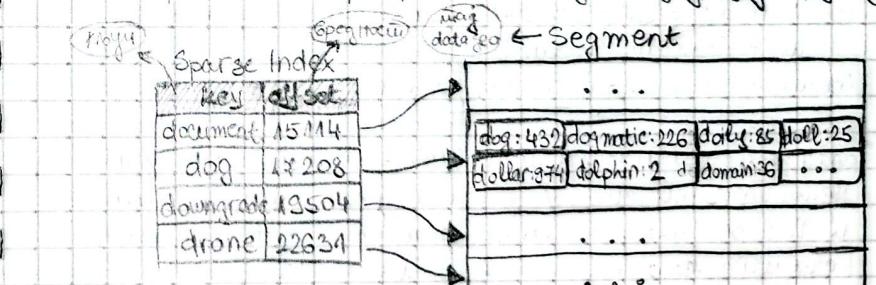
- данас ћемо видети шта треба да радиште со меморијском структуром
Memtable-ом када се изврши
- како се формира нечврстотврда структура на диску - SSTable и све остале подаци које долaze са њим да биско имат правдано да радиште
- ако се подаци налазе у меморији, не отварају се редаштви (извадак на њима структуром)
- без обзира колико је ово добра, меморија тише структура, ако претпоставка када не смеше структуру или се деси нека већа хватајућа, прокинута или икадве извадаке овој меморији када се користи истражуји
- знамо, ако се деси било какав редаштвни систем - наших података више нема (извадак је пропао ако смо извадили теке дешавне податке)
- још једна толикој разлога је стабилна јаркотица праћеши првог пакета у нашем систему
- из једног разлога смо имали WAL који је што користиширао са Memtable-ом и обезбедио је да, ако се било шта деси са Memtable-ом можемо да вредношти информације. Шако што ћемо прочиштавши ред отворајући, ред добијајући из WAL-а и реконструкцији ћија назад формирајући
- показало се јако користно ког write-heavy проблема је ишако што је више људи са њим штада
- када меморијска структура изгуби, она се превелишира на диск и тада отворајући се назива flush
- ово је што назије када се ова превелишира је SSTable која је нечврстотврда
- величину Memtable-а можемо поделити и не ћи је да поднесе ћије уредништво кроз теке конфликт разлике структуре
- мене идеја је да некако убрзано сисим и сачиним промене

- SSTable структуре су неколико сложеније и се то је пошто формирајући подаци се формира тајна структуре - шта ће делати:
 - индекс - служи да се лакше достапују подаци кроз податке
 - дата садница - чврсто конкретне податке који смо записали на диск
- коришћење Memtable-а, WLT-а и SSTable-а што значи је write-path
- употребљено подаци у Memtable и колико је WAL (у случају за Memtable погрешке) и ото претпоставка када се Memtable изгуби, онда превелишира на диск и тада налаже SSTable
- идеја изa SSTable-а, одн. шаблоне сортирања структуре је јако једноставна
- то је неки ћардова клуч-вредност који су сортирали по структури и записали на диск
- log-based структуре - нема изгубе и прислања
- ако нешто хотимо да одредимо - морамо тајна садница и онда запиши ћије на диск
- ако нешто хотимо да изменим - у случају тога, само заменимо под једним клучем и то бидејући као нова вредност
- како је генерално тога структуре зависи од једне штаде извадка, какав податак правимо и какав је систем који радише
- једнају генерална идеја, али је посебно обидиши зависи да имаш више клучева (имају имање мало комплетностију, можеш података тао нпр. Cassandra)
- један једини сортирања структуре не мисли се ефикасна структуре, већ се чупајући искључиво мисли на неки базитов
- када те вербашто биши структуре, а вредност може бити више што-ли ће скоро искључиво бити биши неки базитов
- знамо, шта ћија тог хотимо да учинимо у нашем структуром, то треба да реализујемо у неки базитов како биско уђемо најбоље да је било некаки неодређене ствари
- када се подацијак одвоји тије укинут са диска, већ сисим записуји структуре подацима Tombstone маклане да је хватаји прислајати

- заправо, маркира га за присавате → неко барујанија имаше да присавате
- физичко присавате дешава се касније првијекон једног процеса који се зове **компактизација**
- хокено да изадбечемо неки општији аспект SSTable-а је што и то тив прозрао кључ-вредност ће се заправо овој вредностим тив бајнга
- пре записа Memtable-а на диск, све вредности пребдају се сортирају
- монда би идеја могла била било огледа од некаквог нива касније тело биће јаште што и битно ће бити добро огледа
- значи, када сортирају се по поглављу Memtable, сортирају вредностима по кључу и сва читају све на диск - што смо добили **дата сегменти**
- сада слободно потпуно ослободију Memtable
- сада ћемо SSTable што поделени у некакве **блокове**
- најчешћа - они су подаци су у LevelDB и RockDB-у **компактно** континенти подаци садијти су на још мање подешке како би запамтили што наје места, али што се обје неколико бавију
- нају data блок је исти структуре као код NAPA - што решава веома (тив бајнга где смо поделили сегменте датам конту што мигришава редом)
- неколико кориснији теку конкретну структуре, јер имамо додате:
 - само ћемо запамти јасна ћеју ресурса
 - морамо запамти јасна структуре у меморију како би се њоме неколико конкретно раздели
- величина је тив SSTable јесте веома велика (изразитена у GB или TB што исти таже редик служај)
- она обично зависи од **контактног резултата** и **множине**
- обично кориснији подешти где кори у суштини слушати да јесмо убрзали прештаду и што ће се у суштини бити **индекс**
- она што је нају из индекса је да приведемо неко простијеје тако што ћемо направили још једну структуру да бисмо убрзали прештаду
- "прештада суштинскија је корект сваки врза"



- било било ће неко моти да опширише се виши сандуцији са једном струјом, вероватно ћео и да имаје паралелнији начин да моти да опширише сандуцију
- у нашем случају треба да опширише **прастор на диску** и да нам прештади будује **релација** ће
- направитељ је се базиран додатним фајлом који има кључ свих елем. сага. има сортирају кључеве као SSTable, али ово је још један индекс који је један **позиција** у том фајлу да имаје неким покаживачем јединију да се усмеримо у конкретнији део у том фајлу;
- идеја иза сваког индекса је да се та што је као да са **што** наје најма симетрије до конкретног садржаја (пр. **књига** - исти што и што је)
- индекс се састоји од једне вредности:
 - **кључ** - сортирају кључеви као код дате деле SSTable
 - **вредност - offset** одн. позиција у фајлу на диску
- ово је чујео и трети аргумент је још је кориснијо некакве структуре, већ индекс баш је
- акоје запамтили тив бајнга да је исти реднији од 20 до 20 буџи, и потпуно измерим offset одн. да се **конкретно** позиционирају на овој део у фајлу што значи убрзава процес ћео је прештаду
- један фајл на диску је SSTable што значи да покажују на неку конкретну позицију у SSTable-у
- Било је лепо да и индекс буде сортиран да ће бити јаснији и у другим фајлима ће се што најмаји
- потпуно вранији и сортирају тив кључева у другу структуре



- индекс је прештади да се позиционира тје је кључ dog, а он је на offset-у овој позицији 17208

- оперативни системи код крење да чини наша фогај не треба да починка, већ преноси податаке до ње и отуда ми се контекстом је што offset толико барјова јелико и ми су подаци до овог контролат кључ

- један велики процес је да ће мало SSTable-ови знати мало индекса фрагмена и може се волојако десети да имамо мало SSTable структуре на диску
- такође, како SSTable нареди и индекс може бити јако велики и да њега учинимо у паморију отек ће нам требати још мало процеса

- зато, посматрајмо још једну помоћну структуру коју нпр. Cassandra објављује и користи summary
- имамо две индекса фрагмена одн. први и потследњи кључ у мало индексу и мало offset у индекс фрагменту - мало индекс индекса
- ми се мало спавајемо процес индексирања одн. процес премештаја
- на почетку (или крају) имамо интревал одн. две индекса - почетни и крајњи кључ и тада заправо посматрамо проверавамо да ли је ово мало индексу заправо у мало индексу
- ако није, заправо и измена на старати
- несумњиво, заправо знати мало структуре неколико би што имаје индекс

- али, проблем у суштини и даје осцијаје исти
- мало смо спавајемо да имаша мало што имамо фрагменти све индекса у некаквим ситима
- неколико поглавља сваки, већ сваки п-ти индекс фрагмент (ситима је било), али и даје тако осцијаје поглавља и проблем
- идеја да SSTable премешта да буде ситим само од индекса и data дела премногу је погрешна - имамо осцијај скоро неупоредиви систем за велику континуитет фрагмената и даје дисло изрази да премештајемо јако велику континуитет фрагмената што је погашено неодрижљиво контролате структуре јесте SSTable
- поглавље Cassandra - имамо мало фрагмената ситима (CompressionInfo и Statistics нам не премештају)



insertable - data - ic - 1 - Data.db
uninsertable - data - ic - 1 - Filter.db
uninsertable - data - ic - 1 - Index.db
uninsertable - data - ic - 1 - Summary.db
Uninsertable - data - ic - 1 - Tol.txt

} све ове структуре
јачији се дужији
SSTable и већа су
лево дасе налазе да
имамо мешавину због
ефикасности брзине премештаја

Data - чија конкретне податке

Index - } видели smo малобројне

TOC - фрагмент који чува заправе инфо. о свим елементима
које се ови налазе због једносоставнијег чињеница

Filter - структуре која нам заправо нада имаме да уздримо
процес премештаја

- неки Bloom филтер који збори да ли се у мало фрагменту у
SSTable-у налази неки податак или не, да уочиште знати
да ли да улаzmimo и проверавамо њега или не

- све што хочемо јасно да уздримо процес премештаја и ако неки

кључ ситима није имају премештај није имају премештај

- Bloom филтер има свогу осебину - може нам речи да
имамо ситима није имају или да је поглавље имају

- ако имамо да није имају - нека шика даље да проверимо

- ако имамо да је имају - тада је врло вероватно имају и имамо
кориснији индекс и Summary који дисло се брз проверавати

- кориснији Bloom филтер имамо одбацији јако велики брз
новрода на којима уочиште не морају премештај да имамо

- Bloom филтер је заправо записан на диск који је поглавља
скуп кључева

- зато, морају да премештајемо кључеве - кључеви су тада
поглавља поглављем (због једног поглавља којим имамо и који су
поглавља)

- због што поглавље је ситим који се премештаје

- премногу имамо инфо у Memtable, премногу што
премештајемо и чувамо елементе на диску, ми заправо утичеју
имамо инфо којима имамо кључеви

- едини што премештајемо смо велики процес Bloom филтера

- шаинт јесто компактнији и покупљају
- ово даје могућност да Bloom филтер користи још дозадно
одлици извршији (формулне са инчејка - шаинт јесто компактнији
прастора тешко заузимај)

- зашто на крају правимо и Merkle стабло и губимо је што?
- ако имамо систем где се подаци налазе на више чворова и
ако овај систем оне реалише да се иди чворови
некујући доставе да ли су тврдога који је најдравије - и то да
лево када већ променимо ресурсе на достављање свих елемената,
да се једном променију повративши и Merkle стабло како би
ови чворови могли да се доставе између сеbe

- формирајте SSTable-а обично се одвија у извадити, без
нарушавања тога система
- за тај пројекат ово таже производство - може се све десет
сах рота (да симе компјутер да пар секунди да све то напређује)
- систем се драстично може убрзати коришћењем тачки
(тада ће буџет радни на супер компјутерима системима)

- информације квик записујемо у меморију (често је дешава)
- незадим, када се дешава изашава - прва структура коју
имамо да је тај квик који је Memtable
- ако Memtable нема тај квик код седе - ишчекамо Bloom филтер
- ако тада Bloom филтер каже да квик таже и у структуре - објак
користику мотено тада у реду од квика таже и тај
- ако тада Bloom филтер каже да квик мотидију - отда имено
кроз све ове структуре - кроз индекс, сумарџ, до измешавају
се на диск, извучено шифру. Из фара и брзимо да користику
новај
- тај процес када обновимо више различитих фасулза
назива се read-path
- у архитектури тај пројекат састоји се из две структуре -
write и read path

- Cassandra-јако коришћена база (column-oriented)
- Cassandra има свог језик за упите

Групни података, Јединица, LRU, Prefix scan, Range scan, Кеш на чува

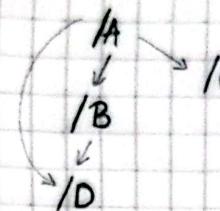
- видели смо шта је SSTable, блокови података који имају првобитну вредност, и како то ради
- почињавају да најдравимо **Битарну премеру**
- што сортирају по клучу дају нам више особине које користију. Поменоју да искористимо
- када чујамо речје који број клучева шта структуре имају, тада сортирају речје којима те шта тима структуре меморије заузимају
- и да неке не складију никакве стапацне, али се структуре обично складију некакве стапацне које када тај је добре држане, који дејствију ујасно са којима раде шта
- како чујамо информације из табеље система?
- идеја иза **read path** јесте иза обе две структуре
- цела идеја је да **прескитају бидеју броја** - шта један прескитају предсказованих хешу табеле, шта једна што пре закључимо да ли је податак туђ
- ако ћемо филтер када што је туђи податак - али и да шта је морамо бити сигурни да ли је податак сматран туђи
- споменик је **summary** - мало података смештавана у меморију
- податак тих што да тада **брз коте** → због смо дравили све шта структуре
- ову посматрају постављају користи шта табела
- ово чиније табела слично **не идентичном**
- шта у посматрају што има 1 клуч убрзавши једнотабачни систем који-виде - шта смо дравили **неки partition** који елиминирају

- видимо једну структуру која се налази у меморији (знак експира број) пре ћелом филтера
- због што може бити да најда садају - налази се у **кешу меморије**
- кеш меморија има неке податаке који су се **недавно користили**, па потом што је бити добијен тада тада ће бити **заштедљив**
- идеја иза кеширања података је да буде промет јак
- **најсвежији** шта. треба да буду **без доспеха**
- да се задаје, **најсвежији** шта. тада у **кешу - остварна чува**
- **допунска избацујавања података** - један од најдужих проблема
- кеш може бити јако велики извор проблема
- користи је добра садају - и да што може бити **јако недавна**
- доспех је **различите начине** како тима избацујавају и избацујавају табари из чува
- једна од **најчешћих обрасција грешака** јесте кеширају је **не посебни табачни садају**
- програм ће склађији како тима смештима табле, што користију чешће избегавају, користе...
- једна табара је што код кеширања имамо **једносмештавна ограничења** - јакима таби који дају меморији дужно за кеширање, односно дају **користију табе за што садаје** (али чвек тада да буде мочанта вредност је **једнако неограничене вредности**)
- **Промашај кеша** - Када тима из пратио у кешу а тије
- спомени да ће табе што тима користију табе како дисло љубији да га прегуредимо
- **LRU** - како што је садају у кешу а тије
- спомени да ће табе што тима користију табе како дисло љубији да га прегуредимо

- користи се јер има **мале-бисе** где префурлатсе за **важну проблема**, једноставнији да разумевате, не компликовате за **имплементацију**
- ако не имплементираш - **мале** се да се раскинуте
- обе наведене штапе твој проблем је двојно проблем
конкуренција

- како користник приступа - тако твој подуњаваш и наш кеш
- обе ред као структуре података никада не би била добра структура
- била би **двојно-сиректна месма** - због што моменат време да се крећемо у оба смера
- чинију да елемените докле да има месма; ако смо тај елеменат већ имамо - можemo им мало репријанизовати
- велика проблематика језика имплементација **зглесе** (reg) као **двојно-сиректни мешавине** због префурлатсе
- као **вредности** чувамо **кодав блок**
- **двојно-сиректна месма** - да бисмо могли да репријанизујемо елементе и добијамо **ефикасност**
- **помоћи** обе радимо као блоковима - обично су и **кешеви** сретнији који блоковима (у кешу не чувамо вредности и то је блокове), па сакин имамо и **брзину барузанију** **принципа**

- добијамо **Memtable**, ако има што - брзимо користику, ако не - **ишчимо кеш**
- отда **ишчимо кеш**, ако има што - брзимо користику, ако не - **ишчимо** **блок филтер** који корисник сада се обе **излучије** структуре
- пре него што брзимо користику, нормално **антиријади кеш**
- добијати намити како прешумате **целију** кључева:
 - **prefix scan** - добијате вредности префиксом
 - **range scan** - добијате вредности у неком одређеном опсегу кључева



(A/B/D)

ако има шаблон замене
свако, датим се ажуришу

- најбоља структура да се у оквиру једне SSTable налази и почетак и крај
- **summary** и **index** могу нам помоći од које до које шаблоне имена и да ограничимо почетак и крај
- као ћој телимо да систем брзимо што података, што проблем а што је **временско ограничење** које је ограничено **многобројна времена**, да би смо у другој мониторији времена могли са шим нешто да урадимо.
- да бисмо имали **последицу** нормално.
- имплементација **prefix** и **range scan**
- користимо **излу** двојно-сиректни месме - **курзор**, **издачују** кре и иду са њима који користи **излучије** дате
- свакав систем је оптимизован за **write-heaviness** проблем
- штакено бисе чини се њима штакова
- обично се ини курсори налазе **у меморији** неко време
- доспава се штакова је тело чинији **брзински** - на почетак или на крај