Óbudai Egyetem

Doktori (PhD) értekezés
tézisfüzete

Adattömörítési és állományszervezési eljárások adatfolyam és kötegelt feldolgozású környezetben

Finta István

Témavezető:
Dr. habil Szénási Sándor

Alkalmazott Informatikai és Alkalmazott Matematikai
Doktori Iskola

Budapest, 2021.

I. A kutatás előzményei

Monitorozó rendszerekkel kapcsolatban természetes igényként fogalmazódik meg az az elvárás, hogy valós idejű képet kapjunk a monitorzott rendszer állapotáról. Azonban a monitorozott rendszer mérete és bonyolultsága, a megfigyelt adat jellege, a megfigyelő hatása a rendszerre és számos egyéb tényező nagyban befolyásolja azt, hogy milyen mértékben lehet ezt az igényt kielégíteni.
Infokommunikációs hálózatokban a hálózati csomópontokban történő eseményekről, forgalomról, stb. rendszeres időközönként szinkron, illetve eseményvezérelten aszinkron módon helyi mérési adatok keletkeznek. A hálózati csomópontokon létrejövő mérési adatok egy részét el kell juttatni a permanens tárolóegységekbe, illetve elő kell készíteni a későbbi feldolgozás számára. Egy hálózatba kapcsolt kommunikációs rendszer megkövetelei, hogy az adatgyűjtés során ezt a fajta elosztottságot figyelembe vegyük.
A 2010-es évek elejéig a telekommunikációs hálózatokban, a hatósági előírásokkal összhangban, jellemzően a pár perces periódusidejű nyers mérésektől kezdve, a több hónapnyi aggregált, illetve származtatott adatok általánosságban a hagyományos, kötegelt feldolgozás keretében voltak elérhetőek, utólag. A forgalom százalékában mérve elenyésző volt a valós időben megjeleníthető információ, aminek főleg technológia és gazdaságossági okai voltak. Tehát, egy hálózat paramétereinek menet közbeni finomhangolása többnyire históríkus adatok alapján történt. Ugyanígy a meghibásodások előrejelezhetőségét is korlátozta ez a késleltetés.

A 2000-es évek második felétől azonban a valós idejű adatok nagyobb arányú gyűjtését és feldolgozását két különböző területen történő változás is lehetővé tette, illetve elvárta:
Egyrészt elérhetővé váltak az olyan felhőszolgáltatások, mint például az AMAZON AWS [1], amelyeknek a segítségével elméletben az üzemben lévő számítási erőforrások pillanatnyi mennyisége mindig az optimálishoz közeli állapotot teszi lehetővé. Ehhez viszont elengedhetetlen volt a mérési adatok nagyobb arányú valós, vagy közel valós idejű feldolgozása.
Másrészt addigra beértek azok a kutatások az internetes techcégek és az akadémia részéről, amelyek segítségével addig nem látott méretű adatokat lehetett tárolni és feldolgozni. Ezeket az úgynevezett enabler technológiákat manapság Big Data névvel illetjük. Kezdetben természetesen itt is a kötegelt feldolgozás jelent meg, aminek egyik jó példája a Map-Reduce programozási paradigma Google-től [2] (ami már 1985-ben megjelent egy diplomamunkában), a Google File System (-GFS) [3] és a belőle építkező Hadoop Distributed File System (-HDFS) [4], vagy később a MAPR [5].
A social media rohamos terjedésével egy időben, hirtelen akár több tíz/száz milliónyi feltöltött posztot, fényképet kellett a korábbiakhoz képest igen rövid időn belül rendszerezni, és felhasználók falán a releváns tartalmakat megjeleníteni. Ebben a közel valós idejű feldolgozásban a 2010-es évek elejére élen a járt Twitter az általa fejlesztett STORM [6] adatfolyam-feldolgozó keretrendszerrel, az alsóbb rétegben olyan üzenetküldési technológiákkal, mint ZeroMQ [7], NETTY [8], RABBIT MQ [9] vagy a KAFKA [10]. Az adafolyam feldolgozásban akkoriban meghatározó volt a Google Millwheel [11], a LinkedIn a Samza [12], illetve a University of California, Berkleyből a Spark [13].

Az újonnan induló cégeknek, startupoknak nem okoz(ott) gondot, hogy az üzleti folyamataik/modelljük idevágó részét a fenti informatikai rendszereknek megfelelően általában java, clojure vagy python nyelven implementálják. Viszont a már meglévő cégek többségének az SQL volt a defacto szabvány az üzleti / informatikai folyamatok terén. Annak érdekében, hogy a nagy adatok feldolgozására alkalmas, olcsó hardvereken futó Big Data rendszerek illeszthetők legyenek a már létező cégek létező SQL alapú üzleti / informatikai modelljeihez, megjelentek a felülről SQL kompatibilis köztes rendszerek, mint a HBASE [14], HIVE [15]. Ezek a köztes rendszerek az SQL utasításokat adott esetben például Map-Reduce jobokká transzlálták.
Egyéb adatbázis alternatívák near real-time feldolgozáshoz: Cassandra [16], CouchDB [17], VoltDB [18].

Továbbá, ezeknek az elosztott rendszereknek az üzemeltetéséhez szükség van olyan felügyeleti eljárásokra, amelyek biztosítják a keretrendszerek folyamatos működését, hibatűrését, belső monitorozását ZOOKEEPER [19], GANGLIA [20].

Időközben megjelent a LAMBDA architektúra [21] elképzelés, ami keveri a folyam illetve kötegelt feldolgozású paradigmákat, olyan optimumként, ahol ugyan nem feltétlenül a leggyorsabb feldolgozási sebességet kapjuk, viszont a kötegelt feldolgozású módszerekben jelen lévő permanent storage miatt jelentősen megnő az ilyen hibrid rendszerek adatvesztéssel szembenni ellenállása.

II. Célkitűzések

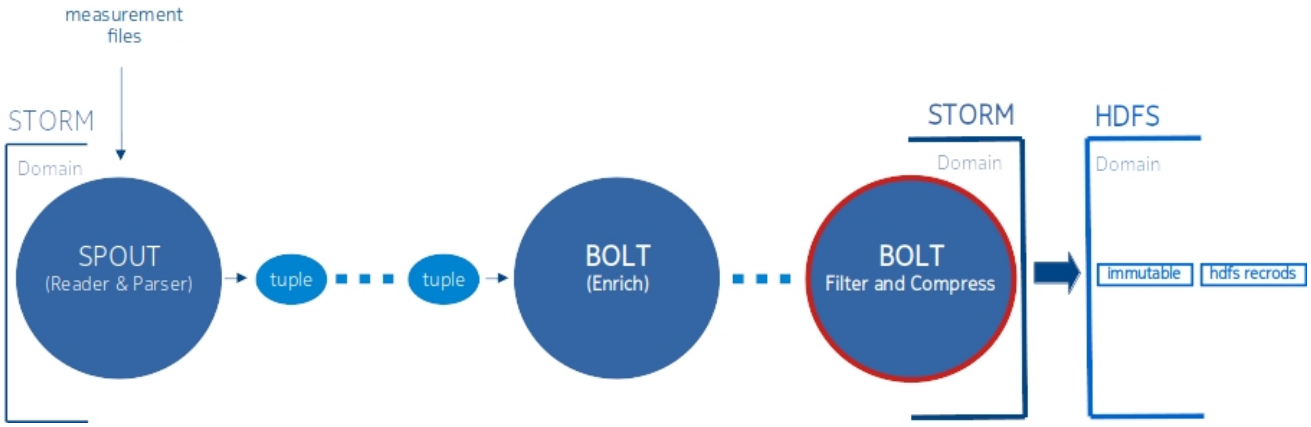
2013-tól a Nokia kutatómérnökeként részt vettem egy olyan ipari kutatási projektben, ami a fent hivatkozott technológiák mobilhálózatokban történő alkalmazhatóságát vizsgálta. A projekt mögött két erőteljes motiváció is húzódott: egyrészt a mérési adatok minél nagyobb arányban történő valós idejű feldolgozásának lehetősége. Másrészt a hagyományos, vendor lockolt relációs adatbázisok elméletileg olcsóbb, open-source NoSQL technológiákkal történő helyettesíthetősége.
A munkám során egyfelől össze kellett állítanom egy olyan futószalag architektúrákat, amely biztosítja, hogy a hálózaton keresztül begyűjtött, illetve már a stream processing környezetben létrejövő, újraküldésből származó többszöröződött mérési adatok nem kerülnek be a végleges tárolóba (HDFS). Ezek a szennyeződéséként megjelenő duplikációk/többszöröződések ugyanis a későbbiek folyamán vagy a származtatott mérési statisztikát torzítják, vagy a tisztításuk jelenik meg jelentős feldolgozási költségként. Ennek érdekében számos hagyományos, illetve a fent már említett in-memory adatbázist vizsgáltam. Azonban az elvárt nagy feldolgozási sebességet egyik ilyen megközelítés sem volt képes a külső hálózati kommunikációs költség mellett teljesíteni. Így a figyelmem a Java Collections Framework (JCF) [22] keretében elérhető „beépített", memóriában működő SET és MAP absztrakt szerkezetekre terelődött, amelyek hátterében vagy valamilyen bináris keresőfa [23-24] (AVL-tree[25-27], RB-tree [28]) vagy hasítótáblázat (Hash [25]) áll. Mivel az általunk alkalmazott rendszerek mindegyike moduláris, így megvizsgáltam más adatszerkezetek, mint például a B-tree [29-30], (a,b)-Tree [31], Interval-Tree [32-34], majd egész layerek, mint a Bloomfilter [36], a Chord [57] (ami egy elosztott hashtábla) használatának lehetőségét is.
A fenti megvalósításokkal kapcsolatban vagy az a probléma merült fel, hogy nem volt pontos a szűrés, vagy pedig az, hogy a szűrőknek a beérkező kulcsokkal együtt 1:1 arányban nőtt a tárhely igényük. Ezért mindig létezett egy olyan hosszú időtartam, amikor már valamilyen ablakozó eljárás keretében ki kellett venni a szűrőből a korábban már megérkezett adatokat, hogy legyen hely az újabbak számára. Ezzel vagy teljesen kizártuk a későn érkező adatokat, vagy pedig megengedtük a duplikáció/többszörözés, és így a szennyezés lehetőségét.
Feldolgozási sebesség szempontjából a fák esetében nem éles problémaként jelentkezett az egyre több kulcs tárolása, hanem logaritmikus léptékben, fokozatosan romlott a teljesítményük.
A Hash szerkezetek az elvárható O(1) időbonyolultsággal dolgoztak egészen az előre beállított kapacitás eléréséig, ami után a megjelenő re-hashing képes volt annyira lerontani a teljesítményt, hogy a feldolgozási láncban lévő korábbi memóriabufferek megteltek, és fellépett az úgynevezett SWAP-pelés jelensége, amitől összeomlott a rendszer.
Ezért a cél egy olyan hatékony szűrési mechanizmus kidolgozása volt, ami a korábbiaknál sokkal memória barátabb, és inkább az átlagos O(1) időbonyolultsággal legyen leírható. Így kidolgoztam egy fix költséggel jellemezhető egyedi sorszámkiosztó eljárást, ami egy számítás útján ugyanahoz az állományhoz mindig ugyanazt a sorszámot, ugyanazt az egyedi azonosítót rendelte, az adat belépési számától és pontjától függetlenül. Az adatok utazása során ez az azonosító váltotta fel a mérési álományok beszédes neveit, és lehetőséget biztosított arra, hogy a nagy sebességű szűrőként is viselkedő folyamfeldolgozóban a végleges

tárolás előtt csak ezeknek az egyedi azonosítóként viselkedő sorszámonak kellett a hatékony szűrését megoldani.

A másik feladatom az önmagukban kis méretű, jellemzően néhány KBytos mérési adatok hatékony tárolásával volt kapcsolatos: a kis méretű állományok egyesével történő gyakori küldése ugyanis szükségtelenül rontja a hálózat teljesítőképességét, önálló tárolása pedig a big data fájlrendszerek 64-128 Mbytos(!) blokk méretét figyelembe véve nagyon gazdaságtalan, és jelentősen csökkenti a fájlrendszer kapacitását. Ezt hívják az irodalomban „small file problem”-nek [37-39]. Ennek elkerülésére léteznek úgynevezett csomagoló módszerek, ahol Sequence vagy Map konténer fájlokba [37] ágyazva tárolják a kis fájlokat. Mivel a naponta keletkezett adatmennyiség már egy közepes hálózat esetén is napi több 10*Gbyte, amit a hatósági előírásoknak megfelelően legalább két évig meg kell őrizni, így szóba jött a különféle tömörítők alkalmazása, mint például az LZ családon [40-41] belül az LZ77 [43], LZ78 [44], LZW [45][47], LZAP [41].
A tárolandó fájlok esetében fontos paraméter az úgynevzett access-pattern [37], ami megmondja, hogy várhatóan milyen típusúak és gyakoriságúak lesznek hozzáférések. A mérési fájlok a későbbiek során nem módosulnak, így ebben az esetben a WRITE_ONCE__READ_MANY_TIMES a várható access-pattern. Ezt a tömörítők esetében is érdemes figyelembe venni, ugyanis az enkódolás és dekódolás idő- és tárhely bonyolultságai különbözhetnek a választott tömörítési eljárás függvényében.
Az access-patternt a Google például olyan szinten figyelembe veszi, hogy a „brotli”-ban [48] (2013-ban létrehozott szótár alapú tömörítési eljárás) egy előre elkészített, fixen besúlyozott, hozzávetőlegesen 13K bejegyzésből álló szótárt alkalmaz, ezzel is gyorsítva a dekódoló oldali eljárást. Két évvel korábban, 2011-ben a Google bemutatott és a fenti big data technológiákban széles körűen alkalmazott egy „elfogadhatóan jó” tömörítési aránnyal rendelkező, de nagyon gyors tömörítési módszert a Snappy tömörítőt [49].
A facebbok pedig a LZSS és a Huffmann kódolási [41] eljárások ötvözéséből hozta létre Zstandard-ot [50-51] 2016-ban, szintén a dekódoló oldalra helyezve hangsúlyt, a lehető legnagyobb tömörítési arány mellett.
A fenti módszerek áttanulmányozása után a célom egy olyan viszonlag gyors, memória barát, könnyen átsúlyozható, az LZW-ből virtuális szótárkiterjesztéssel származtatott, általánosan is alkalmazható szolid tömörítési eljárás kidolgozása volt, ami a legkedvezőbb esetben nagyon gyorsan közelít az elérhető ideális tömörítéshez, és még legkedvezőtlenebb esetben is aszimptotikusan optimális marad. A testreszabott eljárás kidolgozását indokolta a mérési fájlok fejlécére jellemző, gyakran és hosszan ismétlődő szakaszok jelenléte.

III. Vizsgálati módszerek

A helyzetelemzés során a fent leírtaknak megfelelően a kísérletezés játszott szerepet a kutatásaimban, amit valós hálózatokból származó adatokkal végezhettem. Az kidolgozott új eljárások és azok elméleti átlagos viselkedésének, illetve szélsőértékeinek meghatározása során az elméleti-logikai kutatási módszereket alkalmaztam.
Az ellenőrző vizsgálataim során részben matematikai módszerekkel jellemezhető bemeneti mintákat konstruáltam, részben pedig a már említett valós hálózati adatokat használtam.
Ezekkel a kísérleti adatokkal meghajtottam a prototípusként implementált tömörítési és szűrési eljárásokat és mértem az olyan indikátorokat, mint például lépések száma , memória használat, kompressziós arány, intervallum-fa magassága, stb. Végül, ahol lehetett ezeket az eredményeket összahasonlítottam a korábban alkalmazott módszerekkel.
Egy ilyen prototípus elrendezés látható az 1-es ábrán (forrás: [S-1]).



1. ábra: Vizsgálati környezet, pirossal a szűrő és az enkódoló helye

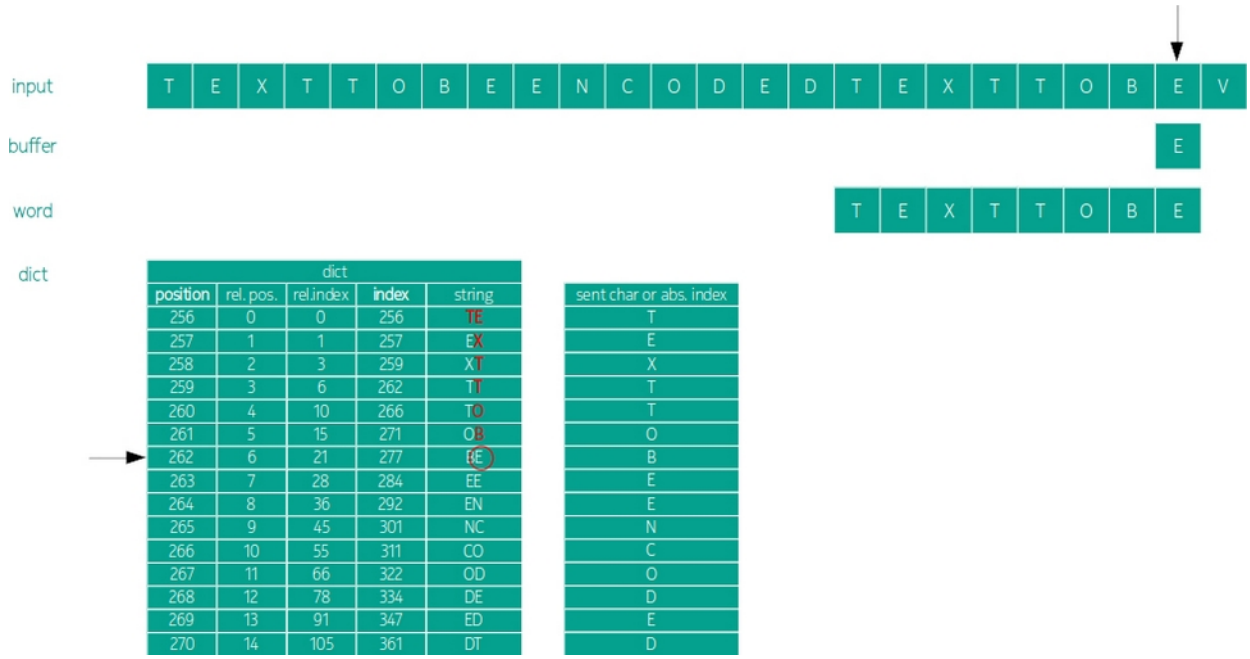
IV. Új tudományos eredmények

IV.1. Téziscsoport – Veszteségmentes adattömörítés

IV.1.1. *Tézis – VDE Tömörítési módszer: Kidolgoztam a VDE-LGD veszteségmentes adattömörítési eljárást, ami biztosítja veszteségmentes tömörítőkkel kapcsolatos azon követelményt, ami szerint a bemeneti A_1 adatra alkalmazott E tömörítési eljárás során kapott $A_2=E(A_1)$ tömörített adatra alkalmazva a D dekódolási eljárást eredményül visszakapjuk az eredeti A_1 bemeneti adatot.*
Képletben:

$$A_1 = D(A_2), \text{ahol } A_2 = E(A_1) \text{ vagyis } A_1 = D(E(A_1)).$$

[S-2] (ICIN2016)
A VDE-LGD tömörítési eljárás az LZW, szótár alapú vesztegségmentes tömörítési módszer egyfajta kiterjesztéseként fogható fel. Az LZW esetében a tömörítési eljárás azt jelenti, hogy az egyes szótárbejegyzésekhez rendelt sorszámokat tároljuk le a bejegyzés helyett.
Az enkódolás/tömörítő oldali VDE kiterjesztés azon alapszik, hogy minden szótárbejegyzéshez a szótárbeli pozíciójából származó sorszámon túl hozzárendelünk egy újabb azonosítót, egy úgynevezett indexet. A közvetlen bejegyzésekhez rendelt úgynevezett elsődleges vagy primary indexek, a nulla és egy pozíciót leszámítva, eltérnek pozícióbeli sorszámtól. LGD esetében éppen a ***háromszögszám****ok*-nak megfelelően növekednek. Ez a kiterjesztési eljárás lehetővé teszi azt, hogy a primary indexek közé eső virtuális indexekkel a már szótárban lévő bejegyzések konkatenációit azonosítsuk. Így, amennyiben a pozíció helyett az indexeket tároljuk le, kedvező bementi esetekben sokkal hosszabb szavakat tudunk helyettesíteni számokkal, mint amit pozíciós helyettesítés lehetővé tesz. Egy ilyen virtuális szó látható a 2-es ábrán (forrás: [S-3]).
Dekódolás esetében, a dekódoló oldali szótár felépítése során kihasználjuk azt az információt, hogy primary indexeknek ***háromszögszámok****at* kell követniük.



dict

position	rel. pos.	rel.index	index	string
256	0	0	256	TE
257	1	1	257	EX
258	2	3	259	XT
259	3	6	262	TT
260	4	10	266	TO
261	5	15	271	OB
262	6	21	277	BE
263	7	28	284	EE
264	8	36	292	EN
265	9	45	301	NC
266	10	55	311	CO
267	11	66	322	OD
268	12	78	334	DE
269	13	91	347	ED
270	14	105	361	DT

sent char or abs. index
T
E
X
T
T
O
B
E
E
N
C
O
D
E
D

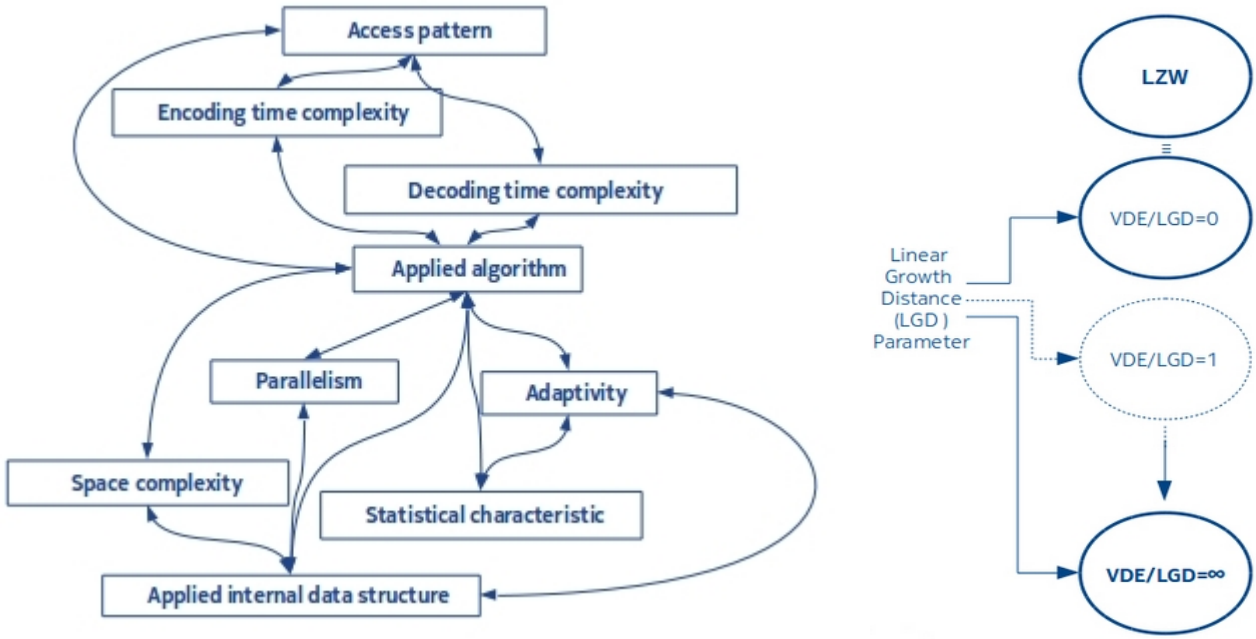
2.ábra: Szótár építés példa, a virtuális szó pirossal jelölve

IV.1.2. *Tézis – VDE Analízis: Kidolgoztam egy tárhely-intenzitáson alapuló modellt, amely segítségével az enkódoló eljárás bemenetére küldött adat hosszának és statisztikai tulajdonságainak szótáron hagyott újjlenyomatainak függvényében lehet meghatározni a VDE-LGD tömörítési eljárás tömörítési arányát, tárhely és idő bonyolultságát. A modellezés során bevezetett R_b (required bits) és f_m (final match) metrikák segítségével pedig a VDE-LGD módszer összehasonlíthatóvá vált az LZW tömörítési eljárással.* [S-3][S-8] (CINTI2016, SACI2021)

A bevezetett modell alapján az enkódoló oldalon a következő extrém esetekben végeztem el az összehasonlításokat a 3-as ábrán is látható függőségek közül:
- leginkább tárhely-intenzív bemeneti adat esetében az aszimptotikus tárhely bonyolultság LZW során: $S_{LZW}(p) = O(p^2)$, összehasonlítva a VDE-LGD: $S_{LGD}(p) = O(2^p)$ értékével, ahol a p az egyes szótárbejegyzések pozícióját (sorszámát) jelöli;
- a legkevésbé tárhely-intenzív bemeneti adat az LZW során egyértelmű és a növekedési rendje megfelel az $S_{LZW}(m) = \sum_{i=1}^{m} V_{S_{in}}^{r,i+1}$ képletnek. Ettől eltérően a VDE-LGD legkevésbé tárhelyintenzív bemeneti adatának meghatározása a konstruálás során fellépő rekurzív függőségek miatt csak szükséges feltételt bebizonyítottam, aminek a létezése mellett a fenti $S_{LZW}(m)$ képlet felső határként a VDE-LGD aszimptotikus közeledése mellett is igaz marad, azaz $S_{LGD}(m) \approx \sum_{i=1}^{m} V_{S_{in}}^{r,i+1}$;
- a tömörítési hányados a tárhely-intenzív bemeneti adat esetén írható, hogy

$$CR = \lim_{f_m \to \infty} CR(f_m) = \frac{R_b}{log_2(S_{in})(em_{S_d}+1)}$$, és véges szótárméretet alapul véve bebizonyítottam, hogy CR_{LGD} sokkal gyorsabban közelít a Shannon határhoz, mint a CR_{LZW} , míg a legkevésbé tárhely-intenzív bemeneti adat esetén bebizonyítottam, hogy $C_{LGD} \approx 2*C_{LZW}$.
- az enkódolási időbonyolultság n hosszúságú bemenetre mind az LZW, mind pedig az LGD esetén kifejezhető a $T_{LZW}(n) = T_r(n) + T_{comp}(n) + T_{de}(n) + T_{ins}(n) + T_{wr}(n)$ képlettel, azonban LGD esetén a T_r, T_{comp} és T_{de}

műveletek négyzetesen viszonyulnak az LZW azonos műveleteihez. Ez fejezi ki azt a tényt, hogy tárhelyet számítási művelettel helyettesítem.

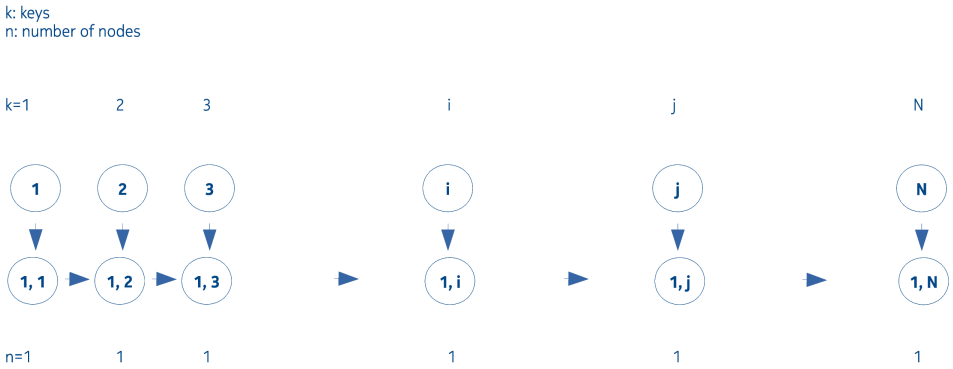


3. ábra: Figyelembe veendő függőségek az alkalmazni kívánt tömörítési módszerek kiválasztása során, és a paraméterezhető VDE-LGD módszer, ami VDE/LGD=0 esetén megfelel az LZW-nek.

IV.2. Téziscsoport – Adatstruktúrák és állományszervezés

IV.2.1. *Tézis – Interval Merging Binary Tree helyessége: Megterveztem egy újszerű adatszerkezetet, az Interval Merging Binary Tree-t (IMBT-t), amely a meglévő adatszerkezetekkel összehasonlítva hatékonyabban használható az egymáshoz közeli kulcsokkal azonosított adatok tárolására. Megmutattam, hogy a módszer a már meglévő elterjedt adatszerkezetekhez képest sokkal hatékonyabban használható a legkedvezőbb kulcsérkezési esetekben.*[S-4] (ICA3PP2017)

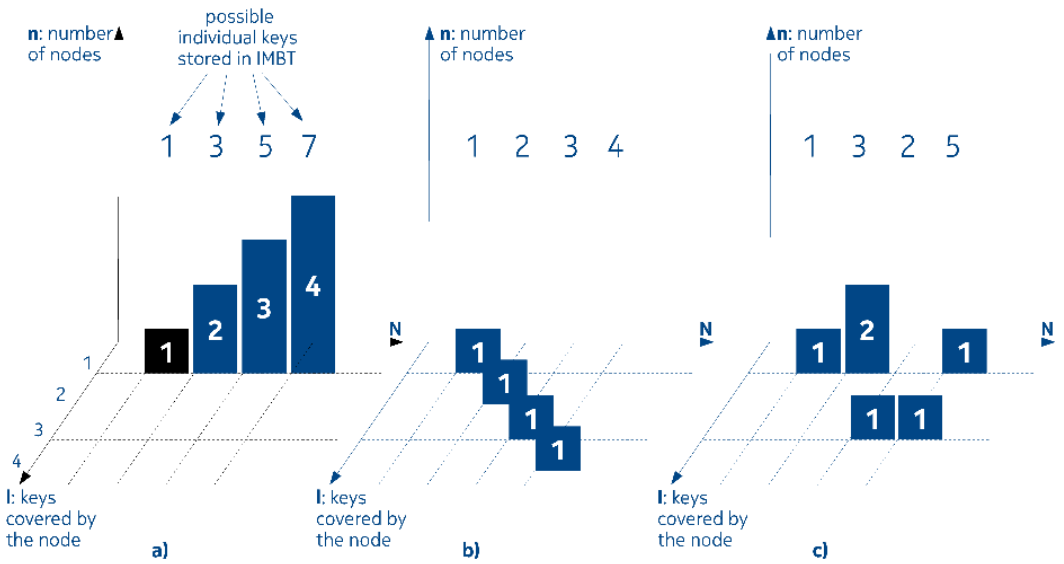
Az egyedi azonosítókkal ellátott adatok szűrése esetén a meglévő adatszerkezetek legtöbbje tárolja az azonsítót. Ez a közel valós idejű feldolgozásoknál még akkor is probléma, ha egyébként olyan O(1) idejű adatszerkezetről van szó, mint a hash-tábla. Ugyanis a gyorsaság egyik feltétele, hogy az adatszerkezet elférjen a memórában. Azonban olyan nagy mennyiségű adat során, mint ami pl. egy távközlési hálózat monitorozása közben keletkezik, a memória igen könnyen szűk keresztmetszetté válik. Ezért került kidolgozásra egy olyan fa szerkezetbe szervezett adatstruktúra ami ideális esetben állandó (ez látható az 4-es ábrán, forrás: [S-4]) egyéb esetben pedig nagyon lassan növekedő memóriaigénnyel és ennek megfelelő állandó, illetve lassan növekedő keresési időbonyolultsággal rendelkezik (ami a IV.2.4 Tézis szerint újból állandóvá tehető)



4. ábra: Interval Merging Binary Tree időfejlődése a kulcsok ideális beérkezése esetén: a memóriában egyetlen csomópont kerül csak lefoglalásra

IV.2.2. *Tézis – IMBT Állapottér: Felállítottam egy matematikai modellt, aminek segítségével felső becslést lehet adni az Interval Merging Binary Tree adatszerkezet keresési időbonyolultságainak a számosságára. Majd beláttam az időbonyolultságok számossága és a kulcsok beérkezési mintázat alapján történő eloszlás-osztályok közötti összefüggést.* [S-5](Acta2019)

A bizonyítás első felében megmutattam, hogy az *N* számelméleti partícióknak száma megfeleltethető az IMBT különböző elrendezéseinek számával. Ilyen naív realizációk láthatók az 5-ös ábrán (forrás: [S-5]). Majd bemutattam a csomópontok darabszáma szerint kiegyensúlyozott fát feltételezve, azon osztályok számát, amelyek a bejárás során különböző lépésszámhoz vezetnek. Végül a páros gráfok és a kontingencia-táblázatok[53-55][58] segítségével, a két eredmény kombinálásával adtam egy felső becslést a keresési időbonyolultságok lehetséges számára vonatkozóan.
Az eredmények rámutatnak arra a tényre, hogy az IMBT hatékonysága, a kiegyensúlyozáson túl, igen sokféleképpen függ a beérkező kulcsok eloszlásától, így előkészíti és megalapozza a statisztikai alapú kulcseloszlások vizsgálati módszereit.



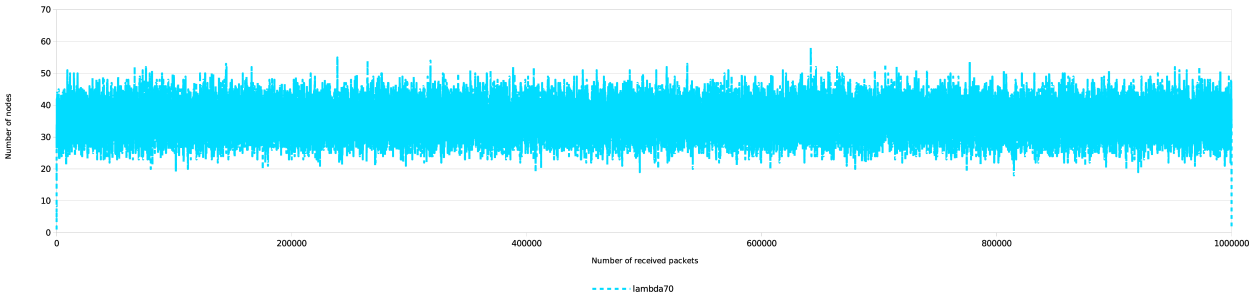
5.ábra: A beérkező kulcsok különböző szomszédsági viszonya által létrejövő intervallumok megfeleltethetők a számelmélei partíciókra bontásnak

IV.2.3. Tézis – IMBT Speciális kulcseloszlás: Kidolgoztam egy kontingencia-táblákon alapuló módszert, amelynek a segítségével meg lehet határozni az Interval Merging Binary Tree keresési műveleteinek az átlagos költségét a bemenő kulcsok jellemző mintázata alapján. Ennek alkalmazásával bemutattam, hogy milyen esetekben célszerű használni ezt az adatszerkezetet.
Levezettem és bebizonyítottam, hogy amennyiben az intervallumok átlagos a *hosszára igaz, hogy egyenletesen jellemzi a fa csomópontjait, akkor a megadott bejárás mellett az IMBT keresési teljesítménye egészen addig jobb, ameddig az*

$$a > \sqrt[3]{N}$$

egyenlőtlenség fennáll, ahol N az összes eddig beérkezett kulcsot jelenti. [S-6](Entropy2020)

Megmutattam, hogy a Bernstein-tétellel[60] jellemezhető feltételek teljesülése esetén, N (a kulcsok száma) növekedésével a fában lévő intervallumok a átlagos hossza egy állandósult várható érték körül ingadozik, darabszáma a fa magasságával együtt folyamatosan nő, így az IMBT-t csupán egy végtelen kontingencia-táblázattal lehet leírni.
Ellenkező esetben az átlagos hosszok tartanak a végtelenhez, kontingencia-táblázat véges és a fa magassága ingadozik egy átlagos várható érték körül, ami N-től független állandó keresési időt eredményez. Ez látható a 6-os ábrán (forrás: [S-6])

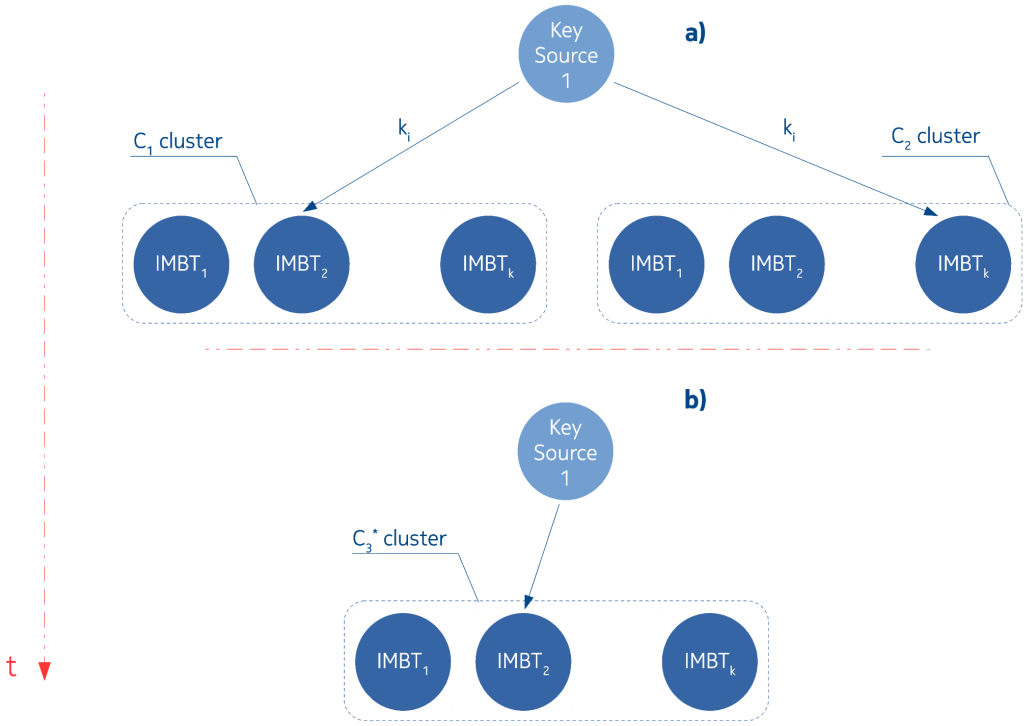


6. ábra: Bizonxos kulcseloszlások esetében az IMBT magassága egy várható érték körül ingadozik a beérkezett kulcsok számától függetlenül. A kék sáv szélessége a kulcsok keveredésétől függ. az ilyen kulcseloszlások stacionárius tárhely és keresési idő bonyolultságot eredményeznek.

IV.2.4. *Tézis - IMBT Mátrix Ábrázolás és Egyensúlyi Feltétel: A kiegyensúlyozott fára vonatkozó képletek alkalmas átalakításával bevezettem egy olyan jellemző mátrixot, aminek a segítségével tetszőleges kulcs/intervallum-eloszláshoz tartozó átlagos keresési idő mechanikusan felírhatóvá és kiszámíthatóvá válik. Ez jelentősen felgyorsítja a szimulációkat. Egy ilyen szimuláció eredményeként sikerült belátni a geomatriai eloszlás alapjának kulcsszámra vontakoztatott függőségét, mint a fa stacionáriusan állandó 'n' számú csomópontjára jellemző egyensúlyi feltétel.* [S-9](CSCE_Las_Vegas2021)

$$b_e(N) = x^{\frac{\log_x(N)}{n}}$$

IV.2.5. *Tézis – IMBT Elosztott Környezetben: Kidolgoztam az IMBT egy olyan párhuzamos környezetre történő alkalmazását, ahol az egyes számítási csomópontok közötti szinkronizácó a többi adatszerekezethez képest lényegesen hatékonyabban valósítható meg, az IMBT bufferelő hatása miatt.*
Bizonyítottam, hogy megfelelő skálázó eljárás segítségével, a fák a bennük lévő csúcsok számától függő úgynevezett "immutable" csoportokba szervezésével, a keresési idő állandósága biztosítható a rendszerben lévő kulcsok számától függetlenül. [S-7](BDS_Oxford2020)



7. ábra: A párhuzamosan végezhető lekérdezések immutable klaszterek esetében és felülről korlátos mutable C_3^*

A 7-es ábrán (forrás: [S-7]) látható C_i klaszterek nem változnak, ezért párhuzamosan lekérdezhetők. Mivel a klasztereket alkotó fák magassága ennél a megoldásnál előre meghatározott, így ez klaszterenként ugyanazt a log(h)+1 maximális keresési időt jelenti. Viszont a magassági korlát a még éppen módosítás alatt lévő C^*ra is igaz. Így ott is igaz, hogy a keresési idő <= log(h)+1, azaz a maximális keresési idő biztosan kisebb, mint 2*(log(h)+1).
Ennél az elrendezésnél a párhuzamosan kiküldött üzenetek száma válhat idővel szűk keresztmetszetté (amennyiben a kulcseloszlás nem teszi lehetővé, hogy a fák magassága egy maximális várható érték körül ingadozzanak). A problémára már kidolgozásra került olyan megoldás ami a gyakorlatban is alkalmazható: kellően általános kulcseloszlás esetén is biztosítja az üzenetek számának korlátosságát, állandó keresési idő mellett, de még nem publikált.

V. Az eredmények hasznosítási lehetősége

Mind a bemutatott tömörítési módszer, illetve mind az elosztott környezetben alkalmazható intervallum-fa egy-egy az iparban fellépő igény hatására került kidolgozásra, így a gyakorlati hasznosítási lehetőség nem igényel különösebb indokálst.
Ezen túlmenően a tömörítési módszer esetében a legrosszabb eset vizsgálata olyan matematikai eljárások területére vezet (combinatorics on words, színezett exponenciális fák), ahonnan várható sokkal absztraktabb, általánosabb eredmény is az ismétlődésmentes sztringekkel kapcsolatban.
Az intervallum-fán elvégzett és eddig bemutatásra még nem került vizsgálatok alapján van okom feltételezni, hogy elosztott környezetben a fák egy olyan elrendezését/topológiáját is létre tudom hozni, ahol a lekérdezéshez használt üzenetek száma is állandó a fa magassága mellett, ami az adatmennyiségtől független, maximált válaszidő lehetőségét hordozza magában. Ennek a gyakorlati jelentősége igen nagy.

VI. Irodalmi hivatkozások listája

[1] Amazon AWS, https://aws.amazon.com/, last visited 2021-02-20

[2] Dean, J.; Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters, http://static.googleusercontent.com/media/research.google.com/es/us/archive/mapreduce-osdi04.pdf, last visited 2021-02-20

[3] GFS Architecture, https://static.googleusercontent.com/media/research.google.com/en//archive/gfs-sosp2003.pdf, last visited 2021-02-20

[4] HDFS Architecture, https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html, last visited 2021-02-20

[5] MAPR (now part of HP) https://www.hpe.com/us/en/software/data-fabric.html, last visited 2021-02-20

[6] STORM - A distributed realtime computation system, http://storm.apache.org/documentation/Home.html , last visited 2021-02-20

[7] ZeroMQ messaging system https://zeromq.org/, last visited: 2021-02-20

[8] Netty messaging system https://netty.io/, last visited: 2021-02-20

[9] RabbitMQ messaging https://www.rabbitmq.com/, last visited: 2021-02-20

[10] Kafka messaging https://kafka.apache.org/, last visited: 2021-02-20

[11] Millwheel https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/41378.pdf , last visited 2021-02-20

[12] Samza https://samza.apache.org/, last visited 2021-02-20

[13] Spark https://spark.apache.org/, last visited 2021-02-20

[14] HBase https://hbase.apache.org/, last visited 2021-02-20

[15] Hive https://hive.apache.org/, last visited 2021-02-20

[16] Cassandra https://cassandra.apache.org/, last visited 2021-02-20

[17] CouchDB https://couchdb.apache.org/, last visited 2021-02-20

[18] VoltDB https://www.voltdb.com/, last visited 2021-02-20

[19] Zookeeper https://zookeeper.apache.org/, last visited 2021-02-20

[20] Ganglia monitoring system http://ganglia.sourceforge.net/, last visited: 2021-02-20

[21] Lambda architecture http://lambda-architecture.net/, last visited: 2021-02-20

[22] Java Collections Frameworks, http://docs.oracle.com/javase/7/docs/technotes/guides/collections/overview.html , last visited 2021-02-20

[23] Knuth, Donald (1997). "6.2.2: Binary Tree Searching". The Art of Computer Programming. 3: "Sorting and Searching" (3rd ed.). Addison-Wesley. pp. 426–458. ISBN 0-201-89685-0.

[24] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.: Introduction to Algorithms (3rd ed.). MIT Press and McGraw-Hill, (2009). ISBN:0-262-03384-4

[25] Adelson-Velsky, G., Landis, E.: Organization and maintenance of large ordered indexes, Acta Informatica, Volume 1, Issue 3, pp. 173-189, (1972). DOI:10.1007/BF00288683

[26] Adelson-Velsky, G., Landis, E.: An algorithm for the organization of information, Proceedings of the USSR Academy of Sciences, Volume 146 , pp. 263-266 (1962).

[27] AVL Tree - Wiki, https://en.wikipedia.org/wiki/AVL-tree , last visited 2021-02-20

[28] Sedgewick, R.: Algorithms 1st edition, Addison-Wesley 1983, ISBN 0-201-06672-6.

[29] Bayer, R.: Symmetric binary B-Trees: Data structure and maintenance algorithms, Acta Informatica, Volume 1, Issue 4, pp. 290-306, (1972). DOI:10.1007/BF00289509

[30] B-Tree - Wiki, https://en.wikipedia.org/wiki/B-tree, last visited 2021-02-20

[31] Paul E. Black, "(a,b)-tree", in Dictionary of Algorithms and Data Structures [online], Paul E. Black, ed. 6 October 2004. (accessed 2021-03-22) Available from: https://www.nist.gov/dads/HTML/abtree.html

[32] de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O.: Interval Trees, Computational Geometry, Second Revised Edition. Springer-Verlag, Section 10.1, pp. 212-217 (2000).

[33] Interval Tree https://en.wikipedia.org/wiki/Interval_tree, last visited 2021-02-20

[34] Bentley, J. L., Ottmann, T. A.: Algorithms for reporting and counting geometric intersections, IEEE Transactions on Computers, C28 (9), pp. 643-647, (1979). DOI:10.1109/TC.1979.1675432

[35] Pfaff, B.: Performance analysis of BSTs in system software, ACM SIGMETRICS '04, Volume 32 Issue 1, pp. 410-422, (2004). ISBN:1-58113-873-3

[36] Bloom, B. H.: Space/time trade-offs in hash coding with allowable errors, Communications of the ACM, Volume 13 Issue 7, pp 422-426, New York, NY, USA, July (1970).

[37] Tom White: Hadoop: The definitive Guide, 2012 O'REILLY, ISBN: 978-1-449-31152-0-1327616795

[38] Jiang Liu; Bing Li; Meina Song: THE optimization of HDFS based on small files, Broadband Network and Multimedia Technology (IC-BNMT), 2010 3rd IEEE International Conference on, pp 913 - 915, 2010

[39] Zhang, Yang; Liu, Dan: Improving the Efficiency of Storing for Small Files in HDFS, Computer Science & Service System (CSSS), 2012 International Conference on, pp 2239 - 2242, 2012

[40] History of Lossless Data Compression Algorithms, http://ethw.org/History_of_Lossless_Data_Compression_Algorithms, last visited 2021-02-20

[41] David Salomon, Giovanni Motta: Handbook of Data Compression, 5th edition London, England: Springer-Verlag, 2010, pp. 377-378. David Salomon, Giovanni Motta: Handbook of Data Compression, 5th edition London, England: Springer-Verlag, 2010, pp. 378-379.

[42] Triangular Number, http://mathworld.wolfram.com/TriangularNumber.html, last visited 2021-02-20

[43] A. Lempel, J. Ziv: On the Complexity of Finite Sequences, IEEE Transactions on Information Theory, Volume 22 Issue 1, pp. 75 - 81, Jan 1976

[44] Jacob Ziv: A constrained-dictionary version of LZ78 asymptotically achieves the finite-state compressibility with a distortion measure, IEEE Information Theory Workshop (ITW), pp. 1-4, 2015, Jerusalem, Israel

[45] Terry Welch: A Technique for High-Performance Data Compression, IEEE Computer Society Journal Volume 17 Issue 6, pp 8 - 19, June 1984

[46] P.C. Shields: Performance of LZ algorithms on individual sequences, IEEE Transactions on Information Theory, Volume 45 Issue 4, pp. 1283-1288, May 1999

[47] Nishad PM, Dr. R. Manicka Chezian: Behavioral Study of Data Structures on Lempel Ziv Welch (LZW) Data Compression Algorithm and ITS Computational Complexity, Intelligent Computing Applications (ICICA), 2014 International Conference on, pp. 268-274, March 2014

[48] Google Brotli https://en.wikipedia.org/wiki/Brotli, last visited 2021-02-20

[49] Google Snappy https://google.github.io/snappy/, last visited 2021-02-20

[50] Facebook Zstandard https://engineering.fb.com/2016/08/31/core-data/smaller-and-faster-data-compression-with-zstandard/, last visited 2021-02-20

[51] Facebook Zstandard https://en.wikipedia.org/wiki/Zstandard, last visited 2021-02-20

[52] Hardy, G.H., Ramanujan, S.: Asymptotic Formulae in Combinatory Analysis, Proceedings of the London Mathematical Society, 1918

[53] Bóna, M.: A Walk Through Combinatorics: An Introduction to Enumeration and Graph Theory. pp. 145-164, World Scientific Publishing, 2002 ISBN 981-02-4900-4.

[54] Cayley, A.: A Theorem on Trees. Quarterly Journal of Pure and Applied Mathematics 23, pp. 376-378, 1889

[55] Barvionk, A.: Enumerating Contingency Tables via Random Permanents, Combinatorics, Probability and Computing, Volume 17, pp. 1-19, 2008 DOI:10.1017/S0963548307008668

[56] Barvinok, A., Luria, A., Samorodnitsky, A., Yong, A.: An approximation algorithm for counting contingency tables, Random Structures Algorithms 37 (2010), no. 1, pp. 25-66, 2010 DOI:10.1002/rsa.20301 arXiv:0803.3948

[57] Stoica, I.; Morris, R.; Liben-Nowell, D.; Karger, D. R.; Kaashoek, M. F.; Dabek, F.; Balakrishnan, H.: Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. IEEE/ACM Trans. Netw. 2003, 11(1), 17–32. DOI:10.1145/964723.383071

[58] Lauritzen, S.L. Lectures on Contingency Tables, 2002, Electronic edition, Aalborg University. Available online:http://www.stats.ox.ac.uk/~steffen/papers/cont.pdf, (accessed on 28 November 2019)

[59] Meyn, S.P. Tweedie, R.L. Markov Chains and Stochastic Stability. Springer: London, UK, 2012. ISBN9781447132677

[60] Bernstein, S.N. Theory of Probabilities. Moskva, Leningrad, 1946.

VII. A tézispontokhoz kapcsolódó tudományos közlemények

[S-1] Finta, I.; Farkas, L.; Sergyán, Sz.; Szénási, S.: Buffering Strategies in HDFS Environment with STORM framework, IEEE 16th International Symposium on Computational Intelligence and Informatics, 19–21 November, 2015, Budapest, Hungary

[S-2] Finta, I.; Farkas, L.; Sergyán, Sz.; Szénási, S.: A Method for Virtual Extension of LZW Compression Dictionary, Innovations in Clouds, Internet and Networks (ICIN), 19th IEEE International Conference on, pp 184 - 188, 2016, Paris

[S-3] Finta, I.; Farkas, L.; Sergyán, Sz.; Szénási, S.: Transient analysis of virtual dictionary extension compression method, IEEE 17th International Symposium on Computational Intelligence and Informatics (CINTI), pp. 67-74, 17-19 Nov. 2016, Budapest, Hungary DOI: 10.1109/CINTI.2016.7846381

[S-4] Finta, I.; Farkas, L.; Sergyán, Sz.; Szénási, S.: Interval Merging Binary Tree, ICA3PP 2017, Helsinki, Finland, August 21-23, 2017 DOI:10.1007/978-3-319-65482-9

[S-5] Finta, I.; Szénási, S.: State-space Analysis of the Interval Merging Binary Tree. Acta Polytech. Hung. 2019 16(5), pp 71–85. DOI: 10.12700/APH.16.5.2019.5.5

[S-6] Finta, I.; Szénási, S.; Farkas, L.: Input Pattern Classification Based on the Markov Property of the IMBT with Related Equations and Contingency Tables. Entropy 2020, 22, 245. https://doi.org/10.3390/e22020245

[S-7] Finta, I.; Szénási, S.; Farkas, L.: Data Structure for Packet De-Duplication in Distributed Environments, 2020 IEEE Sixth International Conference on Big Data Computing Service and Applications, Oxford, United Kingdom, ISBN: 978-1-7281-7022-0

[S-8] Finta, I.; Szénási, S.; Farkas, L.: Quantity Analysis on the Chaining of Repetition-free Words Considering the VDE Composition Rule, IEEE 15th International Symposium on Applied Computational Intelligence and Informatics, SACI2021

[S-9] Finta, I.; Szénási, S.; Farkas, L.: Quantifying the Performance of Interval Merging Binary Trees using a Matrix Representation, The 2021 World Congress in Computer Science, Computer Engineering, & Applied Computing - CSCE ’21, Las Vegas, USA, 2021

VIII. További tudományos közlemények

[S-10] Horvath, I.; Finta, I.; Kovacs, F.; Meszaros, M.; Molontay, R.; and Varga, K.: Markovian queue with garbage collection. In: Thomas N., Forshaw M. (eds) Analytical and Stochastic Modelling Techniques and Applications. ASMTA 2017. Lecture Notes in Computer Science, vol 10378. Springer, Cham. https://doi.org/10.1007/978-3-319-61428-1_8

[S-11] Finta, I.; Élias, G.; Illés, J.: Packet Loss and Duplication Handling in Stream Processing Environment. In Proceedings of the CINTI 2018, Budapest, Hungary, 21–22 November 2018. DOI:10.1007/978-3-319-65482-9