Napredni algoritmi i strukture podataka

Kompresija podataka – osnove, Ograničenje stope pristupa (Rate Limiting)

Token Bucket, TTL

Univerzitet u Novom Sadu
Fakultet Tehničkih Nauka

Osnove

- Nećemo se previše baviti mogućnostima kompresije, pošto je to oblast sama za sebe i prebazilazi granice kursa
- **ALI** neke osnove nije loše čiti i videti kako to možemo iskoristiti u sistemu
- U teoriji informacija, kompresija podataka je proces kodiranja informacija koristecći manje bitova od originalnog prikaza
- Ovaj proces se može podeliti na dve velike grupe:
  1. Kompresija sa gubicima (lossy compression) ili nepovraatna kompresija
  2. Kompresija bez gubitaka (lossless compression)
- Još jedan termin koji je bitan je pojam negative kompresije – proces u kome podaci nakon kompresije zauzimaju više prostora nego početni podaci

Kompresija sa gubicima

- Ovo je klasa a;goritama za kompresiju podataka koja koristi aproksimacije, i delimično odbacivanje podataka za predstavljanje sadržaja
- Ove tehnike se koriste za smanjenje veličine podataka za skladištenje, rukovanje i prenos sadržaja
- Količina smanjenja podatakakorišććenjem kompresije sa gubicima je **mnogo vecća** nego korišćenjem tehnika bez gubitaka
- Dobro dizajniran algoritam kompresije sa gubicima često značajno smanjuje veličinu datoteka pre nego što krajnji korisnik primeti degradaciju

Kompresija bez gubitaka

- Kompresija bez gubitaka je mogucća jer vecćina podataka iz stvarnog sveta pokazuje statističku redundantnost
- Kompresija bez gubitaka se koristi u slučajevima kada je važno da originalni i dekomprimovani podaci budu identični
- **ILI** kada bi odstupanja od originalnih podataka bilo nepovoljno po sam proces
- Uobičajeni primeri su izvršni programi, tekstualni dokumenti i izvorni kod

Negativna kompresija

- Algoritmi za kompresiju, neretko koriste dodatne strukture da sačuvaju elemente kompresije – deskriptor (meta podaci)
- Ovo može biti potencijalno problematično u odredjenim situacijama
- Pošto se proces kompresije oslanja na pronalaženje šablona u originalim podacima, ako njih nema, može doći do problema
- Tipičan primer je slika, koju računarski posmatramo kao matricu gde svakaćelija ima obično 3 kanala (crveni, zeleni, plavi):
  - Ako u jednom redu (ili koloni) imamo odredjen broj piksela obojenih istom bojom, možemo ih zameniti vrednosti obojenosti piksela i brojem ponavljanja – čuvamo u deskriptoru *Run-Length Encoding*
  - **ALI** ako nemamo ovu situaciju, većina piksela su obojeni različito, onda naš deskriptor potencijalno može zauzeti dosta mesta čime imamo veći podataka od inicijalnog – negativna kompresija

Varijabilno i fiksno kodiranje

- Prva zgodna tehnika koju možemo iskoristiti da smanjimo prostor je svakako izbor kodiranja zapisa:
  - U kodiranju fiksne dužine, sva slova/simboli su predstavljeni istim brojem bitova:
    - Kodiranja fiksne dužine ima prednost kod nasumičnog pristupa
    - Npr. kod teksta, svako slovo sadrži jednak broj bitova, npr. skok na 4. slovo, preskačemo odgovarajucću količinu bitova
  - Kod kodiranja varijabilne dužine možemo da iskoristimo mogućnost da svi podaci ne'će biti predstavljeni istim brojem bitova:
    - Medjutim, u vecćini slučajeva stvarna vrednost neće zauzti punu dužinu
    - U slučaju malih vrednosti, neki zavr'v sni bitovi cé biti ostavljeni prazni, – na kraju imamo veliki broj beskorisnih zavr'v snih nula da bi veličina bila npr. 32 bita.

Δ kodiranje

- Δ kodiranje se odnosi na nekoliko tehnika koje čuvaju podatke kao razliku izmedju uzastopnih uzorke (ili karaktere)
- Ovom tehnikom nema potrebe za direktno skladištenje samih uzoraka
- Δ kodiranje se koristi kada susedne vrednosti u originalnim podacima imaju malu promenu izmedju njih
- U nekim specifičnim problemima, ovu ideju možemo pogirati i malo dalje
  - Delta-delta kodiranje primenjuje delta-kodiranje drugi put na delta-kodirane podatke
  - Sa skupovima podataka vremenskih serija u kojima se prikupljanje podataka dešava u redovnim intervalima, možemo primeniti ovaj mehanizam na vremensku kolonu, efektivno treba da sa'v cuvamo samo niz 0.
  - **This compresses a full timestamp (8 bytes = 64 bits) down to just a single bit (64x compression). (In practice we can do even better...**

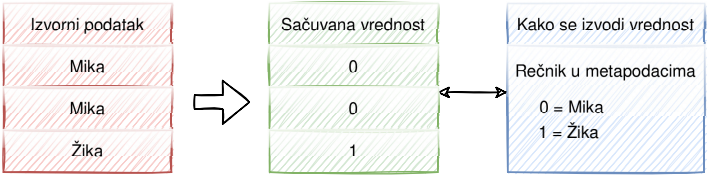
    (https://www.timescale.com/blog/time-series-compression-algorithms-explained/)

Inkrementalno kodiranje

- Tip algoritma za kompresiju Δ kodiranja gde se uobi'v cajeni prefiksi ili sufiksi i njihove dužine beleže tako da ne moraju da se dupliraju
- Vrednost definišemo samo jednom, ponovljenje vrednosti možemo izvesti iz prethodno definisanih, – čuvamo deltu od početne sačuvane vrednosti – posebno pogodan za komprimovanje sortiranih podataka

Kodiranje rečnika sa pakovanjem bitova

- Ako u nskupu podataka, odredjeni podatak, ili grupa se ponavljaju možemo da iskoristimo ovaj algoritam (Dictionary Encoding with Bit-Packing)
- Svaku vrednost u skupu, zamenjuje malim celim brojem i čuva mapiranje u metapodacima – deskriptor, zauzimanje što manje mesta sa podacima.
- ALI kada želimo da pročitamo komprinovane podatke podatke, moramo da konsultujemo deskriptor

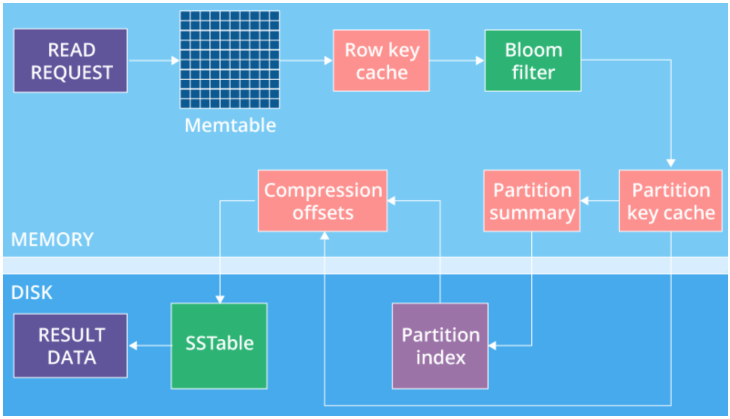


Run-Length Encoding

- Ovaj algoritam može biti koristan ako unapred znamo da imamo puno ponavljanja istog elementa
- Ako imamo 100.000 zapisa istog broja, onda imamo dve mogućnosti:
  - sačuvati sve podatke – besmisleno, bacamo resurse eto tako
  - sačuvati vrednost, i broj ponavljanja i poziciju u deskriptor – pametno oslobadjamo dosta mesta za nove zapise
- Ovim smo dobili dosta manje zauzeća prostora
- **ALI** za ovu tehniku nam je potrebno da su podaci sortirani
- može biti zanimljivo kod aplikacija koje koriste npr. vremenske serije

Proširenje sistema

- Naše SSTabele sada dodatno treba da čuvaju i deskriptor kompresije
- Put čitanja sada mora uzeti u razmatranje deskriptor, da bi pročitana vrednost imala smisla kada se vraća korisniku
- Čuvamo više podataka na istom prostoru



Napomene

- Ovo su samo neke od tehnika kompresije podataka koje postoje, a dosta se koriste u ovakvim sistemima
- Rad sa slikama, video sadržajem isl. imaju drugačije mehanizme kompresije
- **Izbor algoritma kompresije nije nasumičan**
- **Obično** se unapred zna, kakve podatke će korisnik čuvati, stoga, može se izabrati i povoljan algoritam kompresije

Zaštite do sada

- Tokom ovog kursa, radili smo nekoliko različitih tenhnika i algoritama za zaštitu (uglavnom) podataka, ali i konkretnih stuktura
- Videli smo da možemo da štitimo memorijsku strukturu (Memtable) da bi dobili trajnost podataka – koristili smo **Write Ahead Log (WAL)** kao čitavu strukturu sa svojim setom algoritama
- Videli smo da moramo štiti podatke na disku, ne samo u **WAL-u**, već i u **SSTable-u**, i za to smo koristili **CRC** mehanizam
- Da bi ustanovili da li je možda došlo do problema nakon zapisivanja podataka
- Videli smo da prilikom učitavanja podatka, prvo trebamo da konsultujemo CRC da bi bili sigurni da su podaci koje čitamo ispravni

- Nakon toga smo videli kako možemo da razmenimo podatke sa drugim učesnicima, i za te potrebe koristili smo **Merkle** stabla
- Videli smo kako možemo da ustanovimo da li je neki podatak, deo većeg skupa podataka
- Videli smo kako močemo u situacijama kada su podaci na više čvorova da ustanovimo gde su problemi, i da kroz mrežu šaljemo jako malo podataka za te provere
- Zatim smo videli, kako možemo samo da pošaljemo deo podataka koji je problematičan

Još jedna zaštita...

- Sve ove strukture i algoritme smo koristili da bi zaštili neki deo sistema
- Uglavnom podatke, prilikom čitanja, pisanja, razmene
- Ovih tehnika ima još, ali ovde ćemo stati
- **ALI** treba da štitimo i sistem od prevelike količine zahteva po jedinici vremena

Pitanje 1

Ali kako da štitimo celokupan sistem od prevelike količine zahteva u jedinici vremena...?

Ideje :) ?

Pitanje 2

Žasto bi to radili uopšte...?

Ideje :) ?

Ograničenje stope pristupa – uvod

- U računarskim mrežama, ograničenje brzine/stope se koristi za kontrolu brzine/stope zahteva poslatih ili primljenih od strane kontrolera mrežnog interfejsa
- Ograničavanje stope/brzine pristupa (Rate Limiting) je procedura koja nam omogućava kontrolu brzine kojom korisnici mogu da šalju zahteve sistemu
- **Rate Limiting** se uglavnom koristi za zaštitu servera od neželjenih rafala, zlonamernih napada
- Zaštita sistema od prekomerne upotrebe ograničavanjem koliko često korisnici mogu da im pristupe, ima nekoliko prednosti
- Pomaže protiv napada *denial-of-service*, pokušaja prijave *brute-force* i drugih vrsta nasilnog ponašanja korisnika

- Može i da se koristi kod različitih servisa da se vidi da li imamo dovoljno finansija da pristupimo nekakvom resursu
- Web servisi mogu da ga korite kada pružaju usluge korisnicima da bi odbili zahteve, ako su prekoračili limit
- Postoji razni tipovi Rate Limiting-a npr:
  - **Rate limiter korisnika** omogucćava nekim grupama korisnika ograničen pristup sistemu – broj/trajanje zahteva korisnika obično je vezan za njihove ključeve ili IP adrese
  - **Rate limiter istovremene/serverske brzine** prati koliko je paralelnih sesija ili veza dozvoljeno za nekim grupama korisnika – ublažava DDoS napade
  - **Rate limiter lokacije** ograničava brzine/stope pristupa za neke regione, kao i za definisani vremenski period – moguće je definisati različite stepene pristupa za razne lokacije

Primeri algoritama

- Neki od primera algoritama za rešavanje ovog problema:
  - **Token Bucket** – radimo danas
  - Leaky Bucket
  - Fixed Window Counter
  - Sliding Logs
  - Sliding Window Counter

Rate limiter kod sistema za skladištenje podataka

- Neki sistemi za skladištenje velikie količine podatak su implementirali ovaj mehanizam
- To ih nekada izdvaja od drugih sličnih sistema i zato su čest izbor korisnika
- **RocksDB**, na primer, direktno podržava ovaj mehanizam, i zato je ponekad češći izbor od recimo **LevelDB**
- Ali pored problema koje ova grupa algoritama rešava, za ovaj tip sistema vezuje se još zanimljivih upotreba

Pitanje 3

...za ovaj tip sistema vezuje se još zanimljivih upotreba...?

Ideje :) ?

- Kod upotrebe ovih sistema, korisnici možda žele da priguše maksimalnu brzinu pisanja u okviru nekog ograničenja iz mnogo razloga
- Na primer, brzi zapisi izazivaju strašne skokove u kašnjenju čitanja ako prekorače definisani prag
- RocksDB ima mogućnoti da korisnici mogu da podese Rate limiter kako njima odgovara
- Pruža cak i mogućnost dinamičkog ograničenja – **Auto-tuned Rate Limiter**
  (RocksDB Docs, http://rocksdb.org/blog/2017/12/18/17-auto-tuned-rate-limiter.html)

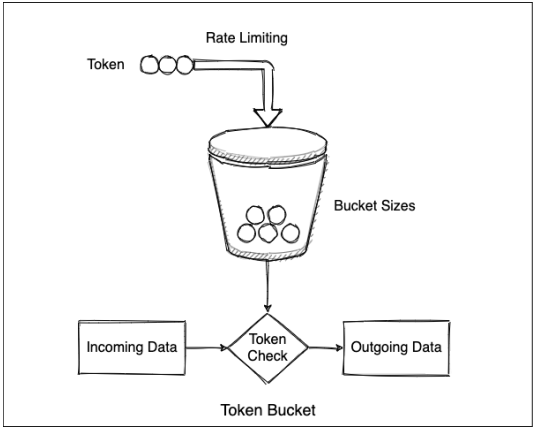
Pitanje 4

Gde bi mogli da čuvamo podešenja vezano za limiter, npr. koliko zahteva po jedinici vremena (sekund, minut, sat, ...)

Ideje :) ?

Tocken Bucket – uvod

- Ovo je najjednostavniji algoritam za ograničavanje brzine pristupa
- Jednostavno pratimo broj zahteva napravljenih u zadatom vremenskom intervalu
- Zbog svoje jednostavnosti, dosta se koristi
  - Google cloud koristi ovaj algoritam (ili je koristio), za Task Queue opciju koja se nudi koirsnicima kao usluga
- Lako može da se poveže sa velikim brojem različitih slučajeva korišćenja



(What is Token Bucket and Leaky Bucket algorithms)

Tocken Bucket – algoritam

- Za svaki zahtev korisnika treba:
  - Proveriti da li je vreme proteklo od poslednjeg resetovanja brojača vremena
  - Ako vreme nije isteklo, treba proveriti da li korisnik ima dovoljno preostalih zahteva da obradi dolazni zahtev
  - Ako korisniku nije preostalo slobodnih zahteva, trenutni zahtev se odbacuje uz nekakvu poruku
  - U suprotnom, smanjujemo brojač za **1**, i vršimo obradu dolaznog zahteva
  - Ako je vreme proteklo, tj. razlika resetovanog vremena i trenutnog vremena je veća od definisanog intervala, resetujemo broj dozvoljenih zahteva na unapred definisano ograničenje, i definišemo novo vreme resetovanja

Tocken Bucket – primer

Pimer: 3/min:

- REQ **11:01:20** – > BUCKET [11:01:05, 3] => OK
- REQ **11:01:25** – > BUCKET [11:01:05, 2] => OK
- REQ **11:01:30** – > BUCKET [11:01:05, 1] => OK
- REQ **11:01:35** – > BUCKET [11:01:05, 0] => FAIL
- REQ **11:03:00** – > BUCKET [11:03:00, 2] => OK // uradimo update vremena, broja tokena, pustimo zahtev i smanjimo broj tokena za **1**
- ...

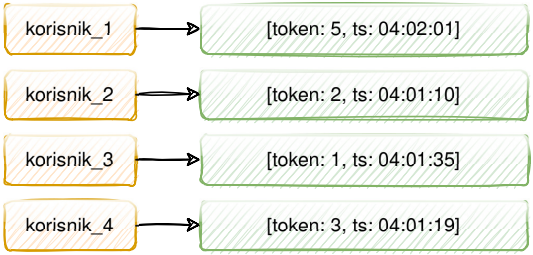
Pitanje 5

Gde čuvati ove informacije...?

Ideje :) ?

Čuvanje informacija

- Za aplikativne stvari, obično u memoriji ili nekoj sistemu koji čuva podatke u memoriji zbog brzine
- Pošto mi pravimo sistemsku stvar, i pravimo storage engine – pa močemo čuvati u našem sistmu :)
- Svaki korisnik može da bude **ključ**, dok **vrednost** može da sadrži vremensku odrednicu i broj tokena



Pitanje 6

Možemo li nekako da automatizujemo proces reseta tokena...? Na kraju krajeva nema potrebe da korisnički zahtev trigeruje i proverava (potencijalno) usporavamo zaheteve...

Ideje :) ?

Time-to-live – TTL

- Ovaj pojam u različitim primenama ima drugačiji kontekst
- Time-to-live (TTL) je definisani vremenski period tokom kojeg paket ili podaci treba da postoje na računaru, bazi, mreži, itd. pre nego što budu odbačeni ili ukljonjeni
  - Sve ovo radimo sa ciljem, da smanjimo čitanje sadržaja sa diska
  - ALI da opet sa druge strane ne opteretimo sistem previše, **ILI** da ne zavisimo od korisnika
- Ova tehnika se dosta koristi kod recimo keširanja sadržaja
- Ovo nam omogućava da ne pravimo dodatne strukture
  - ALI zahteva da imamo pozadinski brojač da li je vreme isteklo
- U zavisnosti od primene, i tipa aplikacije, podataka, načina pristupa možmo da izaberemo neku od tehnika

Sistemi za skladištenje podatka i TTL

- Dosta sistema omogućava korisnicima da naprave poseban tip koji će se skladištiti – TTL
- Danas, manje više svaki Key-Value store omogućava ovaj tip
- Korisnici treba da obezbede **ključ**, **ttl** odnosno koliko dugo (u nekoj jedinici vremena) podataka biti aktivan, i sam podatak
- Ove tipove obično ne možemo menjati
- AKO TTL istekne, mi kao korisnici moramo da napravimo nov zapis – obično, ali nije striktno pravilo

Pitanje 7

AKO TTL istekne, mi kao korisnici moramo da napravimo nov zapis...kako da sistem zna da je TTL istekao...?

Ideje :) ?

- Kada se podaci dodaju u sistem, oni se svakako mogu zapisati u Memtable, pa i u SSTable kada dodje do kompakcije
- Sistem može da održava posebnu strukturu sa ključevima, podacima i vremenom
- Za svaki podatak može da se pokrene poseban sat u pozadini koji odbrojava
- Kada vreme istekne, podatak biva obrisan
- Ova ideja se može iskoristiti recimo za token bucket
- Kada vreme istekne signalizirati da se tokeni resetuju
- **Ovaj mehanisam nije obavezan za implementaciju u vašem projektu**

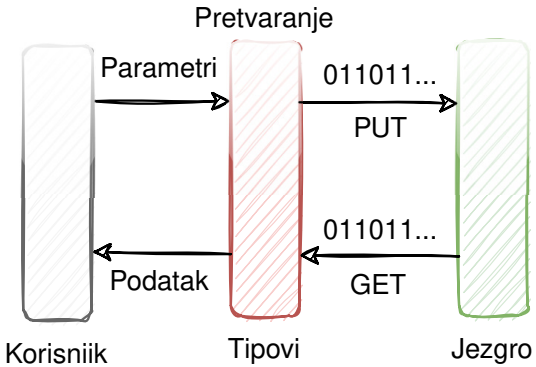
Pitanje 8

Dosta sistema omogućava korisnicima da naprave poseban TTL tip koji će se skladištiti.
Da li možemo dodati još neke tipove sa kojima smo radili na ovom predmetu..

Ideje :) ?

Tipovi

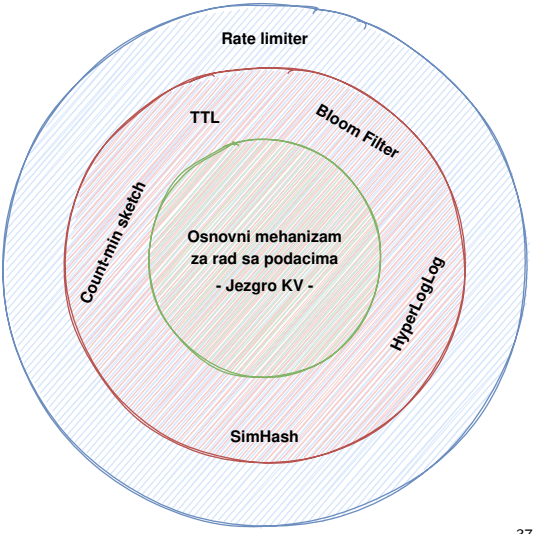
- Pored TTL-a, razni sistemi omogućavaju dodatne tipove podataka
  - Ovi tipovi podataka mogu biti specifični za neke primene, recimo **streaming**
  - Kao na primer sistmei kao što su **Redis** i **Riak** omogućavaju
    - **HyperLogLog**, **Bloom Filter**, **Count-min sketch** itd.
- Ako pogledamo, ovo nije ništa čudno
  - Do sada smo pravili te struture, uglavnom, u memoriji – ako sistem padne ?
  - Bilo bi lepo sačuvati na stibilan medijum te informacije



- Ako pogledamo našu struturu, vidmo da mi čuvamo **ključ kao string, a vrednost kao niz bajtova** – opšti oblik
- Ovo nam daje mogućnost da u taj niz bajtova smetimo šta god hoćemo :D
- To znači da recimo **HyperLogLog**, **Bloom Filter**, **Count-min sketch**, **SimHash** možemo da serijalizujemo u niz bajtova i da dodamo pod nekim ključem
- **ALI** klasičan **PUT** neće raditi posao, pošto on očekuje 2 stvari ključ i vrednost
- A nama će trebati **bar** još jedna vrednost za ispravan rad
  - Za te, specifične tipove podataka, možemo da napravimo i specifične funkcije
  - Na taj način smo relativno jednostavno obezbedili da korisnicima pružimo neke napredne funkcije
  - Nismo narušili model, nismo ugrozili sistem, korisnisci su srećni i zadovoljni, a i vi ste, pošto izmene nisu tako grandiozne
- **Think Twice Code Once!** :)

Slojevi

- Kada imate ispravno jezgro dalje možete šta god hoćete
- Ako dalje želite da proširite vaš sistem, možete napraviti novi sloj na odgovarajućem nivao!
- Uraditi ispravnu podelu nadležnosti, da svaki sloj radi SAMO jednu stvar i da je radi dobro!!
- "do one thing and do it well" – *Unix philosophy*



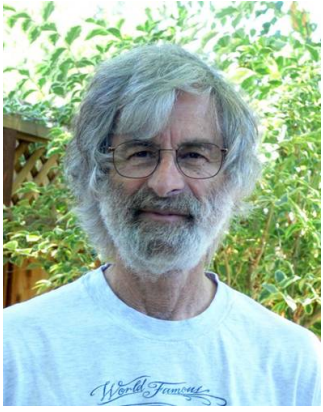
Šta dalje?

- Na ovom predmetu samo smo zagrebali jedno veliku i interesantnu oblast koja je povezana sa dosta drugih :)
- **AKO** nekoga bude zanimalo možda više da se uključi u oblast, par stvari koje možete kasnije dodati:
  - Pozadinske aktivnosti (niti, procesi) **izazovno i obavezno**
  - (Ozbiljnija) Kompresiju podataka
  - Optimizacija parametara veličine/broja SSTable-a
  - Optimizacija parametara za cache
  - Optimizacija parametara za rate limiting
  - Pristup preko interneta
  - Distribuirati sadržaj i mogućnost rada na više čvorova **zanimljivo, izazovno, teško i obavezno**
  - ....
- *Don't tell me the sky's the limit when there are footprints on the moon. – Paul Brandt*

Zaključak

Ako bi probali da zaključimo sve o čemu smo pričali tokom ovog predmeta, to bi bilo:

Thinking doesn't guarantee that we won't make mistakes. But not thinking guarantees that we will.



(Leslie Lamport, Turing Award, amongst others)

Dodatni materijali

- Tokenbucket
- Understanding Rate Limiting Algorithms
- Managing Your Data Lifecycle with Time to Live Tables
- An Improved Token Bucket Algorithm for Service Gateway Traffic Limiting
- Understanding Compression
- A Guide to Data Compression Methods