Klassifisering av mekanismer

Block-baserte

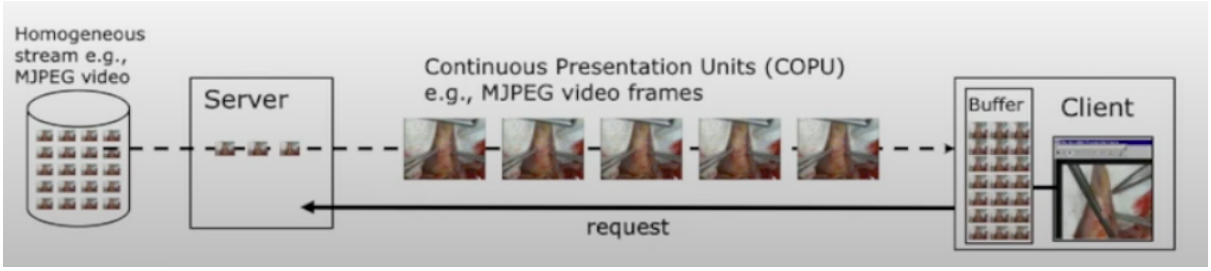
“FIFO, LRU, CLOCK, SC, LFU, …” er alle “block-level caching”/block-baserte. Dvs, at disse algoritmene behandler hver eneste blokk som en uavhengig enhet. “Block” vil i denne sammenhengen angi de rammene i bufferet hvor “A” er en blokk f.eks. Disse algoritmene vil da bytte ut blokkene basert på algoritmene sine karakteristikker som at FIFO håndterer blokkene slik at de første som kom inn, er også de som kommer først ut.

Streaming-baserte

“BASIC, DISTANCE, IC(Interval Caching), GIC(Generalized Interval Caching), SAM(Split and Merge, …)” ansees som “stream-dependent caching“/stream-baserte. Enheten er ikke som i block-baserte hvor vi ser på enhver blokk som en enhet, men vi ser nå på strømmen av data som enhet. Disse er knyttet til videoavspilling som er viktig å merke seg.

Streaming-baserte-Caching-strategier

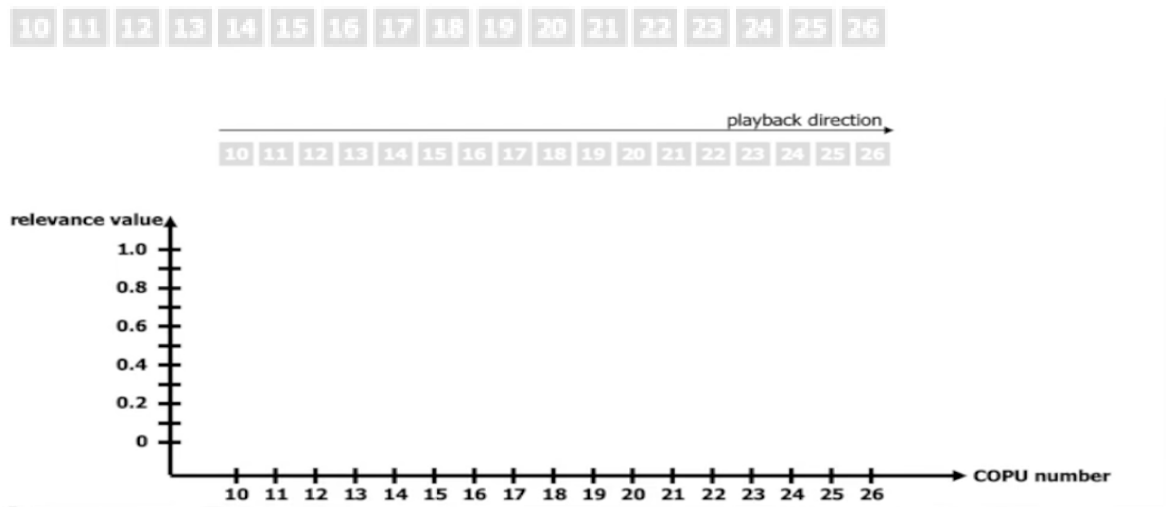
Least/Most Relevant for Presentation (L/MRP)



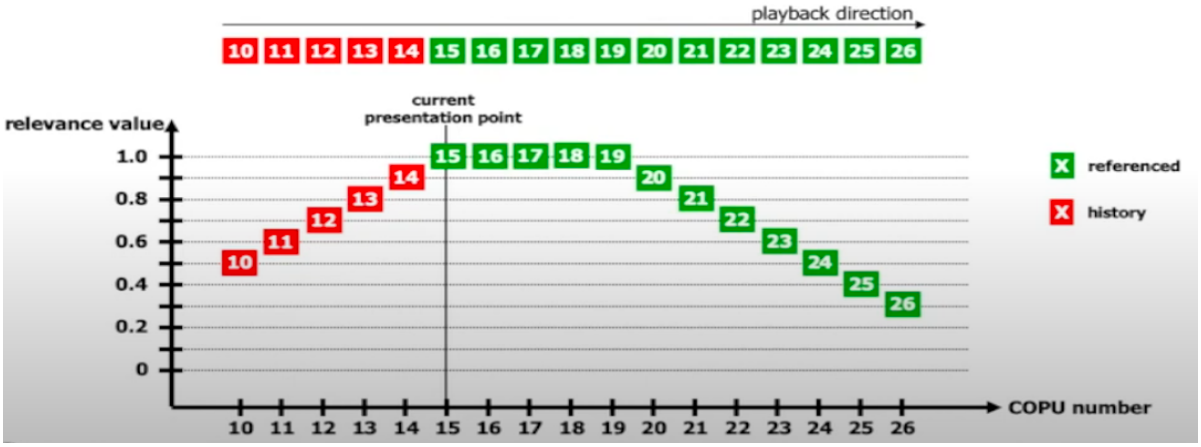
Dette er en type streaming-basert-caching-strategy som har et buffer-håndteringsmekanisme for en interaktiv, kontinuerlig strøm av data. Denne strategien fungerer ved at den prøver å laste inn og oppbevare de enhetene(data-streamene) som er mest relevante for en presentasjon som f.eks en videoavspilling, fra disken. De enhetene som er minst relevante er de som erstattes/kastes med de mer relevante enhetene fra disken.

Ideen bak strategien, er at hver enhet vil bli gitt en “relevans” verdi, tilsvarende en prioritsverdi i dynamiske algoritmer i henhold til hvor vi er i videoavspillingen. Dette avhenges av hvilken modus man har i videoavspillingen(framover, bakover, fast-forward, fast-backward, jump). Vi har også et “presentation point”/presentasjonspunkt som understreker det elementet vi er i presentasjonen altså den bildesnuttet vi er i videoen. Det finnes ulike funksjoner for hvordan elementene er satt opp, men i hovedsak så har man

brukt “COPUS” som er “Continuous object presentation units”. Går ut på at vi stokker om enhetene kontinuerlig etter hverandre. “COPUS” blir vist i bildet under:

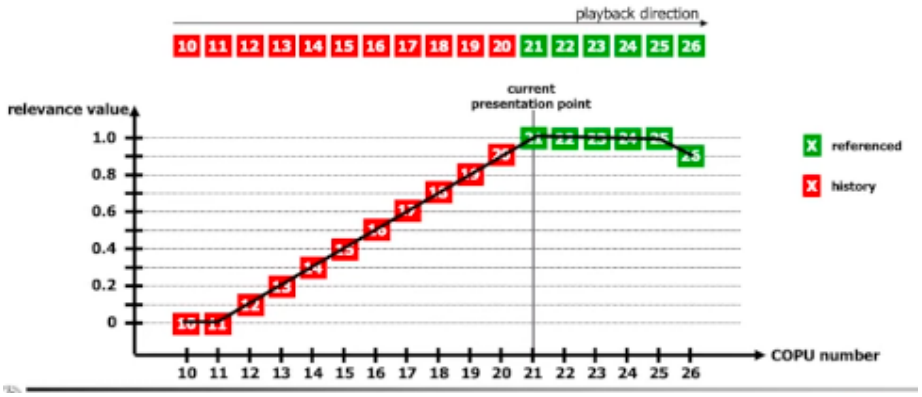


Nå skal vi se på et eksempel for hvordan (L/MRP) fungerer. Vi har en relevanse-verdi meter til venstret som angir hvilken relevanse-verdi som en enhet har blitt tildelt.

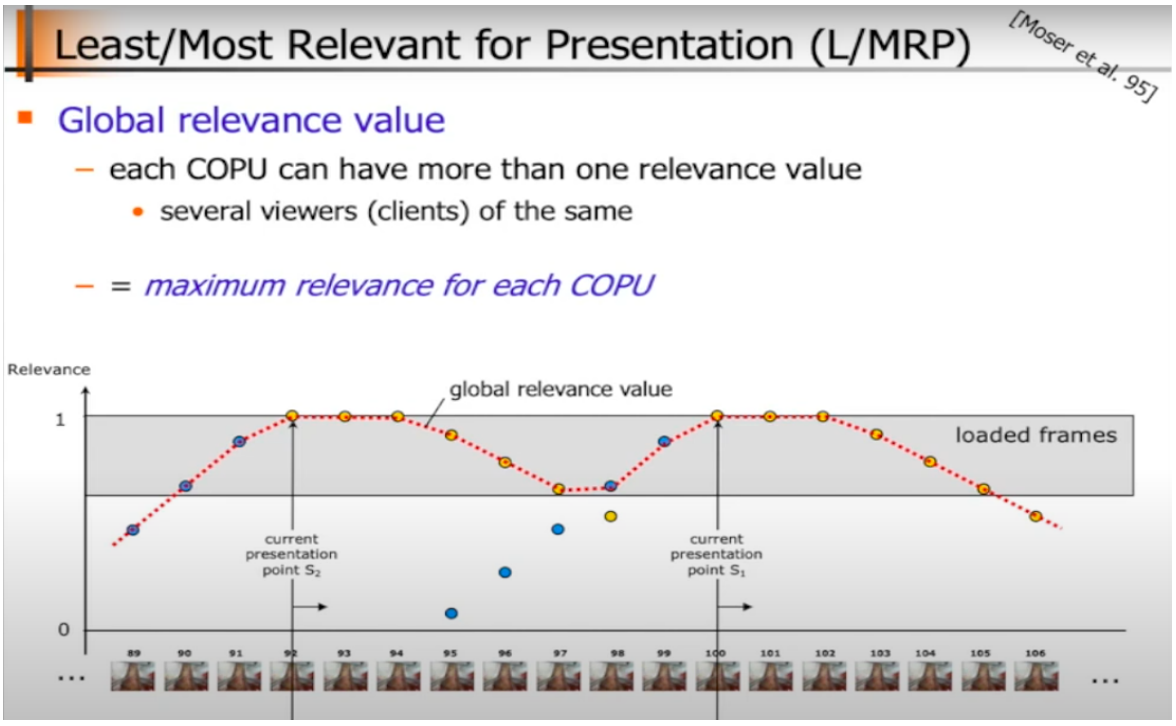


I bildet over, så er vi foreløpig på element “15” i vår presentasjon altså presentasjonspunktet. Da har vi to fordelinger for relevanse-verdien hvor vi har “history”. Vi ser at “15” for den høyeste verdien, imens de som kom før, har lavere relevans-verdi. Grunnen er at vi allerede har spilt disse elementene i presentasjonen så de vil få lavere relevans-verdi og desto lenger unna du er fra “15” desto mindre verdi for du.

Den andre fordelingen, er “referenced”. Disse er elementer som vil ha høyere relevans-verdi siden vi vil trenge dem i nærere fremtid og i henhold til modusen også. Siden vi spiller framover så vil jo “15,16,17,18,19” være relevante siden vi spiller fremover, ikke bakover. Hadde vi spilt bakover, så ville “14,13,12,11,10” vært mer relevante.



Nå er den foreløpige presentasjonspunktet “21” hvor vi ser at “15,16,17,18” er i “history” fordelingen. Men det man kan si er at relevansverdien vil man få avhengig av hvilken modus man befinner seg i og distansen mellom et element og presentasjonspunktet.



Så dette bildet over er ment for å vise tilfeller hvor, hva hvis vi har flere klienter? Hvordan skal vi da vite hvilke elementer som er mest relevante? Vel, gjennom den globale relevanse-verdien så vet vi de elementene som er mest relevante som burde prioriteres. Dette er avmerket i den gråsonen over.

Fordelene ved (L/MRP)

- Føre til være diskaksesser siden vi angir en relevans-verdi til hvert element og gjennom “COPUS” vi elementene ligge etter hverandre hvor vi ikke vil kaste mange elementer vekk fra presentasjonen. Dette skyldes av at modusen som blir spilt i

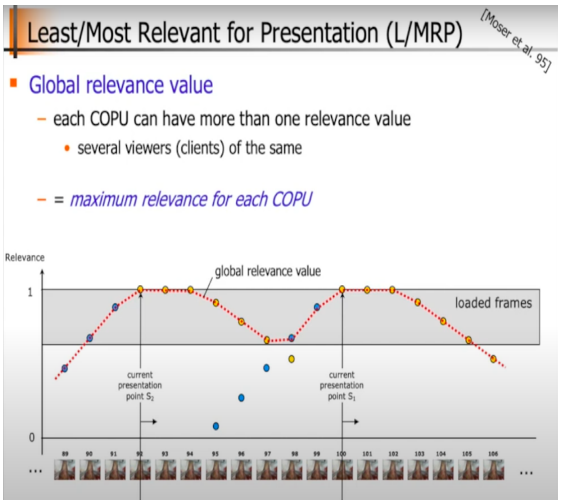
videosnuttet, gjør at man nødvendigvis ikke kaster de fleste elementer siden man kan gå fra å spille fremover eller spille bakover, noe som endrer verdiene til elementene.

- Støtter interaktivitet, ganske enkelt å hoppe fram og tilbake
- Støtter prefetching, henter en større mengde data som er av type prefetching

Ulempene ved (L/MRP)

- Kostbart, du skal beregne en relevans-verdi for hvert eneste dataelement som blir kastet inn og ut av minnet. Man må kalkulere relevansverdi for alle "COPUS" for hver runde.
- Er også "targeted" kun for single "streams" (users)

Interval Caching



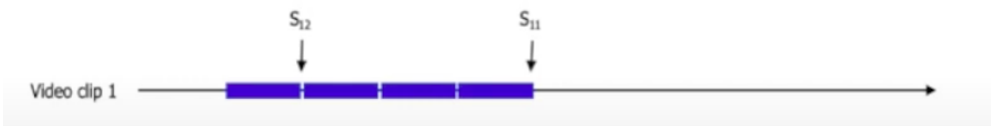
Interval caching (IC) is a caching strategy for streaming servers

- caches data between requests for same video stream – based on playout intervals between requests
- *following* requests are thus served from the cache filled by *preceding* stream
- sort intervals on length, buffer requirement is data size of interval
- to maximize cache hit ratio (minimize disk accesses) the shortest intervals are cached first

Dette er en strategi som blir benytter for "streaming servers", altså streams hvor vi har flere klienter som ønsker å se på samme video-stream. Hvis vi bruker eksempelet i bildet over, hvor vi har "S1" og "S2", ser vi at noen av elementene i enden av "S1", er de som "S2" starter med å kjøre. Da kan vi oppbevare noen av elementene i Cachen siden noen av elementene er innenfor intervallet mellom "S1" og "S2". Da slipper vi å aksessere ting fra disken når vi kan heller bruke Cache istedenfor. Sjekk eksempelet under for bedre forklaring.

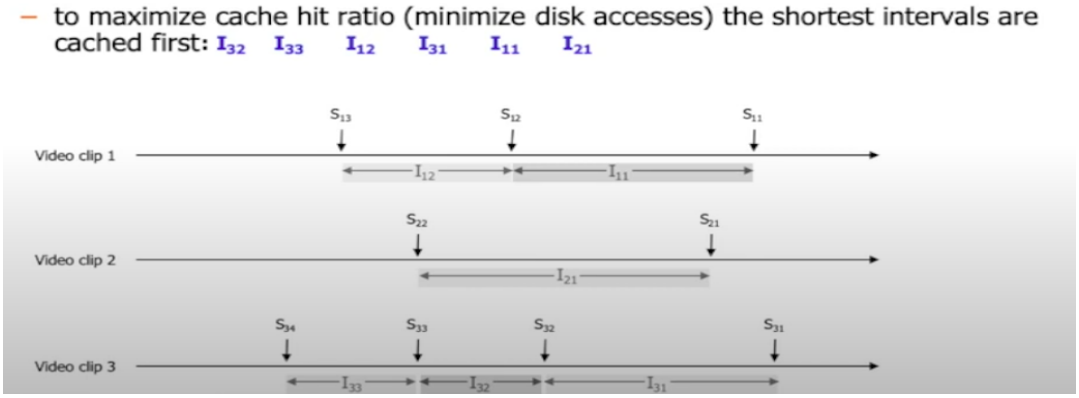
1:

2:



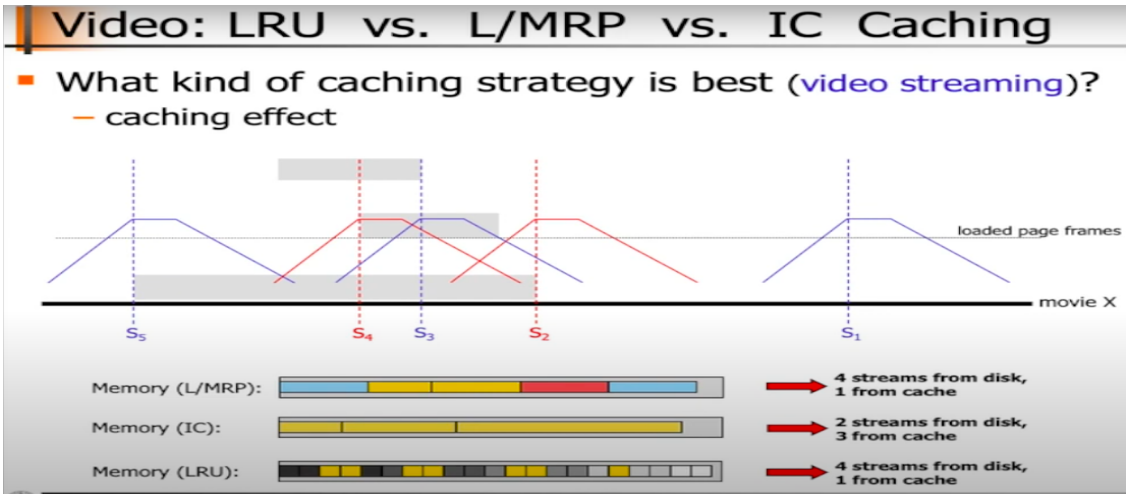
3: 

I bildet “1”, har vi at “S1” har kjørt til et punkt lenger enn “S2”. “S2” må aksessere disken til å kjøre frem til den første blå blokken som “S1” har aksessert fra disken selv. Når “S2” kommer til dette punktet, så kan “S2” aksessere dataen fra cache istedenfor disken siden “S1” allerede har gjort arbeidet fra før av. “S2” får hjelp fra “S1” til å da aksessere data fra chachen og kan følge veien gjennom “S1”. Slik fungerer denne strategien i nøyere detalj.

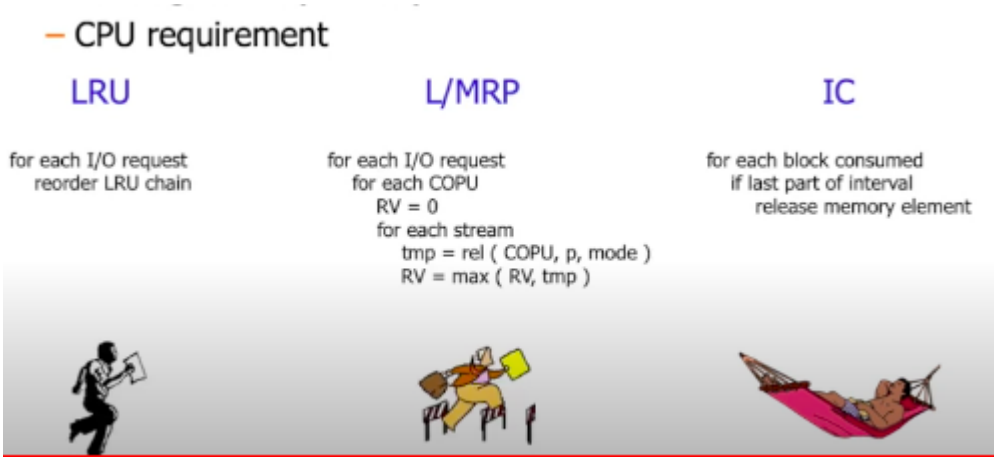


Her ser vi et nærmere eksempel på flere klienter og et videosnutt hvor vi maksimerer cache-hit ratioen siden klientene utnytter intervallet mellom hverandre.

Hvilken er best: LRU vs L/MRP vs IC Caching



Med tanke på caching, så ser vi at “Interval-Cache” var best når det kom til å minimere disk-aksessering, men har noe ubrukt minne. De to andre derimot aksesserer mer fra disken, men (L/MRP) har noe ubrukt minne.



Her ser vi nødvendigheter ved algoritme-oppbyggingen til de ulike strategiene hvor LRU, kan være komplisert med tanke på å stacke om på lenkelisten etter responsbittet til en prosess(Sjekk “Minne II” til å skjønne hva som menes). Dette kan være langt, men hvis denne er lang og komplisert, så er L/MRP mer komplisert og lengre på grunn av responsverdien for hver stream og at man må håndtere i henhold til hvilken modus man befinner seg i. Interval Caching er enklere siden man har forhåndsdefinert hvilke intervaller som skal caches og hvilke som skal kastes ut. Men for å konkludere så avhenges det av aksessmønstret som er blitt oppgitt hvor da en LRU er mer nyttig en IC f.eks.