LINKÖPINGS UNIVERSITET

TDIU11 - Operativsystem

Rapport 2

Authors

Villard Claymore Littecke, villi002@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

Vårtermin 2026
Version 1.1

2026-02-23

1 Revisionshistorik

Ver.	History	Date
1.0	First version	260223

2 Introduktion

Utvecklingen av moderna lagringsmedier har förändrat förutsättningarna för filsystemdesign. Traditionella filsystem utvecklades för mekaniska hårddiskar där söktid och rotationslatens dominerade åtkomstkostnaden. I dagens system används istället NAND-flash i form av exempelvis eMMC och SSD. Dessa saknar rörliga delar och erbjuder låg åtkomstlatens men har andra begränsningar. Flashminne kräver att block raderas innan de kan skrivas om och varje block har ett begränsat antal skrivcykler. Den fysiska lagringen hanteras dessutom via en Flash Translation Layer(FTL) som utför adressöversättning och intern garbage collection.

Dessa egenskaper innebär att slumpmässiga skrivningar och frekventa metadatauppdateringar kan leda till ökad write amplification, försämrad prestanda och minskad livslängd. Filsystem som är optimerade för hårddiskar utnyttjar därför inte flashminnets egenskaper fullt ut. Log-strukturerade filsystem introducerades för att omvandla små slumpmässiga skrivningar till större sekventiella operationer. Klassiska implementationer är dock inte direkt anpassade till moderna flashenheter och deras underliggande FTL.

Flash-Friendly File System (F2FS) utvecklades för att hantera dessa problem. Filsystemet bygger vidare på principerna från log-strukturerade filsystem men inför designval som är anpassade för flashminnets egenskaper. Dessa omfattar en flashmedveten diskstruktur, alternativ metadatahantering genom Node Address Table, separering av data baserat på uppdateringsfrekvens samt adaptiv loggning för att bibehålla stabil prestanda vid hög lagringsgrad.

Syftet med denna rapport är att beskriva hur F2FS är uppbyggt, hur det vidareutvecklar den log-strukturerade modellen och varför dess design lämpar sig för moderna flashbaserade lagringsenheter. Rapporten analyserar centrala arkitekturval och diskuterar hur dessa påverkar prestanda, write amplification och systemets robusthet.

3 Bakgrund

Log-strukturerade filsystem(LFS) introducerades för att hantera det växande prestandagapet mellan processor och sekundärlagring. Medan CPU-prestanda har förbättrats snabbt har diskars åtkomsttid särskilt söktid och rotationslatens förbättrats betydligt långsammare. Samtidigt har större primärminnen gjort det möjligt att cacha en stor andel av alla läsoperationer. Detta innebär att diskens arbetsbörda i allt större utsträckning domineras av skrivningar. Eftersom skrivningar i slutändan måste genomföras fysiskt på lagringsmediet blir skrivprestanda en avgörande faktor för systemets totala genomströmning.

Traditionella filsystem såsom Unix Fast File System (FFS) organiserar data och metadata på fasta platser på disken. En enskild filoperation kan därför kräva flera separata diskaccesser exempelvis för filens inode, katalogposter, data och tillhörande metadata. Även om systemen försöker placera relaterad information nära varandra leder informationsspridning ofta till flera sökningar per operation. Problemet förstärks av att metadatauppdateringar i många fall sker synkront vilket innebär att applikationer måste vänta på att diskoperationer slutförs innan exekveringen kan fortsätta.

LFS introducerar en alternativ strategi där alla uppdateringar både data och metadata skrivs sekventiellt till disk som en del av en växande logg. Istället för att skriva små block utspridda över disken samlas ändringar i minnet och skrivs ut i större sekventiella överföringar. Detta gör det möjligt att utnyttja diskens maximala sekventiella bandbredd och omvandla många små slumpmässiga skrivningar till effektiva sekventiella operationer.

Eftersom inoder inte längre ligger på fasta platser använder LFS en inode map som mappar inode-nummer till aktuell fysisk adress. Varje gång en inode uppdateras skrivs en ny version i loggen, och inode map uppdateras därefter. Systemet använder dessutom en checkpoint-region på en fast plats på disk som pekar ut den senaste konsistenta versionen av centrala datastrukturer. Vid en krasch kan systemet återställas genom att läsa senaste checkpointen och därefter utföra roll-forward genom att analysera loggens senare delar.

En central utmaning i log-strukturerade filsystem är hanteringen av fri diskplats. När block skrivs om lämnas äldre versioner kvar i loggen, vilket skapar fragmenterad friyta. För att kunna fortsätta skriva sekventiellt måste systemet genomföra segment cleaning. Detta innebär att segment läses in, giltiga block identifieras och kopieras till nya segment så att hela segment kan frigöras. Segmentens utnyttjandegrad det vill säga andelen live-data påverkar direkt systemets write cost vilket beskriver hur mycket extra diskaktivitet som krävs per byte ny data.

Valet av städpolicy är avgörande för prestandan. En greedy-policy som alltid städar segment med minst live-data är enkel att implementera men inte alltid optimal. En cost-benefit-policy som även tar hänsyn till segmentens ålder kan bättre separera block som uppdateras ofta från block som är stabila över tid. På så sätt kan write cost minskas och städningen bli mer effektiv över längre tidsperioder.

Trots sina fördelar har klassisk LFS begränsningar. Rekursiva metadatauppdateringar ofta kallade wandering tree-problemet kan leda till att en enda dataändring orsakar flera extra skrivningar. Vid hög lagringsgrad kan cleaning bli kostsam och orsaka betydande prestandaförluster. Dessa utmaningar blir särskilt relevanta i moderna lagringsmiljöer där NAND-flash dominerar och har egna egenskaper, såsom erase-before-write och ett begränsat antal skrivcykler.

Det är mot denna bakgrund som F2FS introduceras som ett vidareutvecklat log-strukturerat filsystem designat för att hantera både LFS inneboende utmaningar och flashminnets specifika egenskaper.

4 Översikt av F2FS

F2FS är ett filsystem utvecklat för att utnyttja moderna flashbaserade lagringsenheter såsom eMMC och SSD. Till skillnad från traditionella filsystem som i första hand designades för mekaniska hårddiskar har F2FS utformats med hänsyn till flashminnets specifika egenskaper. Dessa inkluderar kravet på erase-before-write, begränsat antal skrivcykler per block samt den indirekta hanteringen av fysisk lagring via en Flash Translation Layer. [1]

F2FS bygger vidare på den log-strukturerade modellen där uppdateringar skrivs sekventiellt för att omvandla små slumpmässiga skrivningar till större sekventiella operationer. Samtidigt adresserar F2FS flera av de begränsningar som identifierats i klassiska log-strukturerade filsystem. Istället för att enbart förlita sig på append-only loggning introducerar F2FS en diskstruktur som är anpassad till flashens interna organisation, en alternativ metod för metadatahantering genom Node Address Table samt mekanismer för att separera data baserat på uppdateringsfrekvens. [1]

Ett centralt mål med F2FS är att minska write amplification och därigenom både förbättra prestanda och förlänga lagringsenhetens livslängd. Systemet använder flera parallella loggar för att separera så kallad varm och kall data, vilket gör att segmentstädning kan genomföras mer effektivt. Vid hög lagringsgrad växlar F2FS dessutom loggningsstrategi för att undvika de kraftiga prestandafall som annars kan uppstå i log-strukturerade filsystem. [1]

Sammantaget kan F2FS beskrivas som en vidareutveckling av LFS där designvalen är explicit anpassade för moderna flashmiljöer. I följande avsnitt analyseras dessa designkomponenter mer detaljerat.

5 Design och arkitektur av F2FS

F2FS är konstruerat från grunden för att ta hänsyn till de begränsningar och egenskaper som kännetecknar moderna flash-baserade lagringsenheter. Även om filsystemet bygger på principerna bakom LFS introducerar det flera arkitektoniska förbättringar för att minska write amplification, förbättra cleaning-effektivitet och stabilisera prestanda vid hög lagringsgrad. [1] Nedan beskrivs de centrala komponenterna i F2FS design.

5.1 On-Disk Layout

F2FS organiserar lagringen i en hierarkisk struktur bestående av segment, sektioner och zoner. Ett segment är den grundläggande allokeringsenheten och används för att skriva data sekventiellt. Flera segment bildar en sektion vilket är den enhet som används vid cleaning. Zoner består i sin tur av flera sektioner och används för att bättre anpassa filsystemets loggar till den underliggande Flash Translation Layer (FTL). [1]

Disklayouten är uppdelad i flera områden: Superblock (SB), Checkpoint (CP), Segment Information Table (SIT), Node Address Table (NAT), Segment Summary Area (SSA) samt Main Area. Main Area innehåller faktiska data- och node-block. SIT håller reda på giltiga block per segment och används vid cleaning, medan SSA innehåller metadata som kopplar block till deras ägare. [1]

Denna struktur är utformad för att minimera konflikter med FTLs interna garbage collection genom att separera skrivströmmar och anpassa allokeringsenheter till flash-minnets raderingsblock. [1]

5.2 Node-struktur och NAT

En central innovation i F2FS är användningen av Node Address Table (NAT). I traditionell LFS leder uppdateringar av datablock till rekursiva uppdateringar av pekarblock, ett fenomen känt som wandering tree-problemet. Detta orsakar omfattande metadata-skrivningar. [1]

F2FS löser detta genom att tilldela varje node-block ett unikt node-ID och lagra dess fysiska adress i NAT. När ett node-block flyttas behöver endast dess post i NAT uppdateras, istället för hela pekarstrukturen. [1]

Node-strukturen består av inode-block, direkta node-block och indirekta node-block. Inode-blocket innehåller filmetadata och pekare till data medan indirekta noder refererar vidare till andra node-block. Genom NAT isoleras adressuppdateringar vilket reducerar write amplification och förbättrar skalbarheten för stora filer. [1]

5.3 Multi-head Logging

F2FS använder flera parallella loggar så kallad multi-head logging för att separera data baserat på uppdateringsfrekvens. Data och metadata klassificeras som hot, warm eller cold. Till exempel betraktas katalogdata och direkt noder som hot, medan multimediafiler ofta klassificeras som cold. [1]

Genom att skriva data med liknande temperatur till samma logg minskar risken att stabil (cold) data måste flyttas vid cleaning. Detta skapar en mer effektiv segmentfördelning där vissa segment snabbt blir helt ogiltiga och kan raderas utan att giltiga block behöver migreras. [1]

Multi-head logging förbättrar därmed cleaning-effektiviteten och reducerar write amplification. [1]

5.4 Cleaning

Cleaning är processen där ogiltiga block rensas för att frigöra nya segment för framtida skrivningar. F2FS utför cleaning på sektionsnivå och använder två olika strategier: foreground cleaning och background cleaning. [1]

Foreground cleaning aktiveras när antalet fria sektioner är lågt och använder en greedy-policy där den sektion med minst antal giltiga block väljs för att minimera latens. Background cleaning körs periodiskt och använder

en cost-benefit-policy som även tar hänsyn till segmentens ålder för att förbättra separationen mellan hot och cold data. [1]

Under cleaning identifieras giltiga block via bitmap-information i SIT och migreras till nya segment. Denna design begränsar overhead och förbättrar långsiktig prestanda. [1]

5.5 Adaptiv loggning

Klassiska log-strukturerade filsystem kan drabbas av kraftiga prestandafall när lagringsutrymmet nästan är fullt, eftersom strikt sekventiell loggning kräver kontinuerlig cleaning. [1]

F2FS implementerar därför adaptiv loggning, där systemet växlar mellan normal logging och threaded logging beroende på antalet fria sektioner. Vid normal logging skrivs data enbart till rena segment för att uppnå sekventiella skrivningar. När fria sektioner sjunker under en tröskel aktiveras threaded logging där nya data skrivs till lediga utrymmen i redan använda segment. [1]

Även om threaded logging kan generera mer slumpmässiga skrivningar är detta acceptabelt på moderna SSD-enheter och bidrar till att undvika drastiska prestandafall vid hög fyllnadsgrad. [1]

5.6 Checkpointing och återhämtning

För att säkerställa konsistens vid systemkrascher använder F2FS checkpointing. Vid en checkpoint skrivs filsystemets aktuella metadata, inklusive NAT- och SIT-information, till disk tillsammans med en versionsidentifierare. [1]

Vid återstart kan systemet genom roll-back recovery återgå till den senaste konsistenta checkpointen. För att förbättra prestanda vid frekventa fsync-anrop använder F2FS även roll-forward recovery. Istället för att skriva fullständig metadata vid varje synkronisering skrivs endast nödvändiga data- och node-block. Vid återstart kan resterande metadata rekonstrueras genom att analysera loggen. [1]

Denna mekanism reducerar latens vid databastunga arbetslaster och minskar mängden metadata-skrivningar. [1]

6 Varför F2FS passar för flash-lagring

F2FS är särskilt anpassat för flash-lagring genom att kombinera en log-strukturerad design med mekanismer som tar hänsyn till flash-minnets begränsningar. Genom att skriva data sekventiellt minskas effekten av slumpmässiga skrivningar vilket är viktigt då NAND-flash kräver erase-before-write. Segment-, section- och zone-indelningen är anpassad för att kordinera med FTL och minska intern garbage collection. Vidare bidrar temperaturseparering och adaptiv loggning till lägre write amplification och stabil prestanda även vid hög fyllnadsgrad.

7 Prestandautvärdering

Den experimentella utvärderingen visar att F2FS ger betydande prestandaförbättringar jämfört med EXT4, BTRFS och NILFS2, särskilt vid arbetslaster med många slumpmässiga skrivningar och frekventa fsync-anrop. På mobila system uppmättes upp till 3.1× bättre prestanda vid random writes och omkring 2× förbättring för SQLite-arbetslaster. På serversystem visade F2FS stabil prestanda och överträffade EXT4 i flera scenarier. Resultaten indikerar att designvalen effektivt reducerar write amplification och förbättrar sustained throughput. [1]

8 Sammanfattning

F2FS är ett flash-optimerat filsystem som bygger vidare på log-strukturerade principer men introducerar flera förbättringar, såsom NAT, multi-head logging och adaptiv loggning. Genom att anpassa sig till flash-minnets egenskaper och FTL:s beteende uppnår F2FS förbättrad prestanda och stabilitet jämfört med traditionella filsystem.

9 Egna tankar

F2FS framstår som en genomtänkt vidareutveckling av LFS där praktiska begränsningar i flash-minne adresseras på flera nivåer i designen. Särskilt intressant är hur enkla heuristiker för temperaturseparering ger märkbara prestandavinster utan komplex runtime-analys. Samtidigt kan den ökade komplexiteten i loggning och recovery innebära underhållsutmaningar.

10 Diskussions frågor

- Hur representativa är de experimentella arbetslasterna för olika typer av verkliga användare?
- Vilka är de potentiella nackdelarna med multi-head logging?
- Finns det situationer där EXT4 kan vara ett bättre val än F2FS?

Referenser

[1] C. Lee, D. Sim, J.-Y. Hwang, and S. Cho, “F2FS: A New File System for Flash Storage,” in *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST ’15)*, Santa Clara, CA, USA, 2015.