دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

عنوان:

پروژه درس سیستم‌های عامل

نگارش

پوریا محمودخان شیرازی

امیرحسین ملک‌محمدی

عسل مسکین

سید علیرضا میرکنی بنادکی

بهمن ۱۴۰۳

۱ مقدمه

در دنیای امروز با افزایش سرعت و حجم داده‌های تولیدشده، نیاز به زیرساخت‌های ذخیره‌سازی کارآمد بیش از پیش احساس می‌شود. یکی از تکنولوژی‌های مهم در این زمینه، حافظه نهان است که با هدف افزایش سرعت پردازش و کاهش تأخیر در دسترسی به داده‌ها به‌کار گرفته می‌شود. ما با استفاده از حافظه نهان می‌توانیم داده‌های پرکاربرد را به‌صورت موقت ذخیره کنیم و با دسترسی سریع‌تر به این داده‌ها، عملکرد سامانه‌های ذخیره‌سازی را بهبود بخشیم. با این حال، چالش اصلی انتخاب سیاست مناسب مدیریت حافظه نهان است که تأثیر مستقیمی بر کارایی سیستم دارد.

در این پروژه، به بررسی و شبیه‌سازی سیاست‌های مختلف مدیریت حافظه نهان خواهیم پرداخت. هدف ما این است که با استفاده از داده‌های واقعی و متن‌باز شرکت Alibaba، عملکرد این سیاست‌ها را تحلیل و مقایسه کنیم. با پیاده‌سازی چند سیاست پرکاربرد و مقایسه آن‌ها با سیاست ایده‌آل Belady/Oracle، تلاش خواهیم کرد تا بینشی دقیق‌تر در مورد رفتار حافظه نهان به دست آوریم و روش بهینه را برای استفاده در سامانه‌های ذخیره‌سازی شناسایی کنیم.

۱-۱ تعریف مسئله

در این پروژه، قصد داریم یکی از عوامل کلیدی مؤثر بر عملکرد سامانه‌های ذخیره‌سازی، یعنی سیاست مدیریت حافظه نهان را بررسی کنیم. ما خواهیم دید که چگونه انتخاب یک سیاست مناسب می‌تواند نرخ Cache Hit را افزایش داده و زمان پاسخ‌دهی سیستم را بهبود بخشد. در مقابل، یک سیاست نامناسب می‌تواند منجر به افزایش Cache Miss شود و عملکرد کلی سیستم را کاهش دهد.

ما در این پروژه سیاست‌های مختلفی همچون LRU (Least Recently Used)، N-Hit، ARC (Adaptive Replacement Cache) و LARC (Learning Adaptive Replacement Cache) را پیاده‌سازی خواهیم کرد. سیاست LARC یک نسخه توسعه‌یافته از ARC است که تلاش می‌کند با یادگیری رفتار درخواست‌ها و پیش‌بینی دسترسی‌های آینده، عملکرد بهتری ارائه دهد.

همچنین، سیاست ایده‌آل Belady/Oracle را مرجع برای مقایسه عملکرد سایر سیاست‌ها در نظر خواهیم گرفت. این سیاست با پیش‌بینی دقیق درخواست‌های آینده بهترین تصمیمات ممکن را می‌گیرد، اما به دلیل نیاز به اطلاعات آینده، در محیط واقعی قابل پیاده‌سازی نیست و صرفاً مبنایی برای تحلیل الگوریتم‌هاست.

هدف ما این است که با شبیه‌سازی این سیاست‌ها و تحلیل معیارهای عملکردی همچون نرخ

Cache Hit و Cache Miss، زمان و حافظه استفاده‌شده، نقاط قوت و ضعف هر سیاست را بررسی و مقایسه کنیم.

۲ سیاست LRU

الگوریتم LRU (Least Recently Used) یکی از سیاست‌های جایگزینی در حافظه نهان است که برای مدیریت کارآمد داده‌ها طراحی شده است. این الگوریتم بر اساس میزان استفاده اخیر از داده‌ها عمل می‌کند. زمانی که حافظه نهان پر شود، داده‌ای که کمترین استفاده را در بازه اخیر داشته است، حذف می‌شود تا فضای لازم برای داده جدید فراهم گردد.

در این روش، هر بار که یک داده در حافظه نهان مورد دسترسی قرار می‌گیرد، اولویت آن افزایش می‌یابد، به این معنا که در مکان جلوتری در لیست داده‌های پرکاربرد قرار می‌گیرد. اگر داده‌ای برای مدت طولانی بدون استفاده باقی بماند، اولویت آن کاهش یافته و در صورت نیاز به فضای جدید، این داده حذف خواهد شد.

۲-۱ توضیح برنامه شبیه‌سازی

کد مربوط به این سیاست را به کمک زبان پایتون شبیه‌سازی کرده‌ایم که در فایل LRU.py موجود است. حال به توضیح بخش‌های مهم آن می‌پردازیم:

۲-۱-۱ ساختار کد و اجزای اصلی

این کد از یک کلاس به نام LRUCache و یک تابع cache_simulator تشکیل شده است. متغیر اصلی در این کد ظرفیت حافظه نهان است که آن را به صورت max_capacity نمایش داده‌ایم. هم‌چنین حافظه نهان را به کمک داده‌ساختار OrderedDict پیاده‌سازی کرده‌ایم. در واقع به این صورت عمل می‌کند که داده‌ای که جدیدتر استفاده شده را به انتهای آن می‌بریم.

۲-۱-۲ توابع اصلی کد

ابتدا توابع مربوط به کلاس LRUCache را توضیح می‌دهیم.

- در تابع load_and_filter_data فایل csv ورودی را می‌خوانیم.

- در تابع access_or_update_cache در صورتی که داده جدید از قبل در حافظه نهان وجود داشت آن را به انتهای دیکشنری می‌بریم تا دیرتر حذف شود. هم‌چنین از آن جا که hit رخ داده‌است true را خروجی می‌دهیم.

  در غیر این صورت اگر حافظه نهان پر بود داده‌ای که اخیرا استفاده نشده را pop می‌کنیم. سپس داده جدید را در حافظه نهان قرار می‌دهیم. هم‌چنین از آن‌جا که miss اتفاق افتاده‌است مقدار false را خروجی می‌دهیم.

- در تابع simulate_lru_policy به ازای داده‌های موجود تابع access_or_update_cache را فراخوانی کرده و مقادیر مربوط را اپدیت می‌کنیم.

- در دو تابع collect_statistics و display_reuslt نتایج به دست آمده را ذخیره و نمایش می‌دهیم.

در نهایت به کمک تابع cache_simulator یک حافظه نهان با سیاست LRU می‌سازیم و داده ها را روی آن شبیه‌سازی می‌کنیم.

۲-۲ نتایج عملکرد

در این قسمت نتایج عملکرد حافظه پنهان با سیاست LRU روی سه اندازه مختلف حافظه نهان نشان می‌دهیم:

جدول ۱: نتیجه شبیه‌سازی مجموعه‌داده A669 با سیاست LRU و اندازه حافظه نهان ۵۰۰۰

Metric	Count	Ratio
Read Requests	۲۵۹۵۹۱۹۵۶	
Read Hits	۲۰۸۴۲۸۵۵	۸۰٫۳۱٪
Read Misses	۵۱۰۹۱۰۱	۱۹٫۶۹٪
Write Requests	۱۲۹۲۸۷۶	
Write Hits	۵۹۶۹۹۸	۴۶٫۱۸٪
Write Misses	۶۹۵۸۷۸	۵۳٫۸۲٪
Total Requests	۲۷۲۴۴۸۳۲	
Total Hits	۲۱۴۳۹۸۵۳	۷۸٫۶۹٪
Total Misses	۵۸۰۴۹۷۹	۲۱٫۳۱٪

جدول ۲: نتیجه شبیه‌سازی مجموعه‌داده A129 با سیاست N-hit و اندازه حافظه نهان ۵۰۰۰

Metric	Count	Ratio
Read Requests	۴۸۶۸۱۷۲	
Read Hits	۱۴۰۷۷۸	۲٫۸۹٪
Read Misses	۴۷۲۷۳۹۴	۹۷٫۱۱٪
Write Requests	۱۲۸۴۴۳۲۰	
Write Hits	۲۹۳۲۸۸۶	۱۷٫۹۳٪
Write Misses	۱۰۵۴۱۴۳۴	۸۲٫۰۷٪
Total Requests	۱۷۷۷۱۲۴۹۲	
Total Hits	۲۴۴۳۶۶۴	۱۳٫۸۰٪
Total Misses	۱۵۲۶۸۸۲۸	۸۶٫۲۰٪

جدول ۳: نتیجه شبیه‌سازی مجموعه‌داده A108 با سیاست N-hit و اندازه حافظه نهان ۵۰۰۰

Metric	Count	Ratio
Read Requests	۱۰۶۳۰۱۷۲	
Read Hits	۵۹۳۶۷۳	۵٫۵۸٪
Read Misses	۱۰۰۳۶۴۹۹	۹۴٫۴۲٪
Write Requests	۹۵۷۵۱۴۵	
Write Hits	۱۷۲۲۸۳۱	۱۷٫۹۹٪
Write Misses	۷۸۵۲۳۱۴	۸۲٫۰۱٪
Total Requests	۲۰۲۰۵۳۱۷	
Total Hits	۲۳۱۶۵۰۴	۱۱٫۴۶٪
Total Misses	۱۷۸۸۸۸۱۳	۸۸٫۵۴٪

جدول ۴: نتیجه شبیه‌سازی مجموعه‌داده A42 با سیاست N-hit و اندازه حافظه نهان ۵۰۰۰

Metric	Count	Ratio
Read Requests	۳۰۹۸۷۳۴	
Read Hits	۷۴۰۶۱	۲٫۳۹٪
Read Misses	۳۰۲۴۶۷۳	۹۷٫۶۱٪
Write Requests	۱۹۸۹۱۹۷	
Write Hits	۸۵۷۶۸۴	۴۳٫۱۲٪
Write Misses	۱۱۳۱۵۱۳	۵۶٫۸۸٪
Total Requests	۵۰۸۷۹۳۱	
Total Hits	۹۳۱۷۴۵	۱۸٫۳۱٪
Total Misses	۴۱۵۶۱۸۶	۸۱٫۶۹٪

جدول ۵: نتیجه شبیه‌سازی مجموعه‌داده A669 با سیاست N-hit و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۲۵۹۵۹۱۹۵۶	
Read Hits	۲۳۹۷۸۵۰۷	۹۲٫۴۰٪
Read Misses	۱۹۷۳۴۴۹	۷٫۶۰٪
Write Requests	۱۲۹۲۸۷۶	
Write Hits	۵۰۳۰۵۳	۳۸٫۹۱٪
Write Misses	۷۸۹۸۲۳	۶۱٫۰۹٪
Total Requests	۲۷۲۴۴۸۳۲	
Total Hits	۲۴۴۸۱۵۶۰	۸۹٫۸۶٪
Total Misses	۲۷۶۳۲۷۲	۱۰٫۱۴٪

جدول ۶: نتیجه شبیه‌سازی مجموعه‌داده A129 با سیاست N-hit و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۴۸۶۸۱۷۲	
Read Hits	۱۶۱۶۲۹	۳٫۳۲٪
Read Misses	۴۷۰۶۵۴۳	۹۶٫۶۸٪
Write Requests	۱۲۸۴۴۳۲۰	
Write Hits	۲۵۵۳۶۵۷	۱۹٫۸۸٪
Write Misses	۱۰۲۹۰۶۶۳	۸۰٫۱۲٪
Total Requests	۱۷۷۷۱۲۴۹۲	
Total Hits	۲۷۱۵۲۸۶	۱۵٫۳۳٪
Total Misses	۱۴۹۹۷۲۰۶	۸۴٫۶۷٪

جدول ۷: نتیجه شبیه‌سازی مجموعه‌داده A108 با سیاست N-hit و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۱۰۶۳۰۱۷۲	
Read Hits	۶۰۲۹۸۴	۵٫۶۷٪
Read Misses	۱۰۰۲۷۱۸۸	۹۴٫۳۳٪
Write Requests	۹۵۷۵۱۴۵	
Write Hits	۱۷۲۶۳۱۹	۱۸٫۰۳٪
Write Misses	۷۸۴۸۸۲۶	۸۱٫۹۷٪
Total Requests	۲۰۲۰۵۳۱۷	
Total Hits	۲۳۲۹۳۰۳	۱۱٫۵۳٪
Total Misses	۱۷۸۷۶۰۱۴	۸۸٫۴۷٪

جدول ۸: نتیجه شبیه‌سازی مجموعه‌داده A42 با سیاست N-hit و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۳۰۹۸۷۳۴	
Read Hits	۱۶۰۱۵۸	۵٫۱۷٪
Read Misses	۲۹۳۸۵۷۶	۹۴٫۸۳٪
Write Requests	۱۹۸۹۱۹۷	
Write Hits	۸۷۷۵۴۲	۴۴٫۱۲٪
Write Misses	۱۱۱۱۶۵۵	۵۵٫۸۸٪
Total Requests	۵۰۸۷۹۳۱	
Total Hits	۱۰۳۷۷۰۰	۲۰٫۴۰٪
Total Misses	۴۰۵۰۲۳۱	۷۹٫۶۰٪

جدول ۹: نتیجه شبیه‌سازی مجموعه‌داده A669 با سیاست N-hit و اندازه حافظه نهان ۵۰۰۰۰

Metric	Count	Ratio
Read Requests	۲۵۹۵۹۱۹۵۶	
Read Hits	۲۵۳۷۴۹۶۲	۹۷٫۷۸٪
Read Misses	۵۷۶۹۹۴	۲٫۲۲٪
Write Requests	۱۲۹۲۸۷۶	
Write Hits	۹۸۰۲۱۹	۷۵٫۸۲٪
Write Misses	۳۱۲۶۵۷	۲۴٫۱۸٪
Total Requests	۲۷۲۴۴۸۳۲	
Total Hits	۲۶۳۵۵۱۸۱	۹۶٫۷۳٪
Total Misses	۸۸۹۶۵۱	۳٫۲۷٪

جدول ۱۰: نتیجه شبیه‌سازی مجموعه‌داده A129 با سیاست N-hit و اندازه حافظه نهان ۵۰۰۰۰

Metric	Count	Ratio
Read Requests	۴۸۶۸۱۷۲	
Read Hits	۱۱۷۵۲۲۰	۲۴٫۱۴٪
Read Misses	۳۶۹۲۹۵۲	۷۵٫۸۶٪
Write Requests	۱۲۸۴۴۳۲۰	
Write Hits	۷۳۵۵۳۶۵	۵۷٫۲۷٪
Write Misses	۵۴۸۸۹۵۵	۴۲٫۷۳٪
Total Requests	۱۷۷۷۱۲۴۹۲	
Total Hits	۸۵۳۰۵۸۵	۴۸٫۱۶٪
Total Misses	۹۱۸۱۹۰۷	۵۱٫۸۴٪

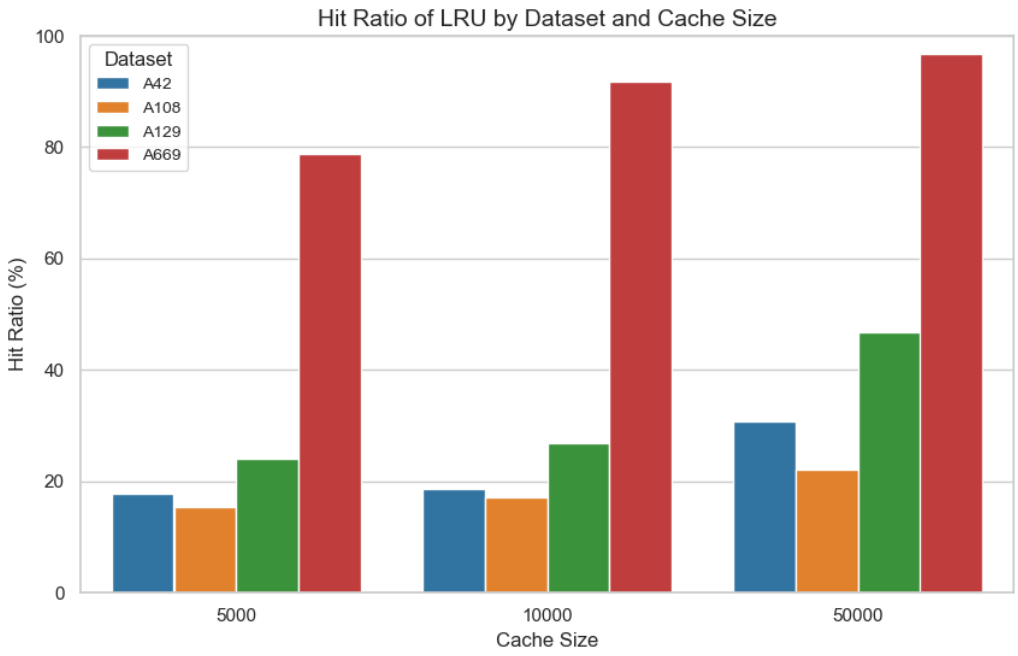
جدول ۱۱: نتیجه شبیه‌سازی مجموعه‌داده A108 با سیاست N-hit و اندازه حافظه نهان ۵۰۰۰۰

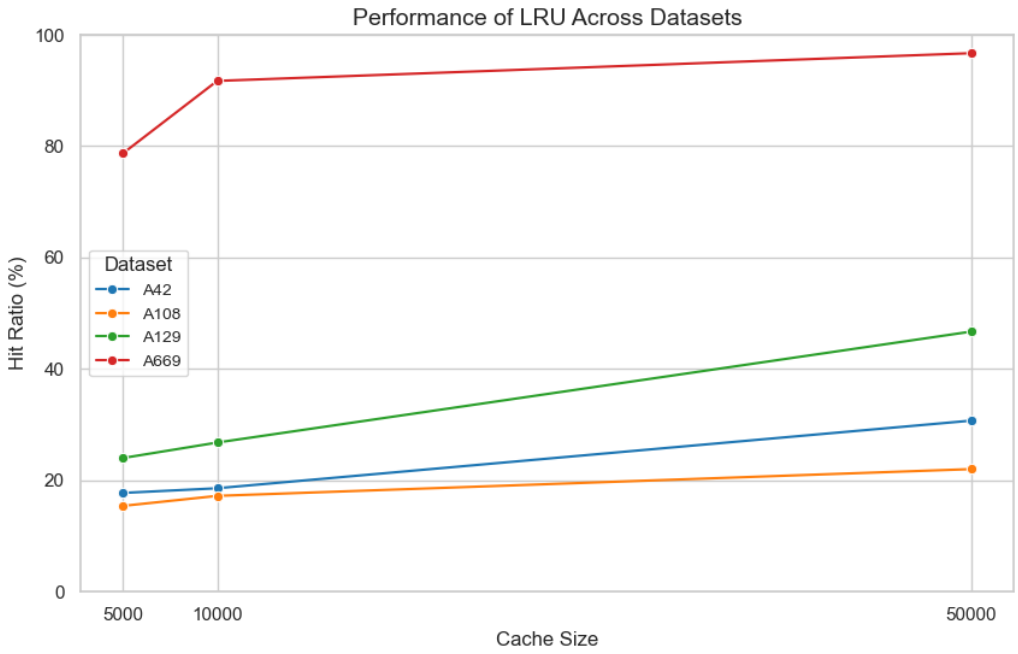
Metric	Count	Ratio
Read Requests	۱۰۶۳۰۱۷۲	
Read Hits	۱۴۴۵۷۰۰	۱۳٫۶۰٪
Read Misses	۹۱۸۴۴۷۲	۸۶٫۴۰٪
Write Requests	۹۵۷۵۱۴۵	
Write Hits	۳۰۳۸۹۳۱	۳۱٫۷۴٪
Write Misses	۶۵۳۶۲۱۴	۶۸٫۲۶٪
Total Requests	۲۰۲۰۵۳۱۷	
Total Hits	۴۴۸۴۶۳۱	۲۲٫۲۰٪
Total Misses	۱۵۷۲۰۶۸۶	۷۷٫۸۰٪

جدول ۱۲: نتیجه شبیه‌سازی مجموعه‌داده A42 با سیاست N-hit و اندازه حافظه نهان ۵۰۰۰۰

Metric	Count	Ratio
Read Requests	۳۰۹۸۷۳۴	
Read Hits	۱۱۵۲۹۱۰	۳۷٫۲۱٪
Read Misses	۱۹۴۵۸۲۴	۶۲٫۷۹٪
Write Requests	۱۹۸۹۱۹۷	
Write Hits	۹۶۰۰۷۰	۴۸٫۲۶٪
Write Misses	۱۰۲۹۱۲۷	۵۱٫۷۴٪
Total Requests	۵۰۸۷۹۳۱	
Total Hits	۲۱۱۲۹۸۰	۴۱٫۵۳٪
Total Misses	۲۹۷۴۹۵۱	۵۸٫۴۷٪

نمودارهای مربوط به این قسمت نیز در اینجا آورده شده است:





۳ سیاست N-hit

الگوریتم N-hit یکی از سیاست‌های مدیریت حافظه نهان است که با هدف بهره‌وری بیشتر از داده‌های پرتکرار طراحی شده است. این سیاست به این صورت عمل می‌کند که تعداد درخواست‌های هر بلوک داده را دنبال می‌کند و زمانی که تعداد این درخواست‌ها به یک مقدار آستانه مشخص N برسد، آن بلوک به حافظه نهان منتقل می‌شود. در صورت پر بودن حافظه نهان، بلوکی که کمترین تعداد درخواست را داشته باشد، حذف می‌شود تا فضای لازم برای داده جدید فراهم شود. هدف اصلی این سیاست، کاهش نرخ Cache Miss از طریق ذخیره داده‌هایی است که بیشترین استفاده را دارند.

این روش با تمرکز بر داده‌های پرتکرار، عملکرد کلی سیستم را بهبود می‌بخشد اما به دلیل نیاز به نگهداری شمارنده‌های مرتبط با تعداد درخواست‌ها، ممکن است سربار محاسباتی و حافظه‌ای به همراه داشته باشد. علاوه بر این، شناسایی بلوک قربانی ممکن است به دلیل نیاز به بررسی و مقایسه شمارنده‌های بلوک‌ها، زمان‌بر باشد.

۳-۱ توضیح برنامه شبیه‌سازی

۳-۱-۱ ساختار کد و اجزای اصلی

این کد از یک کلاس به نام NHitCache و یک تابع simulate_nhit تشکیل شده است. متغیرهای کلیدی این کد شامل ظرفیت حافظه نهان (capacity)، آستانه‌ی فعال‌سازی رهگیری (trigger_threshold) و حداقل تعداد دسترسی برای ورود به حافظه نهان (insertion_threshold) می‌باشند. حافظه نهان در قالب یک دیکشنری و ساختار داده‌ای SortedList برای مرتب‌سازی آیتم‌ها پیاده‌سازی شده است.

۳-۱-۲ توابع اصلی کد

ابتدا توابع مربوط به کلاس NHitCache را توضیح می‌دهیم.

- در تابع evict_ آیتمی که کمترین مقدار NHit را دارد (و در صورت تساوی، قدیمی‌تر است) از حافظه نهان حذف می‌شود.
- در تابع access هر بار که آیتمی درخواست شود، تعداد دفعات دسترسی آن در ساختار رهگیری افزایش می‌یابد.
- در تابع promote در صورتی که آیتم معیارهای لازم را داشته باشد، به حافظه نهان اضافه می‌شود.
- در تابع should_promote بررسی می‌شود که آیا آیتم بر اساس تعداد دفعات دسترسی و وضعیت اشغال حافظه نهان واجد شرایط ورود به حافظه نهان هست یا خیر.

در ادامه به توابع مربوط به شبیه‌سازی می‌پردازیم:

- در تابع simulate_nhit داده‌های ورودی از یک فایل CSV خوانده شده و پردازش می‌شوند. برای هر درخواست، ابتدا بررسی می‌شود که آیا در حافظه نهان قرار دارد یا خیر. در صورت وجود، آمار hit و در غیر این صورت، miss به‌روز می‌شود. سپس اگر داده قبلاً دیده نشده باشد، به عنوان cold miss علامت‌گذاری می‌شود و در صورت داشتن شرایط مناسب، به حافظه نهان اضافه می‌شود.
- در تابع collect_statistics آمار مربوط به میزان hit و miss محاسبه می‌شود.

- در تابع display_results نتایج حاصل از شبیه‌سازی به صورت جدولی نمایش داده می‌شوند.

در نهایت، تابع main چندین فایل ورودی را پردازش کرده و برای هر یک از آن‌ها شبیه‌سازی سیاست NHit را انجام می‌دهد.

۳-۲ نتایج عملکرد

در این قسمت نتایج عملکرد حافظه پنهان با سیاست N-hit روی سه اندازه مختلف حافظه نهان نشان می‌دهیم:

جدول ۱۳: نتیجه شبیه‌سازی مجموعه‌داده A669 با سیاست N-hit و اندازه حافظه نهان ۵۰۰۰

Metric	Count	Ratio
Read Requests	۲۵۹۵۹۱۹۵۶	
Read Hits	۲۰۹۵۶۶۳۸	۸۰٫۷۵٪
Read Misses	۴۹۹۵۳۱۸	۱۹٫۲۵٪
Write Requests	۱۲۹۲۸۷۶	
Write Hits	۳۷۶۷۱۷	۲۹٫۱۴٪
Write Misses	۹۱۶۱۵۹	۷۰٫۸۶٪
Total Requests	۲۷۲۴۴۸۳۲	
Total Hits	۲۱۳۳۳۳۵۵	۷۸٫۳۰٪
Total Misses	۵۹۱۱۴۷۷	۲۱٫۷۰٪

جدول ۱۴: نتیجه شبیه‌سازی مجموعه‌داده A129 با سیاست N-hit و اندازه حافظه نهان ۵۰۰۰

Metric	Count	Ratio
Read Requests	۴۸۶۸۱۷۲	
Read Hits	۷۷۷۵۷۷	۱۵٫۹۷٪
Read Misses	۴۰۹۰۵۹۵	۸۴٫۰۳٪
Write Requests	۱۲۸۴۴۳۲۰	
Write Hits	۳۴۶۸۴۹۸	۲۷٫۰۰٪
Write Misses	۹۳۷۵۸۲۷	۷۳٫۰۰٪
Total Requests	۱۷۷۷۱۲۴۹۲	
Total Hits	۴۲۴۶۰۷۰	۲۳٫۹۷٪
Total Misses	۱۳۴۶۶۴۲۲	۷۶٫۰۳٪

جدول ۱۵: نتیجه شبیه‌سازی مجموعه‌داده A108 با سیاست N-hit و اندازه حافظه نهان ۵۰۰۰

Metric	Count	Ratio
Read Requests	۱۰۶۳۰۱۷۲	
Read Hits	۸۹۹۴۵۰	۸٫۴۶٪
Read Misses	۹۷۳۹۷۲۲	۹۱٫۵۴٪
Write Requests	۹۵۷۵۱۴۵	
Write Hits	۲۲۰۶۶۹۲	۲۳٫۰۵٪
Write Misses	۷۳۶۸۴۵۳	۷۶٫۹۵٪
Total Requests	۲۰۲۰۵۳۱۷	
Total Hits	۳۱۰۶۱۴۲	۱۵٫۳۷٪
Total Misses	۱۷۰۹۹۱۷۵	۸۴٫۶۳٪

جدول ۱۶: نتیجه شبیه‌سازی مجموعه‌داده A42 با سیاست N-hit و اندازه حافظه نهان ۵۰۰۰

Metric	Count	Ratio
Read Requests	۳۰۹۸۷۳۴	
Read Hits	۷۹۶	۰٫۰۳٪
Read Misses	۳۰۹۷۹۳۸	۹۹٫۹۷٪
Write Requests	۱۹۸۹۱۹۷	
Write Hits	۸۹۹۸۹۴	۴۵٫۲۴٪
Write Misses	۱۰۸۹۳۰۳	۵۴٫۷۶٪
Total Requests	۵۰۸۷۹۳۱	
Total Hits	۹۰۰۶۹۰	۱۷٫۷۰٪
Total Misses	۴۱۸۷۲۴۱	۸۲٫۳۰٪

جدول ۱۷: نتیجه شبیه‌سازی مجموعه‌داده A669 با سیاست N-hit و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۲۵۹۵۹۱۹۵۶	
Read Hits	۲۴۳۵۳۸۱۳	۹۳٫۸۴٪
Read Misses	۱۵۹۸۱۴۳	۶٫۱۶٪
Write Requests	۱۲۹۲۸۷۶	
Write Hits	۶۳۳۱۸۳	۴۸٫۹۷٪
Write Misses	۶۵۹۶۹۳	۵۱٫۰۳٪
Total Requests	۲۷۲۴۴۸۳۲	
Total Hits	۲۴۹۸۶۹۹۶	۹۱٫۷۱٪
Total Misses	۲۲۵۷۸۳۶	۸٫۲۹٪

جدول ۱۸: نتیجه شبیه‌سازی مجموعه‌داده A129 با سیاست N-hit و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۴۸۶۸۱۷۲	
Read Hits	۸۸۴۱۱۱	۱۸٫۱۶٪
Read Misses	۳۹۸۴۰۶۱	۸۱٫۸۴٪
Write Requests	۱۲۸۴۴۳۲۰	
Write Hits	۳۸۵۳۲۸۶	۳۰٫۰۰٪
Write Misses	۸۹۹۱۰۳۴	۷۰٫۰۰٪
Total Requests	۱۷۷۷۱۲۴۹۲	
Total Hits	۴۷۳۷۳۹۷	۲۶٫۷۵٪
Total Misses	۱۲۹۷۵۰۹۵	۷۳٫۲۵٪

جدول ۱۹: نتیجه شبیه‌سازی مجموعه‌داده A108 با سیاست N-hit و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۱۰۶۳۰۱۷۲	
Read Hits	۱۰۳۹۹۸۲	۹٫۷۸٪
Read Misses	۹۵۹۰۱۹۰	۹۰٫۲۲٪
Write Requests	۹۵۷۵۱۴۵	
Write Hits	۲۴۲۸۲۹۰	۲۵٫۳۶٪
Write Misses	۷۱۴۶۸۵۵	۷۴٫۶۴٪
Total Requests	۲۰۲۰۵۳۱۷	
Total Hits	۳۴۶۸۲۷۲	۱۷٫۱۷٪
Total Misses	۱۶۷۳۷۰۴۵	۸۲٫۸۳٪

جدول ۲۰: نتیجه شبیه‌سازی مجموعه‌داده A42 با سیاست N-hit و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۳۰۹۸۷۳۴	
Read Hits	۱۸۰۳	۰٫۰۶٪
Read Misses	۳۰۹۶۹۲۹	۹۹٫۹۴٪
Write Requests	۱۹۸۹۱۹۷	
Write Hits	۹۴۱۳۱۳	۴۷٫۳۲٪
Write Misses	۱۰۴۷۸۸۳	۵۲٫۶۸٪
Total Requests	۵۰۸۷۹۳۱	
Total Hits	۹۴۳۱۱۸	۱۸٫۵۴٪
Total Misses	۴۱۴۴۸۱۳	۸۱٫۶۴٪

جدول ۲۱: نتیجه شبیه‌سازی مجموعه‌داده A669 با سیاست N-hit و اندازه حافظه نهان ۵۰۰۰۰

Metric	Count	Ratio
Read Requests	۲۵۹۵۹۱۹۵۶	
Read Hits	۲۵۳۸۰۹۰۰	۹۷٫۸۰٪
Read Misses	۵۷۱۰۵۶	۲٫۲۰٪
Write Requests	۱۲۹۲۸۷۶	
Write Hits	۹۶۱۵۱۳	۷۴٫۳۷٪
Write Misses	۳۳۱۳۶۳	۲۵٫۶۳٪
Total Requests	۲۷۲۴۴۸۳۲	
Total Hits	۲۶۳۴۲۴۱۳	۹۶٫۶۹٪
Total Misses	۹۰۲۴۱۹	۳٫۳۱٪

جدول ۲۲: نتیجه شبیه‌سازی مجموعه‌داده A129 با سیاست N-hit و اندازه حافظه نهان ۵۰۰۰۰

Metric	Count	Ratio
Read Requests	۴۸۶۸۱۷۲	
Read Hits	۱۰۷۶۱۵۷	۲۲٫۱۱٪
Read Misses	۳۷۹۲۰۱۵	۷۷٫۸۹٪
Write Requests	۱۲۸۴۴۳۲۰	
Write Hits	۷۱۹۲۵۲۷	۵۶٫۰۰٪
Write Misses	۵۶۵۱۷۹۳	۴۴٫۰۰٪
Total Requests	۱۷۷۷۱۲۴۹۲	
Total Hits	۸۲۶۸۶۸۴	۴۶٫۶۸٪
Total Misses	۹۴۴۳۸۰۸	۵۳٫۳۲٪

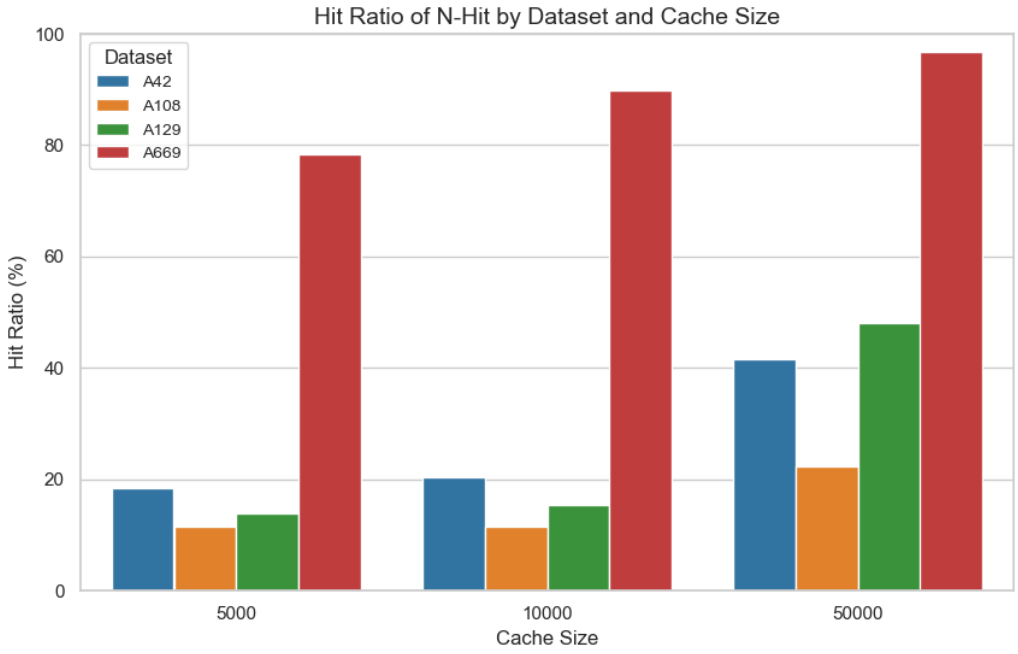
جدول ۲۳: نتیجه شبیه‌سازی مجموعه‌داده A108 با سیاست N-hit و اندازه حافظه نهان ۵۰۰۰۰

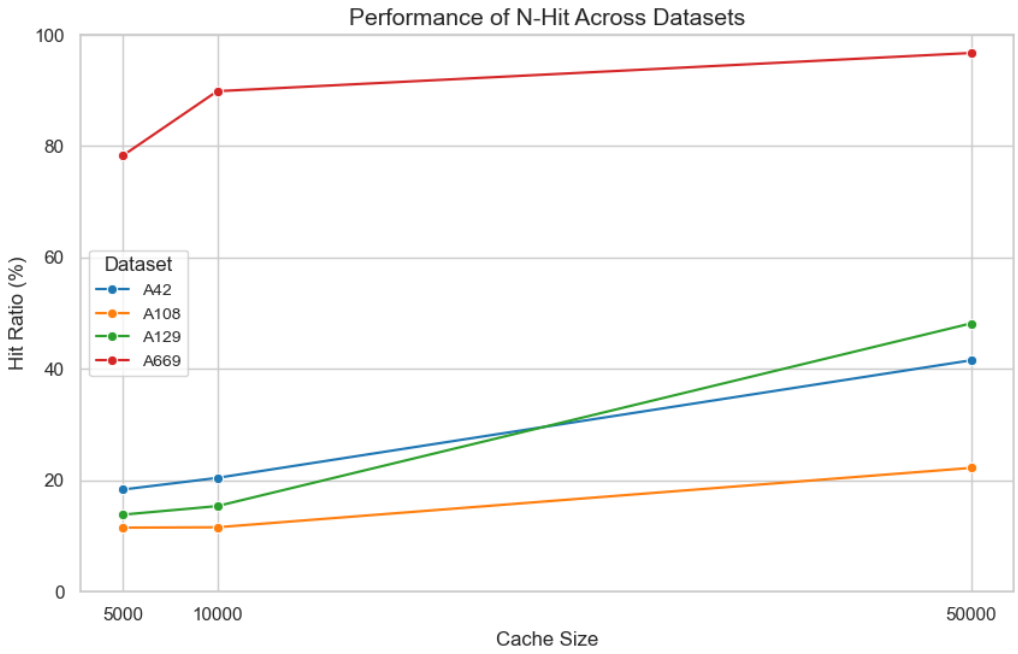
Metric	Count	Ratio
Read Requests	۱۰۶۳۰۱۷۲	
Read Hits	۱۳۵۲۱۰۹	۱۲٫۷۲٪
Read Misses	۹۲۷۸۰۶۳	۸۷٫۲۸٪
Write Requests	۹۵۷۵۱۴۵	
Write Hits	۳۰۸۹۳۸۱	۳۲٫۲۶٪
Write Misses	۶۴۸۵۷۶۴	۶۷٫۷۴٪
Total Requests	۲۰۲۰۵۳۱۷	
Total Hits	۴۴۴۱۴۹۰	۲۱٫۹۸٪
Total Misses	۱۵۷۶۳۸۲۷	۷۸٫۰۲٪

جدول ۲۴: نتیجه شبیه‌سازی مجموعه‌داده A42 با سیاست N-hit و اندازه حافظه نهان ۵۰۰۰۰

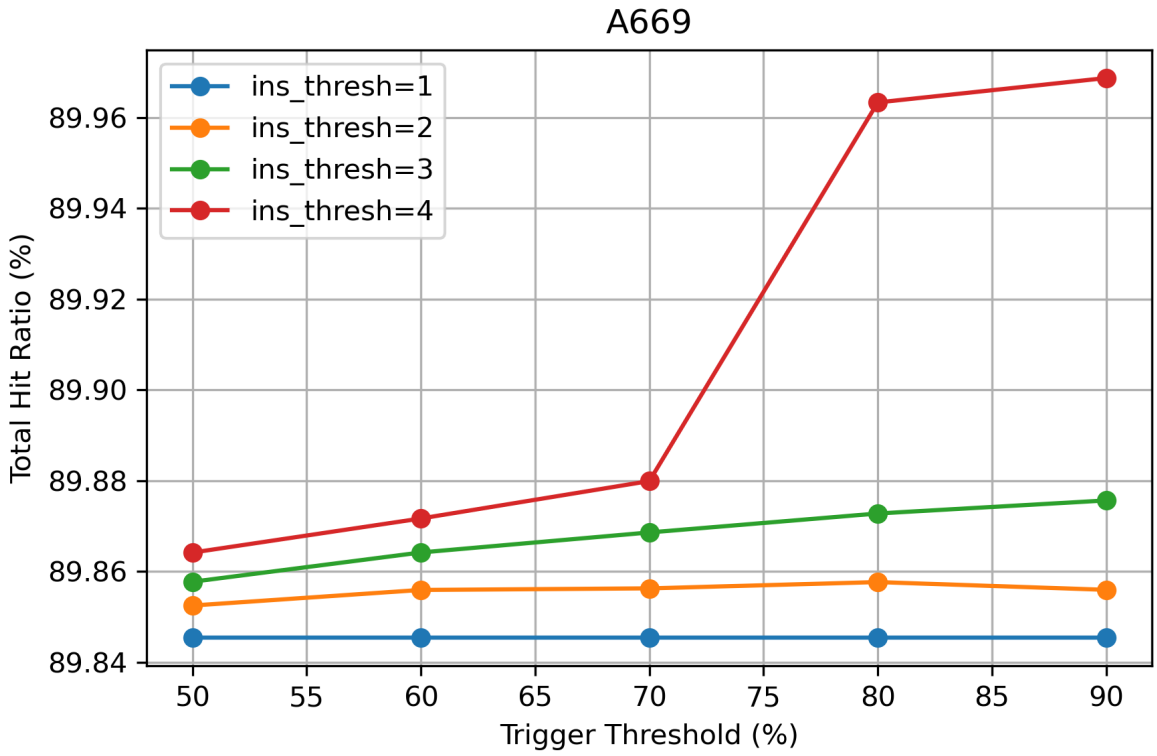
Metric	Count	Ratio
Read Requests	۳۰۹۸۷۳۴	
Read Hits	۵۲۱۴۳۳	۱۶٫۸۳٪
Read Misses	۲۵۷۷۳۰۱	۸۳٫۱۷٪
Write Requests	۱۹۸۹۱۹۷	
Write Hits	۱۰۴۰۲۸۰	۵۲٫۳۰٪
Write Misses	۹۴۸۹۱۷	۴۷٫۷۰٪
Total Requests	۵۰۸۷۹۳۱	
Total Hits	۱۵۶۱۷۱۳	۳۰٫۶۹٪
Total Misses	۳۵۲۶۲۱۸	۶۹٫۳۱٪

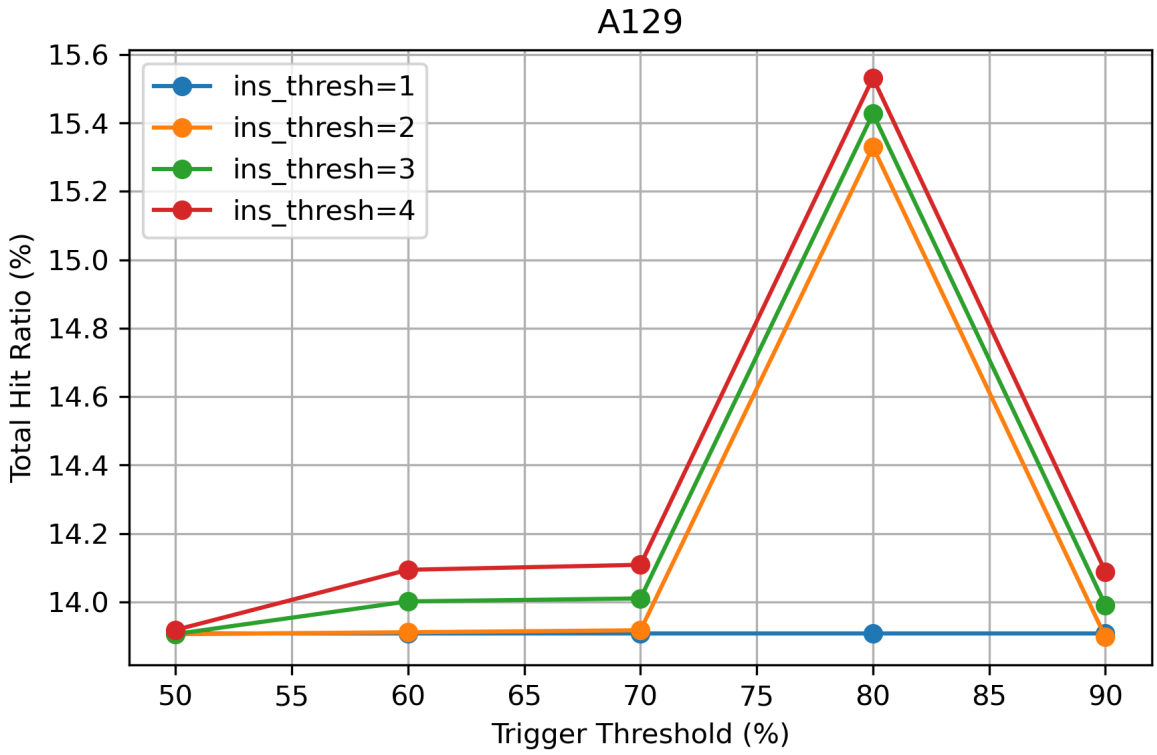
نمودارهای مربوط به این قسمت نیز در اینجا آورده شده است:

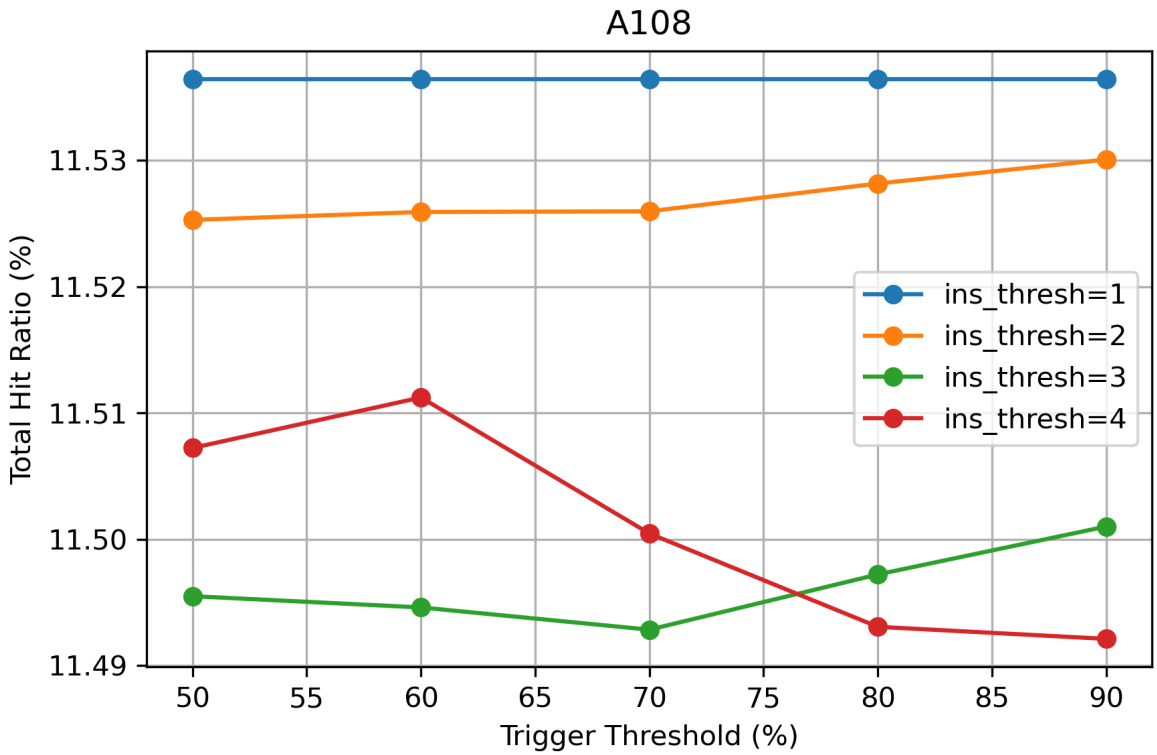


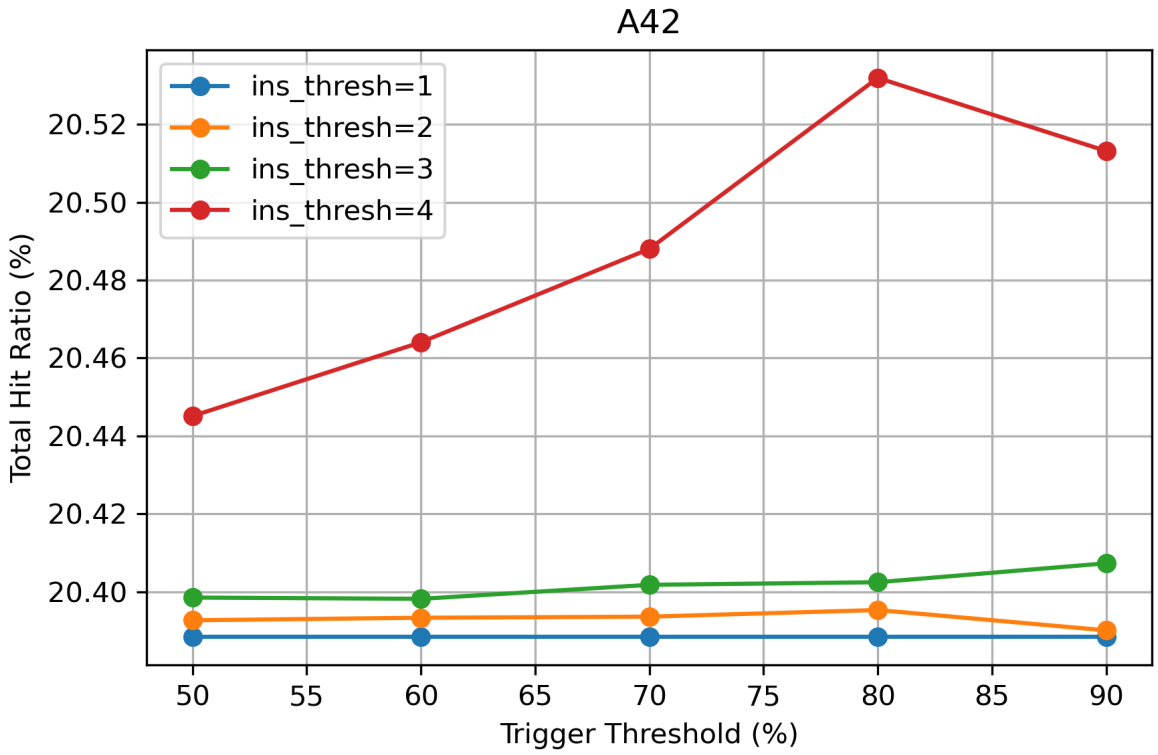


هم‌چنین نمودار مجزا برای هر یک از مجموعه‌های داده به شکل زیر است:




A129
ins_thresh=1
ins_thresh=2
ins_thresh=3
ins_thresh=4
Total Hit Ratio (%)
15.6
15.4
15.2
15.0
14.8
14.6
14.4
14.2
14.0
50
55
60
65
70
75
80
85
90
Trigger Threshold (%)


A108
ins_thresh=1
ins_thresh=2
ins_thresh=3
ins_thresh=4
Total Hit Ratio (%)
11.53
11.52
11.51
11.50
11.49
50
55
60
65
70
75
80
85
90
Trigger Threshold (%)



۴ سیاست ترکیبی LRU و N-hit

در این قسمت یک کد ترکیبی از دو سیاست قبل پیاده سازی کرده ایم.

۴-۱ توضیح برنامه شبیه‌سازی

در این کد، ترکیبی از دو سیاست جایگزینی حافظه نهان، یعنی LRU (Least Recently Used) و N-Hit برای مدیریت داده‌ها استفاده شده است. در ادامه ساختار کد و اجزای اصلی آن را توضیح می‌دهیم.

۴-۱-۱ ساختار کد و اجزای اصلی

این کد شامل دو کلاس اصلی LRUCache و NHitPolicy است که مسئول مدیریت حافظه نهان و سیاست ارتقای داده‌ها هستند. تابع simulate_nhit_lru داده‌ها را پردازش کرده و رفتار حافظه نهان را شبیه‌سازی می‌کند.

۴-۱-۲ توابع اصلی کد

ابتدا توابع مربوط به کلاس LRUCache را توضیح می‌دهیم.

- تابع is_present بررسی می‌کند که آیا یک داده در حافظه نهان موجود است یا نه.
- تابع access اگر داده در حافظه نهان باشد، آن را به انتهای OrderedDict منتقل می‌کند تا به‌عنوان اخیراً استفاده‌شده در نظر گرفته شود.
- تابع insert داده جدید را به حافظه نهان اضافه می‌کند و در صورت پر شدن ظرفیت، داده‌ای که کمترین استفاده را داشته است، حذف می‌شود.
- تابع occupancy تعداد آیتم‌های موجود در حافظه نهان را برمی‌گرداند.

در ادامه، توابع مربوط به کلاس NHitPolicy را بررسی می‌کنیم.

- تابع should_promote مشخص می‌کند که آیا یک داده باید به حافظه نهان منتقل شود یا خیر، با توجه به میزان دسترسی‌های قبلی به آن.
- تابع record_access تعداد دفعات دسترسی به هر داده را ذخیره کرده و یک صف برای پیگیری داده‌های تحت نظارت نگه می‌دارد.
- تابع remove_from_tracking داده‌هایی را که به حافظه نهان منتقل شده‌اند، از فهرست نظارت حذف می‌کند.

تابع اصلی simulate_nhit_lru وظیفه اجرای شبیه‌سازی را بر عهده دارد.

- داده‌های فایل csv ورودی را می‌خواند.
- به ازای هر داده، بررسی می‌کند که آیا در حافظه نهان موجود است یا خیر.
- در صورت وجود، داده را در LRU به‌روزرسانی می‌کند و در غیر این صورت، بسته به سیاست N-Hit تصمیم می‌گیرد که آیا باید در حافظه نهان قرار بگیرد یا خیر.
- در نهایت، آمار مربوط به تعداد hit و miss را محاسبه کرده و نتایج را نمایش می‌دهد.

با استفاده از تابع main، این شبیه‌سازی برای چندین فایل مختلف اجرا می‌شود.

۲-۴ نتایج عملکرد

جدول ۲۵: نتیجه شبیه‌سازی مجموعه‌داده A669 با سیاست LRU / N-hit و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۲۵۹۵۹۱۹۵۶	
Read Hits	۲۴۴۴۸۰۶۴	۹۴٫۲۱٪
Write Requests	۱۲۹۲۸۷۶	
Write Hits	۶۹۵۰۲۵	۵۳٫۵۳٪
Total Requests	۲۷۲۴۴۸۳۲	
Total Hits	۲۵۱۴۰۰۸۹	۹۲٫۲۷٪
Total Misses	۲۱۰۴۷۴۳	۷٫۷۳٪

جدول ۲۶: نتیجه شبیه‌سازی مجموعه‌داده A129 با سیاست LRU / N-hit و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۴۸۶۸۱۷۲	
Read Hits	۷۱۰۰۲۵	۱۴٫۵۹٪
Write Requests	۱۲۸۴۴۳۲۰	
Write Hits	۴۰۷۳۷۰۸	۳۱٫۷۲٪
Total Requests	۱۷۷۷۱۲۴۹۲	
Total Hits	۴۷۸۳۷۳۳	۲۷٫۰۱٪
Total Misses	۱۲۹۲۸۷۵۹	۷۲٫۹۹٪

جدول ۲۷: نتیجه شبیه‌سازی مجموعه‌داده A108 با سیاست LRU / N-hit و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۱۰۶۳۰۱۷۲	
Read Hits	۷۱۹۲۸۸	۶٫۷۷٪
Write Requests	۹۵۷۵۱۴۵	
Write Hits	۲۲۷۶۱۰۱	۲۳٫۷۷٪
Total Requests	۲۰۲۰۵۳۱۷	
Total Hits	۲۹۹۵۳۸۹	۱۴٫۸۲٪
Total Misses	۱۷۲۰۹۹۲۸	۸۵٫۱۸٪

جدول ۲۸: نتیجه شبیه‌سازی مجموعه‌داده A42 با سیاست LRU / N-hit و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۳۰۹۸۷۳۴	
Read Hits	۳۷۰۵۲۴	۱۱٫۹۶٪
Write Requests	۱۹۸۹۱۹۷	
Write Hits	۱۰۳۲۹۳۲	۵۱٫۹۳٪
Total Requests	۵۰۸۷۹۳۱	
Total Hits	۱۴۰۳۴۵۶	۲۷٫۵۸٪
Total Misses	۳۶۸۴۴۷۵	۷۲٫۴۲٪

۵ سیاست جایگزینی بهینه (Belady/Oracle)

سیاست جایگزینی Belady/Oracle، یکی از الگوریتم‌های مرجع در مدیریت حافظه نهان است که در آن تصمیم‌گیری برای جایگزینی بلوک‌های حافظه به گونه‌ای انجام می‌شود که عملکرد بهینه تضمین شود. این سیاست به این صورت عمل می‌کند که در هر لحظه، بلوکی را که دیرتر از بقیه در آینده استفاده خواهد شد، از حافظه نهان حذف می‌کند. هدف اصلی این سیاست، به حداقل رساندن تعداد Cache Miss از طریق انتخاب هوشمندانه صفحاتی است که باید از حافظه نهان حذف شوند. این روش به دلیل استفاده از اطلاعات کامل درخواست‌های آینده، تضمین می‌کند که نرخ Cache Hit بالاترین مقدار ممکن باشد.

با این حال، این سیاست به دلیل ماهیت پیش‌بینانه‌اش (نیاز به دانستن تمامی درخواست‌های آینده)، تنها در تحلیل‌های برون‌خط (offline) قابل استفاده است و امکان پیاده‌سازی عملی آن در محیط‌های واقعی وجود ندارد. در این پروژه، ما از این الگوریتم به‌عنوان مبنایی برای مقایسه عملکرد سایر سیاست‌ها بهره خواهیم گرفت.

۵-۱ شرح برنامه

کد ارائه‌شده، شبیه‌سازی سیاست Belady/Oracle را با استفاده از توابع و ساختارهای مناسب پیاده‌سازی کرده است. مراحل کلیدی در کد عبارتند از:

۱. پیش‌پردازش رخدادهای آینده: تابع `preprocess_future_occurrences` برای هر صفحه در توالی درخواست‌ها، فهرست زمان‌های استفاده بعدی را ایجاد می‌کند. این اطلاعات برای تصمیم‌گیری در مورد حذف صفحات استفاده می‌شود.

۲. استفاده از ساختار `multiset`: الگوریتم از یک `multiset` برای نگهداری صفحات در حافظه نهان استفاده می‌کند. این ساختار داده به‌گونه‌ای پیاده‌سازی شده که صفحات بر اساس زمان استفاده بعدی مرتب باشند. این ویژگی امکان حذف سریع صفحه‌ای که دیرتر از بقیه استفاده می‌شود را فراهم می‌کند.

۳. مدیریت حافظه نهان:

هنگام ورود صفحه جدید، اگر حافظه نهان پر باشد، صفحه‌ای که در آینده دیرتر استفاده می‌شود حذف می‌گردد. اما در صورت وجود فضای خالی، صفحه جدید مستقیماً به حافظه نهان اضافه می‌شود.

۵-۲ نتایج عملکرد

جدول ۲۹: نتیجه شبیه‌سازی مجموعه‌داده A669 با سیاست Belady و اندازه حافظه نهان ۵۰۰۰

Metric	Count	Ratio
Read Requests	۲۵۹۵۱۹۵۶	
Read Hits	۲۳۴۲۸۷۷۹	۹۰٫۲۸٪
Read Misses	۲۵۲۳۱۷۷	۹٫۷۲٪
Write Requests	۱۲۹۲۸۷۷	
Write Hits	۷۴۳۴۷۴	۵۷٫۵۱٪
Write Misses	۵۴۹۴۰۳	۴۲٫۴۹٪
Total Requests	۲۷۲۴۴۸۳۳	
Total Hits	۳۰۷۲۵۸۰	۸۸٫۷۲٪
Total Misses	۱۷۹۰۳۱	۱۱٫۲۸٪

جدول ۳۰: نتیجه شبیه‌سازی مجموعه‌داده A129 با سیاست Belady و اندازه حافظه نهان ۵۰۰۰

Metric	Count	Ratio
Read Requests	۴۸۶۸۱۷۳	
Read Hits	۹۷۷۷۷۵	۲۰٫۰۹٪
Read Misses	۳۸۹۰۳۹۸	۷۹٫۹۱٪
Write Requests	۱۲۸۴۴۳۲۰	
Write Hits	۵۱۴۲۰۰۳	۴۰٫۰۳٪
Write Misses	۷۷۰۲۳۱۷	۵۹٫۹۷٪
Total Requests	۱۷۷۷۱۲۴۹۳	
Total Hits	۶۱۱۹۷۷۸	۳۴٫۵۵٪
Total Misses	۲۹۹۹۱۹۷	۶۵٫۴۵٪

جدول ۳۱: نتیجه شبیه‌سازی مجموعه‌داده A108 با سیاست Belady و اندازه حافظه نهان ۵۰۰۰

Metric	Count	Ratio
Read Requests	۱۰۶۳۰۱۷۳	
Read Hits	۱۴۱۰۴۷۱	۱۳٫۲۷٪
Read Misses	۸۲۱۹۷۰۲	۸۶٫۷۳٪
Write Requests	۹۵۷۵۱۴۵	
Write Hits	۳۱۰۷۰۰۲	۳۲٫۴۵٪
Write Misses	۶۴۶۸۱۴۳	۶۷٫۵۵٪
Total Requests	۲۰۲۰۵۳۱۸	
Total Hits	۴۵۱۷۴۷۳	۲۲٫۳۶٪
Total Misses	۱۵۶۸۷۸۴۵	۷۷٫۶۴٪

جدول ۳۲: نتیجه شبیه‌سازی مجموعه‌داده A42 با سیاست Belady و اندازه حافظه نهان ۵۰۰۰

Metric	Count	Ratio
Read Requests	۳۰۹۸۷۳۵	
Read Hits	۳۴۸۲۵۱	۱۱٫۲۴٪
Read Misses	۲۷۵۰۴۸۴	۸۸٫۷۴٪
Write Requests	۱۹۸۹۱۹۷	
Write Hits	۱۰۳۹۷۸۵	۵۲٫۲۷٪
Write Misses	۹۴۹۴۱۲	۴۷٫۷۳٪
Total Requests	۵۰۸۷۹۳۲	
Total Hits	۱۳۸۸۰۳۶	۲۷٫۲۸٪
Total Misses	۳۶۹۹۸۹۶	۷۲٫۷۲٪

جدول ۳۳: نتیجه شبیه‌سازی مجموعه‌داده A669 با سیاست Belady و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۲۵۹۵۱۹۵۶	
Read Hits	۲۴۹۷۴۶۱۸	۹۶٫۲۳٪
Read Misses	۹۷۷۷۳۳۸	۳٫۷۷٪
Write Requests	۱۲۹۲۸۷۷	
Write Hits	۸۰۰۹۵۴	۶۱٫۹۵٪
Write Misses	۴۹۱۹۲۳	۳۸٫۰۵٪
Total Requests	۲۷۲۴۴۸۳۳	
Total Hits	۲۵۷۷۵۵۷۲	۹۴٫۶۱٪
Total Misses	۱۴۶۹۲۶۱	۵٫۳۹٪

جدول ۳۴: نتیجه شبیه‌سازی مجموعه‌داده A129 با سیاست Belady و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۴۸۶۸۱۷۳	
Read Hits	۱۰۴۳۴۲۱	۲۱٫۴۳٪
Read Misses	۳۸۲۴۷۵۲	۷۸٫۵۷٪
Write Requests	۱۲۸۴۴۳۲۰	
Write Hits	۶۴۳۱۷۱۳	۵۰٫۰۷٪
Write Misses	۶۴۱۲۶۰۷	۴۹٫۹۳٪
Total Requests	۱۷۷۷۱۲۴۹۳	
Total Hits	۷۴۷۵۱۳۴	۴۲٫۲۰٪
Total Misses	۱۰۲۳۷۳۵۹	۵۷٫۸۰٪

جدول ۳۵: نتیجه شبیه‌سازی مجموعه‌داده A108 با سیاست Belady و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۱۰۶۳۰۱۷۳	
Read Hits	۱۶۳۵۶۷۶	۱۵٫۳۹٪
Read Misses	۸۹۹۴۴۹۷	۸۴٫۶۱٪
Write Requests	۹۵۷۵۱۴۵	
Write Hits	۳۵۲۳۶۵۱	۳۶٫۸۰٪
Write Misses	۶۰۵۱۴۹۴	۶۳٫۲۰٪
Total Requests	۲۰۲۰۵۳۱۸	
Total Hits	۵۱۵۹۳۲۷	۲۵٫۵۳٪
Total Misses	۱۶۵۸۵۳۰	۷۴٫۴۷٪

جدول ۳۶: نتیجه شبیه‌سازی مجموعه‌داده A42 با سیاست Belady و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۳۰۹۸۷۳۵	
Read Hits	۶۷۲۰۱۲	۲۱٫۶۹٪
Read Misses	۲۴۲۶۷۲۳	۷۸٫۳۱٪
Write Requests	۱۹۸۹۱۹۷	
Write Hits	۱۰۴۶۴۶۹	۵۲٫۶۱٪
Write Misses	۹۴۲۷۲۸	۴۷٫۳۹٪
Total Requests	۵۰۸۷۹۳۲	
Total Hits	۱۷۱۸۴۸۱	۳۳٫۷۸٪
Total Misses	۳۳۶۹۴۵۱	۶۶٫۲۲٪

جدول ۳۷: نتیجه شبیه‌سازی مجموعه‌داده A669 با سیاست Belady و اندازه حافظه نهان ۵۰۰۰۰

Metric	Count	Ratio
Read Requests	۲۵۹۵۱۹۵۶	
Read Hits	۲۵۴۷۸۸۴۱	۹۸٫۱۸٪
Read Misses	۴۷۳۱۱۵	۱٫۸۲٪
Write Requests	۱۲۹۲۸۷۷	
Write Hits	۱۱۸۲۶۹۲	۹۱٫۴۸٪
Write Misses	۱۱۰۱۸۵	۸٫۵۳٪
Total Requests	۲۷۲۴۴۸۳۳	
Total Hits	۲۶۶۶۱۵۳۳	۹۷٫۸۶٪
Total Misses	۱۷۹۰۳۱	۲٫۱۴٪

جدول ۳۸: نتیجه شبیه‌سازی مجموعه‌داده A129 با سیاست Belady و اندازه حافظه نهان ۵۰۰۰۰

Metric	Count	Ratio
Read Requests	۴۸۶۸۱۷۳	
Read Hits	۱۷۵۴۶۱۵	۳۶٫۰۴٪
Read Misses	۳۱۱۳۵۵۸	۶۳٫۹۶٪
Write Requests	۱۲۸۴۴۳۲۰	
Write Hits	۸۷۶۷۴۳۱	۶۸٫۲۶٪
Write Misses	۴۰۷۶۸۸۹	۳۱٫۷۴٪
Total Requests	۱۷۷۷۱۲۴۹۳	
Total Hits	۱۰۵۲۲۰۴۶	۵۹٫۴۰٪
Total Misses	۲۹۹۹۱۹۷	۴۰٫۶۰٪

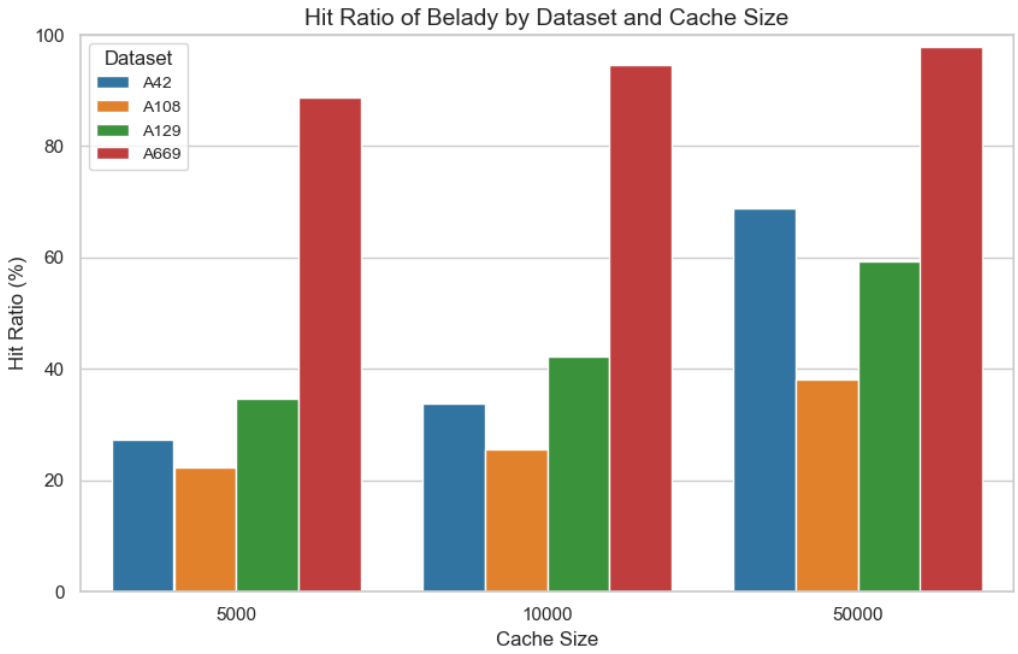
جدول ۳۹: نتیجه شبیه‌سازی مجموعه‌داده A108 با سیاست Belady و اندازه حافظه نهان ۵۰۰۰۰

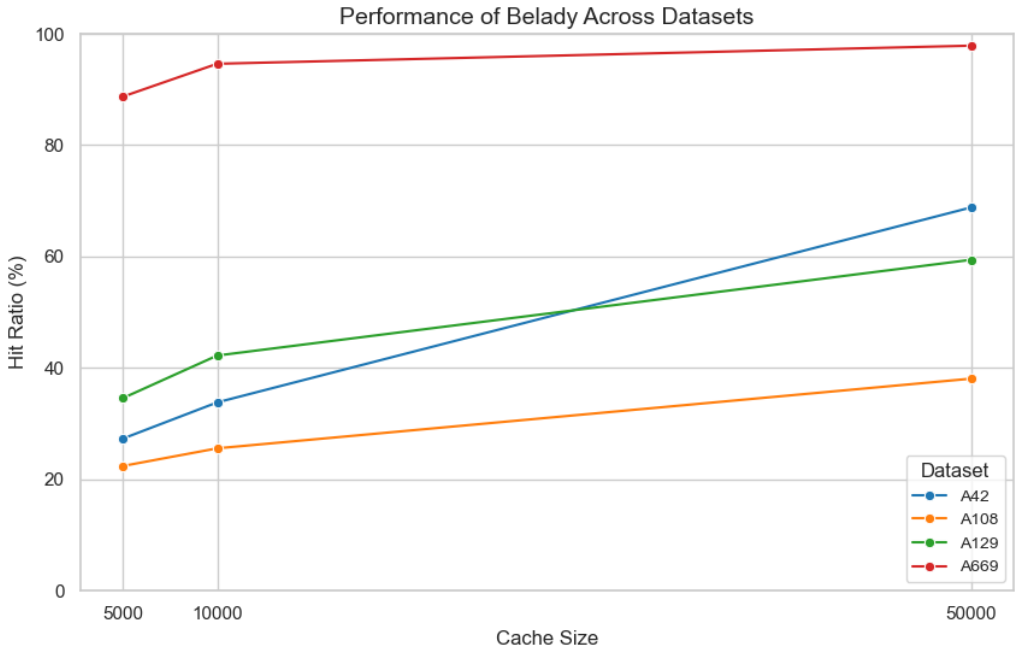
Metric	Count	Ratio
Read Requests	۱۰۶۳۰۱۷۳	
Read Hits	۲۸۲۷۵۶۱	۲۶٫۶۰٪
Read Misses	۷۸۰۲۶۱۲	۷۳٫۴۰٪
Write Requests	۹۵۷۵۱۴۵	
Write Hits	۴۸۵۹۹۱۶	۵۰٫۷۶٪
Write Misses	۴۷۱۵۲۲۹	۴۹٫۲۴٪
Total Requests	۲۰۲۰۵۳۱۸	
Total Hits	۷۶۸۷۴۷۷	۳۸٫۰۵٪
Total Misses	۱۲۵۱۷۸۴۱	۶۱٫۹۵٪

جدول ۴۰: نتیجه شبیه‌سازی مجموعه‌داده A42 با سیاست Belady و اندازه حافظه نهان ۵۰۰۰۰

Metric	Count	Ratio
Read Requests	۳۰۹۸۷۳۵	
Read Hits	۲۳۶۶۳۸۰	۷۶٫۳۷٪
Read Misses	۷۳۲۳۵۵	۲۳٫۶۳٪
Write Requests	۱۹۸۹۱۹۷	
Write Hits	۱۱۳۴۷۷۶	۵۷٫۰۵٪
Write Misses	۸۵۴۴۲۱	۴۲٫۹۵٪
Total Requests	۵۰۸۷۹۳۲	
Total Hits	۳۵۰۱۱۵۶	۶۸٫۸۱٪
Total Misses	۱۵۸۶۷۷۶	۳۱٫۱۹٪

نمودارهای مربوط به این قسمت نیز در اینجا آورده شده است:





۶ سیاست ARC

الگوریتم ARC (Adaptive Replacement Cache) یکی از سیاست‌های جایگزینی پیشرفته برای مدیریت حافظه نهان است که به صورت پویا میان داده‌های پرتکرار و داده‌هایی که اخیراً دسترسی

یافته‌اند تعادل برقرار می‌کند. این الگوریتم از دو لیست اصلی برای مدیریت بلوک‌ها استفاده می‌کند: یکی برای داده‌هایی که در حافظه نهان قرار دارند (T) و دیگری برای داده‌هایی که اخیراً از حافظه نهان حذف شده‌اند (B). هر یک از این لیست‌ها خود به دو بخش تقسیم می‌شوند: T1 و B1 مربوط به داده‌هایی که اخیراً به آن‌ها دسترسی پیدا کرده‌ایم، و T2 و B2 مربوط به داده‌هایی که به دفعات بیشتری دسترسی داشته‌اند. این ساختار به الگوریتم اجازه می‌دهد تا به طور خودکار بین داده‌های پرتکرار و داده‌های اخیراً استفاده‌شده تعادل برقرار کند.

زمانی که یک بلوک داده درخواست می‌شود، ابتدا بررسی می‌شود که آیا در یکی از بخش‌های T1 یا T2 قرار دارد. در صورت وجود (Cache Hit)، این بلوک به ابتدای بخش مربوطه منتقل می‌شود تا نشان‌دهنده دسترسی مجدد به آن باشد. اگر بلوک در بخش‌های B1 یا B2 قرار داشته باشد (یعنی در گذشته حذف شده اما همچنان به صورت شبح ذخیره شده است)، بلوک به حافظه نهان بازگردانده شده و به یکی از بخش‌های T1 یا T2 منتقل می‌شود. در این حالت، متغیر P که نشان‌دهنده اندازه ایده‌آل بخش T1 است، به‌روزرسانی می‌شود. اگر بلوک درخواست‌شده در هیچ‌یک از بخش‌های T1، T2، B1 یا B2 موجود نباشد، به بخش T1 اضافه می‌شود. در صورتی که مجموع اندازه بخش‌های T1 و T2 از ظرفیت حافظه نهان بیشتر شود، الگوریتم با توجه به مقدار P تصمیم می‌گیرد که کدام بلوک باید حذف شود.

از ویژگی‌های برجسته الگوریتم ARC می‌توان به توانایی آن در تنظیم پویا اندازه بخش‌های T1 و T2 برای انطباق با تغییرات بارکاری اشاره کرد. این قابلیت به الگوریتم اجازه می‌دهد که به طور هم‌زمان داده‌های پرتکرار و داده‌های با دسترسی اخیر را به صورت بهینه مدیریت کند. با این حال، مدیریت چهار بخش و به‌روزرسانی مداوم آن‌ها ممکن است باعث افزایش سربار محاسباتی شود.

توضیح برنامه شبیه‌سازی ARC

کد شبیه‌سازی سیاست ARC (Adaptive Replacement Cache) را به زبان پایتون نوشتیم. این کد در فایل `ARC.py` در پیوست‌ها قرار گرفته است. در ادامه بخش‌های این کد را توضیح می‌دهیم.

ساختار کد و اجزای اصلی

کد شبیه‌سازی از چندین تابع و متغیر برای پیاده‌سازی الگوریتم ARC استفاده می‌کند که توضیح آن‌ها در ادامه آمده:

۱. متغیرهای اصلی

T1 و T2: این دو لیست OrderedDict نماینده حافظه نهان (*cache*) هستند. T1 شامل داده‌هایی است که اخیراً به آن‌ها دسترسی پیدا شده و T2 داده‌هایی را ذخیره می‌کند که به دفعات بیشتری مورد دسترسی قرار گرفته‌اند.

B1 و B2: این دو لیست شامل داده‌هایی هستند که قبلاً از حافظه نهان حذف شده‌اند اما به عنوان شبح (ghost entries) نگهداری می‌شوند.

p: متغیری که نشان‌دهنده اندازه ایده‌آل T1 است و به طور پویا با توجه به رفتار الگوریتم تنظیم می‌شود.

۲. تابع run_arc_python

این تابع هسته اصلی شبیه‌سازی ARC است و از مراحل زیر تشکیل شده است:

بررسی Cache Hit: اگر صفحه موردنظر در T1 یا T2 باشد، به عنوان Cache Hit محسوب شده و صفحه به ابتدای بخش مربوطه منتقل می‌شود.

بررسی Cache Miss: اگر صفحه در B1 باشد، مقدار p افزایش یافته و صفحه به T2 منتقل می‌شود. اگر صفحه در B2 باشد، مقدار p کاهش یافته و صفحه به T2 منتقل می‌شود. اگر صفحه در هیچ‌یک از لیست‌ها نباشد، به T1 اضافه می‌شود.

حذف صفحات اضافی: در صورتی که تعداد صفحات T1 و T2 از ظرفیت حافظه نهان بیشتر شود، صفحه‌ای که در آینده دیرتر مورد استفاده قرار می‌گیرد حذف و به B1 یا B2 منتقل می‌شود.

۳. سایر توابع

read_csv_first_column: این تابع داده‌ها را از فایل CSV خوانده و بر اساس بازه زمانی مشخص فیلتر می‌کند. داده‌های استخراج‌شده به عنوان توالی صفحات به الگوریتم ARC ارسال می‌شوند.

run_all_policies: این تابع سیاست‌های مختلف مدیریت حافظه نهان (از جمله ARC) را روی مجموعه داده ورودی اجرا کرده و نتایج را جمع‌آوری می‌کند.

نتایج عملکرد

در آخر خروجی برنامه مقادیر Cache Hits Cache Misses و است.

همچنین اندازه نهایی T_1 ، T_2 ، B_1 و B_2 را چاپ می‌کنیم.

خروجی برنامه

نتایج شبیه‌سازی برای اندازه حافظه نهان ۵۰۰۰، برای هر مجموعه‌داده مختلف در زیر آمده.

جدول ۴۱: نتیجه شبیه‌سازی مجموعه‌داده ۶۶۹A با سیاست ARC و اندازه حافظه نهان ۵۰۰۰

Metric	Count	Ratio
Read Requests	۲۵۹۵۱۹۵۶	
Read Hits	۲۱۱۷۵۲۲۲	۸۱٫۵۹٪
Read Misses	۴۷۷۶۷۳۴	۱۸٫۴۱٪
Write Requests	۱۲۹۲۸۷۷	
Write Hits	۶۵۶۶۶۴	۵۰٫۷۹٪
Write Misses	۶۳۶۲۱۳	۴۹٫۲۱٪
Total Requests	۲۷۲۴۴۸۳۳	
Total Hits	۲۱۸۳۱۸۸۶	۸۰٫۱۳٪
Total Misses	۵۴۱۲۹۴۷	۱۹٫۸۷٪

جدول ۴۲: نتیجه شبیه‌سازی مجموعه‌داده ۱۲۹A با سیاست ARC و اندازه حافظه نهان ۵۰۰۰

Metric	Count	Ratio
Read Requests	۴۸۶۸۱۷۳	
Read Hits	۷۷۵۷۹۳	۱۵٫۹۴٪
Read Misses	۴۰۹۲۳۸۰	۸۴٫۰۶٪
Write Requests	۱۲۸۴۴۳۲۰	
Write Hits	۳۸۵۴۲۱۰	۳۰٫۰۱٪
Write Misses	۸۹۹۰۱۱۰	۶۹٫۹۹٪
Total Requests	۱۷۷۱۲۴۹۳	
Total Hits	۴۶۳۰۰۰۳	۲۶٫۱۴٪
Total Misses	۱۳۰۸۲۴۹۰	۷۳٫۸۶٪

جدول ۴۳: نتیجه شبیه‌سازی مجموعه‌داده ۱۰۸A با سیاست ARC و اندازه حافظه نهان ۵۰۰۰

Metric	Count	Ratio
Read Requests	۱۰۶۳۰۱۷۳	
Read Hits	۹۴۱۶۶۷	۸٫۸۶٪
Read Misses	۹۶۸۸۵۰۶	۹۱٫۱۴٪
Write Requests	۹۵۷۵۱۴۵	
Write Hits	۲۴۰۶۴۶۶	۲۵٫۱۳٪
Write Misses	۷۱۶۸۶۷۹	۷۴٫۸۷٪
Total Requests	۲۰۲۰۵۳۱۸	
Total Hits	۳۳۴۸۱۳۳	۱۶٫۵۷٪
Total Misses	۱۶۸۵۷۱۸۵	۸۳٫۴۳٪

جدول ۴۴: نتیجه شبیه‌سازی مجموعه‌داده ۴۲A با سیاست ARC و اندازه حافظه نهان ۵۰۰۰

Metric	Count	Ratio
Read Requests	۳۰۹۸۷۳۵	
Read Hits	۱۲۷۸۰۶	۴٫۱۲٪
Read Misses	۲۹۷۰۹۲۹	۹۵٫۸۸٪
Write Requests	۱۹۸۹۱۹۷	
Write Hits	۱۰۲۷۰۷۵	۵۱٫۶۳٪
Write Misses	۹۶۲۱۲۲	۴۸٫۳۷٪
Total Requests	۵۰۸۷۹۳۲	
Total Hits	۱۱۵۴۸۸۱	۲۲٫۷۰٪
Total Misses	۳۹۳۳۰۵۱	۷۷٫۳۰٪

نتایج شبیه‌سازی برای اندازه حافظه نهان ۱۰۰۰۰، برای هر مجموعه‌داده مختلف در زیر آمده.

جدول ۴۵: نتیجه شبیه‌سازی مجموعه‌داده A۶۶۹ با سیاست ARC و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۲۵۹۵۹۱۹۵۶	
Read Hits	۲۴۳۶۵۸۱۷	۹۳٫۸۹٪
Read Misses	۱۵۸۶۱۳۹	۶٫۱۱٪
Write Requests	۱۲۹۲۸۷۷	
Write Hits	۶۸۳۷۲۷	۵۲٫۸۸٪
Write Misses	۶۰۹۱۵۰	۴۷٫۱۲٪
Total Requests	۲۷۲۴۴۸۳۳	
Total Hits	۲۵۰۴۹۵۴۴	۹۱٫۹۴٪
Total Misses	۲۱۹۵۲۸۹	۸٫۰۶٪

جدول ۴۶: نتیجه شبیه‌سازی مجموعه‌داده A۱۲۹ با سیاست ARC و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۴۸۶۸۱۷۳	
Read Hits	۹۳۰۴۲۵	۱۹٫۱۱٪
Read Misses	۳۹۳۷۷۴۸	۸۰٫۸۹٪
Write Requests	۱۲۸۴۴۳۲۰	
Write Hits	۴۲۱۰۳۶۴	۳۲٫۷۸٪
Write Misses	۸۶۳۳۹۵۶	۶۷٫۲۲٪
Total Requests	۱۷۷۷۱۲۴۹۳	
Total Hits	۵۱۴۰۷۸۹	۲۹٫۰۲٪
Total Misses	۱۲۵۷۱۷۰۴	۷۰٫۹۸٪

جدول ۴۷: نتیجه شبیه‌سازی مجموعه‌داده A۱۰۸ با سیاست ARC و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۱۰۶۳۰۱۷۳	
Read Hits	۱۱۲۹۰۱۳	۱۰٫۶۲٪
Read Misses	۹۵۰۱۱۶۰	۸۹٫۳۸٪
Write Requests	۹۵۷۵۱۴۵	
Write Hits	۲۵۲۵۷۷۳	۲۶٫۳۸٪
Write Misses	۷۰۴۹۳۷۲	۷۳٫۶۲٪
Total Requests	۲۰۲۰۵۳۱۸	
Total Hits	۳۶۵۴۷۸۶	۱۸٫۰۹٪
Total Misses	۱۶۵۵۰۵۳۲	۸۱٫۹۱٪

جدول ۴۸: نتیجه شبیه‌سازی مجموعه‌داده A۴۲ با سیاست ARC و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۳۰۹۸۷۳۵	
Read Hits	۲۲۴۸۵۶	۷٫۲۶٪
Read Misses	۲۸۷۳۸۷۹	۹۲٫۷۴٪
Write Requests	۱۹۸۹۱۹۷	
Write Hits	۱۰۶۹۳۲۵	۵۳٫۷۶٪
Write Misses	۹۱۹۸۷۲	۴۶٫۲۴٪
Total Requests	۵۰۸۷۹۳۲	
Total Hits	۱۲۹۴۱۸۱	۲۵٫۴۴٪
Total Misses	۳۷۹۳۷۵۱	۷۴٫۵۶٪

نتایج شبیه‌سازی برای اندازه حافظه نهان ۵۰۰۰۰، برای هر مجموعه‌داده مختلف در زیر آمده.

جدول ۴۹: نتیجه شبیه‌سازی مجموعه‌داده A۶۶۹ با سیاست ARC و اندازه حافظه نهان ۵۰۰۰۰

Metric	Count	Ratio
Read Requests	۲۵۹۵۱۹۵۶	
Read Hits	۲۵۳۸۱۴۹۱	۹۷٫۸۰٪
Read Misses	۵۷۰۴۶۵	۲٫۲۰٪
Write Requests	۱۲۹۲۸۷۷	
Write Hits	۱۰۳۸۷۰۹۸	۸۰٫۲۲٪
Write Misses	۲۵۵۷۷۹	۱۹٫۷۸٪
Total Requests	۲۷۲۴۴۸۳۳	
Total Hits	۲۶۴۱۸۵۸۹	۹۶٫۹۷٪
Total Misses	۸۲۶۲۴۴	۳٫۰۳٪

جدول ۵۰: نتیجه شبیه‌سازی مجموعه‌داده A۱۲۹ با سیاست ARC و اندازه حافظه نهان ۵۰۰۰۰

Metric	Count	Ratio
Read Requests	۴۸۶۸۱۷۳	
Read Hits	۱۱۳۹۴۳۸	۲۳٫۴۱٪
Read Misses	۳۷۲۸۷۳۵	۷۶٫۵۹٪
Write Requests	۱۲۸۴۴۳۲۰	
Write Hits	۸۱۵۷۱۱۰	۶۳٫۵۱٪
Write Misses	۴۶۸۷۲۱۰	۳۶٫۴۹٪
Total Requests	۱۷۷۱۲۴۹۳	
Total Hits	۹۲۹۶۵۴۸	۵۲٫۴۹٪
Total Misses	۸۴۱۵۹۴۵	۴۷٫۵۱٪

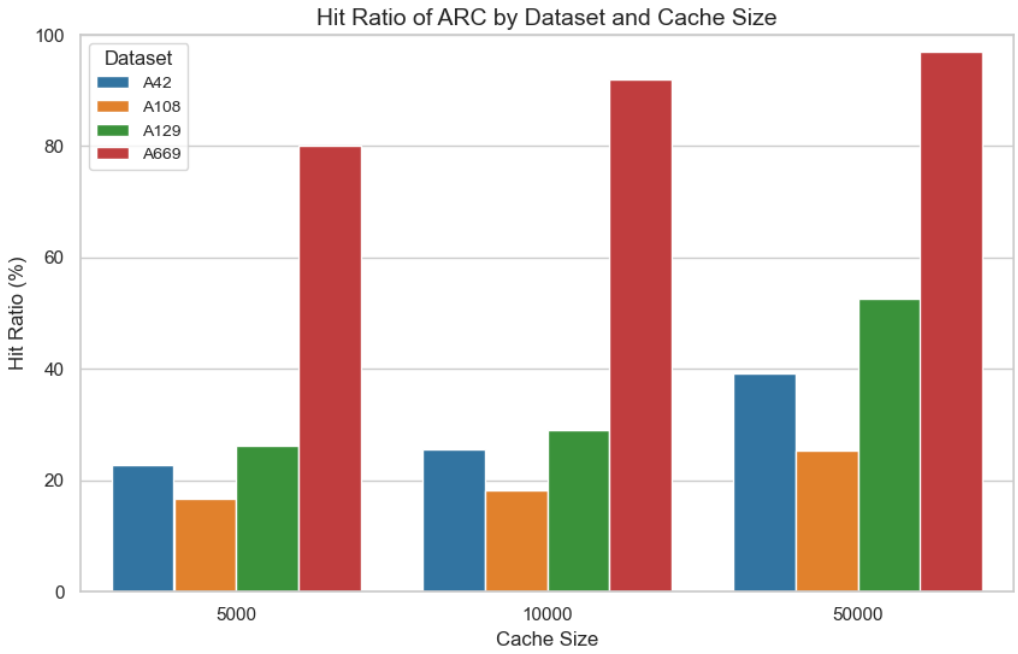
جدول ۵۱: نتیجه شبیه‌سازی مجموعه‌داده ۱۰۸A با سیاست ARC و اندازه حافظه نهان ۵۰۰۰۰

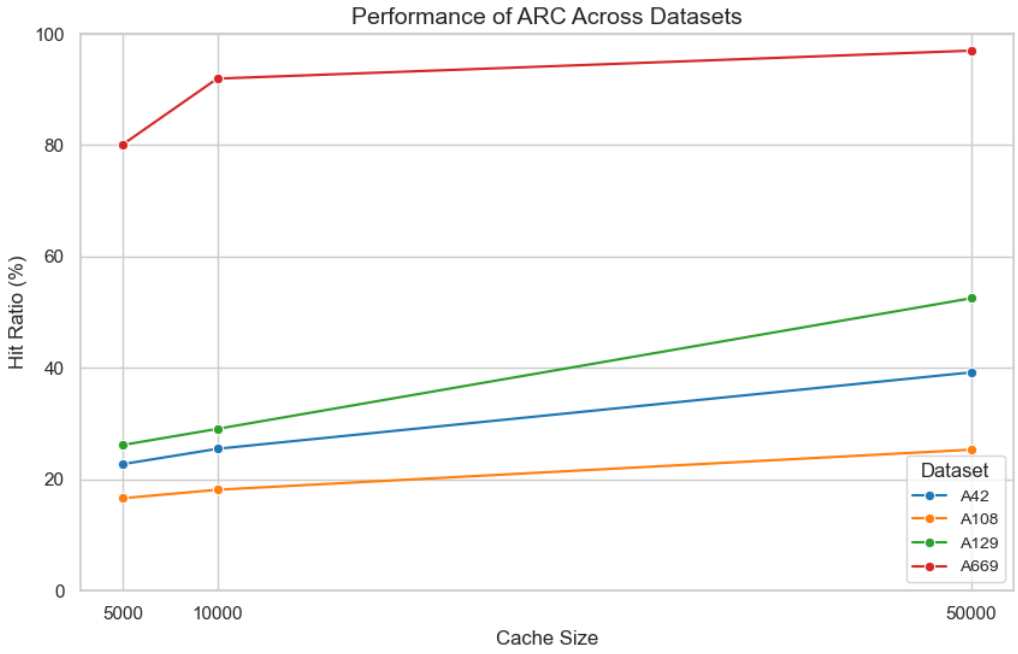
Metric	Count	Ratio
Read Requests	۱۰۶۳۰۱۷۳	
Read Hits	۱۵۳۹۴۵۴	۱۴٫۴۸٪
Read Misses	۹۰۹۰۷۱۹	۸۵٫۵۲٪
Write Requests	۹۵۷۵۱۴۵	
Write Hits	۳۵۶۹۴۸۰	۳۷٫۲۸٪
Write Misses	۶۰۰۵۶۶۵	۶۲٫۷۲٪
Total Requests	۲۰۲۰۵۳۱۸	
Total Hits	۵۱۰۸۹۳۴	۲۵٫۲۹٪
Total Misses	۱۵۰۹۶۳۸۴	۷۴٫۷۱٪

جدول ۵۲: نتیجه شبیه‌سازی مجموعه‌داده ۴۲A با سیاست ARC و اندازه حافظه نهان ۵۰۰۰۰

Metric	Count	Ratio
Read Requests	۳۰۹۸۷۳۵	
Read Hits	۹۵۵۴۲۵	۳۰٫۸۳٪
Read Misses	۲۱۴۳۳۱۰	۶۹٫۱۷٪
Write Requests	۱۹۸۹۱۹۷	
Write Hits	۱۰۳۸۱۰۹	۵۲٫۱۹٪
Write Misses	۹۵۱۰۸۸	۴۷٫۸۱٪
Total Requests	۵۰۸۷۹۳۲	
Total Hits	۱۹۹۳۵۳۴	۳۹٫۱۸٪
Total Misses	۳۰۹۴۳۹۸	۶۰٫۸۲٪

نمودارهای مربوط به این قسمت نیز در اینجا آورده شده است:





۷ سیاست LARC

الگوریتم LARC (Lazy Adaptive Replacement Cache) نسخه‌ای بهینه از ARC است که با هدف کاهش سربار محاسباتی و جلوگیری از آلودگی حافظه نهان طراحی شده است. در این الگوریتم،

برخلاف ARC، داده‌ها به محض یک Cache Miss وارد حافظه نهان نمی‌شوند. به جای آن، از یک حافظه شبح (Ghost Cache) به نام Qr استفاده می‌شود که تنها مشخصات بلوک‌های داده (و نه محتوای آن‌ها) را ذخیره می‌کند. اگر یک بلوک در دسترسی‌های بعدی دوباره درخواست شود و در Qr موجود باشد، این بلوک به حافظه نهان اصلی (Q) منتقل می‌شود و جایگاه آن در Q به‌روزرسانی می‌شود. در غیر این صورت، مشخصات بلوک به Qr اضافه شده و در صورت پر بودن Qr، قدیمی‌ترین بلوک از آن حذف می‌شود.

ویژگی مهم LARC در مقایسه با ARC این است که داده‌هایی که تنها یک بار درخواست شده‌اند، وارد حافظه نهان نمی‌شوند و تنها پس از دسترسی مجدد، بلوک داده به Q منتقل می‌شود. این روش نه‌تنها از آلودگی حافظه نهان جلوگیری می‌کند، بلکه به کاهش هزینه‌های محاسباتی کمک می‌کند. علاوه بر این، با استفاده از متغیر cr که اندازه ایده‌آل حافظه شبح را تنظیم می‌کند، الگوریتم به صورت پویا خود را با تغییرات بارکاری تطبیق می‌دهد. به این ترتیب، LARC با حفظ عملکرد مشابه ARC، کارایی بیشتری در مدیریت حافظه نهان ارائه می‌دهد و برای سناریوهایی با تغییرات سریع در الگوی دسترسی به داده‌ها مناسب‌تر است.

توضیح برنامه شبیه‌سازی LARC

کد شبیه‌سازی سیاست LARC به زبان پایتون نوشته شده و در فایل `LARC.py` قرار دارد. در ادامه اجزای این کد و عملکرد آن‌ها توضیح داده می‌شود.

ساختار کد و اجزای اصلی

کد شبیه‌سازی از چندین تابع و متغیر اصلی تشکیل شده که عبارتند از:

- **`Q` و `Qr`:** این دو لیست OrderedDict هستند که به ترتیب حافظه نهان فعال و حافظه نهان ثانویه را نشان می‌دهند.
  - `Q`: شامل داده‌هایی است که مستقیماً در حافظه نهان نگهداری می‌شوند.
  - `Qr`: شامل داده‌هایی است که به صورت شبح (ghost entries) ذخیره شده‌اند.
- **`cr`:** متغیری که نشان‌دهنده ظرفیت حافظه نهان ثانویه (`Qr`) است و به صورت پویا بر اساس رفتار الگوریتم تنظیم می‌شود. مقدار اولیه `cr` برابر با ۱۰% ظرفیت کل حافظه نهان در نظر گرفته می‌شود.

- **`cache_hits` و `cache_misses`:** این متغیرها تعداد دفعات Cache Hit و Cache Miss را ثبت می‌کنند.

توابع اصلی کد

۱. **`read_csv_first_column`:** این تابع فایل CSV را خوانده و داده‌ها را بر اساس محدوده زمانی مشخص فیلتر می‌کند. داده‌های خروجی به عنوان توالی صفحات به الگوریتم ارسال می‌شوند.

۲. **`run_larc_python`:** این تابع اصلی شبیه‌سازی LARC است که شامل مراحل زیر است:

- **بررسی Cache Hit:** اگر صفحه موردنظر در `Q` وجود داشته باشد، به ابتدای لیست منتقل شده و مقدار `cache_hits` افزایش می‌یابد. همچنین مقدار `cr` کاهش می‌یابد تا حافظه نهان ثانویه کوچک‌تر شود.
- **بررسی Cache Miss:** اگر صفحه در `Qr` باشد، به `Q` منتقل شده و مقدار `cr` افزایش می‌یابد. در غیر این صورت، صفحه به `Qr` اضافه می‌شود.
- **حذف صفحات اضافی:** در صورتی که تعداد صفحات در `Qr` از مقدار `cr` بیشتر شود، قدیمی‌ترین صفحه حذف می‌شود.

۳. **`run_all_policies`:** این تابع سیاست‌های مختلف مدیریت حافظه نهان، از جمله LARC را روی مجموعه داده ورودی اجرا کرده و نتایج را گزارش می‌کند.

نتایج عملکرد

در آخر خروجی برنامه مقادیر Cache Misses Cache Hits و است.

همچنین اندازه نهایی Q و Qr و cr را چاپ می‌کنیم.

خروجی برنامه

نتایج شبیه‌سازی برای اندازه حافظه نهان ۵۰۰۰، برای هر مجموعه‌داده مختلف در زیر آمده.

جدول ۵۳: نتیجه شبیه‌سازی مجموعه‌داده A۶۶۹ با سیاست LARC و اندازه حافظه نهان ۵۰۰۰

Metric	Count	Ratio
Read Requests	۲۵۹۵۹۱۹۵۶	
Read Hits	۲۲۱۰۸۶۹۲	۸۵٫۱۹٪
Read Misses	۳۸۴۳۲۶۴	۱۴٫۸۱٪
Write Requests	۱۲۹۲۸۷۷	
Write Hits	۵۹۸۲۰۸	۴۶٫۲۷٪
Write Misses	۶۹۴۶۶۹	۵۳٫۷۳٪
Total Requests	۲۷۲۴۴۸۳۳	
Total Hits	۲۲۷۰۶۹۰۰	۸۳٫۳۴٪
Total Misses	۴۵۳۷۹۳۳	۱۶٫۶۶٪

جدول ۵۴: نتیجه شبیه‌سازی مجموعه‌داده A۱۲۹ با سیاست LARC و اندازه حافظه نهان ۵۰۰۰

Metric	Count	Ratio
Read Requests	۴۸۶۸۱۷۳	
Read Hits	۶۳۶۶۱۷	۱۳٫۰۸٪
Read Misses	۴۲۳۱۵۵۶	۸۶٫۹۲٪
Write Requests	۱۲۸۴۴۳۲۰	
Write Hits	۳۱۰۳۳۴۸	۲۴٫۱۶٪
Write Misses	۹۷۴۰۹۷۲	۷۵٫۸۴٪
Total Requests	۱۷۷۷۱۲۴۹۳	
Total Hits	۳۷۳۹۹۶۵	۲۱٫۱۱٪
Total Misses	۱۳۹۷۲۵۲۸	۷۸٫۸۹٪

جدول ۵۵: نتیجه شبیه‌سازی مجموعه‌داده ۱۰۸A با سیاست LARC و اندازه حافظه نهان ۵۰۰۰

Metric	Count	Ratio
Read Requests	۱۰۶۳۰۱۷۳	
Read Hits	۶۸۳۲۴۲	۶٫۴۳٪
Read Misses	۹۹۴۶۹۳۱	۹۳٫۵۷٪
Write Requests	۹۵۷۵۱۴۵	
Write Hits	۲۰۰۰۷۹۶	۲۰٫۹۰٪
Write Misses	۷۵۷۴۳۴۹	۷۹٫۱۰٪
Total Requests	۲۰۲۰۵۳۱۸	
Total Hits	۲۶۸۴۰۳۸	۱۳٫۲۸٪
Total Misses	۱۷۵۲۱۱۲۸۰	۸۶٫۷۲٪

جدول ۵۶: نتیجه شبیه‌سازی مجموعه‌داده ۴۲A با سیاست LARC و اندازه حافظه نهان ۵۰۰۰

Metric	Count	Ratio
Read Requests	۳۰۹۸۷۳۵	
Read Hits	۱۲۸۱۹۲	۴٫۱۴٪
Read Misses	۲۹۷۰۵۴۳	۹۵٫۸۶٪
Write Requests	۱۹۸۹۱۹۷	
Write Hits	۹۵۸۶۷۸	۴۸٫۱۹٪
Write Misses	۱۰۳۰۵۱۹	۵۱٫۸۱٪
Total Requests	۵۰۸۷۹۳۲	
Total Hits	۱۰۸۶۸۷۰	۲۱٫۳۶٪
Total Misses	۴۰۰۱۰۶۲	۷۸٫۶۴٪

نتایج شبیه‌سازی برای اندازه حافظه نهان ۱۰۰۰۰، برای هر مجموعه‌داده مختلف در زیر آمده.

جدول ۵۷: نتیجه شبیه‌سازی مجموعه‌داده ۶۶۹A با سیاست LARC و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۲۵۹۵۹۱۹۵۶	
Read Hits	۲۴۵۹۸۲۴۹	۹۴٫۷۸٪
Read Misses	۱۳۵۳۷۰۷	۵٫۲۲٪
Write Requests	۱۲۹۲۸۷۷	
Write Hits	۶۳۵۴۶۹	۴۹٫۱۵٪
Write Misses	۶۵۷۴۰۸	۵۰٫۸۵٪
Total Requests	۲۷۲۴۴۸۳۳	
Total Hits	۲۵۲۳۳۷۱۸	۹۲٫۶۲٪
Total Misses	۲۰۱۱۱۱۵	۷٫۳۸٪

جدول ۵۸: نتیجه شبیه‌سازی مجموعه‌داده ۱۲۹A با سیاست LARC و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۴۸۶۸۱۷۳	
Read Hits	۶۸۹۸۰۲	۱۴٫۱۷٪
Read Misses	۴۱۷۸۳۷۱	۸۵٫۸۳٪
Write Requests	۱۲۸۴۴۳۲۰	
Write Hits	۳۴۳۲۵۲۸	۲۶٫۷۲٪
Write Misses	۹۴۱۱۷۹۲	۷۳٫۲۸٪
Total Requests	۱۷۷۷۱۲۴۹۳	
Total Hits	۴۱۲۲۳۳۰	۲۳٫۲۷٪
Total Misses	۱۳۵۹۰۱۶۳	۷۶٫۷۳٪

جدول ۵۹: نتیجه شبیه‌سازی مجموعه‌داده ۱۰۸A با سیاست LARC و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۱۰۶۳۰۱۷۳	
Read Hits	۷۴۰۰۷۶	۶٫۹۶٪
Read Misses	۹۸۹۰۰۹۷	۹۳٫۰۴٪
Write Requests	۹۵۷۵۱۴۵	
Write Hits	۲۲۲۰۱۴۰	۲۳٫۱۹٪
Write Misses	۷۳۵۵۰۰۵	۷۶٫۸۱٪
Total Requests	۲۰۲۰۵۳۱۸	
Total Hits	۲۹۶۰۲۱۶	۱۴٫۶۵٪
Total Misses	۱۷۲۴۵۱۰۲	۸۵٫۳۵٪

جدول ۶۰: نتیجه شبیه‌سازی مجموعه‌داده ۴۲A با سیاست LARC و اندازه حافظه نهان ۱۰۰۰۰

Metric	Count	Ratio
Read Requests	۳۰۹۸۷۳۵	
Read Hits	۱۵۵۹۶۸	۵٫۰۳٪
Read Misses	۲۹۴۲۷۶۷	۹۴٫۹۷٪
Write Requests	۱۹۸۹۱۹۷	
Write Hits	۱۰۲۹۷۸۱	۵۱٫۷۷٪
Write Misses	۹۵۹۴۱۶	۴۸٫۲۳٪
Total Requests	۵۰۸۷۹۳۲	
Total Hits	۱۱۸۵۷۴۹	۲۳٫۳۱٪
Total Misses	۳۹۰۲۱۸۳	۷۶٫۶۹٪

نتایج شبیه‌سازی برای اندازه حافظه نهان ۵۰۰۰۰، برای هر مجموعه‌داده مختلف در زیر آمده.

جدول ۶۱: نتیجه شبیه‌سازی مجموعه‌داده ۶۶۹A با سیاست LARC و اندازه حافظه نهان ۵۰۰۰۰

Metric	Count	Ratio
Read Requests	۲۵۹۵۹۱۹۵۶	
Read Hits	۲۵۴۴۴۳۹۳	۹۸٫۰۴٪
Read Misses	۵۰۷۵۶۳	۱٫۹۶٪
Write Requests	۱۲۹۲۸۷۷	
Write Hits	۶۸۷۳۵۷	۵۳٫۱۶٪
Write Misses	۶۰۵۵۵۲۰	۴۶٫۸۴٪
Total Requests	۲۷۲۴۴۸۳۳	
Total Hits	۲۶۱۳۱۷۵۰	۹۵٫۹۱٪
Total Misses	۱۱۱۳۰۸۳	۴٫۰۹٪

جدول ۶۲: نتیجه شبیه‌سازی مجموعه‌داده ۱۲۹A با سیاست LARC و اندازه حافظه نهان ۵۰۰۰۰

Metric	Count	Ratio
Read Requests	۴۸۶۸۱۷۳	
Read Hits	۹۷۶۳۲۹	۲۰٫۰۶٪
Read Misses	۳۸۹۱۸۴۴	۷۹٫۹۴٪
Write Requests	۱۲۸۴۴۳۲۰	
Write Hits	۷۵۲۹۴۱۹	۵۸٫۶۲٪
Write Misses	۵۳۱۴۹۰۱	۴۱٫۳۸٪
Total Requests	۱۷۷۷۱۲۴۹۳	
Total Hits	۸۵۰۵۷۴۸	۴۸٫۰۲٪
Total Misses	۹۲۰۶۷۴۵	۵۱٫۹۸٪

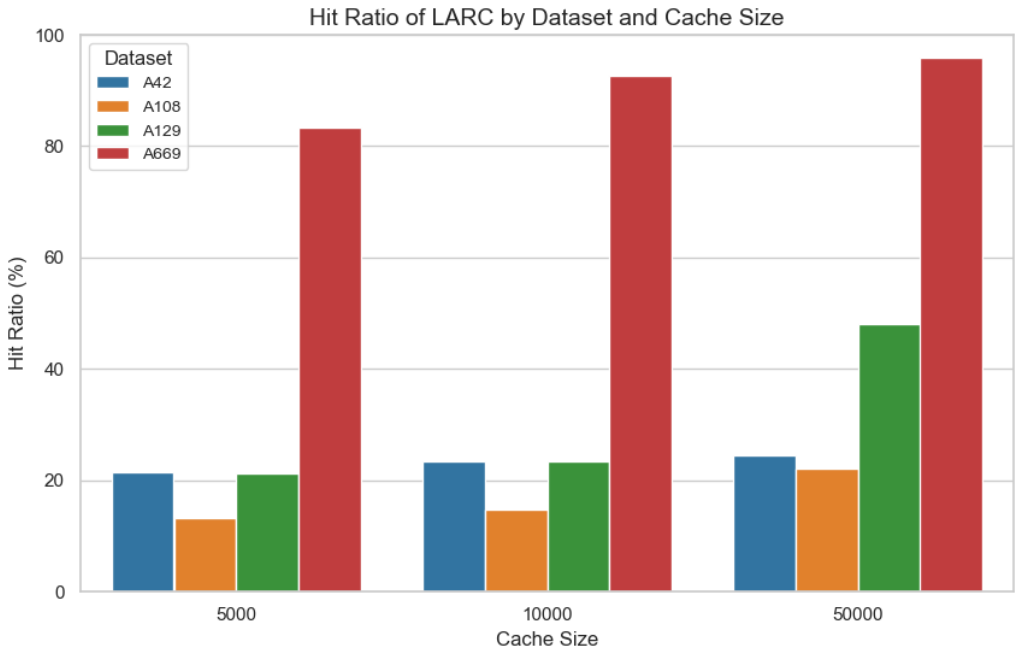
جدول ۶۳: نتیجه شبیه‌سازی مجموعه‌داده ۱۰۸A با سیاست LARC و اندازه حافظه نهان ۵۰۰۰۰

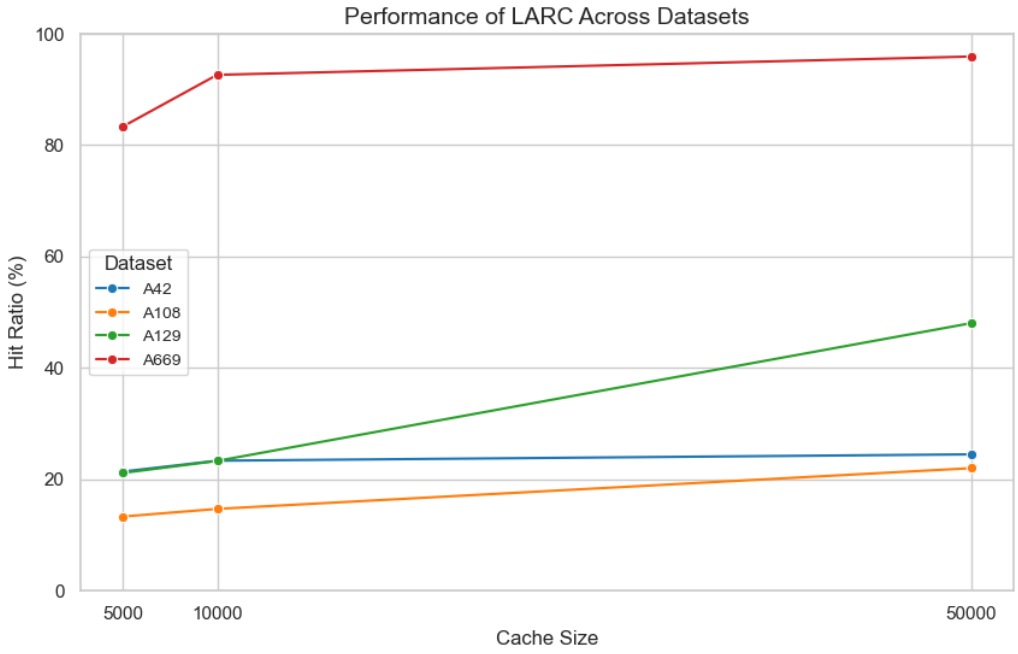
Metric	Count	Ratio
Read Requests	۱۰۶۳۰۱۷۳	
Read Hits	۱۶۵۶۳۸۷	۱۵٫۵۸٪
Read Misses	۸۹۷۳۷۸۶	۸۴٫۴۲٪
Write Requests	۹۵۷۵۱۴۵	
Write Hits	۲۷۸۳۵۲۶	۲۹٫۰۷٪
Write Misses	۶۷۹۱۶۱۹	۷۰٫۹۳٪
Total Requests	۲۰۲۰۵۳۱۸	
Total Hits	۴۴۳۹۹۱۳	۲۱٫۹۷٪
Total Misses	۱۵۷۶۵۴۰۵	۷۸٫۰۳٪

جدول ۶۴: نتیجه شبیه‌سازی مجموعه‌داده ۴۲A با سیاست LARC و اندازه حافظه نهان ۵۰۰۰۰

Metric	Count	Ratio
Read Requests	۳۰۹۸۷۳۵	
Read Hits	۱۹۰۳۳۹	۶٫۱۴٪
Read Misses	۲۹۰۸۳۹۶	۹۳٫۸۶٪
Write Requests	۱۹۸۹۱۹۷	
Write Hits	۱۰۵۳۵۰۷	۵۲٫۹۶٪
Write Misses	۹۳۵۶۹۰	۴۷٫۰۴٪
Total Requests	۵۰۸۷۹۳۲	
Total Hits	۱۲۴۳۸۴۶	۲۴٫۴۵٪
Total Misses	۳۸۴۴۰۸۶	۷۵٫۵۵٪

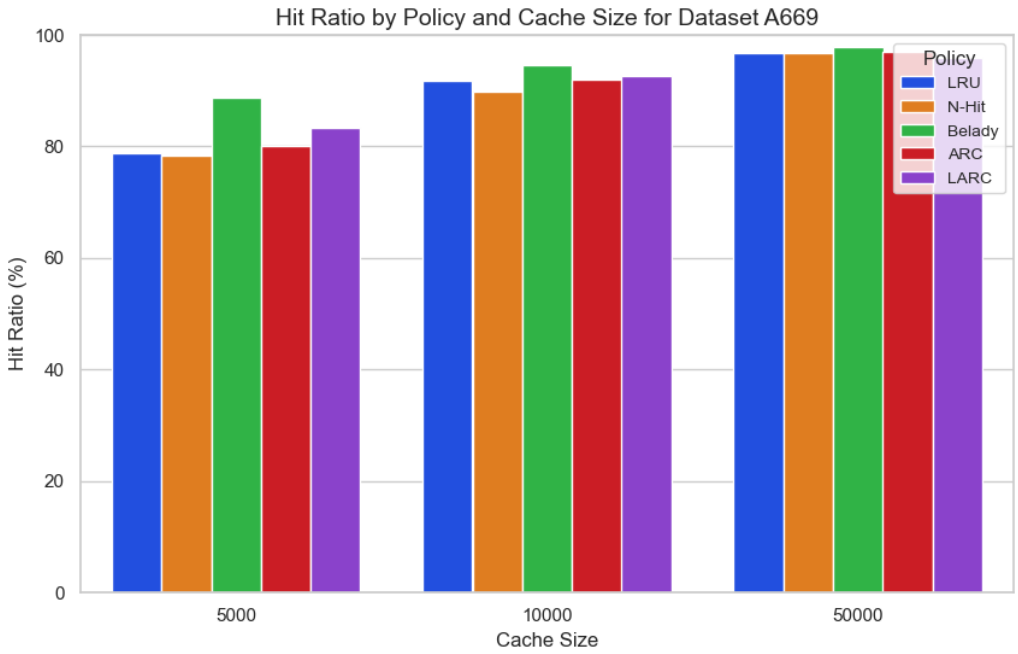
نمودارهای مربوط به این قسمت نیز در اینجا آورده شده است:

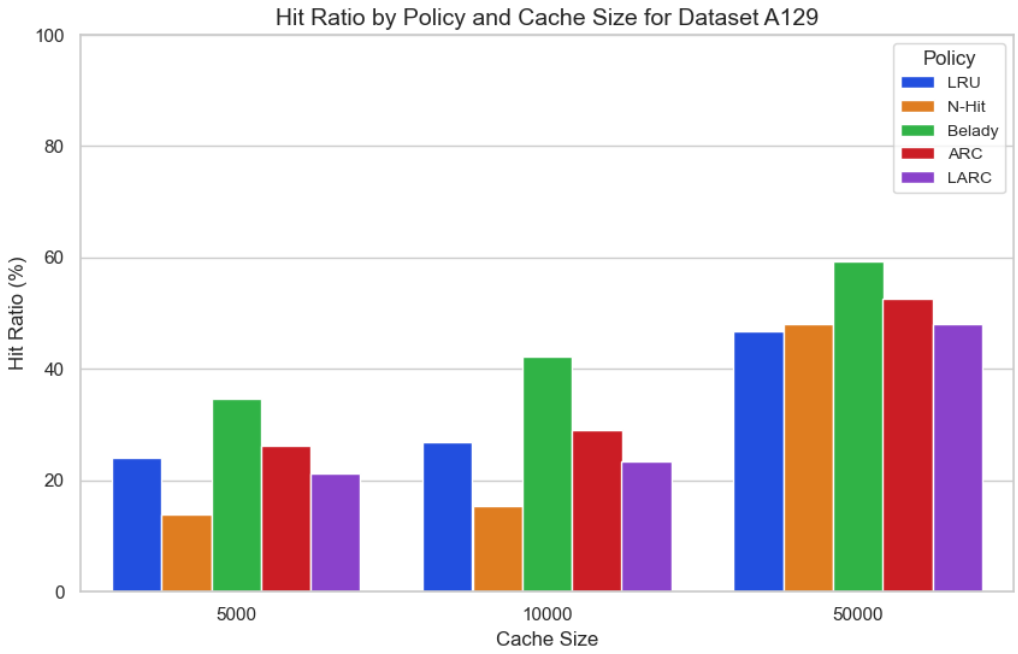


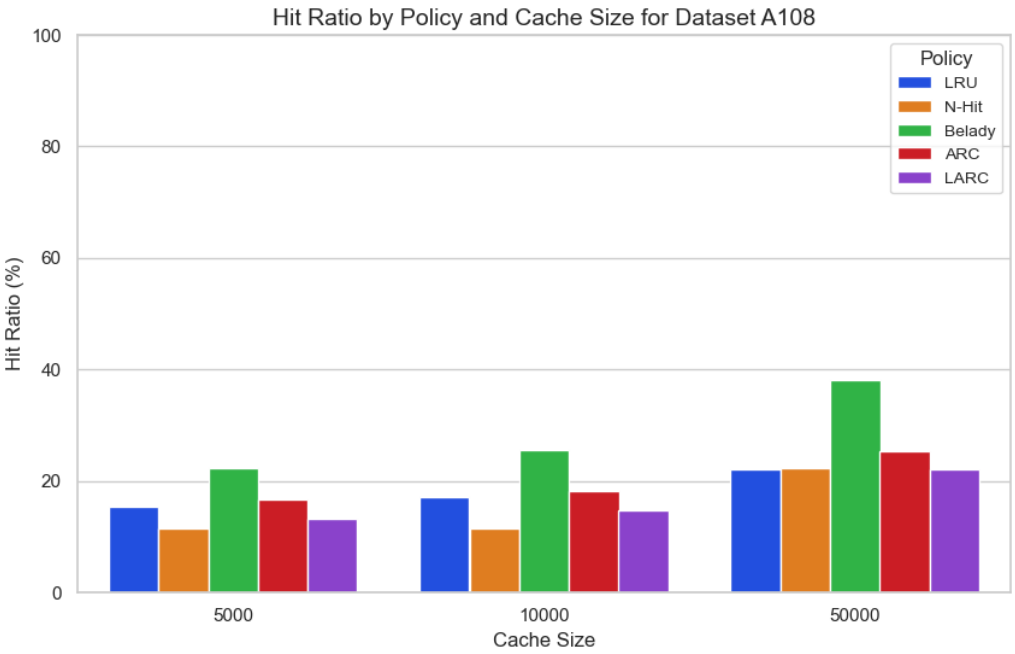


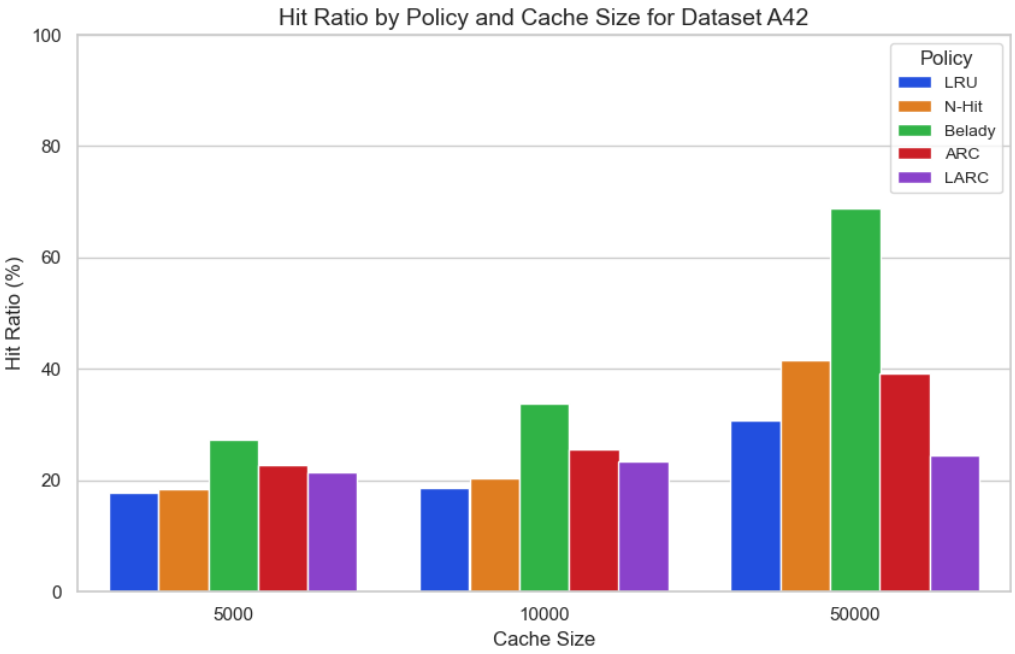
۸ مقایسه و جمع‌بندی

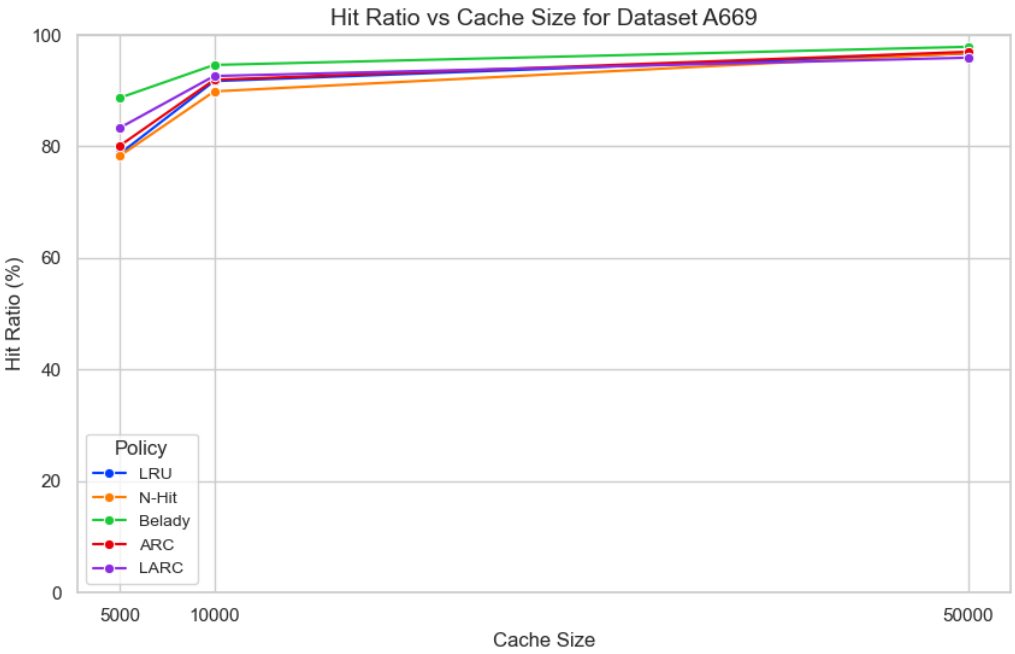
در نمودارهای زیر تمام الگوریتم‌ها را روی مجموعه‌داده‌های متفاوت مقایسه کرده‌ایم.

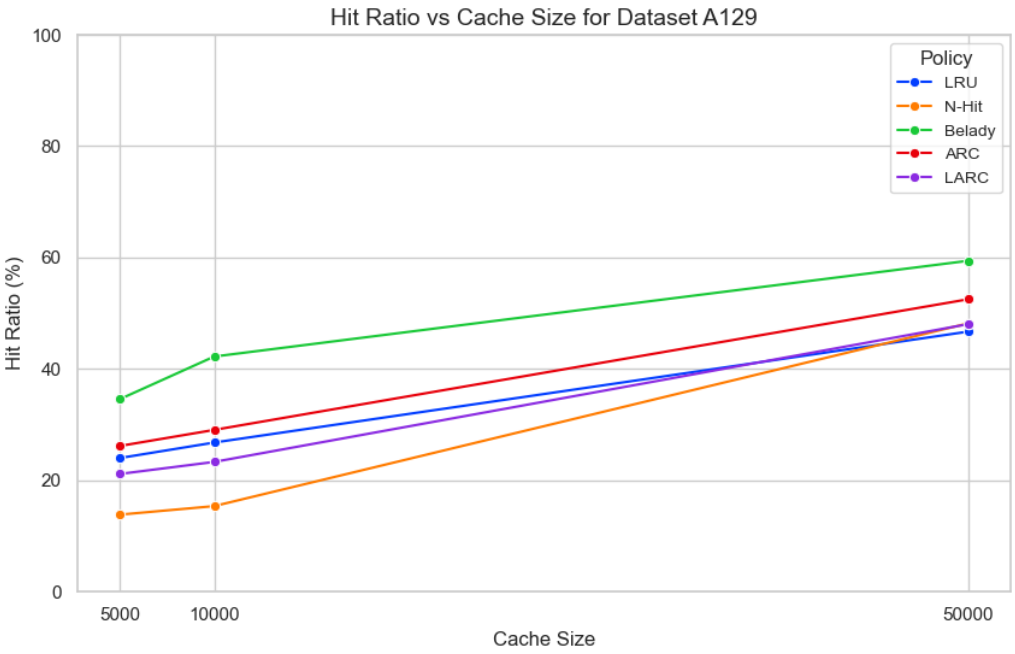

Hit Ratio by Policy and Cache Size for Dataset A669
Hit Ratio (%)
100
80
60
40
20
0
5000
10000
50000
Cache Size
Policy
LRU
N-Hit
Belady
ARC
LARC

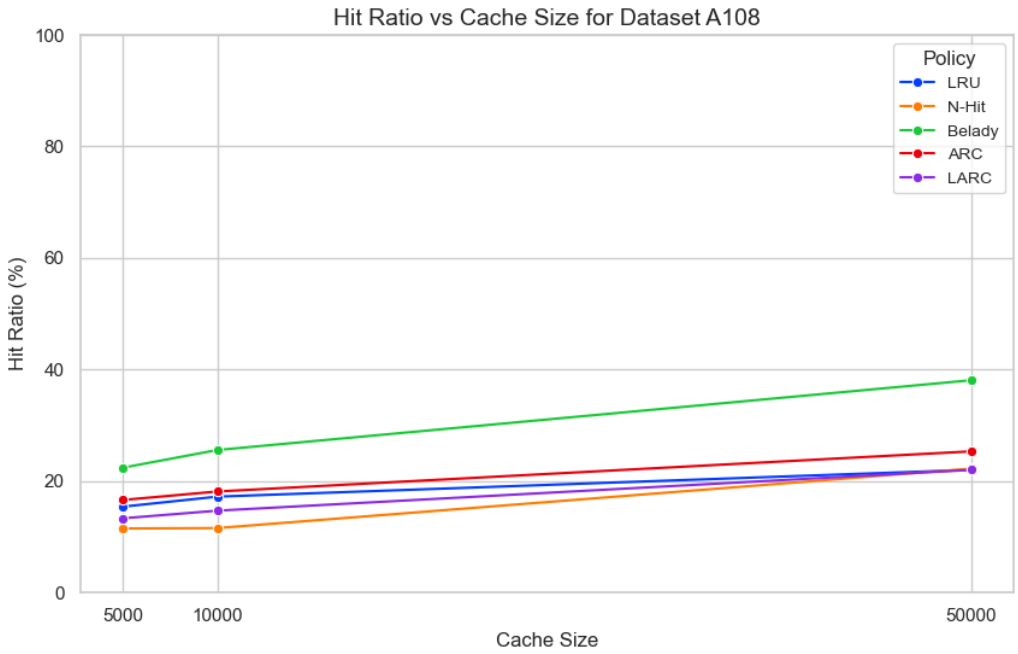

Hit Ratio by Policy and Cache Size for Dataset A129
Hit Ratio (%)
100
80
60
40
20
0
5000
10000
50000
Cache Size
Policy
LRU
N-Hit
Belady
ARC
LARC

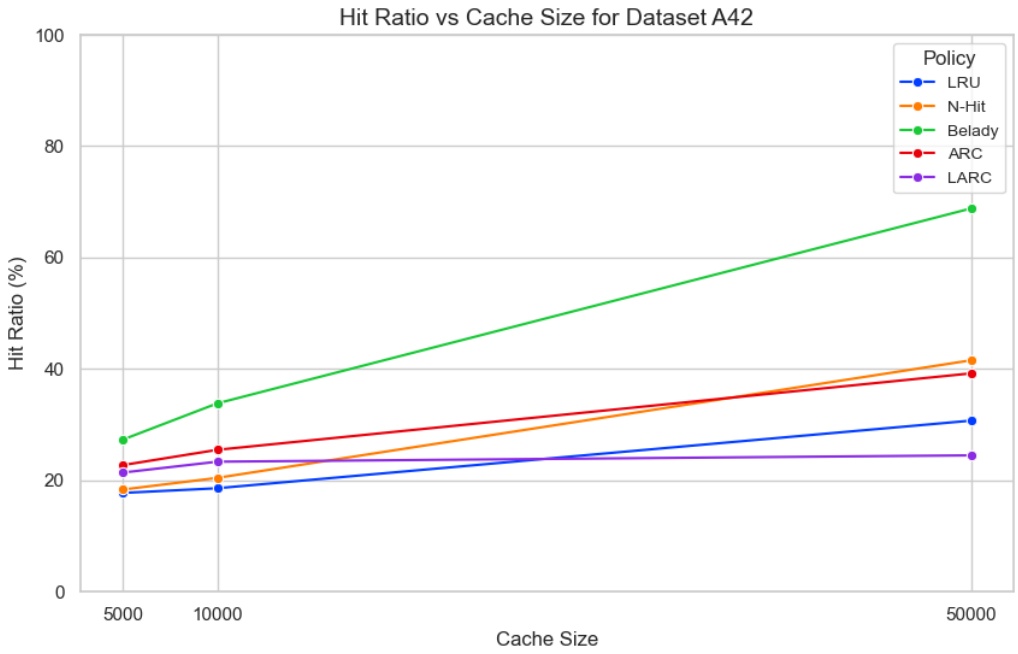

Hit Ratio by Policy and Cache Size for Dataset A108
Hit Ratio (%)
100
80
60
40
20
0
5000
10000
50000
Cache Size
Policy
LRU
N-Hit
Belady
ARC
LARC

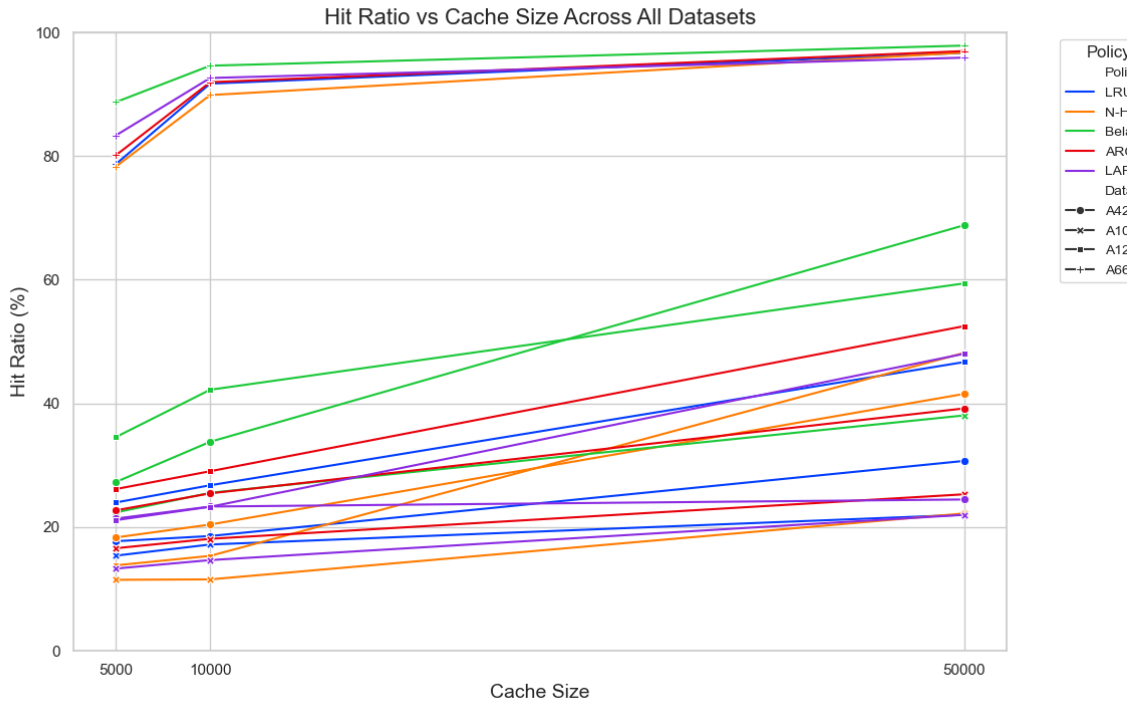

Hit Ratio by Policy and Cache Size for Dataset A42
Hit Ratio (%)
100
80
60
40
20
0
5000
10000
50000
Cache Size
Policy
LRU
N-Hit
Belady
ARC
LARC


Hit Ratio vs Cache Size for Dataset A669
Hit Ratio (%)
Cache Size
Policy
LRU
N-Hit
Belady
ARC
LARC
0
20
40
60
80
100
5000
10000
50000


Hit Ratio vs Cache Size for Dataset A129
Hit Ratio (%)
Cache Size
Policy
LRU
N-Hit
Belady
ARC
LARC
0
20
40
60
80
100
5000
10000
50000


Hit Ratio vs Cache Size for Dataset A108
Hit Ratio (%)
Cache Size
Policy
LRU
N-Hit
Belady
ARC
LARC
0
20
40
60
80
100
5000
10000
50000


Hit Ratio vs Cache Size for Dataset A42
Hit Ratio (%)
Cache Size
Policy
LRU
N-Hit
Belady
ARC
LARC
0
20
40
60
80
100
5000
10000
50000



نتایج نشان می‌دهد که سیاست **Belady** به‌طور قابل‌توجهی عملکرد بهتری نسبت به سایر الگوریتم‌ها دارد و با افزایش اندازه حافظه نهان، میزان برخورد آن نیز افزایش می‌یابد. این نتیجه قابل انتظار است، زیرا **Belady** یک الگوریتم بهینه است. در مقابل، سیاست **N-hit** در بیشتر اندازه حافظه‌ها و مجموعه‌داده‌ها، کمترین میزان برخورد را نشان می‌دهد که نشان‌دهنده عملکرد ضعیف‌تر آن نسبت به سایر روش‌های جایگزینی است. البته این سیاست عملکرد خوبی را روی مجموعه‌داده A42 داشت.

سیاست **ARC** عملکرد قابل‌قبولی دارد و از LRU بهتر عمل می‌کند، اما همچنان به میزان برخورد **Belady** نمی‌رسد.

سیاست **LARC** در اکثر مقایسه‌ها بدتر از **ARC** عمل‌کرد و تنها در یکی از مجموعه‌داده‌ها (مجموعه‌داده A669) بهتر عمل کرد.

در مجموع، اگرچه **Belady** به‌عنوان یک معیار ایده‌آل بالاترین میزان ضربه را دارد، اما در عمل نمی‌توان از آن استفاده کرد. الگوریتم **ARC** گزینه بهتری نسبت به **LRU** است و عملکرد مناسبی در مقایسه با سایر روش‌ها ارائه می‌دهد.

اما یک درس دیگری که می‌توان از این شبیه‌سازی‌ها گرفت این است که سیاست‌های مختلف در مجموعه‌داده‌های متفاوت، عملکرد یکسانی ندارند و سیاستی که روی یک مجموعه‌داده خاص

خوب عمل می‌کند ممکن است روی مجموعه‌داده‌ی دیگری، از بقیه سیاست‌ها بدتر باشد.