دانشگاه صنعتی شریف
دانشکده مهندسی کامپیوتر

**پایان‌نامه کارشناسی ارشد**
**معماری کامپیوتر**

# بهبود کارایی حافظه‌ی نهان سامانه‌های ذخیره‌سازی داده با استفاده از یادگیری ماشین

نگارش

**شهریار ابراهیمی**

استاد راهنما

**دکتر حسین اسدی**

تابستان ۱۳۹۶

تقدیم به پدر و مادر عزیزم که در تمامی مراحل زندگی یار و یاور همیشگی من بودند.

**به نام او**

**دانشگاه صنعتی شریف**
**دانشکده مهندسی کامپیوتر**

پایان‌نامه کارشناسی ارشد

عنوان: بهبود کارایی حافظه‌ی نهان سامانه‌های ذخیره‌سازی داده با استفاده از یادگیری ماشین

نگارش: شهریار ابراهیمی

کمیته داوران

استاد راهنما: دکتر حسین اسدی امضاء: . . . . . .

ممتحن داخلی: دکتر امیرحسین جهانگیر امضاء: . . . . . .

داور خارجی: دکتر احمد خونساری امضاء: . . . . . .

تاریخ: ۱۳۹۶/۶/۲۰

# قدردانی

با تشکر از پدر و مادرم بابت زحمات بی‌دریغی که در این سال‌ها برای من کشیدند و همچنین جناب آقای دکتر اسدی استاد راهنمایم بابت راهنمایی‌های بسیار ارزشمندشان که بدون آن انجام این پایان‌نامه میسر نبود.

# بهبود کارایی حافظه‌ی نهان سامانه‌های ذخیره‌سازی داده با استفاده از یادگیری ماشین

## چکیده

امروزه در مراکز داده، زیرسامانه‌های ذخیره‌سازی داده به دلیل زمان پاسخ‌گویی طولانی‌تر نسبت به دیگر مولفه‌های سامانه‌های رایانه‌ای به یکی از گلوگاه‌های اصلی کارایی تبدیل شده‌اند. با پیدایش فناوری‌های جدید همچون دیسک‌های حالت جامد، زمان پاسخ کاهش یافته است. اما در کنار کارایی بالاتر، دیسک‌های حالت جامد، مشکلاتی همچون طول عمر محدود و هزینه‌ی بالاتر نسبت به دیسک‌های سخت دارند که از جایگزینی کامل آنها جلوگیری می‌کند. برای بهره‌گیری از کارایی بالای دیسک‌های حالت جامد و کاستن از مشکلات آن‌ها، از این دیسک‌ها می‌توان به عنوان حافظه‌های نهان برای دیسک‌های سخت استفاده نمود. کارهای پیشین در این زمینه، عموما بر روی یک نوع بار کاری خاص تمرکز کرده و بهینه‌سازی‌های مبتنی بر مشاهده و اکتشاف انجام داده‌اند. در این پژوهش روشی برای بهبود کارایی حافظه‌ی نهان همراه با قابلیت بازپیکربندی در صورت تغییر بارهای کاری ارائه گردیده است. برای این منظور، یک معماری حافظه‌ی نهان با قابلیت بازپیکربندی ارائه شده که به صورت برخط و با استفاده از یادگیری ماشین، بار کاری در حال اجرا را بررسی کرده و به مدیریت پویای درخواست‌های ورودی/خروجی بپردازد. در ادامه با استفاده از شبیه‌سازی و پیاده‌سازی روش یادگیری ماشین، معماری پیشنهادی راستی آزمایی شده است. نتایج بدست آمده در این پژوهش نشان می‌دهد که روش ویژگی‌شناسی پیشنهادی در بدترین شرایط بارهای کاری را با دقت بیش از ۹۴٪ به درستی تشخیص داده و معماری پیشنهادی نسبت به کارهای پیشین، نرخ برخورد را تا ۷ برابر افزایش و تعداد عملیات جایگزینی را تا ۱۰ برابر کاهش می‌دهد.

**کلمات کلیدی:** سامانه‌های ذخیره‌سازی داده، یادگیری ماشین، حافظه‌ی نهان، دیسک حالت جامد، ویژگی‌شناسی بار کاری.

# فهرست مطالب

# فهرست شکل‌ها

# فهرست جدول‌ها

# فصل ۱

## مقدمه

هرساله میزان اطلاعات ذخیره شده در مراکز داده چندین برابر می‌شود. علاوه بر این، الگوهای دسترسی[۲] به داده‌ها نیز همواره در حال تغییر است. این تغییرات همراه با افزایش تعداد پردازنده‌های مورد نیاز برنامه‌ها و سیستم عامل، میزان دسترسی به دستگاه‌های ذخیره‌سازی داده[۳] را بیش از پیش افزایش داده است. در سامانه‌های ذخیره‌سازی داده[۴] ادواتی با اجزای مکانیکی همانند دیسک‌های سخت[۵] و نوارهای مغناطیسی[۶] ، همواره گلوگاه کارایی[۷] سامانه در مقایسه با سایر بخش‌ها همچون پردازنده و حافظه‌ی اصلی بوده است. همچنین فاصله‌ی کارایی میان ادوات ذخیره‌سازی و سایر بخش‌های سامانه‌های رایانه‌ای نیز همواره در حال افزایش است.

با پیدایش فناوری‌های نوظهور مانند حافظه‌های غیرفرار[۸] و به خصوص دیسک‌های حالت جامد[۹] ، این فاصله‌ی کارایی کاهش یافته است. با این حال، این فناوری‌ها در کنار مزایایی مانند زمان پاسخ[۱۰] پایین و مصرف انرژی کم‌تر، دارای ضعف‌هایی نیز هستند. اصلی‌ترین ضعف دیسک‌های حالت جامد، تعداد محدود دفعات مجاز عملیات نوشتن است که طول عمر آن‌ها را به شدت محدود می‌نماید. علاوه بر این، قیمت بالای این دیسک‌ها نسبت به دیسک‌های سخت، باعث شده است

2. Access Patterns
3. Data Storage Devices
4. Data Storage Systems
5. Hard Disk Drive (HDDs)
6. Magnetic Tapes
7. Performance Bottleneck
8. None Volatile Memory (NVM)
9. Solid State Disk (SSD)
10. Response Time

امکان جایگزینی کامل دیسک‌های سخت با دیسک‌های حالت جامد وجود نداشته باشد.

برای بهره‌مندی از مزایای دیسک‌های حالت جامد و رفع چالش‌های آن‌ها، سامانه‌های پیشرفته‌ی ذخیره‌سازی داده از این دیسک‌ها در کنار دیسک‌های سخت در معماری‌های ترکیبی[۱] استفاده می‌کنند. چالش اصلی موجود برای طراحی این نوع معماری، چگونگی استفاده‌ی بهینه از دیسک‌های سخت و حالت جامد می‌باشد. در معماری‌های ترکیبی، به محدودیت‌های استفاده از دیسک‌های حالت جامد نیز بایستی توجه شود. در نتیجه، علاوه بر کارایی سامانه، تعداد درخواست‌های نوشتن در دیسک‌های حالت جامد و جلوگیری از انتقال بلوک‌های داده‌ای که به صورت ترتیبی[۲] دسترسی می‌شوند، باید مورد توجه قرار داده شود. به طور کلی دو روش اصلی حافظه‌ی نهان[۳] [۱، ۲] و رده‌بندی داده[۴] [۳] برای طراحی معماری‌های ترکیبی وجود دارد.

برای حل چالش‌های ذکر شده در حافظه‌ی نهان مبتنی بر دیسک حالت جامد، نیاز به مدیریت بهینه‌ی حافظه‌ی نهان با توجه به اهمیت هر درخواست نسبت به کل درخواست‌های بار کاری[۵] وجود دارد. از طرفی، در صورت تغییر بار کاری در حال اجرا، سامانه‌ی مدیریت حافظه‌ی نهان در صورت توانایی بازپیکربندی[۶] می‌تواند خود را با بار کاری جدید تطبیق داده و به کارایی بالاتری در این بار کاری دست یابد. در کارهای پیشین، برای پاسخ به این نیازها، روش‌های متنوعی بر اساس کاربردهای مختلف ابداع شده است. از این روش‌ها می‌توان به سامانه‌ی بازخورد[۷] در حافظه‌ی نهان با توجه به نرخ برخورد[۸] لحظه‌ای، تخصیص نشان[۹] به درخواست‌های ارسالی از طریق سیستم فایل[۱۰] و یا تفکیک درخواست‌های نوشتن از خواندن و اعمال سیاست‌های از پیش تعیین شده در مدیریت درخواست‌ها بر اساس نوع درخواست، اشاره کرد.

اصلی‌ترین نقطه ضعف کارهای پیشین، استفاده از الگوریتم‌های سنتی[۱۱] (مانند اخیرا کمتر

1. Hybrid Architecture
2. Sequential
3. Cache
4. Data Tiering
5. Workload
6. Reconfiguration
7. Feedback
8. Hit Ratio
9. Flag
10. Filesystem
11. Conventional

استفاده شده[1] ) معرفی شده در لایه‌های بالاتر معماری کامپیوتر همچون پردازنده‌ها در معماری حافظه‌ی نهان سامانه‌ی ذخیره‌سازی داده است. الگوریتم‌های سنتی، برای کارایی بالا به مجاورت[2] مکانی[3] و زمانی[4] خطی وابسته هستند. در حالی که بارهای کاری سامانه‌های ذخیره‌سازی داده، برخلاف بارهای کاری پردازنده‌ها، از مجاورت‌های خطی کمی برخوردارند . این موضوع موجب کاهش نرخ برخورد و افزایش عملیات جایگزینی[5] در حافظه‌ی نهان مبتنی بر الگوریتم‌های سنتی شده و علاوه بر کاهش کارایی سامانه، عمر دیسک حالت جامد را نیز کوتاه می‌کند.

هدف از این تحقیق طراحی یک حافظه‌ی نهان مبتنی بر دیسک حالت جامد است که می‌تواند در بارهای کاری مختلف، کارایی بهینه‌ای داشته باشد. از این رو یک معماری حافظه‌ی نهان جدید، با قابلیت بازپیکربندی که در هنگام تغییر بار کاری، این تغییر را با استفاده از روش‌های یادگیری ماشین[6] تشخیص و سیاست‌های حافظه‌ی نهان را بر اساس آن تغییر می‌دهد ارائه می‌شود. علاوه بر این، سیاست‌های حافظه نهان در معماری پیشنهادی توسط یادگیری ماشین اعمال شده که این روش برای اولین بار در حافظه‌ی نهان مبتنی بر دیسک حالت جامد مورد استفاده قرار می‌گیرد. در ادامه، خلاصه‌ای از نوآوری‌های این پژوهش آورده شده است:

- ارائه‌ی اولین روش ویژگی‌شناسی بار کاری مبتنی بر شبکه‌ی عصبی تکرار شونده که نتیجه‌ی آن در عملیات بازپیکربندی معماری پیشنهادی استفاده می‌شود.
- طراحی و پیاده‌سازی سیاست جدید مدیریت حافظه‌ی نهان مبتنی بر دیسک جامد، بر پایه‌ی الگوریتم مطلوب اراکل[7] [۴] و استفاده از یادگیری ماشین.
- ارائه‌ی اولین معماری حافظه‌ی نهان قابل بازپیکربندی بر پایه‌ی شبکه‌ی عصبی تکرار شونده برای سامانه‌های ذخیره‌سازی داده.

معماری پیشنهادی با استفاده از زبان C++ پیاده‌سازی و برای آزمون آن نیز از چندین بار کاری

1. Least Recently Used (LRU)
2. Locality
3. Spatial
4. Temporal
5. Replacement
6. Machine Learning
7. Oracle/Belady

استفاده شده است. پیاده‌سازی‌های روش یادگیری ماشین در زبان پایتون[1] و توسط کتابخانه‌ی Keras انجام گرفته است. برای آزمون عملکرد معماری پیشنهادی، سه روش از کارهای پیشین به همراه الگوریتم مطلوب ارکل در شبیه‌ساز پیاده‌سازی شده‌اند. آزمون‌های انجام شده نشان می‌دهد که معماری پیشنهادی می‌تواند در برخی از بارهای کاری نرخ برخورد[2] را تا بیش از ۷ برابر بهبود و عملیات جایگزینی حافظه‌ی نهان را تا ۱۰ برابر کاهش دهد. روش پیشنهادی برای ویژگی‌شناسی[3] بارهای کاری نیز در بدترین شرایط می‌تواند تا بیش از ۹۴% دقت داشته باشد.

در ادامه‌ی این پایان‌نامه، در فصل ۲ پیش‌زمینه‌ای از مباحث مورد استفاده در این پژوهش بیان خواهد شد. فصل ۳ شامل توضیحاتی در مورد کارهای مرتبط انجام شده در زمینه‌ی پژوهشی این پایان‌نامه است. در فصل ۴ معماری روش پیشنهادی با جزئیات کامل تشریح و در فصل ۵ پیاده‌سازی‌های انجام شده و نتایج حاصل، ارائه شده است. در انتها نیز در فصل ۶ نتیجه‌گیری از این پژوهش و کارهای آتی قابل انجام در این زمینه آورده شده است.

1. Python
2. Hit Ratio
3. Characterization

# فصل ۲

## پیش‌زمینه

در این فصل به توضیح و بررسی مواردی که برای درک کامل مطالب این پایان‌نامه مورد نیاز است، می‌پردازیم. هدف اصلی از این فصل، آشنایی خواننده با حوزه‌ی کاری پایان‌نامه و شرح برخی از مسائل مطرح در سامانه‌های پیشرفته‌ی ذخیره داده است. در ابتدا به بررسی دیسک حالت جامد و خصوصیات آن و سپس به بررسی مبانی حافظه نهان می‌پردازیم. این فصل را با تحلیل الگوهای مختلف حافظه نهان و در نهایت با توضیحی اجمالی درباره‌ی یادگیری ماشین به پایان می‌رسانیم.

### ۲-۱ دیسک‌های حالت جامد

دیسک‌های حالت جامد، بر خلاف دیسک‌های سخت، از هیچ ابزار مکانیکی برای عملیات درخواست‌های ورودی/خروجی[۲] استفاده نمی‌کنند و از کنار هم قرار دادن سلول‌های حافظه‌ی فلش[۳] تشکیل شده و تمام تراکنش‌های داده‌ای در آنها به صورت کاملا الکترونیکی انجام می‌گیرد. با حذف اجزای مکانیکی، دیسک‌های حالت جامد قادر به ارائه‌ی کارایی به مراتب بهتری نسبت به دیسک‌های سخت هستند. دیسک‌های حالت جامد نه تنها زمان پاسخ‌دهی بسیار پایینی نسبت به دیسک‌های سخت در خواندن و نوشتن اطلاعات دارند، بلکه دارای گذردهی[۴] بسیار بالاتری نیز هستند. [۵]

عدم استفاده از مؤلفه‌های مکانیکی، علاوه بر کارایی سخت‌افزاری بالا و سرعت پاسخگویی بسیار بالا، به دیسک‌های حالت جامد خصوصیاتی همچون استحکام بالا، مقاومت بیشتر در برابر

2. I/O Requests
3. Flash Memory Cells
4. Thoughput

ضربه و عدم تولید صدا در هنگام کار می‌دهد. البته دیسک‌های حالت جامد نقاط ضعفی از جمله قیمت بالا دارند که در مقایسه با دیسک‌های سخت بیش از ده برابر گران‌تر هستند.

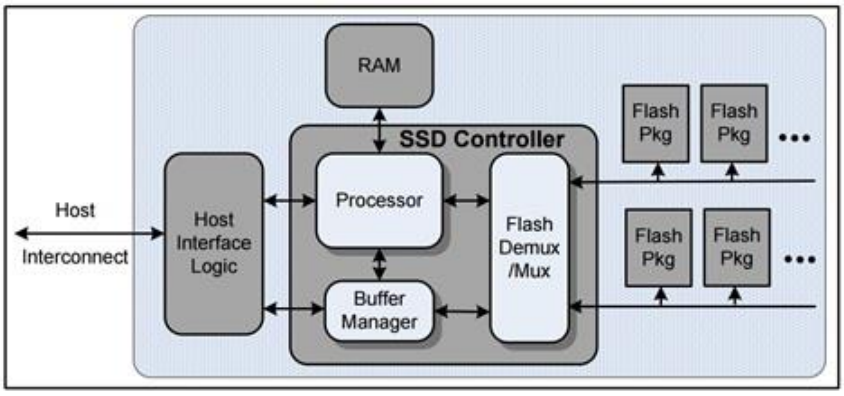
در حال حاضر اکثر دیسک‌های حالت جامد از سلول‌های حافظه‌ی فلش NAND استفاده می‌کنند که قابلیت حفظ داده را پس از قطع جریان برق دارا و در واقع حافظه‌ی غیر فرار می‌باشند. از این رو از دیسک‌های حالت جامد می‌توان به عنوان حافظه‌ای با قابلیت دسترسی تصادفی استفاده کرد که قابلیت ذخیره و حفظ داده را در هنگام قطعی برق نیز داراست.

نکته‌ی مهمی که در مورد دیسک‌های حالت جامد نباید از آن غافل بود، محدودیت تعداد درخواست‌های نوشتن در هر سلول از حافظه‌های NAND تشکیل دهنده‌ی دیسک‌های حالت جامد است. این سلول‌ها به دلیل نوع ترانزیستورهای تشکیل دهنده‌ی آنها، که از نوع نقطه‌ی شناور[1] هستند، قبل از هر دستور نوشتن نیاز به یک پاکسازی دارند. این پاکسازی دو ایراد اصلی دارد. اول این که به دلیل این پاکسازی، پاسخگویی به درخواست‌های نوشتن بسیار کندتر از درخواست‌ها خواندن خواهد بود. از طرفی، هر سلول حافظه‌ی NAND تحمل تعداد محدودی از این عملیات پاکسازی را خواهد داشت و پس از رسیدن به آن تعداد پاکسازی دیگر قابل استفاده نخواهد بود. به همین دلیل، در کل دیسک‌های حالت جامد دارای طول عمر محدودی هستند. از این رو، روش‌ها و الگوریتم‌های پیشنهادی بایستی تا حد ممکن از انجام عملیات نوشتن و در نتیجه پاکسازی قبل از آن جلوگیری کند. ثانیا تمام درخواست‌های نوشتن بین تمامی سلول‌های NAND موجود به طور مساوی پخش شوند تا تعداد درخواست‌های نوشتن یک سلول بالا نرفته و سلول از بین نرود.

امروزه دیسک‌های حالت جامد شامل دو مؤلفه‌ی اصلی هستند. اولین مؤلفه حافظه نام دارد که شامل دسته‌ی آرایه‌ی سلول‌های NAND در دیسک‌های حالت جامد می‌شود. دومین مؤلفه کنترل کننده[2] است که داده‌ها و آدرس‌های آنها و کارایی و عمر مفید سلول‌ها را مدیریت می‌کند. در شکل ۲-۱ ارتباط منطقی بین مولفه‌های یک دیسک حالت جامد نشان داده شده است.

همانطور که در این شکل مشخص است، کنترلر و آرایه‌ی سلول‌های NAND بخش‌های اصلی هر دیسک حالت جامد هستند. از دیگر مولفه‌ها می‌توان به رابط میزبان[3] اشاره کرد که مسئولیت

1. Floating Point
2. Controller
3. Host Interface




**شکل ۲-۱:** نمای داخلی دیسک حالت جامد [۵]

ارتباط دیسک حالت جامد را با خارج بر عهده دارد. مولفه‌ی حافظه دسترسی تصادفی نیز همانند یک حافظه نهان و میانگیر درخواست‌های نوشتار برای دیسک حالت جامد عمل می‌کند.

## ۲-۲ رده‌بندی داده

رده‌بندی داده در سامانه‌های ذخیره‌سازی داده، با الهام از ساختار سلسله مراتبی حافظه در معماری کامپیوتر، به انتقال داده‌های سامانه بین رده‌های مختلف می‌پردازد. هر رده از این سلسله مراتب، دارای هزینه و کارایی مشخصی می‌باشد. هدف از این روش، انتقال داده‌های با دسترسی بیشتر به لایه‌های بالاتر سلسله مراتب است تا زمان پاسخ‌گویی کلی بارهای کاری کاهش یابد. در صورت کارکرد صحیح این روش، می‌توان با هزینه‌ی کم‌تر نسبت به تغییر دستگاه‌های ذخیره‌سازی داده، کارایی سامانه را افزایش داد. [۶، ۷]

برای مدیریت سامانه‌ی رده‌بندی نیاز به ارائه‌ی سیاستی برای انتقال داده‌ها بین رده‌های مختلف است. مزیت رده‌بندی داده نسبت به حافظه‌ی نهان هزینه‌ی پایین‌تر از لحاظ حجم قابل استفاده برای ذخیره‌سازی است. در مقابل، هزینه‌ی انتقال داده در حافظه‌ی نهان بسیار کم‌تر از رده‌بندی داده است. زیرا در رده‌بندی داده نیاز به مهاجرت بلوک داده[1] بین دو رده هستیم که کارایی این عملیات توسط رده‌ی کندتر محدود می‌شود. این موضوع باعث شده تا رده‌بندی داده در سامانه‌هایی با تغییرات بالای بارهای کاری، کارایی بسیار کمی داشته باشد. لذا از این روش معمولا در سامانه‌هایی با بارهای کاری

1. Data Block Migration

ثابت و در درازمدت استفاده می‌شود. [۸، ۹]

## ۲-۳ حافظه‌ی نهان

به طور کلی هدف از بکارگیری حافظه‌ی نهان در سامانه‌های ذخیره‌سازی داده، پاسخ‌دهی به درخواست‌های آینده با کارایی بیشتر است. حافظه‌ی نهان در قسمت‌های مختلف سامانه‌های کامپیوتری کاربرد دارد ولی پرکاربردترین حافظه‌ی نهان در فضای بین پردازنده و حافظه اصلی یا میان حافظه‌ی اصلی و حافظه‌ی جانبی است. [۱، ۲، ۱۰، ۱۱]

حافظه‌هایی که به عنوان حافظه‌ی نهان مورد استفاده قرار می‌گیرند، همواره دارای سرعت بسیار بالاتری نسبت به حافظه‌ی اصلی بوده و دسترسی به داده‌ها را برای رده‌ی بالاتر تسریع می‌کنند. بدیهی است که پاسخگویی یک درخواست توسط حافظه‌ی نهان نیازمند این است که داده‌ی مورد نیاز قبلا در حافظه‌ی نهان واکشی[1] شده باشند. [۱۲]

یکی از موارد بسیار موثر در کارایی حافظه‌ی نهان، روش مدیریتی بکار رفته در آن است که مهم‌ترین وظیفه‌ی آن، انتخاب بلوک‌های داده برای واکشی یا پیش‌واکشی در حافظه‌ی نهان است. روش‌های متعددی به عنوان الگوریتم‌های واکشی برای حافظه‌های نهان بویژه در پردازنده‌ها ارائه شده‌اند. در ادامه به بررسی برخی از این روش‌ها می‌پردازیم. [۱۳، ۱۴، ۱۵]

### ۲-۳-۱ الگوریتم اولین ورودی، اولین خروجی

الگوریتم اولین ورودی، اولین خروجی[2] یکی از ساده‌ترین و ابتدایی‌ترین الگوریتم‌های واکشی داده برای حافظه‌ی نهان است. در این الگوریتم، داده‌ها در حافظه‌ی نهان به صورت ترتیبی و بر حسب اولویت زمان اولیه‌ی ورودشان مرتب‌شده قرار دارند و هر داده‌ی جدید به انتهای صف اضافه می‌شود. از آنجا که سخت‌افزار حافظه‌ی نهان، فضای ذخیره‌ی داده‌ی محدودی دارد، پس از مدتی پر می‌شود. از این به بعد، با ورود هر داده‌ی جدید، داده‌ای که در اول صف قرار داشت، از صف بیرون انداخته می‌شود. به این ترتیب، همواره اندازه‌ی صف ثابت باقی می‌ماند.

1. Fetch
2. First In First Out (FIFO)

از مزایای اصلی این الگوریتم می‌توان به سادگی پیاده‌سازی، پیچیدگی بسیار کم و درنتیجه عیب‌یابی سریع و آسان آن اشاره کرد. این روش از داده‌های درون حافظه‌ی نهان هیچ اطلاع اضافی نداشته و به همین دلیل توانایی تشخیص داده‌های مهم‌تر را از سایر داده‌ها ندارد. هر داده‌ای که وارد حافظه‌ی نهان شود، پس از مدت زمان مشخصی خارج خواهد شد. این موضوع بزرگترین مشکل این روش بوده کارایی این روش را در بسیاری از بارهای کاری پایین می‌آورد. بعنوان یک مثال فرض کنید یک سری داده‌ی بسیار پرکاربرد و مهم در ابتدای بار کاری دسترسی شده و درنتیجه در حافظه‌ی نهان واکشی می‌شوند. حال یک سری داده‌ی نه‌چندان مهم نیز تنها یک بار دسترسی شده و درنتیجه در حافظه‌ی نهان واکشی می‌شوند. ورود این داده‌های جدید باعث خروج یک سری داده‌ی مهم از حافظه‌ی نهان می‌شود که قبلا به حافظه واکشی شده بودند در حالی که اگر سامانه از اهمیت داده‌ی قبلی اطلاع داشت، نباید داده‌های کم اهمیت جدید را واکشی نماید.

## ۲-۳-۲ الگوریتم شانس دوم

این الگوریتم مشابه الگوریتم اولین ورود، اولین خروج است اما با یک تغییر کوچک که باعث می‌شود کمی کارایی آن بالاتر برود. در الگوریتم شانس دوم[1] ، هر صفحه دارای یک بیت دستیابی است. هر زمان که صفحه مورد دستیابی قرار گرفت، این بیت توسط سخت‌افزار یک (۱) می‌شود. مانند الگوریتم اولین ورود اولین خروج، صفحه‌ای که در جلوی صف قرار داشته باشد حذف می‌شود. اما به جای آنکه صفحه مورد نظر بی درنگ حذف شود، سامانه ابتدا به بیت دستیابی آن صفحه نگاه می‌کند، اگر بیت دستیابی صفر بود، صفحه حذف می‌شود. اما اگر بیت دستیابی ۱ بود، سامانه آن بیت را صفر کرده و صفحه را به انتهای صف منتقل می‌کند. این فرآیند به همین ترتیب ادامه می‌یابد. بدین ترتیب صفحه مورد نظر شانس دوباره‌ای برای باقی‌ماندن در حافظه‌ی نهان کسب کرده است. می‌توان این صف را مانند یک صف حلقوی فرض کرد که ابتدای صف به انتهای آن متصل است. اگر بیت دستیابی تمام صفحات ۱ باشد، آنگاه الگوریتم شانس دوم هم به مانند الگوریتم اولین ورود اولین خروج عمل می‌کند.

1. Second Chance Algorithm

### ۲-۳-۳ الگوریتم اخیرا کمتر استفاده شده

ایده اصلی الگوریتم اخیرا کمتر استفاده شده[1] این است که صفحاتی که در چند لحظه گذشته مورد استفاده قرار گرفته‌اند، در چند لحظه آینده هم مورد استفاده خواهند بود. همچنین این الگوریتم مبتنی بر این مفهوم است که میزان دسترسی صفحات در لحظه‌های آینده متناسب با میزان دسترسی به آن‌ها در لحظات گذشته است. در این الگوریتم وقتی سامانه قصد وارد کردن صفحه‌ای جدید و خروج صفحه‌ای قدیمی‌تر را داشته باشد، صفحه‌ای از حافظه‌ی نهان خارج می‌شود که نسبت به دیگر صفحات، مدت طولانی‌تری بدون دسترسی بوده است. از مزایای این الگوریتم می‌توان به هزینه‌ی پیاده‌سازی پایین و کارایی بالا در بارهای کاری با نسبت مجاورت زمانی بالا اشاره کرد که موجب محبوبیت این الگوریتم در معماری کامپیوتر شده است. البته این الگوریتم مشکلاتی همچون کوبیدگی حافظه‌ی نهان[2] در بارهای کاری تصادفی با مجاورت زمانی پایین دارد. روش‌های فراوانی برای بهبود این الگوریتم ارائه شده‌اند که الگوریتم اخیرا کمتر استفاده شده‌ی بخش‌بندی شده[3] این دسته است.

### ۲-۳-۴ الگوریتم اخیرا کمتر استفاده شده‌ی بخش‌بندی شده

این الگوریتم برگرفته از الگوریتم اخیرا کمتر استفاده شده است. اما تغییرات ساختاری در آن ایجاد شده تا تعداد دسترسی‌ها نیز علاوه بر زمان دسترسی در مدیریت بلوک‌های حافظه‌ی نهان موثر باشد. در این الگوریتم، بلوک‌های حافظه‌ی نهان به دو بخش آزمایشی[4] و حفاظت شده[5] تقسیم می‌شوند. هر بلوک داده که برای اولین بار به حافظه‌ی واکشی می‌شود، در ابتدا در صف بخش آزمایشی قرار می‌گیرد. در صورت وجود برخورد در این صف، بلوک داده به بخش محافظت شده انتقال می‌یابد. به این ترتیب، بلوک‌های داده در بخش محافظت شده حداقل دو بار دسترسی داشته و بلوک‌های داده در بخش آزمایشی تنها یک بار دسترسی داشته‌اند. با این روش، بلوک‌های با تعداد دسترسی بالاتر، مدت زمان بیشتری را در حافظه‌ی نهان سپری می‌کنند. از طرفی، به دلیل کوچک‌تر بودن اندازه‌ی بخش آزمایشی نسبت به کل حافظه‌ی نهان، دسترسی‌های تصادفی فرصت کوبیدگی حافظه‌ی نهان را

1. Least Recently Used (LRU)
2. Cache Thrashing
3. Segmented Least Recently Used (SLRU)
4. Probationary
5. Protected

از دست می‌دهند.

### ۲-۳-۵ الگوریتم اولویت

این الگوریتم نسبت به بقیه‌ی الگوریتم‌ها کم‌کاربردتر است زیرا بسته به نوع بار کاری در حال اجرا در سامانه، نوع برخورد با دسترسی‌ها نیز تغییر پیدا می‌کند. به این ترتیب این روش باید برای هر سامانه‌ای به طور اختصاصی پیاده‌سازی شود. در این روش همواره صفحات موجود در حافظه‌ی نهان بر حسب یک تابع که تابع اولویت مرتب شده و صفحه‌ای که اولویت کمتری داشته‌باشد قربانی بوده و در صورت نیاز حذف می‌شود. بدیهی است که تابع اولویت از اطلاعات مختلفی برای تخصیص اولویت به صفحات استفاده می‌کند که این اطلاعات و همچنین الگوریتم استفاده شده در این تابع، برای هر سامانه به طور مختص پیاده‌سازی می‌شود.

### ۲-۳-۶ سیاست مطلوب اراکل

در معماری حافظه‌ی نهان، برای اتخاذ بهترین تصمیم ممکن در برخورد با یک درخواست جدید، نیاز است تا از آینده به طور کامل آگاه باشیم اما این در واقعیت امکان پذیر نیست. لذا تمامی معماری‌های موجود برای حافظه‌ی نهان، فاصله‌ی زیادی با مطلوب دارند. علت این موضوع، غیر قابل پیش‌بینی بودن دسترسی‌های آینده در بارهای کاری در سطح دیسک‌هاست.

الگوریتم اراکل به الگوریتمی گفته می‌شود که به صورت مطلوب از آینده به طور دقیق اطلاع دارد. این موضوع باعث می‌شود که در هنگام استفاده از این نوع الگوریتم بتوان کارآمدترین بلوک‌های داده را به حافظه‌ی نهان انتقال داد. الگوریتم اراکل به صورت سنتی در گذشته برای حافظه‌های نهان در سطح پردازنده‌ها معرفی شده است ولی برای استفاده از این نوع الگوریتم در سطح درخواست‌های روردی‌ـ‌خروجی، نیاز است که تغییراتی در نحوه‌ی محاسبات این الگوریتم داده شود. در الگوریتم اراکل به ازای هر درخواست جدید برای سامانه‌ی ذخیره‌سازی، در صورت نبود بلوک داده‌ی مورد نظر در حافظه‌ی نهان، الگوریتم به بررسی درخواست‌های آینده‌ی بار کاری پرداخته و برای تمامی بلوک‌های موجود در حافظه‌ی نهان میزان هزینه‌ای را که در صورت اخراج آن بلوک از حافظه‌ی نهان به سامانه تحمیل می‌شود حساب می‌کند. در انتها بلوک با کمترین هزینه‌ی تحمیلی در آینده را با بلوک

داده‌ی جدید مقایسه و در صورت بیشتر بودن هزینه‌ی بلوک داده‌ی جدید، این بلوک داده را جایگزین بلوک داده‌ی قبلی می‌کند. محاسبه‌ی هزینه‌ی تحمیلی در آینده، توسط تابعی به نام تابع هزینه صورت می‌گیرد. انتخاب تابع هزینه‌ی مناسب تاثیر بسیار زیادی در رفتار تابع اراکل خواهد داشت. در تابع اراکل کلاسیک که برای حافظه‌ی نهان در سطح پردازنده تعریف شده است، این تابع هزینه دقیقا برابر با تعداد دسترسی‌ها در آینده است. دلیل این کار یکسانی هزینه‌های زمانی دسترسی‌ها در آینده به ازای هر بلوک است. اما در سطح درخواست‌های ورودی‌ـ‌خروجی این مساله صادق نیست. به طور مثال هزینه‌ی زمانی یک دسترسی ترتیبی با یک درخواست تصادفی تفاوت بسیار زیادی دارد. از این رو برای استفاده از الگوریتم اراکل در سطح درخواست‌های ورودی‌ـ‌خروجی، نیاز است که تابع هزینه‌ی جدیدی تعریف شود. در حافظه‌ی نهان سامانه‌های ذخیره‌سازی، محاسبات کلی تابع هزینه برابر با رابطه‌ی ۲-۱ است.

$$Cost_{Request} = \sum Cost_{HDD} - \sum Cost_{SSD} \qquad (۱-۲)$$

در رابطه‌ی ۲-۱، مقدار $\sum Cost_{HDD}$ نشان دهنده‌ی تمام هزینه‌های تحمیلی به سامانه در آینده در صورت پاسخ‌گویی به این درخواست توسط دیسک سخت است. به همین ترتیب، مقدار $\sum Cost_{SSD}$ نشان دهنده‌ی تمام هزینه‌های تحمیلی به سامانه در آینده در صورت انتقال بلوک داده‌ی مورد نظر به حافظه‌ی نهان و در نتیجه، پاسخ‌گویی به این درخواست توسط دیسک حالت جامد است. الگوریتم اراکل اختلاف هزینه‌های تحمیلی در آینده را توسط این دو مقدار محاسبه کرده و در نهایت با توجه به هزینه‌ی تمامی بلوک‌های داده، تصمیم به انتقال یا عدم انتقال بلوک داده‌ی دسترسی شده به حافظه‌ی نهان می‌گیرد. به این ترتیب، در الگوریتم اراکل به ازای هر درخواست، مشخص می‌شود که آیا درخواست به حافظه‌ی نهان منتقل شده یا خیر و در صورت نیاز کدام بلوک‌ها از حافظه‌ی نهان اخراج شوند. با توجه به اهداف اصلی معماری‌های حافظه‌ی نهان، روش‌های متونعی برای محاسبه‌ی مقادیر $Cost_{HDD}$ و $Cost_{HDD}$ ارائه می‌شود.

## ۲-۴ یادگیری ماشین

امروزه در سامانه‌های کامپیوتری، با افزایش داده‌ها و پیدایش کاربردهای جدید و متنوع، نیاز به تحلیل و شناسایی ساختارهای داده‌ای برای کنترل بهتر و دستیابی به کارایی بالاتر در سامانه افزایش یافته است. به دلیل تنوع داده‌ای و نیازهای پیچیده‌ی کاربران، کاوش عمیق و ارائه‌ی چارچوبی قاطع برای تحلیل نظری این داده‌ها غیر ممکن است. لذا استفاده از شاخه‌های مختلف هوش مصنوعی و بخصوص یادگیری ماشین برای تحلیل و درک ارتباطات بین داده‌ای در دهه‌ی اخیر رشد چشم‌گیری داشته است. یادگیری ماشین، یکی از زیرشاخه‌های هوش مصنوعی است که این امکان را به یک ماشین می‌دهد که بتواند بدون برنامه‌ریزی قبلی، مدل‌ها و رفتارهای جدیدی را یاد بگیرد. در الگوریتم‌های بکار گرفته شده در یادگیری ماشین، برخلاف الگوریتم‌های سنتی این داده‌ها هستند که به الگوریتم یاد می‌دهند چه تصمیمی درست است. این موضوع باعث می‌شود که این‌گونه برنامه‌ها در صورت قرارگیری در برابر داده‌ها و ورودی‌های جدید بتوانند خود را بروزرسانی کرده و به آموخته‌های خود اضافه کنند. سطح اختیار یادگیری ماشین بر اساس نوع داده‌های در اختیار برنامه، به سه دسته‌ی کلی تقسیم می‌شود:

- **یادگیری با نظارت**[1] : در این روش، ورودی‌های برنامه به همراه خروجی‌های صحیح آن‌ها به رایانه داده می‌شوند. با الگوبرداری از خروجی‌های صحیح، خروجی مناسب برای ورودی‌های جدید تشخیص داده خواهد شد. این روش معمولا در مواردی که دید کافی و دسترسی به پاسخ‌های درست وجود دارد، کاربردی می‌باشد.

- **یادگیری تقویتی**[2] : برخلاف روش قبلی، در این روش ورودی‌ها به همراه خروجی‌های صحیح به رایانه داده نمی‌شود. بلکه به برنامه اجازه داده می‌شود که برای یافتن خروجی‌ها، شروع به کاوش کرده و فقط در هنگام اعلام یک خروجی، به برنامه بازخوردی مبنی بر صحیح یا اشتباه بودن خروجی داده می‌شود. این روش در مواردی مانند مهندسی معکوس کاربرد دارد.

1. Supervised Learning
2. Reinforcement Learning

- **یادگیری بدون نظارت**[1] : در این روش، هیچ بازخوردی به برنامه داده نمی‌شود و برنامه خود شروع به تحلیل ورودی‌ها برای یافتن الگوی قابل فهمی در بین آن‌ها می‌کند. این روش بیشتر در مواردی کاربرد دارد که هیچ دیدی نسبت به داده‌های ورودی وجود نداشته و از یادگیری ماشین برای دسته‌بندی، الگویابی و کاوش داده‌ها استفاده می‌شود.

## ۲-۴-۱ شبکه‌ی عصبی تکرار شونده

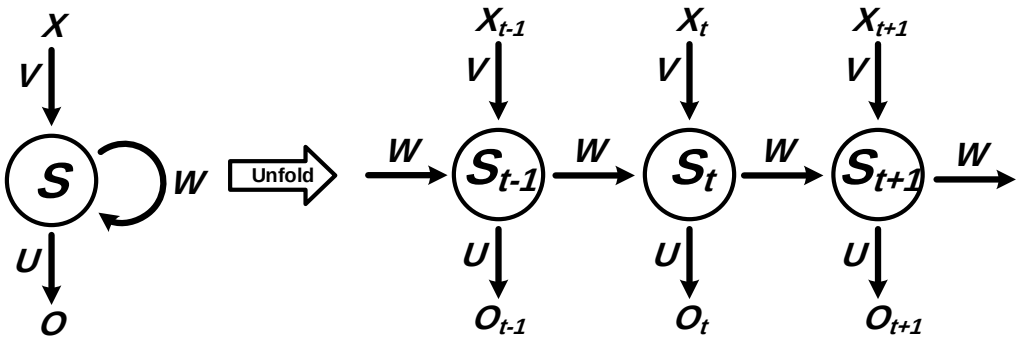
به طور کلی شبکه‌های عصبی[2] به عنوان تقلیدی از شبکه‌ی عصبی مغز انسان هستند که ارتباط بین نرون‌های عصبی مغز را با گراف‌های مختلف شبیه‌سازی می‌کنند. یکی از نمونه‌های اصلی و پرکاربرد از این نوع شبکه‌ها، شبکه‌های عصبی تکرار شونده[3] هستند که تحلیل‌های زمانی را با استفاده از گراف‌های جهت‌دار شبیه‌سازی می‌کنند. شبکه‌های عصبی تکرار شونده توانایی بالایی در پردازش داده‌های زمانی همچون سیگنال‌های صوتی دارند. داده‌ها به صورت زمانی (گام به گام) توسط شبکه طی شده و در هر مرحله حالت شبکه در مراحل قبلی در حافظه‌ی هر گره وجود دارد. به این ترتیب، این نوع شبکه‌ها می‌توانند با بهره‌گیری از ساختار پویا، به طور کامل فضای حالت سامانه را طی کنند. با توجه به رفتار غیر خطی سامانه‌های واقعی، توابع مورد استفاده در هر مرحله برای تحلیل ارتباط بین داده‌ها به صورت غیر خطی همچون tanh و $\sigma$ در نظر گرفته شده‌اند.

شکل ۲-۲ نمای شماتیک بسته و باز یک شبکه‌ی عصبی تکرار شونده را نشان می‌دهد. معادله‌ی ۲-۲ هم روابط ریاضی مربوط به این شبکه را ارائه می‌کند. همانطور که در این معادله مشخص است، در هر مرحله، حالت فعلی برگرفته از مجموعه حالت‌های قبلی است.

$$S(t+۱) = f(wS(t) + uU(t+۱)), Y(t) = g(vX(t)) \qquad (۲-۲)$$

برای تحلیل داده‌ها با دقت بالاتر، می‌توان از نوع دیگری از شبکه‌های عصبی به نام شبکه‌ی عصبی تکرار شونده‌ی عمیق[4] بهره برد. دراین نوع شبکه، در هر گره بر خلاف مدل ساده، بیش از یک حافظه‌ی حالت وجود دارد. به این ترتیب، شبکه‌ها عصبی تکرار شونده‌ی عمیق می‌توانند در هر

1. Un-Supervised Learning
2. Neural Networks
3. Recurrent Neural Network (RNN)
4. Deep RNN




**شکل ۲-۲:** نمای شماتیک بسته و باز شبکه‌ی عصبی تکرار شونده [۱۶]

مرحله بیش از یک حالت را به صورت همزمان ذخیره کنند و لذا می‌توانند الگوهای ورودی بسیار پیچیده‌تری را تحلیل کنند.

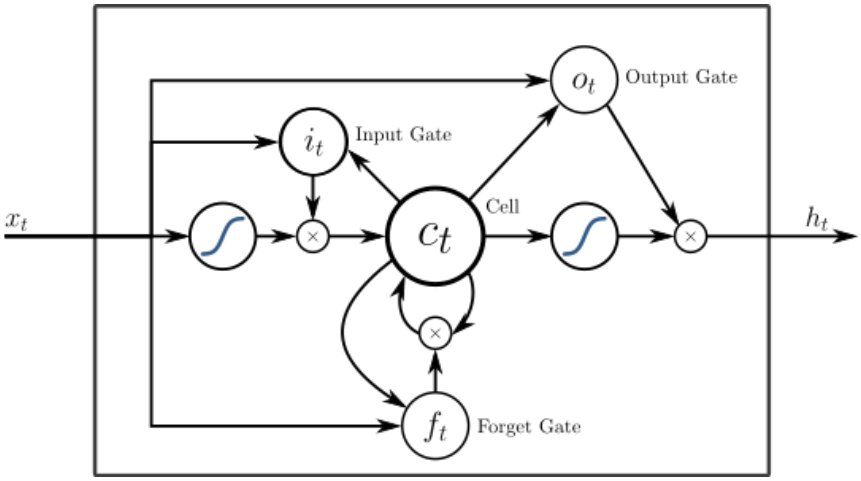
## ۲-۴-۲ حافظه‌ی طولانی کوتاه‌مدت

یکی از اصلی‌ترین معماری‌ها برای پیاده‌سازی بهینه‌ی شبکه‌های عصبی تکرار شونده، استفاده از حافظه‌ی طولانی کوتاه‌مدت[1] می‌باشد. در این نوع معماری، اطلاعات به مدت زمان دلخواهی توسط حافظه‌ی هر گره از شبکه‌ی عصبی ذخیره شده و در نهایت از حافظه پاک می‌شوند. در هر مرحله، اطلاعات موجود در گره‌ی اصلی بروز شده ولی پس از ثبت اطلاعات، دیگر محاسبه یا تغییری در اطلاعات صورت نخواهد گرفت. این نوع معماری اجازه‌ی حرکت رو به جلو[2] و رو به عقب[3] در طول زمان را به شبکه‌ی عصبی می‌دهد.

از حافظه‌ی طولانی کوتاه‌مدت در مواردی همچون طبقه‌بندی داده‌ها و یا پیش‌بینی‌های مربوط به دنباله‌های زمانی[4] که دارای دوره‌های مشخص نبوده و نیاز به کشف ارتباط دارند، استفاده می‌شود. بزرگ‌ترین مزیت حافظه‌ی طولانی کوتاه‌مدت، قابلیت بازگشت رو به عقب در زمان بدون نیاز به تکرار پردازش‌های گذشته است. از این رو این نوع از معماری شبکه‌های عصبی، دارای سربار محاسباتی کم‌تری بوده و بهترین روش برای استفاده در شبکه‌های عصبی تکرار شونده‌ی عمیق هستند. برای

1. Long short-term memory (LSTM)
2. Forward
3. Backward
4. Time Series

جلوگیری از سربار حافظه‌ی مورد نیاز برای ذخیره‌سازی اطلاعات حالت‌های قبلی شبکه، این حافظه‌ها به صورت کوتاه‌مدت بوده و شبکه‌ی عصبی به دلخواه، اطلاعاتی را که در آینده کاربرد کمتری دارند حذف می‌کند. شکل ۲-۳ ساختار هر گره در این معماری را نشان می‌دهد.




**شکل ۲-۳:** ساختار معماری حافظه‌ی طولانی کوتاه‌مدت

در این معماری، هر گره از شبکه، از سه دروازه[1] با دو حالت ۰ یا ۱ به صورت دودویی[2] تشکیل شده است. همانطور که در شکل مشخص است، این سه دروازه، ورودی، خروجی و فراموشی نام دارند که به ترتیب مقادیر داده‌های ورودی از حالت قبلی شبکه، داده‌های ارسالی و ذخیره‌شده برای حالت بعدی شبکه و اطلاعات قابل حذف از حافظه را کنترل می‌کنند. با بهره‌گیری از حافظه‌های طولانی کوتاه‌مدت، امکان تحلیل دنباله‌های درخواست‌های ورودی/خروجی با دقت بالا وجود دارد. از این‌رو در پیاده‌سازی شبکه‌های عصبی استفاده شده در این پژوهش، از این نوع حافظه‌ها استفاده شده است.

## ۲-۴-۳ جمع‌بندی

در این فصل، اطلاعات پایه‌ای مورد نیاز خواننده برای فهم بهتر دامنه‌ی کاری پایان‌نامه ارائه شده است. در ابتدای این فصل، به بررسی فناوری‌های نوین دستگاه‌های ذخیره‌سازی داده همچون دیسک‌های حالت جامد پرداخته شده است. همچنین، معماری‌های پرکاربرد در سامانه‌های ذخیره‌سازی داده و

1. Gate
2. Binary

الگوریتم‌های پایه‌ای حافظه‌های نهان مورد بررسی قرار گرفتند. این فصل با معرفی شبکه‌های عصبی و روش‌های پیاده‌سازی آن‌ها به پایان رسید.

# فصل ۳

## کارهای مرتبط پیشین

در این فصل، کارهای پیشین مرتبط با این پژوهش معرفی شده و مزایا و معایب هر کدام به صورت کامل مورد بحث قرار گرفته است. به صورت کلی، کارهای پیشین را می‌توان در سه دسته‌ی (۱) حافظه‌ی نهان مبتنی بر دیسک حالت جامد (۲) ویژگی‌شناسی بارهای کاری و (۳) استفاده از روش‌های یادگیری ماشین در سامانه‌های کامپیوتری تقسیم‌بندی نمود. در دسته‌ی اول تلاش می‌شود تا کارایی حافظه‌ی نهان مبتنی بر دیسک‌های حالت جامد با توجه به نرخ برخورد، زمان پاسخ‌گویی و عمر دیسک حالت جامد بهبود یابد. در دسته‌ی دوم، تمرکز اصلی روی بارهای کاری بوده و روش‌های متعددی برای تحلیل و تشخیص بارهای کاری ارائه شده است. دسته‌ی سوم نیز مربوط به کارهایی می‌باشد که از روش‌های یادگیری ماشین برای بهینه‌سازی بخش‌های مختلف سامانه‌های کامپیوتری استفاده نموده‌اند. در ادامه‌ی این فصل، کارهای پیشین در هر کدام از این سه دسته به صورت کامل معرفی و در انتهای هر بخش با هم مقایسه شده‌اند.

### ۳-۱ حافظه‌ی نهان مبتنی بر دیسک حالت جامد

در [۱۷] الگوریتمی به نام آرک بر پایه‌ی الگوریتم سنتی حداقل استفاده در گذشته‌ی نزدیک (LRU) برای حافظه‌ی نهان مبتنی بر دیسک حالت جامد ارائه شده است. این الگوریتم به ازای یک حافظه‌ی نهان با اندازه‌ی c بلوک داده، دو لیست LRU هرکدام به طول c نگهداری می‌کند. اولین لیست حاوی بلوک‌های داده‌ای است که به تازگی و برای اولین بار دسترسی شده‌اند. دومین لیست، حاوی بلوک‌هایی

است که در گذشته حداقل ۲ بار دسترسی شده‌اند. در واقع با این کار، الگوریتم هر دو ماهیت تازگی و تعدد را به صورت مستقل در نظر می‌گیرد. در نهایت با در نظر گرفتن متغیر p که در طول اجرای برنامه به صورت برخط در حال تغییر است، همواره p بلوک داده از لیست اول و c-p بلوک داده از لیست دوم را در حافظه‌ی نهان نگه می‌دارد. این روش در بسیاری از کاربردها به دلیل انعطاف‌پذیری آن (توسط متغیر p) از الگوریتم سنتی LRU بهتر عمل کرده و افزونه‌ی پیاده‌سازی بسیار پایینی دارد. با این حال، این روش مشکلاتی همچون عدم در نظر گرفتن محدودیت‌ها و خصوصیات دیسک حالت جامد و همچنین عدم امکان بهینه‌سازی برای بارهای کاری مختلف را دارد.

در [۲] یک الگوریتم به نام لارک، با الهام گیری از الگوریتم آرک [۱۷] ارائه شده است که دارای دو لیست LRU می‌باشد. به ازای یک حافظه‌ی نهان به اندازه‌ی c بلوک داده، لیست اول دارای اندازه‌ی ثابت c و لیست دوم دارای اندازه‌ی متغیر p است. بلوک‌های موجود در لیست اول کاملا به بلوک‌های حافظه‌ی نهان منتاظر می‌شوند. وظیفه‌ی لیست دوم، جلوگیری از ورود بلوک‌های داده‌ای است که دسترسی به آنها موقتی بوده و در آینده اهمیت خود را از دست می‌دهند. به این ترتیب که به ازای اولین دسترسی به یک بلوک داده، این بلوک ابتدا وارد لیست دوم شده و اگر قبل از خروج از این لیست، دوباره به آن دسترسی صورت بگیرد، وارد لیست اصلی شده و به حافظه‌ی نهان انتقال می‌یابد. این روش در بسیاری از موارد از LRU بهتر عمل کرده و از ورود درخواست‌های نوشتن مرتبط با بلوک‌های داده‌ای که موقتا دسترسی می‌شوند به حافظه‌ی نهان جلوگیری می‌کند. یکی از معایب اصلی این روش، عدم آگاهی از بار کاری موجود است و همانطور که اشاره شده، این روش با تمرکز در افزایش نرخ رجوع و جلوگیری از ورود اولیه‌ی تمامی درخواست‌ها، در برخی بارهای کاری کارایی بسیار پایینی دارد.

در [۱۰] با تجمیع دو الگوریتم قبلی و با توجه به نقات قوت و ضعف هر یک از این الگوریتم‌ها، به تحلیل رفتار این دو الگوریتم در بازه‌های مختلف بار کاری پرداخت شده است. در این روش سه وضعیت حالت پایدار، ناپایدار و دسترسی تکی برای بار کاری تعریف و روش بهینه‌ی حافظه‌ی نهان در این سه وضعیت تعیین شده است. در ادامه این روش معیارهایی برای گذر از یک وضعیت به وضعیت دیگر بیان شده که به صورت تجربی بدست آمده است. در نهایت این روش ادعا می‌کند که می‌تواند جایگزین مناسبی برای هر یک از دو روش قبلی باشد. ضعف اصلی این روش همانند

دو روش قبلی، عدم آگاهی از بار کاری است. در این روش سه حالت تعریف شده برای بار کاری، مستقیما از تحلیل درخواست‌های ورود/خروجی بدست نیامده و بر اساس بازخوردی که از نرخ برخورد حافظه‌ی نهان گرفته می‌شود تعریف شده‌اند. از طرفی تمام معیارهای گذر به حالت‌های دیگر، نیز با توجه به همان بازخوردها از نرخ رجوع بوده و رفتار خود بار کاری مستقل از حافظه‌ی نهان در آن تأثیر داده نشده است. این موضوع باعث می‌شود که درخواست‌های ارسال شده به دیسک حالت جامد سازگاری کاملی با ویژگی‌های این دیسک نداشته باشند.

در [۱۸] یک معماری حافظه‌ی نهان با تاکید بر افزایش قابلیت اطمینان و دردسترس بودن داده‌ها حتی بعد از رخداد یک خرابی کلی در سامانه، ارائه شده است. در این روش با تجمیع تمامی فراداده‌های سامانه در حافظه‌ی نهان و انتخاب سیاست ارسال نوشتار به دیسک سخت و سپس به حافظه‌ی نهان، از بوجود آمدن بلوک‌های داده‌ی کثیف در حافظه‌ی نهان جلوگیری شده است. با واکشی همه‌ی بلوک‌های فراداده به حافظه‌ی نهان، هنگام دسترسی به هر داده، فراداده‌ی مربوط به آن سریعا از طریق حافظه‌ی نهان در دسترس خواهد بود که این موضوع باعث کاهش قابل توجه زمان‌های دسترسی می‌شود. یکی از نقاط ضعف این روش کاهش کارایی به دلیل استفاده از سیاست رونوشت به صورت پیش‌فرض است. از طرفی، بارهای کاری استفاده شده برای آزمون این روش استاندارد نیست و تجمیع تمامی فراداده‌ها در حافظه‌ی نهان لزوما کارایی را بخصوص در بارهای کاری تجاری افزایش نخواهد داد.

در [۱] روشی متفاوت با سایر کارهای پیشین در زمینه‌ی حافظه‌ی نهان مبتنی بر دیسک‌های حالت جامد ارائه شده است. در این روش برخلاف تمامی روش‌های قبلی، دیگر از لیست‌های LRU برای حافظه‌ی نهان استفاده نشده و معیار انتخاب بلوک‌ها برای واکشی در حافظه‌ی نهان تعداد دسترسی‌های بلوک‌ها می‌باشد. علاوه بر این، با اختصاص اولویت بیشتر به درخواست‌های فراداده، اهمیت درخواست‌های فراداده نیز در تصمیم‌گیری‌ها لحاظ شده است. این روش در بسیاری از بارهای کاری از روش‌های بر پایه‌ی LRU کارایی مناسب‌تری دارد. با این حال مشکلاتی همچون عدم آگاهی از بار کاری و عدم توجه به خصوصیات دیسک حالت جامد در آن نیز وجود دارد. از طرفی اولویت‌دهی به فراداده‌ها نسبت به تمامی بلوک‌های داده، همواره روش بهینه‌ای برای حافظه‌ی نهان نخواهد بود زیرا اگر در بارکاری تعداد فراداده‌های دسترسی شده بسیار بالا باشد ولی تعداد

دسترسی به این فراداده‌ها نسبت به بسیاری از داده‌های موجود کم باشد، این روش همواره از واکشی داده‌ها و جایگزینی فراداده‌ها با این داده‌ها جلوگیری خواهد کرد.

در [۱۹] با استناد به وجود مجاورت مکانی در بارهای کاری سطح دیسک، با واکشی هر بلوک داده، بلوک‌های مجاور آن نیز پیش‌واکشی می‌شوند. به این ترتیب، در بارهای کاری با میانگین اندازه‌ی درخواست‌های نسبتا بزرگ، این روش می‌تواند از روش‌هایی که بلوک‌های داده‌ی کوچک دارند بهتر عمل کرده و تعداد جایگزینی‌های حافظه‌ی نهان را نیز کاهش دهد. این روش تنها برای بارهای کاری با درصد درخواست‌های تصادفی پایین می‌تواند کارایی مناسبی داشته باشد و در بارهای کاری با درصد بالای درخواست‌های تصادفی بالا نه تنها کارایی پایینی خواهد داشت، بلکه پیش‌واکشی‌های نابجا تعداد جایگزینی‌ها و در نتیجه تعداد نوشتن‌های حافظه‌ی نهان را به شدت افزایش داده و عمر حافظه‌ی نهان را کاهش می‌دهد.

در [۲۰] روشی به نام لوکا ارائه شده است که در آن با سربار محاسباتی و حافظه‌ای پایین انتخاب بلوک‌های داده برای انتقال به حافظه‌ی نهان بر اساس تعداد دفعات دسترسی با سربار محاسباتی و حافظه‌ای پایین انجام می‌شود. در این روش برای اخراج یک بلوک از حافظه‌ی نهان به معیارهایی مانند نسبت هزینه‌ی خواندن به نوشتن و همینطور نسبت تعداد بلوک‌های کثیف به تعداد بلوک‌های تمیز توجه می‌کند. در لوکا هدف اصلی کاهش تعداد درخواست‌های نوشتن در حافظه‌ی نهان و در نتیجه عمر دیسک حالت جامد بدون افت کارایی حافظه‌ی نهان نسبت به یک الگوریتم LRU ساده است. مشکل اصلی این روش کارایی در حد و حتی بدتر از LRU است. از طرفی این روش تنها معیار نرخ برخورد به عنوان کارایی در نظر گرفته است و سایر معیارهای کارایی حافظه‌ی نهان در محاسبات لحاظ نشده‌اند.

در [۲۱] یک معماری جدید به نام های‌استور برای حافظه‌های نهان مبتنی بر دیسک حالت جامد ارائه شده که درآن دیسک حالت جامد به دو بخش حافظه‌ی نهان و میانگیر درخواست‌های نوشتن تقسیم می‌شود. های‌استور در سیستم عامل لینوکس و با دستکاری و شخصی‌سازی هسته‌ی آن پیاده‌سازی شده است. یکی از مشکلات این روش، استفاده از بخشی از دیسک حالت جامد به عنوان میانگیر درخواست‌های نوشتن است. این موضوع باعث کاهش شدید عمر دیسک حالت جامد به دلیل درخواست‌های نوشتن زیاد خواهد شد. از طرفی اگر تعداد درخواست‌های نوشتن در بارکاری

کم باشد، این بخش ثابت از دیسک حالت جامد هیچ استفاده‌ای نخواهد داشت. دومین مشکل این روش معیار اصلی واکشی داده به حافظه‌ی نهان است که به فراداده بودن درخواست‌های ورودی تاکید می‌کند. این موضوع کارایی سامانه را در برخی از بارهای کاری به شدت کاهش خواهد داد.

در [۲۲] به افزایش طول عمر دیسک حالت جامد که به عنوان حافظه‌ی نهان در مراکز داده استفاده می‌شود، پرداخته شده است. در این کار، شش الگوی دسترسی ارائه شده که به صورت آماری از دنباله‌های ورودی/خروجی ۳۲ مرکز داده گرفته شده‌اند. با توجه به این شش الگو، یک معماری حافظه‌ی نهان با استفاده از صف‌های داده‌ای مجازی برای جلوگیری از ورود داده‌های ناکارا به حافظه‌ی نهان، طراحی شده است. در نتایج این کار، نشان داده شده که روش ارائه شده می‌تواند تعداد درخواست‌های نوشتن ارسالی به حافظه‌ی نهان را کاهش داده و تاثیر منفی در نرخ برخورد نهایی نداشته باشد.

در [۲۳] با تحلیل و بررسی توزیع‌هایی از درخواست‌های ورودی/خروجی در سطح لایه‌ی بلوک داده‌ی ورودی/خروجی[1] که منجر به کاهش کارایی اجرای بار کاری در سامانه‌های ذخیره‌سازی می‌شوند، معماری حافظه‌ی نهان مبتنی بر دیسک حالت جامد ارائه داده است. در این روش، دنباله‌ی ورودی/خروجی از ۱۳ کارگزار ذخیره‌سازی داده مورد بررسی قرار گرفته و در نهایت نشان داده شده که معماری پیشنهادی می‌تواند با استفاده از تنها ۱۵% حجم دیسک حالت جامد نسبت به روش‌های قبلی، نرخ برخورد سامانه را به طور میانگین تا ۴۰% افزایش دهد. اشکال اصلی این روش سنجش معماری پیشنهادی در سامانه‌ای خاص و با بارهای کاری غیر استاندارد است.

در [۲۴] یک معماری حافظه‌ی نهان مبتنی بر دیسک حالت جامد برای سامانه‌های ارائه‌ی خدمات مجازی‌سازی[2] طراحی و پیاده‌سازی شده است. در این معماری، با توجه به نیازهای هر کاربر و با استفاده از سیاست‌های مربوط به مباحث کیفیت خدمت[3] اندازه‌ی حافظه‌ی نهان اختصاصی به هر کاربر تغییر می‌کند. به این ترتیب، تغییرپذیری[4] معماری حافظه‌ی نهان ارائه شده بالا بوده و با تغییر کاربری می‌تواند خود را با نیازهای کاربران تطبیق دهد. یکی از مشکلات این معماری عدم توجه به

1. Block I/O Layer
2. Virtualization
3. Quality of Service (QoS)
4. Flexibility

الگوهای دسترسی کاربران سامانه است و تنها با استفاده از بازخوردهایی از نرخ برخورد حافظه‌ی نهان، تصمیم به تغییر اندازه‌ی حافظه‌ی اختصاصی برای حافظه‌ی نهان می‌گیرد.

در [۲۵] یک معماری جدید برای دیسک حالت جامد با هدف افزایش کارایی در استفاده از آن به عنوان حافظه‌ی ذخیره‌سازی اصلی انبارهای داده[1] ارائه شده است. برای مدیریت انبارهای داده، معمولا از موتور[2] های پایگاه داده‌ی رابطه‌ای[3] استفاده می‌شود که منجبر به ارسال تعداد زیادی از درخواس‌های نوشتن دوره‌ای به سامانه‌ی ذخیره‌سازی می‌شود. این معماری با استفاده از یک حافظه‌ی نهان غیر فرار در داخل دیسک حالت جامد، تعداد درخواست‌های نوشتن ارسالی به سلول‌های فلش این دیسک را کاهش داده و با مدیریت و زمان‌بندی[4] این درخواست‌ها، در برخی از موارد، میانگین زمان پاسخ‌گویی دیسک حالت جامد را نیز افزایش می‌دهد.

در [۲۶] یک معماری جدید برای حافظه‌ی نهان مبتنی بر دیسک حالت جامد با هدف افزایش کارایی در مدیریت پایگاه‌های داده ارائه شده است. در این روش، یک سامانه‌ی مدیریت پایگاه داده[5] جدید با قابلیت دسته‌بندی درخواست‌های ورودی/خروجی بارهای کاری با بهره‌گیری از اطلاعات پایگاه داده ارائه شده است. با توجه به سطح بالا بودن اطلاعات در دسترس در لایه‌ی مدیریت پایگاه داده، دسته‌بندی ارائه شده از دقت بالایی برخوردار است. با استفاده از این دسته‌بندی، یک معماری حافظه نهان طراحی و پیاده‌سازی شده است که به صورت بهینه، کارایی اجرای بارهای کاری پایگاه داده را افزایش می‌دهد. این معماری به صورت اختصاصی برای پایگاه‌های داده ارائه شده و در صورت عدم استفاده از سامانه‌ی مدیریت پایگاه داده‌ی طراحی شده، نمی‌توان از معماری حافظه‌ی نهان نیز استفاده کرد.

در [۱۸] برخلاف روش‌های قبلی، از دیسک سخت به عنوان میانگیر[6] برای دیسک حالت جامد استفاده شده است. در این کار، برای مدیریت درخواست‌های ورودی/خروجی، از دیسک حالت جامد به عنوان حافظه‌ی اصلی که به تمامی درخواست‌های نوشتن به طور مستقیم پاسخ می‌دهد، استفاده شده است. در مقابل، درخواست‌های نوشتن توسط دیسک سخت پاسخ‌دهی می‌شوند و پس

1. Data Warehouses
2. Engine
3. Relational
4. Scheduling
5. Database Management System (DBMS)
6. buffer

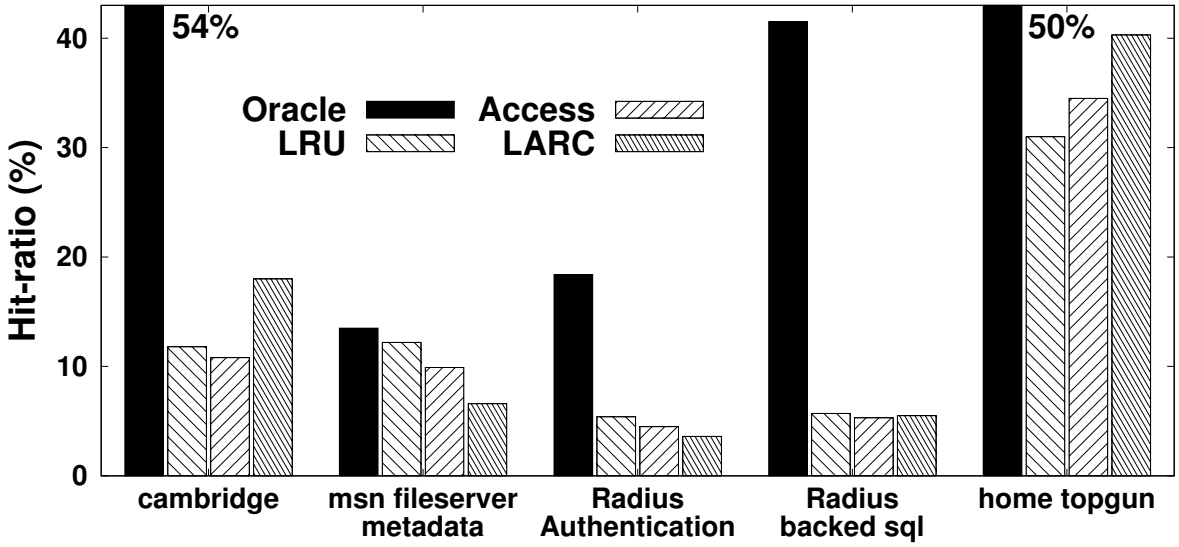
از اطمینان از عدم تغییر و یا بروزرسانی در آینده‌ای نزدیک، داده‌های مربوطه به دیسک حالت جامد ارسال می‌شوند. نتایج بدست آمده نشان می‌دهد که معماری استفاده شده می‌تواند عمر دیسک حالت جامد را تا دو برابر افزایش و در مقایسه با یک دیسک سخت، میانگین زمان پاسخ‌گویی را تا ۵۶% کاهش دهد. این روش کارایی پایین‌تری نسبت به دیگر کارهای پیشین داشته ولی عمر دیسک حالت جامد را با اختلاف زیادی افزایش می‌دهد.

برای بررسی و مقایسه‌ی کارهای پیشین در این زمینه، سه روش پرکاربرد LRU، Access و LARC در شبیه‌ساز پیاده‌سازی شدند. در روش Access، بلوک‌های داده بر اساس تعداد دفعات تکرار در درخواست‌های ورودی/خروجی گذشته مرتب‌سازی شده و بلوک‌های با تعداد دفعات دسترسی بیشتر به حافظه‌ی نهان انتقال می‌یابند. در روش LARC نیز رویکرد استفاده شده در [۲] پیاده‌سازی شده است. علاوه بر این سه روش، الگوریتم مطلوب اراکل نیز پیاده‌سازی شده تا علاوه بر مقایسه‌ی کارهای قبلی با یکدیگر، اختلاف کارایی آن‌ها با روش مطلوب نیز مشخص شود. در این آزمایش، عملکرد کارهای پیشین توسط پنج بار کاری مختلف به نام‌های cambridge، msn fileserver، metadata، Radius Authentication، Radius backed sql و home topgun از پایگاه اینترنتی اسنیا[1] [۲۷] مورد آزمایش قرار گرفت. شکل ۳-۱ نرخ برخورد هر یک از کارهای پیشین را در بارهای کاری مختلف نشان می‌دهد.

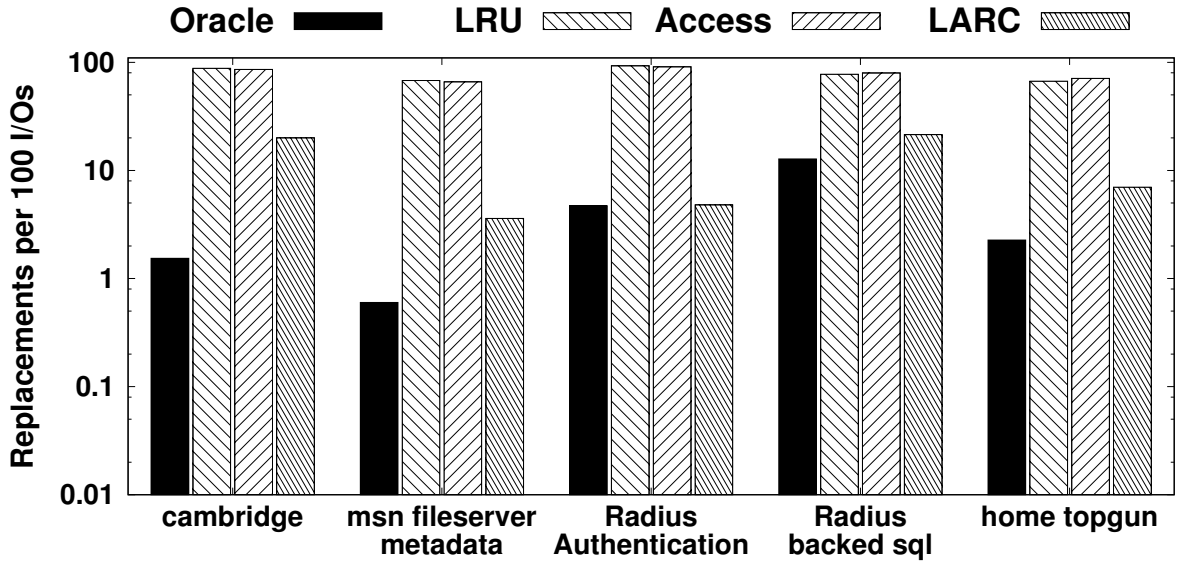
با توجه به نتایج، روش‌های پیشین در بارهای کاری مختلف تقریبا در یک سطح عمل کرده و اختلاف نرخ برخورد آن‌ها با یکدیگر در مقایسه با اختلاف این نرخ با الگوریتم اراکل ناچیز است. این نتایج نشان می‌دهند که روش‌های پیشین که متکی به مجاورت‌های مکانی و زمانی در بار کاری هستند، در مواجهه با بارهای کاری با مجاورت‌های زیاد، می‌توانند عملکرد قابل قبولی نسبت به الگوریتم مطلوب داشته باشند. اما عملکرد این روش‌ها در بارهای کاری با ارتباطات پیچیده‌تر بین درخواست‌ها، به سرعت کاهش پیدا کرده و اختلاف نرخ برخورد آن‌ها با الگوریتم مطلوب اراکل تا بیش از ۷ برابر نیز می‌رسد. این موضوع نشانگر کارایی پایین کارهای پیشین در مدیریت حافظه‌ی نهان در بارهای کاری با مجاورت‌های کم است.

در تحلیل عملکرد حافظه‌ی نهان مبتنی بر دیسک حالت جامد، علاوه بر معیار نرخ برخورد، باید

1. Storage Networking Industry Association (SNIA)




**شکل ۳-۱:** نرخ برخورد کارهای پیشین در مقایسه با الگوریتم مطلوب اراکل




**شکل ۳-۲:** تعداد عملیات جایگزینی کارهای پیشین در مقایسه با الگوریتم مطلوب اراکل

به عمر محدود دیسک حالت جامد نیز توجه داشت. لذا در آزمایش طراحی شده، تعداد دفعات جایگزینی بلوک‌های داده در حافظه‌ی نهان نیز گزارش شده است. تعداد دفعات جایگزینی کم‌تر، نشان دهنده‌ی عمل‌کرد بهینه‌تر در مدیریت حافظه‌ی نهان است که عمر دیسک حالت جامد را به دلیل کاهش تعداد نوشتن‌ها، افزایش می‌دهد. شکل ۳-۲ میانگین تعداد عملیات جایگزینی در حافظه‌ی نهان را در ازای هر ۱۰۰ درخواست ورودی/خروجی به صورت لگاریتمی نشان می‌دهد. همانطور

که در نتایج مشخص است، روش LARC در کاهش تعداد عملیات جایگزینی بسیار بهتر از دیگر روش‌ها عمل می‌کند اما در بارهای کاری همچون cambridge، تا ۱۰ برابر بیش‌تر از اراکل جایگزینی انجام می‌دهد.

این دو آزمایش در کنار هم نشان می‌دهند که الگوریتم‌های سنتی برای تعریف روش‌های حافظه‌ی نهان در سطح سامانه‌های ذخیره‌سازی داده مناسب نیستند. تمامی کارهای پیشین با اتکا به این الگوریتم‌ها، روش‌های خود را بسط داده‌اند که در نهایت باز هم محدودیت‌ها و عدم کارایی بهینه‌ی آن‌ها در مقایسه با الگوریتم مطلوب مشهود است. به عنوان مثال، در اجرای یک بار کاری مانند cambridge، الگوریتم مطلوب با بیش از ۱۰ برابر عملیات جایگزینی کمتر، به بیش از ۷ برابر نرخ برخورد بهتری دست یافته است که نشان دهنده‌ی محدودیت‌های روش‌های پیشین می‌باشد. برای برطرف کردن این محدودیت‌ها، به الگوریتم‌های پیچیده‌تر و پویا همچون روش‌های یادگیری ماشین نیاز است.

## ۳-۲ ویژگی‌شناسی بار کاری

برای مدیریت بهینه‌ی درخواست‌های ورودی/خروجی در سامانه، علاوه بر اطلاعات کارایی سخت‌افزاری، نیاز به آگاهی و دانش کافی از بارهای کاری در حال اجرا نیز وجود دارد.[۲۸، ۲۹] برای بدست آوردن این اطلاعات از بارهای کاری، تحلیل ارتباطات و خصوصیات درخواست‌های ورودی/خروجی از اهمیت بالایی برخوردار هستند.[۳۰، ۳۱، ۳۲] بارهای کاری در سطح سامانه‌های ذخیره‌سازی داده دارای ارتباطات غیرخطی و پیچیده هستند که این موضوع، تحلیل این نوع بارهای کاری را سخت‌تر کرده و تحلیل‌های آماری ساده را در تشخیص بارهای کاری ورودی/خروجی، ناکار آمد می‌سازد. [۳۳، ۳۴، ۳۵]

با توجه به خصوصیات اولیه‌ی سخت‌افزارهای ذخیره‌سازی داده و به صورت تجربی، برخی از ویژگی‌های کلی بارهای کاری ورودی/خروجی داده مشخص شده‌اند که تاثیر مستقیم در کارایی نهایی دارند. از این ویژگی‌ها می‌توان به ۱) نسبت تعداد درخواست‌های خواندن به نوشتن، ۲) اندازه‌ی متوسط درخواست‌های ورودی/خروجی و ۳) فواصل زمانی بین ارسال درخواست‌ها اشاره کرد. در ادامه به بررسی برخی از روش‌های مهم ارائه شده در زمینه‌ی ویژگی‌شناسی بار کاری می‌پردازیم. در

نهایت سه نمونه از روش‌های پرکاربرد این حوزه در شبیه‌ساز پیاده‌سازی شده و با هم در شرایط مختلف مقایسه می‌شوند تا دقیق‌ترین روش در بین کارهای پیشین مشخص شود.

در [۳۳] تمرکز اصلی بر روی افزایش کارایی سامانه‌های ذخیره‌سازی داده مبتنی بر دیسک‌های سخت بوده است. در این روش، برای دسته‌بندی بارهای کاری، ویژگی‌های کلی مانند میزان همجواری نسبی در نظر گرفته شده‌اند. از طرفی با تعریف و برچسب‌گذاری "درخواست‌های شاخص" به عنوان درخواست‌های با تاثیرگزاری منفی زیاد در کارایی کلی بار کاری، تلاش برای کنترل و مهار این‌گونه درخواست‌ها شده است. اشکال اصلی وارد بر این روش، دسته‌بندی درخواست‌های ورودی/خروجی مبتنی بر کارایی دیسک‌های سخت در پاسخ‌گویی به آن‌ها می‌باشد که استفاده از این روش را برای سامانه‌هایی با سخت‌افزار متفاوت بی‌نتیجه خواهد کرد. به طور مثال، درخواست‌هایی که به عنوان درخواست شاخص در این دسته‌بندی مشخص شده‌اند، لزوما در دیسک‌هایی مانند دیسک حالت جامد دارای کاری پایین نبوده و لذا دسته‌بندی به درستی نتیجه‌بخش نخواهد بود.

در [۲۸] روشی بر پایه‌ی رگبار[1] های بار کاری ارائه شده و با توجه به ویژگی‌هایی همچون الگوی دسترسی در دوره‌ی رگبار، میزان رگبارها و مدت زمان رگبار، به دسته‌بندی بارهای کارهایی پرداخته شده است. در این روش، بارهای کاری به سه دسته‌ی صنعتی، خانگی و دستگاه‌های الکتریکی تقسیم‌بندی شده‌اند که از لحاظ ویژگی‌های رگباری متفاوت هستند. مشکل اصلی این روش، کلی‌گرایی آن در مقایسه با کارهای دیگر است که تمام بارهای کاری به دسته‌های گسترده تقسیم شده‌اند. به طور مثال، در دسته‌ی صنعتی بیش از صدها بار کاری مختلف وجود دارد که از لحاظ الگوهای دسترسی به کلی با هم متفاوت هستند.

در [۳۲] بیش از ۱۲ دسته بار کاری از کارگزار ویندوز با ابزارهای متفاوت بررسی شده‌اند. در این کار از روش‌هایی همچون آمار ساده‌ی در سطح بلوک داده، توزیع‌های آماری چند پارامتری، رتبه‌بندی فایل‌ها بر اساس دسترسی‌ها و دیگر روش‌های ایستا مانند خودهمسانی[2] زمانی و مکانی درخواست‌ها برای ویژگی‌شناسی بارهای کاری استفاده شده است. اصلی‌ترین اشکال وارد بر این کار عدم کمی‌سازی این روش‌ها برای پیاده‌سازی ویژگی‌شناسی به صورت برخط است. در واقع این کار،

1. Burst
2. Self-Similarity

کاملا به صورت برون‌خط انجام گرفته و در مجموع یک تحلیل دقیق از بارهای کاری مورد بررسی است.

در [۳۴] یک ابزار گزارش‌گیری برای استفاده در ویژگی‌شناسی بارهای کاری ورودی/خروجی ارائه شده است. هدف اصلی از این ابزار، سربار پایین آن در گزارش‌گیری از سامانه‌های ذخیره‌سازی داده با بیش از چند هزار فرآیند است. این ابزار، مواردی همچون تعداد دسترسی‌ها به پوشه‌ها، میانگین اندازه‌ی درخواست‌های نوشتن و خواندن، الگوی دسترسی‌های یک فرآیند و دسته‌بندی‌های کلی درخواست‌ها مانند وابسته یا مستقل را به کاربرا ارائه می‌دهد تا بتوانند سامانه‌ی ذخیره‌سازی خود را بر اساس این اطلاعات پیکربندی کنند. ضعف اصلی این روش، عدم امکان استفاده برای تشخیص بارهای کاری از یکدیگر است. لذا کاربران باید به صورت دستی و بدون ابزاری پویا با اطلاع از بارهای کاری موجود، سامانه‌ی ذخیره‌سازی را پیکربندی کنند.

در [۳۱] ابزار محک[1] جدید برای مدل کاهش نگاشت[2] در چهارچوب Hadoop ارائه شده است. توسط این ابزار محک، تحلیل‌های بارهای کاری با روش‌های آماری همچون زمان پاسخ‌گویی، پهنای باند نهایی و میزان استفاده‌ی بهینه از پردازنده و حافظه‌ی اصلی و دیگر اجزای سامانه انجام گرفته است. این ابزار یکی از اولین ابزار ارائه شده در زمینه‌ی کاهش نگاشت بوده ولی روش‌های تحلیلی ساده، نقطه ضعف اصلی این کار می‌باشد.

در [۳۵] تمرکز اصلی روی بارهای کاری کارگزاران وب بوده که شش بار کاری متفاوت مورد بررسی و تحلیل قرار گرفته است. با توجه به نتایج بدست آمده از این بررسی‌ها، دو روش متفاوت حافظه‌ی نهان برای کارگزاران وب ارائه شده است. اصلی‌ترین مشکل این روش، ساده‌نگری و استفاده از تحلیل‌های آماری ساده در ویژگی‌شناسی بار کاری است.

در [۳۶] روشی برای حافظه‌ی نهان مبتنی بر دیسک حالت جامد ارائه شده که از ویژگی‌شناسی بار کاری برای افزایش کارایی استفاده می‌کند. در این روش چهار صفت ۱) پی‌درپی، ۲) هم‌پوشان[3] ۳) گام‌دار[4] و ۴) تصادفی[5] برای درخواست‌های ورودی/خروجی تعریف شده است. سپس بارهای

1. Benchmark
2. MapReduce
3. Overlapped
4. Strided
5. Random

**جدول ۳-۱ :** سناریوهای سامانه‌های ذخیره‌سازی داده

| نام سناریو | تعداد بارهای کاری | بارهای کاری |
|---|---|---|
| متمرکز | ۲ | Radius_Auth<br>mail_index |
| نیمه‌ـ متمرکز | ۳ | home_ikki<br>Radius_Auth<br>mail_index |
| عمومی | ۴ | enterprise_tpc_۱<br>home_ikki<br>Radius_Auth<br>mail_index |

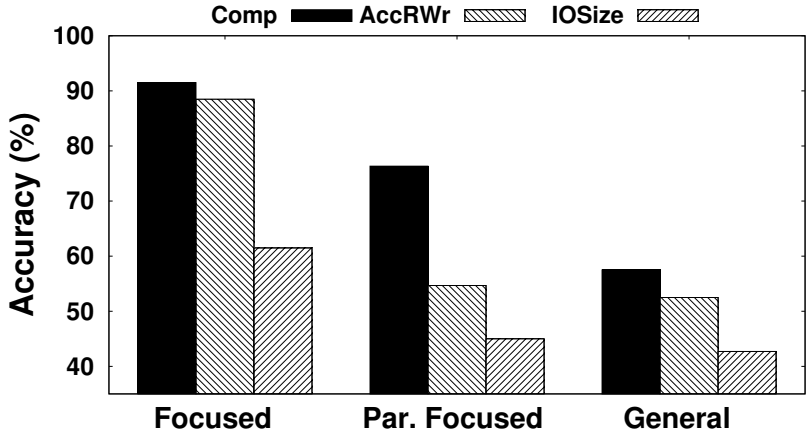
کاری با توجه به میزان نرخ این چهار نوع درخواست، با هم مقایسه شده‌اند. این روش نسبت به روش‌های قبلی از دقت بیش‌تری، به دلیل تمرکز بر روی الگوهای دسترسی مختلف، برخوردار است. البته اصلی‌ترین محدودیت این کار مانند سایر کارهای پیشین، بهره‌گیری از روش‌های آماری ساده در تحلیل‌های بارهای کاری است.

برای بررسی نحوه‌ی عملکرد کارهای پیشین و مقایسه‌ی روش‌های ارائه شده با یکدیگر، سه رویکرد کلی در تحلیل بارهای کاری به نام‌های Comp، AccRWr و IOSize در شبیه‌ساز پیاده‌سازی شدند که به ترتیب از [۳۶]، [۱] و [۲] الهام گرفته شده‌اند. در روش IOSize، تنها معیار تشخیص درخواست‌های ورودی/خروجی، اندازه‌ی آن‌ها می‌باشد. در روش AccRWr علاوه بر اندازای درخواست‌ها به فراوانی نوع درخواست‌ها (نوشتن یا خواندن) نیز توجه می‌شود. در نهایت در روش Comp، علاوه بر اطلاعات روش‌های قبلی، نرخ هر کدام از چهار صفت ۱) پی‌درپی، ۲) هم‌پوشان ۳) گام‌دار و ۴) تصادفی نیز در بارهای کاری بررسی می‌شوند.

برای آزمایش روش‌های پیاده‌سازی شده، سه سناریو با هدف شبیه‌سازی سامانه‌های مختلف ذخیره‌سازی داده در شرایط متفاوت تعریف شدند. در این سناریوها با استفاده از دنباله‌های بارهای

کاری موجود در پایگاه اینترنتی اسنیا [۲۷] طراحی شده‌اند. جدول ۳-۱ اطلاعات دقیق مربوط به این سه سناریو را نمایش می‌دهد. سناریوی متمرکز شامل تنها دو بار کاری مختلف است، در حالی که سناریوهای نیمه‌ـ متمرکز و عمومی به ترتیب شامل سه و چهار بار کاری مختلف هستند. هر چه تعداد بارهای کاری در حال اجرا در سامانه افزایش پیدا کند، ویژگی‌شناسی آن‌ها دشوارتر و در نتیجه دقت دسته‌بندی و در نهایت قدرت تشخیص بار کاری در حال اجرا کاهش پیدا می‌کند.

با استفاده از شبیه‌ساز پیاده‌سازی شده، بخشی از هر بار کاری به عنوان نمونه‌ای برای تحلیل توسط هر روش قرار گرفت. در نهایت هر روش با توجه به بخش دیگر همان بار کاری در کنار بارهای کاری دیگر مورد سنجش قرار گرفت. شکل ۳-۳ درصد دقت روش‌های پیاده‌سازی شده را در این سه سناریو نشان می‌دهد. همانطور که در نتایج مشخص است، هنگامی که تعداد بارهای کاری افزایش می‌یابد، دقت روش‌های پیشین که مبتنی بر تحلیل‌های آماری ساده هستند، به شدت کاهش پیدا می‌کند. در بین این سه روش، Comp از دیگر روش‌ها عملکرد بهتری داشته که این موضوع نتیجه‌ی تحلیل دقیق‌تر در الگوهای دسترسی این روش است. البته در سناریویی مانند سناریوی عمومی، دقت عمل هیچ یک از این روش‌ها به ۶۰% نمی‌رسد که دقت قابل قبولی نیست. این نتایج نشان می‌دهند که برای تحلیل بهتر بارهای کاری نیاز به روش‌های پیچیده‌تر و دقیق‌تری داریم.




**شکل ۳-۳:** درصد دقت روش‌های ویژگی‌شناسی پیشین در سناریوهای مختلف

## ۳-۳ استفاده از یادگیری ماشین در معماری کامپیوتر

در این بخش به بررسی کارهایی پرداخته می‌شود که با استفاده از یادگیری ماشین، تلاش می‌کنند تا کارایی بخشی از سامانه‌های کامپیوتری را در سطوح مختلف بهبود دهند. به دلیل میانگین زمان پاسخ‌گویی بسیار بالاتر در سطح ادوات ذخیره‌سازی نسبت به پردازنده و حافظه‌ی اصلی، می‌توان در زمان‌های انتظار، الگوریتم‌های به مراتب پیچیده‌تری نسبت به سطوح پردازنده و حافظه‌ی اصلی اجرا کرد. لذا استفاده از الگوریتم‌هایی با افزونگی زمانی بالا همچون الگوریتم‌های یادگیری ماشین در این سطح از سامانه امکان‌پذیر می‌باشد. در ادامه‌ی این بخش به برخی کارهای پیشین که سعی بر استفاده از راه‌کارهای یادگیری ماشین در سامانه‌های کامپیوتری داشتند، اشاره شده است.

در [۳۷] با بهره‌گیری از ابزار خوشه‌بندی[1] ، مساله‌ی پیش‌واکشی در یک کارگزار صنعتی بررسی شده است. در این پژوهش، دو رویکرد کلی پیش‌واکشی[2] سخت‌افزاری و پیش‌واکشی نرم‌افزاری (به کمک هم‌گردان[3] ) ارائه شده که از یادگیری ماشین برای بهینه‌سازی و تبدیل فازهای بارکاری بهره می‌برد. در انتها نشان داده شده که روش ارائه شده می‌تواند در موارد مختلفی مانند تشخیص الگوهای دسترسی در سامانه‌های دارای چند پردازه، یافتن فازهای هر یک از برنامه‌ها، کنترل و تنظیم خودکار معیارهای متغیر سامانه‌ی پیش‌واکشی و قابلیت بهینه‌سازی در جهات مختلف مانند کارایی یا کاهش انرژی، کارا باشد.

در [۳۸] با تحلیل حجم بسیار بزرگی از دنباله‌های ترافیک شبکه، روشی با استفاده از ابزارهای یادگیری ماشین برای ویژگی‌شناسی این دنباله‌ها و پیش‌بینی حملات نفوذ به شبکه ارائه شده است. برای رسیدن به دقت بالا، نیاز به بررسی دنباله‌های بسیار بزرگ از ترافیک شبکه بوده که برای ویژگی‌شناسی این حجم از داده نیاز به یادگیری ماشین برای دسته‌بندی دنباله‌ها و تحلیل رفتار شبکه می‌باشد. این مقاله به بررسی روش‌های یادگیری با نظارت، یادگیری نمایش‌های هندسی و روش‌های مدرن داده‌های حجیم می‌پردازد.

در [۳۹] با استفاده از یادگیری ماشین، روش‌های متفاوتی برای پیش‌بینی درخواست‌های منابع

1. Clustering
2. Prefetching
3. Compiler

**جدول ۳-۲:** مقایسه‌ی روش‌های بهره‌گیری از یادگیری ماشین در معماری کامپیوتر

| مرجع | حوزه‌ی کاری | راه‌کار |
|---|---|---|
| [۳۷] | کارگزار صنعتی | پیش‌واکشی — سامانه‌های چند پردازه |
| [۳۸] | شبکه | تحلیل ترافیک و پیش‌بینی حملات نفوذ |
| [۳۹] | رایانش ابری | پیش‌بینی درخواست مشتریان |
| [۴۰] | حافظه‌ی نهان وب | شبکه‌ی عصبی — یادگیری روش مطلوب ـ برون‌خط |

مشتری‌های سامانه‌های ابری ارائه شده است. در این روش با تحلیل دنباله‌های به دست آمده از بارکاری TCP-W مدل‌های پیش‌بینی توسط سه روش ۱) شبکه عصبی ۲) ماشین بردار پشتیبانی[1] و ۳) پس‌گرایی خطی[2] ایجاد شده و با تحلیل نتایج بدست آمده از آزمون‌های نهایی این سه روش باهم مقایسه شده‌اند. در انتها نشان داده شده که در بارکاری مورد بررسی روش ماشین بردار پشتیبانی می‌تواند بهتر از دیگر روش‌ها عمل کند.

در [۴۰] برای رسیدن به نرخ برخورد مطلوب در حافظه‌ی نهان سامانه‌های اینترنتی، از شبکه‌های عصبی استفاده شده است. در این روش، با بررسی مطلوب‌ترین شرایط برای پیکربندی حافظه‌ی نهان به صورت برون‌خط، شبکه‌ی عصبی تکرار شونده‌ی ساده با حافظه تک لایه، مراحل یادگیری لازم را طی کرده و در مقایسه با کارهای پیشین در زمینه‌ی حافظه‌ی نهان کارگزاران وب، به کارایی بهتری دست یافته است.

جدول ۳-۲ کارهای پیشین در زمینه‌ی بهره‌گیری از یادگیری ماشین در معماری کامپیوتر را با هم مقایسه می‌کند. با توجه به تفاوت در حوزه‌ی کاری، روش‌های مختلفی از کاربردهای یادگیری ماشین برای افزایش کارایی به کار رفته است.

1. Supprt Vector Machine
2. Linear Regression

## ۳-۴ جمع‌بندی

در این فصل، کارهای پیشین مرتبط با این پژوهش در سه حوزه‌ی متفاوت بررسی شده‌اند. در دسته‌ی اول از کارهای پیشین، معماری‌های متفاوت برای افزایش کارایی حافظه‌ی نهان مبتنی بر دیسک‌های حالت جامد بررسی شده‌اند. هر یک از کارهای پیشین در این حوزه، با توجه به نرخ برخورد، زمان پاسخ‌گویی و عمر دیسک حالت جامد از روش‌های متفاوتی استفاده کرده که در نهایت با کارایی مطلوب مقایسه شده‌اند. در دسته‌ی دوم، روش‌های متعدد برای تحلیل و ویژگی‌شناسی بارهای کاری مورد بررسی قرار گرفته و کارایی آن‌ها در تشخیص بارهای کاری با هم مقایسه شده است. کارهای مرتبط در دسته‌ی سوم از روش‌های یادگیری ماشین برای بهبود کارایی بخش‌های مختلف سامانه‌های کامپیوتری استفاده کرده‌اند.

# فصل ۴

## معماری پیشنهادی

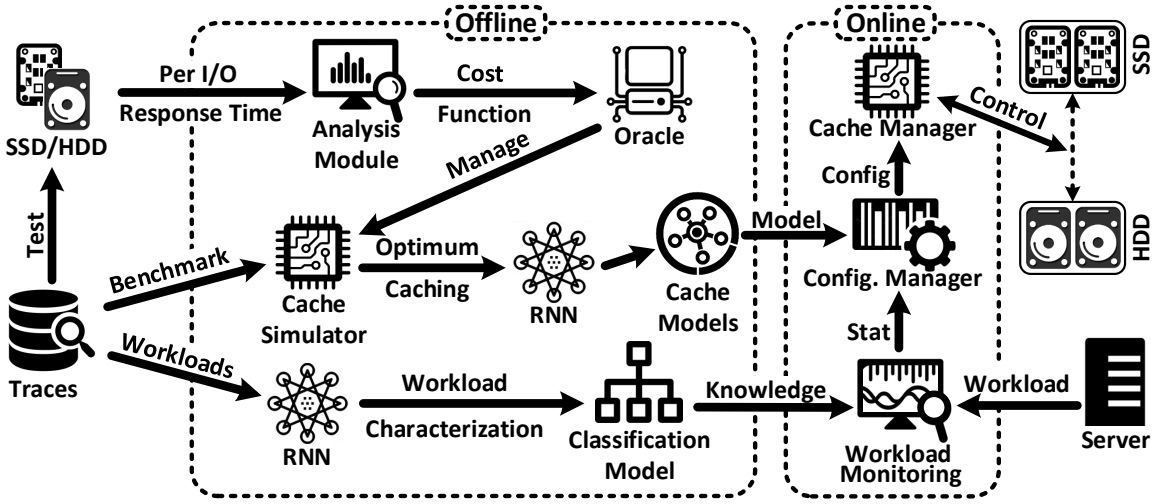
هدف از این پژوهش، طراحی یک حافظه‌ی نهان مبتنی بر دیسک حالت جامد است که می‌تواند در بارهای کاری مختلف، کارایی بهینه‌ای داشته باشد. از این رو یک معماری حافظه‌ی نهان جدید، با قابلیت بازپیکربندی ارائه شده که قابلیت تشخیص تغییرات بار کاری را با استفاده از روش‌های یادگیری ماشین داشته و سیاست‌های حافظه‌ی نهان را بر اساس آن تغییر می‌دهد. علاوه بر این، سیاست‌های حافظه نهان در روش پیشنهادی توسط یادگیری ماشین اعمال شده که این روش برای اولین بار در حافظه‌ی نهان مبتنی بر دیسک حالت جامد مورد استفاده قرار می‌گیرد.

در معماری پیشنهادی، برای ویژگی‌شناسی بارهای کاری در حال اجرا از شبکه‌ی عصبی تکرار شونده‌ی ساده استفاده می‌شود. در این روش، شبکه‌ی عصبی به صورت دوره‌ای بارهای کاری را تحلیل و دانش خود را نسبت به بارهای کاری سامانه وسیع‌تر می‌کند. پس از تشخیص نوع بار کاری در حال اجرا، یک شبکه‌ی عصبی تکرار شونده‌ی عمیق که دارای حافظه‌ی سه لایه بوده و برای آن نوع از بار کاری خاص بهینه‌سازی شده است، مدیریت حافظه‌ی نهان را به عهده می‌گیرد. بهینه‌سازی مدل‌های شبکه‌ی عصبی تکرار شونده‌ی عمیق، با یادگیری رفتار الگوریتم مطلوب اراکل صورت می‌گیرد.

معماری پیشنهادی شامل دو مرحله‌ی برون‌خط[۲] و برخط[۳] می‌باشد که مرحله‌ی برون‌خط آن تنها یکبار انجام شده و مرحله‌ی برخط به صورت مداوم در حال اجرا خواهد بود. شکل ۴-۱ روند کلی و

2. Offline
3. Online

ارتباط این دو مرحله را نشان می‌دهد. در ادامه‌ی این فصل، هر یک از این بخش‌ها با جزئیات شرح داده می‌شوند.




**شکل ۴-۱:** ساختار دو مرحله‌ای معماری پیشنهادی

## ۴-۱ مرحله‌ی برون‌خط

همانگونه که در شکل ۴-۱ نشان داده شده است، این مرحله با ویژگی‌شناسی دنباله‌های جمع‌آوری شده از سامانه آغاز می‌گردد. برای انجام این ویژگی‌شناسی، تمامی دنباله‌های بارهای کاری ورودی/خروجی موجود در پایگاه اینترنتی اسنیا [۲۷] به دنباله‌های قابل درک شامل چهار مقدار ۱) زمان دسترسی، ۲) آدرس شروع درخواست، ۳) اندازه‌ی درخواست و ۴) نوع درخواست تبدیل شدند. دنباله‌های جدید با استفاده از یک مدل شبکه‌ی عصبی تکرار شونده‌ی ساده یادگیری می‌شوند. نام این مدل، مدل دسته‌بندی[1] است. پس از یادگیری تمامی بارهای کاری مجموعه و تفاوت‌های آن‌ها، این مدل قادر به تشخیص یک بار کاری جدید و درک ارتباط آن با بارهای کاری یاد گرفته شده خواهد بود. این موضوع به معماری پیشنهادی امکان تشخیص تغییر بار کاری در حال اجرا در سامانه را خواهد داد که می‌تواند آغازگر عملیات بازپیکربندی باشد.

از طرف دیگر، تمامی دنباله‌های بارهای کاری جمع‌آوری شده با سخت‌افزار واقعی آزمایش شده

1. Classification

و زمان پاسخ‌گویی هر درخواست در هر دو دیسک سخت و حالت جامد ثبت می‌شود. بدین ترتیب، اطلاعات آماری کاملی از نحوه‌ی عملکرد هر دیسک در انواع درخواست‌ها در دسترس خواهد بود. این اطلاعات به عنوان یکی از ورودی‌های اصلی تابع اولویت[1] در الگوریتم مطلوب اراکل استفاده خواهد شد. به این ترتیب، تصمیم نهایی برای واکشی یا عدم واکشی یک بلوک داده به حافظه‌ی نهان، ارتباط مستقیم با نسبت زمان پاسخ‌گویی دیسک سخت به زمان پاسخ‌گویی دیسک حالت جامد برای آن بلوک داده، دارد.

برای بهینه‌سازی الگوریتم مطلوب اراکل برای حافظه‌ی نهان مبتنی بر دیسک حالت جامد، نیاز به تعریف تابع اولویت خاص وجود دارد. در این نوع حافظه‌ی نهان، علاوه بر نرخ برخورد، باید به مواردی همچون عمر محدود دیسک حالت جامد نیز توجه داشت. برای افزایش طول عمر دیسک‌های حالت جامد، نیاز است تا تعداد عملیات نوشتن ارسالی به دیسک تا حد ممکن کاهش یابد. لذا در تابع اولویت طراحی شده، نوع درخواست‌های ورودی/خروجی (نوشتن یا خواندن) در آینده نیز برای هر بلوک داده در نظر گرفته شده است. معادله‌ی ۴-۱ مدل ریاضی تابع اولویت پیشنهادی را نشان می‌دهد. در این معادله، $T_{SSD}$ و $T_{HDD}$ به ترتیب زمان پاسخ‌گویی به درخواست را توسط دیسک حالت جامد و دیسک سخت نشان می‌دهند. هر کدام از $N_{reads}$ و $N_{acc}$ نیز به ترتیب تعداد درخواست‌های خواندن و کل درخواست‌های بلوک مورد نظر در آینده را مشخص می‌کنند. با استفاده از مقدار $\frac{1}{Req_{size}}$ که معکوس اندازه‌ی درخواست ورودی/خروجی است، احتمال واکشی درخواست‌های بزرگتر به حافظه‌ی نهان کاهش یافته است. از طرفی برای جلوگیری از ورود درخواست‌هایی به حافظه‌ی نهان که در آینده تنها یک بار دسترسی خواهند شد (مانند درخواست‌های نوشتن ـ و ـ سنجش) مقدار $(N_{acc}-1)$ در نظر گرفته شده است. مقدار $N_{acc}$ در کم‌ترین حالت برابر با ۱ است که نشان دهنده‌ی عدم درخواست بلوک در آینده است. در این شرایط، مقدار $(N_{acc}-1)$ برابر با صفر شده تا از انتقال بلوک داده به حافظه‌ی نهان جلوگیری شود. با استفاده از نسبت $\frac{N_{reads}}{N_{acc}}$، به بلوک‌های داده با تعداد درخواست‌های خواندن بیشتر نسبت به درخواست‌های نوشتن در آینده، اولویت بیشتری داده می‌شود تا در نهایت درخواست‌های خواندن بیشتری به دیسک حالت جامد ارسال شود. به این ترتیب، علاوه

1. Priority Function

بر کاهش زمان پاسخ‌گویی سامانه، طول عمر دیسک حالت جامد نیز افزایش می‌یابد.

$$benefit = \frac{T_{HDD}}{T_{SSD}} \times (N_{acc} - ۱) \times \frac{۱}{Req_{size}} \times (۱ + \frac{N_{reads}}{N_{acc}}) \quad (۱-۴)$$

با استفاده از تابع اولویت پیشنهادی، الگوریتم اراکل می‌تواند به بهترین روش حافظه‌ی نهان مبتنی بر دیسک حالت جامد دست پیدا کند. الگوریتم اراکل در شبیه‌ساز برپایه‌ی دنباله پیاده‌سازی شده و ورودی‌های تابع اولویت توسط همان دنباله‌ی در حال اجرا به صورت پویا مقداردهی می‌شوند. تمامی تصمیمات اراکل به ازای هر درخواست ورودی/خروجی توسط شبیه‌ساز گزارش‌گیری شده و یک دنباله‌ی جدید شامل اطلاعات مربوط به هر درخواست و نحوه‌ی تصمیم‌گیری اراکل تولید می‌شود. گزارش تصمیمات اراکل شامل سه مولفه می‌باشد:

- **واکشی:** مشخص می‌کند که آیا الگوریتم اراکل درخواست را واکشی یا رد کرده است.
- **طول عمر در حافظه‌ی نهان:** در صورت واکشی درخواست، مشخص می‌کند که بلوک داده مربوطه بعد از چند درخواست از حافظه‌ی نهان اخراج شده است.
- **سیاست نوشتن**[۱] **:** در صورت واکشی درخواست از نوع نوشتن، سیاست نوشتن ( رونوشت[۲] یا پس‌نوشت[۳] ) را مشخص می‌کند.

برای کاهش ابعاد دامنه‌های ورودی یادگیری شبکه‌ی عصبی تکرار شونده، سه ثابت عددی بر اساس اندازه‌ی حافظه‌ی نهان برای دسته‌بندی پارامتر طول عمر در حافظه‌ی نهان در نظر گرفته شد که مقدار عددی این پارامتر را به یک بازه‌ی تقریبی تغییر داده و تمامی داده‌های این پارامتر را به سه دسته‌ی ۱) کوتاه، ۲) متوسط و ۳) بلند تغییر دهد. جدول ۴-۱ اطلاعات مربوط به این دسته‌بندی را نشان می‌دهد.

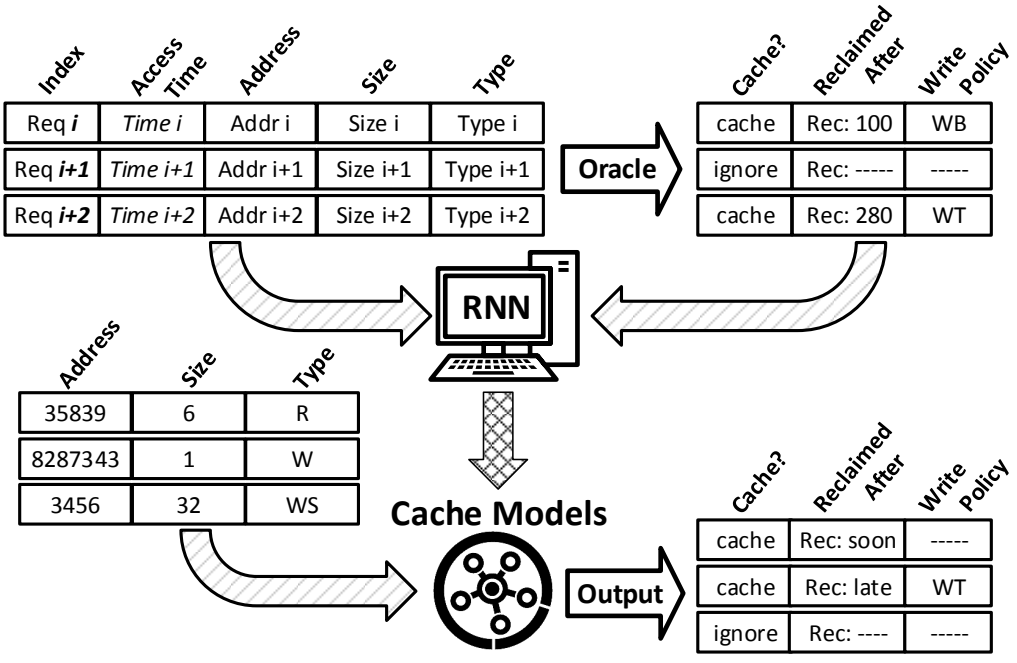
شکل ۴-۲ تمامی اطلاعات مربوط به دنباله‌های ورودی به اراکل و دنباله‌ی متشکل از تصمیمات اراکل را نشان می‌دهد. همانطور که در شکل مشخص است، مجموع این دو دنباله به عنوان دنباله‌ی ورودی شبکه‌ی عصبی تکرار شونده برای یادگیری به مدلی عمیق با حافظه‌ی سه لایه از این شبکه

1. Write Policy
2. Write-Through
3. Write-Back

**جدول ۴-۱:** جدول دسته‌بندی عمر بلوک‌های داده در حافظه‌ی نهان

| نام دسته | مقدار عددی عمر دقیق بلوک |
|---|---|
| *کوتاه* | $x <$ *اندازه‌ی حافظی نهان* |
| *متوسط* | *اندازه‌ی حافظی نهان* $\times ۵ < x <$ *اندازه‌ی حافظی نهان* |
| *بلند* | $x >$ *اندازه‌ی حافظی نهان* $\times ۵$ |

ارسال می‌شوند. این مدل پس از یادگیری رفتار الگوریتم اراکل در بار کاری، توسط بار کاری جدیدی از همان نوع سنجیده می‌شود تا میزان دقت یادگیری بدست آید. نتیجه‌ی نهایی این روش، تعدادی مدل شبکه‌ی عصبی است که هر کدام برای بار کاری خاصی بهینه‌سازی شده‌اند. این مدل‌ها در بخش برخط معماری پیشنهادی، کنترل درخواست‌های ورودی/خروجی را به عهده می‌گیرند.




**شکل ۴-۲:** دنباله‌های ورودی و خروجی اراکل و شبکه‌ی عصبی تکرار شونده

## ۴-۲ مرحله‌ی برخط

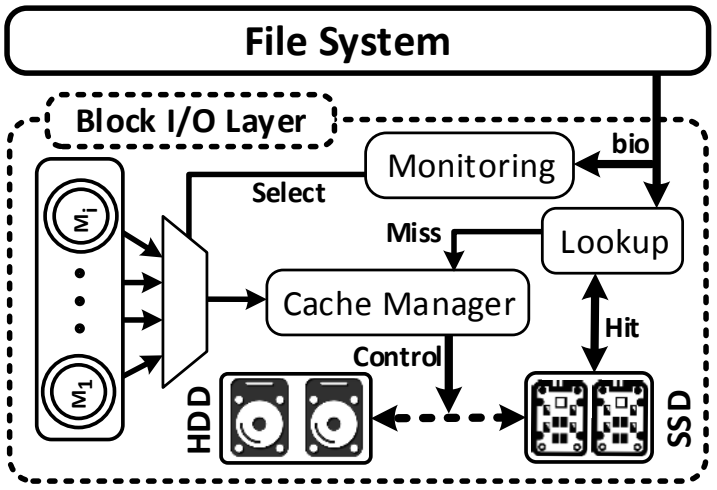
این مرحله، برخلاف مرحله‌ی برون‌خط، به طور مداوم در حال اجرا می‌باشد. در طراحی مولفه‌های این مرحله، تمرکز اصلی بر پایه‌ی کاهش سربار چه از نظر حافظه و چه از نظر محاسبات بوده تا عملیات بازپیکربندی در حافظه‌ی نهان با کمترین تاخیر انجام گیرد. همانطور که در شکل ۴-۱ مشخص است، در این مرحله سه مولفه‌ی اصلی ۱) ناظر بار کاری[1] ، ۲) مدیر پیکربندی[2] و ۳) مدیر حافظه‌ی نهان[3] ، تمامی عملیات لازم برای مدیریت کلی درخواست‌های ورودی/خروجی و بازپیکربندی را به عهده دارند.

در ابتدای دریافت یک درخواست جدید از کاربران سامانه، ناظر بار کاری اطلاعات مربوط به درخواست را ثبت می‌کند. این مولفه با استفاده از مدل دسته‌بندی (که در مرحله‌ی برون‌خط یادگیری و آماده شده بود)، به صورت دوره‌ای به بررسی و ویژگی‌شناسی اطلاعات جمع‌آوری شده از بارهای کاری می‌پردازد. این موضوع کمک می‌کند تا سامانه توانایی تشخیص تغییر در بار کاری را داشته باشد. در صورت تغییر بار کاری در حال اجرا، این مولفه اطلاعات جدید را برای مدیر پیکربندی ارسال می‌کند.

مدیر پیکربندی مسوولیت انتخاب مدل‌های بهینه‌سازی شده در مرحله‌ی برون‌خط را بر عهده دارد. این مولفه با توجه به اطلاعات دریافتی از ناظر بار کاری، یکی‌از مدل‌های شبکه‌ی عصبی را برای مدیریت حافظه‌ی نهان انتخاب می‌کند. شکل ۴-۳ نحوه‌ی پیاده‌سازی و ارتباط بین این مولفه‌ها را نشان می‌دهد. همانطور که در شکل مشخص است، پس از تشخیص نوع بار کاری، متناسب‌ترین مدل از چندین مدل موجود انتخاب و برای مدیر حافظه‌ی نهان ارسال می‌گردد.

مدیر حافظه‌ی نهان، مسوولیت ارسال درخواست ورودی/خروجی به مدل انتخابی توسط مدیر پیکربندی را دارد. به ازای هر درخواست، خروجی مدل شامل سه پارامتر واکشی، عمر بلوک در حافظه‌ی نهان و سیاست نوشتن در صورت واکشی خواهد بود که در شکل ۴-۲ با جزئیات کامل نمایش داده شده‌اند. در صورتی که درخواست جدید در حافظه‌ی نهان نباشد و مدل شبکه‌ی عصبی

1. Workload Monitoring
2. Reconfiguration Manager
3. Cache Manager




**شکل ۴-۳:** پیاده‌سازی و نحوه‌ی ارتباط بین مولفه‌های مرحله‌ی برخط

نظر به واکشی بلوک داده‌ی مربوطه داشته باشد، نیاز است تا بلوک داده‌ای از حافظه‌ی نهان اخراج شود. در ساختار مدیر حافظه‌ی نهان، سه صف LRU برای بلوک‌های حافظه‌ی نهان با توجه به دسته‌بندی عمر بلوک‌ها (کوتاه، متوسط و بلند) توسط مدل شبکه‌ی عصبی وجود دارد. مدیر شبکه برای انتخاب بلوکی از حافظه‌ی نهان برای اخراج، ابتدا به سراغ صف مربوط به بلوک‌های با عمر کوتاه می‌رود. اگر صف خالی بود، نوبت به صف بلوک‌های با عمر متوسط می‌رسد. در انتها در صورت خالی بودن هر دو صف، مدیر حافظه‌ی نهان آخرید بلوک از صف بلوک‌های با عمر بلند را به عنوان اخراجی انتخاب می‌کند. به این ترتیب، در صورت تشخیص طول عمر بلندتر یک بلوک موجود در حافظه‌ی نهان توسط مدل شبکه‌ی عصبی، بلوک داده در واقعیت نیز مدت زمان بیشتری را در حافظه‌ی نهان باقی خواهد ماند.

## ۴-۳ جمع‌بندی

در این فصل، معماری پیشنهادی برای حافظه‌ی نهان مبتنی بر دیسک جامد در سامانه‌های ذخیره‌سازی داده با استفاده از یادگیری ماشین تحلیل گردید. معماری پیشنهادی از دو بخش کلی برخط و برون‌خط تشکیل شده‌است. در بخش برون‌خط که تنها یکبار اجرا می‌شود، بارهای کاری سامانه مورد تحلیل

و بررسی توسط شبکه‌ی عصبی قرار می‌گیرند. علاوه بر این، الگوریتم مطلوب اراکل، گزارشی از تصمیمات حافظه‌ی نهان برای رسیدن به بهترین کارایی را تویلد می‌کند. خروجی الگوریتم اراکل، به همراه اطلاعات کامل از تمامی درخواست‌های بارهای کاری، توسط شبکه‌ی عصبی عمیق یاد گرفته می‌شوند. خروجی ای مرحله، تعدادی مدل شبکه‌ی عصبی بوده که در مرحله‌ی برخط، کنترل حافظه‌ی نهان را به عهده می‌گیرند. در مرحله‌ی برخط، بارهای کاری به صورت دوره‌ای توسط مدل شبکه‌ی عصبی بررسی شده و با توجه به بار کاری در حال اجرا، پیکربندی حافظه‌ی نهان تغییر می‌کند.

# فصل ۵

# پیاده‌سازی و نتایج

در این بخش ابتدا به بررسی روش پیاده‌سازی مولفه‌های مختلف معماری پیشنهادی می‌پردازیم. سپس روش‌های سنجش عملکرد این مولفه‌ها چه از لحاظ دقت و چه از نظر کارایی بیان شده و نتایج بدست آمده تحلیل می‌شوند.

## ۵-۱ پیاده‌سازی معماری پیشنهادی

پیاده‌سازی روش پیشنهادی نیازمند یک بستر شبیه‌سازی با قابلیت ارسال درخواست‌های دنباله‌های ورودی/خروجی به سامانه بود. لذا با استفاده از زبان برنامه‌نویسی C++ بستر شبیه‌سازی حافظه‌ی نهان پیاده‌سازی شد. علت استفاده از زبان C++ سرعت بالای این زبان نسبت به دیگر زبان‌ها است. بستر پیاده‌سازی شده در شبیه‌ساز، قابلیت تعریف انواع سیاست‌های حافظه‌ی نهان را به صورت پیمانه‌ای[۲] دراد و لذا می‌توان به صورت همزمان تمامی سیاست‌های حافظه‌ی نهان را در کنار هم شبیه‌سازی کرد.

در این شبیه‌ساز، سیاست‌های LRU، درصد فراوانی، لارک [۲]، اراکل [۴] و معماری پیشنهادی پیاده‌سازی شده‌اند. علاوه بر پیمانه‌های نام برده شده، تابعی برای تبدیل دنباله‌ی تصمیمات مدل ارائه شده از شبکه‌ی عصبی تکرار شونده به تصمیمات مدیر حافظه‌ی نهان پیاده‌سازی شد. به این ترتیب، خروجی مدل شبکه‌ی عصبی، مستقیما به حافظه‌ی نهان منتقل و امکان کنترل درخواست‌های ورودی/خروجی به صورت برخط در شبیه‌ساز بوجود آمد.

2. Modular

برای پیاده‌سازی شبکه‌ی عصبی تکرار شونده، از زبان برنامه‌نویسی پایتون استفاده شد. علت استفاده‌ی این زبان، وجود کتابخانه‌های وسیع و کامل در زمینه‌ی یادگیری ماشین بستر این زبان است. در پیاده‌سازی شبکه‌ی عصبی از کتابخانه‌ی Keras [۴۱] استفاده شد. دو کلاس ۱) ساده و ۲) عمیق از شبکه‌های عصبی تکرار شونده پیاده‌سازی شدند که به ترتیب برای ویژگی‌شناسی بار کاری و مدیریت حافظه‌ی نهان استفاده شدند.

## ۵-۲ آزمایش‌ها و نتایج

در این بخش، روش‌های سنجش و نتایج بدست آمده از مقایسه‌ی معماری پیشنهادی با کارهای پیشین در دو بخش متفاوت ارائه می‌شوند. در بخش اول، روش ویژگی‌شناسی پیشنهادی که بر پایه‌ی شبکه‌های عصبی پیاده‌سازی شده، با روش‌های پیشین که به صورت آماری و ایستا هستند، در سناریوهای متفاوت مقایسه می‌شود. در بخش دوم، هدف اصلی معماری پیشنهادی، یعنی مدیریت بهینه‌ی سامانه‌ی حافظه‌ی نهان، در کنار کارهای پیشین مورد سنجش قرار می‌گیرد. لازم به ذکر است که تمام مراحل یادگیری در تمامی آزمایش‌ها مربوط به شبکه‌ی عصبی تکرار شونده، به صورت برون‌خط و با استفاده از واحد پردازش گرافیکی Nvidia Titan X [1] انجام شده‌اند. در مقابل، مرحله‌ی سنجش مربوط به شبکه‌ی عصبی به صورت برخط بوده و لذا تمام آزمایش‌ها با استفاده از واحد پردازش مرکزی انجام گرفته و سربار محاسباتی کمی دارند.

1. Graphics Proccessing Unit (GPU)

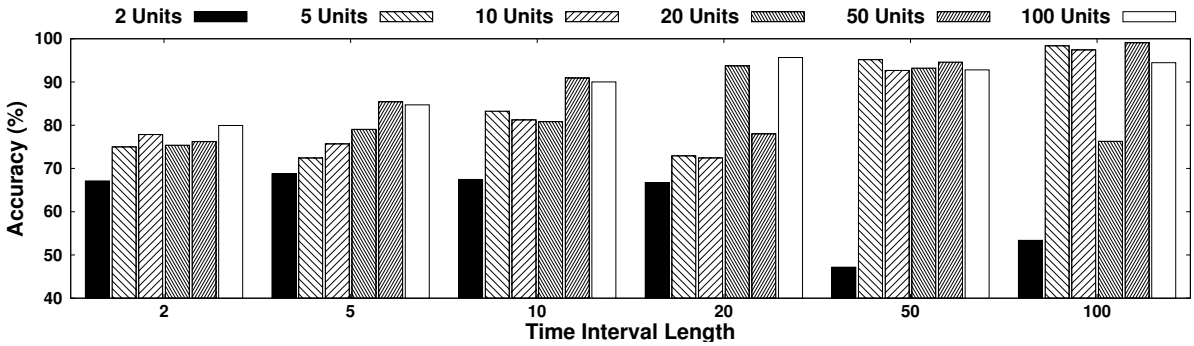
### ۵-۲-۱ ویژگی‌شناسی بار کاری

قبل از تشریح روش‌های سنجش و نتایج بدست آمده، نیاز به بررسی درستی پیکربندی شبکه‌ی عصبی پیاده‌سازی شده وجود دارد. برای این موضوع آزمایشی ساده طراحی شد تا پیکربندی‌های مختلف شبکه‌ی عصبی سنجیده شوند. در این آزمایش دو بار کاری مختلف از مجموعه بارهای کاری پایگاه اسنیا [۲۷] به نام‌های Radius_Authentication و Mail_Index برای ویژگی‌شناسی ارائه شده و شبکه‌ی عصبی با پیکربندی‌های مختلف مورد سنجش قرار گرفت. در پیکربندی شبکه‌ی عصبی، دو مورد از اهمیت بالایی برخوردار هستند:

- **۱ـ بازه‌های زمانی**[1] : اندازه‌ی بازه‌های زمانی که شبکه به بررسی ارتباط بین درخواست‌ها می‌پردازد.
- **۲ـ تعداد واحدهای پنهان**[2] : تعداد مولفه‌های پنهان موجود در شبکه‌ی عصبی تکرار شونده.

شکل ۵-۱ نتایج مربوط به این آزمایش را نشان می‌دهد. با توجه به نتایج بدست آمده، ابتدا مشخص می‌شود که با افزایش بازه‌های زمانی می‌توان به دقت بیشتری در تشخیص بارهای کاری رسید که نتیجه‌ای منطقی است. البته بدیهی است که نیازی به بررسی بازه‌های زمانی بیش از ۱۰۰ نیست زیرا شبکه به دقت کافی رسیده و از نظر یادگیری اشباع شده است. از طرفی با بررسی تعداد واحدهای پنهان در شبکه‌ی عصبی، می‌توان مشاهده کرد که بالاترین دقت مربوط به شبکه‌ی عصبی با ۵۰ واحد پنهان است. با توجه به این نتایج، پیکربندی شبکه‌ی عصبی تکرار شونده با ۵۰ واحد پنهان و با طول بازه‌ی زمانی ۱۰۰ به عنوان پیکربندی نهایی انتخاب شد. پس از اطمینان از درستی پیکربندی شبکه‌ی عصبی مورد استفاده، آزمایشی برای مقایسه‌ی عملکرد روش پیشنهادی با روش‌های پیشین طراحی شد. در این آزمایش، سناریوهای تعریف شده در جدول ۳-۱ در شبیه‌ساز پیاده‌سازی شدند. این آزمایش از دو فاز ۱) یادگیری و ۲) سنجش تشکیل شده است. فاز یادگیری، مربوط به مرحله‌ی برون‌خط در معماری پیشنهادی بوده و لذا هر یک از روش‌ها به اندازه‌ی کافی زمان برای یادگیری بارهای کاری دارد. در این فاز، ۴۰ هزار درخواست از هر بار کاری به عنوان نمونه‌های یادگیری به

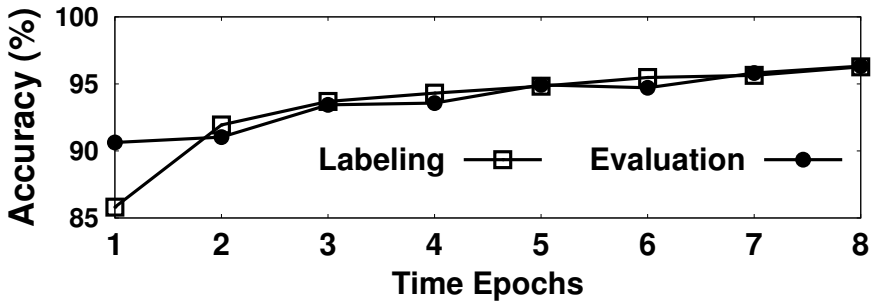
1. Time Intervals
2. Hidden Units




**شکل ۵-۱:** درصد دقت بدست آمده در ویژگی‌شناسی بارهای کاری با پیکربندی‌های متفاوت شبکه‌ی عصبی

روش‌های ویژگی‌شناسی داده شدند. کارهای پیشین، همگی از روش‌های آماری ساده مانند درصد فراوانی و یا میانگین اندازه‌ی درخواست‌ها و غیره استفاده کرده و اطلاعات مورد نیاز خود را از بارهای کاری جمع‌آوری کردند.

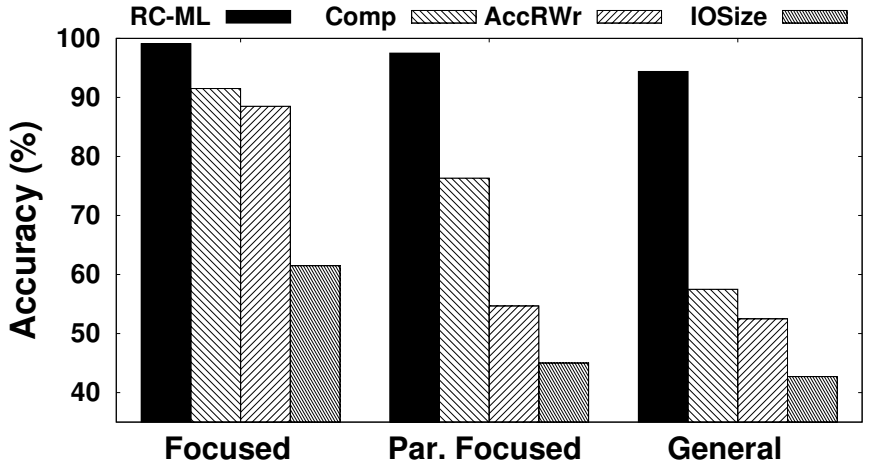
روش پیشنهادی که بر پایه‌ی شبکه‌های عصبی تکرار شونده پیاده‌سازی شده است، نیاز به مدت زمان بیشتری نسبت به کارهای پیشین برای یادگیری بارهای کاری دارد. در این روش، شبکه‌ی عصبی در ابتدا اقدام به برچسب‌گذاری[1] درخواست‌ها کرده و سپس دقت بدست آمده را می‌سنجد. این کار تا رسیدن به نقطه‌ی اشباع[2] شبکه و یادگیری کامل الگوهای داده ادامه خواهد داشت. شکل ۵-۲ درصد دقت شبکه‌ی عصبی را برای برچسب‌گذاری و سنجش این فاز در سناریوی عمومی نشان می‌دهد. همانطور که در شکل پیداست، شبکه‌ی عصبی توانسته در هشت مرحله به مرحله‌ی اشباع رسیده و با دقت بیش از ۹۶% فاز یادگیری را به پایان برساند.




**شکل ۵-۲:** دقت مراحل یادگیری روش پیشنهادی تا بلوغ کامل مدل

1. Labeling
2. Saturation

پس از فاز یادگیری، نوبت به فاز سنجش می‌رسد. در این فاز ۱۰ هزار درخواست جدید از هر بار کاری برای سنجش در نظر گرفته شدند. تمامی روش‌های ویژگی‌شناسی در ۱۰۰ بازه‌ی ۱۰۰ تایی از درخواست‌ها سنجش شده و در هر سناریو با یکدیگر مقایسه می‌شوند. شکل ۵-۳ نتایج مربوط به فاز سنجش از این آزمایش را نشان می‌دهد. نتایج بدست آمده نشان دهنده‌ی دقت بسیار بالای روش پیشنهادی نسبت به روش‌های آماری پیشین می‌باشد. نقطه ضعف اصلی روش‌های پیشین، کاهش شدید دقت نهایی در مواجهه با بارهای کاری با تعداد بالای درخواست‌های متنوع است که نتیجه‌ی تخمین‌های موجود در این روش‌هاست. در مقابل، روش پیشنهادی در تمامی سناریوها به دقت بالای ۹۵% دست یافته است. این نتایج نشان می‌دهند که برای تحلیل کامل بارهای کاری سامانه‌های ذخیره‌سازی داده، نیازمند به الگوریتم‌های پویا و پیچیده‌ای مانند شبکه‌ی عصبی تکرار شونده هستیم.




**شکل ۵-۳:** مقایسه‌ی دقت روش‌های ویژگی‌شناسی در سناریوهای مختلف

## ۵-۲-۲ مدیریت حافظه‌ی نهان

برای بررسی عملکرد معماری پیشنهادی برای حافظه‌ی نهان مبتنی بر دیسک حالت جامد، همانطور که در فصل پیش تشریح شد، رفتار الگوریتم مطلوب اراکل به ازای هر بار کاری به صورت دنباله‌ای جدید به شبکه‌ی عصبی تکرار شونده‌ی عمیق سه لایه داده شد. جدول ۵-۱ نتایج بدست آمده از یادگیری رفتار اراکل و در نهایت سنجش صحت یادگیری را در شبکه‌ی عصبی عمیق نشان می‌دهد.

**جدول ۵-۱:** یادگیری رفتار اراکل به وسیله‌ی شبکه‌ی عصبی تکرار شونده

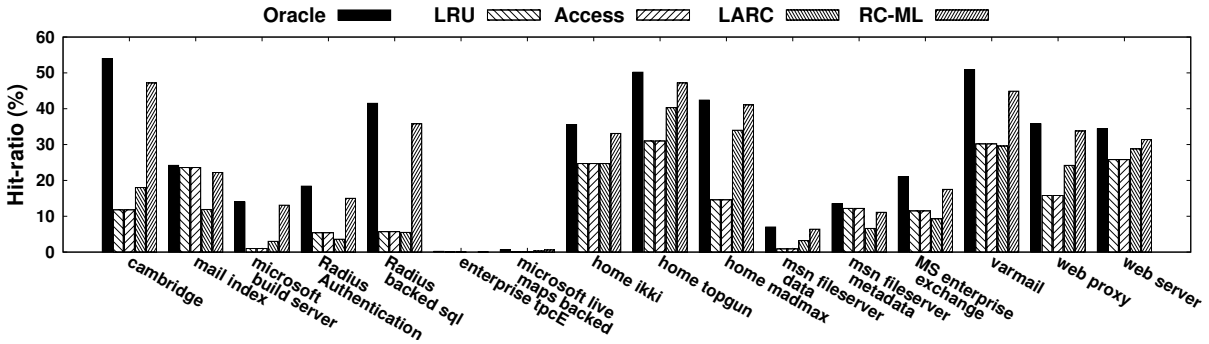
| tier_dummy | y_dummy | tier_acc | y_acc | بارهای کاری |
|---|---|---|---|---|
| ۷۷,۶۹% | ۶۲,۶۹% | ۹۳,۶۵% | ۹۱,۲۳% | ex. enterprise MS |
| ۸۸,۹۲% | ۷۲,۶۳% | ۹۵,۱۷% | ۹۷,۹۳% | cambridge |
| ۱۰۰,۰۰% | ۷۳,۴۲% | ۱۰۰,۰۰% | ۹۲,۳۴% | ikki home |
| ۷۴,۴۹% | ۵۸,۴۰% | ۹۱,۴۲% | ۸۹,۳۱% | Auth. Radius |
| ۴۹,۷۱% | ۶۶,۶۲% | ۷۲,۵۴% | ۹۵,۴۵% | sql backed Radius |
| ۹۹,۸۴% | ۸۳,۳۱% | ۱۰۰,۰۰% | ۸۸,۳۱% | tpcE enterprise |
| ۹۹,۸۷% | ۹۱,۸۱% | ۹۹,۹۰% | ۹۸,۵۹% | casa home |
| ۹۸,۹۸% | ۸۱,۶۸% | ۹۹,۵۳% | ۹۴,۳۲% | index mail |
| ۹۹,۷۰% | ۷۷,۶۰% | ۹۹,۹۰% | ۹۷,۵۹% | server build MS |
| ۱۰۰,۰۰% | ۸۲,۷۵% | ۱۰۰,۰۰% | ۹۳,۷۴% | maps live MS |
| ۸۱,۷۶% | ۷۸,۱۴% | ۹۱,۵۵% | ۹۷,۱۷% | data FS msn |
| ۹۴,۱۵% | ۷۵,۴۰% | ۹۷,۰۷% | ۹۴,۹۹% | metadata FS msn |
| ۹۷,۳۴% | ۶۳,۴۷% | ۹۸,۴۴% | ۹۶,۷۴% | varmail |
| ۷۹,۸۵% | ۷۱,۲۰% | ۹۹,۶۷% | ۹۳,۵۱% | proxy web |
| ۹۱,۷۰% | ۷۸,۳۶% | ۹۸,۵۰% | ۹۵,۲۳% | server web |

برای اطمینان از درستی یادگیری، تابعی به نام روش پیش پا افتاده[1] نیز روی داده‌ها سنجش شد. در روش پیش پا افتاده، تمام تصمیمات به صورت تصادفی و کاملا بر اساس شانس صورت می‌گیرد. برای اثبات درستی یادگیری، نیاز است تا تمام نتایج بدست آمده در شبکه‌ی عصبی همواره بهتر از روش پیش پا افتاده باشد. در این جدول، ستون y_acc مربوط به نتایج سنجش نهایی نسبت به رفتار اراکل در زمینه‌ی تشخیص واکشی یا عدم واکشی و ستون tier_acc مربوط به تشخیص بازه‌ی طول عمر بلوک داده در حافظه‌ی نهان است. ستون‌های چهارم و پنجم نیز به ترتیب مربوط به نتایج بدست آمده از روش پیش پا افتاده نسبت به ستون‌های دوم و سوم هستند.

1. Dummy Method

برای سنجش معماری پیشنهادی، دو آزمایش مختلف طراحی شد. در آزمایش اول، هدف اصلی رسیدن به بیشترین نرخ برخورد با بهترین تعداد عملیات جایگزینی در حافظه‌ی نهان می‌باشد. در آزمایش دوم، قابلیت بازپیکربندی معماری پیشنهادی در مقایسه با کارهای قبلی درستی آزمایی می‌شود. در ادامه به بررسی این دو آزمایش می‌پردازیم.

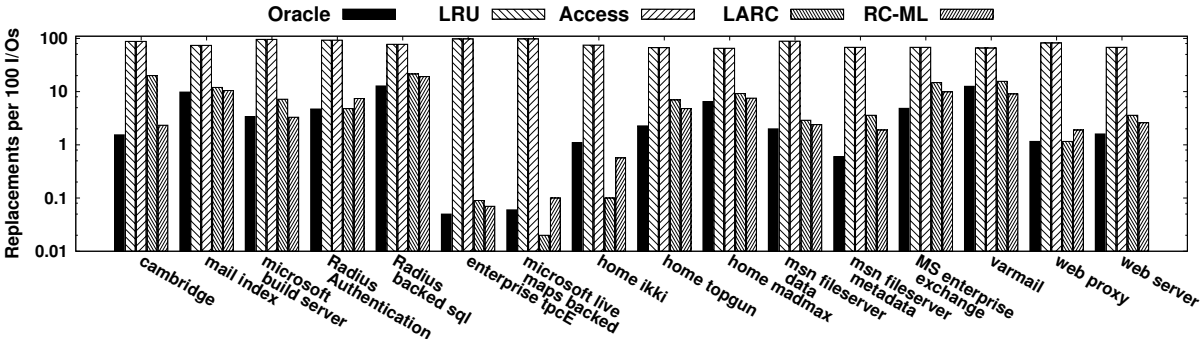
در آزمایش اول، ۱۶ بار کاری مختلف از پایگاه اینترنتی اسنیا [۲۷] و ابزار filebench [۴۲] جمع‌آوری شدند. در جدول ۵-۱ نشان داده شد که شبکه‌ی عصبی پیاده‌سازی شده توانسته به ترتیب بیش از ۹۶% و ۹۸% تشخیص درست در واکشی و عمر بلوک در حافظه‌ی نهان داشته باشد. لذا رفتار شبکه‌ی عصبی تا حد بالایی شبیه به الگوریتم اراکل است. دنباله‌ی تصمیمات شبکه‌ی عصبی، به عنوان ورودی به شبیه‌ساز داده شده تا بتوان با شرایط یکسان تمامی روش‌های حافظه‌ی نهان موجود را با هم مقایسه کرد. شکل ۵-۴ نرخ برخورد روش‌های پیشین در مقایسه با معماری پیشنهادی و اراکل نشان می‌دهد. با استناد به نتایج بدست آمده، در برخی از بارهای کاری، معماری پیشنهادی تا بیش از ۷ برابر نرخ برخورد بیشتری نسبت به کارهای پیشین دارد.




**شکل ۵-۴:** نرخ برخورد معماری پیشنهادی در مقایسه با کارهای پیشین

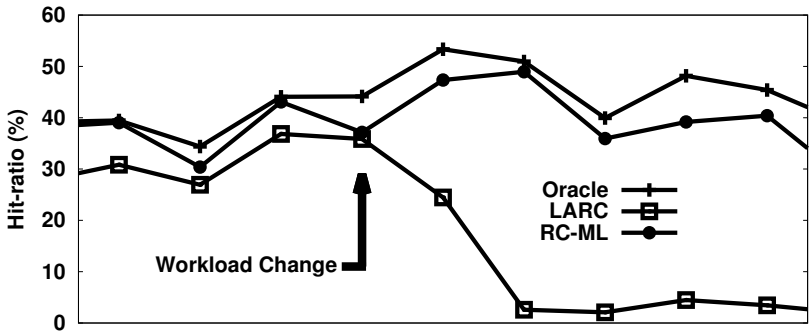
علاوه بر نرخ برخورد، در حافظه‌ی نهان مبتنی بر دیسک حالت جامد، به دلیل محدودیت تعداد درخواست‌های نوشتن در دیسک حالت جامد، تعداد عملیات جایگزینی نیز از اهمیت بالایی برخوردار است. شکل ۵-۵ تعداد عملیات جایگزینی به ازای هر ۱۰۰ درخواست ورودی/خروجی در بارهای کاری مختلف را به صورت لگاریتمی نشان می‌دهد. با توجه به نتایج بدست آمده، معماری پیشنهادی در برخی از بارهای کاری همچون cambridge [۲۷]، تا بیش از ۱۰ برابر کمتر از کارهای پیشین عملیات جایگزینی دارد. این موضوع نشان می‌دهد که معماری پیشنهادی علاوه بر افزایش کارایی

حافظه‌ی نهان، از عملیات جایگزینی بی‌مورد نیز جلوگیری کرده و در نتیجه عمر دیسک حالت جامد را افزایش می‌دهد.




**شکل ۵-۵:** تعداد عملیات جایگزینی معماری پیشنهادی در مقایسه با کارهای پیشین

هدف اصلی از این پروژه ارائه‌ی یک معماری حافظه‌ی نهان مبتنی بر دیسک حالت جامد است که به ازای هر بار کاری بتواند با تقریب خوبی بهترین روش مدیریت درخواست‌های ورود/خروجی را ارائه دهد. علاوه بر این، یکی از برتری‌های اصلی این معماری نسبت به کارهای پیشین قابلیت بازپیکربندی خودکار در مواجهه با تغییر بارهای کاری می‌باشد. به طور کلی در آزمایش‌های قبلی، قدرت تشخیص بارهای کاری و تغییر آنها و همینطور بهینه‌سازی حافظه‌ی نهان به ازای هر بار کاری مورد سنجش قرار گرفت. در آخرین آزمایش، تمام سامانه به صورت یکجا در سناریویی با تغییر بارهای کاری سنجیده می‌شود.




**شکل ۵-۶:** نرخ برخورد دوره‌ای روش‌های متفاوت حافظه نهان در مواجهه با تغییر بار کاری

در سناریوی آزمایش نهایی، در ابتدا ۵۰ هزار درخواست از بار کاری home_ikki [۲۷] به عنوان روردی شبیه‌ساز داده می‌شود. سپس ۵۰ هزار درخواست دیگر از بار کاری radius_authentication

[۲۷] به شبیه‌ساز داده شده تا تغییر بار کاری شبیه‌سازی شود. شکل ۵-۶ نرخ برخورد دوره‌ای معماری پیشنهادی را در مقایسه با الگوریتم اراکل و روش لارک نشان می‌دهد. همانطور که مشخص است، معماری پیشنهادی به دلیل دقت تشخیص بالای ۹۸% در بارهای کاری، توانسته بار کاری جدید را تشخیص بدهد. پس از تشخیص تغییر بار کاری، مدل حافظه‌ی نهان بهینه‌سازی شده برای بار کاری جدید، کنترل حافظه‌ی نهان را به عهده گرفته و نرخ برخورد متناسب با الگوریتم اراکل افزایش می‌یابد. در مقابل، در روش لارک، هیچ‌گونه قابلیت بازپیکربندی وجود نداشته و لذا نرخ برخورد حافظه‌ی نهان کاهش پیدا می‌کند.

# فصل ۶

## نتیجه‌گیری و کارهای آتی

در این پژوهش، برای اولین بار، یک معماری حافظه‌ی نهان مبتنی بر دیسک حالت جامد با استفاده از شبکه‌ی عصبی تکرار شونده ارائه شده است که قابلیت بازپیکربندی خودکار در هنگام تغییر بار کاری را دارد. این معماری با بهره‌گیری از روش ویژگی‌شناسی بار کاری بر پایه‌ی یادگیری ماشین، می‌تواند بارهای کاری را از یک دیگر تشخیص داده و بهترین پیکربندی موجود را برای مدیریت حافظه‌ی نهان ارائه دهد.

روش پیشنهادی برای ویژگی‌شناسی بار کاری، در بدترین شرایط می‌تواند با دقت بالای ۹۶% چهار بار کاری در حال اجرا را از هم تشخیص دهد در حالی که دقت روش‌های ارائه شده در کارهای پیشین کمتر از ۶۰% است. معماری پیشنهادی برای مدیریت حافظه‌ی نهان می‌تواند با دقت بیش از ۹۵% رفتار الگوریتم ایده‌آل اراکل را الگو برداری کند. لذا ارزیابی‌های مختلف توسط بارهای کاری متعدد، نشان می‌دهد که معماری پیشنهادی نرخ برخورد را نسبت به کارهای پیشین تا ۷ برابر افزایش می‌دهد. علاوه بر این، معماری پیشنهادی تعداد عملیات جایگزینی حافظه‌ی نهان را تا ۱۰ برابر نسبت به کارهای پیشین کاهش می‌دهد که این موضوع منجر به افزایش طول عمر دیسک حالت جامد می‌شود.

برای ارزیابی روش پیشنهادی، از ۱۶ بار کاری متفاوت از منابع قابل استناد استفاده شده است. برای افزایش قابلیت اطمینان و درستی‌آزمایی معماری پیشنهادی، در کارهای آتی نیاز است تا تعداد بارهای کاری افزایش یابد. علاوه بر این، روش‌های متعددی در کنار شبکه‌ی عصبی تکرار شونده در یادگیری ماشین وجود دارند که می‌توانند در این معماری استفاده شوند. این روش‌ها ممکن است چه

از لحاظ کارایی و چه از لحاظ سربار محاسباتی و حافظه نسبت به شبکه‌ی عصبی تکرار شونده بهبود داشته باشند. لذا بررسی و پیاده‌سازی برخی از پرکاربردترین روش‌های یادگیری ماشین می‌تواند یکی از اصلی‌ترین شاخه‌های کارهای آتی این پژوهش باشد.

در الگوریتم اراکل استفاده شده، برای ساده‌سازی، از پیش‌واکشی بلوک‌های داده صرف نظر شده است. در حالی که ایده‌آل‌ترین روش ممکن به ازای هر بار کاری با بهره‌گیری از پیش‌واکشی داده به نرخ برخورد ۱۰۰% خواهد رسید. در کارهای آتی می‌توان به الگوریتم اراکل، قابلیت پیش‌واکشی را نیز اضافه کرد. البته در این شرایط، نحوه‌ی ارائه‌ی رفتار اراکل به مدل یادگیری ماشین، پیچیده‌تر خواهد شد. از طرفی، در این پژوهش، به دلیل کمبود وقت و محدودیت‌ها سخت‌افزاری، معماری پیشنهادی در شبیه‌ساز پیاده‌سازی شده است. یکی از مهم‌ترین کارهای آتی، پیاده‌سازی واقعی معماری پیشنهادی در یک کارگزار ذخیره‌سازی داده‌ی مجهز به واحد پردازش گرافیکی است.

# مراجع


[1] Y. Klonatos, T. Makatos, M. Marazakis, M. D. Flouris, and A. Bilas, "Azor: Using two-level block selection to improve ssd-based i/o caches," in *2011 IEEE Sixth International Conference on Networking, Architecture, and Storage*, pp.309–318, July 2011.

[2] D. F. S. Huang, Q. Wei, "Improving flash-based disk cache with lazy adaptive replacement," in *Proceedings of the 29th IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, p.1–10, May 2013.

[3] R. Salkhordeh, H. Asadi, and S. Ebrahimi, "Operating system level data tiering using online workload characterization," *J. Supercomput.*, vol.71, pp.1534–1562, Apr. 2015.

[4] L. A. Belady, R. A. Nelson, and G. S. Shedler, "An anomaly in space-time characteristics of certain programs running in a paging machine," *Commun. ACM*, vol.12, pp.349–353, June 1969.

[5] M. Saxena and M. M. Swift, "Design and prototype of a solid-state cache," *Trans. Storage*, vol.10, pp.10:1–10:34, Aug. 2014.

[6] X. Wu and A. L. N. Reddy, "Exploiting concurrency to improve latency and throughput in a hybrid storage system," in *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp.14–23, Aug 2010.

[7] J. Guerra, H. Pucha, J. Glider, W. Belluomini, and R. Rangaswami, "Cost effective storage using extent based dynamic tiering," in *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, FAST'11, (Berkeley, CA, USA), pp.1–20, USENIX Association, 2011.

[8] S. Liu, J. Jiang, and G. Yang, "Macss: A metadata-aware combo storage system," in *2012 International Conference on Systems and Informatics (ICSAI2012)*, pp.919–923, May 2012.

[9] R. Appuswamy, D. C. van Moolenbroek, and A. S. Tanenbaum, "Integrating flash-based ssds into the storage stack," in *012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, pp.1–12, April 2012.

[10] S. L. R. Santana and J. Liu, "To arc or not to arc," in *Proceedings of the 7th USENIX Conference on Hot Topics in Storage and File Systems (HotStorage)*, p.4–14, 2015.

[11] I. Stefanovici, E. Thereska, G. O'Shea, B. Schroeder, H. Ballani, T. Karagiannis, A. Rowstron, and T. Talpey, "Software-defined caching: Managing caches in multi-tenant data centers," in *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC '15, (New York, NY, USA), pp.174–181, ACM, 2015.

[12] Y. Chai, Z. Du, X. Qin, and D. A. Bader, "Wec: Improving durability of ssd cache drives by caching write-efficient data," *IEEE Transactions on Computers*, vol.64, pp.3304–3316, Nov 2015.

[13] R. Koller, L. Marmol, R. Rangaswami, S. Sundararaman, N. Talagala, and M. Zhao, "Write policies for host-side flash caches," in *Proceedings of the 11th USENIX Conference on File and Storage Technologies*, FAST'13, (Berkeley, CA, USA), pp.45–58, USENIX Association, 2013.

[14] S. Byan, J. Lentini, A. Madan, and L. Pabón, "Mercury: Host-side flash caching for the data center," in *012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*, pp.1–12, April 2012.

[15] R. Koller, A. J. Mashtizadeh, and R. Rangaswami, "Centaur: Host-side ssd caching for storage performance control," in *2015 IEEE International Conference on Autonomic Computing*, pp.51–60, July 2015.

[16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[17] N. Megiddo and D. Modha, "Outperforming lru with an adaptive replacement cache algorithm," *Computer*, vol.37, p.58–65, April 2004.

[18] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber, "Extending ssd lifetimes with disk-based write caches," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, FAST'10, (Berkeley, CA, USA), pp.1–8, USENIX Association, 2010.

[19] F. Ye, J. Chen, X. Fang, J. Li, and D. Feng, "A regional popularity-aware cache replacement algorithm to improve the performance and lifetime of ssd-based disk cache," in *2015 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pp.45–53, Aug 2015.

[20] Y. Shen, L. Luo, and G. Zhang, "Loca: A low-overhead caching algorithm for flash-based ssds," *Int. J. Wire. Mob. Comput.*, vol.10, pp.13–19, Mar. 2016.

[21] F. Chen, D. A. Koufaty, and X. Zhang, "Hystor: Making the best use of solid state drives in high performance storage systems," in *Proceedings of the International Conference on Supercomputing*, ICS '11, (New York, NY, USA), pp.22–32, ACM, 2011.

[22] Y. Ni, J. Jiang, D. Jiang, X. Ma, J. Xiong, and Y. Wang, "S-rac: Ssd friendly caching for data center workloads," in *Proceedings of the 9th ACM International on Systems and Storage Conference*, SYSTOR '16, (New York, NY, USA), pp.8:1–8:12, ACM, 2016.

[23] T. Pritchett and M. Thottethodi, "Sievestore: A highly-selective, ensemble-level disk cache for cost-performance," *SIGARCH Comput. Archit. News*, vol.38, pp.163–174, June 2010.

[24] F. Meng, L. Zhou, X. Ma, S. Uttamchandani, and D. Liu, "vcacheshare: Automated server flash cache space management in a virtualization environment," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, (Philadelphia, PA), pp.133–144, USENIX Association, 2014.

[25] W.-H. Kang, S.-W. Lee, B. Moon, Y.-S. Kee, and M. Oh, "Durable write cache in flash memory ssd for relational and nosql databases," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, (New York, NY, USA), pp.529–540, ACM, 2014.

[26] T. Luo, R. Lee, M. Mesnier, F. Chen, and X. Zhang, "hstorage-db: Heterogeneity-aware data management to exploit the full capability of hybrid storage systems," *Proc. VLDB Endow.*, vol.5, pp.1076–1087, June 2012.

[27] "Storage networking industry association (snia)," 2017. http://ww.snia.org.

[28] A. Riska and E. Riedel, "Disk drive level workload characterization," in *Annual Tech '06: 2006 USENIX Annual Technical Conference*, 2006.

[29] M. Calzarossa and G. Serazzi, "Workload characterization: a survey," *Proceedings of the IEEE*, vol.81, pp.1136–1150, Aug 1993.

[30] L. Calzarossa, Mariaand Massari and D. Tessera, "Workload characterization issues and methodologies," *Performance Evaluation: Origins and Directions*, pp.459–482, 2000.

[31] S. Huang, J. Huang, J. Dai, T. Xie, and B. Huang, "The hibench benchmark suite: Characterization of the mapreduce-based data analysis," in *2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010)*, pp.41–51, March 2010.

[32] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda, "Characterization of storage workload traces from production windows servers," in *2008 IEEE International Symposium on Workload Characterization*, pp.119–128, Sept 2008.

[33] I. Ahmad, "Easy and efficient disk i/o workload characterization in vmware esx server," in *2007 IEEE 10th International Symposium on Workload Characterization*, pp.149–158, Sept 2007.

[34] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley, "24/7 characterization of petascale i/o workloads," in *2009 IEEE International Conference on Cluster Computing and Workshops*, pp.1–10, Aug 2009.

[35] M. F. Arlitt and C. L. Williamson, "Web server workload characterization: The search for invariants," *SIGMETRICS Perform. Eval. Rev.*, vol.24, pp.126–137, May 1996.

[36] M. Tarihi, H. Asadi, A. Haghdoost, M. Arjomand, and H. Sarbazi-Azad, "A hybrid non-volatile cache design for solid-state drives using comprehensive i/o characterization," *IEEE Transactions on Computers*, vol.65, pp.1678–1691, June 2016.

[37] D. Guttman, M. T. Kandemir, M. Arunachalam, and R. Khanna, "Machine learning techniques for improved data prefetching," in *5th International Conference on Energy Aware Computing Systems Applications*, pp.1–4, March 2015.

[38] S. Suthaharan, "Big data classification: Problems and challenges in network intrusion prediction with machine learning," *SIGMETRICS Perform. Eval. Rev.*, vol.41, pp.70–73, Apr. 2014.

[39] S. A. Ajila and A. A. Bankole, "Cloud client prediction models using machine learning techniques," in *2013 IEEE 37th Annual Computer Software and Applications Conference*, pp.134–142, July 2013.

[40] A. A. Sarina Sulaiman, Siti Mariyam Shamsuddin and S. Sulaiman, "Intelligent web caching using machine learning methods," *Neural Network World*, vol.21, pp.429–452, 2011.

[41] F. Chollet *et al.*, "Keras," 2017. https://github.com/fchollet/keras/tree/master/keras.

[42] A. Wilson, "The new and improved filebench," in *Proceedings of the 6th USENIX Conference on File and Stroage Technologies*, FAST, USENIX Association, 2008.

# واژه‌نامه انگلیسی به فارسی

## A

الگوهای دسترسی ..... Access Patterns

## B

رو به عقب ..... backward

ابزار محک ..... Benchmark

دودویی ..... Binary

.....

میانگیر ..... buffer

رگبار ..... Burst

## C

حافظه‌ی نهان ..... Cache

مدیر حافظه‌ی نهان ..... Cache Manager

ویژگی‌شناسی ..... Characterization

دسته‌بندی ..... Classification

خوشه‌بندی ..... Clustering

هم‌گردان ..... Compiler

کنترل کننده ..... Controller

سنتی ..... Conventional

.....

## D

مهاجرت بلوک داده ..... Data Block Migration

دستگاه‌های ذخیره‌سازی داده ... Data Storage Devices

سامانه‌های ذخیره‌سازی داده ... Data Storage System

انبارهای داده ..... Data Warehouses

سامانه‌ی مدیریت پایگاه داده .. Database Management System (DBMS)

شبکه‌ی عصبی تکرار شونده‌ی عمیق ..... Deep RNN

.....

## E

موتور ..... Engine

## F

بازخورد ..... Feedback

واکشی ..... Fetch

.....

الگوریتم اولین ورودی، اولین خروجی ... First In First Out (FIFO)

نشان ..... Flag

سلول‌های حافظه‌ی فلش ..... Flash Memory Cells

تغییرپذیری ..... Flexibility

. . . . . .

حافظه‌ی طولانی کوتاه‌مدت Long short-term memory (LSTM)

## M

یادگیری ماشین . . . . . Machine Learning

نوارهای مغناطیسی . . . . . Magnetic Tape

کاهش نگاشت . . . . . MapReduce

پیمانه‌ای . . . . . Modular

## N

شبکه‌های عصبی . . . . . Neural Networks

حافظه‌های غیرفرار . . None Volatile Memory (NVM)

. . . . . .

## O

برون‌خط . . . . . Offline

برخط . . . . . Online

اراکل . . . . . Oracle/Belady

هم‌پوشان . . . . . Overlapped

## P

گلوگاه کارایی . . . . . Performance Bottleneck

پیش‌واکشی . . . . . Prefetching

پایتون . . . . . Python

نقطه‌ی شناور . . . . . Floating Point

رو به جلو . . . . . Forward

## G

دروازه . . . . . Gate

واحد پردازش گرافیکی . . . Graphics Proccessing Unit (GPU)

## H

. . . . . .

دیسک‌های سخت . . . . . Hard Disk Drive (HDD)

واحدهای پنهان . . . . . Hidden Units

نرخ برخورد . . . . . Hit Ratio

رابط میزبان . . . . . Host Interface

معماری‌های ترکیبی . . . . . Hybrid Architecture

## I

درخواست‌های ورودی/خروجی . . . . . I/O Requests

## K

. . . . . .

## L

. . . . . .

اخیرا کمتر استفاده شده . Least Recently Used (LRU)

پس‌گرایی خطی . . . . . Linear Regression

# Q

کیفیت خدمت ..... Quality of Service (QoS)

# R

تصادفی ..... Random

بازپیکربندی ..... Reconfiguration

مدیر پیکربندی ..... Reconfiguration Manager

شبکه‌های عصبی تکرار شونده ..... Recurrent Neural Network (RNN)

یادگیری تقویتی ..... Reinforcement Learning

..... 

جایگزینی ..... Replacement

.....

# S

اشباع ..... Saturation

زمان‌بندی ..... Scheduling

الگوریتم شانس دوم ..... Second Chance Algorithm

خودهمسانی ..... Self-Similarity

ترتیبی ..... Sequential

دیسک‌های حالت جامد ..... Solid State Disk (SSD)

مکانی ..... Spatial

اسنیا .... Storage Networking Industry Association (SNIA)

گام‌دار ..... Strided

یادگیری با نظارت ..... Supervised Learning

ماشین بردار پشتیبانی ..... Supprt Vector Machine

# T

زمانی ..... Temporal

.....

رده‌بندی داده ..... Tiering

بازه‌های زمانی ..... Time Intervals

دنباله‌های زمانی ..... Time Series

# U

یادگیری بدون نظارت ..... Un-Supervised Learning

# V

مجازی‌سازی ..... Virtualization

# W

بار کاری ..... Workload

ناظر بار کاری ..... Workload Monitoring

.....

پس‌نوشت ..... Write-Back

رونوشت ..... Write-Throgh

# واژه‌نامه فارسی به انگلیسی

## A

Access Patterns ..... الگوهای دسترسی

## B

backward ..... رو به عقب

Benchmark ..... ابزار محک

Binary ..... دودویی

buffer ..... میانگیر

Burst ..... رگبار

## C

Cache ..... حافظه‌ی نهان

Cache Manager ..... مدیر حافظه‌ی نهان

Characterization ..... ویژگی‌شناسی

Classification ..... دسته‌بندی

Clustering ..... خوشه‌بندی

Compiler ..... هم‌گردان

Controller ..... کنترل کننده

Conventional ..... سنتی

## D

Data Block Migration ..... مهاجرت بلوک داده

Data Storage Devices ... دستگاه‌های ذخیره‌سازی داده

Data Storage System ... سامانه‌های ذخیره‌سازی داده

Data Warehouses ..... انبارهای داده

Database Management System (DBMS) .. سامانه‌ی مدیریت پایگاه داده

Deep RNN ..... شبکه‌ی عصبی تکرار شونده‌ی عمیق

## E

Engine ..... موتور

## F

بازخورد ..... Feedback

واکشی ..... Fetch

الگوریتم اولین ورودی، اولین خروجی ... First In First Out (FIFO)

نشان ..... Flag

سلول‌های حافظه‌ی فلش ..... Flash Memory Cells

تغییرپذیری ..... Flexibility

نقطه‌ی شناور ..... Floating Point

رو به جلو ..... Forward

## G

دروازه ..... Gate

واحد پردازش گرافیکی ... Graphics Proccessing Unit (GPU)

## H

..... .....

دیسک‌های سخت ..... Hard Disk Drive (HDD)

واحدهای پنهان ..... Hidden Units

نرخ برخورد ..... Hit Ratio

رابط میزبان ..... Host Interface

معماری‌های ترکیبی ..... Hybrid Architecture

## I

درخواست‌های ورودی/خروجی ..... I/O Requests

## K

..... .....

## L

..... .....

اخیرا کمتر استفاده شده . Least Recently Used (LRU)

پس‌گرایی خطی ..... Linear Regression

حافظه‌ی طولانی کوتاه‌مدت . Long short-term memory (LSTM)

## M

یادگیری ماشین ..... Machine Learning

نوارهای مغناطیسی ..... Magnetic Tape

کاهش نگاشت ..... MapReduce

پیمانه‌ای ..... Modular

## N

Neural Networks ..... شبکه‌های عصبی

None Volatile Memory (NVM) .. حافظه‌های غیرفرار

## O

Offline ..... برون‌خط

Online ..... برخط

Oracle/Belady ..... اراکل

Overlapped ..... هم‌پوشان

## P

Performance Bottleneck ..... گلوگاه کارایی

Prefetching ..... پیش‌واکشی

Python ..... پایتون

## Q

Quality of Service (QoS) ..... کیفیت خدمت

## R

Random ..... تصادفی

Reconfiguration ..... بازپیکربندی

Reconfiguration Manager ..... مدیر پیکربندی

Recurrent Neural Network (RNN) شبکه‌های عصبی تکرار شونده

Reinforcement Learning ..... یادگیری تقویتی

Replacement ..... جایگزینی

## S

Saturation ..... اشباع

Scheduling ..... زمان‌بندی

Second Chance Algorithm ..... الگوریتم شانس دوم

Self-Similarity ..... خودهمسانی

Sequential ..... ترتیبی

Solid State Disk (SSD) ..... دیسک‌های حالت جامد

Spatial ..... مکانی

Storage Networking Industry Association (SNIA) . اسنیا

Strided ..... گام‌دار

Supervised Learning ..... یادگیری با نظارت

Supprt Vector Machine ..... ماشین بردار پشتیبانی

# T

Temporal ..... زمانی

Tiering ..... رده‌بندی داده

Time Intervals ..... بازه‌های زمانی

Time Series ..... دنباله‌های زمانی

# U

Un-Supervised Learning ..... یادگیری بدون نظارت

# V

Virtualization ..... مجازی‌سازی

# W

Workload ..... بار کاری

Workload Monitoring ..... ناظر بار کاری

Write-Back ..... پس‌نوشت

Write-Throgh ..... رونوشت

# Improving Cache Performance of Data Storage Systems Using Machine Learning


## Abstract

Emerging *Solid State Drives* (SSDs) have performance advantages over traditional *Hard Disk Drives* (HDDs). Higher price per capacity and limited lifetime, however, prevents enterprise data centers to entirely replace HDD-based storage subsystems with SSDs. Thus, SSD-based caching has been widely employed in data centers to benefit from higher performance of SSDs while minimizing overall cost. *Input/Output* (I/O) workloads exhibit unpredictable and highly random behavior which makes conventional algorithms such as *Least Recently Used* (LRU) not able to provide high hit ratio as they employ linear localities. In addition to poor performance, such algorithms also shorten SSD lifetime with unnecessary cache replacements. In this research, we propose the first reconfigurable SSD-based cache architecture using *Recurrent Neural Networks* (RNNs) to characterize ongoing workloads and can optimize itself towards higher cache performance while increasing SSD lifetime. Proposed method consists of an offline and an online phase. In the offline phase, we try to learn various workloads and predict their behavior. In the second phase, collected information gets used to identify performance critical data pages to be cached. Experimental results show that proposed method can characterize workloads with an accuracy up to 94.6% for SNIA I/O workloads. The proposed method can perform similarly to optimal cache algorithm by an accuracy of 95% on average and outperforms previous SSD caching architectures by having up to 7x and 10x higher hit ratio and endurance, respectively.



**Keywords:** Data Storage Systems, Machine Learning, Cache, SSD, Characterization.

Sharif University of Technology
Department of Computer Engineering

## Ms Thesis
## Computer Architecture Engineering

# Improving Cache Performance of Data Storage Systems Using Machine Learning

By

**Shahriar Ebrahimi**

Supervisor

**Dr. Hossein Asadi**

Summer of 2017