زبان پیاده سازی : جاوا

محیط های مورد بررسی : لپتاپ - Digital Ocean (Cloud SSD) و برد RaspberryPI

نکته :در انجام تمامی مراحل پروژه ، حافظه به صورت کنترل شده (حداکثر ۱۲۸ مگابایت حافظه اصلی) و با خط فرمان زیر اجرا شد :

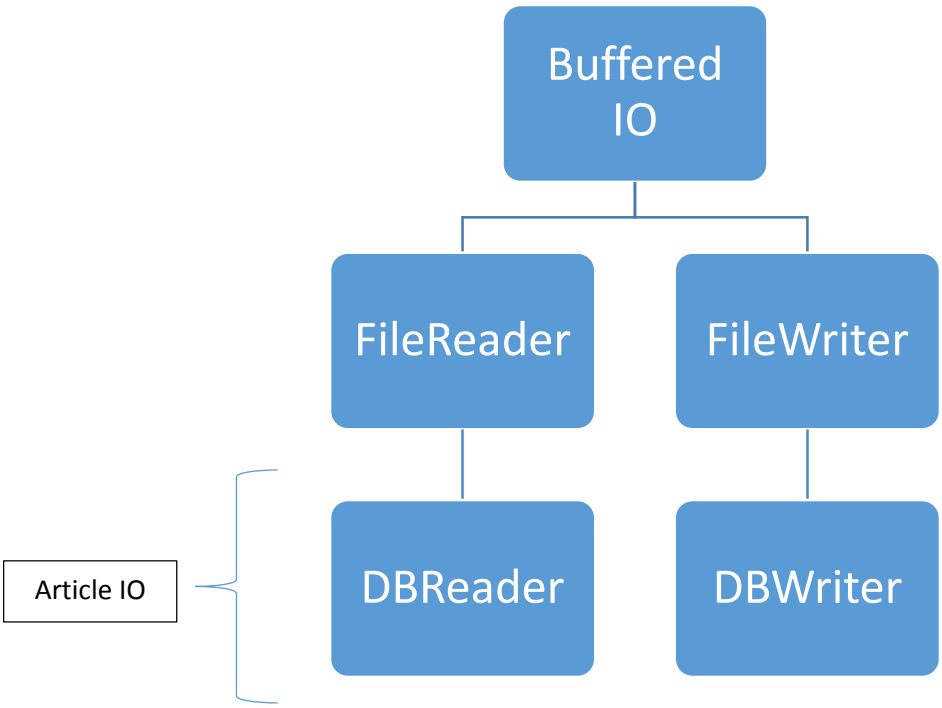
```
java -Xms128m -Xmx128m -cp ...
```

خواندن تمام فایل و بررسی اندازه‌بافر بهینه

کلاس‌هایی که زبان برنامه نویسی جاوا برای خواندن و نوشتن روی فایل در اختیار قرار میدهد به طور کلی دو دسته‌ی Stream ها و Reader ها هستند که Stream ها قابلیت خواندن و نوشتن در سطح بایت را در اختیار ما قرار می دهند و طبعا کارایی و سرعت بسیار بالاتری دارند (این موضوع مورد بررسی عملی هم قرار گرفت) اما به دلیل نوع رمزگزاری فایل دیتابیس که به صورت UTF-8 و بیش از یک بایت برای هر کاراکتر مصرف می‌شود ، استفاده از این نوع io ها راحت نیست.

از طرفی کلاس های *Data هم به دلیل بررسی های چند باره ای که انجام می دهند سربار بسیار بالایی را هنگام اجرا دارند.

راهکاری که استفاده شد استفاده از Buffered Reader ها بود که امکان خواندن سریع داده های UTF-8 و همچنین بافر کردن داده ها از حافظه‌ی ثانویه برای سرعت پردازش بالا را دارا می باشد و تغذیه کردن ورودی آن از یک IOStream برای سرعت خواندن و نوشتن بالاتر بود . (در ادامه به بررسی اشکالات Buffered Reader و راهکارهایی برای آن پرداخته خواهد شد)



(شمای کلی پیاده سازی)

نکته ای که در اینجا مطرح هست نحوی خواندن داده ها از یک Buffered Reader هست . بهترین راهکاری که این کلاس در اختیار ما قرار می داد خواندن فایل به صورت خط به خط (و بافر شده) بود که کارایی خوبی برای این ساختار داده ی دیتابیس ندارد. از طرفی امکان Extend کردن تمامی بخش های این کلاس وجود نداشت - در پکیج io یک نسخه ی کامل بازنویسی شده از این کلاس افزوده شده که امکان تعیین کاراکتر پایان خط را دارا می‌باشد.

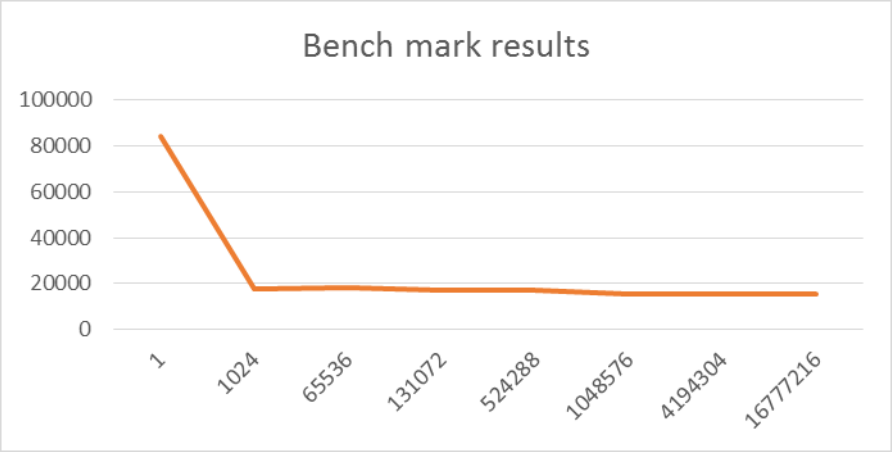
البته تنها بررسی یک کاراکتر برای پیدا کردن محدوده‌ی یک مقاله کافی نبود و این ایراد در لایه بالاتر یعنی DB Reader به درستی رفع می شود.

نتایج بررسی

نکته ای که لازم به تاکید هست تمامی این بررسی ها با این فرض در نظر گرفته شده اند که دیتای بافر شده به صورت مقاله مقاله باید تفکیک شود . (در صورتی که اگر برای خواندن از Buffered Reader بلاک هایی منظم و با طول ثابت و بدون مکث صرفا خوانده شود قطعا زمان پردازشی بسیار پایین تری خواهد داشت - که البته اینگونه نتایج عملا غیرواقعی می بودند ...)

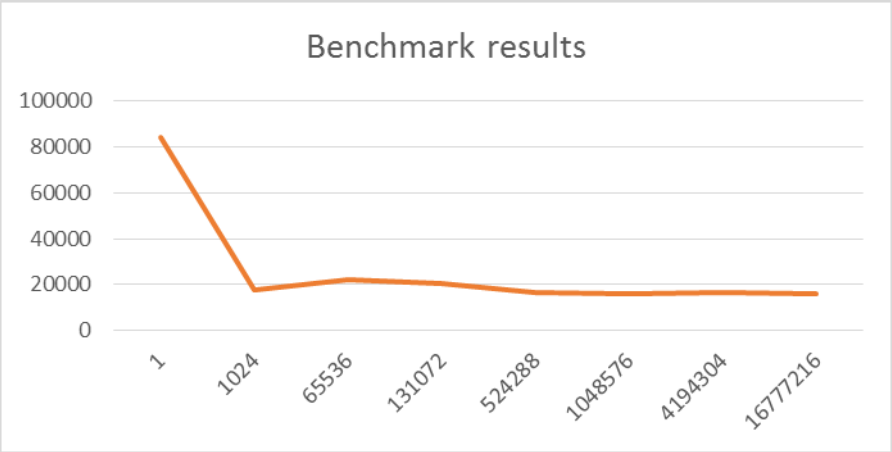
سیستم HDD1 - AData – A

سیتم عامل : لینوکس فایل سیستم : EXT4 Block Size : 4096 IO Min : 512



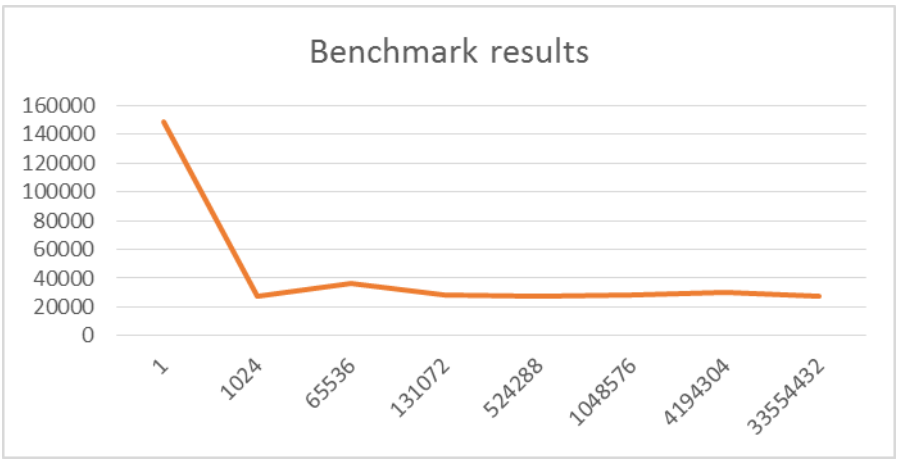
سیستم Sumsung -HDD2 - – A

سیتم عامل : لینوکس فایل سیستم : NTFS Block Size : 4096 IO Min : 4096



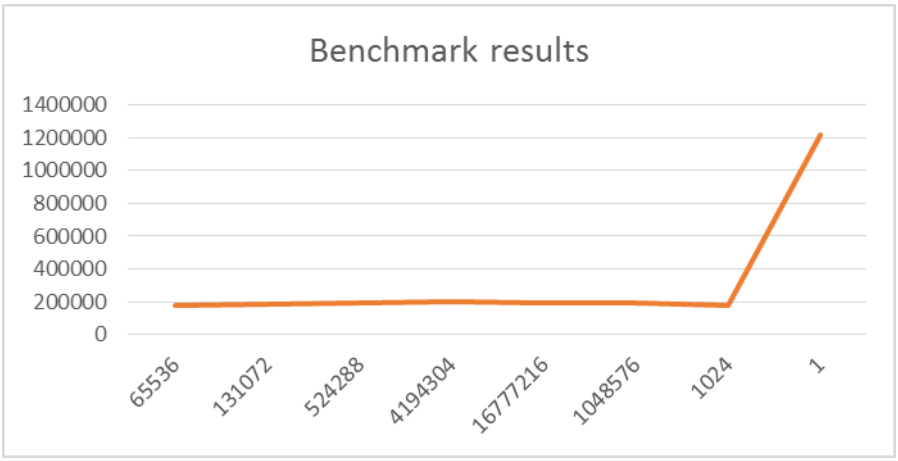
سیستم C (digital ocean)

سیتم عامل : لینوکس فایل سیستم : EXT4 Block Size : 4096 IO Min : 512



سیستم D - برد Raspberry Pi

سیتم عامل : لینوکس فایل سیستم : EXT4 Block Size : 4096 IO Min : 512



جست و جو و بررسی آماری کلمات در دیتابیس

با توجه به پیاده سازی جامعی که در مرحله قبل صورت گرفته بود این مرحله تنها با افزودن چند خط میسر شد.

برای اطمینان این کار به روش های خط به خط و کاراکتر به کاراکتر هم صورت گرفت که بیشترین بازدهی را روش خواندن بافر شده ی مقالات داشت.

./run.sh i IR_FarsiDatabase.txt

Indexing using buffer size : 4.0 KiB

Read Tags : 4808

Took : 00:00:30:723

قرآن Count[۴۵۳۱۵]: Observed in tags[۴۰۸۳]: Max count in tags[۲۴۰]:

تعطیل Count[۲۸۷۹۲]: Observed in tags[۴۵۳۴]: Max count in tags[۸۹]:

سریال Count[۱۳۲۰۲]: Observed in tags[۲۳۲۲]: Max count in tags[۶۵]:

مرتب سازی و ذخیره ی دیتابیس

./run.sh s IR_FarsiDatabase.txt out.txt

Finishing all Queue jobs

Took : 00**:02:57:548**

شاید یکی از سخت ترین و چالش برانگیز ترین قسمت های این پروژه ذخیره سازی دیتابیس بود. و مشکل اساسی سرعت پایین پردازش مقاله های حجیم و نوشتن آن ها بر روی دیسک بود.

چندین راهکار برای مدیریت زمان های بین خواندن و نوشتن پیاده سازی شد که مبنای تمامی راهکار ها استفاده از یک صف و ۲ (Thread) مجزا یکی برای استخراج داده و یکی برای تبدیل آن به داده های مرتب شده برای بازنویسی بر روی دیسک بود.

ایراد استفاده از صف این بود که به زودی به خاطر اختلاف سرعت صف بیش از اندازه طولانی می شد و به دلیل محدودیتی که برای حافظه وجود داشت امکان نگه داشتن آن وجود نداشت.

راهکار پیشنهادی : تعیین محدودیت برای صف و توقف عملیات پرکردن صف تا خالی شدن آن

محدودیت های متفاوتی از جمله حافظه باقی مانده برای جاوا (heap) و تعداد مقاله های در صف بررسی شد. و در مجموع از ترکیبی از این ۲ استفاده شد.

البته تنها بخش نوشتن بر روی فایل زمانگیر نبود و پردازش متن هم زمان زیادی صرف می کرد (Regex ها) بنابراین این امکان وجود داشت که با تعریف کردن ۲ صف جداگانه اینگونه پردازش ها را به چندین Thread مجزا واگذار کرد که متاسفانه فرصت پیاده سازی آن نبود.

به عنوان یه راهکار جایگزین Priorityاین Thread بالابرده شد.