باسمه تعالی

با توجه به کدهاب داخل MAS\_system میخوام این سناریو را پیاده سازی کنی.

کدها را داخل یک فولدر JDL\_system پیاده سازی کن. من از رویکرد چندعامله نمیخوام استفاده بشه.

من میخوام دو حالت centralize agent و distribute agents را درنظر بگیری.

در حالت centralize، تنها یک عامل وجود دارد که میخواهد دو کوئری q1 و q2 را به صورت متوالی اجرا کند به صورتی که نتایج q1 به q2 به عنوان ورودی وارد شود.

در حالت distribute agents ما دو عامل worker و master داریم. در عامل worker، کوئری q1 اجرا می شود و در عامل master کوئری q2 اجرا می گردد. لذا این دو کوئری روی دو عامل مجزا اجرا می گردد.

در هر دو سیستم (متمرکز و توزیعی) نتایج کوئری q1 روی یک صف (queue) به نام BUS قرار داده می شوند. یک فضای مشخصی برای BUS باید درنظر بگیری یعنی اینکه چند تا triple گنجایش دارد.

من میخوام نشان دهم که رویکرد توزیعی از رویکرد متمرکز بهتر است از منظر زمان اجرای کوئری، فضای حافظه مصرفی .

در حالت اول : من فقط میخوام که به تعداد 60 هزار triple، داده خام ارسال شود و عملیات در هر دو سیستم شروع به کار کند (اول متمرکز بعد توزیعی) و q1 اجرا و بعدش q2 اجرا شود. و نتایج کارایی (زمان پایان کار) مشخص شود. کارایی هم ارزیابی شود در هر دو روش.

در حالت دوم: من میخوام ...

**JDL\_SYStem (1)**

- دلیل اصلی بهتر شدن زمان در توزیعی: هم‌پوشانی مراحل. در روش توزیعی، تولید خروجی q1 (worker) و ساخت/انباشت گراف مشاهدات برای q2 (master) به‌صورت هم‌زمان پیش می‌رود. این پایپ‌لاین کردن، زمان بیکاری بین مراحل را حذف می‌کند؛ در متمرکز، q1 کاملاً تمام می‌شود و سپس q2 اجرا می‌شود.

- دلیل کاهش پیک حافظه: BUS با ظرفیت محدود نقش backpressure دارد و master به‌صورت تدریجی خروجی‌های q1 را در گراف وارد می‌کند. چون مصرف هم‌زمان با تولید رخ می‌دهد، داده‌های میانی کمتر در حافظه معطل می‌مانند؛ در نتیجه پیک حافظه پایین‌تر می‌آید.

- نکته فنی در پایتون: با وجود GIL، بخشی از کار rdflib در توابع C/IO-مانند انجام می‌شود و الگوی تولید-مصرف باعث هم‌پوشانی مؤثر می‌شود. بنابراین حتی با threads هم بهبود زمان مشاهده می‌شود.

اگر بخواهید بیشتر بهبود دهید:

- کاهش یا تنظیم ظرفیت BUS برای تعادل بین فشار حافظه و هم‌پوشانی

- افزایش موازی‌سازی واقعی با multiprocessing به‌جای threading برای بخش‌های CPU-bound

- اجرای q2 نهایی روی گراف در یک پردازش مجزا یا استفاده از backendهای سریع‌تر SPARQL

**JDL\_system (2)**

**Prompt:**

حالا میخوام این تغییرات را بدی.  
میخوام هر دو روش توزیعی و متمرکز به صورت هم‌پوشانی انجام شوند (رویکرد متمرکز هم مثل توزیعی باشد و همپوشانی داشته باشد).  
اما میخوام گه اجرای کوئری q2 روی متمرکز سنگین تر شود نسبت به اجرای q2 در توزییع و دیرتر بتواند داده از BUS بردارد و لذا BUS فول و پر بشه و منجر به توقفاجرای q1 بشه در متمرکز و اینطوری تاخیر بخوره (چون q2 نتونسته سریع داده ها بردارد از BUS و لذا BUS توسط خروجی های q1 پر شده و باید یک توقفی روی اجرای q1 انجام شود.)  
از طرفی این سنگین شدن اجرای کوئری q2 در متمرکز میخوام به صورت مصرف شدن زیاد RAM نشون داده بشه و ارزیابی ها که در متمرکز چون q2 سنگین بوده ram زیاد مصرف شده و BUS پر شده.  
اما در توزیعی جون فرض میکنیم هر کوئری روی یک سیستم مجرا است، اجرای q2 منجر به تاخیر در اجزای q1 نمی شود.

- Centralized

- Peak RAM ≈ 381 MB: مطابق تنظیمات شما، مصرف RAM اضافی 300MB در مصرف‌کننده q2 فعال بود (+ حافظه پایه پردازش). این باعث شد حافظه اوج به‌طور محسوسی بالا برود.

- تاخیر 20ms در مصرف هر batch: این کندی در مصرف باعث شد BUS گاهی پر شود و تولید q1 منتظر بماند (backpressure). با این وجود، هم‌پوشانی q1/q2 حفظ شد و کل زمان به 15.425s رسید.

- خروجی 12 تریپل: q2 یک‌بار روی کل گراف Observations اجرا شده و تعداد خروجی ثابت است.

- Distributed

- Peak RAM ≈ 59 MB: در رویکرد توزیعی، q2 سنگین‌سازی نشده و مصرف RAM پایین مانده است. ضمن اینکه تولید و مصرف روی دو نقش جدا انجام می‌شود و حافظه تجمعی در یک فرآیند کمتر می‌شود.

- زمان 15.903s: با وجود حافظه کمتر، سربار همزمانی/هماهنگی و انتقال داده بین نقش‌ها، و اجرای نهایی q2 روی گراف کامل باعث شده زمان کمی بیشتر از حالت مرکزی شود. هم‌پوشانی کمک کرده که هم‌سطح بماند، اما چون در مرکزی q2 هم‌فرایند است و هزینه‌های بین‌نقشی ندارد، کمی جلو افتاده است.

- خروجی 12 تریپل: همان منطق q2 روی گراف نهایی، پس خروجی برابر است.

- برداشت کلیدی

- تفاوت اصلی: در مرکزی q2 عمداً سنگین شده و RAM بالایی مصرف کرده که به پر شدن BUS و توقف‌های q1 منجر می‌شود، اما به‌خاطر نبود overhead بین‌فرایندی، زمان کل از توزیعی اندکی کمتر شده است. در توزیعی، حافظه بسیار کمتر است ولی اندکی سربار هماهنگی زمان را بالا برده است.

- اگر هدف نشان‌دادن برتری زمانی توزیعی هم هست، می‌توانید:

- ظرفیت BUS را کمتر کنید تا backpressure مؤثرتر شود.

- delay مرکزی را افزایش دهید (مثلاً 40–60ms).

- q2 توزیعی را سبک‌تر یا با chunkهای کوچک‌تر مصرف کنید تا سریع‌تر تخلیه کند.