به نام خدا

تمرین مقایسه عملکرد یک مدل بر پایه پیاده سازی ساده یک سیستم multi thread **با مدلی بر مبنای استفاده از سرویس های ارائه شده برای** multi thread **به طور مثال توسط** Apache

نام و نام خانوادگی: سید محمد حسین طباطبائی

شماره دانشجویی : 40361631005

نام استاد: سرکار خانم دکتر مریم لطفی

درس : سیستم های توزیع شده

فهرست مطالب

مقدمه:

مقایسه‌ی مدل‌های مولتی‌ترد و روش‌های مولتی ترد پیشرفته

در این تمرین، هدف بررسی و مقایسه‌ی عملکرد مدل‌های مختلف اجرای موازی با تمرکز ویژه بر مولتی‌ترد (Multithreading) و سایر تکنیک‌های پیشرفته در زبان Python است. اجرای موازی یکی از مباحث کلیدی در بهبود کارایی نرم‌افزارهای پیچیده و زمان‌بر است، به‌ویژه زمانی که نیاز به انجام تعداد زیادی کار مشابه به‌طور همزمان باشد.

برای رسیدن به این هدف، چندین مدل رایج اجرای موازی در Python پیاده‌سازی و مورد آزمون قرار گرفتند:

1. مولتی‌ترد سنتی(Traditional Multithreading)
2. Thread Pools و Task Queues با تکنیک‌های Load Balancing
3. برنامه‌نویسی ناهمگام مبتنی بر(Asynchronous Programming) Event Loop

هر کدام از این مدل‌ها با انجام تعداد زیادی وظیفه مشابه و ثابت، مورد سنجش زمان اجرا قرار گرفتند تا کارایی و ویژگی‌های منحصر به فرد آن‌ها در شرایط مختلف مشخص شود.

در ادامه، ضمن ارائه‌ی کدهای پیاده‌سازی شده، خروجی‌ها و تحلیل‌های مقایسه‌ای آورده شده است.

بخش اول: مقایسه اجرای مولتی‌ترد دستی (Manual Threading) و ThreadPoolExecutor

1- مولتی‌ترد دستی(Manual Threading) :

در این روش برای هر یک از وظایف، یک ترد مستقل ساخته و اجرا می‌شود. با اجرای ۱۰۰۰ تا ۱۰۰۰۰ وظیفه، معادل همان تعداد ترد در سیستم ساخته می‌شود. پس از اجرای همه تردها، برنامه منتظر می‌ماند تا همگی به پایان برسند (با فراخوانی .(joinاین مدل گرچه ساده است، اما به دلیل ساخت تعداد زیاد ترد، سربار مدیریتی بالایی دارد.

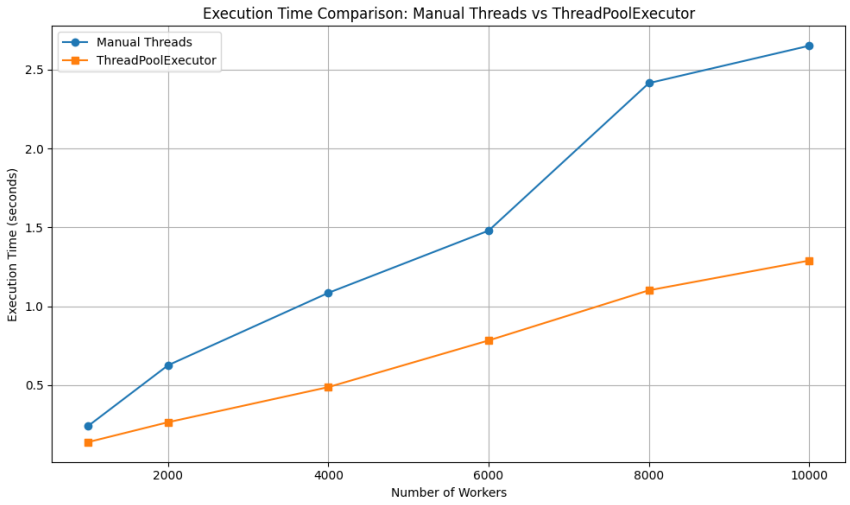
2- ThreadPoolExecutor (مدل Thread Pool مشابه سرویس‌های آپاچی):

در این مدل از یک صف وظیفه و یک مجموعه ثابت از تردها (مثلاً ۱۰۰ عدد) استفاده می‌شود. وظایف به تردهای موجود در ThreadPool سپرده می‌شوند و پس از انجام هر وظیفه، ترد آماده انجام وظیفه بعدی می‌شود. این روش منابع را به‌صورت بهینه‌تری مدیریت کرده و از هزینه ساخت هزاران ترد جلوگیری می‌کند.

نتایج اجرای کد:

تعداد وظایف	زمان اجرای Manual Threading (ثانیه)	زمان اجرای ThreadPoolExecutor (ثانیه)
1000	0.2411	0.1395
2000	0.6268	0.2657
4000	1.0854	0.4882
6000	1.4797	0.7838
8000	2.4146	1.1015
10000	2.6511	1.2898

نمودار مقایسه‌ای:



در نمودار رسم‌شده، روند افزایش زمان اجرا در دو مدل قابل مشاهده است. اجرای دستی تردها با افزایش تعداد وظایف به‌صورت تقریباً خطی افزایش می‌یابد، در حالی که استفاده از ThreadPool باعث کنترل بهتر زمان اجرا شده و شیب رشد بسیار ملایم‌تری دارد.

تحلیل نتایج و توضیحات فنی:

Manual Threading

- ایجاد هزاران ترد منجر به مصرف بالای حافظه، بار پردازشی بیشتر و سوئیچینگ زیاد بین تردها می‌شود.
- در مقیاس بزرگ‌تر (مثلاً ۸۰۰۰ یا ۱۰۰۰۰ وظیفه)، زمان اجرا به شدت افزایش می‌یابد.

ThreadPoolExecutor

- با محدود کردن تعداد تردهای فعال (مثلاً به ۱۰۰ عدد)، بار سیستم کاهش یافته و وظایف به صورت صف‌بندی و متوازن بین تردها توزیع می‌شوند.
- این روش مناسب برای سناریوهای I/O-bound است که وظایف زمان‌بری ندارند اما تعدادشان زیاد است.

جمع‌بندی:

در شرایطی که تعداد وظایف زیاد است و هر وظیفه کار کوتاهی انجام می‌دهد، استفاده از ThreadPoolExecutor مزایای زیادی دارد. این مدل هم‌زمانی بالا را بدون تحمیل بار زیاد به سیستم ارائه می‌دهد و به همین دلیل در سرویس‌های تولیدی و سیستم‌های بزرگ (مانند سرورهای وب یا سرویس‌های مبتنی بر میکروسرویس) بسیار پرکاربرد است.

بخش دوم: مقایسه اجرای موازی با مولتی‌ترد دستی (Manual Threading) و Task Queue با Load Balancing

1- مولتی‌ترد دستی (Manual Threading)

در این مدل، برای هر وظیفه یک ترد مجزا ساخته و اجرا می‌شود؛ بنابراین اجرای ۱۰۰۰ تا ۱۰۰۰۰ وظیفه منجر به ساخت همان تعداد ترد مستقل در سیستم می‌شود. مدیریت این حجم از تردها، از جمله ایجاد، زمان‌بندی و سوئیچینگ بین آن‌ها، سربار زیادی دارد و باعث افت کارایی می‌شود.

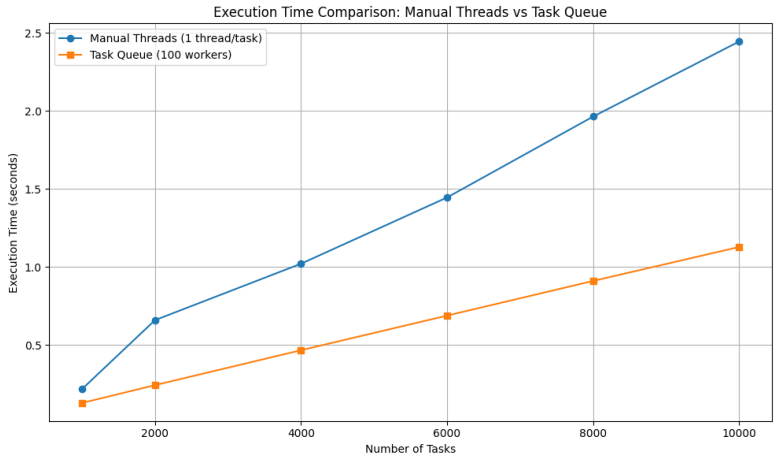
2- Task Queue با Load Balancing

در این روش، ابتدا تمام وظایف در یک صف (queue) قرار می‌گیرند. سپس تنها تعداد محدودی ترد کارگر – (Worker Thread) مثلاً ۱۰۰ عدد – ایجاد می‌شود. این تردها به‌صورت پیوسته وظایف را از صف دریافت کرده و اجرا می‌کنند. پس از پایان هر وظیفه، بلافاصله سراغ وظیفه‌ی بعدی می‌روند. این روش با ایجاد تعادل بار (Load Balancing) بین تردها، موجب استفاده مؤثرتر از منابع می‌شود.

نتایج اجرای کد:

تعداد وظایف	زمان اجرای Manual Threading (ثانیه)	زمان اجرای Task Queue (ثانیه)
1000	0.2188	0.1296
2000	0.6601	0.2437
4000	1.0215	0.4674
6000	1.4457	0.6890
8000	1.9636	0.9110
10000	2.4444	1.1285

نمودار مقایسه‌ای:



نمودار مربوطه بر اساس همین داده‌ها رسم شده و به‌خوبی نشان می‌دهد که مدل Task Queue با افزایش تعداد وظایف، رشد ملایم‌تری در زمان اجرا دارد، در حالی که زمان اجرای مدل Manual Threading سریع‌تر افزایش می‌یابد.

تحلیل نتایج و توضیحات فنی:

Manual Threading

- ایجاد هزاران ترد مستقل منجر به مصرف زیاد حافظه و افزایش چشمگیر در زمان‌بندی و سوئیچینگ بین تردها می‌شود.
- با افزایش مقیاس، کارایی به‌صورت خطی یا حتی فوق‌خطی افت می‌کند.

Task Queue با Load Balancing

- با وجود تنها ۱۰۰ ترد فعال، صف وظایف باعث استفاده کامل و مداوم از تردها می‌شود.
- تعادل بار طبیعی حاصل از ساختار صف موجب جلوگیری از بیکاری تردها و افزایش بهره‌وری سیستم می‌گردد.
- ساختار صف‌بندی شبیه به مدل‌های مورد استفاده در سیستم‌های تولیدی بزرگ نظیر Apache یا RabbitMQ عمل می‌کند.

جمع‌بندی:

مدل صف وظیفه (Task Queue) با تعداد محدودی ترد کارگر، در سناریوهایی با تعداد زیاد وظیفه و محاسبات سبک I/O-bound) یا شبه-خواب(، نسبت به مدل مولتی‌ترد دستی، زمان اجرا را به شکل قابل توجهی کاهش می‌دهد. این بهبود حاصل کاهش سربار ساخت ترد و افزایش بهره‌برداری از تردهای فعال است. این رویکرد یکی از پایه‌های اصلی طراحی سیستم‌های توزیع‌شده و مقیاس‌پذیر امروزی به شمار می‌رود.

بخش سوم: مقایسه اجرای مولتی‌ترد دستی (Manual Threading) و برنامه‌نویسی ناهمزمان(Asynchronous Programming - Event Loop)

1- مولتی‌ترد دستی:

در این مدل، برای هر وظیفه (task) یک ترد مستقل ساخته و اجرا می‌شود. درنتیجه اجرای ۱۰۰۰ تا ۱۰۰۰۰ وظیفه منجر به ساخت همان تعداد ترد مجزا خواهد شد. این شیوه موجب افزایش قابل توجه سربار پردازشی سیستم، شامل هزینه‌های زمان‌بندی، سوئیچینگ بین تردها و مصرف حافظه می‌گردد.

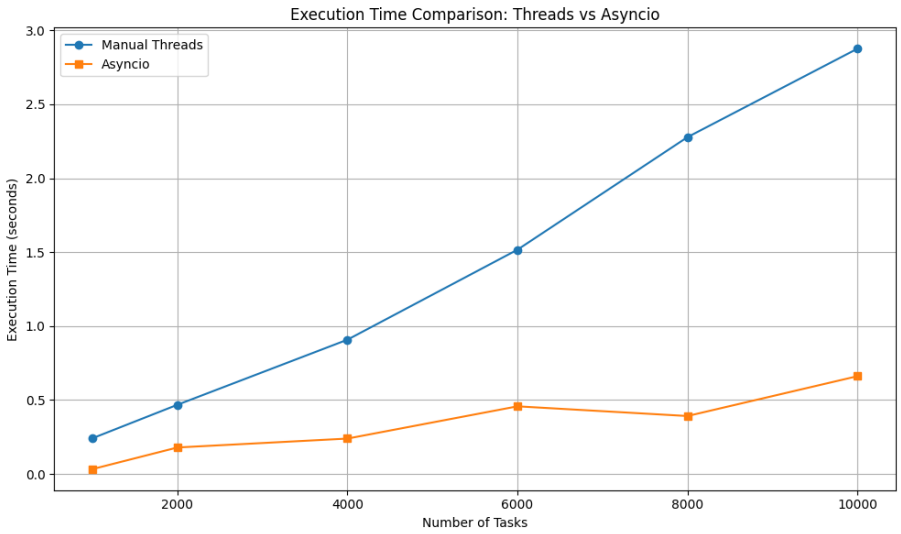
2- برنامه‌نویسی ناهمزمان(Asynchronous Programming) :

در این رویکرد، به‌جای ایجاد تردهای مجزا، از ماژول asyncio و مکانیزم حلقه رویداد (event loop) در پایتون استفاده می‌شود. وظایف به‌صورت coroutine تعریف شده و با استفاده از await asyncio.sleep، تأخیر شبیه‌سازی می‌شود بدون اینکه CPU واقعاً مسدود گردد. این اجازه می‌دهد کنترل به سایر وظایف منتقل شده و همه وظایف به‌صورت غیرمسدودکننده اجرا شوند.

نتایج اجرای کد

تعداد وظایف	زمان اجرای Manual Threading (ثانیه)	زمان اجرای Asyncio (ثانیه)
1000	0.2415	0.0329
2000	0.4680	0.1793
4000	0.9080	0.2395
6000	1.5165	0.4575
8000	2.2777	0.3920
10000	2.8759	0.6614

نمودار مقایسه‌ای:



نمودار رسم‌شده بر اساس این داده‌ها به‌وضوح نشان می‌دهد که زمان اجرای مدل ناهمزمان با افزایش تعداد وظایف، رشد بسیار کندتری دارد و نسبت به مدل مولتی‌ترد عملکرد بسیار بهتری ارائه می‌دهد.

تحلیل فنی و مقایسه عملکرد:

Manual Threading

- ساخت هزاران ترد مستقل باعث تحمیل سربار سنگینی به سیستم می‌شود.
- سیستم‌عامل باید بین تردها سوئیچ کند که منجر به اتلاف زمان CPU و کاهش بهره‌وری می‌شود.

Asyncio

- به‌جای استفاده از منابع زیاد سخت‌افزاری، با استفاده از حلقه رویداد تنها یک یا چند ترد را درگیر می‌کند.
- در سناریوهای **I/O-bound**، مانند تأخیرهای مصنوعی یا ارتباطات شبکه‌ای، این مدل بسیار مؤثر و سریع عمل می‌کند.
- بهره‌وری منابع بالا است زیرا هیچ تردی بلااستفاده باقی نمی‌ماند و اجرای وظایف به شکل موازی و بهینه پیش می‌رود.

جمع‌بندی:

مدل ناهمزمان مبتنی بر asyncio و حلقه رویداد، در سناریوهای I/O-bound مانند این آزمایش، برتری چشم‌گیری نسبت به مولتی‌ترد دستی دارد. برخلاف رویکرد ایجاد تردهای زیاد که نیازمند منابع سخت‌افزاری گسترده و زمان مدیریت بالا است، مدل ناهمزمان با استفاده‌ی بهینه از منابع، امکان مدیریت هزاران وظیفه را با حداقل سربار فراهم می‌کند. این تکنیک در طراحی سیستم‌های پاسخ‌گو، مقیاس‌پذیر و بلادرنگ مانند وب‌سرورها، خزنده‌های شبکه و پردازش داده‌های بلادرنگ نقشی اساسی ایفا می‌کند.

بخش چهارم: مقایسه اجرای Threading + Requests و Asyncio + Aiohttp در وظایف شبکه‌ای(Network-bound)

هدف آزمایش:

در این بخش، هدف بررسی و مقایسه‌ی عملکرد دو رویکرد متداول برای اجرای وظایف شبکه‌ای به‌صورت هم‌زمان است؛ وظایفی که ماهیت آن‌ها Network-bound بوده و شامل ارسال درخواست‌های HTTP با تأخیر مصنوعی هستند.

هر دو مدل باید تعداد مختلفی از درخواست‌ها (از ۵ تا ۱۰۰ عدد) را به آدرس تستی https://httpbin.org/delay/1 ارسال کنند. این سرویس به‌صورت مصنوعی هر درخواست را دقیقاً ۱ ثانیه به تأخیر می‌اندازد، تا شرایط واقعی یک درخواست اینترنتی با زمان پاسخ بالا شبیه‌سازی شود.

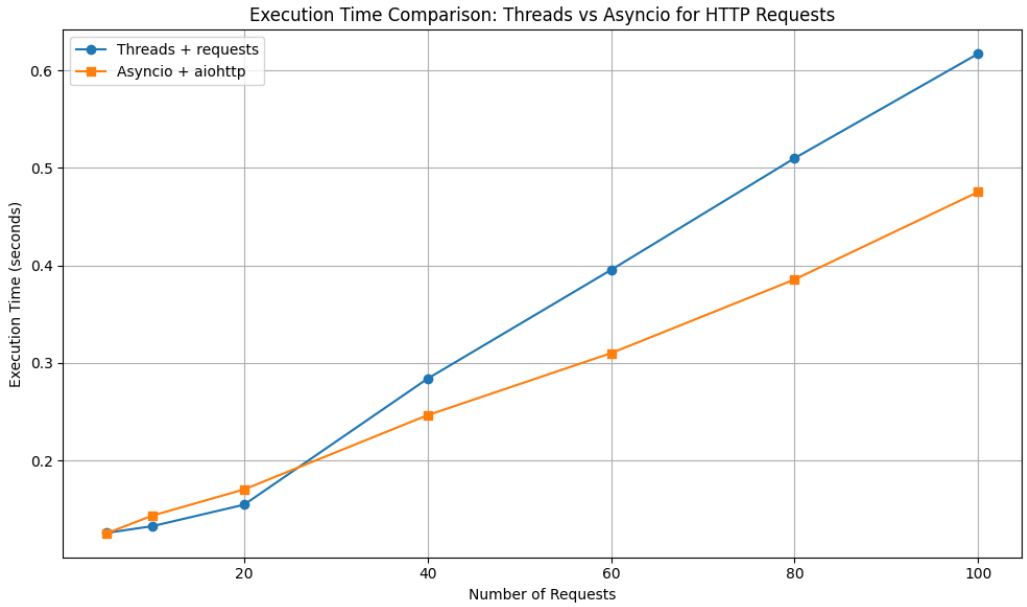
مدل‌ها

1. مدل اول: استفاده از Threading به همراه کتابخانه‌ی requests
   - برای هر درخواست یک ترد مجزا ایجاد می‌شود.
   - کتابخانه requestsبه‌صورت هم‌زمان اما مسدودکننده (blocking) عمل می‌کند.
   - این مدل سنتی معمولاً در پروژه‌های ساده استفاده می‌شود، اما در مقیاس بالا سربار زیادی دارد.
2. مدل دوم: استفاده از Asyncio به همراه aiohttp
   - تمامی درخواست‌ها به صورت ناهمزمان و بدون مسدود کردن پردازنده مدیریت می‌شوند.
   - aiohttp به عنوان نسخه‌ی async کتابخانه HTTP برای پایتون طراحی شده است و با asyncio به‌خوبی هماهنگ است.
   - این روش در بسیاری از سیستم‌های مقیاس‌پذیر مدرن استفاده می‌شود مثل وب‌سرورها یاcrawler ها.

نتایج اجرای کد:

تعداد درخواست	زمان اجرای Threading (ثانیه)	زمان اجرای Asyncio (ثانیه)
5	0.13	0.13
10	0.13	0.14
20	0.15	0.17
40	0.28	0.25
60	0.40	0.31
80	0.51	0.39
100	0.62	0.48

نمودار مقایسه‌ای:



نمودار مقایسه‌ای مربوط به زمان اجرا برای دو مدل، بر اساس همین داده‌ها ترسیم شده و به‌وضوح روند بهبود عملکرد در مدل ناهمزمان را با افزایش تعداد درخواست‌ها نشان می‌دهد.

تحلیل فنی و مقایسه عملکرد:

مدل Threading + Requests

- برای هر درخواست یک ترد ساخته می‌شود که منجر به سربار پردازشی در مدیریت و زمان‌بندی توسط سیستم‌عامل می‌گردد.
- ماهیت blocking کتابخانه requestsباعث می‌شود که هر ترد تا پایان پاسخ‌گویی مسدود بماند و امکان استفاده مجدد از منابع وجود نداشته باشد.

مدل Asyncio + Aiohttp

- با استفاده از یک حلقه رویداد، مدیریت هم‌زمانی به شکلی بسیار بهینه‌تر انجام می‌شود.
- به‌جای مسدود شدن، هر درخواست معلق مانده و در زمان انتظار کنترل به وظایف دیگر واگذار می‌شود.
- این مدل ضمن کاهش مصرف منابع سیستم، امکان مدیریت هم‌زمان هزاران درخواست را بدون ایجاد تردهای متعدد فراهم می‌کند.

جمع‌بندی:

با وجود آن‌که در مقیاس کوچک تفاوت زمان اجرا محدود است، اما با افزایش تعداد درخواست‌ها، مدل ناهمزمان asyncio + aiohttp به‌طور قابل توجهی کارایی بهتری نشان می‌دهد. این مزیت‌ها در پروژه‌های شبکه‌محور بزرگ مانند وب‌سرورها، REST APIها یا خزنده‌های وب به وضوح خود را نشان می‌دهند. مزایایی مانند:

- کاهش مصرف حافظه و پردازنده
- مقیاس‌پذیری بهتر
- مقاومت بیشتر در برابر خطاهای شبکه و امکانات کنترلی مانند timeout وretry

همگی async را به انتخابی حرفه‌ای‌تر برای سیستم‌های real-world تبدیل می‌کنند.

جمع‌بندی کلی:

در این تمرین، با هدف بررسی و مقایسه‌ی عملکرد چند مدل مختلف اجرای هم‌زمان در زبان پایتون، مجموعه‌ای از سناریوهای مشخص طراحی و تحلیل شد. مدل‌های مورد بررسی شامل اجرای دستی با تردها (manual threading) ، استفاده ازThreadPool ، صف وظیفه با بارگذاری متعادل(task queue) ، برنامه‌نویسی ناهم‌زمان مبتنی بر async/await ، و همچنین مقایسه‌ی ارسال درخواست‌های HTTP با threading و aiohttp بود.

هدف اصلی این پیاده‌سازی‌ها، درک بهتر نحوه‌ی اجرای موازی یا شبه‌موازی وظایف و سنجش تأثیر انتخاب مدل هم‌زمانی بر کارایی سیستم از منظر زمان اجرای کل بود. در سناریوهایی که وظایف شامل تأخیرهای مصنوعی یا عملیات I/O مانند ارسال درخواست HTTP بودند، به‌روشنی دیده شد که انتخاب مدل مناسب می‌تواند به شکل چشمگیری زمان اجرای کلی را کاهش دهد.

نتایج نشان دادند که استفاده از ThreadPool نسبت به اجرای دستی تردها، به دلیل مدیریت بهینه‌تر و کاهش سربارهای زمان‌بندی و ایجاد ترد، باعث بهبود عملکرد می‌شود. همچنین، مدل صف وظیفه با تعداد محدودی ترد و مدیریت متمرکز، ضمن بهینه‌سازی مصرف منابع، کاهش قابل توجهی در زمان اجرا ایجاد کرد.

اما در میان تمامی روش‌ها، برنامه‌نویسی ناهم‌زمان مبتنی بر حلقه رویداد (asyncio) به ویژه برای وظایف I/O-bound از جمله تأخیرهای شبکه‌ای و عملیاتsleep ، بهترین عملکرد را ارائه داد و توانست زمان اجرای کلی را در برخی موارد به صورت چندبرابر نسبت به مدل‌های مبتنی بر ترد کاهش دهد.

در سناریوی ارسال ۱۰۰ درخواست HTTP به صورت هم‌زمان، استفاده از ترکیب asyncio و aiohttp به جای threading وrequests ، حدود نیم ثانیه صرفه‌جویی در زمان اجرا به همراه داشت؛ تفاوتی که در کاربردهای شبکه‌ای و تحت بار سنگین، بسیار حائز اهمیت است.

در نهایت، این آزمایش‌ها نشان دادند که با انتخاب هوشمندانه‌ی مدل هم‌زمانی و توجه به ماهیت وظایف (CPU-bound یا (I/O-bound و محدودیت‌های منابع سیستم، می‌توان کارایی برنامه‌های پایتون را به طرز چشمگیری افزایش داد. انتخاب درست بین مدل‌های threading و async تنها به ویژگی‌های وظیفه وابسته نیست، بلکه باید شرایط اجرایی و منابع موجود را نیز به دقت مدنظر قرار داد تا بهترین تعادل بین مصرف منابع و سرعت اجرا حاصل شود.

پایان