Real-World-Application-Distributively-Asteroids-Processor

*Tal Yitzhak - 204260533*

*Assaf Magrisso - 201247020*

Technical Points:

Running the project –

Running the project requires exporting the worker and the manager code to jars (worker.jar & manager.jar), uploading them to an S3 bucket named Constants.bucketName, make them public (so that anybody can download them).

The project also requires that an AWSCredentials.zip file will be uploaded to S3 with a relevant password Constants. ZipFilePassword.

All the relevant jar libraries are in the maven dependencies, using maven build they will automatically be downloaded.

The local application can run throw the command line using the following command

Java –jar local.jar inputFileName.txt outputFileName n d

And a last argument that is up to the user`s choice – is it terminate / not.

\*We used t2.micro instances (ami-b73b63a0)

Did you think for more than 2 minutes about security?

Of course. We decided to secure the credentials like it is mentioned at the assignment instructions: "One way of doing that is by compressing the jar files with a password.".  
We have a file named AWSCredentials.zip that is protected by a password which is available on S3. Then, If the Local Application/Manager/Worker needs to perform some action that requires the access key/secret key, they download the file from S3 and with the password they have they can unzip it and extract the information they need.

- Did you think about scalability? Will your program work properly when 1 million clients connected at the same time? How about 2 million? 1 billion? Scalability is very important aspect of the system, be sure it is scalable!

In order to answer this question, we`ll first give an explanation of the way the manager handle local applications: We have 2 kinds of threads running inside the manager:

**Local Application Handler** – this thread is accepting new messages from local applications, and when a new one is coming, it doesn`t handle its request – it just opens up a new thread inside its own thread pool (with a fixed size), and keep answering new local application requests – so it is basically passing the mission of taking care of a local application to another thread – which makes him available to handle other requests. When million users send message to the Manager – he will be able to accept their messages, and the actual treatment is done by the thread pool. Handling a request also means the thread is responsible for opening a unique queue for that specific local application, so he could get the summary file ASAP when the workers are done analyzing his NASA relevant data. The fixed size thread pool represents the amount of locals that the manager can take care of.

**Workers Handler** – this thread is accepting new messages from workers when they are done with their work. It is very similar to the Local Application Handler – it`s responsibility is to get messages from workers and to pass that task to another thread in its own thread pool (also, fixed sized one). If a lot of analyzed data is sent from the workers side – it still gives attention to each message, which is processed and calculated by an available thread from the thread pool.

Once all tasks of a specific local application is done – we [immediately](https://www.google.co.il/search?client=firefox-b-ab&q=immediately&spell=1&sa=X&ved=0ahUKEwjmppa4hYvRAhWVOVAKHWR-DhEQvwUIFygA) (by the same thread that handled the last Atomic Task that was related to the tasks that was relevant to that specific local application) handle it and send a response back to the local application – so it can get the response ASAP, and we can clear the data structures.

***Aviv&Shahar :  yes, we did thought about scalability.***

***As part of out implementation, the number of Threads that we open is limited by the threadPool arguments,***

***that means, more clients to come doesn't overload the system over the limit we decided, but might take longer time to response to clients, because of limitation such as number of workers in EC2, number of api calls to nasa in an hour (1000).***

- What about persistence? What if a node dies? Have you taken care of all possible outcomes in the system? Think of more possible issues that might arise from failures. What did you do to solve it? What about broken communications? Be sure to handle all fail-cases!

We have a queue named "workersListener" for the workers to listen to new tasks. Once a worker got a new task to handle, it doesn`t delete the message from the queue until it`s done with the task – meaning if he dies, the task will still be visible in the queue for other workers to handle.

- maximum amount of 20 instances check.

- new api key from NSAS?

- close workers that does`nt

**OVER\_RATE\_LIMIT**

You have exceeded your rate limit. Try again later or contact us at https://api.nasa.gov/contact/ for assistance

- Threads in your application, when is it a good idea? When is it bad? Invest time to think about threads in your application.

תכלס אפשר להוסיף פה חלק מהתוכן של התרדים שנכתב למעלה, למעלה עדיף אולי לכתוב על הפרמטרים של התרד פול ופה לפרט על התרדים

- Do you understand how the system works? Do a full run using pen and paper, draw the different parts and the communication that happens between them

מגריסו תן פה ציור יפה או משהו של איך שעשינו את המערכת עם כל התורים.

- Did you manage the termination process? Be sure all is closed once requested!

להסביר פה איך כל התהליכים מסתיימים ומי סוגר אותם

אפשר לפרט גם על מחיקת כל האלמנטים הרלוונטים

- Did you take in mind the system limitations that we are using? Be sure to use it to its fullest

-NASA API limit

-20 instances limit

- Are all your workers working hard? Or some are slacking? Why?

- add a feature to support that

תלוי באנליזה שיש לוורקר לעשות

אם כולם עובדים ביחד

בתהליך העליה הגיוני שאחד יקח עבודה לשני

-Lastly, are you sure you understand what distributed means? Is there anything in your system awaiting another

<http://pastebin.com/x9wry4bg>

זה מה ששחר ואביב עשו

שיהיה לנו לעזר כשנכתוב את התשובות

הם כתבו דברים נכונים..