Task Classification using Lexical Analysis of Program files

Shreyas Seshadri
Information Technology
National Institute of Technology
Karnataka
Surathkal
shreyasseshadri@gmail.com

Akhil N Kashyap
Information Technology
National Institute of Technology
Karnataka
Surathkal
akkashyap8991@gmail.com

G S Dhanush
Information Technology
National Institute of Technology
Karnataka
Surathkal
dhanush191@gmail.com

Biju R Mohan
Information Technology
National Institute of Technology Karnataka
Surathkal biju@nitk.edu.in

***Abstract*—Scheduling of tasks or jobs is a very well studied and very basic domain of study in the field of computer science. After the evolution of cloud computing, scheduling of tasks was no longer restricted to only operating systems. With the extensive use of cloud computing environments, coupled with the amount of computation that is being carried out, there is a need for a more efficient and dynamic approach towards task scheduling. One of the ways of achieving this would be to analyze and pre-process the characteristics of a task, before trying to schedule it. By analyzing the characteristics of the tasks, one can learn about the requirement of various resources by the task and hence schedule it accordingly. In this paper, we take a look at the possibility of lexically analyzing the program files in order to learn about the various characteristics of the task and use it further to classify the tasks depending on the type of resources required.**

Keywords—Task classification, Lexical Analysis, Task Scheduling.

I. Introduction

Classification of tasks is an important step in order to design a scheduling algorithm to optimize the usage of resources and ensure Quality of Service (QoS). This classification can be used by scheduling algorithms along with virtual machine classification to provide the optimal pairing of tasks and virtual machines.

Existing task classification methods, require some of the task's characteristics to be explicitly given by the user (eg. Length of the task, Millions of Instructions per second (MIPS), T_{ij} Time taken by a task i on a Virtual Machine j). Such methods have a shortcoming, as it is difficult to find such values.

The classification method proposed involves analyzing the program submitted by the user, to categorize the task based on the priority of resources that are required by the task. We analyze the program by finding specific tokens (All the keywords in the program) that are related to a specific resource (eg, printf in C is related to I/O resource). A lexer (A program that separates and gives all the tokens from another program) is used to separate out tokens. For each resource and a specific language, there exist fixed tokens that are used to indicate that this particular resource is being used. The ratio of the number of tokens of a particular resource to the total number of tokens can be used as the weightage for that resource over other resources that are needed to be given for that program. Also, weightage for one task over the other tasks can be measured to compete over one resource by the ratio of tokens of that resource in the task to the sum total of tokens of that resource in all tasks.

II. RelatedWork

Task classification or characterization, take in multiple inputs from the user [1], such as length of the task, user priority, etc. and classify the tasks based on these inputs. Previous works [2] involve grouping tasks into different classes based on the length of the task. Characteristics like cost (Millions of Instructions Per Second), Deadline constraints [3] have also been used for grouping tasks. Another approach to grouping tasks has been based on the availability of resources required for the task [4], classifying them into dependent and independent tasks. User-defined priority [5] can also be used to classify tasks into various classes, and schedule tasks by giving higher weightage to high priority tasks. There have been a few attempts at making use of Machine Learning techniques [6] to estimate the resources required for a task, task execution time and classifying them. Although, such approaches heavily depend on the already available data for training the models. Approaches mentioned above might not be suitable for real-world applications wherein the user seldom knows the length of the task that needs to be scheduled.

III. PROPOSEDTASKCLASSIFIER**Assumptions**

1. Three types of resources (Can be generalized for any number)
   (a) Memory
   (b) Compute
   (c) Network
2. A tokenization function token(task) that returns all the tokens (function names, variable names, etc...)
3. A dictionary of tokens for each type of resource *resource_token* (A dictionary). It is dependent on the language in which the task is written.
4. Programs (not executables) are submitted to the task classifier.

Algorithm

Input: Set of Tasks {T_1, T_2, T_3, ..., T_n}

Output : Classify the tasks into categories based on the resource each classifier needs.

Priority or weight is a real number between 0 and 1. There are two priority lists.

- For each task the priority of which type of resource it requires (Within a task)
- For each resource which task requires more priority among tasks (Among tasks)

Data Structure:

Number of tokens in a task for each type of resource: *count_token*, a dictionary with key as resource type. This is calculated from the resource token dictionary. Let **N** be the total number of tokens.

(Priority Within task) For each task T_i:

```
    Let $Token_i = token(T_i)$
    Compute count token for each resource type.
    For each type of resource :
    Priority or Weight of that resource =$count\_token_{[type]}/N$
```

(Priority Among tasks)

N_{type} = sum of$count_token_{[type]}$ of all tasks For each type of resource:

```
    Priority for a type of task = $count\_token_{[type]}/N_{type}$
```

Hence two lists are obtained

- Priority list for a task $_i$for different types of resources i.e,

Priority list for $T_i = [P_{type1}, P_{type2}, ...]$

- Priority among tasks for one type of resource (say type $_i$)

Priority list for $type_i = [P_{T1}, P_{T2}, P_{T3}, ...]$

IV. IMPLEMENTATIONOFTHETASKCLASSIFIER

In this section we analyze the requirements of the task classifier and it’s implementation specifications.

The following are the necessary requirements for task classification :

- All the submitted tasks must be in the form of program files (using Python for demonstration purpose)
- The program files submitted have read permission

The program files, thus submitted are passed through a parser, which performs the function of extracting all the keywords/tokens from the given program. A module named tokenize is being used to extract the tokens from the programs. This module is supported for both python2 and python3, which can be installed through package managers such as pip and anaconda.

A python dictionary is built which connects which tokens fall under which category of task is initialized. For example, keywords such as for and while will contribute to computationally expensive components of a task.Then the given program is analyzed

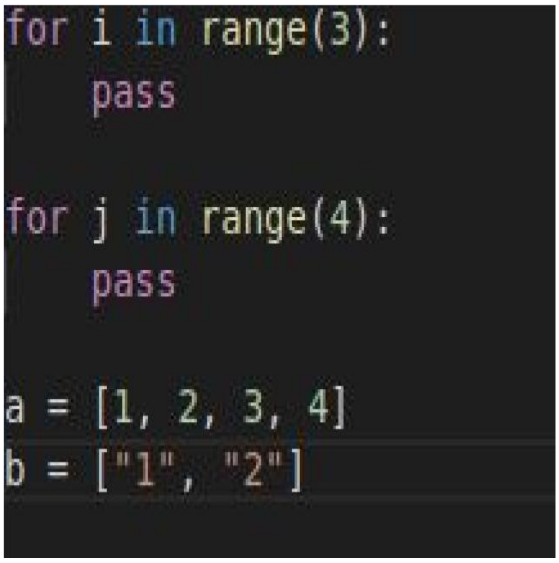
in the following way to classify it:

- In case of declaration of lists, the length of the list is calculated and used as a measure to indicate the memory utilization by the program.
- In case of for loops, a similar approach is used to determine the number of iterations and then use it as a measure of the computational intensity of the program.
- In case of presence of socket connections, the length of the buffers necessary, number of connections and other parameters are extracted and used as a measure of the network requirements of the task.

Once the tokens belonging to the various categories are identified, the score of each resource requirement of the task is calculated as proposed. These scores are then made use of to classify the tasks into computationally intensive, memory intensive or network intensive ones.

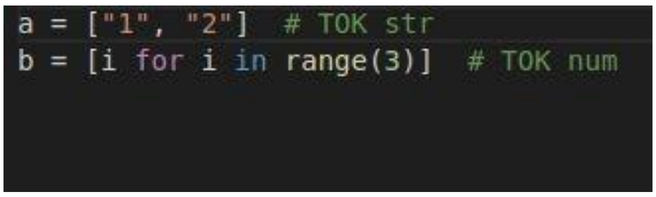
V. RESULTANDANALYSIS

The following example illustrates 3 sample programs, which have been analyzed using the python program and then classified into different categories (the sample programs have been deliberately chosen to fit into computation as shown in Fig. 1, memory as shown in Fig. 2 and network as shown in Fig. 3, categories).



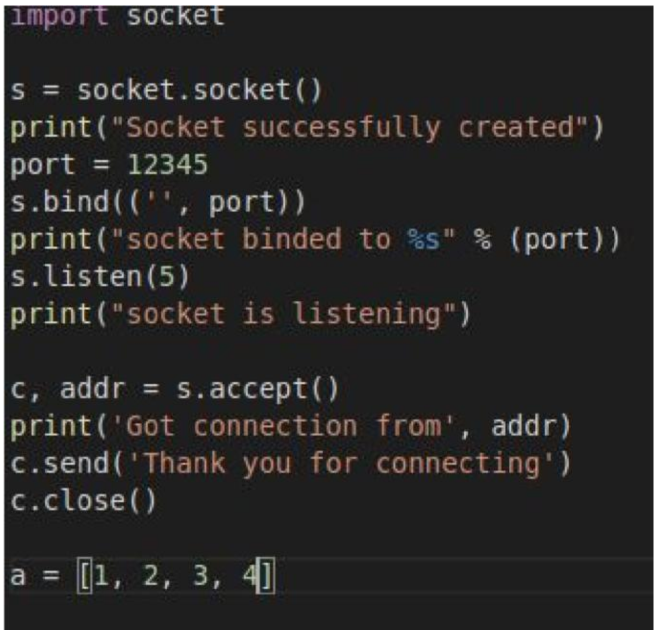
```
for i in range(3):
    pass

for j in range(4):
    pass

a = [1, 2, 3, 4]
b = ["1", "2"]
```

*Fig. 1*Computationally intensive program.



```
a = ["1", "2"]  # TOK str
b = [i for i in range(3)]  # TOK num
```

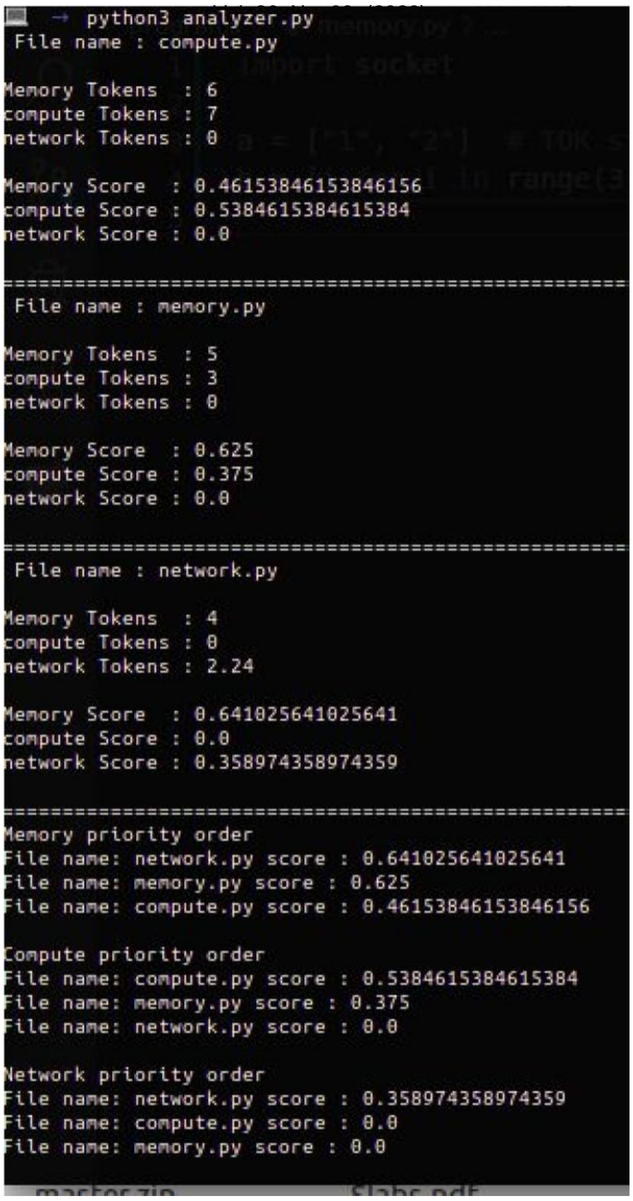
Fig. 2 Memory intensive program.



```
import socket

s = socket.socket()
print("Socket successfully created")
port = 12345
s.bind(('', port))
print("socket binded to %s" % (port))
s.listen(5)
print("socket is listening")

c, addr = s.accept()
print('Got connection from', addr)
c.send('Thank you for connecting')
c.close()

a = [1, 2, 3, 4]
```

Fig. 3 Network intensive program.

The Fig. 4 illustrates the output of the task classifier program after running it on the three sample programs.



```
→ python3 analyzer.py
 File name : compute.py

Memory Tokens  : 6
compute Tokens : 7
network Tokens : 0

Memory Score  : 0.46153846153846156
compute Score : 0.5384615384615384
network Score : 0.0

=================================================
 File name : memory.py

Memory Tokens  : 5
compute Tokens : 3
network Tokens : 0

Memory Score  : 0.625
compute Score : 0.375
network Score : 0.0

=================================================
 File name : network.py

Memory Tokens  : 4
compute Tokens : 0
network Tokens : 2.24

Memory Score  : 0.641025641025641
compute Score : 0.0
network Score : 0.358974358974359

=================================================
Memory priority order
File name: network.py score : 0.641025641025641
File name: memory.py score : 0.625
File name: compute.py score : 0.46153846153846156

Compute priority order
File name: compute.py score : 0.5384615384615384
File name: memory.py score : 0.375
File name: network.py score : 0.0

Network priority order
File name: network.py score : 0.358974358974359
File name: compute.py score : 0.0
File name: memory.py score : 0.0
```

*Fig. 4*Output of the Task Classifier.

VI. CONCLUSIONANDFUTUREWORK

Currently a token analyzer algorithm is proposed which analyses tokens of the program to classify the program. This task classification algorithm can be used in task scheduling algorithms by taking as input the two priority lists generated by this algorithm. Apart from cloud computing, it can also be used in operating systems instead of the conventional algorithms which do not consider any lexical approach. This approach may also be used in compilers to classify programs during compiler optimizations, while taking into consideration the available resources of the machine. The aim is to reduce the wait time of the scheduling process. Hence it should be of importance that the analyzer algorithm which is to be run before the scheduling process is as optimized and fast as possible without utilizing a lot of resources. The algorithm must also be as general as possible to consider all cases, as programs with a similar goal can vary a lot depending on the programmer. Hence the future work will involve optimizing the algorithm to the maximum extent thereby reducing the run time, generalizing the analyzer algorithm to deal with a variety of cases. Conventions can be introduced in the form of comments through which the programmer can inform the analyzer on important details that affect the classification of the algorithm.

REFERENCES

[1] Aladwani, Tahani. "Impact of selecting virtual machine with least load on tasks scheduling algorithms in cloud computing." In Proceedings of the 2nd international Conference on Big Data, Cloud and Applications, pp. 1-7. 2017.

[2] Chitgar, Negar, Hamid Jazayeriy, and Milad Rabiei. "Improving Cloud Computing Performance Using Task Scheduling Method Based on VMs Grouping." In 2019 27th Iranian Conference on Electrical Engineering (ICEE), pp. 2095-2099. IEEE, 2019.

[3] Choudhary, Monika, and Sateesh Kumar Peddoju. "A dynamic optimization algorithm for task scheduling in cloud environment." International Journal of Engineering Research and Applications (IJERA) 2, no. 3 (2012): 2564-2568.

[4] Parikh, Shachee, and Richa Sinha. "Double level priority based optimization algorithm for task scheduling in cloud computing." International Journal of Computer Applications 62, no. 20 (2013).

[5] Ali, Hend Gamal El Din Hassan, Imane Aly Saroit, and Amira Mohamed Kotb. "Grouped tasks scheduling algorithm based on QoS in cloud computing network." Egyptian informatics journal 18, no. 1 (2017): 11-19.

[6] Pham, Thanh Phuong, Juan J. Durillo, and Thomas Fahringer. "Predicting workflow task execution time in the cloud using a two-stage machine learning approach." IEEE Transactions on Cloud Computing (2017).