Message Passing Interface – Parallel Programming for Computer Clusters

Igor Nelson Garrido da Cruz
Faculdade de Ciências e Tecnologia
Universidad de Coimbra
Portugal
igorcruz@student.dei.uc.pt

Francisco Javier Farfán Conteras
Faculdade de Ciências e Tecnologia
Universidad de Coimbra
Portugal
contreras@student.dei.uc.pt

I. *Introduction*

The purpose of this practical work is to explore how we were able to adapt the problem "the big problem" to a configuration where MPI is used to distribute the work to solve the problem on several machines at the same time, so try to reduce the time implementation to a minimum. To do this we use several machines of the UC, who will be connected to a central machine in order to receive the global data flow, the central computer will be able to unite all the result for only having a common output. This planning will allow you previously performed a single machine with multiple threads, now several computers execute parts of the overall problem, lowering the number of processes much staying in a machine, and greatly accelerating problem resolution.

We are going to give an overview of the fundamental problem, to review the concepts.

The Grand Rule Problem was proposed and written by Professor Paulo Marques. An industrial client (a bank) has recently approached a company in our town to classify the expenses of its clients according to a number of rules.

In practice, for each bank transaction of an input file, they want to see if it matches a number of predefined rules. A bank transaction is a tuple with 10 attributes – *(a0, a1, a2, a3, …, a9)*, where each attribute is an integer (32 bit). Each rule is also a tuple with 10 attributes plus a classification "c". I.e., *(r0, r1, r2, r3, …, r9,c)*. Each rule attribute can either be a number (32 bit), or a "*", which means "don't care". For example, the transaction (1,2,3,4,5,6,7,8,9,0) matches the rule (1,2,3,4,5,6,7,8,9,0,111) and the rule (1,*,*,*,*,*,*,*,*,0,222), but it does not match either the rules (2,2,3,4,5,6,7,8,9,0,333) or (2,*,*,*,*,*,*,*,*,0,444). Our challenge is to implement a system that allows efficiently solving this problem for large amounts of data. In particular, the objective is to process 10 input file containing the transaction of the last 10 days. Each input file has 1M tuples. The rule table is fixed and has 2M rules. Most of the rules are actually empty (i.e., with"*"), having
numbers in only a few columns of each rule.

II. *Architecture*

The first computer (MPI rank 0) is the one responsible for the coordination of the outputs of the other machines and further writing into output file.

The other computers (MPI rank > 0) will be responsible for building a tree similar to the one used for practical assignment number 1 and consequently analyze the transactions returning the corespondent categories to the main machine. For example, computer2 will be responsible for analyze the lines 0, 8, 16. The computer3 will be responsible for the lines 1, 9, 17 and so on regarding the remaining machines. During the verification process the machines also send messages to first computer in order to write the output into the file system.

III. ***How to Run***

For the development of this project we used FastMPJ in order to implement MPI for the java language.

To run the program we should use:

```
fmpjrun -np 9 -hostfile hosts -Xmx4g -jar cad.jar /dataset/THE_PROBLEM/rules2M.csv /dataset/THE_PROBLEM/trans_day_0.csv output2.csv
```

Note that the hosts file should contain all the machines but the current one (8 machines in total).

IV. ***Data Analysis***

After some tests in these machines with the previous work the execution times were the following:

	Time to Build The Tree	Total Time
Run 0	59 s	101s
Run 1	58 s	99 s
Run 2	61 s	102 s

As we can see, the execution times needed to build the tree and to process the transactions file were higher than the ones achieved with the machine used to test the previous work. We are not sure why this happens, but we think it is related with the difference in the CPU, although both CPUs have the same clock frequency the first one had 6 cores and the second had only 2.

As for the implementation for practical assignment number 2 we tried different approaches. Firstly we create the trees necessary to analyze the transactions in all the machines that are going to analyze inputs. These trees are explained in more detail in the report number one. After this structure is created we proceed reading the transactions file and selecting lines which are multiple to the rank of a given machine. These lines are processed and returned to the rank 0 machine which after that writes them to file.

Creating the trees and processing the transactions with the schema explained above but **not allowing to send them to machine 0** in order to be written to file took around 90 seconds.

Regarding the transmission of data between the processing machines and the machine 0 which writes to file, we tried to implement multiple architectures. We started with a simple implementation that for each match would return the input and the corresponding category. We used a window for this transition, for example with window 10, the program would only start to send data when 10 rules are found or when the end of transactions file is reached.

	Window 1K	Window 50K	Window 500K
Run 0	175 s	195 s	154 s
Run 1	221 s	193 s	211 s
Run 2	229 s	250 s	228 s

Note: Results obtained from running the transactions_day_0.csv

From the results above the only conclusion we can achieve is that this window size does not affect the program performance in any way.

After these testes we also tried to store the transactions in the machine 0 and instead of send all of the input to machine 0, just send the index of the rule and the corresponding category, in order to reduce the amount of data transferred between machines. Although this implementation resulted in results similar to the ones presented above, so we can say it was unsuccessful, from the performance point-of-view.

We also tried a different architecture. Instead of building the tree in every machine and analyze just a part of transactions in each machine, we tried to build smaller trees for each machine dividing the rules by category.

For example, in a file with 10 categories the distribution would be the following:

Machine 1 – category [0,3]
Machine 2 – category [3,6]
Machine 3 – category [6,9]
Machine 4 – category [9,10]

This implementation, creates the trees faster, but also verifies the transactions faster, since the trees have less elements. From the point of view of memory utilization, it requires way lower amounts of memory to store the structures.

The results of this implementation were :

Run	Time (s)
0	167 s
1	182 s
2	170 s

Note: Results obtained from running the transactions_day_0.csv

We cannot say for sure that second implementation (a smaller tree for each machine) is better than the first one, because the amount of executions made were not enough for that conclusion. The amount of projects from other subjects was too big and we were let without enough time for more testing. Also the existence of only 1 cluster for all the students, made the things harder because in order to have reliable results we have to wait for a moment when anyone else is running his project, thing that is hard at the end of the deadline. However as far as we saw this implementation performs slightly better. Even if each machine has to verify all of the inputs file, building the tree is faster and each verification is also faster, because the tree is smaller.

Number of transactions (Building all the tree): around 5000 per second
Number of transactions (Different trees): around 6000 per second
Verification: around 50000 per second

Performance is mainly lost in the transmission of data.

V. ***Future Work***

More work can be done with the code we created, since our object was just to play around with the MPI communication and achieve a reasonable performance.

We just used one buffer to store the transactions and categories that belong to each input, forcing the verifications to stop when the buffer is full. We could use a different architecture that would allow the verifications to continue and in background send the values to the rank 0 machine.

VI. ***Conclusions***

Somehow the machines used to calculate the outputs in the assignment number two had worse performance than the machine used to test the assignment one, as explained above in the data analyzes.

The number of messages passed from each processing machine to a single writing machine is too big causing an unbalance between the calculation and the communication.

Although even with this details explained, we still think that our results should be better, and we think that the achievement of this results can be related with the API used the “FastMPJ”, since we don’t know how far it is has good implementation of the MPI directives, and optimization of communication between machines.

One last point I would like to comment is the possibility of the implementation of virtualization from helpdesk cause some bottleneck in the loading of files to machines concurrently. We suspect of this because when we run the work from assignment one we build the rules tree in about 55 seconds. However when we do it concurrently this time fire up to 70 seconds. So there is possibility of losing performance when the sames files are being accessed concurrently, but we are not sure about this.

It was important to implement this project in MPI in order to have a better insight about this interface and it´s power. Also was interesting to be able to work with architectures in which we have a tradeoff between computation and communication.

The results achieved with this work were not as good as we thought, a couple of things can contribute to this low performance.