Code Notes

Rishit.C — PES1201800316

Time Complexity for finding a node that Matches the Task Placement Constraints

$O(n \times \log_{size} partition\_size)$

Where:
n : The number of task placement constraints, for the task **T**.
$size$: The operand size in the CPU architecture of the GM's machine.
$partition\_size$: The number of worker nodes in the *(internal* | *external)* partitions considered.

Time Complexity for finding a node that Matches the Number Resource Requirements Constraints

$O(m \times parition\_size)$

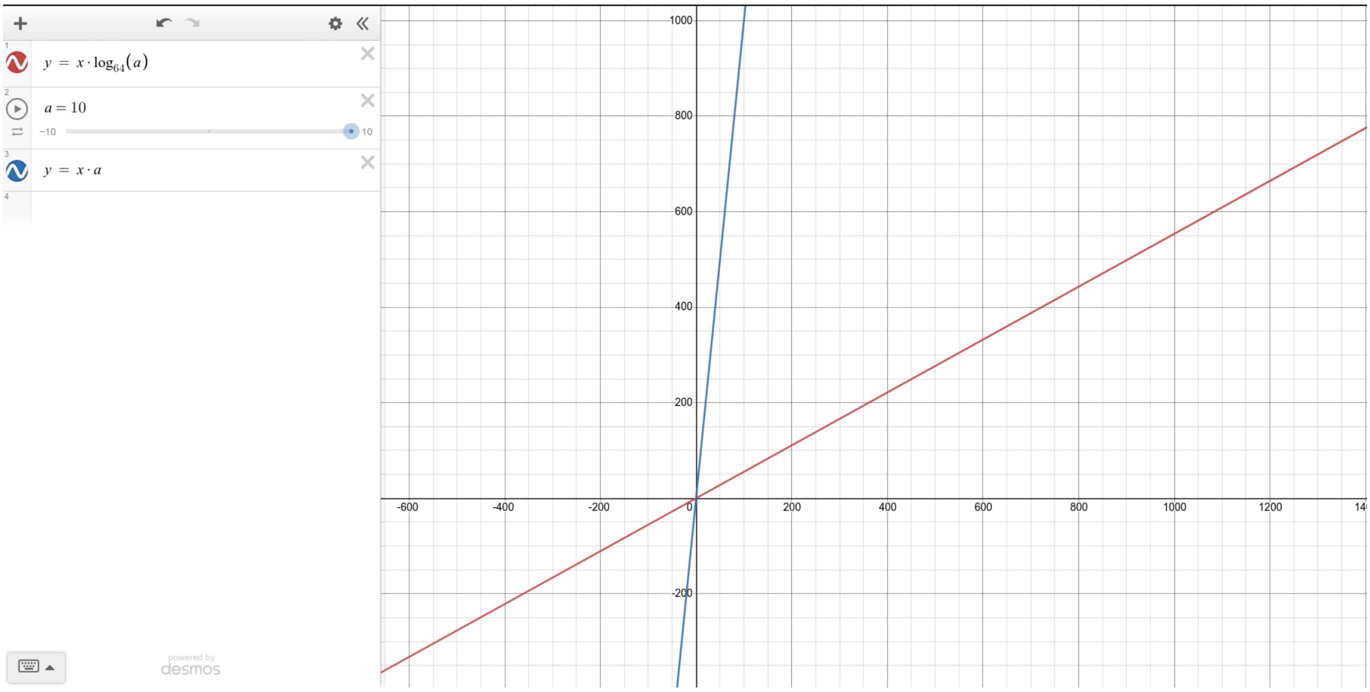
Where:
m : The number resource requirements, for the task **T**.
$partition\_size$: The number of worker nodes in the *(internal* | *external)* partitions considered.

Comparison

Considering the $size$ to be 64, as it is common for modern processors to be 64 bits, i.e. *word size = 64 bits* and the $partition\_size$ to be 10, i.e. *10 worker nodes* are checked, then we get the following graph.

NOTE:

Activity	Colour in the Graph
Time Complexity for finding a node that Matches the Task Placement Constraints	Red
Time Complexity for finding a node that Matches the Number Resource Requirements Constraints	Blue



Delays accounted for in Megha Simulator

```
NETWORK_DELAY = 0.0005  # Same as sparrow
```

Serial No.	Delay	Line Number(s)	Notes
1.	$NETWORK\_DELAY$	63, 86!, 110, 278, 281!, 291, 293!	Only one way network delay is considered.
2.	$2 \times NETWORK\_DELAY$	303!	Update from *(worker)* node to LM: $1 \times NETWORK\_DELAY$ **+** Update from LM to GM: $1 \times NETWORK\_DELAY$ **=** $2 \times NETWORK\_DELAY$
3.	$PROCESSING\_DELAY$	N/A	Not accounted for.

Events in Megha

Serial No.	Event	Description	What happens next?
1.	TaskEndEvent	This event is created when a task has completed. The `end_time` is set as the `current_time` of running the event.	The task is marked as completed in the local master.
2.	LaunchOnnodeEvent	This event is created when a task is sent to a particular worker node in a particular partition, selected by the global master and verified by the local master.	The task is simulated to run on the selected worker node and a `TaskEndEvent` is generated after the task's `duration` and the `NETWORK_DELAY` of transmitting the task from the local master to the worker node.

Serial No.	Event	Description	What happens next?
3.	InconsistencyEvent	This event is generated when the task launch requested by the global master is not possible given the current state of the worker node either: **1.** The worker node has been moved to another partition due to a `repartition operation`. **2.** The worker node in the external partition is now busy with a task assigned by its global master. **3.** The worker node in the external partition is reassigned and busy due to a repartition operation involving that worker, requested by a third global master.	The system first decides whether the inconsistency is an **internal inconsistency** *(reason: 1)* or an **external inconsistency** *(reasons: 2 and 3)*. The task is then unscheduled and the job is also removed from the `jobs_scheduled` queue and moved to the front of the `job_queue`. Finally a `LMUpdateEvent` is scheduled to **1.** update the global master(s)? about the failure to place the task on the worker node **2.** Update the global master(s)? with the updated state of the cluster.

Serial No.	Event	Description	What happens next?
4.	MatchFoundEvent	This event is generated when the global master finds a matching worker node that meets the placement constraints and the resource requirements as demanded by the task.	After this the global master requests the local master to verify the global master's task placement request. If the task placement request is consistent with the current state of the cluster then a `LaunchOnnodeEvent` event will be generated. If the task placement request is **not** consistent with the current state of the cluster then an `InconsistencyEvent` is generated.
5.	LMUpdateEvent	This event is created to update all the global masters about the current state of the cluster.	If the simulation event is a periodic one *(i.e. not due to an `InconsistencyEvent`)* and there are more events left in the `event_queue` to simulate, then it will add another `LMUpdateEvent` to continue the periodic cycle, with an interval of `LM_HEARTBEAT_INTERVAL`
6.	JobArrival	This event is created on the arrival of a job. It selects the global master to which the job will be assigned to, in a round-robin fashion, and then enqueue the job in the selected global master.	The input jobs file (trace file) is read to see if there are any more jobs to read. If so then the job is read from the file, the `jobs_scheduled` counter is incremented, a `Job` object is created and it is added to the `new_events` list and this list is returned.

Doubts

Serial No.	Question	Answer	Comments	Line(s)
1.	Why does the `InconsistencyEvent` take into account the `NETWORK_DELAY` when it is a **LM local** action? There is no network transfer yet. The network transfer takes place in the `LMUpdateEvent` which already accounts for the `NETWORK_DELAY`.		In addition, `LaunchOnnodeEvent` (the pre-step) also correctly takes into account the `NETWORK_DELAY` so why does `InconsistencyEvent` also take this into account?	86, 279, 293
2.	On line *126* in `LMUpdateEvent`: why do we add the `NETWORK_DELAY` here again? We would only need to add `LM_HEARTBEAT_INTERVAL` to the current time so that repeats after a set interval.		This is also an **LM local** operation, so would we need to add `NETWORK_DELAY` here again?	126
3.	`LMUpdateEvent` updates all global masters irrespective of whether all the global masters need to be updated as in the case of a **periodic update** or only the global master which caused a pervious `InconsistencyEvent` needs to be updated			122-123

Serial No.	Question	Answer	Comments	Line(s)
4.	On line *148* shouldn't this `JobArrival.gm_counter=(JobArrival.gm_counter) % self.simulation.NUM_GMS + 1` be `JobArrival.gm_counter = JobArrival.gm_counter % self.simulation.NUM_GMS + 1` followed by the line `JobArrival.gm_counter += 1` on possibly line *153*		Essentially the value of `JobArrival.gm_counter` is not incremented after it is used. The `gms` keys start from '1', '2' and so on.	148 - 153
5.	In the simulator, why are we selecting the global master in a round-robin fashion, when as per the Megha architecture, the global master selects a user queue in round-robin order and from the selected queue it processes a job			148
6.	Can this `job_args = (line.split('\n'))[0].split()` be converted to `job_args = line.split()`		Both mean the same, but the second version is clearer.	194
7.	What does the lines *216 - 218* do?			216-218

Serial No.	Question	Answer	Comments	Line(s)
8.	Are the names **external inconsistency** and **internal inconsistency** switched around for the types parameter when creating the `InconsistencyEvent` object			77-86 and 278-293
9.	Here isn't the `NETWORK_DELAY` of update from LM to GM already accounted for in the event object `LMUpdateEvent`.		Aren't we double counting the `NETWORK_DELAY` value. We need one `NETWORK_DELAY` addition for the message to be sent from worker to LM which is accounted for in line **303**, but the next step of send the message from LM to the GM happens by the `LMUpdateEvent` (*non-periodic event* as per line **303**) which already accounts for the `NETWORK_DELAY` of communication between the LM and GM. The *data of the cluster* **piggybacks** with the *data of the tasks successfully completed* event	303

Serial No.	Question	Answer	Comments	Line(s)
10.	Can we have a `break` in the *if-block* on line **362**		This would help early exiting the loop, once the match is found. Also it is dangerous currently to continue looping once we remove the job. Once we remove the element from the list, the list size decreases so we may give an index that is invalid during this loop causing run-time errors.	362
11.	The term **internal paritition** refers to the **internal partition** of the other Global Masters in cluster.		In the repartition operation, our global master searches the other global master's internal paritions for a worker node that meets the requirements	385, 389
12.	Here we are going for **repartition** for the entire set of remaining jobs and corresponding tasks, but in a more detailed simulator **(where not just `CPU = (1|0)`)** is considered; there we would only do **repartition** for the task we are not able to place in the internal paritions of the GM right?		It is possible that only that particular tasks needs reparition operation for it to be placed on a suitable worker node but the others are alright with nodes in the internal parition. **(this applies when the simulator is more detailed and granular about resources than it is now)**	432

Serial No.	Question	Answer	Comments	Line(s)
13.	Should this condition on line **440** be `len(self.job_queue)>= 1` as there can be more than a single job in the `job_queue` that need to be scheduled?		The `job_queue` size can be greater than 1, when a job and its corresponding task(s) cannot be placed on any worker node on the entire cluster (so neither `schedule_tasks` nor `repartition` worked) so the condition should be `len(self.job_queue)>= 1`	440
14.	Why are we doing `pickle.dumps` immediately followed by `pickle.loads`?			466, 474
15.	Why are we not adding `NETWORK_DELAY` *(of LM to GM)* here when we did so on **303**			495