A Scheduling Platform for Computing Big Data in the Cloud

Logan Schneider, Dr. Aravind Mohan
Department of Computer Science, McMurry University

Abstract

The era of big data has started. Over the past decade, the digital revolution has resulted in three major challenges that define big data. The large volume, variety of types, and velocity of data generation make an arbitrary dataset be classified as big data. On the other side, computing big data is presented with the grand challenge of transforming billions of bits and bytes into insights in a time-effective manner. One solution to this problem is to represent computation as a directed acyclic graph called workflow and use the cloud for allocating tasks in the graph to a computational resource that is more suited to the task. This solution allows the effective use of computational resources through scheduling and provides a framework for data to be processed more quickly. In this research, a popular scheduling algorithm known as Heterogenous Earliest-Finish-Time (HEFT) algorithm was implemented to solve the issue of computing big data in a usable and scalable manner using the Java programming language in an object-oriented approach. An important part of this algorithm is to rank the tasks in the workflow based on their computational and data transfer time. The HEFT algorithm prioritizes the task with the highest upward rank value at each computational step and applies a series of rules to identify the best processor for the task based on their computational and transfer time. The preliminary experiments conducted in this research are instrumental in understanding existing research and thinking of new ways to solve the important problem of computing big data effectively in the future.

Introduction

We have been propelled into an era of remarkable technological advancements, from many improvements in the field of storage to upgrades in displays, the technology we currently have is quite advanced and even seems uncanny compared to the early days of technology. Despite taking these strides towards progress, there exists a bad habit of taking these advancements for granted by continuing to cling to some outdated methods that, while proven to be functional, have become an inefficient and clumsy use of our higher-level technology. One area where this inefficiency is very relevant is in the under utilization of processing in general. The speed at which we can process tasks has significantly improved, yet an over reliance squanders this potential in processing on the outdated method of Homogenous processing. This practice not only hinders progress but also results in an unwieldy use of this improved technology. A more viable solution to this problem lies in heterogeneous processing, which is vastly different from the homogeneous approach. Heterogeneous processing involves distributing tasks across multiple processors instead of a single processor, thereby optimizing efficiency, reducing processing time, and lowering overall costs. For a broad example, imagine you hired one worker to do five different tasks each with a different completion time it would probably take quite some time, now compare that to hiring around four workers to do the same tasks, which would not only greatly reduce the amount of time it would take to do the task, but would also save cost in a variety of ways and greatly reduce the strain on the workers. A notable algorithm focused on the implementation of heterogeneous processing is the Heterogeneous Earliest-Finish-Time (HEFT) algorithm, this proposed algorithm offers a better approach to heterogeneous processing, which ultimately helps process tasks more efficiently and cost-effectively.

Methodology

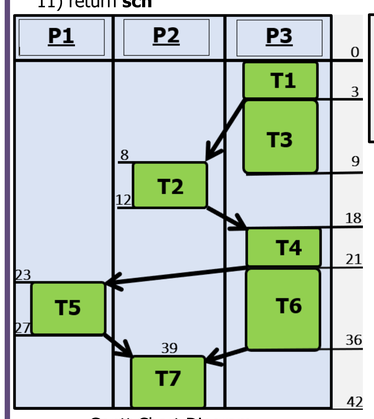
The algorithm being used needs a graph matrix that tells the program the edges of the nodes. It also needs a cost matrix which the program uses to output a schedule with the HEFT algorithm. To start, find the rank of a task. If it has no successors, the rank will be the average weight of the cost. Otherwise, the rank is the average weight of the task's computation cost plus the max value of the successor's edge weight and its rank. Looking at the workflow graph below, for example, task 7's rank cost is 6.67 because it's the last task and has no successors. To further add to this example, consider a successor to task 7, task 5. We can find its rank by adding its cost, 6.67, to the highest number by adding the weight of the successor's edge and its rank. To find the max successor value, you can get the edge weight between tasks 5 and 7, which is 6. This weight is then added to the successor's rank. in this case, the weight and rank sum to 12.67. This is then added to the weight of task 5 to get the rank value of 19.34. We chose task 7 to compute the max value because it is the only successor to task 5. If there was another successor to a task like with task 4, you would find its max successor value the same way. You'd add the edge weight to the rank of each successor. Then, you'd compare the numbers. You compare the values of Task 5, 21.33, and Task 6, 32.33, with Task 6 being the max value. Then, you'd add the weight of Task 4, 5.67. After computing the ranks, you would sort them into a list of descending values. Then, you would check which machine should process them. First, make an empty schedule from the sorted list. Then, find the machine with the cheapest cost and transfer time. Assign the task to that machine, unless it is the first task. In that case, assign the task to the machine with the cheapest cost.

Scheduling Algorithm

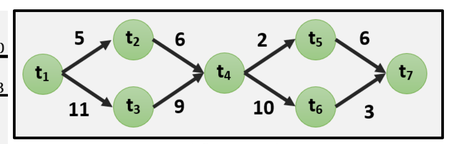
Algorithm – Scheduler

Input: Graph Matrix, Computation Cost Matrix
Output: Schedule

1) compute rank
2) sch = create an empty schedule
3) T = sort tasks in descending order based on their rank
4) for each task t in T
5) if t is an entry task then
6) sch = assign t to machine with the cheapest cost
7) else
8) sch = assign t to machine with a combination of cheapest cost and transfer time
9) end if
10) end for
11) return **sch**



Gantt Chart Diagram



Workflow Graph

	P1	P2	P3
t_1	8	5	3
t_2	7	4	9
t_3	2	11	6
t_4	5	9	3
t_5	4	7	9
t_6	10	13	15
t_7	9	3	8

Computation Table

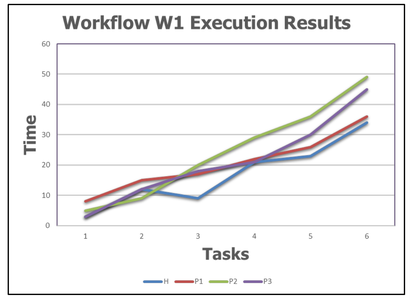
Sorted	R	ST	ET	P
t_1	69.67	0	3	3
t_2	50.67	3	9	3
t_3	53.33	8	12	2
t_4	38	18	21	3
t_5	19.34	21	36	3
t_6	22.34	23	27	1
t_7	6.67	39	42	2

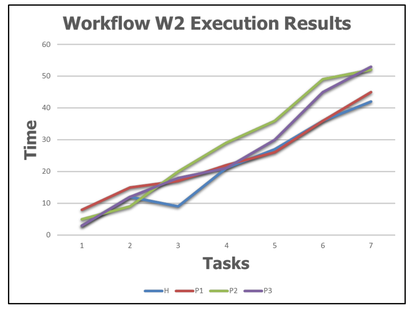
Schedule Table

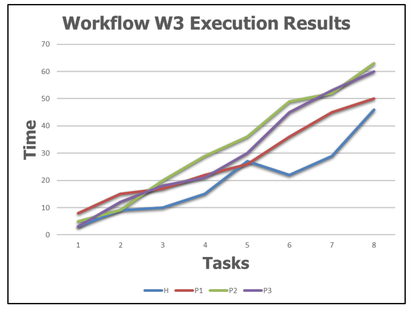
Experiments

We had three cases in which we had experimented using the HEFT algorithm vs a Homogenous algorithm. In all three of these cases, it was shown that HEFT, represented by the blue line[H] in the figures, was more efficient in time than using a homogenous method for each process.

The red[P1], green[P2], and purple[P3] lines in the figures represent the use case if we had instead used a specific machine to process all the tasks, aka the Homogenous method. We did this to prove whether or not using the HEFT algorithm would present a more efficient time than using a Homogenous method on any one of the machines. It was seen in the experiments that using HEFT was indeed a more efficient method and showed a lower time.







Conclusion

After running these experiments we have concluded that HEFT is a reliable and effective use of computational resources when compared to a Homogenous method. Using HEFT to prioritize the rank value from the highest to the lowest, helps not only identify the best processor to use but also the best processing order. It is apparent that by utilizing the HEFT algorithm we can cut down a lot of the waste in time and computing power. In some cases, it may seem to only cut a little time, but in the long and short run every little second counts. This is true when it comes to increasing speeds and handling large amounts of data efficiently. The larger the amount of data being computed means a much wider gap in accumulated time between the use of the HEFT algorithm and the use of a Homogenous method, the HEFT algorithm being the larger time saver between the two as more data is entered. Looking towards the future, using HEFT will allow the ability to compute an even larger amount of data and a more complex set of data while being much smoother, cheaper, and quicker than the current methods.