Course Project

CS528: High Performance Computing

Project Report: Problem One

Task:

To place and schedule data-intensive jobs in edge-cloud system with real coefficients

Authors:

Group 13

Adittya Gupta, 210101007
Gautam Sharma, 210101042
Sparsh Mittal, 210101100

Instructor:

Prof. Aryabhartta Sahu, Dept. of CSE, IITG

Preface

This project was assigned to **Group 13** as the final project in the course CS528: High Performance Computing, Spring 2024 Semester, at IIT Guwahati.

Group 13 consists of the following students (listed roll number-wise):

1. Adittya Gupta, 210101007
2. Gautam Sharma, 210101042
3. Sparsh Mittal, 210101100

The project can be found on Github (link to the project).

Contents

1 Introduction

Today's trend of using edge-cloud computing is increasing daily due to the stagnation in advancing technology of optimizing personal computers. As more and more people start using cloud servers, the number of jobs occurring on a cloud cluster increases exponentially, the placement and scheduling of jobs becomes increasingly important to meet all the deadlines while using the least resources.

This report proposes a **novel heuristic** for placing and scheduling data-intensive jobs in an edge-cloud system with real coefficients.

2 Problem Statement

Consider a set of J jobs (arrived at time 0) that need to be processed by a cloud consisting of N physical machines (i.e., nodes) that are homogeneous. Each job j has a deadline d_j and is required to access a set C_j of equal-sized chunks, we can assume each job j has $|C_j|$ number of tasks accessing one data chunk each and can be run in parallel on the same machine or different machine.
The chunks are stored in a distributed file system on the cloud. Each node is capable of hosting up to B data chunks and is equipped with S Virtual Machines (VMs) which implies each node is able to simultaneously process S jobs. Let $C = \cup C_j$ be the set of all data chunks available at the central storage server, before processing the request we need to bring the data chunk to the physical machine.
Replication of data chunks in different machines is allowed and data chunk needs to be placed only once at the beginning (before any job starts execution, once any one job starts the execution, we are not allowed to change the data placement) and during run time we can not place/replace/replicate the data chunk.
The time for each job j to process a required data chunk is unit time and it is the same for all the jobs. Completing a job j before the deadline d_j is equivalent to processing all the required chunks c of C_j before the deadline. Only one VM can access a data chunk in a given time slot of the same physical machine.
The problem aims to **minimize the total number of active nodes (N_a) to process all the jobs**. The active node means the node stores at least one data chunk and is processed by at least one job.

3 Background literature review

There has been intensive research on improving the performance of cloud-based frameworks. Data locality significantly impacts system performance and is considered an important factor for scheduling [1].
Achieving efficient data locality in a cloud cluster is critical for performance and reduces data transfer costs over the network. [2] designed and implemented a resource-aware scheduler to alleviate job starvation and avoid unfavourable data locality. ActCap [3], and MRA++ [4] proposed solutions for data placement to improve MapReduce performance on a heterogeneous cluster.
Deadline-aware schedulers for cloud-based frameworks have also been well-studied. For example, [5] developed a deadline constraint scheduler and derived closed-form expressions for the minimum Map/Reduce tasks required to meet deadlines in the MapReduce framework. SAMES [6] proposed a scheduling algorithm for MapReduce jobs with deadlines. Its goal is to maximize the number of jobs that finish before their deadlines. [7] aims to improve re-resource utilization

while observing deadlines.
The algorithms in this report are inspired from [8].

4 Problem Statement

4.1 Problem Formulation

The following is the input-output structure of the problem:

1. **Input**:
   (a) N: Total number of machines available
   (b) C: List of all the data chunks
   (c) J: Total number of jobs
   (d) d_j: Deadline of each job
   (e) C_j: Data-chunk required by each job
   (f) S: Total number of VMs a node(machine) can spawn
   (g) B: Total number of data chunks a node can hold
2. **Output**:
   (a) N_a: Least number of active nodes to process every job before their deadline

4.2 Constraints

Any feasible solution to the problem must satisfy the below constrains:

1. Any node can hold **atmost** B data chunks
2. Any node can spawn **atmost** S virtual machines
3. Given a data chunk that is being processed at a node, it can only be accessed by **atmost** one VM of the node
4. Every job has to be solved **on or before** it's deadline
5. Number of active nodes has to be **less than** the total number of available nodes

4.3 Facts about the problem

The following are the facts, which are maintained by the input:

1. All machine nodes are homogenous
2. Deadline of every job is known *a priori*

5 Solution

To solve the above problem, we propose a **novel heuristic**.

5.1 Solving a reduced problem: Same deadlines

We first solve an easier version of the problem, a problem where there are multiple jobs but with the same deadline, so that we can extend the solution of this problem later to solve the main problem.

Consider the pair f_i, id where id is the ID of the data chunk, and f_i is the number of times the data chunk has to be processed to complete all the jobs that require it.

For any node, we can define a **virtual deadline** (v_d) of $S * t_d$, where t_d is the true deadline of the job, which is actually the computational power of the node in reality.

For any deadline, sort the pairs f_i, id in decreasing order, and choose the first set of B data chunks from the tail such that their sum of f_i is greater than or equal to v_d. Schedule this set of B data chunks into a node. Repeat this process over r iterations until we are left with less than or equal to B chunks, whose sum of f_i is less than or equal to v_d. Schedule this last set of nodes in another node.

While scheduling on nodes, one needs to take care of 4.2 (3), that one data chunk is accessed by only one VM of the node, hence the computing power a node can give to a data chunk is at most t_d. Hence the computing power of a node is not being completely utilized. So, while scheduling chunks, one has to iterate through all the existing nodes to see whether the chunk can be scheduled in an existing node.

Our proposed algorithm is shown in Algorithm 1.

We describe the algorithm **SCHEDULE** below:

The basic idea behind SCHEDULE is to greedily place and schedule the least computationally expensive chunk to a node that doesn't already access the chunk through a VM (4.2 (3)).

Algorithm 1 SCHEDULE

Require: List of chunk pair f_i, id (denoted as C), NTS(i), node ID i, true deadline t_d
Ensure: List of chunk pair f_i, id (denoted as C) which couldn't be scheduled on the given node

```
sort chunks based on the number of frequencies in ascending order
for chunk c_i from the start do
    check if c_i can be given to node i or not
    if c_i has been already given to i till t_d then
        Scheduling of c_i can't take place again
    else
        schedule c_i on i, either use i fully or schedule c_i fully
    end if
end for
```

5.2 Solving the main problem

Solving the main problem becomes easy after solving the easy problem. For any deadline d_i, first call 2, and then schedule the already scheduled chunk on the nodes again if they are needed for any other job in the future. The algorithm is summarized in 3.

5.3 Why our algorithm is better

Our algorithm is better than most of the common algorithms such as **First-Fit** or **Earliest Deadline First** in the respect that each node is utilized to maximum capacity and jobs are scheduled in an efficient way. The jobs are primarily scheduled using early deadlines, but if

Algorithm 2 CRED-S

Require: List of chunk pair f_i, id (denoted as C), virtual deadline (v_d), and true deadline (t_d)
Ensure: N_a

$C^{(r)} = C$
while $C^{(r)}.size() > 0$ **do**
    sort chunks based on the number of frequencies in descending order
    if there exists a set of B data chunks from the tail such that their sum of f_i is greater than or equal to v_d **then**
        denote such set by H_{B,t_d}
        for $i = 1$ to N_a **do** ▷ NTS is a function which returns the total number of free slots on a node i, before t_d
            SCHEDULE(H_{B,t_d}, NTS(i), i, t_d)
        end for
        if H_{B,t_d}.size() > 0 **then**
            Make a new node
            Increment N_a, call the new node N_a
            Set NTS(N_a) $= S * t_d$
            SCHEDULE(H_{B,t_d}, NTS(N_a), N_a, t_d)
        end if
    else
        denote such set by L_{B,t_d}
        for $i = 1$ to N_a **do**
            SCHEDULE(L_{B,t_d}, NTS(i), i, t_d)
        end for
        if H_{B,t_d}.size() > 0 **then**
            Make a new node
            Increment N_a, call the new node N_a
            Set NTS(N_a) $= S * t_d$
            SCHEDULE(L_{B,t_d}, NTS(N_a), N_a, t_d)
        end if
    end if
end while

Algorithm 3 CRED-M

Require: List of jobs
Ensure: N_a
for $i = 1$ to D **do**
  Construct the chunks set C based on the jobs that need to be completed before d_i
  temp = SCHEDULE(C, $S * d_i$, d_i)
  prev = d_i
  for $j = 1$ to $j =$ temp **do**
    for $k = i + 1$ to $k = D$ **do**
      Find the chunks that are required in this future deadline and are also scheduled on this node j
      Denote this set by C_k
      SCHEDULE(C_k, $S * (d_k - prev)$, $d_k - prev$)
      prev = d_k
    end for
  end for
end for
Output N_a = machine_set.size()

there is any chance to process a future job in a node, the future job is processed, giving an edge in minimizing the nodes and satisfying the deadlines.

5.4 Complexity analysis

5.4.1 `schedule`

- **Time Complexity:** $O(N \cdot M)$
- **Explanation:**
  - The function iterates through the given chunks and nodes, which takes $O(N \cdot M)$ time in the worst case.
  - Sorting the chunks takes $O(N \log(N))$ time.
- **Space Complexity:** $O(N + M)$
- **Explanation:**
  - Additional space is used for maps and sets to track chunk-node mappings and node loads, leading to $O(N + M)$ space complexity.

5.4.2 `cred_s`

- **Time Complexity:** $O(N \cdot M)$
- **Explanation:**
  - The function involves sorting chunks, which takes $O(N \log(N))$ time. Then, it iterates through chunks and machines, resulting in $O(N \cdot M)$ time complexity.
- **Space Complexity:** $O(1)$
- **Explanation:**

– The function mainly uses additional variables for calculations without significantly increasing space usage, leading to constant space complexity.

5.4.3 `cred_m`

- **Time Complexity:** $O(D \cdot N \cdot M)$
- **Explanation:**
  - The function calls `cred_s` for each distinct deadline, resulting in $O(D \cdot N \cdot M)$ time complexity, where D is the number of distinct deadlines.
- **Space Complexity:** $O(N + M)$
- **Explanation:**
  - Similar to the time complexity, the space complexity depends on the number of chunks and machines involved in scheduling, along with additional data structures used.

6 Results

We ran 3 on **ten cases**, result of which are summarized below.

Table 1: Table caption

# of jobs	# of data chunks	Average job size	Average job deadline	N_a
68	15	10.08	4.86	248
89	7	10.12	5.94	205
42	18	10.66	5.04	181
59	15	10.03	5.32	98
39	3	10.84	4.94	56
84	13	11.39	5.89	46
10	11	11.7	4.5	31
77	17	11.22	5.18	27
12	16	8.5	5.3	24
10	3	11.4	6.4	3

It can been seen that N_a increases almost linearly if number of jobs are increased, and exponentially if the number of chunks are increased.
Below are the graphs related to the above experiments.

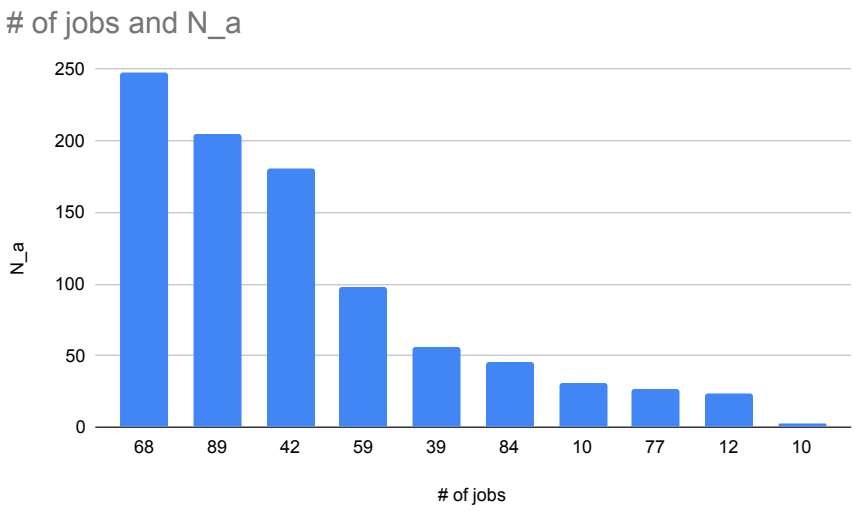

Figure 1: of jobs vs N_a

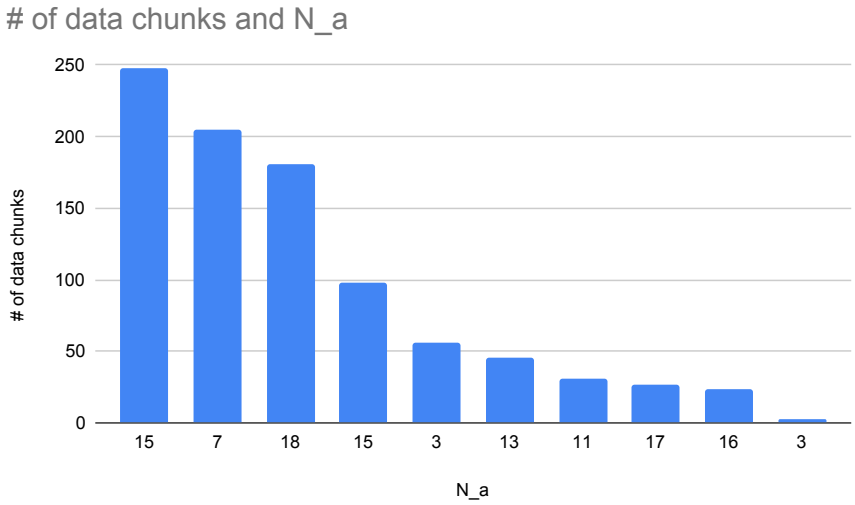

Figure 2: of data chunks vs N_a

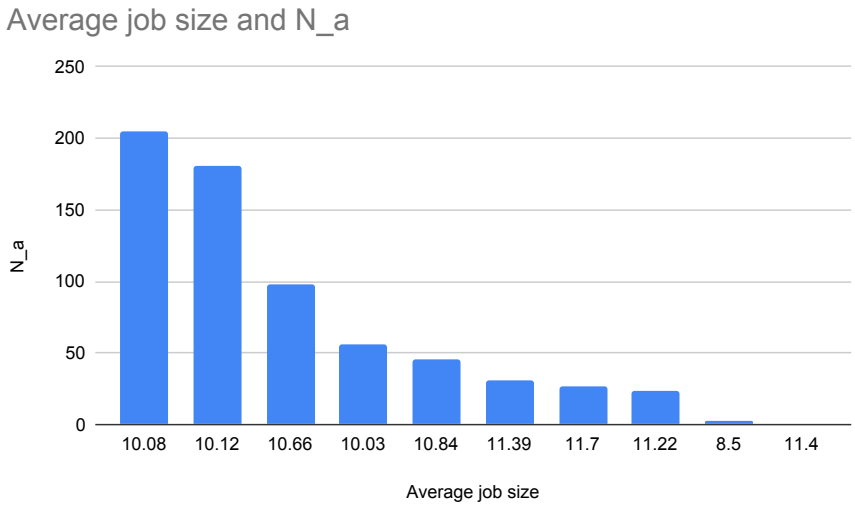

Figure 3: Average job size vs N_a

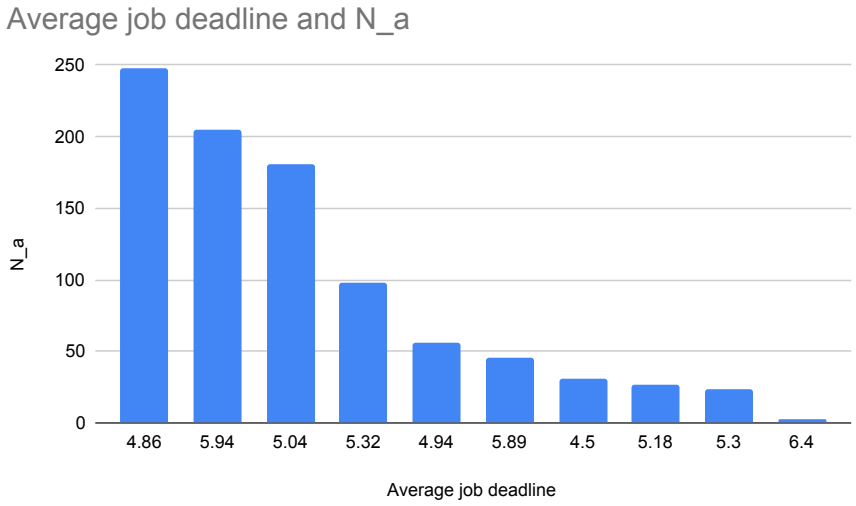

Figure 4: Average job deadline and N_a

References

[1] Zhenhua Guo, Geoffrey Fox, and Mo Zhou. "Investigation of Data Locality in MapReduce". In: *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*. CCGRID '12. USA: IEEE Computer Society, 2012, pp. 419–426. ISBN: 9780769546919. DOI: 10.1109/CCGrid.2012.42. URL: https://doi.org/10.1109/CCGrid.2012.42.

[2] Jian Tan, Xiaoqiao Meng, and Li Zhang. "Coupling task progress for MapReduce resource-aware scheduling". In: *2013 Proceedings IEEE INFOCOM*. 2013, pp. 1618–1626. DOI: 10.1109/INFCOM.2013.6566958.

[3] Bo Wang, Jinlei Jiang, and Guangwen Yang. "ActCap: Accelerating MapReduce on heterogeneous clusters with capability-aware data placement". In: *2015 IEEE Conference on Computer Communications (INFOCOM)*. 2015, pp. 1328–1336. DOI: 10.1109/INFOCOM.2015.7218509.

[4] Julio C.S. Anjos et al. "MRA++: Scheduling and data placement on MapReduce for heterogeneous environments". In: *Future Generation Computer Systems* 42 (2015), pp. 22–35. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2014.09.001. URL: https://www.sciencedirect.com/science/article/pii/S0167739X14001642.

[5] Kamal Kc and Kemafor Anyanwu. "Scheduling Hadoop Jobs to Meet Deadlines". In: *2010 IEEE Second International Conference on Cloud Computing Technology and Science*. 2010, pp. 388–392. DOI: 10.1109/CloudCom.2010.97.

[6] Xite Wang et al. "SAMES: deadline-constraint scheduling in MapReduce". In: *Front. Comput. Sci.* 9.1 (Feb. 2015), pp. 128–141. ISSN: 2095-2228. DOI: 10.1007/s11704-014-4138-y. URL: https://doi.org/10.1007/s11704-014-4138-y.

[7] Jordà Polo et al. "Adaptive MapReduce Scheduling in Shared Environments". In: (2014), pp. 61–70. DOI: 10.1109/CCGrid.2014.65.

[8] Maotong Xu et al. "CRED: Cloud Right-Sizing with Execution Deadlines and Data Locality". In: *IEEE Transactions on Parallel and Distributed Systems* 28.12 (2017), pp. 3389–3400. DOI: 10.1109/TPDS.2017.2726071.