CRED: Cloud Right-sizing to Meet Execution Deadlines and Data Locality

Sultan Alamro, Maotong Xu, Tian Lan, and Suresh Subramaniam
The George Washington University
Department of Electrical and Computer Engineering

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

School of Engineering & Applied Science
THE GEORGE WASHINGTON UNIVERSITY

Introduction

- Demand for cloud-based processing frameworks continues to grow
- Cloud providers seek efficient techniques that deliver value to the business without violating the Service Level Agreement
- Cloud right-sizing makes clouds more cost-effective

Motivation

The nature of cloud applications is becoming increasingly mission-critical and deadline sensitive, e.g., traffic simulation and real time web indexing. These applications are evolving in the direction of demanding hard completion times, and are likely to play crucial roles in the national infrastructure in the not too distant future.

Objective

Our goal is to minimize the total number of active nodes needed to complete the jobs satisfying a deadline constraint dj for each job j, the data locality constraint and physical resource constraints on each node, i.e., B and S. We consider a time-slotted model where jobs are scheduled to execute in fixed-length time slots. Since each node is equipped with S VMs, it has S time slots available at each time t . Our control knobs in the optimization include data chunk placement, job scheduling, and cloud sizing.

Challenge

The CRED problem is challenging because it is an integer optimization that is known to be NP-hard in general. Even existing approximate algorithm fall short simply due to the overwhelming size of the underlying decision space, which involves CNJ non-binary variables and N(C+1) binary variables. Here C is the total number of chunks, J the total number of jobs, and N the total number of nodes.

Example

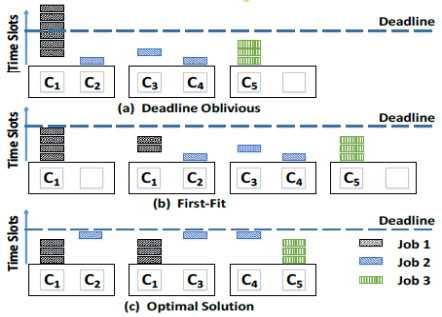


Figure 1

Problem Formulation

Consider a set of J jobs that need to be processed by a cloud consisting of N physical machines (i.e., nodes) in a timely fashion. Each job j has a deadline dj and requires to access a data object that is split into a set Cj of equal-sized chunks. The chunks are spread over the cloud in a distributed file system. Each node is capable of hosting up to B data chunks and is equipped with S VMs. We consider a cloud framework similar to MapReduce, where jobs are partitioned into small tasks that are processed in parallel by different VMs. Thus, each node is able to simultaneously process S tasks. In this paper, we consider heterogeneous jobs with different processing times. In particular, the time for each job j to process a required data chunk, denoted by Tj, can vary from job to job. A job is completed once all required chunks are processed and will then exit the system.

For cloud right-sizing, we may not necessarily use all available nodes to process these jobs. Our goal is to minimize the total number of active nodes needed to complete the jobs satisfying (1) a deadline constraint dj for each job j, (2) a data locality constraint that requires each job to be only assigned to nodes hosting its required data chunks, and (3) physical resource constraints on each node, i.e., B and S.

Solution to CRED Problem

The key idea from our illustrative example in Fig. 1 is that solving the CRED problem requires a joint optimization of job scheduling and chunk placement that addresses both execution deadline and data locality constraints in a collaborative fashion. In this section, we propose a novel algorithm that harnesses workload aware chunk placement to partition data chunks based on their workload and schedules jobs to efficiently utilize both space and computing resources on active nodes, thus minimizing the number of nodes required to process all jobs. To illustrate our key solution concept.

Single Deadline:
Consider the chunk set Ci for deadline di. We sort all chunks in descending order based on the number of required time slots for each chunk, and record the order in an array. The chunk recorded in the head of array has the largest number of required time slots. We place the last B chunks and subtract S*di time slots, since we can schedule at most S*di time slots in each node. If the remaining number of required time slots for chunk c is 0, we can remove the chunks c from the chunk set Ci. We repeat this step until the first B chunks has <= S*di time slots, then we can place any B chunks into one node. It's easy to see that we keep adding new nodes until all chunks get their required time slots scheduled. Processing chunk c is only permitted on a node where chunk c is placed. This is to improve data access efficiency and task throughput. Thus, the algorithm is guaranteed to generate a feasible solution to the CRED problem.

Multiple Deadlines:
Our idea to solve CRED with multiple, arbitrary deadlines is to iteratively apply single deadline algorithm to incrementally find chunk placement and time-slot scheduling to meet each deadline one-by-one. More precisely, after finding a solution to meet deadlines d1, d2, …, di, we fix the chunk placement on existing nodes and optimize for the next di+1 and minimize the number of new nodes.

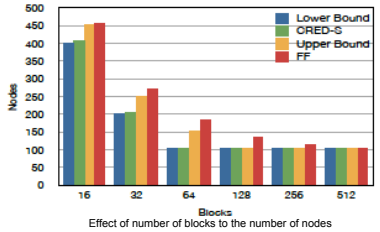
Theorems

Theorem 1. *When* $\hat{K}_1 > \lfloor \frac{2C_1}{B} \rfloor$, *the bounds are given by* $K_1 \le \hat{N} \le K_1+1$. *When* $\lfloor \frac{2C_1}{B} \rfloor \ge K_1 \ge \lfloor \frac{C_1}{B-1} \rfloor$, *the bounds are given by* $\max(\lceil \frac{C_1}{B} \rceil, K_1) \le \hat{N} \le \frac{K_1}{2} + \frac{C_1}{B} + 1$. *When* $K_1 < \lfloor \frac{C_1}{B-1} \rfloor$, *the bounds are given by* $\lceil \frac{C_1}{B} \rceil \le \hat{N} \le \frac{K_1}{B} + \frac{C_1}{B} + 1$.

Theorem 2. *The number of nodes needed is* $\hat{N}$.

$$\max\left(\max_i(K_i), \frac{\sum_{i=1}^{D}\sum_{j:d_j=d_i^\uparrow} T_j}{S d_D^\uparrow}, \frac{C}{B}\right) \le \hat{N} \le \sum_{i=1}^{D} \max\left(K_i, \frac{K_i}{2} + \frac{C_i}{B}\right) + D.$$

Evaluation



Effect of number of blocks to the number of nodes

Effect of the ratio between d1 and d2 type of jobs to the number of nodes needed

Conclusion

We introduce an optimization framework, namely CRED, for cloud right-sizing under deadline and locality constraints. Algorithms are proposed to solve CRED optimization, which minimizes the number of nodes needed by jointly optimizing task scheduling and data placement while the jobs' deadlines and data locality are met. The algorithms significantly outperform a first-fit heuristic in terms of cloud-size