

TASK ALLOCATION IN HIGH PERFORMANCE PROCESSING OF GEOSPATIAL DATA

V. Syrris, D. Rodriguez, and P. Soille

European Commission, Joint Research Centre (JRC)

Institute for the Protection and Security of the Citizens, Global Security and Crisis Management Unit

ABSTRACT

In a sandbox framework for scientific computing, we deal with the task allocation problem when processing a high volume of geospatial data. A predefined meta-information catalogue guides the data selection and a profiling procedure based on data sub-sampling estimates the memory and processing requirements. The task allocation is formulated as an optimization problem conditioned by time dependencies and job execution priorities. The procedure can work as add-on to any task scheduler that provides configuration options for computational resources allocation. Experiments demonstrate that the SLURM fine-tuning with the proposed method leads to better resource management and shorter schedules.

Index Terms— Scheduling, Resource allocation, Sandbox, High Performance Computing, SLURM, Geospatial data

1. INTRODUCTION

The context of this work can be determined by a sandbox environment where a number of users are developing code prototypes and testing their processing chains over Earth Observation imagery and in general on geospatial data. The sandbox is supported by a cluster manager equipped with a job allocation and scheduling controller. Our starting point for this study is that the average user is not knowledgeable on how to optimally configure the jobs distribution, especially when the input data are different in size and the computational resources are heterogeneous in terms of processing units (CPU/GPU, number of cores, processing power, etc.) and memory cells. Intuitively, the user tends to overestimate the processing requirements a fact that leads to high latency and ultimately to suboptimal exploitation of the cluster platform. We note here that a task is often a part of a job or a job itself; in this work these two terms are used interchangeably.

As cluster manager, we consider SLURM [3, 8] that is an open source Resource and Job Management System designed for clusters of all sizes. SLURM uses a general purpose plugin mechanism to select various features such as scheduling policies, process tracking or node allocation mechanisms.

The paper is organized as follows: Section 2 describes the automation of the task allocation set up by means of: i) a catalogue that comprises meta-information of the input data, ii) an estimation of the computational requirements based on

task execution profiling and iii) the confrontation of task allocation as an optimization problem. Section 3 reports on the experimental results. Conclusions are given in Section 4.

2. METHOD

First, indicators from a geospatial meta-information catalogue are collected about the data volume to be processed (especially the minimal bounding box containing all the data values), overlapped geographical areas, and coordinate reference system and projection. This information sets the constraints that bound the amount of data to be processed and avoid repetitions [7].

Then, through profiling [9] on a carefully chosen subset of images, the computational requirements like processing speed and memory cells per task, network bandwidth [4], efficient number of concurrent jobs and others are estimated. The selection of the images is automatic: based on criteria defined with the aid of the meta-information catalogue as described previously, a systematic sampling takes place with the purpose of representing adequately the entire data universe. By surface fitting we extrapolate the computational requirements to the whole dataset.

The problem of task allocation on the different nodes of an heterogeneous computer cluster takes the form of the following constraint problem: Let I be the number of tasks, N the number of available nodes, R_n the number of cores in node n , M_n the number of memory cells in node n , m_i the lower memory threshold that is necessary for the task i to be executed, s_i the variable designating the order of sequential tasks and p_i the task priority of independent tasks. The execution time T_i of the task i is a function of two variables: r_{in} number of cores in node n and b_{in} number of memory cells in node n that are both necessary for the execution of i task: $T_i(r_{in}, b_{in})$. That is, the optimal task allocation can be expressed as follows:

$$\text{minimize} \sum_{i \in I} \sum_{n \in N} x_{in} (1 - p_i) s_i T_i(r_{in}, b_{in}) \quad (1)$$

subject to:

$$x_{in} \in \{0, 1\}, \forall i \in I, n \in N \quad (2)$$

$$\sum_{n \in N} x_{in} = 1, \forall i \in I \quad (3)$$

$$\sum_{i \in I} \sum_{n \in N} x_{in} \leq \sum_{n \in N} R_n \quad (4)$$

$$0 \leq p_i < 1 \quad (5)$$

$$s_i - s_j > 0, i \neq j, i, j \in I \quad (6)$$

$$1 \leq r_{in} \leq R_n, n \in N, i \in I \quad (7)$$

$$m_i \leq b_{in} \leq M_n, n \in N, i \in I \quad (8)$$

The decision variable x_{in} represents the assignment of task i to node n . Condition (2) bounds the x_{in} values to 0 and 1. Condition (3) states that a task i can be assigned to only one node n , taking value 1 if the assignment is done and 0 otherwise. Condition (4) signifies that the number of allocated tasks cannot surpass the total number of cores over all the nodes. Since there are limited resources (expressed by constraints (4), (7) and (8)), in order for all the tasks to be processed, the optimization problem needs to be solved several times considering always the currently available resources.

Listing Alg.1 contains the algorithm's description. Steps 1–4 are executed once while steps 6–9 are running in several cycles and manipulating each task incrementally.

Algorithm 1: Task allocation algorithm

Input : Georeferenced images

Output: Sequence of tasks

- 1 Collect information from the catalogue;
- 2 $S \leftarrow$ sample, execute and profile subset of tasks;
- 3 Apply surface fitting and extrapolate parameter values to the entire input set;
- 4 $p_i, s_i \leftarrow$ Define values $\forall i$

5 **while** $i \leq I$ **do**

- 6 Collect available resources from each node;
- 7 Set bounds m_i, M_n and R_n , take the optimal decision and select one task i ;
- 8 Allocate resources r_{in} and b_{in} and send task i to node n ;
- 9 $I \leftarrow I \setminus \{i\}$;

10 **end**

The problem belongs to the general family of optimization problems of finding the shortest path $\{y_t\}_{t=0}^{<\infty}$ in finite time instants t . The goal is to minimize the *makespan* (1), that is the total length of the schedule (total execution time). The application of standard methods like mixed-integer linear programming is not straightforward; the component T_i of the objective function does not have a closed-form expression in terms of r_{in} and b_{in} , while the heterogeneous configuration of the cluster nodes increases the amount of constraints and the number of parameters that need fine-tuning. In the next section we show two greedy strategies that confront the specific optimization as a dynamic (online) scheduling problem.

3. EXPERIMENTS

The test case refers to the execution of an experimental cloud detector (an adjusted version of ACCA algorithm [6]) over a set of 1,000 Sentinel-2A (S2A) images. For the experiments, we used a partition of our in-house computer cluster infrastructure. The heterogeneity of the computer cluster is reflected by Table 1 which comprises the technical characteristics of the five cluster nodes.

Table 1: Specifications of the used computer cluster nodes

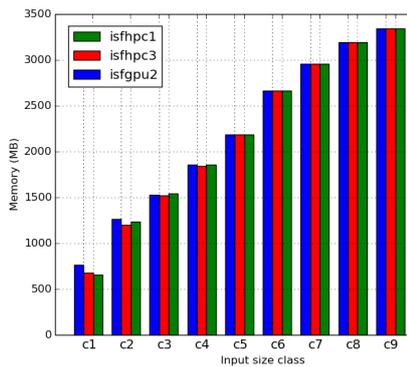
Name	Proc. units	Clock freq	Memory	Swap
isfhpc1	16(4;4;1)	2.1GHz	64GB	64GB
isfhpc2-3	8(2;4;1)	2.1GHz	32GB	32GB
isfgpu1-2	24(2;6;2)	2.8GHz	32GB	32GB

Notation (.,.,.) means: (socket;cores per socket;threads per core)

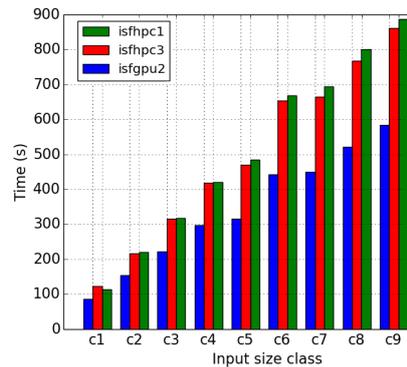
Figures 1a and 1b illustrate the variety and the velocity of the data under processing. They show the resource requirements, in terms of memory and execution speed, given by the profiling process.

First, nine groups were formed based on the size of the bounding box of the valid data pixels; value 0 has been assumed as *nodata* value. Figure 2 displays the 9-bin data pixels histogram of the available S2A B01 bands (until 31/1/2016). The explanation for the high size of group **e9** is that for these specific images, the entire image extent has been considered even if *nodata* may exist at some parts of the image. For instance, an image with zero *nodata* values and a lower triangular image belong both to the **e9** group. Then, the right-most image was selected from each group as the worst-case scenario, i.e. the largest image in terms of number of pixels. Each of these images was sent to the cluster manager with 5 different resource allocation settings and the respective performance was monitoring and recorded. Numbers for the isfhpc2 and isfgpu1 nodes are not provided since they have identical configuration with isfhpc3 and isfgpu2 respectively. The cloud detector that takes as input four 10m resolution S2A bands and two of 20m resolution, exploits the fact of knowing the data domain (as this information is provided by the catalogue) and reads only the bounding box of the data area of the images. Figure 1c shows the memory profile of the cloud detector on isfgpu2, when reading the data domain of a tile that has been classified to the largest size group **e9**.

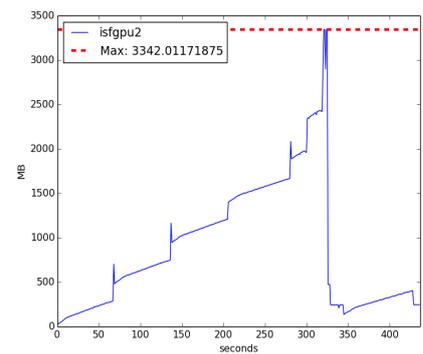
Task profiling helps to define statistically the function $T_i(r_{in}, b_{in})$ and provides output based on which the lower and upper thresholds of the optimization constraints are set up. Two greedy strategies were tested in order to incrementally schedule the tasks: i) Sort the list of tasks based on the priority and the estimated memory allocation in descending order and then, schedule the next task of the list at the first available node, ii) Apply the same ranking on the task list and then, if one of the faster nodes (isfgpu1-2) is available then select a task associated with an image from the medium to the



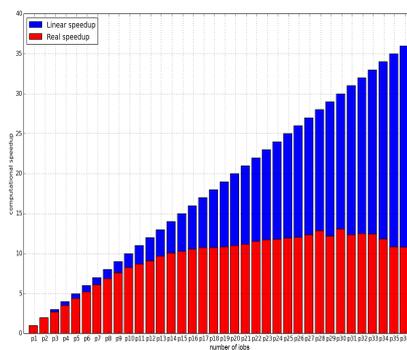
(a) Memory needed by the node to execute tasks associated with different data pixels groups.



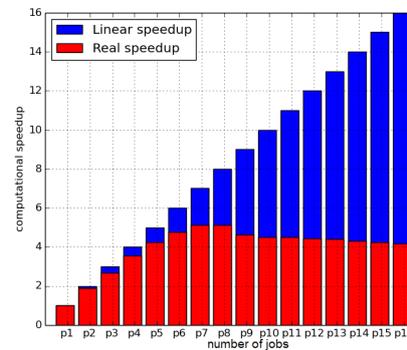
(b) Node time performance by data pixels group.



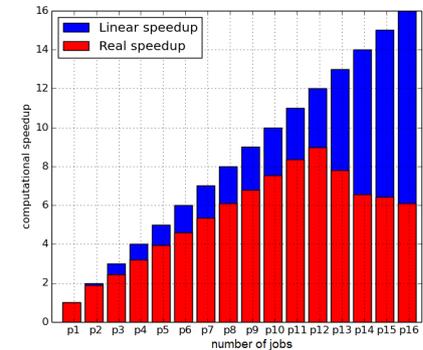
(c) ACCA memory usage in isfgpu2 when gradually reads four 10m and two 20m bands of a c9 tile having full data domain.



(d) isfhpc1 speedup tested on group c9.



(e) isfhpc3 speedup tested on group c9.



(f) isfgpu2 speedup tested on group c9.

Fig. 1: Space-Time complexity provided through profiling.

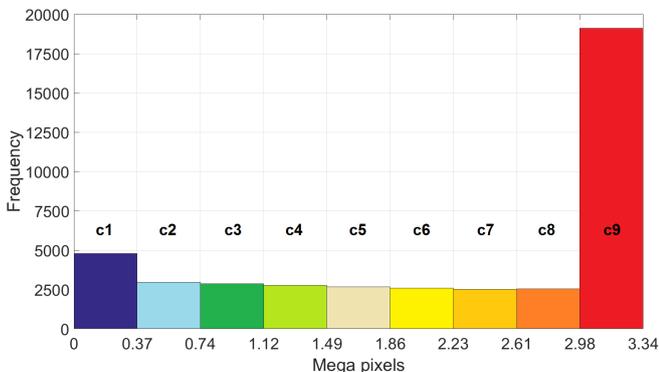


Fig. 2: Frequency according to the size of the bounding box of the data domain of the available (42,776) S2A 60m resolution bands (B01).

low size (c1–c5), whereas if one of the remaining three nodes (isfhpc1-2-3) becomes available, schedule a task associated with an image from the medium to the high size (c6–c9).

These two naive approaches together with the precise definition of the resources allocation demonstrate in practice a significant performance improvement in terms of schedule length. Figure 3 displays five makespans based on different approaches. From bottom to top: random sequence of

tasks, read the full image domain and allocate 4GB for every task; the same conditions plus the bounding on the number of concurrent jobs; random sequence of tasks, read only the data domain, allocate the appropriate amount of memory according to the groups c1–c9 at which the tile bands under

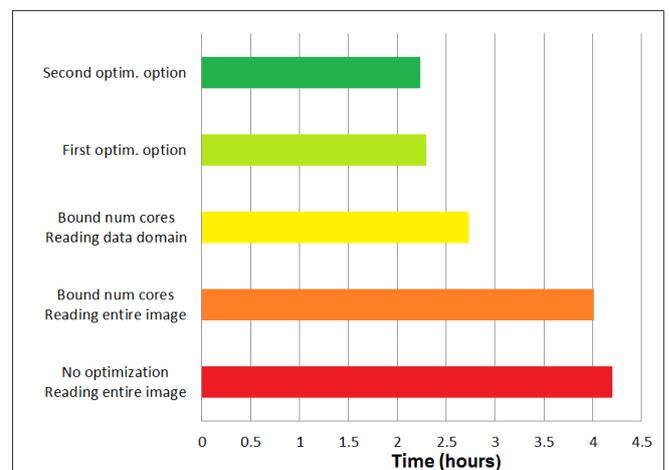


Fig. 3: Total execution time of the cloud detector (adjusted ACCA) based on the 6 bands of 1,000 S2A tiles, derived from five different schedules and managed by SLURM.

processing belong and bound the number of concurrent jobs; optimized task schedule based on the first strategy; optimized task schedule based on the second strategy. The bounding on the concurrent jobs is equivalent to the constraint imposed over the number of cores per node according to the profiling outcome (Figures 1d–1f). The explanation of the fact that a sorted list based on the size of input data domain leads to a shorter schedule length is that a homogeneous resource allocation can be achieved, permitting the fast nodes to process the larger amount of input data with the lowest waiting time. This phenomenon is amplified with the second strategy where the slower nodes are imposed to process the heavier tasks, leaving the space to the fast nodes to process evenly all the different types of tasks; in that way, they remain always occupied but in an effective mode. Table 2 shows the stability on the results when different proportions on the size of the nine groups are considered. The first column contains four different data domain distributions based on the nine bins: almost even, **c3–c6** reinforced, skewed in favour of **c1–c5**, skewed in favour of **c7–c9**. The scenarios **sc3–sc5** refer to the last three approaches respectively presented in Figure 3. The **sc1–sc2** scenarios refer to two ideal cases where the available memory is unlimited and not considered as constraint; the only difference is that in **sc1** a random task list is used as opposed to the sorted list in **sc2** case. The results derived from simulations executed 10 times for each scenario. One conclusion by observing these results is that both optimization strategies give stable schedules independently of the task list length or the input data distribution. In addition, the second strategy starts to lead to better performance in cases where the input data are of medium to low size; if the task list is dominated by input data of high size then both approaches give very similar performance.

The results validate the need for an informed preparation of the task list, and subsequently, for an adaptive decision making during the tasks execution. Apart from this finding, this work shows how the whole process can be automatized and relief the non specialized user from the burden of the task list formation.

Table 2: Simulation results when considering different data domain distributions and five scenarios (Time in hours).

Ratio by group	sc1	sc2	sc3	sc4	sc5
.1,.09,.08,.08,.08, .09,.12,.2,.16	1.68±.05	1.54±.00	2.97±.06	2.40±.01	2.35±.01
.02,.02,.2,.2,.2, .22,.03,.05,.04	1.49±.02	1.38±.00	2.62±.02	1.94±.01	1.83±.00
.2,.18,.16,.16,.16, .02,.02,.04,.03	1.11±.03	0.98±.00	1.92±.03	1.28±.01	1.16±.01
.02,.08,.01,.01, .01,.02,.23,.37,.3	2.15±.01	2.03±.00	3.85±.02	3.43±.00	3.43±.02

4. CONCLUSION AND OUTLOOK

This work demonstrates an automated method for fine-tuning a workload manager (SLURM in this specific application) in

order to optimally distribute tasks in an heterogeneous computer cluster and do large-scale experiments with geospatial data. Job scheduling and resource allocation in high performance distributed environments [2, 5, 1] is now a wide area application, including also cloud computing resource scheduling [10]. Future work involves: i) the extension of the optimization problem in order to take into consideration factors like network bandwidth, users collision and composite jobs, and ii) the transferring and adaptation of the problem formulation to similar applications such as the dynamic allocation of virtual machines or containerised environments.

5. REFERENCES

- [1] T. D. Braun, et al. Static resource allocation for heterogeneous computing environments with tasks having dependencies, priorities, deadlines, and multiple versions. *J. Parallel Distrib. Comput.*, 68(11):1504–1516, November 2008. doi: 10.1016/j.jpdc.2008.06.006.
- [2] H. Hussain, et al. A survey on resource allocation in high performance distributed computing systems. *Parallel Computing*, 39(11):709–736, 2013. doi: 10.1016/j.parco.2013.09.009.
- [3] M. Jette et al. Slurm: Simple linux utility for resource management. In *Proc. of ClusterWorld Conference and Expo (San Jose, California)*, 2003.
- [4] S. Kiemle. Requirements for EO data processing farms. https://wiki.services.eoportal.org/attachments/HMA_WORKSHOP/20121012/Presentations-Thursday/PM/2012-10-11_DLR-Requirements-for-EO-Data-Processing-Farms.1.0.pptx, 2012.
- [5] J. Polo, et al. Resource-aware adaptive scheduling for mapreduce clusters. In *Proc. of the 12th ACM/IFIP/USENIX International Conference on Middleware*, Middleware’11, pages 187–207, 2011. doi: 10.1007/978-3-642-25821-3_10.
- [6] P. Soille. *IMAGE-2006 Mosaic: SPOT-4 HRVIR, SPOT-5 HRG, and IRS-LISS III Cloud Detection*. JRC, European Commission, 2008. doi: 10.2788/49355.
- [7] P. Soille. Seamless mosaicing of very high resolution satellite data at continental scale. In P. Soille et al., editors, *Proc. of the 2014 Conference on Big Data from Space*, pages 222–223, 2014. doi: 10.2788/1823.
- [8] S. Soner et al. Topologically aware job scheduling for slurm. <http://www.prace-ri.eu/IMG/pdf/WP180.pdf>.
- [9] Q. Wu et al. Runtime task allocation in multicore packet processing systems. *IEEE Trans. Parallel Distrib. Syst.*, 23(10):1934–1943, 2012.
- [10] Z. Zhan, et al. Cloud computing resource scheduling and a survey of its evolutionary approaches. *ACM Comput. Surv.*, 47(4):63:1–63:33, July 2015. doi: 10.1145/2788397.