ECAI 2016
G.A. Kaminka et al. (Eds.)

doi:10.3233/978-1-61499-672-9-1598

DARDIS: Distributed And Randomized DIspatching and Scheduling

Thomas Bridi [1] and **Michele Lombardi** [1] and **Andrea Bartolini** [2 3] and **Luca Benini** [2 3] and **Michela Milano** [1]

Abstract. Scheduling and dispatching are critical enabling technologies in supercomputing and grid computing. In these contexts, scalability is an issue: we have to allocate and schedule up to tens of thousands of tasks on tens of thousands of resources. This problem scale is out of reach for complete and centralized scheduling approaches.

We propose a distributed allocation and scheduling paradigm called DARDIS that is lightweight, scalable and fully customizable in many domains. In DARDIS each task offloads to the available resources the computation of a probability index associated with each possible start time for the given task on the specific resource. The task then selects the proper resource and start time on the basis of the above probability.

1 Introduction

Large-scale computing infrastructure like grids and High-Performance Computing (HPC) facilities require efficient workload scheduling and dispatching.

Consider for example the number of computational nodes a scheduler has to manage for high-performance computers like the top 1 HPC in 2015 or the future exascale HPC. This machine features a number of nodes estimated between 50'000 and 1'000'000 [3]. Classical job schedulers are rule-based. These are heuristic schedulers that use rules to prioritize jobs. In these scheduling systems, a job requests a set of resources on which the job will execute. The scheduler checks for each job if it can execute on a node while respecting the capacity of the target resources. If the job can use the requested amount of resources, the job is executed. It is quite clear that for these large scale machines a centralized, optimization-based scheduler [2, 1], is not a feasible option. Hence, scalable, distributed schedulers are needed to handle thousands of nodes while at the same time optimizing efficiency metrics.

This work takes inspiration from Randomized Load Control proposed in [4]. We substantially extend this work to the case of multiple resources and introduce new start times generators and dispatching policies.

In this work we present a Distributed And Randomized DIspatching and Scheduling (DARDIS) approach that is:

- **Distributed** to scale to an ultra-large system. The scheduler and dispatcher basically leave the dispatching choice to the task and each resource then schedules its own tasks.
- **Supporting variable resources' utilization profile**. Each resource, besides its capacity, exhibits a (variable) desired utilization profile.
- **Randomized**. The scheduler can choose the proper probability distribution for selecting resources and start times to optimize different objective functions.
- **Deadlines aware**. Each activity can specify a time window in which it should start.

Tests against classical commercial scheduler, on standard job traces, show an improvement of 2.6% in makespan and 18% in the total job waiting while having a scheduler 42 times faster in term of computational overhead.

2 Approach

The workload dispatching and scheduling problem can be modeled by a set of resources res_r, with $r \in R$ and a set of activities a_i with $i \in A$. Each resource has a capacity c_r, a desired profile $dp_r(t)$ and a utilization profile $up_r(t)$, with $t \in [0,..,Eoh]$. The desired profile is a profile decided by the administrator that shows how the resource should be used (in term of number of amount of resource used by activities) in time. The utilization profile is the amount of resource already used and reserved to scheduled activities. This profile is a periodic profile repeated in time. As in example for HPC and grid computing this could be a daily utilization profile.

Each activity is submitted to the system in a time instant q_i. At the submission it specifies its earliest start time est_i, the latest start time lst_i, its duration wt_i, and the amount of resource required req_i.

The scheduling problem consists of allocating each activity to a given resource and assigning it a start time st_i and a resource ur_i such that

$$\begin{aligned}
&st_i :: [est_i..lst_i] \ \forall i \in A \\
&ur_i \in res_r \ \forall i \in A, r \in R \\
&\sum_i req_i \leq c_r \ \forall i \in A | st_i \leq t \wedge st_i + wt_i > t, \forall t \in [0,..,Eoh] \\
&\sum_i req_i \leq dp_r(t) \ \forall i \in A | st_i \leq t \wedge st_i + wt_i > t, \forall t \in [0,..,Eoh]
\end{aligned} \tag{1}$$

The main idea is to partition the decision process in two main phases performed by two separate software entities: the task agent and the resource manager. The *task agent* is responsible for the activity submission and the dispatching. This agent resides into the user-space. The *resource manager* is responsible for the scheduling. This agent resides into the resources host.

[1] DISI, University of Bologna, Viale Risorgimento 2, 40123, Bologna, Italy. {thomas.bridi,michele.lombardi2,michela.milano}@unibo.it
[2] DEI, University of Bologna, Viale Risorgimento 2, 40123, Bologna, Italy. {a.bartolini,luca.benini}@unibo.it
[3] Integrated Systems Laboratory, ETH Zurich, Switzerland.

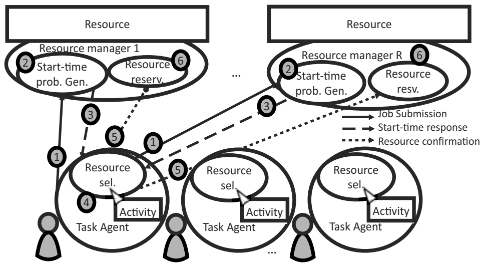

Figure 1: DARDIS architecture and phases (number ordering corresponds to time progression)

Figure 1 shows the different phases of our approach. These phases are subdivided in:

Job Submission (1) - Our approach starts with a task agent submitting an activity to all the resource managers of the system. After the submission to all the resource managers, the task agent waits for the responses.

Start time probability generation (2) - Each resource manager receives the submitted activity and starts the start time probability generation phase in which the manager generates a start time for the activity according to an internal rule (Section 2.1).

Start time response (3) - After the start time generation, the resource manager sends a generated start time to the task agent.

Resource selection (4) - The task agent, after receiving the responses from all the resources, applies a policy (Section 2.2) to select the resources for the activity execution.

Resource confirmation (5) - The task agent sends the result to all the resource managers involved in the submission, namely, the one selected and those not selected.

Resource reservation (6) - The resource managers in which the activity has to execute, reserves the proper capacity for the execution, by modifying the utilization profile.

2.1 Start time probability generation

The start time generation process for the resource j starts by computing a fitting index for the submitted activity i. This index indicates how many parallel runs of the same activity could be executed in a given start time s while satisfying the desired utilization profile for the resource. Due to the variability over time of the desired profile, we have to check for each time instant $t \in \{s, .., s + wt_i\}$ how many times the activity's resource requirement req_i can fit the space left between the utilization profile and the desired profile (equation 2).

$$I'(s) = min_t(\frac{dp_j(t) - up_j(t)}{req_i})\ \forall t \in \{s, .., s + wt_i\} \qquad (2)$$

Note that $I'(s) = 1$ means that the activity perfectly fits into the resource without exceeding the desired profile. $I'(s) > 1$ means that the activity fits the desired profile and leaves some resource for other activities. If $I'(s) < 1$, it means that the activity exceeds the desired profile. The capacity instead cannot be exceeded by definition. To handle this case, we use equation 3. Where $\setminus$ represents integer division.

$$I(s) = min(I'(s), min_t((c_j - up_j(t)) \setminus req_i)) \forall t \in \{s, .., s + wt_i\} \qquad (3)$$

The index distribution I is calculated for each possible start time between the earliest start time est_i and the latest start time lst_i of the activity: $I = \{I(est_i), .., I(lst_i)\}$. In this way, we obtain the fitting profile for the activity.

We defined three generators for the start time selection:

- **First**: the goal of this star-time generation procedure is to maximize the throughput of the entire system. This deterministic selection works by picking up the first feasible start time.
- **Uniform**: the goal of this generator is to produce a scheduler that allocates resources following the shape of the desired profile for its entire window. This is a probabilistic selection that chooses the start times randomly.
- **Exponential**: this generator has been designed to reach a trade-off between throughput and profile chase. This is a probabilistic generator that chooses a start time following an exponential distribution.

2.2 Resource selection

After the start time generation process, the task agent receives the responses from all the resource managers involved in the submission. This algorithm selects the resources for the activity execution.

The designed policies are:

- **MIN_START**: it selects the resource that will execute the activity first. This approach goes in the direction of optimizing the activity throughput.
- **MAX_PROB**: it selects the resource that gives the highest fitting index. This means that it selects the most unloaded resource. This approach is designed to minimize the standard deviation from the desired profile.
- **MIN_PROB**: it selects the resource that gives the lowest fitting index. This policy is designed to ensure the best fitting for the desired profile. This is useful when we have to prefer solutions that saturate one resource before starting filling another one.
- **RANDOM**: it selects randomly the resources using a uniform. This policy is designed to enforce each resource to have the same probability of hosting an activity.

ACKNOWLEDGEMENTS

This work was partially supported by the FP7 ERC Advance project MULTITHERMAN (g.a. 291125), by the YINS RTD project (no. 20NA21 150939), evaluated by the Swiss NSF and funded by Nano-Tera.ch with Swiss Confederation financing and by CINECA.

REFERENCES

[1] Andrea Bartolini, Andrea Borghesi, Thomas Bridi, Michele Lombardi, and Michela Milano, 'Proactive workload dispatching on the eurora supercomputer', in *Principles and Practice of Constraint Programming*, ed., Barry OSullivan, Lecture Notes in Computer Science, Springer International Publishing, (2014).

[2] Thomas Bridi, Andrea Bartolini, Michele Lombardi, Michela Milano, and Luca Benini, 'A constraint programming scheduler for heterogeneous high-performance computing machines', *IEEE Transactions on Parallel & Distributed Systems*, (2016).

[3] JF Lavignon et al. Etp4hpc strategic research agenda achieving hpc leadership in europe. `http://www.etp4hpc.eu/wp-content/uploads/2013/06/ETP4HPC_book_singlePage.pdf`, 2013.

[4] Menkes Van Den Briel, Paul Scott, and Sylvie Thiébaux, 'Randomized load control: A simple distributed approach for scheduling smart appliances', in *Proceedings of the 23th international joint conference on Artificial Intelligence*. AAAI Press, (2013).