Relationships between Scheduling and Autonomic Computing Techniques Applied to Parallel Computing Resource Management

Manal Benaissa

Relationships between Scheduling and Autonomic Computing Techniques Applied to Parallel Computing Resource Management

Manal BENAISSA
Master 2 MOSIG Data Infrastructure
Université Grenoble Alpes

Supervisors: Raphaël BLEUSE and Eric RUTTEN
Team CTRL-A

This Masters research project will be defended before a jury composed of:
Bruno RAFFIN (President of the jury)
Martin HEUSSE (Examiner)
Olivier RICHARD (External Expert)
Raphaël BLEUSE (Supervisor)

Abstract

The scheduling field regroups various methods by which work is distributed across available computational resources. Considering the complexity of modern infrastructures, particularly in Cloud and HPC computing, schedulers might face difficulties to propose an efficient scheduling while fitting to the system reality and it implied uncertainties. The autonomic computing field suggests a more practical approach, by controlling constantly a system and adjusting taken decisions at runtime, via feedback loops. Applying this strategy in the scheduling context bring a less complex model with a more modular structure. Likewise, scheduling community may bring a new vision of autonomic computing challenges. This work presents some possible models, from the most basic scheduling algorithm, to the most complex Cloud infrastructures.

***

Le domaine de l'ordonnancement regroupe diverses méthodes, où le travail est distribué à travers les différentes unités de calcul disponibles. En considérant la complexité des infrastructures actuelles, particulièrement dans le domaine du Cloud et du HPC, les ordonnanceurs peuvent rencontrer des difficultés à concilier la proposition d'ordonnancements efficaces et les incertitudes liées à la réalité du système. Le domaine de l'informatique autonome propose une approche plus pratique, en contrôlant régulièrement un système et en ajustant les décisions prises durant l'exécution, par l'intermédiaire de boucles de rétroaction. Appliquer cette stratégie dans le contexte de l'ordonnancement apporte un modèle moins complexe et une structure plus modulaire. De même, la communauté de l'ordonnancement pourrait apporter un regard nouveau sur les problèmes que rencontre la communauté de l'informatique autonome. Ce travail présente quelques modèles possibles, du plus basique algorithme d'ordonnancement aux plus complexes infrastructures Cloud.

1. CONTEXT

Cloud and HPC computing are both based on a complex infrastructure composed of servers, data storage units, network connections and even software, dedicated to users. The system offer platform access as a service, for personal or professional purpose in Cloud computing, and mainly for scientific purpose in HPC context. More and more technologies in these fields need complex and time-consuming computations, which require to parallelize several tasks in highly complex infrastructures. Using classical scheduling algorithms that only take in account the amount of work and the number of compute units is not realistic in such systems. In fact, many other parameters like the cost of communications, dependencies between tasks or elasticity of the infrastructure bring complications in the search of optimal scheduling. Typically, Cloud infrastructure is generally composed of tens of data-centers scattered around the world and inter-connected by wide area networks (WAN). Every days, each data-center has to process some data, and execute many jobs. Most of the time, these jobs are dependent of each others, and need data from a far away data-center. Scheduler function is to distribute these jobs across different resources, taking into account Quality of Service and infrastructure constraints.

In 2000s, IBM proposed the *autonomic computing* approach to address such challenges. A controller is added to monitor the system and adjust decision when this one is no more adapted. This implies a constant checking of the system state and the computation of an adapted response of a non desired evolution, forming a feedback loop. Even if this strategy exists in Cloud computing, it is not always clearly defined. Most of the time, a static analysis is done, despite uncertainties due to the overall architecture. Nevertheless, the control loop concept can even be found in HPC systems, where the adopted approach is based on a more theoretical performance analysis. In this work, after a brief

definition of each community politic, some possible strategies that promote collaboration between scheduling and autonomic community will be explored.

2. STATE OF THE ART: SCHEDULING

More and more computer science fields require to solve complex and time-consuming problems in short time, and the idea of dividing these kinds of problems in small blocks distributed across several compute units came quite intuitively. However, the parallelization of algorithms implies other problems like dependencies between these blocks, distribution between all available compute units, time and resource constraints and so on. R.L Graham [1] was the first to formalize scheduling constraints and to propose an approach in 1966 with the *List Scheduling*. After that, many strategies were explored, particularly in operational research, HPC and Cloud computing.

2.1 Scheduling problem

Scheduling problems are a part of constrained optimization problems, widely known in operational research field. Scheduling regroups methods by which work is distributed across available computational resources. Infrastructure is considered by following a model where work can be a task (block of code in program) or a job (an entire program). This work can be distributed across physical nodes like CPU, machines or even clusters, or logical nodes like threads.

Hence this notation allows to formalize the associated problem, by considering three aspects:

α the state of the system where parallelism is applied: Number of compute units, and whether all of them are identical.

β constraints on work: Number of task/job, dependencies between them, and processing time.

γ desired objective: minimizing execution time over all tasks, maximizing the total amount of work per time unit, or limiting lateness when a deadline is defined for example.

Graham gathered all these constraints in a notation system, the Graham notation [2]. In this work, only the main criteria will be described. SchedulingZoo [3] gathers a more complete description about this notation. Here, $< \alpha|\beta|\gamma >$ is chosen depending of the criteria described in Table 1 (this table is only an extract of the Graham notation).

For example, to describe a scheduling problem that minimizes the execution time and work on n identical parallel nodes, with k jobs linked by precedence constraints, we will write:

$$< Pn|prec|C_{max} >$$

A way to visualize this problem is to consider n identical nodes P_i, $1 \leq i \leq n$ and m tasks T_j, $1 \leq j \leq m$ with dependencies. These dependencies are specified with $\prec$ order relation: $T_i \prec T_j$ if T_j is dependent of T_i, which means that T_j can not start its execution if T_i is not finished. Each task needs a certain amount of time to be executed, given by the function μ. With this data, we obtain a dependency graph $G(\prec, \mu)$, where each vertex indicates a task and its processing time, and each edge gives existing dependency between two tasks.

Machine environment α	
1	Single node
2	Two nodes
m	m nodes
P	All nodes are identical, aka execution time is the same, no matter the chosen node
R	Nodes are unrelated, which means execution time is not the same
Constraints β	
prec	Dependencies exist between tasks, which means task T1 has to be finished before starting task T2, if T2 is dependent of T1.
$pj = p$	Each task has the same processing time p.
pmtn	Allows preemption, tasks can be suspended and continued later, possibly by an other node.
Objective function γ	
C_{max}	Makespan, the total execution time between the start of the first job and the end of the last job.

Table 1: Criteria in Graham notation [2].

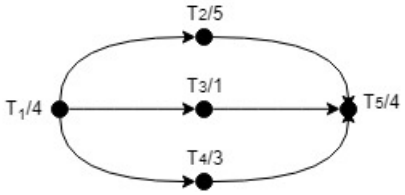

Figure 1: Dependency graph example [1]

For such a problem where the set of tasks and execution time are given, solutions exists and they are well known. Given a dependency graph $G(\prec, \mu)$,a certain number of computing nodes n and possibly some other elements, a Gantt diagram ξ can be obtained (cf Figure 2). This diagram will depend on the following algorithm, and must respect all specified constraints.

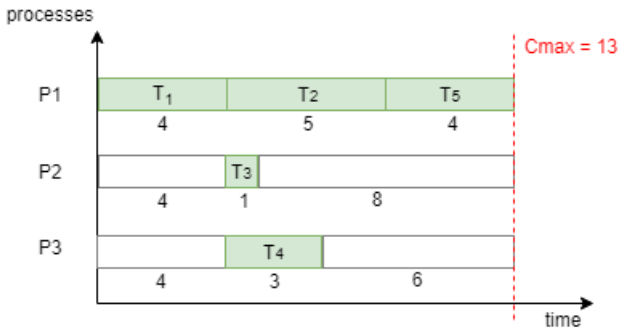

Figure 2: A possible Gantt diagram[1] of the dependency graph in the Figure 1, for $n = 3$

Online Scheduling

In online scheduling, the scheduler receives jobs that arrive over time, and generally must schedule the jobs without any knowledge of the future. The lack of knowledge of the future precludes the scheduler from guaranteeing optimal schedules[4]. The easiest way to understand scheduling is to imagine a finite set of tasks where we know duration of each tasks. These problems where all constraints are fixed are well known. However, many systems don't know in advance neither the number of tasks, nor their duration. In Cloud context for example, each client request come as a job, and they might arrive one by one as an input stream. On top of that, we distinguish non-clairvoyant problems which job computation time is not known. Despite this lack of information, solution exists and they are based on offline algorithm approximation. More precisely, we compare an online algorithm A to an optimal offline algorithm OPT that knows the entire input in advance.

Clairvoyant.

Clairvoyant scheduling constitutes a part of online scheduling, where processing requirement of each job is specified, but existence of jobs stay unknown until a certain *release time*.The jobs characteristics are not known until they arrive, restricting the scheduler to schedule jobs only with current information. However, once these ones are released, the job size (μ function) is known.

Non-Clairvoyant.

In more realistic models, the processing requirement of each job is also unknown, and can only be determined by processing the job and observing its execution time.

2.2 Scheduling algorithms

2.2.1 *Offline scheduling*

In the offline scheduling context, the *List Scheduling* is the most intuitive solution, particularly for the $< Pn|prec|C_{max} >$ problem described in section 2.1. Given a dependency graph $G(\prec, \mu)$, a certain number of processing nodes n, and an ordered list L, an optimal scheduling can be obtained by following some rules. We consider that a task T_j is available, and can be executed by a node if:

- It is not taken by an other node.
- It is not dependent of an other task.
- It is dependent, but all preceding tasks are already executed and finished.

The processing node P_i takes the **first available** task T_j in the list L. If no task is available in L, P_j become idle, by executing an empty task φ_k. This task ends when an other task finishes in an other node. We obtains the following Gantt diagram in the figure 3.

FIFO scheduling is the most used algorithm in Cloud computing. It's a variant of the List Scheduling, where the ordered list L is fixed: tasks are treated with the order they arrive. They are ordered in L by they release date, and the first task to be executed is the one with the earliest release time.

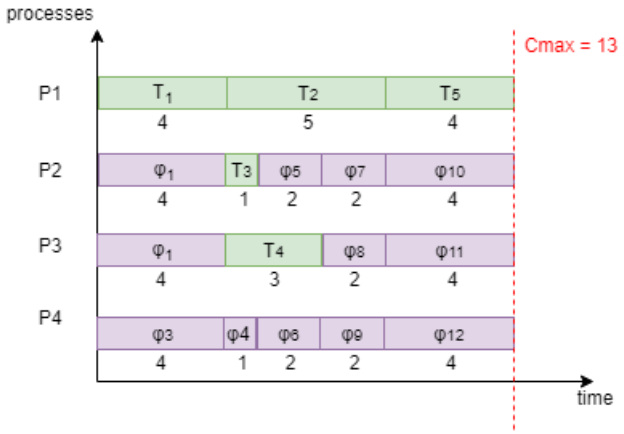

Figure 3: Gantt diagram[1] of the dependency graph in the Figure 1, for $n = 4$ and $L = [T_1, T_2, T_3, T_4, T_5]$

2.2.2 *Online clairvoyant scheduling*

To adapt offline solutions to an online model, Schmoys described in 1995 a method to converts algorithms that need complete knowledge of the input data into ones that need less knowledge. The idea is to use an offline algorithm (such as List Scheduling) to schedule a subset of jobs, released at each time interval. The system is defined by a set of computing nodes and a waiting queue for arriving jobs.
At time $t = 0$, a certain number S_0 of jobs was queued in the system queue. At $t = 0$, a snapshot of the queue state is taken by the scheduler. Since we are in a clairvoyant scheduling, we know the execution time of each jobs, and then we can estimate C_{max} of the S_0 scheduling. The offline scheduler schedules only these S_0 jobs in the queue, until this queue is considered empty by the snapshot view. In the same time, some jobs arrive in the queue and wait until S_0 schedule ends. These waiting jobs compose S_1 session. So when S_0 schedule ends (i.e at $t = C_{max}(S0)$), S_1 jobs are captured in an other snapshot and they are scheduled while S_2 jobs arrive in system queue. In the Figure 4, the release of S_0, S_1 etc (in red) corresponds to the moment when the snapshot is taken by the scheduler. Only jobs taken in each snapshot are scheduled (in green), the other arriving jobs wait until the next session.

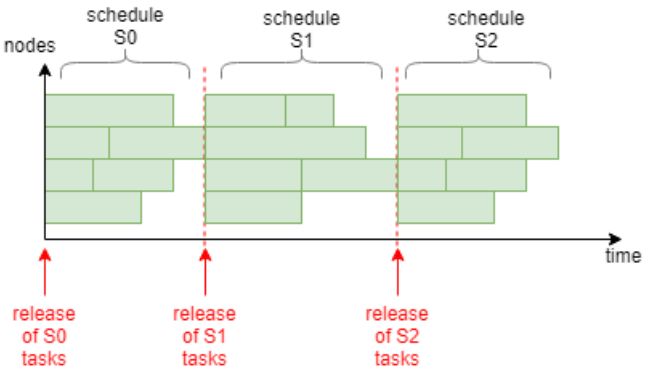

Figure 4: Shmoys algorithm [5]

2.3 Resources and Jobs Management System

Resources and Jobs Management Systems (RJMSs) [6] [7] are specific software specialized in distribution of computing power to user jobs within a parallel computing infrastructure. RJMS has a key role in HPC infrastructures: it collects all requests from users and scheduling them, while managing all resources of the system. An application (or executable) is grouped with some data like an estimation of the computation time and needed resources. All applications are queued to be scheduled next. The RJMS must manage these two part, to satisfy users demands for computation on one hand, and achieve a good performance in overall system utilization by efficiently assigning jobs to resources on the other hand.

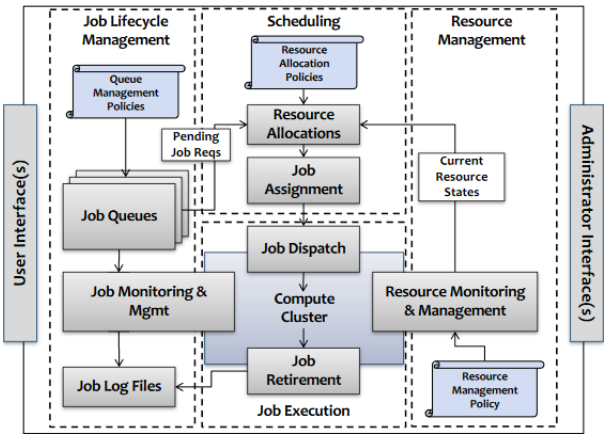

Figure 5: RJMS in HPC infrastructure [8]

The work of a RJMS is classically divided in three parts [8] [7] :

- Transforming a user request into an application, where all needed characteristics are specified.
- Scheduling these applications.
- Placement and execution of these applications in their allocated resources.

These three parts are respectively managed by the jobs manager, the scheduler and the resources manager (cf Figure 5).

2.3.1 Resource Manager

This part is responsible to collect and provide all information concerning computing nodes. This information is used by the scheduler and by users to inform about the availability and the state of the cluster. Resource Manager is responsible of these tasks:

Resource Treatment This part collect information about cluster structure and node characteristics. In fact, clusters are currently heterogeneous systems with different hardware components and different levels of hierarchies inside one node (cpu, core, thread...), for a given network.

Resource Deployment Resource Manager takes information about jobs from the Job Manager and the Scheduler, and initializes required resources.

Task placement Nowadays, a single computing node can have multiple cpu. In Cloud context particularly, since several users can share the same computing node, tasks need to be placed correctly, to avoiding collisions.

Resource Management Advanced Features Includes faults control and energy optimization.

2.3.2 Job Manager

Job Manager manages tasks related to the declaration and control of users jobs. Its role is decomposed in these functions:

Job declaration Users use RJMS interface to describe their jobs characteristics and select the resources of their preference. A job can be *interactive* (user want to be directly connected to the node, to launch his experiment manually), a *batch* job (used for direct script execution upon the allocated computing node), *bulk* jobs (one large job can be decomposed into smaller jobs, for a better scheduling), *best-effort* job (this kind of job has a low priority and can be killed if a normal job demands the resource) etc...

Job Control users can keep control on submitted jobs that are not executed yet, by changing initial parameters (priority, input/output files...).

Job Monitoring and Visualization Allows users to follow the execution of jobs upon the cluster.

Job Management Advanced Features Includes job preemption, faults control and security features.

2.3.3 Scheduler

The scheduler constitutes the main part of the RJMS: it assigns user jobs with chosen parameter to available nodes that match with the demands.

Scheduling Algorithms Different kind of scheduling policies (described in the section 2) can be used, but the most common in this context is FIFO, with which some optimization like the *backfilling* are possible.

Queues Management All tasks are placed in one queue or more, waiting to be scheduled. These tasks can be grouped in a specific queue when they sharing some similar characteristics. The scheduler takes task in these queues, depending of its policy. An important point is to avoid famine: tasks in the lowest priority queue might be never executed if the highest priority queue is always filled. A solution to bypass this problem is to set an increasing priority depending on the waiting time.

3. STATE OF THE ART: AUTONOMIC COMPUTING

Despite all efforts to optimize the system robustness, anticipating non desired behavior in complex infrastructure without controlling it regularly can be tricky. The notion of autonomic computing[9] was introduced by IBM in 2001, but the concept of feedback loop was commonly used in engineering. The aim was to tend toward a fully autonomous infrastructure, with a self-management of work. A security

is added to control periodically system outputs and adjust inputs when these ones exceed permitted values. A common example is temperature control in machines: A controller will check constantly temperature evolution. When this one indicates an overheating, the controller adjust machine behavior by increasing its cooling system or decreasing energy consumption. Adequate responses to an anomaly are not necessarily based on a solid knowledge of the system, only few elements are enough. In this way, a feedback loop provides several advantages without changing all the system. Some models exist but we will focus on one of them: MAPE-K [10].

3.1 MAPE-K

The main characteristic of Autonomic computing is to offer self-management. The goal is to exempt system administrators from maintenance and to allow a continuous system working without a drop of performance. Autonomic systems adjust their behavior in the face of external changes. This characteristic includes:

Self-configuration Adding a new component in the system should not involve difficult manual operations. Instead, it must incorporate itself seamlessly, and lets the rest of the system to adapt to its presence.

Self-optimization The whole system has to improve its operations, in order to be more efficient. This process go through a learning phase, where the system monitors, experiments and adjust its behavior until a certain optimization level is reached.

Self-healing The system can detect and repair failures.

Self-protection The system must be prepared to handle malicious attacks or accidental failures, by anticipate them.

To fulfill these objectives, a controller is grafted to the managed system, forming a feedback look (cf Figure 6). The aim of the controller is to ensure that the system converges toward the desired behavior by controlling system when inputs are out of bounds, and without too large fluctuations in responses. To ensure that, we will focus on one model: *MAPE-K*. This model is divided in three main parts:

Monitor This part collects data y_k at time k, via sensors, and possibly makes some pre-processing. The main challenge here is to know collecting data frequency.

Analysis This part includes data analysis by making some estimations.

Planning This part makes an adequate action plan to adjust the system. To do that, controller follows control laws based on the knowledge of the controlled system. These control laws can follow various models based on machine learning, rule-based systems or even basic programming.

Executor This part applies decision-maker action plan, by sending data to modify u_k to the system, via actuators.

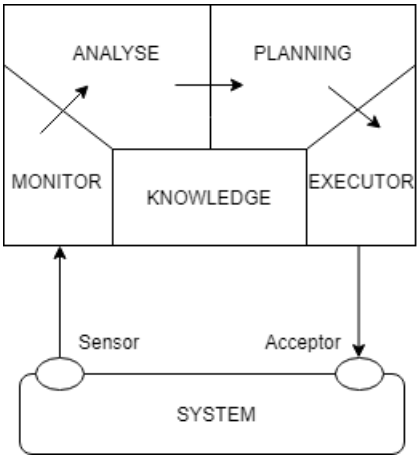

Figure 6: MAPE-K loop for control[10].

3.2 Control Theory as MAPE-K

On top of the MAKE-K model, control laws can be draw from Control theory[10] ideas. As we said before, the decision part can be based on various models, depending on the system needs, and the field of automatic control provides many potential control laws. The main interests of this field are:

Stability The system is considered as *stable*, if its inputs and disturbances stay bounded. Simply speaking, the system must converge toward the wanted behavior, by detecting inputs out of its bounds. With this approach, controller will naturally stabilize system, even when this one is instable. In fact, when inputs stay bounded, output and state will stay bounded too.

Robustness Because the system is periodically checked, with few amount of time between each control, an anomaly is quickly detected. With this method, we don't need all details about the system evolution. Only few data but regularly collected allows to correct system with enough precision.

Tracing of performance Controller checks *first* and adjust *next*. Because of that, we can draw evolution of the system at any moment.

Since the decision part is the controller core, and it will define the efficiency of this one. A bad decision will impact directly the system, and it's why the choice of control law is important. However, even if systems are different, in most case they follow the same rule: Only one variable is collected, and if its value falls outside the accepted range, only one other variable has to be changed. In the other hand, the chosen model has to be simple. A complex model will take time to compute and slow down the whole system. For these reasons, the *PID control law* (for Proportional, Integral, Derivative) is the most used. The u_k value is written as a function of the error signal $e_k = r_k - y_k$, where e_k is difference between the measured value y_k and the expected value r_k (also called *reference value*), at time k. We obtain u_k, expressed by the function:

$$u_k = K.e(t) + KT\frac{d}{dt}te(t) + KT_i\int_t^0 e(x)dx$$

In this function, we can identifies three terms:

Proportional term $K.e(t)$ where K controls the the rising time.

Derivative term $KT\frac{d}{dt}te(t)$: this part, based on change frequency, absorbs the oscillations and overshoots.

Integral term $KT_i \int_t^0 e(x)dx$: this part "memorizes" old values, to nullifies the static errors.

All factors (like K) are chosen depending on the system, by testing different values and estimate the most adapted ones. In many simplest models, only the proportional term is used and completed by derivative and integral term when it's required. Of course, PID model is not always sufficient, particularly if system behavior cannot be expressed by linear models.

4. PROBLEM STATEMENT

Scheduling computing field gathers many optimization methods to distribute work across all resources. As we saw in Graham notation, a precise representation of the system should be relevant (number and characteristics of computing nodes, description of the work...). But this representation reflects partially the reality of the system, despite a worse case analysis. Dues to this partial capture of the reality, the representation implies uncertainties. This variability, mainly explained by the architecture, can be captured with an online approach. Autonomic computing community suggests a more practical strategy, by adapting to the system. It gives a compromise between a more simple model but lacks of performance and stability guarantees.

In this way, autonomic computing and scheduling computing stay two distinct fields in computer science. However, scheduling community would benefit from control loop concept, autonomic computing community can use scheduling techniques as decision-maker in controller, and the merge of these two domains deserves further consideration. Scheduling community uses worst case analysis to evaluate a scheduling method, but these worst case can be avoided with a control loop. Some scheduling system already includes feedback loops in the structure, but in an implicit way. Highlighting these control loop in such a system will allow to fully exploit autonomic computing methods and formalize properly this kind of approach. In this way, autonomic computing community can bring a better expertise in scheduling computing field. In this paper, we will present some case where we can bring out a feedback loop in a scheduling system, by identifying each part of the potential MAPE-K pattern.

5. COMBINING SCHEDULING AND AUTONOMIC MANAGEMENT

The intersection between scheduling and autonomic computing field can be seen in different ways. In this part, we study existing cases and re-interpret them through the prism of autonomic computing. Each example will focus on a certain abstraction level, from the fine grain at the scheduler level to the Cloud infrastructure.

5.1 Feedback loop in a scheduling algorithm context

A first approach in the merge of these two domains is to reinterpret an online scheduler as a controller with MAPE-K pattern and scheduling policies as control laws. To do that, we will use Shmoys[5] technique seen previously: This method consists to use an offline scheduler at each cycle, defined by the state of system queue at a certain time. The controlled system is defined by a set of computing nodes and a waiting queue of jobs. In the clairvoyant context, released jobs and their execution times are known. Let us consider sensors that capture queue snapshots described in the section 2.2.2, allowing a monitoring of the queue state. An execution time C_{max} can be estimated (in the analysis part) and a decision (the scheduling process itself) can be made. This decision is injected directly on the nodes set via implicit actuators, thus closing the loop (cf Figure 7). MAPE-K pattern is clearly defined here, where the chosen control law is an offline scheduling algorithm.

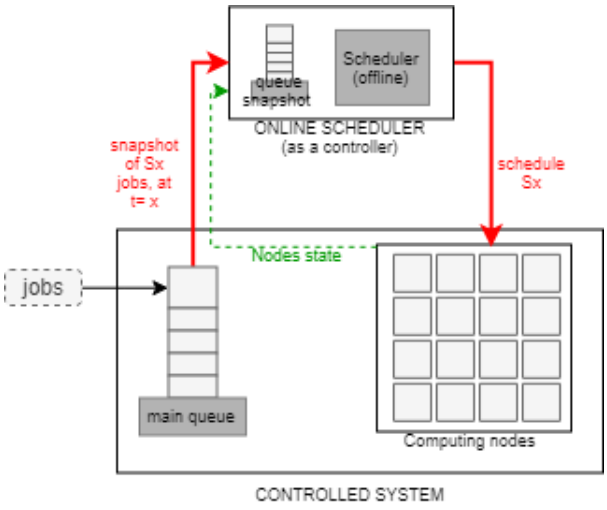

Figure 7: Online scheduler as a controller. Dotted arrows corresponds to possible additional monitoring. Plain arrows constitute the main control loop.

This approach can be adapted to pure online algorithms, by considering a sensor in the system queue and actuators that apply scheduler decisions. Capture frequency can be adjusted, depending on the computing cost of the scheduling algorithm and the jobs themselves. In the case above, capturing a snapshot not at each C_{max} but earlier may end up in a better result. It also may interesting to capture more information during monitoring, like the state of each node (green dotted arrow in the Figure 7). Knowing if some nodes crashed, or if they are overloaded may lead to a finer analysis, and thus to a better decision.

5.2 Feedback loop in a RJMS context

A scheduler can be interpreted as a closed loop, as shown in the previous section. It is therefore natural to consider feedback loops at the RJMS level, not only in the scheduling part itself but at the jobs and resources management level too. This approach was adopted in the HPC context, to exploit unused resources in a cluster. Grid5000[11] is a grid for computing experiments with more than 5000 cores dispatched in 11 sites, mainly in France. OAR[12] is its associated RJMS, highly configurable and based on a batch scheduler. Like many RJMS, OAR can use many queues

to place user tasks, depending on the characteristic and the priority of the submitted task. This strategy allows to respect QoS constraints, while exploiting the maximum of the clusters capacity. However, some resources can remain unused despite these efforts. It's why CiGri was implemented in top of OAR.

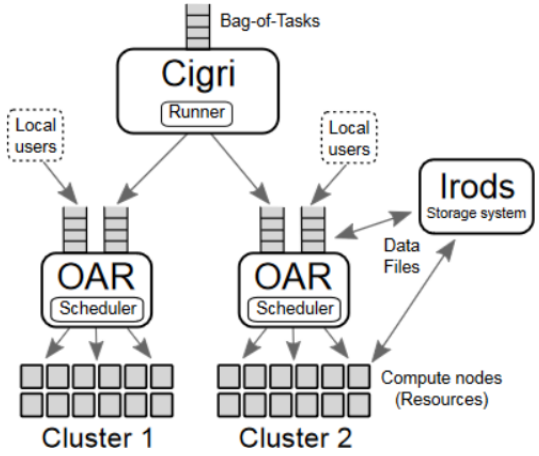

Figure 8: Overview of Grid5000 RJMS[13].

Each OAR scheduler is configured with a dedicated queue for CiGri. When a user submit a task, this task can be placed directly in OAR priority queue, to be executed in the best conditions possible. Otherwise, it can be placed in the CiGri queue, particularly if the task is small and it has a low priority. In this way, CiGri can place these numerous small tasks on the remaining resources. At each cycle, CiGri submits a number of tasks called *rate* to the clusters, and wait the end of the execution of all these tasks. Then, in the next cycle, Cigri submits an other number of tasks, with an higher *rate*, and wait. This rate increases at each cycle, depending on clusters availability. This method can be seen as a tasks flow, where CiGri is the tap. This flow is nevertheless mismanaged:

- CiGri submits tasks to OAR scheduler only if OAR priority queue is empty. Therefore, resources might remain unused while tasks in CiGri queue are still waiting (cf red dotted arrow in Figure 10).
- Clusters (or the storage system) are overloaded because too many tasks are submitted by CiGri.

To avoid these problems, a feedback loop was implemented (in green in Figure 9). This control loop reuses all MAPE-K and control theory concepts by monitoring the load in clusters queues and controlling the number of tasks submitted by CiGri. We recognize all parts of the MAPE-K pattern, with:

Monitoring The number of tasks y_k in clusters queues is measured.

Analysis The gap e_k between y_k and the expected value r_k is computed, giving the idea of CiGri behavior.

Planning Depending on e_k, more or less tasks will be submitted by CiGri. To know how many tasks u_k CiGri should submit, the controller use PI control law.

Executing At the end, the final decision is injected to CiGri.

These decisions are drawn on system knowledge that give the controller sensibility and responsiveness. This knowledge is mainly based on experimentation. Adding a controller in the system permitted to gain 8% in clusters usage, and increase efficiency of resources management.

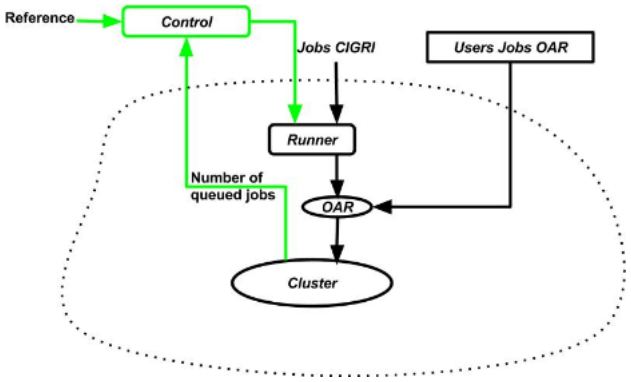

Figure 9: Feedback loop in CiGri cluster[13].

Another way to see this infrastructure is to consider CiGri as a single controller, and OAR as a basic RJMS. The controller seen previously becomes a integral part of CiGri: The number of tasks in clusters queues is monitored as before, but here, the number of submitted jobs in OAR low priority queue is directly monitored (cf. Figure 10).The system knowledge of this version is not static anymore, as long as the number of jobs in CiGri queue changes. However, we saw in the previous section that a scheduler can be seen as a controller too. Similarly, OAR can be interpreted as a feedback loop, where jobs and resources are monitored to influence scheduling decision. So here, the overall system can be viewed as the cooperation of the OAR controller and the CiGri controller. The OAR controller manages high priority jobs that they can not be preempted (i.e they can not be interrupted to be continued later by a possible other computing node). CiGri controller injects small preemptive jobs to maximize the resource usage without affecting QoS constraints. It's important to note that a feedback loop already existed before adding the controller: CiGri controls clusters and OAR state and injects more or less jobs in OAR low priority queue, depending on this information. This loop can be viewed in Figure 10 (with the red plain arrows).

As we said previously, more sensors can be added, to capture more information about OAR state. CiGri already controls the OAR high priority queue, but the only thing captured here is if this queue is empty or not. Rather than monitoring only that, CiGri may check the queue content: maybe some low priority jobs can be placed if this placement will not disrupt the execution of the future high priority jobs. The scheduling strategy adopted by OAR can be monitored too: we assume that this strategy don't change dynamically, depending on the arriving jobs. If this point changes, checking OAR behavior may be relevant, to adapt CiGri to each new strategy.

5.3 Feedback loop in the Cloud context

In previous sections, we saw that control loop can be rep-

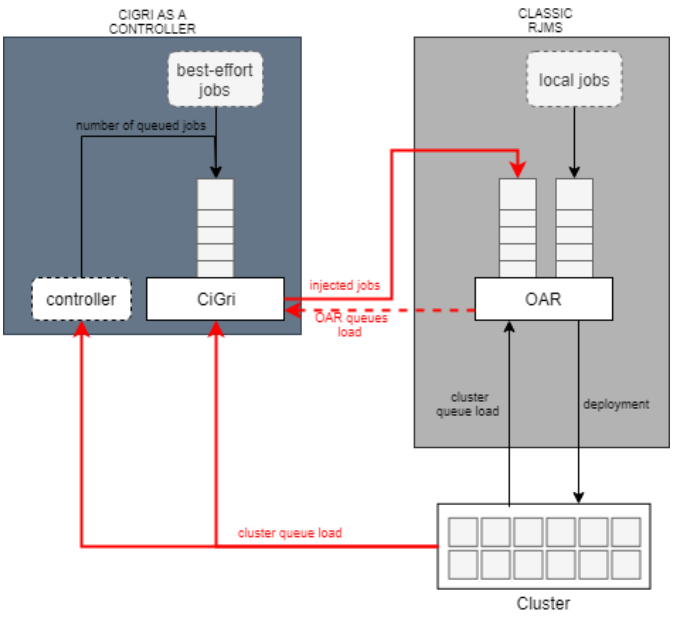

Figure 10: CiGri as an unique controller. Red plain arrows represent the CiGri control loop (even without the added intern controller). The red dotted arrow is additional information. Nothing excludes the possibility of monitoring more information.

resented at the scheduler level and it can be well used in the HPC context. Finding this control mechanism will be quite easy in a Cloud context in that autonomic computing was pretty popular in this kind of infrastructure. However, the previous section showed us two feedback loops that cooperate but stay independents. We will try to show in this section how control loops can cooperate efficiently. Cloud infrastructure benefits from the same RJMS model as HPC infrastructure: A job manager, the scheduler and a resource manager that communicate each other, forming two loops (see Figure 5). To show a possible finer interaction between these two loops, the model described by Hadrien CROUBOIS[7] in 2019 will be adopted. In his paper, the author suggests a modular structure for a fully autonomous workflow manager, with three main modules:

Static Analysis This module is in charge of the offline optimization of workflows. This part is the equivalent of the job manager seen in the RJMS section, apart from the fact that a workflow is a set of tasks with dependencies, and it can be viewed like an unique job. The objective is to precompute meta-data that will later help in the placement and execution of the relative workflows on the shared platform. In this part, all tasks will be grouped, depending on the dependencies and data locality, to optimize the execution.

Dynamic scheduling This module is based on the previous module meta-data, to suggest a scheduling while taking into account system elasticity.

Autonomic platform management This module is in charge of balancing the number of available machines to meet users demand. It will therefore control periodically state of the system and the workflows queue.

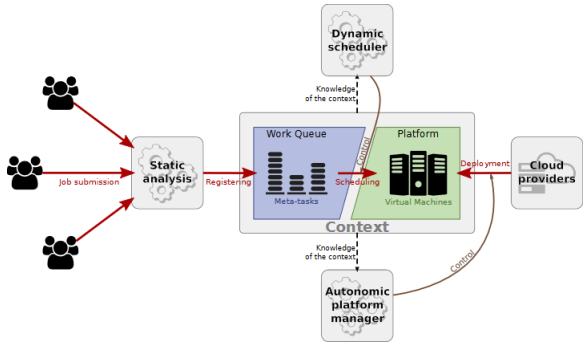

Figure 11: Overview of the system described in the paper[14].

One of the critical points is data-locality, which can be improved by placing tasks in the same location as their data dependencies, or by duplicating tasks that produce large datasets. The challenge here is to find a efficient placement to reduce communication cost, while preserving workflow structure. During the static analysis, tasks are grouped into clusters, correspondingly to their dependencies with data and other tasks: The aim is to execute one cluster in one single machine to achieve good data locality. After the mapping of the clusters onto the nodes has been performed, the nodes will be in charge of the scheduling of the tasks in the clusters they were assigned. We distinguish two kind of schedulers here: The core scheduler which assigns clusters to machine, and the node scheduler (in each machine) which assigns each tasks from the same cluster across all cpu of the said machine. As we said before, each scheduler can be viewed as a control loop.

On top of that, resources must be managed despite the complexity of the work structure. An other control loop is needed to manages all these resources in the highest level (core scheduler level). The estimation of a cluster needed resources is compared to the number of available resources. When a node is unused, it turn off itself. A single loop may be enough, but a finer approach was chosen here: The core scheduler is already aware of the resources state and the workload distribution across the platform. Since it has a global view of the system, information about machines can be reused. Binding the scheduler control loop with the resource management loop seems to be a better strategy. In the other hand, when more resource is needed, we should allocate new nodes, this part is managed by a dedicated control loop.

To keep it simple: the scheduler loop manage the work distribution and shut down useless nodes. The resource manager loop allocates new nodes, according to the information given by the core scheduler. In this way, these two loops share the responsibility of managing the platform resources.

However, this configuration stays complex, and many loops have to collaborate each other, adding more challenges. Rather than distribute decisions across all nodes, maybe a more centralized approach may be more efficient. Here, the decision of activating or not a node is made by two loops. Merging these two loops in one single loop may simplify the overall structure.

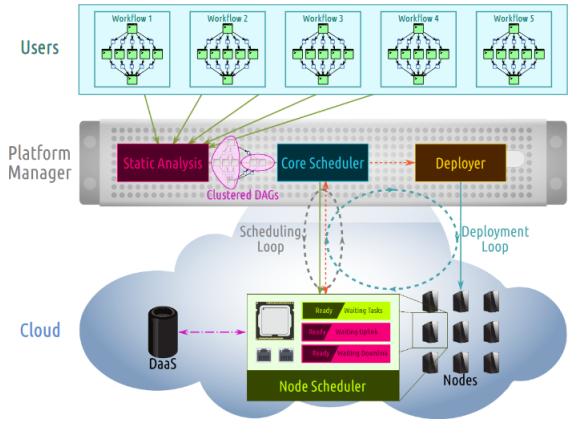

Figure 12: Autonomic scheduling and allocation loops[14].

6. CONCLUSION

In summary, this paper presents some possible explorations between scheduling and autonomic computing field. Although the intersection of these two domains was not really studied, a potential cooperation between these two fields can lead to interesting results. Some scheduling algorithms and RJMS already includes an autonomic approach without formalizing their feedback loop explicitly. Likewise, a scheduling algorithm can be seen as a control law by the autonomic computing community. Formalizing this merge allows a modular approach, quite appreciated in software engineering. Instead of seeing a scheduler as a monolithic constituent of an HPC or Cloud infrastructure, this one is divided on several sections with the system itself on one hand, and its controllers on the other hand. Proposing simpler solutions for scheduling problems adds an other good point at this collaboration. Adding a controller in an existing scheduler will not offer performance guarantees but may nevertheless increase its efficiency, by a constant control of each instance, and some adjustments when the scheduling doesn't meet requirements. However, creating multiple loops and trying to make them work together can make the system more complex.

Despite that, scheduling and autonomic community has much to gain by sharing their respective expertise, and this exploration can lead to many perspectives: Which classes of scheduling algorithms can be converted on control loop ? Which properties are required to convert such algorithms to control loops ? Which data can be added during monitoring for a better analysis ? Cohabitation between several loops must be more studied, and the feedback loops authentication in schedulers, started in this work, may be continued in next researches.

Acknowledgements

Since this project was performed during the covid-19 lockdown, a simple *"I want to thank..."* will be not enough to describe the kindness and the patience of Raphaël BLEUSE, who helped me and support me despite this context. He was present and helpful, like no one, and I will definitely recommend him to the futures trainees. His inspiring ideas and valuable discussions led to progress on this research, and his advice concluded my formation in style. I particularly want to honor Eric RUTTEN for his teaching skills. His clear approach, with a hint of humor, only highlight his unquestionable expertise. His freshness was a breath of air in these difficult times, and I want to thank him for making me discover the autonomic computing world with such an appreciable approach. This section will not be complete without mentioning all my colleagues, and particularly Quentin GUILLOTEAU, who shared with me all the vagaries of this internship. Thanks to him, my coffee times (during videoconferencing included) was really qualitative.

References

[1] Ronald Lewis Graham. "Bounds for Certain Multiprocessing Anomalies". In: *Bell System Technical Journal* (1966).

[2] Ronald Lewis Graham, Eugene Leighton Lawler, Jan Karel Lenstra, and Alexander Hendrik George Rinnooy Kan. "Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey". In: *Annals of Discrete Mathematics* 5.2 (1979), pp. 287–326. doi: 10.1016/S0167-5060(08)70356-X.

[3] Christoph Dürr, Sigrid Knust, Damien Prot, and Óscar C. Vásquez. *The Scheduling Zoo*. url: http://schedulingzoo.lip6.fr (visited on 2019-06-07).

[4] Joseph Y.-T. Leung, ed. *Handbook of Scheduling. Algorithms, Models, and Performance Analysis*. Chapman and Hall CRC, Apr. 2004. doi: 10.1201/9780203489802.

[5] David B. Shmoys, Joel Wein, and David P. Williamson. "Scheduling Parallel Machines On-Line". In: *SIAM J. Comput.* 24.6 (Dec. 1995), pp. 1313–1331. doi: 10.1137/S0097539793248317.

[6] Raphaël Bleuse. "Apprehending heterogeneity at (very) large scale. (Appréhender l'hétérogénéitéà (très) grande échelle)". PhD thesis. Grenoble Alpes University, France, 2017. url: https://tel.archives-ouvertes.fr/tel-01722991.

[7] Yiannis Georgiou. "Contributions for Resource and Job Management in High Performance Computing. (Contributions à la Gestion de Ressources et de Tâches pour le Calcul de Haute Performance)". PhD thesis. Grenoble Alpes University, France, 2010. url: https://tel.archives-ouvertes.fr/tel-01499598.

[8] Albert Reuther et al. "Scalable system scheduling for HPC and big data". In: *J. Parallel Distributed Comput.* 111 (2018), pp. 76–92.

[9] Jeffrey O. Kephart and David M. Chess. "The Vision of Autonomic Computing". In: *IEEE Computer* 36.1 (2003), pp. 41–50. doi: 10.1109/MC.2003.1160055.

[10] Éric Rutten, Nicolas Marchand, and Daniel Simon. "Feedback Control as MAPE-K Loop in Autonomic Computing". In: *Software Engineering for Self-Adaptive Systems III. Assurances - International Seminar, Dagstuhl Castle, Germany, December 15-19, 2013, Revised Selected and Invited Papers*. Ed. by Rogério de Lemos, David Garlan, Carlo Ghezzi, and Holger Giese. Vol. 9640. Lecture Notes in Computer Science. Springer, 2013, pp. 349–373. doi: 10.1007/978-3-319-74183-3_12.

[11] Daniel Balouek et al. "Adding Virtualization Capabilities to the Grid'5000 Testbed". In: *Cloud Computing and Services Science*. Ed. by Ivan I. Ivanov, Marten van Sinderen, Frank Leymann, and Tony Shan. Vol. 367. Communications in Computer and Information Science. Springer International Publishing, 2013, pp. 3–20. doi: 10.1007/978-3-319-04519-1_1.

[12] N. Capit, G. Da Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounie, P. Neyron, and O. Richard. "A batch scheduler with high level components". In: *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.* Vol. 2. 2005, 776–783 Vol. 2.

[13] Emmanuel Stahl, Agustín Gabriel Yabo, Olivier Richard, Bruno Bzeznik, Bogdan Robu, and Éric Rutten. "Towards a control-theory approach for minimizing unused grid resources". In: *Proceedings of the 1st International Workshop on Autonomous Infrastructure for Science, AI-Science@HPDC 2018, Tempe, AZ, USA, June 11, 2018*. ACM, 2018, 4:1–4:8. doi: 10.1145/3217197.3217201. url: https://doi.org/10.1145/3217197.3217201.

[14] Hadrien Croubois. "Toward an autonomic engine for scientific workflows and elastic Cloud infrastructure. (Etude et conception d'un système de gestion de workflow autonomique)". PhD thesis. University of Lyon, France, 2018. url: https://tel.archives-ouvertes.fr/tel-01988995.