Adaptive Execution Planning in Workflow Management Systems

Marta Jaros
3rd year, full-time study
Supervisor: Jiri Jaros

Centre of Excellence IT4Innovations, Faculty of Information Technology, Brno University of Technology
Bozetechova 1/2, 612 66 Brno, Czech Republic
martajaros@fit.vutbr.cz

***Abstract*—Workflow management systems try to move grid, cloud and high performance computing (HPC) services closer to scientific and industrial community by providing a user-friendly interface enabling definition of complex workflows. Workflows provide a formal way to define and automate multi-step procedures reflecting real-world phenomena. However, this still places demands on users to decide how to execute particular tasks in workflows. k-Dispatch, a platform providing automated tasks execution, planning and monitoring, focuses on selected workflows from medical environment. For security reasons, only in-house code binaries tuned for specific HPC resources are used. k-Dispatch screens out users from the complexity of HPC systems. This paper describes how the presented framework deals with the task execution planning. Static planning using default execution parameters may not be sufficient for the effective execution due to time and cost constraints. Adaptive planning discussed in this paper may improve this process.**

***Keywords*—workflow management system, automated execution planning, adaptive planning, job scheduling simulator**

I. Introduction

Workflow management systems enable users to use grid, cloud and high performance computing (HPC) services easily. These systems (see [1] for related state-of-the-art) describe complex problems using workflows. Workflows may be presented as directed weighted graphs providing a formal way to define and automate multi-step computational procedures. The graph nodes present individual tasks that may differ in their nature as well as computational demands. The nodes also encapsulate lower level details about the task specific parameters. HPC environments, however, are highly dynamic and heterogeneous. Thus, efficient manual task execution, its tuning to the specific HPC, monitoring and dealing with various types of failures is very challenging even for expert users. Since execution planning is very time consuming, the task execution throughput may be very limited.

The presented framework called *k-Dispatch* [1], [2] is trying to respond to this problem. *k-Dispatch* mainly focuses on computational problems related to medical environment and uses only predefined workflows. Although workflow structures are predefined, they have a level of adaptivity based on the provided input data. *k-Dispatch* can be extended to support other workflows, but due to security reasons, only a system administrator can implement them. The end users only provide input data and workflow specific parameters.

allocation	The total amount of resources on a given HPC assigned to a particular recipient. In k-Dispatch: It is a database record holding a pointer to the group of users (owing the allocation), a pointer to the HPC where the allocation is active, core hours assigned to the allocation and core hours left, cost per core hour, allocation state, and dates of the start and expiry.
code type	Generic name for the calculated task. There may exist several binaries tuned for different HPCs and queues. The binary may affect the execution time and cost.
queue	HPC queue. It is an ordered set of tasks. Queue performs an access to a collection of computing nodes (i.e. a set of specific hardware cores). Queues differ in hardware resources, priority definitions, available number of nodes, wall-clock time, etc.

TABLE I: Term definitions.

The users are offered to use a user-friendly interface. *k-Dispatch* provides a *"run and forget"* approach where the users are completely cut off from the complexity of current HPC systems.

The following text operates with specialized terms which explanation can be found in Table I.

To be able to plan the workflow execution, required code types (e.g., ultrasonic simulation, thermal simulation, etc.) are hard-coded in the structure of each supported workflow. There may be available several binaries for each code type, accessible only on some HPCs, selected queues and hardware architectures which they were tuned to. Moreover, the cost factor influencing the final computational cost may vary. All mentioned information is stored in the *k-Dispatch* database.

Currently, *k-Dispatch* allows a kind of static execution planning, similar to many related tools [1], where tasks in a given workflow are assigned default binaries and execution parameters. This selection, however, does not reflect suitability of different binaries for a given problem instance (i.e. the code type), computational cost, input data size or actual utilization of the cluster. Static planning may cause ineffective tasks execution, i.e. long waiting times in queues and long execution times.

However, the goal of my PhD thesis is to plan the workflow execution more effectively to (1) satisfy the time-constrained result delivery requirements, and (2) minimize the computational cost. Thus, adaptive execution planning may strongly improve the current solution and better utilize the current HPC system as well. The design of this new approach is described in this paper along with the goals and open problems.

II. Thesis Objectives

The hypothesis, open problems and thesis goals have been described in detail in [1]. Since the last year, the platform prototype has been implemented and tested using a set of unit tests locally, and on IT4Innovations' clusters. Currently, a decision making process for selecting an appropriate execution configuration for each task is being implemented. This process is a base for the adaptive execution planning.

Briefly, the hypothesis of my PhD thesis says *"Contemporary complex HPC systems do not enable common end users an easy and efficient use without having deep and proper knowledge. An appropriate interface and automated simulation planning is supposed to (a) increase the processing efficiency, and (b) save resources, reduce the price of calculation or decrease computational time."*.

A short summary of the thesis goals is given below.

1) **Create a prototype of the software providing planning, executing and monitoring.**
2) *Investigate convenient heuristics and description formalisms.*
3) **Support heterogeneous architectures (CPUs, GPUs, other accelerators) within the same cluster.**
4) *Implement a logic responsible for choosing the most favorable computational machine and an optimal task run configuration.*
5) Provide an effective workflow planning with the latency minimalization or the throughput maximalization.
6) Test this software on two selected medical applications.
7) Evaluation of the benefits and limits.

Thesis goals 1) and 3) have been fulfilled while goal 2) has been extended for the decision making Algorithm 1. The future work focuses on the run configuration selection techniques and their evaluation (goal 4). This is discussed more in detail in sections III and V.

Here, the features being in progress now are listed:

- Hard-coded simple workflows reflecting a single program type only. This will provide us with enough scaling (performance) data for the decision making process of the run configuration selection.
- Implementation of the method returning the number of actually free nodes in the cluster. Cluster utilization monitoring may strongly improve the decision making process.

Here, a list of new features or improvements follows:

- Remote tasks monitoring and detection of faulty tasks on the HPC site finished. Faulty tasks and their dependencies are resubmitted.
- Fundamental accounting policy implemented.
- Task graphs generation and evaluation reimplemented. This is discussed more in detail in section III.
- A support of logic responsible for the allocation (remote machine) and tasks' run configuration selection. For more details, see section III.
- Heterogeneous architectures support (CPUs, GPUs, other accelerators) within the same cluster.
- Two dummy workflows containing tasks that can be executed concurrently, and tasks with dependencies created. One of the workflows runs a simple Python program with a default run configuration (default allocation, computational queue, number of cores, etc.). The second one receives a real data set and runs C, Python and Matlab codes. The default run configuration for each task is selected as well. Failures to these workflows were injected and the ability to deal with them was successfully tested.

III. Implementation

This section discusses the decision making process in a task run configuration, points out open problems and shows ideas how to deal with them.

Currently, *k-Dispatch* allows the static task planning only. For each task in the workflow, a default binary for the specific code type and its default run configuration are selected. This approach enables users to run their workflows without any advanced knowledge of the current HPC service, however, the tasks may not be executed efficiently and may spend a long time by waiting in queues (i.e. waiting for free time slots and computational resources). Inappropriate execution parameters influence the code scaling and the execution time.

The solution is to implement an adaptive execution planning and a decision making process that will choose an appropriate run configuration for each task based on collected historical performance data and current cluster utilization.

The first attempt is to implement a one-pass decision making process. The run configuration will be optimized for each task independently. However, the implementation of a multi-pass process in the future may better optimize run configurations for the whole workflow.

Following constraints shall be considered:

- Execution planning is time-constrained since the HPC environment is highly dynamic and changes very quickly. To reduce the dynamism, dedicated resources may be used, however, the computational cost may be higher.
- In order to provide a quality of service, workflow calculations are time-constrained. Thus, it is very undesirable to have tasks waiting in queues very long due to unsuitable run configurations.

Let's consider to have collected performance data for each code type. Available code types and binaries correspond to predefined use-cases (see Figure 1). To give an idea what affects the execution time of our simulations from the input data point of view, here's a list of items: input data size, number of time steps, length of a time step, medium properties, used binary and cluster, and a number of nodes.

Algorithm 1: Adaptive execution planning algorithm

```
Presumptions:
1 Consider a set of allocations $A^+ \subseteq A$ the user can use.
2 All possible binary executables for $a \in A^+$ are defined
    as $D \in (B_1, B_2, \ldots, B_n)$, where $n$ is the number of
    code types within the workflow.
    $B_i = \{b_1, b_2, \ldots, b_m\}$ is a set of available binaries
    for a given code type. $B_i$ may be an empty set.
3 $p$ is a price function returning the aggregated
    computational cost of the workflow.
    $p : G \times C \times D \to \mathbb{R}^+$.
4 $t$ is a time function returning the aggregated execution
    time of the workflow. $t : G \times C \times D \to \mathbb{R}^+$.
5 Workflow evaluation is defined as $\mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$
    and may be calculated using the formula
    $f = \alpha \cdot p + (1 - \alpha) \cdot t$ where $\alpha$ is a selectable ratio
    prioritizing the minimal computational cost or the
    execution time.
6 Best evaluated workflow is given by
    $\mathrm{argmin}_{(c \in C, d \in D)} f$.
Algorithm :
1 Create a workflow $G = (V, E)$ from the workflow
    template and input data. $V$ is a set of tasks and
    $E \subseteq V \times V$ is a set of task dependencies.
2 Select candidate allocations $C = \{c \in A^+ \mid$
    $c.status == active \wedge c.hours\_left > 0.0\}$
3 Generate and evaluate workflows for all combinations
    of candidate allocations $C$ and binary executables $D$.
```

Since collected performance data may be sparse and incomplete (e.g. missing measurements for particular input data sizes), the nearest suitable record will be selected and used for the execution planning. The records should also be filtered by their age using moving windows. It is not very convenient to use records older than three months, for example. Later, interpolation techniques, such as a polynomial interpolation, or machine learning methods may be used on collected performance data.

The wall-clock time determines the maximum time the task can be assigned to remote computational resources. This time is determined by the execution time obtained from the historical performance data, selected number of cores and a load balancing constant (e.g. 1.2). This constant is expected to balance small fluctuations in the code and cluster performance.

A couple of ideas may be implemented in the development:

- An ability to change a run configuration of already submitted tasks (that are waiting in queues) according to actual cluster utilization.
- Change slightly the run configuration of a few tasks of the same kind to bring a bit of diversity into the collected data and enable to explore better solutions.

In other words, the goal of the discussed decision making process is to find a suitable allocation and execution configuration in order to minimize computational cost of the calculated workflow while meeting given time constraints. The idea is described in Algorithm 1.

IV. Testing

While working on the adaptive execution planning process, a question about the evaluation of the selected run configuration appeared. Due to many reasons such as the cost of resources, the reliability, the varying background load or the dynamic cluster behavior, the experimental evaluation cannot be mostly performed on the real systems. Moreover, to obtain reliable results, multiple workflows with various run configurations need to be performed using the same and controllable conditions that simulate different real-life scenarios which is, however, often unreachable. For this purpose, a short research in the latest job scheduling simulators emulating real HPC environments has been done (see section V).

The simulator will be used to evaluate the run configuration at first. We can simulate both, static planning using default binaries and settings, and various versions of adaptive planning. This should give a hint whether the proposed strategy may be useful in real systems. For this purpose, HPC workload traces need to be acquired.

Subsequently, the testing will be provided on real HPC systems. In this step, the calculations will be observed for unexpected situations.

Firstly, testing will be performed on smaller workflows consisting of one or two tasks. After this being tuned, the algorithm will be tested against bigger and real-world workflows (see Figure 1).

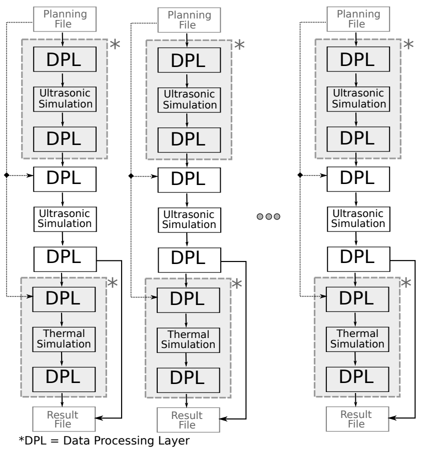
V. Related Work

An extensive research in the workflows management systems and HPC schedulers was introduced in [1]. Since the need to test the selection of run configurations outside the real cluster environment emerged, a brief research in cluster scheduling simulators follows here.

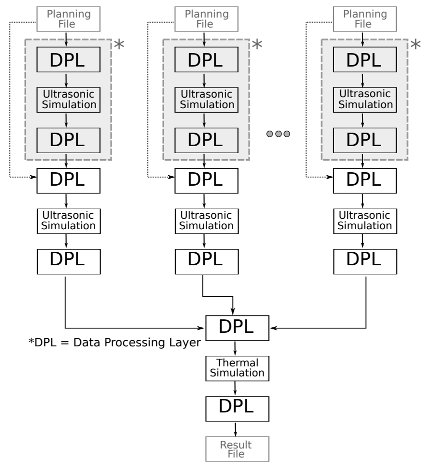
Simple job scheduler simulators often provide a detail model of the queuing behavior as the jobs arrive at the system upon submission, wait for available resources, start their execution, and eventually leave the system upon their completion. For example, PySS[1] is a trace-driven scheduler simulator. It implements a couple of scheduling algorithms, including several backfilling ones. The problem with simple simulators is that they do not really model the target HPC system or the runtime behavior of the applications. PySS takes the job runtime directly from the job trace, although in reality a job's runtime should be affected by the specific resources allocated to the job and by the application's runtime behavior, which can be affected by other jobs running simultaneously [3]. Thus, more sophisticated simulators need to be used instead.

Alea 4 [4] is an event-based, the GridSim toolkit [5] based, grid and cluster scheduling simulator. The simulator is able to deal with common problems related to the job scheduling

[1]https://code.google.com/archive/p/pyss/



(a) The Neurostimulation workflow.



(b) The HIFU (High Intensity Focused Ultrasound) workflow.

Fig. 1: Figures show generic templates of two different medical applications. According to the planning file, the specific structure of the workflow is created. This structure is planned and executed. Gray blocks marked with a star may be replicated to extend the fundamental workflow structure. DPL is a data processing layer.

like the heterogeneity of jobs, resources, and the dynamic runtime changes such as the arrivals of new jobs or the resource failures and restarts. The main part of the simulator is a complex scheduler which incorporates several common scheduling algorithms working either on the queue or the schedule (plan) based principle. The latest version of Alea uses a dynamic workload adjustment technique enabling to model user-to-system interactions properly. The input is still a static workload (historical workload traces extracted from the HPC itself, or from a public workload trace repository) but transformed into the dynamic one afterwards.

Performance Prediction Toolkit (PPT) [3] is a full-scale HPC simulator.It can use synthetic workload models or adopt job traces from existing HPC workload archives. The simulator implements a couple of commonly used scheduling algorithms, however, it does not include backfilling algorithms.

Other complex frameworks for studying grids, clouds, HPC or peer-to-peer systems have been developed. However, majority of these projects seem to be inactive or abandoned. [4]

VI. Conclusions

k-Dispatch is a workflow manager, developed in Python, providing workflows automated execution planning and monitoring. It completely screens out users from the complexity of current HPC systems. Presented version of *k-Dispatch* enables users to execute predefined workflows only, using fine tuned codes for available HPCs. The users provide *k-Dispatch* with input data only. *k-Dispatch*'s current state and a progress in development within the last year has been presented in this paper. Actually addressed problems have been described and ideas how to deal with them presented.

Since the last year, thesis goals 1) and 3) have been satisfied. The goal 2) has been extended for the algorithm describing the decision making process in adaptive execution planning which introduces a novel approach in related tools. At this time, the goal 4) is of the highest priority. [1]

Next steps in the development are to (1) implement simple workflows to collect scaling data for various code types, (2) implement the presented logic to select the run configuration, (3) study the selected job scheduling simulator (Alea 4), (4) evaluate the implemented logic and its run configuration selections on both, simple and real-world workflows, and finally (5) execute tested workflows in a real HPC environment.

Acknowledgment

This work was supported by the FIT-S-17-3994 Advanced parallel and embedded computer systems project. This work was supported by The Ministry of Education, Youth and Sports from the National Programme of Sustainability (NPU II) project IT4Innovations excellence in science - LQ1602 and by the IT4Innovations infrastructure which is supported from the Large Infrastructures for Research, Experimental Development and Innovations project IT4Innovations National Supercomputing Center - LM2015070.

References

[1] M. Jaroš, "Scientific workflows management," in *Počítačové architektúry & diagnostika PAD 2018*. University of West Bohemia in Pilsen, 2018, pp. 25–28.

[2] M. Čudová, "Framework for planning, running and monitoring cooperating computations," in *Počítačové architektúry & diagnostika PAD 2017*. Slovak University of Technology in Bratislava, 2017, pp. 20–23.

[3] M. A. Obaida and J. Liu, "Simulation of HPC job scheduling and large-scale parallel workloads," in *2017 Winter Simulation Conference (WSC)*. IEEE, dec 2017, pp. 920–931.

[4] D. Klusacek, S. Toth, and G. Podolnikova, "Complex Job Scheduling Simulations with Alea 4," *CEUR Workshop Proceedings*, vol. 1828, pp. 53–59, 2017.

[5] R. Buyya and M. Murshed, "GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing," *CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE Concurrency Computat.: Pract. Exper*, vol. 14, pp. 1175–1220, 2002.