BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2026
http://excel.fit.vutbr.cz

System for Orchestrating and Monitoring Jobs in Personalized Medicine

Oleksandr Musiichuk*

Abstract
Executing complex medical simulation workflows on HPC clusters is challenging due to dynamic resource constraints and strict job dependency requirements. We present a modular orchestration system,called k-Dispatch that automatically assigns execution configurations, such as wall-clock time, number of nodes, nunber of CPU cores to jobs in a workflow(Directed Acyclic Graph) and submits them to a Slurm-based HPC scheduler with an internal queue mechanism for handling capacity limits. The system successfully executes multi-step simulation workflows, correctly manages sequential and parallel job chains, and handles partial submission scenarios where HPC capacity is limited. This work provides a maintainable and extensible foundation for automated workflow execution in the k-Dispatch platform for personalized ultrasound-based medicine.

*xmusii00@vut.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Personalized medicine refers to an approach in which treatment is tailored to the individual patient so that it is applied at the right place and at the right time. Instead of relying on one-size-fits-all protocols, therapies are carefully planned using patient-specific data to maximize effectiveness and safety. Ultrasound-based treatments are one example of this paradigm, including applications such as neurostimulation or targeted tumor ablation.

From a computational perspective, such treatment planning relies on multi-step workflows that can be modeled as directed acyclic graphs (DAGs) [1], where nodes represent individual computational tasks and edges capture dependencies between them.
These workflows are typically executed on high-performance computing (HPC) systems due to their complexity and computational demands **Fig.1**.

k-Dispatch[2] automatically orchestrates ultrasound-based treatment planning. The system takes a planning input file, decodes it, and determines which simulations need to be performed. Based on this analysis, it estimates computational requirements and assigns appropriate parameters to individual tasks. These tasks are then translated into executable scripts, offloaded to remote HPC resources, and executed **Fig.2**.

During execution, the workflow is continuously monitored to ensure correctness and efficiency. After completion, results are delivered back to the user, followed by accounting and reporting procedures. This end-to-end automation enables efficient and scalable execution of complex personalized treatment workflows.

This work focuses on the orchestration layer of k-Dispatch. The main challenge lies in transforming high-level workflow inputs into an executable representation: decoding the planning specification, assigning appropriate binaries and runtime parameters to individual tasks, selecting suitable execution environments, and preserving task dependencies inherent to the underlying DAG. All of these steps must be performed while respecting the constraints of HPC systems, such as limited capacity, resource availability, and scheduling policies. General-purpose workflow systems (such as Pegasus, Taverna, HyperLoom) exist [3, 4, 5], but they are not tightly integrated with domain-specific logic and database models used in k-Dispatch.

The main contribution of this work is a redesigned orchestration pipeline for k-Dispatch, built around three key components: the DispatchAndTransfer control module, a modular Orchestrator, and an extensible Optimizer responsible for selecting execution parameters. This design improves flexibility, enables clearer separation of concerns, and allows the system to adapt

more easily to different workflow requirements and HPC environments.

2. Workflow Evaluation

The system transforms a planning file into a directed acyclic graph (DAG) of dependent computational tasks. The `DispatchAndTransfer` module serves as the main control component: it detects new workflows, invokes decoding of the input into a dependency graph, delegates graph evaluation, submits jobs to HPC, monitors their state, updates the database, and retrieves results after completion.

The Orchestrator assigns execution parameters to tasks in the graph to satisfy various optimization criteria, such as task throughput, execution time, and computational cost **Fig.4**. For each task, it assigns an executable binary, estimates execution requirements, selects a target execution environment, and determines scheduling parameters such as computational queue, number of cores, and wall-clock time. This makes the orchestration layer adaptable to different workflow types and heterogeneous HPC resources.

To keep the decision logic extensible, the Orchestrator delegates evaluation of execution options to a dedicated `Optimizer`. The Optimizer computes a score for candidate configurations using a postfix scoring expression combining criteria such as execution time and computational cost. The best-scoring configuration is selected for execution. Resource information is accessed through a `RemoteMachine` abstraction, which provides a unified interface to available HPC environments.

However, the optimization process relies on performance data collected from previously executed tasks. Therefore, optimization strategies cannot be directly applied to entirely new tasks without historical data. This data-driven approach represents a key difference from general-purpose workflow systems, as k-Dispatch leverages its domain-specific database model and accumulated execution history to improve scheduling decisions over time.

The optimization approach builds on previous work focused on workflow scheduling and parameter tuning [6, 7]. In particular, the use of the HeuristicLab framework is assumed for implementing optimization strategies. At the same time, the Optimizer is designed as a generic module, allowing integration of alternative optimization methods without changes to the core system.

This separation of control, decision-making, and resource abstraction results in a modular architecture that supports extensibility and future integration of additional optimization strategies and execution backends **Fig.5**.

3. Deployment & Testing

The system is deployed as part of the k-Dispatch infrastructure and integrated with an HPC environment using a Slurm-based scheduler. Jobs are generated as execution scripts and submitted to remote clusters via the RemoteMachine interface.

The system was tested on both synthetic and real-world workflow scenarios to validate correct task execution, dependency handling, and orchestration logic.

An important aspect of the system is logging and monitoring. Workflow execution, job states, and scheduling decisions are continuously logged, enabling debugging and system analysis. These logs are integrated with Grafana[8], which provides real-time visualization of system behavior, job execution, and resource utilization.

This enables continuous monitoring of the system and supports analysis of workflow execution.

This deployment and testing approach ensures that the system is robust, observable, and ready for further extension.

4. Results and Conclusion

The proposed solution was validated on the `sc-gpu` HPC cluster at FIT BUT using a synthetic three-step chain workflow. The experiments confirmed correct dependency-aware execution, partial submission when only limited HPC capacity was available, correct propagation of remote job states to the database, and accurate computation of billed core-hours.

The results show that the redesigned orchestration layer can reliably automate execution of medical workflows under realistic HPC constraints. The proposed architecture replaces the legacy dispatcher with a more modular and extensible solution and creates space for future optimization strategies, including advanced runtime prediction and broader support for cloud environments.

Acknowledgements

I would like to thank my supervisor Marta Jaroš for guidance and for providing access to the k-Dispatch codebase and HPC infrastructure at FIT BUT.

References

[1] Yves Robert. *Task Graph Scheduling*, pages 2013–2025. Springer US, Boston, MA, 2011.

[2] Marta Jaroš, Bradley Treeby, Panayiotis Georgiou, and Jiří Jaroš. k-dispatch: A workflow management system for the automated execution of biomedical ultrasound simulations on remote computing resources. In *Proceedings of the Platform for Advanced Scientific Computing Conference, PASC 2020*, pages 1–10, New York, 2020. Association for Computing Machinery.

[3] Ewa Deelman, Karan Vahi, Mats Rynge, Rajiv Mayani, {Rafael Ferreira} {Da Silva}, George Papadimitriou, and Miron Livny. The evolution of the pegasus workflow management software. *Computing in Science and Engineering*, 21(4):22–36, July 2019. Publisher Copyright: © 1999-2011 IEEE.

[4] Katherine Wolstencroft, Robert Haines, Donal Fellows, Alan Williams, David Withers, Stuart Owen, Stian Soiland-Reyes, Ian Dunlop, Aleksandra Nenadic, Paul Fisher, Jiten Bhagat, Khalid Belhajjame, Finn Bacall, Alex Hardisty, Abraham Nieva de la Hidalga, Maria P. Balcazar Vargas, Shoaib Sufi, and Carole Goble. The taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud. *Nucleic Acids Research*, 41(W1):W557–W561, 07 2013.

[5] Vojtěch Cima, Stanislav Böhm, Jan Martinovic, Jiří Dvorský, Kateřina Janurová, Tom Vander Aa, Thomas Ashby, and Vladimir Chupakhin. Hyperloom: A platform for defining and executing scientific pipelines in distributed environments. pages 1–6, 01 2018.

[6] Tomáš Sasák. Optimization of run configurations of k-wave jobs, 2020.

[7] Martin Buchta. Estimation of algorithm execution time using machine learning. Master's thesis, Brno University of Technology, Faculty of Information Technology, Brno, 2023.

[8] Grafana. https://grafana.com. Accessed: 2026.