DOI: 10.14529/jsfi260203

Multi-Agent Task Flow Dispatching in a Heterogeneous Shared Supercomputer Center

Igor A. Kaliaev[1], *Anatoly I. Kaliaev*[1], *Sergei A. Semenistyi*[1]

The slowdown in performance growth of general-purpose processors is driving the adoption of specialized accelerators such as GPUs and FPGAs in shared supercomputer centers (SSCs). FPGAs are attractive due to hardware-level parallelism and energy efficiency, but their integration significantly increases system heterogeneity: even with similar hardware, differences in bitstreams make nodes functionally nonequivalent. For streams of short jobs, frequent reconfiguration causes substantial overhead, while limited configuration-memory endurance constrains the number of configuration cycles. Classical HPC schedulers usually ignore current FPGA configurations and reconfiguration costs, leading to suboptimal task placement and reduced throughput. This paper presents a multi-agent task dispatcher for a heterogeneous SSC that explicitly accounts for FPGA configuration state and reconfiguration latency. Software agents on each FPGA node make local decisions on task admission and configuration changes, coordinating via bulletin-board queues. The model incorporates computation time, data transfer, and reconfiguration overhead into the scheduling objective. A prototype was implemented on a cluster with up to 18 Tertius2T reconfigurable units and two server nodes. Experiments with queues of up to 3500 tasks show that the multi-agent dispatcher reduces average task completion time and keeps placement and reconfiguration overheads within 40–1200 ms despite individual FPGA configuration times of at least 13 s, demonstrating resilience to heterogeneity and improved resource utilization.

Keywords: high performance computing, hybrid computing systems, multi-agent scheduler, reconfigurable systems, DAG scheduling.

Introduction

High-performance computing clusters (HPCs) are a key tool for solving problems characterized by high computational complexity, the need for accurate numerical modeling, and large volumes of floating-point operations. Traditional HPCs are dominated by architectures based on CPUs and GPUs. However, increasing demands for performance, energy efficiency, and data processing flexibility are leading to the proliferation of scalable heterogeneous architectures incorporating specialized hardware accelerators, including FPGAs [12]. FPGAs enable the implementation of computing cores directly in hardware, providing massive parallelism, low latency, and the ability to adapt the architecture to a specific task. This makes them attractive for high-performance computing, data stream processing, and the implementation of specialized machine learning algorithms. At the same time, the inclusion of FPGAs in large shared supercomputer centers (SSCs) raises new challenges in resource management and dispatching.

A distinctive feature of FPGA operation in HPCs is the presence of a heterogeneous set of configurations used by various users and applications. Switching between configurations requires a configuration procedure, which can be significant and comparable in time to task execution [15]. For a stream of short tasks, frequent reconfigurations lead to significant overhead and reduced utilization of the computing infrastructure. Additionally, the limited resource of configuration cycles when using non-volatile memory must be taken into account.

Classical task management systems in HPCs are typically focused on CPU/GPU nodes and do not take into account the dynamics of FPGA configurations or the resource of the number

[1]Scientific Research Institute of Multiprocessor Computing Systems of the Southern Federal University, Taganrog, Russian Federation

of reconfigurations. This leads to inefficient task distribution, increased latency, and reduced system throughput.

Team management systems in HPCs are traditionally based on centralized or hierarchical schedulers implemented within software packages such as SLURM, PBS, LSF, and others. Such schedulers take into account priorities, resource constraints, and allocation policies, but are primarily designed for homogeneous or slightly heterogeneous CPU/GPU node configurations. Support for FPGA specificity is typically limited to a rough estimate of the presence or absence of the corresponding resource on a node [16].

In recent years, approaches to scheduling in heterogeneous clusters have been actively developing using heuristic and metaheuristic algorithms, as well as machine learning methods. A number of studies consider models that account for the heterogeneity of computing devices and performance differences for different types of tasks [1]. However, the problem of explicitly accounting for FPGA reconfiguration time and resources during stream processing remains less well-developed.

A separate area of research is related to the application of multi-agent technologies to the management of distributed computing systems. Agent-based approaches enable decentralized decision making, increased fault tolerance, and reduced load on the central dispatcher. In the context of heterogeneous FPGA systems, this opens the possibility of locally accounting for the current state of configurations and predicting future reconfigurations at the level of individual nodes. The approach proposed in this paper combines the ideas of multi-agent control and a specialized model of reconfigurable nodes, which allows not only to distribute tasks between nodes, but also to optimize the sequence of FPGA configurations in order to reduce overhead costs.

The article is organized as follows. Section 1 introduces the problem of task dispatching in heterogeneous shared supercomputer centers with FPGA-based nodes and outlines the motivation for adopting a multi-agent approach. Section 2 presents the computing environment model and formulates the task dispatching problem with explicit consideration of computation, communication, and reconfiguration costs. Section 3 describes the proposed multi-agent dispatching system for a cluster based on Tertius-2T reconfigurable computing units, including its architecture, agent functionality, and coordination mechanisms. Section 4 reports the results of experimental studies and evaluates the effectiveness of the proposed approach. Section 5 discusses the results and the limitations of the proposed method. Conclusion summarizes the study and points directions for further work.

1. Review of Approaches to Task Dispatching in Heterogeneous High-Performance Computing Clusters with Reconfigurable Accelerators

1.1. Problem Statement

The task of job dispatching in high-performance computing clusters (HPCs) involves distributing a set of jobs across multiple computing nodes to minimize certain target metrics: make-span, average latency, network infrastructure load, and energy consumption. In the case of heterogeneous clusters incorporating FPGA-based reconfigurable computing units (RCUs), the task becomes significantly more complex: the functional equivalence of nodes is violated

due to configuration differences, and the job execution time includes not only the computational component but also the reconfiguration cost.

1.2. Traditional Job Management Systems in HPCs

Job management in HPCs has historically developed within the framework of centralized batch schedulers. The most common of these are SLURM (Simple Linux Utility for Resource Management), PBS/Torque, LSF, and the Microsoft HPC platform. These systems implement a rich set of queuing, prioritization, resource reservation, and fault-tolerance policies. In particular, SLURM supports backfill scheduling and heterogeneous jobs (hetjobs), which allow for the allocation of different types of resources to individual task components [13].

However, the standard capabilities of such schedulers when applied to FPGA accelerators are significantly limited. In its basic configuration, SLURM does not support the selection of nodes with FPGA resources; this requires the development of specialized extensions. A more fundamental limitation is that classic schedulers abstract from the internal state of accelerators: they do not track the current FPGA configuration, do not consider the cost of configuration switching when assigning tasks, and do not limit the number of configuration cycles based on the resource of non-volatile memory. In conditions where the configuration time of a single FPGA node can be tens of seconds, ignoring this factor leads to significant performance losses when processing queues of short, heterogeneous tasks.

Among domestic developments, the TaskMaster HPC system, developed at the National Research University Higher School of Economics for the cHARISMa supercomputer, is noteworthy [8]. The system monitors task performance, identifies incorrectly launched calculations, and recommends optimizations. According to the developers, the implementation of the system increased the supercomputer's effective performance by 20.5% in the first half of 2023. However, this system is focused on a CPU/GPU cluster and does not cover the specifics of managing reconfigurable resources.

1.3. Task Scheduling Algorithms in Heterogeneous Systems

1.3.1. Classical heuristics

Among the task scheduling algorithms for heterogeneous systems, the HEFT (Heterogeneous Earliest Finish Time) algorithm occupies a special place. This algorithm ranks tasks from bottom to top based on computational and data transfer cost estimates, and then assigns each task to the resource that provides the earliest completion time. HEFT has low computational complexity and produces good results in practice; however, it does not take resource dynamics into account and does not implement load balancing. Subsequently, improvements to HEFT were proposed, including the lookahead-HEFT algorithm, which takes into account the impact of locally made decisions on the descendants of a task in the graph, and experiential HEFT for cloud environments, which takes into account historical data on the effectiveness of assignments [3].

1.3.2. Accounting for FPGA reconfiguration overhead

In the context of systems with reconfigurable accelerators, a key challenge is optimizing configuration switches. Complete FPGA reconfiguration involves loading a new bitstream for the entire circuit, which is associated with a significant time cost. An alternative is dynamic

partial reconfiguration (DPR), which allows changing only selected regions of the die while the rest of the circuit is running. DPR provides significant acceleration of configuration switches and opens up opportunities for multitasking on a single FPGA chip, but requires significant engineering effort during design and is not applicable to all classes of tasks [11].

In the work of Rodriguez-Canal et al. an approach to preemptive task scheduling on FPGAs using DPR is proposed, allowing for the simultaneous execution of multiple cores in different reconfigurable regions and the preemption of lower-priority tasks. The authors show that the worst-case overhead of their approach does not exceed 10%, while the performance gain compared to full reconfiguration is at least 24%. In the broader context of real-time FPGA systems, the FRED architecture is proposed, which uses ticket queues to control access to reconfigurable regions and eliminate task contention.

The problem of integrated optimization of task scheduling, partitioning, and placement on FPGAs with DPR is considered as a joint problem in a number of studies. The proposed methods typically rely on Integer Linear Programming (ILP) or heuristic algorithms and are aimed at minimizing the total execution time, taking into account the reconfiguration overhead. In this paper, a specialized task scheduling algorithm is proposed for reconfigurable computing systems, taking into account both communication dependencies and the cost of switching configurations [2, 5].

It is important to note that most of the listed approaches consider a single FPGA resource or a small group of them, while the problem of scheduling task flows across a large heterogeneous cluster of dozens of FPGA nodes with different configuration states and limited configuration memory resources remains significantly less studied.

1.3.3. Configuration memory lifespan

Non-volatile memory (NVM) used to store FPGA configuration bitstreams has a limited lifespan in terms of write/erase cycles. For flash memory, this figure is typically around 100,000 Program/Erase cycles, while for some NVM types (PCM, RRAM) it ranges from 10^6 to 10^9 cycles. With intensive use of various configurations in a large data center, the memory lifespan can be exhausted significantly faster than the design lifespan without special measures. The problem of wear leveling has been well studied in relation to data memory in file systems and storage systems, but in the context of managing FPGA configuration memory in data centers, specialized solutions are significantly fewer. This is an important argument in favor of scheduling algorithms that specifically reduce the number of reconfigurations [9, 17].

1.4. Multi-Agent Approaches to Computing Resource Management

1.4.1. Conceptual foundations of multi-agent dispatching

Multi-agent systems (MAS) represent a paradigm in which distributed system control is implemented by multiple autonomous software agents interacting with each other and the environment. Each agent has a local state, goals, and behavioral strategy; decision making is decentralized, ensuring scalability and fault tolerance [10]. Applied to HPC, the multi-agent approach eliminates the need for a single central scheduler as a bottleneck and implements dynamic, self-organizing resource management.

Theoretical and methodological foundations of multi-agent scheduling of distributed systems were developed at the Scientific Research Institute of Multiprocessor Computing Systems

of the Southern Federal University (Taganrog), including for GRID computing and heterogeneous distributed systems. The paper [7] formulates formal models of multi-agent dispatching for homogeneous and heterogeneous distributed systems of the first and second types, and develops strategies and algorithms for the behavior of resource agents under conditions of limited inventory, uncertainty, and a competitive environment.

1.4.2. Contract Net Protocol and its extensions

The most widely used mechanism for coordinating agents in task allocation is the Contract Net Protocol (CNP), proposed by Smith in 1980. Within the CNP framework, a manager agent announces a task, contractor agents submit proposals with cost estimates, and the manager selects the winner and concludes the "contract". The protocol allows for a hierarchical subcontracting structure and is similar to a sealed auction mechanism. The use of CNP in MAS enables efficient task allocation without global knowledge of the system state, relying solely on locally available information and consistent messages.

A number of studies aim to improve the efficiency of CNP by modifying bid evaluation strategies, introducing pheromone mechanisms, taking into account the current workload, and varying the number of bidders. Thus, an improved CNP with a limited number of bidders allows for a simultaneous reduction in message traffic and task allocation time while maintaining a high percentage of successfully assigned tasks. A formal analysis of CNP and the conditions for its correct application are discussed in [4, 18].

1.4.3. Blackboard architecture in MAS

A blackboard architecture is an alternative and complementary coordination mechanism in MAS. In this architecture, agents do not exchange messages directly, but read and write data to a shared structure – a blackboard – containing the current state of tasks, resources, and results. This ensures loose coupling between agents, transparency of the system state, and simplifies the addition of new participants. Blackboard architecture is actively used in real-time systems and, more recently, in multi-agent systems [6].

In the context of HPC, the use of a blackboard as a centralized task queue fits well with the decentralized logic of agents: the queue acts as a task "market" in which agents compete or cooperate for assignments while maintaining full control over local resources and configurations.

1.4.4. MAS for HPC: current state

The paper [14] proposes a hierarchical MAS for distributed task scheduling in HPC, combining a global coordinator, cluster brokers, and node-level executor agents. Interaction is based on a hybrid of CNP and auction mechanisms. The authors demonstrate that this scheme provides significantly better resilience to queue instability and resource load fluctuations than traditional centralized schedulers.

A promising approach is also the use of dynamic MAS with adaptive task reassignment when the composition of agents or task parameters change. The DRAMA system offers a unified abstraction of agents and tasks as resource objects with explicit attributes, enabling scheduling based on the affinity of a task to a specific agent – a concept directly applicable to the compatibility of a task with the current FPGA configuration.

1.4.5. Comparative analysis of approaches

Table 1 provides a comparison of the approaches considered based on key characteristics relevant to the dispatching task in a heterogeneous HPC with FPGA.

Table 1. Considered approaches

Approach	Centralization	FPGA Configuration Awareness	Reconfiguration Time Awareness	NVM Resource Awareness	Scalability
SLURM/PBS (basic)	Yes	No	No	No	High
SLURM + FPGA extension	Yes	Partial	No	No	High
HEFT and variations	Centralized offline	No [3]	No	No	Medium
DPR schedulers (FRED, etc.)	Local on-chip	Yes (partial)	Partial [11]	No	Low (one FPGA)
Hierarchical MAS (CNP)	Decentralized	No [14]	No	No	High
Proposed MAS (present work)	Decentralized	Yes	Yes	Yes	High

An analysis of the table reveals that existing approaches either fail to consider the specifics of FPGA nodes at all (traditional schedulers, classic algorithms like HEFT), or only consider configuration at the individual FPGA chip level (DPR schedulers), or fail to explicitly limit the configuration memory resource. The scheduling MAS proposed in this paper is designed to simultaneously consider all of these factors across a cluster of dozens of RDBs.

2. System Models

2.1. Computing Environment Model

HPC can be represented as a set of computing nodes $N = \{n_1, n_2, \ldots, n_m\}$ of various configurations, providing various resources for task execution. The nodes are connected to the communication network via communication channels B_i with varying bandwidths.

The parameters of the computing node n_j include:

- r_j^{CPU} – the number of available processor cores.
- r_j^{RAM} – the amount of available RAM.
- r_j^{SCONF} – static configuration;
- r_j^{DCONF} – dynamic configuration;
- W_j – the node's computing power allocated to all cores.

The static configuration of the compuitation unit (CU) describes parameters that do not change during operation. For example, the presence of a specific GPU, NPU, or FPGA model, connected peripherals, etc. The dynamic configuration of a CU describes the parameters that can be influenced by the corresponding software available on the CU: installed software libraries, loaded data files, FPGA firmware, and settings of specialized hardware used during task execution (e.g., the position of controlled surveillance cameras, power supply mode for autonomous CUs). For each pair of dynamic configurations a and b, the reconfiguration time $T_j^{RECONF}(a, b)$ from state a to state b is known.

For the dispatching system to operate, a number of task-related information is required: the received input data, the task structure (operation graph), the task type, time constraints for its execution, and its metadata – information that allows for task classification when assessing its complexity. Next, we consider the HPC task model.

2.2. Task Model

The vast majority of known works in the field of task scheduling in distributed systems are devoted to optimizing the placement of batch tasks based on their structure, the operations they comprise, and the parameters of the computing system. Given the above considerations, a task for HPC can be described by a tuple $z_i = \langle D_i^{in}, G_i, D_i^{out}, T_i^{MAX}, m_i \rangle$, where D_i^{in} is the input data descriptor, G_i is the directed acyclic graph of the task, D_i^{out} is the output data descriptor, T_i^{MAX} is the required execution time, and m_i is the task metadata (user ID, task ID, etc.).

The task graph $G = (V, E)$ consists of a set of nodes $V = \{v_1, v_2, \ldots, v_n\}$ representing the operations that must be performed by HPC to complete the task and a set of edges $e_{i,j} \in E \subseteq V \times V$ describing the dependencies between the operations.

Each edge $e_{i,j} \in E$ represents a constraint that operation v_i is a source of data for operation v_j and must be executed before it. Each edge $e_{i,j}$ is associated with a data transfer volume D_{ij} (in bytes). An example of a task graph is shown in Fig. 1.

For each operation v_k, there is a set of operations $pred(v_k)$ on which v_k depends and a set of operations $succ(v_k)$ that depend on v_k. Operations $v_i \in pred(v_k)$ are connected to v_k by edges $e_{i,k}^{pred} \in E_k^{pred}$. Operations $v_j \in succ(v_k)$ are connected to v_k by edges $e_{k,j}^{succ} \in E_k^{succ}$.

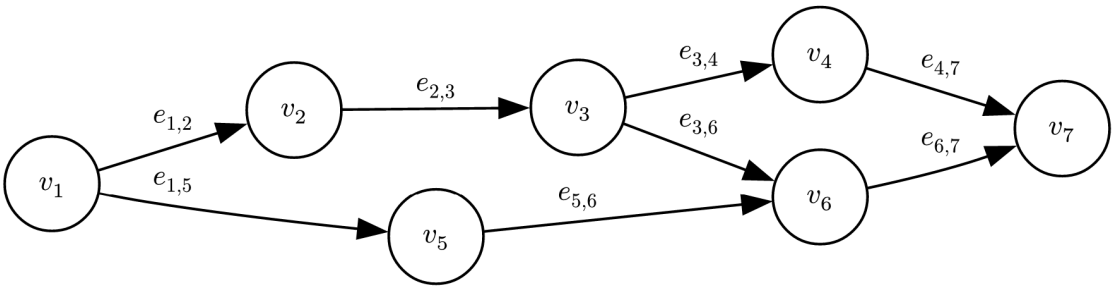

Figure 1. Example of a task graph

Regardless of the composition of HPC, the task execution process can be simplified as the following sequence:

- placing the task in the HPC queue;
- assigning nodes to execute the task by the HPC dispatcher;
- preparing the selected nodes for the task: loading data, deploying and configuring the necessary software components, reconfiguring hardware accelerators (loading FPGA configurations, ANN weights), and
- configuring other hardware;
- launching the appropriate software modules to execute the task;
- transmitting the output data to the recipient (user).

Thus, the actual execution of any computational operations takes only a fraction of the time. The total execution time of an individual task z can be simplified by the formula:

$$\tau(z) = T_{wait}(z) + T_{sched}(z) + T_{conf}(z) + T_{calc}(z), \tag{1}$$

where $T_{wait}(z)$ is the wait time for task z in the queue, $T_{sched}(z)$ is the time it takes to assign the task to the HPC nodes, $T_{conf}(z)$ is the time it takes to configure the corresponding HPC resources, and $T_{calc}(z)$ is the calculation time.

Next, let us consider the factors that influence these variables.

The wait time for a task in the queue, T_{wait}, is determined by how busy the dispatching system is processing other tasks. In fact, this indicator determines the throughput of the dispatching system as a whole.

The time it takes to assign the task to the HPC nodes, T_{sched}, is determined by the performance of the scheduling algorithm used by the dispatching system. Different algorithms have different computational complexity depending on the number of scheduled operations, the connections between them, the number of HPC nodes, and other parameters.

The time it takes to configure the HPC T_{conf} resource is determined by the complexity of bringing the HPC nodes selected during scheduling to a state of readiness to perform the assigned operations. It is worth noting that incoming tasks may require different dynamic configurations of computing nodes, and with a high influx of such tasks, frequent reconfiguration of the computing nodes will lead to unnecessary time expenditures. Therefore, when implementing task dispatching in HPC, it is necessary to consider the compatibility of different types of operations when assigning them to a single node.

Each node n_i can execute a certain subset of operation types $O(n_j) \equiv O^j \subseteq O$ according to its static and dynamic configuration, and each operation corresponds to one of the operation types $O(v_i) \in O$. Moreover, the task graph may contain several operations of the same type ($\bigcup O(v_i) \subseteq O$, $\bigcup O(n_j) = O$).

The parameters of operation v_k include:

- d_k^{CPU} – the required number of processor cores;
- d_k^{RAM} – the required memory size;
- d_k^{SCONF} – the required static configuration;
- d_k^{DCONF} – the required dynamic configuration.

We define a function $\mathcal{A} : V \to N$ that determines the assignment of each operation $v_i \in V$ to node $n_j \in N$. Operation v_k can be executed on node n_j if the compatibility conditions are met:

$$d_k^{CPU} \le r_j^{CPU}, \quad d_k^{RAM} \le r_j^{RAM}, \quad d_k^{SCONF} = r_j^{SCONF}, \quad d_k^{DCONF} = r_j^{DCONF}. \tag{2}$$

To run multiple operations on node n_j simultaneously, the total values of the d_k^{CPU} and d_k^{RAM} parameters must be less than or equal to the corresponding node parameters, and d_k^{SCONF} and d_k^{DCONF} must match for all operations. That is, the compatibility conditions must be met:

$$\begin{cases} \sum_{\mathcal{A}(v_k)=n_j} d_k^{CPU} \le r_j^{CPU}, \\ \sum_{\mathcal{A}(v_k)=n_j} d_k^{RAM} \le r_j^{RAM}, \\ d_k^{SCONF} = r_j^{SCONF}, \forall k : \mathcal{A}(v_k) = n_j, \\ d_k^{DCONF} = r_j^{DCONF}, \forall k : \mathcal{A}(v_k) = n_j, \end{cases}, \tag{3}$$

where $\mathcal{A}(v_k)$ is the node on which operation v_k is located.

2.3. Task Execution Time Estimation

As previously noted, each change of the dynamic configuration of node n_j from state a to state b takes a certain time $T_j^{RECONF}(a,b)$. Thus, the preparation time can be expressed as follows:

$$T_{conf} = \max_{v_i \in V} \left(T_{\mathcal{A}(v_i)}^{RECONF} \left(r_{\mathcal{A}(v_i)}^{DCONF}, d_i^{DCONF} \right) \right). \tag{4}$$

The computation time T_{calc} generally depends on the quality of the placement proposed by the scheduler, that is, on how well the selected HPC nodes match the assigned operations.

Consider a task described by a graph $G(V,E)$ and distributed across a set of nodes $\mathcal{N}\prime \subseteq \mathcal{N}$ belonging to the HPC. Let us estimate the execution time of such a task.

If operations v_i and v_j are executed on different nodes, then the data transfer time between them is:

$$T_{transfer}(v_i, v_j) = \frac{D_{ij}}{\min\left(B\left(\mathcal{A}(v_i)\right), B(\mathcal{A}(v_j))\right)} \quad , \tag{5}$$

where D_{ij} is the volume of data transferred between operations v_i and v_j, $B(n)$ is the throughput of the channel connecting node n to the communication medium.

Let $start(v_i)$ and $finish(v_i)$ denote the start and end moments of the execution of operation v_i, and $T(v_i, n_j)$ be the execution time of operation v_i on node n_j. Then:

$$finish(v_i) = start(v_i) + T(v_i, \mathcal{A}(v_i)). \tag{6}$$

For operation v_i depending on the completion of operation v_p:

$$start(v_i) = \max_{v_p \in pred(v_i)} (finish(v_p) + T_{transfer}(v_i, v_p)\,[\mathcal{A}(v_i) \neq \mathcal{A}(v_p)]), \tag{7}$$

where $[\mathcal{A}(v_i) \neq \mathcal{A}(v_p)]$ is a condition indicating that the operations are performed on different nodes and that data transfer time must be taken into account.

Thus, the total task completion time is calculated recursively using the formula:

$$T_{calc} = finish(v_{exit}) = T(v_{exit}, \mathcal{A}(v_{exit})) + \max_{v_p \in pred(v_{exit})} (finish(v_p) + T_{transfer}(v_{exit}, v_p)\,[\mathcal{A}(v_{exit}) \neq \mathcal{A}(v_p)]),\ v_{exit} : succ(v_{exit}) = \emptyset. \tag{8}$$

3. Resource Dispatching for a High-Performance Computing Cluster Based on the Tertius-2 RCU

When using traditional job scheduling methods in high-performance computing clusters (HPCs) that include reconfigurable computing units, the execution time of the corresponding operations $T(v_i, n_j)$ includes not only the actual computation time T_{calc} (v_i, n_j) but also the resource reconfiguration time $T_{n_j}^{RECONF}\left(r_{n_j}^{DCONF},\ d_i^{DCONF}\right)$. This does not have a significant impact when performing long calculations based on a single reconfiguration (when $T_{calc} \gg T^{RECONF}$), but when intensively processing jobs with short operations, when the computation time is comparable to the reconfiguration time, it will lead to a significant loss of time and delays in job processing.

To overcome the problem described above, a multi-agent dispatching system monitors the state of reconfigurable resources and reduces reconfiguration latency by rationally distributing task operations across HPC nodes, taking into account their current and planned configurations.

As part of the research on "Development of a Multi-Agent Resource Manager for a Heterogeneous Supercomputer Platform Using Machine Learning and Artificial Intelligence Methods", experimental studies of a multi-agent cluster dispatching system using the Tertius-2T RCU manufactured by "SRC SC & NC" Co. Ltd were conducted at the Polytechnic Scientific and Technical Center.

To experimentally study the proposed method for coordinating data flow processing task assignments in HPC, a prototype software system implementing the corresponding algorithms was developed.

The system structure is shown in Fig. 2.

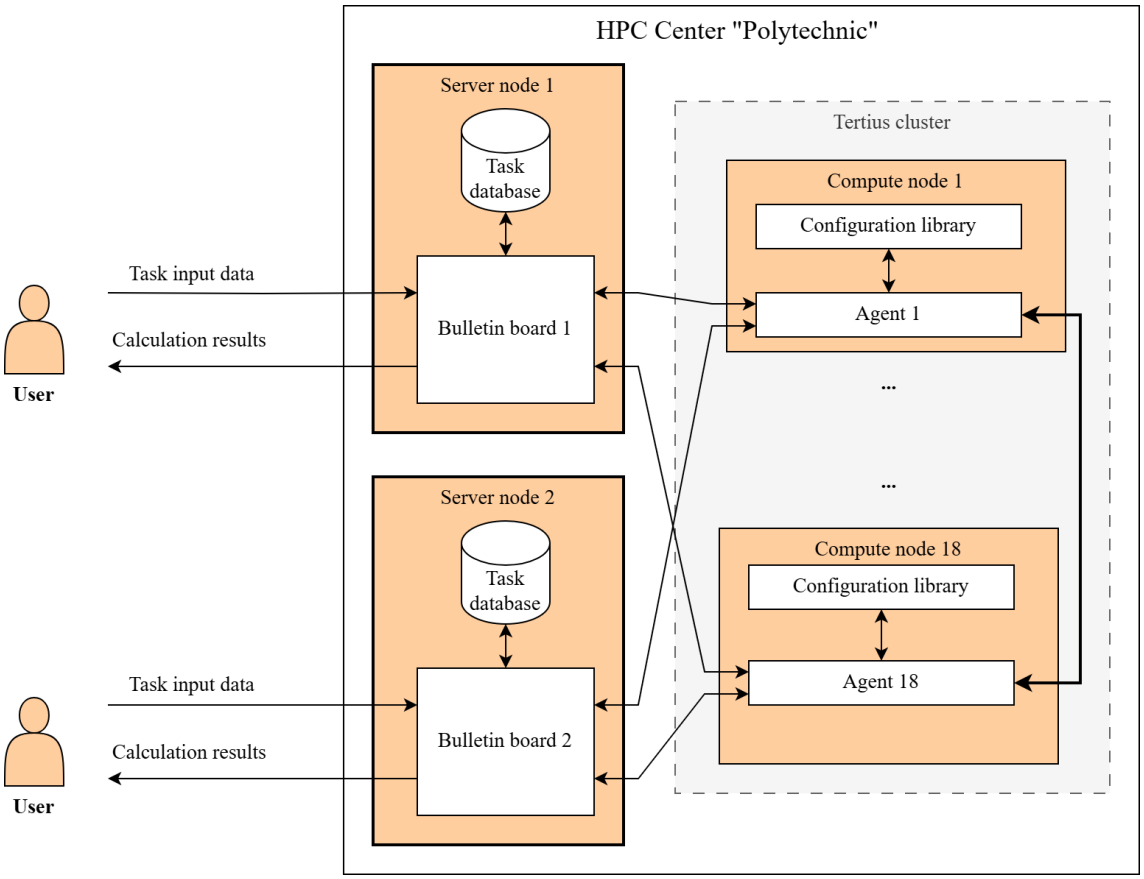

Figure 2. General system structure

The system includes:

- 18 Tertius-2T reconfigurable computation units (RCUs);
- 2 server nodes;
- 18 agents deployed on RCUs;
- 2 bulletin boards serving as job queues on the server nodes.

The software (SW) of the Dispatching MAS agent is designed to organize the execution of user jobs on RCUs within the supercomputer center (SCC).

The agent implements the following functions:

- monitoring available jobs on RCUs;

- receiving job information for RCUs from the bulletin board (BB);
- receiving job input data for RCUs from BB;
- receiving configuration module files for RCU required to execute tasks from the high-performance reconfigurable computing cluster (HPRCC) configuration module library;
- configuring RCU to execute user tasks according to the type of each task by selecting the required configuration module of HPRCC;
- initiating the process of executing user tasks on RCU;
- transmitting the current status of tasks to RCU during their execution;
- receiving the results of user tasks and transmitting these results to RCU.

RCU implements the following functions:

- storing task configuration data and associated application files and data;
- receiving task information for RCU from the user, as well as the task data itself;
- storing task information for RCU, the task data itself, and the task calculation results before transferring them to the user.
- transferring output data (execution results) of tasks to RCU to the user.

The structure of BB is shown in Fig. 3.

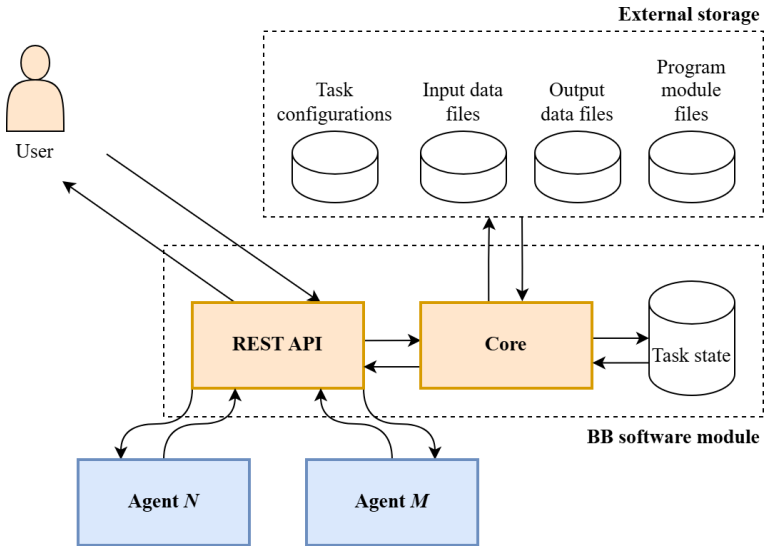

Figure 3. Bulletin board structure

The agent structure is shown in Fig. 4.

The MAS software was developed in C++ using the Qt 5.15 library and is designed to run on GNU/Linux OS.

4. Results of Experimental Studies of the Dispatch MAS

To evaluate the effectiveness of the proposed dispatch MAS, experimental studies were conducted on a cluster consisting of up to 18 Tertius-2T RCUs and two server nodes. The experiments simulated the processing of task queues ranging from 200 to 3,500 tasks with varying numbers of nodes and in two modes:

- without reconfiguration optimization (basic mode);
- with reconfiguration optimization and partial agent consent.

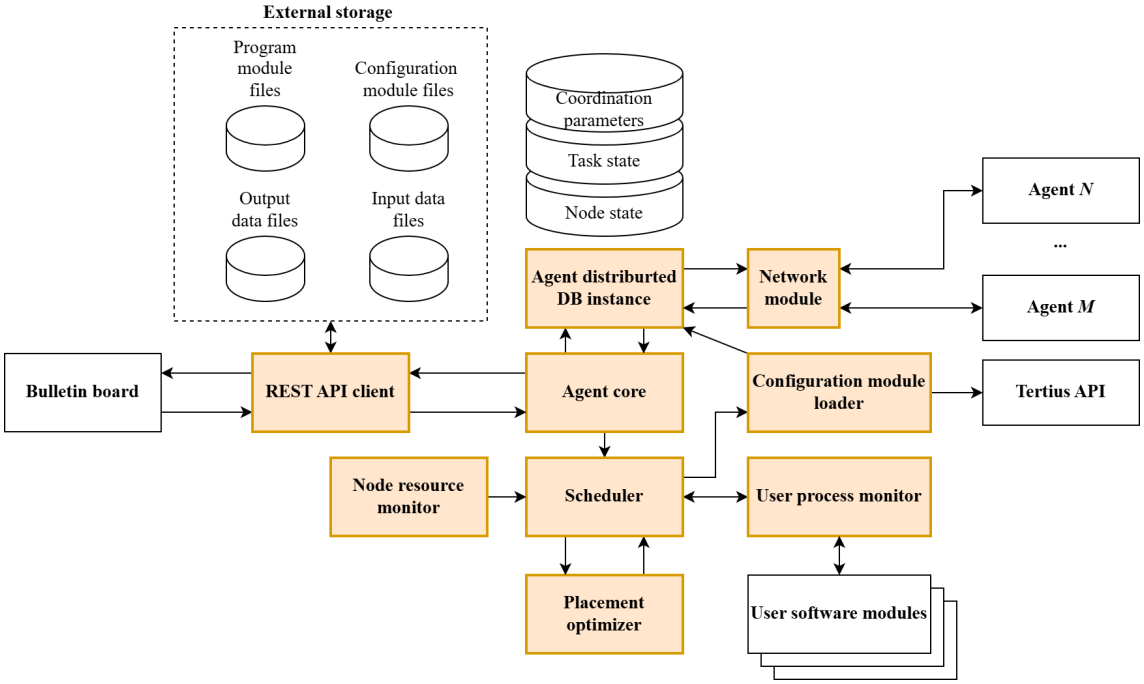

Figure 4. Agent structure

The tasks represented a set of typical operations reflecting target scenarios for using FPGAs in a HPC. For each task, the solution time was measured, including possible FPGA reconfiguration, and the net computation time with a suitable configuration. For each series of experiments, the following were recorded:

- queue size and total number of tasks;
- number of nodes involved;
- total solution time for all tasks in the queue;
- average solution time for one task;
- minimum, maximum, and average net computation time.

The experimental results are presented in Tab. 2.

Table 2. Experimental results

Exp. No	Total number of tasks	Number of nodes	Time to solve all tasks in the queue, sec	Average time to solve 1 task, sec	Net solution time, sec		
					min	max	avg
without reconfiguration optimization							
1	200	17	105	9.0	0.253	3.932	1.602
2	5000	17	3294	11.2	0.247	3.951	1.597
3	3500	6	8458	14.5	0.269	7.851	1.533
4	3500	17	2841	13.8	0.249	3.949	1.567
with reconfiguration optimization and partial agent consent							
5	1000	9	700	6.300	0.251	3.922	1.582
6	1000	10	413	4.130	0.248	3.930	1.613
7	1000	16	762	12.192	0.246	4.019	1.591
8	3500	6	5263	9.022	0.268	7.848	1.531

In the mode without reconfiguration optimization, a significant increase in the average task execution time is observed with increasing queue size and/or decreasing the number of nodes. This is due to the fact that tasks of different types are assigned to nodes without regard for current and future configurations, which leads to frequent FPGA reconfigurations and high overhead.

Enabling the partial agreement algorithm and reconfiguration optimization results in a significant reduction in the average task execution time. For queues of up to 3,500 tasks, the reduction can be multiple compared to the basic mode, while the net computational time remains comparable. This indicates that the gain is achieved precisely due to the reduction in reconfiguration time and more efficient use of already configured nodes.

The dynamics of task execution time during queue processing is illustrated in Fig. 5. In the optimization mode, a more even distribution of solution time is observed, the absence of pronounced peaks associated with large-scale reconfigurations, and faster queue burnout.

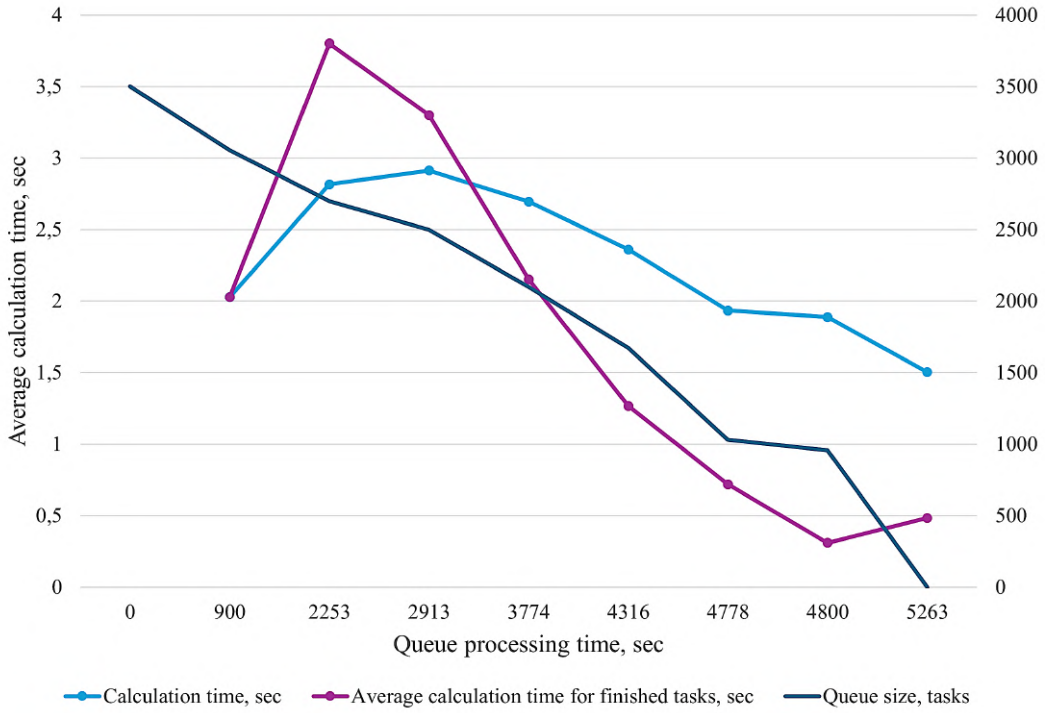

Figure 5. Dynamics of task resolution time changes during queue processing

The study demonstrated a significant reduction in average task resolution time, CPU load, and network infrastructure utilization when using the partial agreement algorithm for dispatching task queues up to 3,500 in size.

5. Discussion of Results and Limitations

The experimental results demonstrate that accounting for FPGA reconfiguration time and the use of a multi-agent approach can significantly improve the efficiency of task flow processing in a heterogeneous cluster. The greatest gains are achieved in modes where:

- tasks are relatively short compared to configuration time;
- the task flow contains several common types that regularly recur in the queue;
- the number of available nodes is sufficient to separate responsibilities by task type.

However, this approach has several limitations. It is necessary to support agent deployment on each node, which increases software complexity and places demands on its reliability. The decentralized nature of the algorithm generates communication overhead between agents and bulletin boards, although experiments show that this overhead is significantly lower than the time savings from reconfigurations.

Another limitation is the dependence of efficiency on the accuracy of computational and configuration time estimates. Without adequate models or statistics, agents may make suboptimal decisions. Furthermore, the current implementation does not take into account such factors as power consumption, FPGA resource degradation, and possible task priorities, which represents a promising area for further research.

Conclusion

This paper examines the problem of task flow dispatching in a heterogeneous shared supercomputer center incorporating reconfigurable FPGA-based computing units. It is shown that reconfiguration overhead can account for a significant portion of the time spent processing short tasks, and that the resource allocated to the number of configuration cycles is an important limiting factor.

A multi-agent dispatching system is proposed in which software agents running on each RCU make decisions on task assignment and FPGA reconfiguration based on local information and partial consensus mechanisms. A system model that takes into account computational and reconfiguration time is described, and optimization criteria are formulated.

Experimental studies on a cluster using the Tertius-2T RCU showed that the proposed methods can significantly reduce the average task resolution time and reconfiguration overhead for queue sizes of up to 3,500 tasks. Simultaneously, the load on the RCU CPU and network infrastructure is reduced, confirming the effectiveness of the multi-agent approach.

Future development plans include enhancing the system to integrate various types of computing devices into a single computing supercluster. This will enable end-to-end task dispatching across CPUs, GPUs, and FPGAs with a unified set of quality-of-service criteria, taking into account their architectural features. This will also enable adaptive load redistribution between heterogeneous nodes based on online performance and workload assessments for individual computing types. Additional constraints related to energy efficiency and the service life of the FPGA's non-volatile configuration memory will be incorporated into the model, allowing for schedule optimization based on these factors.

A separate area of development is the transfer and adaptation of the proposed multi-agent system for industrial use at existing SCCs with heterogeneous configurations, as well as expanding the range of supported applications, including complex pipelined and graph computing workloads.

References

1. Adimora, K., Gundla, S.R., Sun, H.: Machine learning approaches for optimizing high-performance computing scheduling: a comprehensive survey and analysis. Cluster Computing 28(13), 831 (2025). https://doi.org/10.1007/s10586-025-05521-8

2. Chen, S., Huang, J., Xu, X., *et al.*: Integrated optimization of partitioning, scheduling, and floorplanning for partially dynamically reconfigurable systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 39(1), 199–212 (2018). https://doi.org/10.1109/tcad.2018.2883982

3. Houssein, E.H., Gad, A.G., Wazery, Y.M., Suganthan, P.N.: Task scheduling in cloud computing based on meta-heuristics: Review, taxonomy, open challenges, and future trends. Swarm and Evolutionary Computation 62, 100841 (2021). https://doi.org/10.1016/j.swevo.2021.100841

4. Hsieh, F.S.: Analysis of contract net in multi-agent systems. Automatica 42(5), 733–740 (2006). https://doi.org/10.1016/j.automatica.2005.12.002

5. Huang, M., Narayana, V.K., Simmler, H., *et al.*: Reconfiguration and communication-aware task scheduling for high-performance reconfigurable computing. ACM Transactions on Reconfigurable Technology and Systems (TRETS) 3(4), 1–25 (2010). https://doi.org/10.1145/1862648.1862650

6. Jiang, Y., Yi, P., Zhang, S., Zhong, Y.: Constructing agents blackboard communication architecture based on graph theory. Computer Standards & Interfaces 27(3), 285–301 (2005). https://doi.org/10.1016/j.csi.2004.09.003

7. Kaliaev, A.: Multiagent approach for building distributed adaptive computing system. Procedia Computer Science 18, 2193–2202 (2013). https://doi.org/10.1016/j.procs.2013.05.390

8. Kostenetskiy, P., Shamsutdinov, A., Chulkevich, R., *et al.*: HPC TaskMaster – Task Efficiency Monitoring System for the Supercomputer Center. In: Sokolinsky, L., Zymbler, M. (eds.) Parallel Computational Technologies. pp. 17–29. Springer International Publishing, Cham (2022). https://doi.org/10.1007/978-3-031-11623-0_2

9. Macronix International Co., L.: Wear Leveling in NAND Flash Memory. https://www.mxic.com.tw/Lists/ApplicationNote/Attachments/1913/AN0289V1-Wear%20Leveling%20in%20NAND%20Flash%20Memory.pdf (2014), accessed: 2016-04-08

10. Parasumanna Gokulan, B., Srinivasan, D.: An Introduction to Multi-Agent Systems, vol. 310, pp. 1–27 (07 2010). https://doi.org/10.1007/978-3-642-14435-6_1

11. Rodriguez-Canal, G., Brown, N., Torres, Y., Gonzalez-Escribano, A.: Task-based preemptive scheduling on FPGAs leveraging partial reconfiguration. Concurrency and Computation: Practice and Experience 35(25), e7867 (2023). https://doi.org/10.1002/cpe.7867

12. Samayoa, W.F., Crespo, M.L., Cicuttin, A., Carrato, S.: A survey on FPGA-based heterogeneous clusters architectures. IEEE Access 11, 67679–67706 (2023). https://doi.org/10.1109/ACCESS.2023.3288431

13. Shehata, A., Groszkowski, P., Naughton, T., *et al.*: Bridging paradigms: Designing for HPC-Quantum convergence. Future Generation Computer Systems 174, 107980 (2026). https://doi.org/10.1016/j.future.2025.107980

14. Thiyagaraj, B.: Multi-Agent Systems for Distributed Task Scheduling in High-Performance Computing. American International Journal of Computer Science and Technology 3(6), 11–22 (2021). https://doi.org/10.63282/3117-5481/AIJCST-V3I6P102

15. Vipin, K., Fahmy, S.A.: FPGA Dynamic and Partial Reconfiguration: A Survey of Architectures, Methods, and Applications. ACM Comput. Surv. 51(4) (Jul 2018). https://doi.org/10.1145/3193827

16. Wang, B., Chen, Z., Xiao, N.: A Survey of System Scheduling for HPC and Big Data. In: Proceedings of the 2020 4th International Conference on High Performance Compilation, Computing and Communications. pp. 178–183. HP3C 2020, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3407947.3407977

17. Zhang, J., Wang, C., Zhu, Z., *et al.*: Realizing extreme endurance through fault-aware wear leveling and improved tolerance. In: 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA). pp. 964–976. IEEE (2023). https://doi.org/10.1109/HPCA56546.2023.10071093

18. Zhang, J., Wang, G., Song, Y.: Task assignment of the improved contract net protocol under a multi-agent system. Algorithms 12(4), 70 (2019). https://doi.org/10.3390/a12040070