OptiMA: A Transaction-Based Framework with Schedule Optimization for Very Complex Multi-Agent Systems

Umut Çalıkyılmaz[1], Nitin Nayak[1], Jinghua Groppe[1,2], and Sven Groppe[1,2]

[1] University of Lübeck, Ratzeburger Allee 160, 23562 Lübeck, Germany
{umut.calikyilmaz,nitin.nayak,jinghua.groppe}@uni-luebeck.de
[2] TU Bergakademie Freiberg, 09599 Freiberg, Germany
sven.groppe@informatik.tu-freiberg.de

Abstract. In recent years, multi-agent system (MAS) research has increasingly focused on complex systems composed of collaborating agents based on large language models (LLMs) to accomplish sophisticated tasks. These systems rely on dynamic and heterogeneous sets of agents, concurrent access to multiple external tools, and cascading operations that span multiple agents. Existing work on the coordination of MAS mostly adopts semantic approaches for model design such as agent architectures, interaction protocols, and hierarchical structures. However, we argue that greater attention to the execution layer is required to coordinate modern MAS models. To address this gap, we propose OptiMA, a framework that encapsulates atomic agent actions as transactions and uses lock-based concurrency control to provide a high level of consistency and isolation. It gives system designers explicit control over the execution process by incorporating transaction templates into the design process. To reduce potential performance drawbacks of this approach, OptiMA employs a transaction scheduling mechanism. In this paper, we also present the theoretical results on the transaction scheduling problem (TxnSP) and introduce a metaheuristic approach for schedule optimization, which is used within the OptiMA framework. Experimental results show that transaction scheduling can improve performance up to 45% for the tested model configurations, indicating that high system consistency can be achieved while minimizing performance cost.

Keywords: Multi-agent · AI · Transaction Scheduling · Optimization.

1 Introduction

After the recent advances in large language models (LLMs), many studies in the field of artificial intelligence (AI) have focused on applications that maximize the performance of these models. Recent work focuses on large-scale systems that utilize a dynamic set of heterogeneous agents organized in a hierarchical structure. In such systems, an agent can use multiple external tools within a single action, and an individual task might include cascading operations that

span multiple agents. The ultimate goal of this line of research is to develop automated systems capable of accomplishing sophisticated tasks that are usually performed by teams of humans. Building systems of such complexity requires well-designed coordination mechanisms to avoid problems such as conflicting actions, race conditions, and inconsistent states.

Multi-agent systems (MAS) research has studied mechanisms to coordinate distributed agents for decades [9]. The primary focus of existing MAS studies is coordination models that rely on agent architectures [11], organizational structures [10], and interaction protocols [29]. While providing useful abstractions, such models operate at a high level and do not explicitly ensure concurrency control or enforce consistency constraints during parallel execution. Modern systems with LLM-based agents dynamically generate actions and concurrently access shared tools, which may cause conflicts and inconsistencies [38]. However, in certain domains, such as financial systems and healthcare applications, ensuring strong consistency is critical, since inconsistent states can have severe consequences [14]. We offer a transaction-based approach where execution-level concerns are part of the design process to provide stricter control.

OptiMA is a framework developed for the design and execution of complex multi-agent systems. It encapsulates atomic operations as transactions, enforces consistency constraints during execution, and utilizes strict lock-based concurrency control to regulate access to non-shareable tools. To overcome possible shortcomings of this design choice, we integrate a transaction scheduling mechanism based on insights from database transaction management. We build upon studies that treat transaction scheduling as a mathematical optimization problem to develop a scheduling approach. This work includes a thorough theoretical analysis of the transaction scheduling problem (TxnSP) and offers a metaheuristic approach making use of this analysis. To test the impact of scheduling on performance, we designed a synthetic benchmark scenario specifically focused on resource contention caused by tool sharing. The results show that transaction scheduling significantly increases throughput and its effect increases as the number of agents increases, which demonstrates that our approach can provide strong consistency while minimizing the adverse effects of strict locking procedures.

The remainder of the paper is organized as follows. Section 2 and Section 3 review the state of the art in multi-agent systems and database transaction management, respectively. Section 4 presents the OptiMA framework. Section 5 introduces the transaction scheduling problem, provides its theoretical analysis, and presents a metaheuristic approach. Section 6 presents experimental results demonstrating the impact of transaction scheduling in OptiMA. Finally, Section 7 concludes the paper and outlines future research directions.

2 Multi-Agent Systems

One recent trend in AI research is designing more effective systems to enhance the capabilities of LLMs instead of developing and training larger models. For example, chain-of-thought prompting improves performance by applying multiple

steps in the reasoning process [36]. Another popular approach is to provide LLMs with external tools, which allows various functionalities such as data retrieval, API calls, and code execution [23, 20]. In fact, the agents in the latest models are allowed to invoke multiple tools [38], and new protocols such as the model context protocol (MCP) are being used to orchestrate tool sharing [19]. In the meantime, it has been shown that a system composed of a heterogeneous set of agents based on different LLMs performs significantly better than an individual LLM [40, 6]. Heterogeneous systems are becoming increasingly complex by the introduction of hierarchical relations between agents and complex task structures that might span multiple agents [39]. Recent frameworks combine these ideas by introducing specialized agent roles, various external tools, and dynamically changing populations of heterogeneous agents [33, 19]. In [37], it is shown that these complex systems are capable of automating tasks that normally require a team of humans, such as software development. This work aims to ensure strong consistency for similar very complex multi-agent systems, while minimizing performance trade-offs.

In existing MAS research, a wide range of abstract models has been proposed for the coordination of distributed agents. An important example is the Belief-Desire-Intention (BDI) model, which separates the internal state, goals, and actions of an agent and defines how these components affect each other [11]. In contrast, interaction-based approaches prioritize information exchange between agents rather than their internal structures [29]. These approaches define constraints on how agents communicate and act, through structures such as interaction protocols or commitments [30, 31, 7]. Another line of work introduces hierarchical structures between agents to coordinate the behavior of systems. For example, Holonic MAS models treat each agent as a *holon*, an autonomous entity that serves as a part of a larger whole, and forms hierarchies on multiple levels [12]. Several studies follow a softer approach by assigning specific roles to agents to define the limits of each agent and the relationships between them [10, 41]. This particular idea has been used for LLM-based systems in recent work [33], and our framework also depends on role-based modeling for high-level coordination.

Beyond these semantic and organizational models, there are some transaction-based MAS models that target coordination at the execution level to ensure consistency. For example, the TrAM model introduces transaction templates into MAS design and uses low-level control to ensure consistency [18]. TrAM uses agent roles for coordination similar to our approach. Another transaction-based approach given in [27] focuses on improving consistency by dynamically assigning transactions to threads, rather than binding each agent to a single thread. This model follows a different method for organization by forming hierarchies between transactions, such as nested transactions, instead of assigning roles to agents. Both models do not consider tool use or transactions spanning multiple agents, which excludes them from being used to design modern MAS models.

3 Concurrency Control and Transaction Scheduling

A transaction can be defined as the smallest unit of transition of the state of a database. One of the main duties of a database management system (DBMS) is to ensure that each transaction satisfies ACID properties (**A**tomicity, **C**onsistency, **I**solation, and **D**urability), which ensure that the database always stays in a valid state, even under heavy load or in the case of system failures [26]. To ensure the isolation property, the conflicts between transactions must be handled by a DBMS using a concurrency control (CC) method. Two-phase locking (2PL) is a widely used method that uses shared and exclusive locks to regulate the access of transactions to the database [34]. In 2PL, a transaction acquires all locks during the expanding phase and releases them during the shrinking phase, as shown in Figure 1a. In a variant of 2PL, namely the rigorous conservative 2PL (RC2PL), a transaction obtains all locks before starting any action and releases them only after commit or abort (Figure 1b). This provides a higher level of isolation without the need to abort a large number of transactions by forbidding two conflicting transactions to have any overlapping operations. Due to performance concerns, most modern DBMSs use less restrictive CC methods that either accept a lower isolation or risk a high number of aborts [26]. Multi-version concurrency control (MVCC) keeps multiple versions of database rows to allow concurrency without using locks [2]. Optimistic concurrency control (OCC) allows the parallel processing of transactions without locks, but aborts some of the transactions at the commit time in case of conflicts [17].

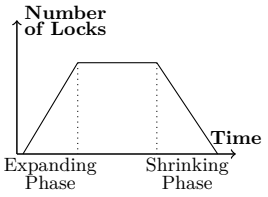
In previous studies, various approaches have been proposed to improve the throughput of a DBMS. Transaction partitioning is one such method, in which conflicting transactions are grouped together, and processed on the same thread [25, 42, 28]. In [5], the authors show that scheduling the execution times of the transactions after partitioning improves the performance even further. These approaches consider only a subset of all feasible schedules. In recent years, more systematic methods have been offered, in which transaction scheduling is treated as a mathematical optimization problem and an efficient schedule is selected from the set of all possible schedules [3, 16]. We follow a similar approach by implementing a TxnSP solver.

4 OptiMA Framework

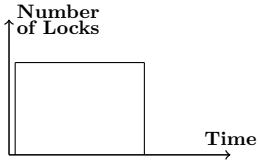
The OptiMA framework[3] combines transaction encapsulation, concurrent tool sharing, and role-based organization to design multi-agent systems with explicit control at the execution layer. This approach enforces ACID-level consistency for each operation and maintains a high level of isolation in tool sharing.

As discussed in Section 2, the state of the art in MAS research has shifted towards increasingly complex models including heterogeneous and dynamic sets of agents with access to multiple tools. Such systems may involve complex operations that span actions from multiple agents. In some cases, a partial completion of

[3] `https://github.com/umutcalikyilmaz/OptiMA`



(a) Standard 2PL



(b) Rigorous Conservative 2PL

Fig. 1: Mechanisms of the Two-Phase Locking variants

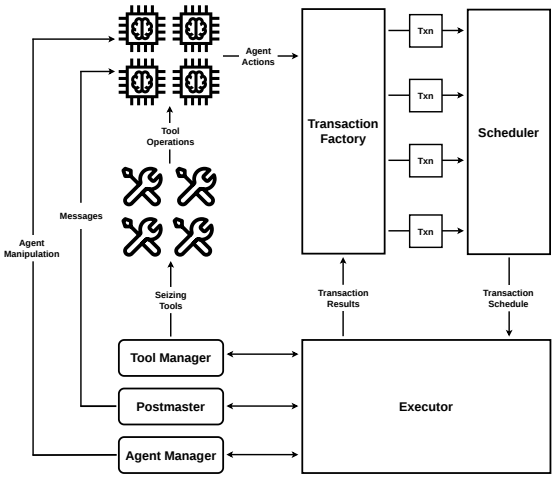


Fig. 2: Structure of the OptiMA Engine

such operations may leave the system in an inconsistent state [14, 22]. OptiMA addresses this issue by enabling system designers to encapsulate such operations in atomic transactions.

Another challenge in modern LLM-based MAS is the coordination of concurrent access to tools. Some external tools in MASs correspond to real-world resources or constrained services that cannot be concurrently accessed without proper isolation [24, 22]. Furthermore, as noted in [15], the effects of real-world actions are usually not reversible, making optimistic concurrency control methods such as MVCC or OCC less suitable due to their reliance on transaction rollbacks. OptiMA uses a strict lock-based concurrency control resembling RC2PL, where a transaction acquires all the locks required for the tools it accesses before execution, and releases them only after completion. According to the CAP theorem [13], prioritizing consistency and isolation in distributed systems causes a trade-off in availability. OptiMA uses transaction scheduling to mitigate this effect, which improves the performance of concurrent execution by decreasing delays caused by conflicts.

4.1 Model Design

The model design process in the OptiMA framework consists of two stages. The first stage is defining the main components of the model, tools, agent roles, and transaction templates. We describe these components below.

Tools: Tools are offered for the use of AI agents to extend the functionality of the model. To ensure system consistency, some tools do not permit concurrent access. These are marked as *non-shareable*, and access to them is controlled by a locking procedure.

Agent Roles: An agent role is a template for an agent that determines its internal processes and possible actions. Model constraints are defined on the level of agent roles, therefore agents sharing the same role have access to the same tools and have the same relationships with other roles. The initial and maximum number of agents that have each role is set as a design parameter.

Transaction Templates: A transaction template defines the operation of a transaction. An operation specified in a template can include actions from multiple agents, enabling the construction of transactions with arbitrary complexity. To ensure consistency and abstraction, transaction operations are not allowed to directly invoke tools, instead tools are accessed through agent actions. At runtime, transactions are instantiated by assigning parameters to a template. Transactions instantiated from the same template have the same transaction type, which is used to estimate execution time during scheduling. In addition to the main operation, templates are allowed to define functions that are invoked in case the transaction is committed or aborted. These functions can be used to log the transaction results after commit and to implement rollback procedures.

The second stage of the model design process in OptiMA is the specification of the model constraints. These constraints define the boundaries of the system by determining the permissions of each agent role and shaping their relationships with other roles. The model constraints in the framework are listed below.

Supervisor-Subordinate Relationship: An agent role can be designated as the supervisor of another role. A supervisor can start, stop, create, or destroy another agent with the subordinate role.

Tool Access: An agent role must be explicitly granted authorization to access a tool at runtime.

Communication Permission: An agent can communicate with another agent only if they share the same role, they have a supervisor-subordinate relationship, or their roles are explicitly given permission to communicate.

Halting Permission: The ability to halt the program is granted to certain agent roles. At least one agent role must keep this permission to ensure that the program can terminate.

4.2 Model Execution

The OptiMA engine consists of three main modules and three auxiliary modules, as shown in Figure 2. The main modules, transaction factory, scheduler, and executor, are responsible for generating transactions, assigning them to execution threads, and executing them while ensuring consistency. These modules form a cyclic workflow, in which the result of each transaction may trigger the creation of subsequent transactions. The auxiliary modules, tool manager, postmaster, and agent manager, assist the main modules in enforcing model constraints and performing system operations during execution. The functionality of each module is described below.

Transaction Factory: The primary responsibility of this module is to create transactions based on the transaction templates defined during model design. The transaction factory initiates execution by creating the initial set of transactions and sending them to the scheduler. The result of each transaction is sent back to the transaction factory, which may trigger the creation of new transactions. This loop continues until the program is halted by an authorized agent.

Scheduler: The scheduler batches the transactions sent by the transaction factory. A scheduling process is triggered either when the batch size reaches a predefined threshold or when a maximum waiting time has elapsed since the last scheduling event. During scheduling, an instance of TxnSP is created and solved using the metaheuristic method developed using the theoretical findings presented in Section 5. To construct a problem instance, the execution time of each transaction is estimated using the statistics collected from the executed transactions of the same type. Conflicts between transactions are identified based on their use of non-shareable tools. If two transactions request access to the same non-shareable tool, they are considered conflicting. The resulting schedule is m transaction queues, where m is the number of available queues. Each queue is assigned to a separate thread in the executor module.

Executor: Transactions assigned to each queue by the scheduler are inserted into the thread-local queues without reordering. Each thread executes the transactions in its queue in a first-in-first-out (FIFO) manner. The executor module ensures consistency during execution. Before execution begins, each transaction must acquire locks for the non-shareable tools it requires for its operation, and it cannot start until all required locks are available. A transaction releases all acquired locks after it commits or aborts. In OptiMA, each tool is assigned a unique identifier, and the locks are acquired and released sequentially in ascending order of the tool identifiers to avoid deadlocks [14]. The executor interacts with the tool manager, agent manager, and postmaster modules to enforce model constraints. After execution, the result of a transaction is sent to the transaction factory.

Tool Manager: When an agent action within a transaction attempts to use a tool, the tool manager verifies the authorization of the agent. If authorization is valid, access to the tool is granted; otherwise, the executor is instructed to abort the transaction.

Agent Manager: When an agent action tries to start, stop, create, or destroy another agent, the executor asks the agent manager to verify the permission of the initiator agent. If the agent lacks the required permission, the executor is notified to abort the transaction. Otherwise, the agent manager performs the requested operation on the target agent.

Postmaster: When an agent attempts to send a message to another agent, the postmaster checks whether the communication between the two agents is allowed. If authorization is verified, the message is delivered. Otherwise, the executor is notified to abort the transaction.

OptiMA allows disabling transaction scheduling for a given execution. In this mode, the scheduler is bypassed, and each transaction is assigned to the first idle

thread in the executor. If scheduling is enabled, two parameters, *batch size* and *timeout*, must be specified. These define the maximum number of transactions in a batch and the maximum waiting time before a batch is processed by the scheduler. In Section 6, we evaluate different execution settings to find the most efficient ones for various model configurations.

5 The Transaction Scheduling Problem

5.1 Definition of the Problem

We define TxnSP as the problem of scheduling a set of jobs on identical parallel machines, where certain pairs of jobs cannot be processed concurrently due to conflicts. Preemption is not allowed and the processing time of a job can be any positive value. The objective is to find the schedule with the minimum makespan. Makespan is defined as the time required for all jobs to be completed. Due to conflicts, a schedule can have idle periods on some machines.

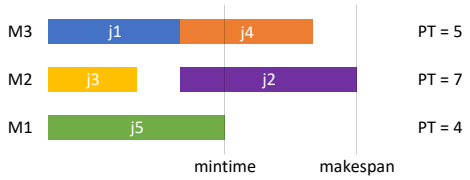


Fig. 3: An example transaction schedule

Figure 3 shows an example transaction schedule. In this schedule, jobs 1 and 2 are in conflict, so job 1 delays the start of job 2. The processing times of the machines are shown on the right. The longest processing time determines the *makespan*, and the shortest determines the *minimum time*. The parameters used to define a TxnSP instance are given below.

n: Number of jobs to be scheduled

m: Number of machines

L: A $1 \times n$ matrix of job lengths

C: A $n \times n$ matrix of binary values such that

$$C_{i,j} = \begin{cases} 1, & if\ i \neq j\ and\ jobs\ i\ and\ j\ conflict \\ 0, & otherwise. \end{cases} \tag{1}$$

C is a symmetric matrix with 0 values on its primary diagonal, since the conflict relation is commutative and anti-reflective.

TxnSP can be considered as a special case of the disjunctive scheduling problem (DSP) [8]. While DSP contains both disjunctive (mutual exclusion)

and conjunctive (precedence relations between jobs) constraints, TxnSP only includes disjunctive constraints. A related problem, namely the mutual exclusive scheduling problem (MESP), was proposed to model the parallel solution of differential equations [1]. In MESP it is assumed that each job has a unit length, therefore TxnSP can be seen as a generalization of MESP. Our formulation of TxnSP is inspired by [3] and [16], which also assume the use of RC2PL and treat transactions as indivisible blocks to formulate a quadratic binary optimization model for the problem. In the remainder of this section, we present a theoretical analysis of the TxnSP that proves the complexity of the problem and introduces useful properties that we use to develop a metaheuristic approach.

5.2 Computational Complexity

Theorem 1. *TxnSP is NP-Hard.*

Proof. If an algorithm exists that solves all TxnSP instances in polynomial time, then it can also solve all MESP instances in polynomial time. It is shown in [1] that MESP is NP-Hard. Therefore, TxnSP is also NP-Hard.

5.3 Permutation Encoding

In [4] a permutation encoding to represent the candidate solutions to TxnSP is proposed. This representation is particularly useful for implementing optimization methods such as the one we introduce in Section 5.4. In the following, we prove that permutation encoding reduces the size of the search space while guaranteeing that at least one optimal solution is representable for any problem instance.

Definition 1 (Subschedule). *A subschedule is a schedule that includes only a subset of the set of jobs. If subschedule i includes only the jobs in subset S, we say that i is formed by S, and its size is $|S|$. In a subschedule, each job is scheduled at its earliest feasible time. For a subschedule i, makespan and minimum time are denoted by ms_i and mt_i, starting and completion times of job α are denoted by $st_i^{(\alpha)}$ and $ct_i^{(\alpha)}$, and the set including the last job processed on each machine is denoted by LJ_i.*

Definition 2 (Derived Subschedule). *If a subschedule j can be obtained by adding $c \geq 0$ number of jobs to a subschedule i, then j is said to be derived from i. In this case i is called a root of j. More specifically, the root of j formed by subset S is called the S root of j.*

If removing the job $\alpha \in S$ causes at least one job not to start at the earliest feasible time, the result is not a subschedule and i does not have a $S - \{\alpha\}$ root.

Definition 3 (Derivation Plan). *A derivation plan is a list of instructions that is used to derive a subschedule from another. If j is derived from i by adding c jobs, then the related derivation plan $\mathcal{D}$ is a list of c instructions. Each instruction specifies which job to be inserted into which machine.*

Definition 4 (Empty Subschedule). *The subschedule that is formed by the empty set is called the empty subschedule.*

Definition 5 (Residual Subschedule). *If subschedule j is derived from i using derivation plan $\mathcal{D}$, residual subschedule $j - i$ is the subschedule derived from the empty subschedule using $\mathcal{D}$.*

Each derivation plan can be represented by a residual subschedule, since a residual subschedule shows the order and place of the jobs to be inserted. We use these two terms interchangeably.

Definition 6 (Equivalent Subschedules). *Subschedules i and j that are formed by S are equivalent if $LJ_i = LJ_j$ and $ct_i^{(\alpha)} = ct_j^{(\alpha)}$ for every $\alpha \in S$.*

Equivalency relation comes from the fact that the machines are identical in TxnSP. Thus, swapping the machines would produce the same set of processing times and the same set of last jobs.

Lemma 1. *Any equivalent of an optimal schedule is also optimal.*

Proof. If schedule i is optimal, its makespan has the minimum possible value. Since swapping machines would not change the makespan, an equivalent of i is also optimal.

Definition 7 (Reducible Subschedule). *Subschedule i is said to be reducible by α if ms_i can be decreased by replacing the job $\alpha \in LJ_i$ on another machine.*

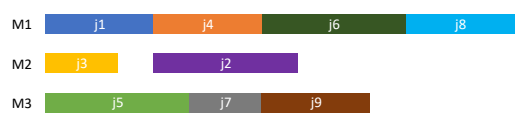


Fig. 4: An Example of Reducible Subschedule

In Figure 4, if $j6$ and $j8$ do not conflict, the subschedule is reducible by $j8$, since makespan is reduced by moving $j8$ to $M2$ or $M3$.

Definition 8 (Prime Subschedule). *Subschedule i of size c is said to be prime if and only if it has at least one non-reducible root of size d for every $d < c$.*

Lemma 2. *Let i and j be subschedules that are formed by S. If $LJ_i = LJ_j$ and $ct_i^{(\alpha)} \leq ct_j^{(\alpha)}$ for every $\alpha \in S$, then for any subschedule l derived from j and formed by S', there exists a subschedule k that is derived from i and formed by S' such that $ms_k \leq ms_l$.*

Proof. Let us choose $k - i$ as the equivalent of $l - j$ where the machines are swapped such that the job that follows each $\alpha \in LJ_j$ is the same in both k and l. In this way, it is certain that $st_k^{(\alpha)} \leq st_l^{(\alpha)}$ and, as a result, $ct_k^{(\alpha)} \leq ct_l^{(\alpha)}$ for each job $\alpha \in S'$. Since makespan is the maximum of completion times, $ms_k \leq ms_l$.

Lemma 3. *Let i be a non-reducible subschedule of size c that is formed by S. If i does not have any non-reducible root of size $c-1$, then there exists a subschedule j formed by S such that j has a non-reducible root of size $c-1$, $LJ_i = LJ_j$, and $ct_j^{(\alpha)} \leq ct_i^{(\alpha)}$ for each job $\alpha \in S$.*

Proof. Let the roots of i formed by $S - \{\alpha\}$ and $S - \{\beta\}$ be j and k respectively. If both roots exist, and if j can be reduced by β, then k cannot be reduced by α. If this were not true, i would be reducible in the first place. In addition, if the $S - \{\gamma\}$ root, l, also exists and if k can be reduced by γ, then l would not be reducible by either α or β. Thus, in the case that all roots exist, there is at least one root that cannot be reduced by any other job in LJ_i.

If no non-reducible root of size $c-1$ exists, there must be a job $\alpha \in LJ_i$ such that any root of size $c-1$ except the one formed by $S - \{\alpha\}$ can be reduced by α, and the $S - \{\alpha\}$ root does not exist. This is only possible in the case that there is a job $\beta \in LJ_i$ such that α and β conflict and $st_i^{(\alpha)} < st_i^{(\beta)}$.

When all these conditions are satisfied, a subschedule j satisfying the lemma can be created by the following steps.

1. Reduce the $S - \{\beta\}$ root by α
2. Derive a new schedule from the result of step 1 by adding β

In the resulting subschedule, starting and completion times of α and β are smaller than in i, while other jobs remain the same.

Fig. 5: An example of the case explained in Lemma 3

The subschedule shown in Figure 5 (left) is non-reducible and does not have a non-reducible root of size 5. If $j5$ is removed, the result is not a valid root, and removing other jobs would give reducible roots. Changing the order of $j4$ and $j5$ gives a prime subschedule (right) that meets the conditions in Lemma 3.

Theorem 2. *There exists at least one optimal schedule that is prime.*

Proof. The first part of the proof is to prove that any optimal schedule is non-reducible. If a schedule i is reducible, it means that there is another schedule j

with the property $ms_i > ms_j$. Then i is not optimal, since it does not have the smallest possible makespan.

The second part is to prove that there exists an optimal schedule that has at least one non-reducible root of size c for every $c < n$. Assume that an optimal schedule i is not prime, and let d be the largest size for which i does not have a non-reducible root. Then i has a non-reducible root k of size $d+1$. By Lemma 3, there exists a subschedule l of size $d+1$ with property $ct_l^{(\alpha)} \leq ct_k^{(\alpha)}$ for each $\alpha \in LJ_k$, and l has a non-reducible root of size d. By Lemma 2, the remaining jobs can be added to l to obtain a schedule j such that $ms_j \leq ms_i$. Since i is optimal, $ms_j = ms_i$, and thus j is also optimal.

This shows that for each optimal schedule that is not prime, another optimal schedule can be obtained that has a non-reducible root for one missing size. By repeating this procedure for the remaining missing non-reducible root sizes, an optimal schedule with a non-reducible root for every size $c < n$ can be obtained. By Definition 8, such a schedule is prime, proving that there exists at least one optimal prime schedule.

Definition 9 (The Derivation Rule). *The derivation rule is a general rule to follow when creating subschedules from permutations of jobs. This rule is stated as: Insert the next job into the machine with the smallest processing time. If more than one machine shares the smallest processing time, insert the job to the one with the smallest index (for instance, to machine 1 instead of machine 2).*

The rule of selecting the machine with the lower index is just a convention. Another rule that favors the highest index for machines (or any other convention) would also be suitable, as long as one of the machines with the lowest processing time is selected.

Definition 10 (Representable Subschedule). *A subschedule that can be created from a permutation by following the derivation rule in Definition 9 is called representable.*

Lemma 4. *Any representable subschedule is prime.*

Proof. This lemma can be proven by induction.

- Representable subschedules of size 1 are created by adding one job to the first machine and starting it at time 0. If the job is moved to another machine, it would still start at time 0, so the subschedule is non-reducible and prime.
- Let i be a prime and representable subschedule of size c and j be the subschedule derived by adding job α to i following the derivation rule. In this case, α is already placed on the machine with the lowest processing time, so j cannot be reduced by α. Also, since i is non-reducible, it cannot be reduced by any job in LJ_i. Adding a new job would increase the processing time of one of the machines and does not change the reducibility of the new schedule by any of these jobs. Thus, j is also prime.

Theorem 3. *The set of all representable schedules includes at least one optimal schedule.*

Proof. We know that any representable subschedule is prime from Lemma 4. We can use this knowledge to form a permutation from any prime subschedule or its equivalent.

For a problem with m machines, a prime subschedule of size m has one job assigned to each machine. If such a subschedule i of size m is representable, then due to the convention of favoring machines with lower indices, it should have the property $st_i^{(\alpha)} \leq st_i^{(\beta)}$ for each pair of jobs α and β, where α is assigned to a machine with a lower index than β.

Let j be a prime subschedule with a representable root of size m. A permutation to represent j can be found by deleting the jobs step by step, by the following rules: *Delete the job that would give the non-reducible root with the highest minimum time, and if there exist two such jobs, delete the one that is assigned to the machine with a higher index.* This rule is the reverse of the derivation rule and the inverse of the permutation of the deleted jobs would give the permutation representation. If there is no representable size m root of j, an equivalent of j has such a root, so it is representable. Since there exists a prime optimal schedule by Theorem 2, this optimal schedule, or one of its equivalents, is representable. Therefore, the set of all representable schedules contains at least one optimal schedule.

5.4 Simulated Annealing for the Transaction Scheduling Problem

In Section 5.2, we show that TxnSP is an NP-hard problem. Therefore, finding optimal solutions using exact methods for even moderate-size TxnSP instances might incur high computational costs and may not be applicable in runtime scheduling scenarios. For this reason, we implemented a metaheuristic approach to find high-quality solutions to TxnSP within a limited execution time[4]. In this implementation, we exploit the permutation representation and search for an efficient solution among the set of representable schedules, which includes at least one optimal schedule as proven in Section 5.3.

Simulated annealing (SA) is a stochastic local search algorithm that probabilistically accepts inferior solutions to escape local optima [21]. It has been successfully applied to hard scheduling problems with permutation solution spaces, similar to TxnSP [35]. As a single-solution metaheuristic, it avoids the algorithmic overhead associated with maintaining and updating multiple candidate solutions or additional search structures, which are required by population-based or swarm intelligence algorithms, such as genetic algorithms, ant colony optimization, or particle swarm optimization [32]. With these characteristics, SA is suitable for real-time scheduling with strict time constraints, as in the case of transaction scheduling in the OptiMA framework.

In SA, the probability of accepting an inferior solution is controlled by a parameter called the *temperature* (T). Higher temperature values increase the

[4] `https://github.com/umutcalikyilmaz/TxnSP`

acceptance probability of inferior solutions, while gradually reducing T makes the algorithm prioritize exploitation over exploration in later iterations. In our implementation, we used a linear cooling schedule, which is defined as

$$T \leftarrow T - \alpha, \tag{2}$$

where the hyperparameter α determines the cooling rate. At each iteration, SA generates a new candidate solution from the neighborhood of the current solution, where the definition of a neighbor depends on the specific implementation. For our permutation representation, two solutions are considered neighbors if one can be obtained from the other by swapping a single pair of transactions.

6 Experimental Results

The premise of OptiMA is to ensure strong consistency and isolation while minimizing the associated performance overhead through transaction scheduling. To evaluate this premise, we developed the Factory Floor Benchmark (FFB), a synthetic workload designed to evaluate system performance under varying levels of resource contention and number of agents. FFB simulates a production facility that is operated by AI-driven robots. The facility consists of an assembly station and an inspection station. Each job is defined as a sequence of assembly operations, where each operation consists of multiple actions that must be executed atomically due to dependencies and exclusive resource requirements. After assembly, a job is transported to the inspection station and leaves the system after being inspected. The duration of each action is determined randomly using a normal distribution to imitate uncertainty in real-world execution times. Actions are simulated by awaiting threads. At initialization, job descriptions are created by generating random sequences of operations, each consisting of a random set of assembly actions. In total, nine such action types are defined (five manual, two drilling, and two welding).

The design of FFB follows the model design principles of OptiMA introduced in Section 4.1. Each tool represents a real-world resource that can be either shareable or non-shareable. The set of tools used in FFB is listed below.

Assembly Queue (Shareable) contains the assembly descriptions of parts.

Conveyor Belt (Shareable) stores assembled parts after processing.

Inspection Queue (Shareable) stores parts to be inspected.

Drill Press (Non-Shareable) performs drilling operations.

Welding Station (Non-Shareable) performs welding operations.

QA Scanner (Non-Shareable) performs quality assessments of parts.

Output Bin (Shareable) stores completed parts after inspection.

The agent roles in FFB are defined below.

Assembler assembles parts by performing manual work, drilling, and welding, and then places the assembled parts on the conveyor belt.

Table 1: Conflict configurations used in experiments

Conflict Levels	**Operation Propensities**		
	Manual	Drilling	Welding
Very Low	20	1	1
Low	10	1	1
High	5	1	1
Very High	1	1	1

Table 2: Agent intensity configurations used in the experiments

Agent Intensities	**Initial Numbers**		
	Assembler	Transporter	Inspector
Low	100	100	100
Medium	200	200	200
High	400	400	400

Transporter transfers assembled parts from conveyor belt to the inspection station and inserts them into the inspection queue.

Inspector evaluates the quality of assembled jobs using the QA scanner, and reports the results to the floor manager.

Floor Manager monitors the number of jobs at each station and dynamically adjusts the number of active agents of each role to balance queue lengths.

To assess the effect of transaction scheduling under various contention levels and numbers of agents, we conducted experiments using different conflict levels and agent intensity settings as shown in Table 1 and Table 2, respectively. Increasing the propensities of drilling and welding operations relative to manual operations increases the conflict level, since these operations rely on non-shareable tools. For each intensity setting, the maximum number of agents for each role is set to twice the initial numbers. We also used three different numbers of available threads (8, 16, 32), resulting in a total of 36 configurations. For each configuration, five random sets of jobs are generated. The system is first executed using each set of jobs without scheduling to establish a baseline. Then it is executed multiple times on the same job sets with scheduling to evaluate different scheduling parameters. Specifically, we considered three batch sizes (50, 100, 200) and two timeout values (500 ms and 1000 ms), which define the batch size threshold and the maximum scheduling delay, respectively. For each configuration, the best set of scheduling parameters is selected based on average throughput over five runs. Throughput is defined by the number of completed jobs in the simulation per second. Optimization was performed using SA with the initial temperature and cooling rate set to 1.2 and 0.005, respectively. Table 3 presents the benchmark results showing the best scheduling parameters for each configuration, and the minimum, maximum, and average improvements observed for those parameters over five runs.

Experimental results show that transaction scheduling significantly increases throughput in almost all test scenarios. In the best-performing configuration, the average and the maximum improvements reach as high as 37.73% and 45.09%, respectively. These results show that transaction scheduling is an effective approach for mitigating the performance overhead introduced by enforcing strict consistency and isolation in a MAS. Furthermore, the improvement in system performance shows a strong positive correlation with the number of agents. This indicates that the proposed approach becomes increasingly effective as the system

Table 3: Factory Floor Benchmark Results for OptiMA Framework

Model Configurations			Execution Results						
			Non-Optimized	Optimized					
Agent Intensity	Conflict Level	Thread Number	Average Throughput	Best Configuration		Average Throughput	Minimum Improvement	Maximum Improvement	Average Improvement
				Batch Size	Timeout				
Low	Very Low	8	4.055	100	1000 ms	4.280	5.09%	6.03%	5.55%
		16	7.840	200	1000 ms	8.304	4.70%	7.37%	5.93%
		32	13.704	100	500 ms	13.557	-2.55%	0.82%	-1.05%
	Low	8	3.875	50	1000 ms	4.140	5.66%	7.73%	6.86%
		16	6.800	100	500 ms	7.051	2.14%	4.62%	3.68%
		32	7.993	100	500 ms	8.047	-3.01%	2.65%	0.64%
	High	8	3.428	100	1000 ms	3.637	3.87%	7.60%	6.11%
		16	4.622	200	1000 ms	4.691	0.69%	2.19%	1.49%
		32	4.635	100	500 ms	4.622	-1.40%	2.17%	-0.24%
	Very High	8	2.008	100	500 ms	2.039	0.42%	2.45%	1.53%
		16	2.005	50	1000 ms	2.039	0.82%	2.60%	0.98%
		32	2.048	200	500 ms	2.051	-1.02%	1.07%	0.18%
Medium	Very Low	8	3.248	50	500 ms	3.687	10.62%	17.46%	13.53%
		16	6.275	50	500 ms	7.390	14.62%	19.74%	17.76%
		32	10.802	200	500 ms	12.619	15.57%	17.56%	16.82%
	Low	8	3.048	50	500 ms	3.621	17.59%	21.62%	18.79%
		16	5.418	100	1000 ms	6.384	15.14%	21.21%	17.84%
		32	6.984	100	1000 ms	7.350	2.81%	11.08%	5.32%
	High	8	2.611	200	500 ms	3.246	22.33%	25.52%	24.34%
		16	3.751	50	1000 ms	4.098	6.54%	13.76%	9.31%
		32	3.890	50	1000 ms	4.138	4.34%	7.59%	6.41%
	Very High	8	1.619	100	500 ms	1.836	10.36%	15.92%	13.39%
		16	1.697	200	1000 ms	1.816	2.62%	8.95%	7.06%
		32	1.748	100	500 ms	1.890	6.86%	9.63%	8.12%
High	Very Low	8	3.080	50	1000 ms	3.638	16.80%	20.27%	18.13%
		16	5.639	50	500 ms	7.353	28.49%	34.24%	30.41%
		32	9.350	50	1000 ms	12.384	29.95%	35.68%	32.45%
	Low	8	2.853	200	500 ms	3.549	21.83%	25.85%	24.41%
		16	4.743	200	500 ms	6.413	32.77%	37.47%	35.02%
		32	6.580	100	1000 ms	7.228	8.02%	12.58%	9.87%
	High	8	2.384	50	500 ms	3.281	33.46%	45.09%	37.73%
		16	3.354	50	1000 ms	4.197	14.71%	22.94%	18.80%
		32	3.787	100	1000 ms	4.182	8.00%	14.34%	10.49%
	Very High	8	1.452	50	1000 ms	1.828	24.51%	27.31%	25.94%
		16	1.670	50	1000 ms	1.817	7.51%	10.47%	8.84%
		32	1.690	200	500 ms	1.870	9.85%	12.03%	10.67%

scales, making it particularly useful for the target set of complex and large-scale MASs. The relationship between throughput improvement, conflict level, and the number of available threads is less regular. The observed improvements do not exhibit a monotonic trend across these parameters. One possible explanation is that the impact of scheduling depends on the gap between the maximum achievable parallelism and the average parallelism obtained by random transaction ordering, and this gap varies non-linearly with respect to resource contention and the number of threads. A secondary observation is that the optimal batch size shows a weak correlation with the number of available threads. Finally, no discernible trend is observed according to the optimal timeout values.

7 Conclusion and Future Research Directions

Multi-agent systems have the potential to undertake a wide range of sophisticated tasks that were previously considered impossible to automate. However, modern systems developed for this purpose are becoming increasingly complex and large-scale, making them more susceptible to inconsistent states. Designing such models in a fault-tolerant manner requires well-designed organization and coordination mechanisms. Most prior work in the field relies on semantic structures for coordination, with limited attention to the execution layer. In this study, we address this gap by offering a model design and execution strategy that enforces strong consistency and isolation at the execution layer. Furthermore, we show that the potential performance trade-off associated with such an approach can be largely mitigated through scheduling.

As future work, one natural next step is to apply the proposed framework to complex real-world scenarios. In addition, the theoretical work and the proposed metaheuristic approach for the transaction scheduling problem provide a basis for future research on improving the performance of transaction-based systems, such as database management systems.

Acknowledgments. This work is funded by the German Federal Ministry of Research, Technology and Space within the funding program quantum technologies — from basic research to market — contract number 13N16090.

Bibliography

[1] Baker, B.S., Coffman Jr, E.G.: Mutual exclusion scheduling. Theoretical Computer Science **162**(2), 225–243 (1996)

[2] Bernstein, P.A., Goodman, N.: Multiversion concurrency control—theory and algorithms. ACM Transactions on Database Systems (TODS) **8**(4), 465–483 (1983)

[3] Bittner, T., Groppe, S.: Avoiding blocking by scheduling transactions using quantum annealing. In: Proceedings of the 24th Symposium on International Database Engineering & Applications. pp. 1–10 (2020)

[4] Çalıkyılmaz, U., Groppe, S., Groppe, J., Winker, T., Prestel, S., Shagieva, F., Arya, D., Preis, F., Gruenwald, L.: Opportunities for quantum acceleration of databases: optimization of queries and transaction schedules. Proceedings of the VLDB Endowment **16**(9), 2344–2353 (2023)

[5] Cao, Y., Fan, W., Ou, W., Xie, R., Zhao, W.: Transaction scheduling: From conflicts to runtime conflicts. Proceedings of the ACM on Management of Data **1**(1), 1–26 (2023)

[6] Chen, W., Su, Y., Zuo, J., Yang, C., Yuan, C., Chan, C.M., Yu, H., Lu, Y., Hung, Y.H., Qian, C., et al.: Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In: The Twelfth International Conference on Learning Representations (2023)

[7] Chopra, A.K., Singh, M.P.: Specifying and applying commitment-based business patterns. In: The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2. pp. 475–482 (2011)

[8] Dorndorf, U., Pesch, E., Phan-Huy, T.: Constraint propagation techniques for the disjunctive scheduling problem. Artificial intelligence **122**(1-2), 189–240 (2000)

[9] Durfee, E.H.: Distributed problem solving and planning. In: ECCAI Advanced Course on Artificial Intelligence, pp. 118–149. Springer (2001)

[10] Ferber, J., Gutknecht, O.: A meta-model for the analysis and design of organizations in multi-agent systems. In: Proceedings international conference on multi agent systems (Cat. No. 98EX160). pp. 128–135. IEEE (1998)

[11] Georgeff, M.P., Rao, A.: An abstract architecture for rational agents. In: Proc. of the Third International Conference on Principles of Knowledge Representation and Reasoning. pp. 439–449. Morgan Kaufmann Publishers Inc San Francisco, CA, USA (1992)

[12] Gerber, C., Siekmann, J., Vierke, G.: Holonic multi-agent systems (1999)

[13] Gilbert, S., Lynch, N.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. Acm Sigact News **33**(2), 51–59 (2002)

[14] Gray, J., Reuter, A.: Transaction processing: concepts and techniques. Elsevier (1992)

[15] Gray, J., et al.: The transaction concept: Virtues and limitations. In: VLDB. vol. 81, pp. 144–154 (1981)

[16] Groppe, S., Groppe, J.: Optimizing transaction schedules on universal quantum computers via code generation for grover's search algorithm. In: Proceedings of the 25th International Database Engineering & Applications Symposium. pp. 149–156 (2021)
[17] Härder, T.: Observations on optimistic concurrency control schemes. Information Systems **9**(2), 111–120 (1984)
[18] Hill, R., Polovina, S., Shadija, D.: Transaction agent modelling: From experts to concepts to multi-agent systems. In: International Conference on Conceptual Structures. pp. 247–259. Springer (2006)
[19] Hou, X., Zhao, Y., Wang, S., Wang, H.: Model context protocol (mcp): Landscape, security threats, and future research directions. ACM Transactions on Software Engineering and Methodology (2025)
[20] Kim, J., Kim, Y., Lee, Y., Byun, H.: Finai data assistant: Llm-based financial database query processing with the openai function calling api. arXiv preprint arXiv:2510.14162 (2025)
[21] Kirkpatrick, S., Gelatt Jr, C.D., Vecchi, M.P.: Optimization by simulated annealing. science **220**(4598), 671–680 (1983)
[22] Kleppmann, M.: Designing Data-Intensive Applications. O'Reilly Media (2017)
[23] Mavroudis, V.: Langchain v0. 3 (2024)
[24] Papazoglou, M.P.: Service-oriented computing: Concepts, characteristics and directions. Proceedings of the Fourth International Conference on Web Information Systems Engineering (2003)
[25] Prasaad, G., Cheung, A., Suciu, D.: Handling highly contended oltp workloads using fast dynamic partitioning. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. pp. 527–542 (2020)
[26] Ramakrishnan, R., Gehrke, J.: Database management systems. McGraw-Hill, Inc. (2002)
[27] Ramamohanarao, K., Bailey, J.: Transaction oriented computational models for multi-agent systems. In: Proceedings 13th IEEE International Conference on Tools with Artificial Intelligence. ICTAI 2001. pp. 11–17. IEEE (2001)
[28] Sheng, Y., Tomasic, A., Zhang, T., Pavlo, A.: Scheduling oltp transactions via machine learning. arXiv preprint arXiv:1903.02990 (2019)
[29] Singh, M.P.: A social semantics for agent communication languages. In: Issues in agent communication, pp. 31–45. Springer (2000)
[30] Singh, M.P., Chopra, A.K., et al.: Argus: Programming with communication protocols in a belief-desire-intention architecture. Artificial Intelligence p. 104398 (2025)
[31] Singh, M.P., Christie V, S.H., Chopra, A.K.: Langshaw: declarative interaction protocols based on sayso and conflict. In: Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence. pp. 202–210 (2024)
[32] Talbi, E.G.: Metaheuristics: From Design to Implementation. Wiley (2009)
[33] Talebirad, Y., Nadiri, A.: Multi-agent collaboration: Harnessing the power of intelligent llm agents. arXiv preprint arXiv:2306.03314 (2023)

[34] Thomasian, A., Ryu, I.K.: Performance analysis of two-phase locking. IEEE transactions on software engineering **17**(5), 386 (1991)
[35] Van Laarhoven, P.J., Aarts, E.H., Lenstra, J.K.: Job shop scheduling by simulated annealing. Operations research **40**(1), 113–125 (1992)
[36] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q.V., Zhou, D., et al.: Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems **35**, 24824–24837 (2022)
[37] Xu, F.F., Song, Y., Li, B., Tang, Y., Jain, K., Bao, M., Wang, Z.Z., Zhou, X., Guo, Z., Cao, M., et al.: Theagentcompany: benchmarking llm agents on consequential real world tasks. arXiv preprint arXiv:2412.14161 (2024)
[38] Xu, H., Li, C., Ma, X., Ou, X., Zhang, Z., He, T., Liu, X., Wang, Z., Liang, J., Chu, Z., et al.: The evolution of tool use in llm agents: From single-tool call to multi-tool orchestration. arXiv preprint arXiv:2603.22862 (2026)
[39] Yang, T., Feng, P., Guo, Q., Zhang, J., Zhang, X., Ning, J., Wang, X., Mao, Z.: Autohma-llm: Efficient task coordination and execution in heterogeneous multi-agent systems using hybrid large language models. IEEE Transactions on Cognitive Communications and Networking **11**(2), 987–998 (2025)
[40] Yin, Z., Sun, Q., Chang, C., Guo, Q., Dai, J., Huang, X., Qiu, X.: Exchange-of-thought: Enhancing large language model capabilities through cross-model communication. arXiv preprint arXiv:2312.01823 (2023)
[41] Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: The gaia methodology. ACM Transactions on Software Engineering and Methodology (TOSEM) **12**(3), 317–370 (2003)
[42] Zhang, T., Tomasic, A., Sheng, Y., Pavlo, A.: Performance of oltp via intelligent scheduling. In: 2018 IEEE 34th International Conference on Data Engineering (ICDE). pp. 1288–1291. IEEE (2018)