A Survey of Cross-Domain Optimization and Hybrid-Parallel Distributed Training in Large-Scale Distributed Systems

Abstract

In the realm of computer science and engineering, optimizing large-scale distributed systems necessitates a sophisticated approach that integrates cross-domain optimization with hybrid-parallel strategies. This survey paper presents a comprehensive exploration of methodologies that enhance performance and scalability by addressing computational and network cost awareness. By leveraging hybrid-parallel distributed training, systems can mitigate communication overhead and improve resource allocation efficiency. Key findings highlight the significance of innovative algorithmic strategies, such as the Synchronization-Avoiding Block Coordinate Descent method and the HeteroG framework, which optimize parallelism and device placements. Furthermore, the survey identifies challenges in computational-network cost management, emphasizing the need for advanced communication protocols and adaptive synchronization techniques to enhance efficiency. Real-world applications, including the ADCME-MPI framework for complex numerical simulations and the Nebula-I framework for collaborative deep learning, demonstrate the transformative potential of these strategies. Future research directions suggest refining task mapping, integrating emerging technologies like quantum computing, and developing robust security mechanisms. These advancements promise to further enhance the adaptability and performance of distributed systems, paving the way for more efficient and scalable computational frameworks. Collectively, this survey underscores the critical role of integrating cross-domain optimization and hybrid-parallel strategies to address the dynamic challenges of large-scale distributed environments.

1 Introduction

1.1 Significance of Optimization in Large-Scale Distributed Systems

Optimization techniques are crucial for enhancing the performance and scalability of large-scale distributed systems, which are vital for executing complex computational tasks across various domains. In hybrid-parallel training, for instance, fail-slow conditions can significantly degrade performance, necessitating robust strategies to maintain efficiency [1]. Addressing resource usage prediction in parallel logic programs can markedly improve system performance and scalability [2].

In high-performance computing, optimal control methods are essential for managing computational resources effectively [3]. Traditional first-order optimization algorithms often require numerous sequential iterations to converge, highlighting the demand for accelerated methods to address complex tasks efficiently [4]. Machine learning algorithms applied to large datasets frequently struggle without scalable optimization techniques to enhance efficiency [5].

Existing clustering algorithms like K-means demonstrate limitations in scalability and computational complexity, underscoring the necessity for optimization in big data environments [6]. Optimizing

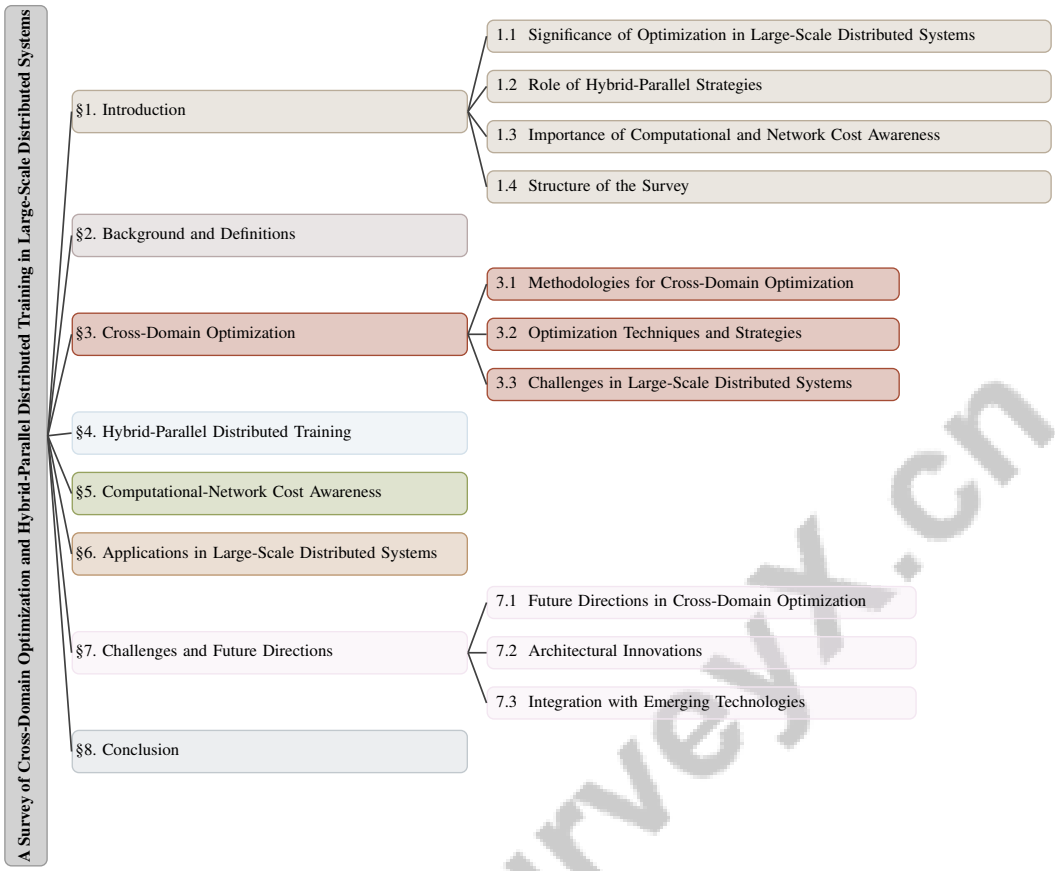


Figure 1: chapter structure

distributed training in heterogeneous GPU clusters is critical due to the varying capabilities of GPUs and network conditions, where traditional methods fall short [7]. In low-bandwidth scenarios, optimization techniques are vital for maintaining performance and scalability [8].

Load balancing inefficiencies during concurrent job execution on shared computer clusters further emphasize the importance of optimization for system performance enhancement [9]. The execution of complex, burst-parallel, directed acyclic graph (DAG) jobs in serverless frameworks presents unique challenges that necessitate innovative methods for enhancing scalability, minimizing data movement, and optimizing resource management [10].

Optimizing analytics jobs in large-scale distributed systems is critical as it directly affects performance and scalability [11]. Furthermore, existing programming models often inadequately support modern distributed applications, particularly in managing consistency during network partitions [12]. Optimization techniques are also pivotal in distributed computation for efficiently reconstructing images from large datasets in computed tomography [13]. In the context of Cloud RAN (C-RAN), virtualization and centralization of real-time network functions necessitate improved fronthaul capacity and latency reduction, underscoring the need for effective optimization in network service deployment [14].

These optimization techniques are essential for overcoming challenges associated with complex computational demands and resource management, driving progress in high-performance computing and distributed systems. They enable the effective utilization of advanced parallel algorithms and machine learning methods to optimize software execution across various domains, including operations research, power systems, and logistics. As the scale and complexity of these systems grow, particularly with the integration of cloud computing and IoT, sophisticated optimization strategies become increasingly vital for ensuring timely and efficient computations [15, 16, 17].

1.2 Role of Hybrid-Parallel Strategies

Hybrid-parallel strategies are pivotal for enhancing the efficiency and scalability of distributed training systems by integrating data and model parallelism, which mitigates communication overhead and improves performance. The Synchronization-Avoiding Block Coordinate Descent (SA-BCD) method exemplifies this by effectively reducing synchronization needs in first-order methods [18]. Similarly, iterative algorithms based on classical additive Schwarz methods illustrate the benefits of hybrid-parallel approaches in partitioned neural networks, minimizing communication costs [19].

In distributed evolutionary computation, hybrid-parallel strategies utilize parallel processing and cooperation among distributed agents to enhance computational efficiency, highlighting their significance across diverse applications [20]. The SDP framework introduces a semi-decentralized approach that decentralizes communication for model synchronization while centralizing group scheduling, thereby improving performance and convergence speed [21].

TensorFlow's dataflow graph model showcases the versatility of hybrid-parallel strategies, facilitating the expression of computations and supporting various parallelism and device types [22]. The HeteroG framework exemplifies these strategies by converting single-GPU models into distributed formats, optimizing parallelism strategies, device placements, and gradient communication methods [7]. Nebula-I, a unified framework for collaborative training of deep learning models across remote clusters, employs parameter-efficient strategies and adaptive communication techniques, underscoring the importance of hybrid-parallel approaches in distributed training [8].

The integration of genetic algorithm models for GPU implementation demonstrates the scalability and processing speed benefits of hybrid-parallel strategies when handling massive datasets [23]. The DiCOM solution, which incorporates direct communication between migrated Open-MPI processes, further illustrates the role of hybrid-parallel strategies in reducing communication overhead [9]. Additionally, the proposed method combining in-memory caching with multi-query optimization highlights the role of hybrid-parallel strategies in improving resource utilization and performance in distributed training environments [11].

In large-scale distributed systems, hybrid-parallel strategies are integral to managing computational and network resources, significantly enhancing the scalability and efficiency of distributed training environments. They provide robust frameworks for optimizing resource distribution and communication, contributing to improved system performance. Furthermore, the exploration of programming models such as Lasp and Austere allows developers to make explicit trade-offs between consistency and availability, facilitating the implementation of hybrid-parallel strategies in diverse applications [12].

1.3 Importance of Computational and Network Cost Awareness

Awareness of computational and network costs is critical for optimizing large-scale distributed systems, where balancing computation and communication is essential for efficiency and scalability. In distributed training environments, particularly those utilizing data parallelism, understanding the intricate balance between computation and communication is vital for enhancing training efficiency [24]. The inefficiencies in hyperparameter optimization, which lead to prolonged training times for each configuration, further underscore the importance of computational and network cost awareness [24].

Research highlights the significance of recognizing computational and network costs, especially in decentralized job scheduling within Grid systems [25]. Key challenges include the complexity of parallelizing algorithms, managing data distribution, and ensuring efficient communication among distributed systems [26]. High communication costs and the absence of a global perspective are critical issues in distributed optimization challenges [20].

Benchmarks address the challenges of efficiently training and deploying machine learning models on heterogeneous distributed systems that require large-scale computations [22]. Existing parallelism strategies often fail to effectively utilize available heterogeneous resources, resulting in imbalanced processing speeds and inefficient communication [7]. Limited bandwidth between clusters significantly impacts overall training speed and model convergence in large-scale distributed systems [8].

Increased communication overhead between heavily communicating processes emphasizes the necessity of being aware of computational costs [9]. The need to optimize multiple concurrent queries in distributed systems to avoid redundant computations further illustrates the importance of computational cost awareness [11]. High condition numbers of discretization operators for high-order discretizations complicate efficiency and convergence, reinforcing the need for computational cost awareness [27].

Efficient training of DNNs coupled with PDEs involves managing significant memory and computational challenges, highlighting the necessity for awareness of computational costs [28]. The iterative nature of topology optimization (TO) incurs high computational costs, requiring numerous physical simulations to compute gradient information [29]. Memory bottlenecks arise as each device must hold a complete model replica during data parallel training, leading to inefficient memory utilization and increased communication overhead [30].

Innovations such as TABS achieve near-optimal service elasticity in a distributed manner while maintaining low communication overhead, contrasting with traditional centralized methods that can induce inefficiencies [31]. Addressing the inefficiencies of existing hyperparameter optimization methods in utilizing parallel resources effectively is crucial for reducing long training times and optimizing resource allocation [32]. The rising energy demands of data centers necessitate new scheduling methods that consider energy costs in large-scale distributed systems [33].

The pretraining process of large foundation models is both memory- and communication-intensive, underscoring the importance of computational and network cost awareness [34]. Algorithms relying on Gauss-Seidel or sequential updates are often unsuitable for distributed parallel computations, leading to excessive computational times [35]. The Hydrozoa framework's reliance on indirect communication methods increases latency, further emphasizing the significance of computational and network cost awareness [36]. The high costs associated with interprocessor communication critically affect the performance of existing optimization methods, particularly as the number of processors increases [18].

1.4 Structure of the Survey

This survey is meticulously organized to provide a comprehensive exploration of cross-domain optimization and hybrid-parallel distributed training in large-scale distributed systems. It begins with an introduction that emphasizes the critical importance of optimization techniques in enhancing system performance and scalability, highlighting the essential role of hybrid-parallel strategies and the necessity for awareness of computational and network costs. This context is vital, as advancements in parallel computing capabilities and the application of meta-heuristics and machine learning are transforming optimization across diverse fields, including operations research, software optimization for parallel computing systems, and large-scale data processing. By addressing the complexities of parameter selection and resource sharing, these techniques enhance efficiency and pave the way for innovative research directions in optimization methodologies [37, 11, 16, 17]. Following the introduction, the paper delves into the background and definitions, offering a thorough overview of core concepts such as cross-domain optimization, hybrid-parallel distributed training, and computational-network cost awareness.

Subsequent sections are dedicated to detailed discussions on key topics. The section on cross-domain optimization explores various methodologies and strategies, addressing challenges faced in implementing such optimization in large-scale systems. The hybrid-parallel distributed training section examines algorithmic strategies, communication, and synchronization aspects critical for enhancing scalability and efficiency. The survey then shifts focus to computational-network cost awareness, analyzing the importance of optimizing these costs and discussing techniques for cost optimization.

The latter part of the survey highlights real-world applications and case studies, demonstrating the successful implementation of the discussed strategies and their impact on efficiency and scalability. The survey concludes with an exploration of current challenges and future directions, identifying potential research avenues and technological advancements that could address existing challenges. This structured approach fosters a comprehensive understanding of the subject, equipping researchers and practitioners with critical insights into the latest advancements in distributed deep learning techniques, such as hybrid-parallel training and adaptive clustering algorithms, essential for opti-

mizing memory usage and computational efficiency in large-scale machine learning applications [38, 26, 39, 40, 5].The following sections are organized as shown in Figure 1.

2 Background and Definitions

2.1 Core Concepts in Parallel and Distributed Computing

Parallel and distributed computing paradigms are fundamental for enhancing the performance and scalability of complex computational tasks by concurrently utilizing multiple processors or computers. These paradigms are underpinned by theoretical models like the Parallel Random Access Machine (PRAM) and practical frameworks such as OpenMP and MPI for real-world parallel algorithm implementation [41]. The shift towards implicit parallel programming in MATLAB exemplifies the optimization of resource utilization [42]. The hybrid computing model (HCM) further illustrates the integration of CPU and GPU resources to boost performance in parallel and distributed settings [43].

In distributed systems, middleware architectures are crucial for resource sharing at scale, as seen in Grid computing environments [44]. The SimGrid framework provides versatile and reproducible simulation tools for exploring large-scale distributed system scenarios [45]. The Distributed Kernel Ridge Regression (DKRR) method showcases the use of parallel computing to enhance statistical performance by distributing data across multiple machines [46].

Hybrid Federated Split Learning (HFSL) combines federated and split learning to optimize training time and energy consumption through adaptive model splitting and resource allocation, highlighting the importance of resource management in distributed environments [47]. The Least Loaded (LL) routing algorithm demonstrates network routing optimization strategies in distributed systems [48]. Preemptive process migration, a critical concept, involves transferring processes between nodes for load balancing and improved resource utilization [9].

These core concepts and terminologies form the foundation of parallel and distributed computing, facilitating effective management and execution of tasks across large-scale systems. The integration of theoretical models, practical programming paradigms, and sophisticated optimization strategies is essential for advancing computational capabilities and efficiency. This approach fosters innovative solutions to complex computational challenges in fields such as operations research, finance, and machine learning. Leveraging modern advancements in parallel computing and machine learning, researchers can address intricate optimization problems more effectively, thereby advancing high-performance computing and enhancing the understanding of these technologies' governing principles [49, 16, 17, 26, 50].

3 Cross-Domain Optimization

Category	Feature	Method
Methodologies for Cross-Domain Optimization	Constraint and Subspace Strategies	N-I[8], GN[51]
	Resource and Communication Optimization	DiCOM[9], HeteroG[7]
	Parallelization Techniques	OptEx[4], NPDC[52], DMA[14]
	Dynamic and Adaptive Methods	DySpGEMM[53], N/A[12]
Optimization Techniques and Strategies	Sample and Batch Selection	q-KG[54], CSS-Big-means[6]
	Evolutionary Principles	JMP[55]
	Gradient and Communication Coordination	HD[56], AdaCons[57]
Challenges in Large-Scale Distributed Systems	Optimization Techniques	DA[58], SA-BCD[18], EGD[59], VCSCHED[60], DIANA[25], CMQO[11], CSGD[13]
	Scheduling Strategies	S&C[61], SJSO[62], MPS-NE[63], SP[64], Wukong[10]
	Communication Efficiency	ADG[65], GC-MMC[66]
	Iterative Evaluation	GSP[67]

Table 1: This table provides a comprehensive overview of methodologies and techniques employed in cross-domain optimization, categorized into three primary areas: methodologies for cross-domain optimization, optimization techniques and strategies, and challenges in large-scale distributed systems. Each category is further detailed with specific features and methods, highlighting the diversity and complexity of approaches used to enhance performance and resource management in distributed computing environments.

Exploring cross-domain optimization methodologies is essential for enhancing performance and resource management in distributed systems. This section delves into specific methodologies that

underpin cross-domain optimization, emphasizing their principles and innovative strategies to improve computational efficiency. Table 1 presents a detailed classification of various methodologies and techniques pertinent to cross-domain optimization, emphasizing their significance in addressing computational efficiency and resource utilization challenges in distributed systems. Additionally, Table 4 presents a comparative overview of various methodologies utilized in cross-domain optimization, focusing on their optimization strategies, method types, and resource utilization. As illustrated in **??**, the hierarchical structure of cross-domain optimization categorizes methodologies into hybrid computing models, optimization frameworks, and deep learning enhancements. The figure further delineates optimization techniques into frameworks and algorithms, gradient-based optimization, and system reliability strategies. Additionally, it outlines challenges faced in large-scale distributed systems, highlighting computational, resource allocation, and infrastructural issues that must be addressed to achieve optimal performance.

3.1 Methodologies for Cross-Domain Optimization

Method Name	Workload Distribution	Optimization Techniques	Computational Frameworks
HCM[43]	Partition Tasks	Hybrid Models	Hybrid Platforms
NPDC[52]	Multiple Cores	Decentralized Algorithms	Not Mentioned
OptEx[4]	Parallel Computing	Kernelized Gradient Estimation	Optex
CSS-Big-means[6]	Parallel Processing	Competitive Optimization Strategy	-
HeteroG[7]	Resource Allocation	Hybrid Parallelism Approach	Automated Framework
N-I[8]	Allocate Resources	Nebula-Optimizer	Nebula-I
AdaCons[57]	Gradient Aggregation	Adaptive Consensus Approach	Aggregation Framework
DiCOM[9]	Process Migration	Gossip Algorithm	Open-MPI Mosix
GC-MMC[66]	Task Allocation	Gradient Coding	Gc-MMC
DySpGEMM[53]	Mpi Processes	Batch-dynamic Approach	3D Spgemm
CMQO[11]	In-memory Caching	Cost-based Optimization	Distributed Computing Frameworks
N/A[12]	-	Declarative Approach	-
CSGD[13]	Parallel Computation	Decentralized Algorithms	Parallel Computing Architectures
DMA[14]	Global Scheduler	Statistical Multiplexing	Oai-based Testbed
GN[51]	-	Second-order Techniques	-
JMP[55]	-	Metaheuristic Algorithms	Jmetalpy

Table 2: Overview of methodologies for cross-domain optimization, detailing workload distribution, optimization techniques, and computational frameworks across various methods. This table highlights the diversity of approaches employed to enhance performance and resource utilization in distributed systems. Each method is associated with specific strategies and frameworks, illustrating the complexity and adaptability required in optimization tasks.

Cross-domain optimization employs diverse methodologies to improve performance and resource utilization in complex distributed environments. The hybrid computing model, which partitions tasks between CPU and GPU resources, exemplifies effective workload distribution by leveraging the strengths of each processor type [43]. This model is particularly suited for tasks divisible into smaller, independent sub-problems, as demonstrated by the Naturally Parallelizable Divide-and-Conquer (NPDC) method, which enables efficient parallel optimization [52].

The OptEx framework utilizes kernelized gradient estimation for approximate parallelization of first-order optimization [4]. The Competitive Sample Size Big-means Algorithm (CSS-Big-means) dynamically optimizes sample sizes, enhancing clustering efficiency in distributed systems [6]. In distributed training, HeteroG adapts single-GPU training models for heterogeneous GPUs by managing resource allocation and communication [7].

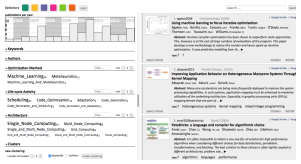
Nebula-I enhances deep learning model training across low-bandwidth cloud clusters through optimization layers that address network constraints [8]. The Adaptive Consensus Gradients Aggregation (AdaCons) method formulates gradient aggregation as a subspace optimization problem, using adaptive coefficients to improve convergence rates [57].

The DiCOM module optimizes performance by reducing communication overhead through direct process communication [9]. The GC-MMC method enhances efficiency by allowing workers to send multiple coded partial gradients per iteration [66]. The batch-dynamic algorithm for MPI-based parallel computing enables fast updates to sparse matrices, showcasing optimization potential in dynamic environments [53].

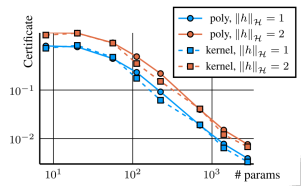
Cache-based Multi-query Optimization (CMQO) optimizes resource usage by identifying common sub-expressions across queries [11]. Spry's declarative specifications enable adaptive management of consistency and availability based on application requirements [12].

The Coordinate-Reduced Steepest Gradient Descent (CSGD) algorithm minimizes residuals of linear inverse problems, facilitating parallel computation on arbitrary subsets of observations [13]. Dynamic multi-threading allows parallel execution of encoding and decoding functions on multi-core platforms, addressing high fronthaul capacity and tight latency requirements [14].

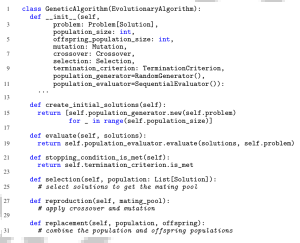
These methodologies highlight the necessity of advanced computational techniques for achieving cross-domain optimization, enhancing performance and resource utilization across distributed systems. The integration of diverse strategies—including hybrid computing models, decentralized algorithms, and adaptive methodologies—illustrates the extensive techniques employed to optimize complex environments, particularly in parallel computing systems. This includes leveraging machine learning and meta-heuristics to navigate software optimization challenges, alongside advancements in parallel optimization frameworks that unify various methodologies and application domains. Such comprehensive approaches are vital for maximizing efficiency and performance across fields like operations research, finance, and logistics [16, 17]. Table 2 provides a comprehensive examination of various methodologies utilized in cross-domain optimization, highlighting their distinct approaches to workload distribution, optimization techniques, and computational frameworks.



(a) Research in Machine Learning and Optimization Methods[16]



(b) Performance comparison of two methods for computing a polynomial certificate.[51]



```python
class GeneticAlgorithm(EvolutionaryAlgorithm):
    def __init__(self,
                 problem: Problem[Solution],
                 population_size: int,
                 offspring_population_size: int,
                 mutation: Mutation,
                 crossover: Crossover,
                 selection: Selection,
                 termination_criterion: TerminationCriterion,
                 population_generator=RandomGenerator(),
                 population_evaluator=SequentialEvaluator()):
        ...

    def create_initial_solutions(self):
        return [self.population_generator.new(self.problem)
                for _ in range(self.population_size)]

    def evaluate(self, solutions):
        return self.population_evaluator.evaluate(solutions, self.problem)

    def stopping_condition_is_met(self):
        return self.termination_criterion.is_met

    def selection(self, population: List[Solution]):
        # select solutions to get the mating pool

    def reproduction(self, mating_pool):
        # apply crossover and mutation

    def replacement(self, population, offspring):
        # combine the population and offspring populations
```

(c) A Python code snippet for a Genetic Algorithm class[55]

Figure 2: Examples of Methodologies for Cross-Domain Optimization

As shown in Figure 2, the example on "Cross-Domain Optimization: Methodologies for Cross-Domain Optimization" provides an overview of various approaches and tools utilized in optimization across different fields. The first image showcases a research database interface that highlights publications in machine learning and optimization methods, revealing trends through a bar chart categorizing publications per year. This visual representation underscores the growing interest and advancements in these areas. The second image presents a comparative analysis of two methods for computing polynomial certificates, illustrating performance differences between polynomial and kernel methods based on parameter counts. This comparison is pivotal for understanding the effectiveness of various computation techniques. Lastly, the third image introduces a Python code snippet for a Genetic Algorithm class, demonstrating the practical implementation of evolutionary algorithms. This class, with its array of methods, offers insight into how genetic algorithms can be structured in a programming environment. Collectively, these examples highlight the multifaceted nature of cross-domain optimization and the methodologies employed to tackle complex problems across various domains [16, 51, 55].

3.2 Optimization Techniques and Strategies

Method Name	Performance Enhancement	System Adaptability	Algorithmic Diversity
HD[56]	Training Efficiency	Heterogeneous Clusters	Dynamic Programming Algorithm
AdaCons[57]	Adaptive Consensus Approach	Dynamically Weighting Gradients	Subspace Optimization Problem
GC-MMC[66]	Iteration Completion Times	Resource Utilization	Gradient Coding Methods
CSS-Big-means[6]	Competitive Optimization Strategy	Adaptive Sampling Strategy	Dynamic Sample Size
q-KG[54]	Parallel Evaluations	Dynamic Adjustment	Bayesian Optimization Algorithms
JMP[55]	Dynamic Optimization Support	Dynamic Optimization Support	Various Metaheuristic Algorithms

Table 3: This table provides a comprehensive overview of various optimization methods used in parallel and distributed computing environments. It categorizes each method by its performance enhancement capabilities, system adaptability, and algorithmic diversity, highlighting their unique contributions to improving computational efficiency and scalability.

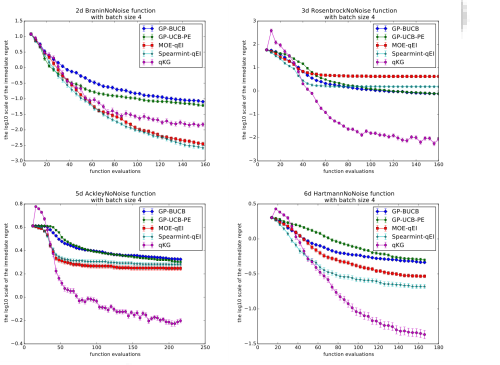
Optimization techniques and strategies are central to enhancing performance across domains in parallel and distributed computing environments. The HiDup framework achieves up to 61% faster training times for Mixture-of-Experts models, exemplifying substantial improvements in training efficiency through innovative strategies [56].

The SDP framework combines centralized and decentralized synchronization, allowing flexible group formation that adapts to dynamic worker performance [21]. This adaptability is critical for optimizing distributed computing environments.

In gradient-based optimization, the Adaptive Consensus Gradients Aggregation (AdaCons) method significantly improves convergence properties and reduces communication overhead compared to standard gradient summing methods [57]. The GC-MMC method introduces multi-message communication, enabling workers to send multiple coded partial gradients, optimizing communication efficiency and reducing latency [66]. A competitive stochastic sample size optimization strategy enhances the traditional Big-means clustering algorithm by adaptively selecting sample sizes for better performance [6].

The FALCON framework provides a non-intrusive, framework-agnostic detection and mitigation approach that dynamically responds to fail-slow incidents, optimizing system reliability and performance [1].

Collectively, these optimization techniques demonstrate a diverse array of approaches employed to enhance performance across domains in distributed systems. By addressing challenges like communication overhead, resource allocation, and system adaptability, these methods improve the efficiency and scalability of computing solutions, particularly in machine learning and cloud computing. Advancements in parallel computing frameworks, such as MapReduce and CUDA, have led to new machine learning libraries capable of handling large datasets effectively. Innovative algorithms, like the parallel adaptive clustering (PAC) algorithm, automate data classification and dynamically determine the optimal number of clusters, significantly reducing computational burdens and improving performance in data analysis tasks. These developments contribute to robust solutions that adapt to evolving data environments and optimize resource utilization in cloud-based systems [5, 38].



(a) Comparison of different optimization algorithms on four functions with batch sizes of 4.[54]

```python
class EvolutionaryAlgorithm(Algorithm, ABC):
  def __init__(self,
               problem: Problem,
               population_size: int,
               offspring_size: int):
    super(EvolutionaryAlgorithm, self).__init__()
    self.problem = problem
    self.population_size = population_size
    self.offspring_size = offspring_size

  @abstractmethod
  def selection(self, population):
    pass

  @abstractmethod
  def reproduction(self, population):
    pass

  @abstractmethod
  def replacement(self, population, offspring):
    pass

  def init_progress(self):
    self.evaluations = self.population_size

  def step(self):
    mating_pool = self.selection(self.solutions)
    offspring = self.reproduction(mating_pool)
    offspring = self.evaluate(offspring)
    self.solutions = self.replacement(self.solutions, offspring)

  def update_progress(self):
```

(b) Evolutionary Algorithm Class Definition[55]

Figure 3: Examples of Optimization Techniques and Strategies

As shown in Figure 3, the exploration of cross-domain optimization techniques is illustrated through two complementary examples. The first example provides a comparative analysis of various optimization algorithms applied to four functions, each evaluated with a batch size of four. The graphical representation showcases the performance of these algorithms in terms of immediate regret, plotted on a log10 scale, against the number of function evaluations. This visualization offers insights into the efficiency of each algorithm over time on functions such as the 2d BraninNoNoise and 3d Rosen. Complementing this, the second example delves into the structural definition of an evolutionary algorithm class in Python. This abstract class, defined using the '@abstractmethod' decorator, outlines essential methods such as 'selection', 'reproduction', 'replacement', and

‘init$_progress$‘, $which are pivotal for the algorithm's operation. The ‘step‘ method orchestrates the overall process by invoki$
55]. $Table 3 presents a detailed comparison of different optimization techniques, illustrating their respective strengths i$

3.3 Challenges in Large-Scale Distributed Systems

Implementing cross-domain optimization in large-scale distributed systems presents numerous challenges spanning computational, algorithmic, and infrastructural domains. A significant issue is the inefficiency of existing serverless frameworks in managing burst-parallel workloads due to centralized scheduling limitations and excessive data movement, which can severely impede performance [10]. High computational effort required for complex simulations, such as those in computational fluid dynamics, further complicates this landscape, as traditional methods struggle to manage these intricacies effectively [58].

Excessive communication overhead associated with parallel stochastic gradient descent constitutes a major bottleneck, resulting in delays and increased energy consumption [59]. High interprocessor communication costs hinder scalability, presenting formidable barriers to effective optimization [18]. Existing methods often discard computations from straggling workers, leading to wasted resources and prolonged computation times [66].

Resource allocation complexities, including the need for integral resource allocations, can lead to underutilization of cluster resources, preventing jobs from dynamically adjusting their resource needs [60]. The scalability of neural network verification methods struggles with large networks, complicating cross-domain optimization [61]. Centralized scheduling systems often result in long queues and execution delays, particularly during bulk job submissions [25].

Performance variation due to resource contention and heterogeneity in workload or computing nodes increases job execution time, posing a significant obstacle to efficient optimization [62]. The scalability of DG-based solvers is hindered by communication overhead and synchronization requirements when solving PDEs at extreme scales [65]. Simplified models used in existing scheduling methods fail to address the complexities of real-world computational environments, leading to suboptimal scheduling [63].

The NP-hard nature of scheduling jobs with synchronization, packing, and placement constraints presents a significant difficulty in cross-domain optimization, which existing methods have not adequately addressed [64]. Statistical and implementation issues in designing RS procedures for parallel environments can lead to inefficiencies and invalid results [67]. High communication costs and the need for effective cooperation among distributed agents further complicate the implementation of cross-domain optimization [20].

Existing methods' requirement for access to entire datasets poses a key obstacle in cross-domain optimization, especially for large datasets typically encountered in computed tomography applications [13]. Additionally, a lack of effective techniques to identify and exploit similarities among concurrent queries prevents optimal resource utilization [11]. The challenges of parallelizing machine learning algorithms and adapting them to modern computing environments further complicate efficient data processing [5].

The challenges faced by large-scale distributed systems—including the need for robust failure detection, efficient resource utilization, and effective load balancing—demand innovative solutions that enhance their robustness, scalability, and efficiency. For instance, adaptive, decentralized failure detection architectures can improve fault tolerance by operating independently of application flow, while joint auto-scaling and load balancing strategies can optimize service elasticity without centralized queue management, thereby minimizing communication overhead. Additionally, advancements in middleware architectures and machine learning frameworks are essential for addressing the complexities of resource sharing and data processing in cloud environments. Collectively, these advancements are crucial for overcoming the limitations of current systems and achieving optimal performance in distributed computing [68, 69, 31, 5, 44]. Addressing these issues is vital for advancing cross-domain optimization and facilitating the development of more effective distributed computing environments.

Feature	Hybrid Computing Model	OptEx Framework	CSS-Big-means
Optimization Focus	Workload Distribution	Gradient Estimation	Clustering Efficiency
Method Type	Task Partitioning	Kernelized Approximation	Sample Size Optimization
Resource Utilization	Cpu And Gpu	Parallelization	Dynamic Allocation

Table 4: This table provides a comparative analysis of three distinct methodologies employed in cross-domain optimization. It highlights the focus, method type, and resource utilization for each approach, illustrating their respective contributions to enhancing computational efficiency and resource management in distributed systems.

4 Hybrid-Parallel Distributed Training

4.1 Algorithmic Strategies

Algorithmic strategies are essential for enhancing computational efficiency and resource allocation in hybrid-parallel distributed training, especially in large-scale environments. The ADCME-MPI method exemplifies the integration of Deep Neural Networks (DNNs) and Partial Differential Equations (PDEs) within a computational graph, facilitating efficient gradient computation and data communication in parallel settings [28]. This integration is crucial for managing complex computations and optimizing resource utilization.

The CSGD algorithm allows arbitrary subsets of observations and reconstructed volumes to be processed simultaneously, optimizing resource usage and improving system performance [13]. Similarly, the Dynamic Multi-threading Approach (DMA) utilizes parallel computing on multi-core servers to execute channel coding functions concurrently, significantly reducing latency in Cloud Radio Access Network (C-RAN) architectures [14].

Multi-Processor Scheduling with NUMA Effects (MPS-NE) enhances resource management by addressing scheduling tasks as Directed Acyclic Graphs (DAGs) on multiple processors, considering communication costs and Non-Uniform Memory Access (NUMA) effects [63]. SynchPack ensures coherent job execution across distributed environments by synchronizing tasks based on processing requirements and resource availability [64].

The General Framework for Static Cost Analysis (GFSCA) predicts resource usage in parallel logic programs, providing insights essential for efficient resource management [2]. The Adaptive Consensus Gradients Aggregation (AdaCons) method dynamically adjusts gradient weights based on consensus, applying momentum to enhance convergence in hybrid-parallel distributed training [57]. DiCOM optimizes communication paths between processes in clustered environments, reducing overhead and enhancing efficiency [9].

The Gradient Coding with Clustering and Multi-Message Communication (GC-MMC) method mitigates delays caused by slower workers by dividing datasets into mini-batches for partial gradient computation [66]. Wukong, a decentralized, locality-aware framework, enhances data locality and resource efficiency in serverless parallel computing, reducing data movement and improving resource utilization [10].

These strategies demonstrate innovative techniques in hybrid-parallel distributed training, focusing on performance enhancement through data and model parallelism, efficient gradient synchronization, and scalability via advanced frameworks like PyTorch, which supports near-linear scaling across multiple GPUs [70, 5, 26]. By addressing challenges such as communication overhead and resource allocation, these methods significantly improve the efficiency of distributed systems.

4.2 Communication and Synchronization

Communication and synchronization are pivotal in hybrid-parallel distributed training, directly influencing the efficiency and scalability of distributed systems. The DMPC model utilizes dynamic algorithms to maintain solutions to graph problems, optimizing communication paths and synchronizing operations across distributed environments [71]. This model highlights the importance of efficient communication protocols for system coherence and performance.

In decentralized environments, the adaptive checkpointing scheme enhances system performance by dynamically adjusting checkpoint intervals based on conditions, minimizing communication overhead while ensuring data consistency [72]. Dynamic algorithms within parallel computing frameworks foster robust communication and synchronization strategies, essential for managing the complexities of hybrid-parallel distributed training. Techniques such as straggler replication and adaptive caching of intermediate results address bottlenecks and minimize redundant computations, reducing latency and enhancing throughput [38, 73, 69, 44, 74].

Adaptive synchronization techniques enable systems to adjust dynamically to workload fluctuations and resource availability, enhancing resource utilization and significantly reducing communication delays. This adaptability is crucial in large-scale distributed environments, where effective load balancing and auto-scaling strategies optimize performance and energy consumption without centralized control [75, 31, 74].

Collectively, these approaches illustrate the critical role of communication and synchronization in hybrid-parallel distributed training. By leveraging dynamic algorithms and adaptive techniques, these methods enhance the efficiency and scalability of distributed systems, enabling them to handle complex computational tasks while optimizing resource allocation. The integration of meta-heuristics and machine learning into software optimization processes further enhances the systems' ability to adapt to evolving datasets and execution contexts, improving management of computational resources and overall system performance [75, 16, 71].

5 Computational-Network Cost Awareness

As distributed systems grow in complexity and scale, managing computational and network costs becomes increasingly crucial. These challenges affect not only system performance but also scalability, necessitating innovative solutions to address the multifaceted issues in computational-network cost management.

5.1 Challenges in Computational-Network Cost Management

The management of computational and network costs in distributed systems is fraught with challenges that impact both performance and scalability. A significant issue is the communication overhead that escalates with the number of processors, necessitating innovative cost management solutions [9]. Traditional Computational Fluid Dynamics (CFD) simulations further complicate this by demanding extensive computational resources, which decoupling approaches aim to alleviate.

The complexity of parallel computing libraries and the need for specific optimizations pose barriers for domain experts lacking high-performance computing expertise [76]. Memory access times also limit Massively Parallel Processing (MPP) system performance, highlighting the need for improved memory management techniques. Centralized systems often struggle to adapt to varying loads, presenting another obstacle in cost management.

The GC-MMC method's communication load, due to multiple coded messages, may not suit all network architectures, complicating cost management [66]. In environments with variable task arrival rates, methods like Balanced-PANDAS may also face challenges. Centralized approaches often lack scalability, especially in large systems.

Stochastic optimization methods require overhead for real-time system adjustments, complicating implementation. The ADG method improves scalability by minimizing communication overhead and facilitating asynchronous computations, crucial in high-performance computing environments where effective performance analysis and optimization are necessary [75, 77, 11, 69, 78]. Sophisticated scheduling algorithms, while effective, can be computationally intensive compared to lightweight heuristics.

Centralized processing of radio signals struggles with latency requirements due to reliance on high-capacity fiber links and resource management [14]. Scheduling algorithm performance may vary with workload and hardware characteristics, limiting applicability [64]. Improper GPU memory management can become a bottleneck, particularly with larger datasets.

Addressing these challenges requires ongoing innovation in methodologies focusing on efficiency and scalability. This includes advanced synchronization techniques, resource allocation optimization, and

thorough analysis of network communication complexities. Frameworks like Centauri and systems such as Colossal-AI exemplify these innovations by facilitating efficient communication-computation overlap and providing unified interfaces for parallel training, respectively [79, 78, 80, 26].

5.2 Techniques for Cost Optimization

Optimizing computational and network costs is essential for enhancing distributed systems' efficiency and scalability, especially under resource constraints. The ADCME-MPI method optimizes computational costs by decoupling data communication from computation in complex simulations [28]. This underscores the importance of task decoupling in performance enhancement.

The OptEx framework accelerates first-order optimization algorithm convergence through parallel computing, reducing sequential iterations and optimizing computational efficiency [4]. This illustrates the potential of parallel frameworks in minimizing expenses.

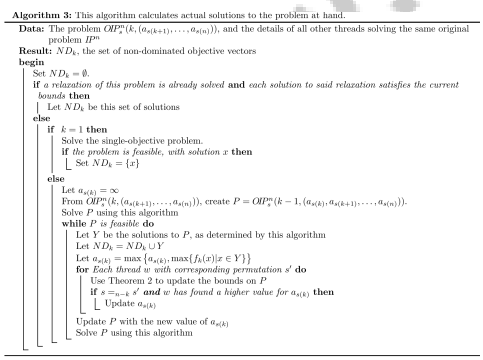
For large-scale data processing, the CSS-Big-means algorithm reduces computational time through dynamic sampling strategies, enhancing cost management efficiency [6]. HeteroG achieves up to 222% training speed-up in heterogeneous environments, showcasing sophisticated parallelism strategies [7].

Nebula-I optimizes communication costs through advanced compression and efficient resource utilization [8]. AdaCons addresses cost awareness with advanced gradient aggregation techniques, enhancing distributed training frameworks [57].

DiCOM minimizes communication latency and improves resource allocation, highlighting direct communication methods' significance [9]. Future research in gradient coding aims to enhance distributed systems' flexibility and efficiency by optimizing partial gradient orders [66].

Wukong significantly improves execution speed and reduces network costs through locality-aware frameworks [10]. Effective caching strategies in multi-query optimization minimize recomputations, optimizing costs [11].

These techniques represent diverse strategies for optimizing computational and network costs. By addressing communication overhead, memory usage, and resource allocation, they enhance distributed computing environments' efficiency and scalability. Innovations like parallel adaptive clustering and dynamic load balancing ensure equitable workload distribution and minimize computational demands, making these environments more suitable for resource-constrained scenarios [69, 38, 75, 60].



Algorithm 3: This algorithm calculates actual solutions to the problem at hand.

Data: The problem $OIP_s^n(k,(a_{s(k+1)},\ldots,a_{s(n)}))$, and the details of all other threads solving the same original problem IP^n

Result: ND_k, the set of non-dominated objective vectors

begin

  Set $ND_k = \emptyset$.

  if *a relaxation of this problem is already solved* **and** *each solution to said relaxation satisfies the current bounds* **then**

    Let ND_k be this set of solutions

  else

    if $k = 1$ **then**

      Solve the single-objective problem.

      if *the problem is feasible, with solution* x **then**

        Set $ND_k = \{x\}$

    else

      Let $a_{s(k)} = \infty$

      From $OIP_s^n(k,(a_{s(k+1)},\ldots,a_{s(n)}))$, create $P = OIP_s^n(k-1,(a_{s(k)},a_{s(k+1)},\ldots,a_{s(n)}))$.

      Solve P using this algorithm

      while P *is feasible* **do**

        Let Y be the solutions to P, as determined by this algorithm

        Let $ND_k = ND_k \cup Y$

        Let $a_{s(k)} = \max\{a_{s(k)}, \max\{f_k(x) | x \in Y\}\}$

        for *Each thread* w *with corresponding permutation* s' **do**

          Use Theorem 2 to update the bounds on P

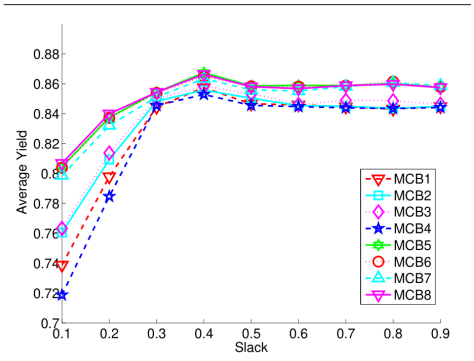
          if $s =_{n-k} s'$ ***and*** w *has found a higher value for* $a_{s(k)}$ **then**

            Update $a_{s(k)}$

        Update P with the new value of $a_{s(k)}$

        Solve P using this algorithm

(a) Algorithm 3: This algorithm calculates actual solutions to the problem at hand.[81]



(b) The image shows the average yield of different models as a function of slack.[60]

Figure 4: Examples of Techniques for Cost Optimization

As shown in Figure 4, optimizing costs in computational-network awareness is critical and achievable through innovative techniques. Algorithm 3 introduces a systematic method for specific optimization problems, while the second example visually represents model performance across varying conditions, aiding in resource allocation strategy identification. These examples emphasize strategic problem-solving and data-driven analysis in achieving cost-effective outcomes in computational networks [81, 60].

6 Applications in Large-Scale Distributed Systems

6.1 Applications and Case Studies

Cross-domain optimization and hybrid-parallel strategies have substantially improved computational efficiency and scalability across diverse fields. The ADCME-MPI framework exemplifies this by enhancing performance in large-scale scientific machine learning, particularly in complex numerical simulations [28]. Similarly, the HFSL framework optimizes training time and energy consumption in distributed systems, demonstrated with the CIFAR-10 dataset using 16 workers [47]. In wireless networks, the Hydrozoa framework efficiently manages resources under constraints, optimizing communication and computation with models like AlexNet, ResNet-34, and BERT-12 [36]. The PDSS's adaptation to parallel computing has enabled simulations of millions of households, significantly reducing runtime and meeting memory requirements [82].

Processor-oriented parallel programming frameworks have achieved high data throughput by maximizing hardware utilization, supporting portable parallel applications across various object-oriented programming languages [83]. In safety-critical applications like aircraft collision avoidance, the SC algorithm has been validated on diverse benchmarks, demonstrating its utility in verifying neural networks in real-world scenarios [61]. The ADG method exemplifies the potential of DG-based PDE solvers for simulations on massively parallel supercomputers, effectively addressing communication bottlenecks [65].

Astrophysical observations utilizing datasets for Globular Cluster classification have demonstrated the advantages of GPU-based implementations over traditional CPU approaches, emphasizing hybrid computing strategies' impact on processing efficiency [23]. Scalable deep reinforcement learning techniques validated with real traces from Facebook further confirm the effectiveness of these strategies in optimizing performance [84]. The Nebula-I framework has significantly enhanced training efficiency and model performance in collaborative settings, achieving state-of-the-art results in cross-lingual natural language inference tasks [8]. In telecommunications, a method validated through an OAI-based testbed has shown considerable performance improvements, paving the way for centralized cloud-native RAN architectures [14].

These applications and case studies collectively demonstrate the transformative capabilities of cross-domain optimization and hybrid-parallel strategies in enhancing computational efficiency, sample complexity, and software optimization across fields such as operations research, machine learning, and advertising systems [16, 17, 26, 39, 37]. By improving performance, reducing energy consumption, and optimizing resource utilization, these strategies significantly advance distributed systems across various domains.

6.2 Real-World Implementations

Real-world implementations of cross-domain optimization and hybrid-parallel strategies have markedly improved computational efficiency, scalability, and resource management across diverse domains. The CSS-Big-means algorithm exemplifies these advancements, achieving significant improvements in big data clustering and demonstrating its practical value in clustering large datasets [6]. In distributed data analysis, the WONDER algorithm has substantially reduced computation time while maintaining high prediction accuracy, proving effective in applications requiring efficient data processing. The asynchronous BCD algorithm has achieved linear convergence in optimizing nonconvex functions under complex conditions, showcasing its robustness in large-scale distributed systems [85].

Parallel optimization techniques have led to notable improvements in engineering applications, such as aircraft CAD models, optimizing approximation errors and computational times. In deep learning, EASGD and its variants have significantly outperformed existing methods like DOWNPOUR, enhancing deep neural network training under communication constraints [86]. The OSDP framework has achieved substantial speedups in training throughput compared to existing parallel training systems, enhancing the efficiency of distributed training environments [30]. Additionally, the asynchronous stochastic block coordinate algorithm has been effectively applied across multi-core processors and GPU-accelerators, demonstrating its versatility in real-world scenarios [87].

In large-scale computational fluid dynamics (CFD) simulations, performance optimizations on the Tianhe-2 supercomputer illustrate the successful application of these optimization strategies, utilizing

over 1.376 million cores for scalable results [88]. The SA-BCD method has further shown its effectiveness in enhancing the scalability of convex optimization methods within large-scale machine learning applications [18].

These implementations collectively highlight the transformative impact of cross-domain optimization and hybrid-parallel strategies in enhancing computational performance and scalability across diverse applications. By addressing challenges such as communication overhead, dynamic resource allocation, and system adaptability, these strategies are vital for improving the efficiency and performance of distributed computing environments. Techniques like dynamic load balancing ensure optimal task distribution across processing elements, while advanced middleware architectures enhance resource sharing in large-scale distributed systems. Innovative approaches such as straggler replication help mitigate bottlenecks in parallel computing tasks, contributing to the overall advancement and effectiveness of distributed computing frameworks [69, 38, 60, 75, 44].

7 Challenges and Future Directions

7.1 Future Directions in Cross-Domain Optimization

Advancements in cross-domain optimization are set to enhance distributed systems' efficiency, scalability, and adaptability. Key areas of exploration include refining task mapping strategies and reducing communication costs in hybrid computing models, which promise improved resource utilization in heterogeneous environments. These efforts align with innovations in GNN architecture, extending the applicability of frameworks like HeteroG to diverse training scenarios [7]. In data-intensive settings, optimizing sample sizes and refining algorithms such as CSS-Big-means are crucial for fully leveraging modern computational architectures [6]. Additionally, enhancing local batch sizes and integrating Hessian-based methods may improve performance in adaptive gradient aggregation methods, indicating a promising research trajectory [57].

Security enhancements for cross-cloud training, alongside optimized communication strategies, are critical for frameworks like Nebula-I [8]. Integrating DiCOM with standard TCP and applying it to parallel computing environments could advance communication efficiency [9]. Developing a queuing mechanism for batching queries aims to improve multi-query processing efficiency [11]. Future research should also refine distributed programming models to adapt to varying network conditions and explore new paradigms that complement existing approaches [12].

In distributed computation for linear inverse problems, adaptive parameter selection methods and regularization techniques are essential for enhancing reconstruction quality [13]. Optimizing scheduling algorithms and exploring functional splits could enhance the efficiency and scalability of C-RAN architectures [14]. Enhancing fault tolerance mechanisms and examining the portability of serverless frameworks like Wukong to other platforms will be vital for advancing distributed systems' robustness and adaptability [10]. These research directions emphasize the need for continuous innovation in cross-domain optimization, aiming to develop robust, efficient, and flexible solutions in dynamic computational environments.

7.2 Architectural Innovations

Architectural innovations are pivotal in enhancing distributed systems' scalability, efficiency, and adaptability. A promising direction is developing hybrid computing architectures that integrate CPU and GPU resources, optimizing task allocation based on each processor type's capabilities [43]. This integration is beneficial for applications requiring high computational throughput, enabling more efficient resource utilization and reduced execution times.

Middleware architecture evolution within Grid computing is another ripe area for innovation. Effective middleware solutions enhance large-scale resource sharing and interoperability among diverse computing environments [44]. Streamlined communication and resource management through these architectures can significantly improve distributed applications' performance and scalability.

In machine learning, architectural innovations like the HeteroG framework demonstrate the potential to transform single-GPU models into distributed ones, optimizing parallelism strategies and communication methods to enhance training efficiency [7]. This highlights the importance of adaptable architectures that dynamically respond to varying computational loads and network conditions.

Developing advanced scheduling algorithms considering Non-Uniform Memory Access (NUMA) effects and communication costs is essential for optimizing task execution in multi-processor environments [63]. Such innovations are crucial for improving distributed systems' efficiency, particularly in scenarios involving complex task dependencies and heterogeneous resources.

Direct communication methods, exemplified by the DiCOM solution, offer significant potential for reducing communication overhead and enhancing data exchange efficiency in clustered environments [9]. Minimizing latency and optimizing resource allocation through such innovations contribute to distributed systems' overall performance.

Exploring serverless architectures, such as Wukong, demonstrates the potential for improving data locality and reducing network I/O costs in parallel computing environments [10]. These architectures provide a flexible and scalable approach to managing computational tasks, making them well-suited for dynamic and resource-constrained environments.

These architectural innovations collectively highlight ongoing efforts to enhance distributed systems' capabilities, paving the way for more efficient, scalable, and adaptable computational frameworks. By addressing significant challenges such as effective resource allocation, minimizing communication overhead, and enhancing system interoperability, these innovations are poised to propel advancements in distributed computing. The development of middleware architectures for large-scale distributed systems (LSDSs) is crucial for enabling resource sharing, while emerging machine learning frameworks and cloud computing solutions tackle the complexities of processing vast datasets. Strategies to mitigate straggler tasks in parallel computing environments, such as task replication, are also being refined to optimize latency and resource costs. Collectively, these advancements improve distributed systems' efficiency and expand their applicability across various domains, including artificial intelligence and big data analytics [69, 5, 44, 26].

7.3 Integration with Emerging Technologies

Integrating emerging technologies into distributed systems presents opportunities to enhance performance and scalability, providing innovative solutions to current computational challenges. A promising avenue involves utilizing machine learning techniques to optimize resource allocation and task scheduling in distributed environments. Machine learning models, such as those optimizing radio network functions, can significantly improve resource management efficiency and reduce latency in cloud-native architectures [48].

The adoption of blockchain technology in distributed systems offers enhancement by providing decentralized and secure frameworks for data exchange and transaction processing. This integration can improve distributed applications' reliability and security, particularly in scenarios requiring robust data integrity and traceability [36]. Additionally, blockchain facilitates the development of decentralized applications (DApps) that leverage distributed computing resources for increased scalability and resilience.

In data processing, incorporating quantum computing techniques holds the promise of exponentially increasing distributed systems' computational capabilities. Quantum algorithms can potentially solve complex optimization problems more efficiently than classical methods, significantly boosting data-intensive applications' performance [71]. As quantum computing technology evolves, its integration into existing systems could revolutionize computational task execution, offering unprecedented speed and efficiency.

Implementing edge computing technologies represents another critical area for enhancing distributed systems. By processing data closer to the source, edge computing reduces latency and bandwidth usage, improving distributed applications' responsiveness and efficiency [72]. This approach is particularly beneficial for real-time data processing applications, such as those in the Internet of Things (IoT) and autonomous systems.

Furthermore, deploying advanced communication protocols, such as those in fifth-generation (5G) networks, can enhance distributed systems' connectivity and data transfer capabilities. These protocols offer higher data rates and lower latency, enabling more efficient communication between distributed nodes and improving overall system performance [14].

Integrating these emerging technologies into distributed systems holds the potential to significantly enhance their performance and scalability. By harnessing advancements in machine learning, blockchain

technology, quantum computing, edge computing, and sophisticated communication protocols, distributed systems can improve their efficiency, resilience, and adaptability. This integration facilitates processing large-scale datasets through distributed machine learning frameworks like TensorFlow and PyTorch while enabling robust data management and security via blockchain. Moreover, incorporating quantum computing promises to accelerate computational capabilities, while edge computing optimizes data processing closer to its source, thereby reducing latency. Collectively, these technologies pave the way for developing more sophisticated and capable computational frameworks that can address increasingly complex challenges across various domains [22, 26, 70, 89, 5].

8 Conclusion

The integration of cross-domain optimization and hybrid-parallel strategies is crucial for enhancing the efficiency and scalability of large-scale distributed systems. This survey underscores the significant role of distributed methodologies, particularly in improving communication efficiency and scalability in fluid flow simulations. The development of specialized theories and algorithms for large-scale stochastic optimization (SO) is essential, with parallel computing being a key factor in advancing high-dimensional SO. Enhancements in coflow scheduling have effectively reduced makespan in both homogeneous and heterogeneous parallel networks, validating the efficacy of the proposed algorithms. The adoption of parallel object-oriented frameworks has reduced the complexity and cost of parallel programming, thereby improving performance in large-scale systems. As model sizes increase, addressing memory and computational constraints becomes increasingly important, highlighting the necessity for integrated optimization strategies. The use of programmable switches as parallel computing devices has demonstrated significant performance improvements by offloading computations to the data plane. Additionally, the separated representation approach has proven effective in propagating uncertainty in coupled domain problems, achieving accurate estimates with reduced computational effort. New heuristics for parallel computing environments bridge existing methodological gaps and enhance scalability. High-performance computing (HPC) has been shown to improve the tractability of complex power system models, offering promising applications for real-time scenarios. The application of HPC techniques in CNN training has resulted in substantial improvements in training time and model accuracy. This survey highlights the importance of machine learning methods for predictive performance modeling, providing greater flexibility and adaptability compared to traditional models. The proposed distributed simulation framework has demonstrated significant enhancements in modeling capabilities for complex Grid systems, offering a more efficient and scalable solution. Moreover, multiple learning approaches have effectively addressed the memory barrier issue in regression analysis for big data, enabling efficient computation of multiple models in a single dataset pass. Collectively, these findings emphasize the critical importance of integrating cross-domain optimization and hybrid-parallel strategies in large-scale distributed systems, driving advancements in computational efficiency, scalability, and resource management. The conclusion reaffirms that NPDC surpasses existing DC-based evolutionary algorithms in both solution quality and efficiency, highlighting the necessity of effective optimization strategies in large-scale distributed systems.

References

[1] Tianyuan Wu, Wei Wang, Yinghao Yu, Siran Yang, Wenchao Wu, Qinkai Duan, Guodong Yang, Jiamang Wang, Lin Qu, and Liping Zhang. Falcon: Pinpointing and mitigating stragglers for large-scale hybrid-parallel training. *arXiv preprint arXiv:2410.12588*, 2024.

[2] Maximiliano Klemen, Pedro Lopez-Garcia, John P. Gallagher, Jose F. Morales, and Manuel V. Hermenegildo. Towards a general framework for static cost analysis of parallel logic programs, 2019.

[3] O. A. Malafeyev and S. A. Nemnyugin. Parallel computing as a congestion game, 2018.

[4] Yao Shu, Jiongfeng Fang, Ying Tiffany He, and Fei Richard Yu. Optex: Expediting first-order optimization with approximately parallelized iterations, 2024.

[5] Daniel Pop. Machine learning and cloud computing: Survey of distributed and saas solutions, 2016.

[6] Rustam Mussabayev and Ravil Mussabayev. Superior parallel big data clustering through competitive stochastic sample size optimization in big-means, 2024.

[7] Xiaodong Yi, Shiwei Zhang, Ziyue Luo, Guoping Long, Lansong Diao, Chuan Wu, Zhen Zheng, Jun Yang, and Wei Lin. Optimizing distributed training deployment in heterogeneous gpu clusters. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, pages 93–107, 2020.

[8] Yang Xiang, Zhihua Wu, Weibao Gong, Siyu Ding, Xianjie Mo, Yuang Liu, Shuohuan Wang, Peng Liu, Yongshuai Hou, Long Li, Bin Wang, Shaohuai Shi, Yaqian Han, Yue Yu, Ge Li, Yu Sun, Yanjun Ma, and Dianhai Yu. Nebula-i: A general framework for collaboratively training deep learning models on low-bandwidth cloud clusters, 2022.

[9] Adam Lev-Libfeld, Alex Margolin, and Amnon Barak. Open-mpi over mosix: paralleled computing in a clustered world, 2019.

[10] Benjamin Carver, Jingyuan Zhang, Ao Wang, Ali Anwar, Panruo Wu, and Yue Cheng. Wukong: A scalable and locality-enhanced framework for serverless parallel computing, 2020.

[11] Pietro Michiardi, Damiano Carra, and Sara Migliorini. Cache-based multi-query optimization for data-intensive scalable computing frameworks, 2018.

[12] Christopher S. Meiklejohn. On the design of distributed programming models, 2017.

[13] Yushan Gao and Thomas Blumensath. Distributed computation of linear inverse problems with application to computed tomography, 2017.

[14] Veronica Quintuna Rodriguez and Fabrice Guillemin. Higher aggregation of gnodebs in cloud-ran architectures via parallel computing, 2019.

[15] Ahmed Al-Shafei, Hamidreza Zareipour, and Yankai Cao. A review of high-performance computing and parallel techniques applied to power systems optimization, 2022.

[16] Suejb Memeti, Sabri Pllana, Alecio Binotto, Joanna Kolodziej, and Ivona Brandic. Using meta-heuristics and machine learning for software optimization of parallel computing systems: A systematic literature review, 2018.

[17] Guido Schryen. Parallel computational optimization in operations research: A new integrative framework, literature review and research directions, 2019.

[18] Aditya Devarakonda, Kimon Fountoulakis, James Demmel, and Michael W. Mahoney. Avoiding synchronization in first-order methods for sparse convex optimization, 2017.

[19] Hee Jun Yang and Hyea Hyun Kim. Iterative algorithms for partitioned neural network approximation to partial differential equations, 2023.

[20] Wei-Neng Chen, Feng-Feng Wei, Tian-Fang Zhao, Kay Chen Tan, and Jun Zhang. A survey on distributed evolutionary computation, 2023.

[21] Xupeng Miao, Yining Shi, Zhi Yang, Bin Cui, and Zhihao Jia. Sdpipe: A semi-decentralized framework for heterogeneity-aware pipeline-parallel training. *Proceedings of the VLDB Endowment*, 16(9):2354–2363, 2023.

[22] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems, 2016.

[23] Stefano Cavuoti, Mauro Garofalo, Massimo Brescia, Antonio Pescapé, Giuseppe Longo, and Giorgio Ventre. Genetic algorithm modeling with gpu parallel computing technology, 2012.

[24] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning, 2020.

[25] A. Anjum, R. McClatchey, H. Stockinger, A. Ali, I. Willers, M. Thomas, M. Sagheer, K. Hasham, and O. Alvi. Diana scheduling hierarchies for optimizing bulk job scheduling, 2007.

[26] Tie-Yan Liu, Wei Chen, and Taifeng Wang. Distributed machine learning: Foundations, trends, and practices. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 913–915, 2017.

[27] M. S. Fabien, M. G. Knepley, R. T. Mills, and B. M. Riviere. Manycore parallel computing for a hybridizable discontinuous galerkin nested multigrid method, 2019.

[28] Kailai Xu, Weiqiang Zhu, and Eric Darve. Distributed machine learning for computational engineering using mpi, 2020.

[29] Sirui Bi, Jiaxin Zhang, and Guannan Zhang. Scalable deep-learning-accelerated topology optimization for additively manufactured materials, 2020.

[30] Youhe Jiang, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, and Bin Cui. Osdp: Optimal sharded data parallel for distributed deep learning. *arXiv preprint arXiv:2209.13258*, 2022.

[31] Debankur Mukherjee, Souvik Dhara, Sem Borst, and Johan S. H. van Leeuwaarden. Optimal service elasticity in large-scale distributed systems, 2017.

[32] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-Tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. *Proceedings of machine learning and systems*, 2:230–246, 2020.

[33] Fredy Juarez, Jorge Ejarque, and Rosa M Badia. Dynamic energy-aware scheduling for parallel task-based application in cloud computing. *Future Generation Computer Systems*, 78:257–271, 2018.

[34] Zhiquan Lai, Shengwei Li, Xudong Tang, Keshi Ge, Weijie Liu, Yabo Duan, Linbo Qiao, and Dongsheng Li. Merak: An efficient distributed dnn training framework with automated 3d parallelism for giant foundation models. *IEEE Transactions on Parallel and Distributed Systems*, 34(5):1466–1478, 2023.

[35] Anirudh Subramanyam, Youngdae Kim, Michel Schanen, François Pacaud, and Mihai Anitescu. A globally convergent distributed jacobi scheme for block-structured nonconvex constrained optimization problems, 2021.

[36] Runsheng Guo, Victor Guo, Antonio Kim, Josh Hildred, and Khuzaima Daudjee. Hydrozoa: Dynamic hybrid-parallel dnn training on serverless containers. *Proceedings of Machine Learning and Systems*, 4:779–794, 2022.

[37] Zishi Zhang and Yijie Peng. "clustering and conquer" procedures for parallel large-scale ranking and selection, 2024.

[38] Benjamin McLaughlin and Sung Ha Kang. A new parallel adaptive clustering and its application to streaming data, 2021.

[39] Han Xu, Hao Qi, Kunyao Wang, Pei Wang, Guowei Zhang, Congcong Liu, Junsheng Jin, Xiwei Zhao, Zhangang Lin, Jinghe Hu, and Jingping Shao. Pcdf: A parallel-computing distributed framework for sponsored search advertising serving, 2023.

[40] Henk Dreuning, Kees Verstoep, Henri E Bal, and Rob V Van Nieuwpoort. Capture: Memory-centric partitioning for distributed dnn training with hybrid parallelism. In *2023 IEEE 30th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pages 76–86. IEEE, 2023.

[41] Jesper Larsson Träff. Lectures on parallel computing, 2024.

[42] Zaid Abdi Alkareem Alyasseri. Survey of parallel computing with matlab, 2014.

[43] Kishore Kothapalli, Dip Sankar Banerjee, P. J. Narayanan, Surinder Sood, Aman Kumar Bahl, Shashank Sharma, Shrenik Lad, Krishna Kumar Singh, Kiran Matam, Sivaramakrishna Bharadwaj, Rohit Nigam, Parikshit Sakurikar, Aditya Deshpande, Ishan Misra, Siddharth Choudhary, and Shubham Gupta. Cpu and/or gpu: Revisiting the gpu vs. cpu myth, 2013.

[44] Florin Pop, Ciprian Mihai Dobre, Alexandru Costan, Mugurel Ionut Andreica, Eliana-Dina Tirsa, Corina Stratan, and Valentin Cristea. Critical analysis of middleware architectures for large scale distributed systems, 2009.

[45] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Simgrid: a sustained effort for the versatile simulation of large scale distributed systems, 2013.

[46] Meimei Liu, Zuofeng Shang, and Guang Cheng. How many machines can we use in parallel computing for kernel ridge regression?, 2019.

[47] Benshun Yin, Zhiyong Chen, and Meixia Tao. Predictive gan-powered multi-objective optimization for hybrid federated split learning, 2022.

[48] Gangxiang Shen, Longfei Li, Ya Zhang, Wei Chen, Sanjay K. Bose, and Moshe Zukerman. Machine learning-assisted least loaded routing to improve performance of circuit-switched networks, 2018.

[49] Shigeo Kawata. Computer assisted parallel program generation, 2015.

[50] Brijender Kahanwal. Towards high performance computing (hpc) through parallel programming paradigms and their principles, 2014.

[51] Gaspard Beugnot, Julien Mairal, and Alessandro Rudi. Gloptinets: Scalable non-convex optimization with certificates, 2023.

[52] Peng Yang, Ke Tang, and Xin Yao. A parallel divide-and-conquer based evolutionary algorithm for large-scale optimization, 2018.

[53] Alexander van der Grinten, Geert Custers, Duy Le Thanh, and Henning Meyerhenke. Fast dynamic updates and dynamic spgemm on mpi-distributed graphs, 2022.

[54] Jian Wu and Peter I. Frazier. The parallel knowledge gradient method for batch bayesian optimization, 2018.

[55] Antonio Benitez-Hidalgo, Antonio J. Nebro, Jose Garcia-Nieto, Izaskun Oregi, and Javier Del Ser. jmetalpy: a python framework for multi-objective optimization with metaheuristics, 2019.

[56] Shiwei Zhang, Lansong Diao, Chuan Wu, Siyu Wang, and Wei Lin. Accelerating large-scale distributed neural network training with spmd parallelism. In *Proceedings of the 13th Symposium on Cloud Computing*, pages 403–418, 2022.

[57] Yoni Choukroun, Shlomi Azoulay, and Pavel Kisilev. Adaptive consensus gradients aggregation for scaled distributed training, 2024.

[58] Janine Glänzel, Andreas Naumann, and Tharun Suresh Kumar. Parallel computing in automation of decoupled fluid-thermostructural simulation approach, 2019.

[59] Soumyadip Ghosh, Bernardo Aquino, and Vijay Gupta. Eventgrad: Event-triggered communication in parallel machine learning, 2021.

[60] Mark Stillwell, David Schanzenbach, Frédéric Vivien, and Henri Casanova. Resource allocation using virtual clusters, 2010.

[61] Haoze Wu, Alex Ozdemir, Aleksandar Zeljić, Ahmed Irfan, Kyle Julian, Divya Gopinath, Sadjad Fouladi, Guy Katz, Corina Pasareanu, and Clark Barrett. Parallelization techniques for verifying neural networks, 2020.

[62] Farshid Farhat, Diman Zad Tootaghaj, and Mohammad Arjomand. Towards stochastically optimizing data computing flows, 2016.

[63] Pál András Papp, Georg Anegg, Aikaterini Karanasiou, and A. N. Yzelman. Efficient multiprocessor scheduling in increasingly realistic models, 2024.

[64] Mehrnoosh Shafiee and Javad Ghaderi. Scheduling parallel-task jobs subject to packing and placement constraints, 2020.

[65] Shubham Kumar Goswami and Konduri Aditya. An asynchronous discontinuous galerkin method for massively parallel pde solvers, 2024.

[66] Emre Ozfatura, Deniz Gunduz, and Sennur Ulukus. Gradient coding with clustering and multi-message communication, 2019.

[67] Eric C. Ni, Dragos F. Ciocan, Shane G. Henderson, and Susan R. Hunter. Efficient ranking and selection in parallel computing environments, 2015.

[68] Ciprian Mihai Dobre, Florin Pop, Alexandru Costan, Mugurel Ionut Andreica, and Valentin Cristea. Robust failure detection architecture for large scale distributed systems, 2009.

[69] Da Wang, Gauri Joshi, and Gregory Wornell. Efficient straggler replication in large-scale parallel computing, 2017.

[70] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. Pytorch distributed: Experiences on accelerating data parallel training. *arXiv preprint arXiv:2006.15704*, 2020.

[71] Giuseppe F. Italiano, Silvio Lattanzi, Vahab S. Mirrokni, and Nikos Parotsidis. Dynamic algorithms for the massively parallel computation model, 2019.

[72] Lei Ni and Aaron Harwood. An adaptive checkpointing scheme for peer-to-peer based volunteer computing work flows, 2007.

[73] Zhengyu Yang, Danlin Jia, Stratis Ioannidis, Ningfang Mi, and Bo Sheng. Intermediate data caching optimization for multi-stage and parallel big data frameworks, 2018.

[74] Tiffany Tuor, Shiqiang Wang, Kin K. Leung, and Bong Jun Ko. Online collection and forecasting of resource utilization in large-scale distributed systems, 2019.

[75] Ardhendu Mandal and Subhas Chandra Pal. An empirical study and analysis of the dynamic load balancing techniques used in parallel computing systems, 2011.

[76] Yu Zhang, Zixiao Wang, Jin Zhao, Yuluo Guo, Hui Yu, Zhiying Huang, Xuanhua Shi, and Xiaofei Liao. Graph for science: From api based programming to graph engine based programming for hpc, 2024.

[77] Yangyang Xu, Yibo Xu, Yonggui Yan, Colin Sutcher-Shepard, Leopold Grinberg, and Jie Chen. Parallel and distributed asynchronous adaptive stochastic gradient methods, 2022.

[78] Guido Schryen. Speedup and efficiency of computational parallelization: A unifying approach and asymptotic analysis, 2023.

[79] Chang Chen, Xiuhong Li, Qianchao Zhu, Jiangfei Duan, Peng Sun, Xingcheng Zhang, and Chao Yang. Centauri: Enabling efficient scheduling for communication-computation overlap in large model training via communication partitioning. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 178–191, 2024.

[80] Shenggui Li, Hongxin Liu, Zhengda Bian, Jiarui Fang, Haichen Huang, Yuliang Liu, Boxiang Wang, and Yang You. Colossal-ai: A unified deep learning system for large-scale parallel training. In *Proceedings of the 52nd International Conference on Parallel Processing*, pages 766–775, 2023.

[81] William Pettersson and Melih Ozlen. Multi-objective integer programming: Synergistic parallel approaches, 2018.

[82] Ning Lu, Z. Todd Taylor, David P. Chassin, Ross T. Guttromson, and R. Scott Studham. Parallel computing environments and methods for power distribution system simulation, 2004.

[83] Edward Givelberg. Process-oriented parallel programming with an application to data-intensive computing, 2014.

[84] Xin Wang and Hong Shen. A scalable deep reinforcement learning model for online scheduling coflows of multi-stage jobs for high performance computing, 2021.

[85] Kasra Yazdani and Matthew Hale. Asynchronous parallel nonconvex optimization under the polyak-lojasiewicz condition, 2021.

[86] Sixin Zhang, Anna Choromanska, and Yann LeCun. Deep learning with elastic averaging sgd, 2015.

[87] Bin Gu, Zhouyuan Huo, and Heng Huang. Asynchronous stochastic block coordinate descent with variance reduction, 2016.

[88] Yong-Xian Wang, Li-Lun Zhang, Wei Liu, Xing-Hua Cheng, Yu Zhuang, and Anthony T. Chronopoulos. Performance optimizations for scalable cfd applications on hybrid cpu+mic heterogeneous computing system with millions of cores, 2017.

[89] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, and Carlos Guestrin. Graphlab: A distributed framework for machine learning in the cloud, 2011.

Disclaimer:

SurveyX is an AI-powered system designed to automate the generation of surveys. While it aims to produce high-quality, coherent, and comprehensive surveys with accurate citations, the final output is derived from the AI's synthesis of pre-processed materials, which may contain limitations or inaccuracies. As such, the generated content should not be used for academic publication or formal submissions and must be independently reviewed and verified. The developers of SurveyX do not assume responsibility for any errors or consequences arising from the use of the generated surveys.