CS60002
Distributed Systems

Intermediate Report

22nd March, 2024

20CS10009 - Ashish Rekhani
20CS10023 - Deepsikha Behera
20CS10046 - Prerit Paliwal
20CS30056 - Umika Agrawal
20CS30063 - Aman Sharma
20CS30071 - Vibhu

Introduction

Parallelism in distributed systems has emerged as a critical area of research and development, driven by the increasing demand for efficient and scalable computation in modern computing environments. Distributed systems, characterized by multiple interconnected nodes working together to achieve a common goal, pose unique challenges and opportunities for achieving parallelism. With the advent of large-scale data processing, machine learning, and other compute-intensive tasks, the effective utilization of parallelism in distributed systems has become paramount for achieving high performance and scalability.

This report aims to provide a comprehensive survey of recent advancements in parallelism techniques for distributed systems, with a focus on their applications, challenges, and future directions. By summarizing and analyzing a wide range of research papers and contributions in this field, we aim to provide insights into the state-of-the-art approaches, trends, and open research questions in parallelism for distributed systems.

The survey covers various aspects of parallelism, including model parallelism, data parallelism, task parallelism, and pipeline parallelism, among others. These techniques are employed in diverse distributed computing scenarios, ranging from traditional cluster computing and cloud computing to emerging paradigms such as edge computing and serverless computing. Additionally, the survey explores the application of parallelism in different domains, such as large-scale data analytics, deep learning, scientific computing, and real-time processing. Overall, this survey aims to serve as a valuable resource for researchers, practitioners, and enthusiasts interested in understanding and advancing the state-of-the-art in parallelism for distributed systems.

Survey Methodology

In order to compile a comprehensive survey of recent advancements in parallelism for distributed systems, a systematic methodology was employed to identify relevant research papers and contributions. The methodology involved several key steps, including literature search, paper selection criteria, and keyword analysis. Below, we outline the methodology used in this survey:

1. Literature Search: Firstly, we searched across various academic databases, including IEEE Xplore, ACM Digital Library, Google Scholar, and arXiv. The following combination of relevant keywords related to parallelism and distributed systems were used mostly:

- Parallelism in distributed systems
- Distributed computing
- Parallel computing
- Distributed systems architecture
- Distributed data processing
- Distributed machine learning

2. Paper Selection Criteria: After finding a few papers, we screened papers based on:
   - Relevance to the topic of parallelism in distributed systems.
   - Publication in reputable peer-reviewed journals, conference proceedings, or technical reports.
   - Recent publication date (typically within the last 5-10 years)

Why is parallelism still considered an unsolved problem despite several existing works?

Parallelism remains an unsolved problem despite several existing works due to a combination of factors inherent to the complexities of distributed systems, deep learning models, and file systems, as outlined in the provided context:

1. **Complexity and Heterogeneity:** The complexity of implementing effective parallelism in distributed systems involves synchronization, load balancing, and fault tolerance. Moreover, the heterogeneity of workloads across distributed file systems adds another layer of complexity. [2]
2. **Resource Constraints and Dynamic Environments:** Despite advancements in hardware capabilities, resource constraints such as CPU, memory, and network bandwidth still pose limitations on the degree of parallelism that can be effectively exploited. Additionally, distributed systems operate in dynamic environments where workloads, resource availability, and network conditions can change rapidly, requiring adaptive mechanisms to maintain efficiency.[2][4]
3. **Trade-offs and Optimization Challenges:** Achieving efficient parallelism involves navigating complex trade-offs between factors like consistency, latency, throughput, and fault tolerance. This challenge is particularly evident in deep learning models, where optimizing parallelization strategies requires considering various factors such as model structure, data distribution, and hardware characteristics. [3][8]
4. **Integration Challenges:** Integrating parallelism into existing architectures and workflows without disrupting functionality or introducing additional complexities can be challenging. Compatibility issues, interoperability concerns, and deployment overheads may hinder the adoption of parallelism-enhancing solutions. [2]
5. **Manual Parallelization and Limited Strategies:** Many existing approaches to parallelization, rely on manual creation of parallelization plans or limited automated techniques. This manual effort is labor-intensive, error-prone, and often suboptimal, hindering scalability and limiting exploration of complex parallelization strategies. [3][5]

6. **Ongoing Research and Development:** Despite significant advancements in parallel and distributed computing, ongoing research and development are required to explore new techniques, algorithms, and architectures that can further improve the parallelism and scalability of distributed systems, deep learning models, and file systems. [2][3][7]

The unsolved nature of parallelism stems from the inherent complexities, resource constraints, dynamic environments, trade-offs involved, integration challenges, and the continuous need for research and development in these domains, as evidenced by the challenges addressed in the papers under consideration.

What are the new invensions in the hardware technology that motivates to look into parallel computing from a different perspective?

Based on the information provided in the three papers, we can synthesize a comparison of the advancements in parallel computing technologies and their applications:

1. **Parallel Computing Approaches:**
   - SDPipe[8] focuses on heterogeneous GPU clusters and dynamic resource provisioning, emphasizing the importance of optimizing parallelization schemes for diverse hardware environments and dynamic resource availability.
   - TopoOpt[9] explores specialized accelerators, high-bandwidth interconnects, non-volatile memory, and quantum computing, highlighting a broader spectrum of hardware innovations that improve parallel computation tasks.

2. **Hardware Technologies:**
   - SDPipe[8] discusses advancements like heterogeneous GPU clusters and high-speed inter-GPU connections, crucial for optimizing parallel computing strategies and reducing communication overheads.
   - TopoOpt[9] introduces specialized accelerators, high-bandwidth interconnects, non-volatile memory, and quantum computing, emphasizing the significant performance improvements and scalability enhancements offered by these technologies.

3. **Applications and Performance:**
   - SDPipe[8] addresses the challenges of heterogeneous environments and dynamic resource provisioning, focusing on improving overall training performance in deep learning model training.
   - TopoOpt[9] highlights the potential for revolutionizing parallel computing systems and inspiring novel parallelization techniques tailored to specific hardware architectures.
   - Efficient FPGA-based Accelerator for Post-Processing in Object Detection[4] introduces a pioneering FPGA-based accelerator designed to enhance the post-processing stage of YOLO object detection algorithms, achieving remarkable speedups and minimal latency compared to traditional CPU-based approaches.

4. **Parallelization Techniques:**
   - SDPipe[8] and TopoOpt[9] discuss various parallelization schemes and techniques suited for different hardware architectures, emphasizing the importance of optimizing parallel computation tasks.
   - Efficient FPGA-based Accelerator for Post-Processing in Object Detection[4] demonstrates the efficacy of parallel microarchitecture and innovative design in achieving unprecedented speedups, particularly in bounding box processing for object detection algorithms like YOLO.

5. **Future Directions:**
   - SDPipe[8] and TopoOpt[9] suggest the importance of rethinking parallel computing strategies and exploring novel parallelization techniques to harness the full potential of modern computing infrastructure.
   - Efficient FPGA-based Accelerator for Post-Processing in Object Detection[4] sets a benchmark for real-time object detection by leveraging FPGA-based acceleration, paving the way for future advancements in the field.

In the era of ML and DL algorithms, how does parallelization help?

In the era of ML and DL algorithms, parallelization plays a crucial role in enhancing efficiency and scalability.

Orca[7] propose a distributed serving system specifically tailored for large language models (LLMs) like GPT-3. They address the challenges posed by the autoregressive nature of LLMs, where each token generation depends on previous tokens, leading to fine-grained dependencies and variable execution times. Orca employs parallelization techniques to mitigate these challenges:

Iteration-Level Scheduling: Orca introduces iteration-level scheduling, breaking down requests into individual iterations (token generation steps) and scheduling them independently. This approach allows Orca to return partial results to clients as soon as iterations complete, improving system responsiveness.

Selective Batching: Traditional batching techniques may not be suitable for LLMs due to fine-grained dependencies. Orca utilizes selective batching to identify independent operations within iterations and batch them together, optimizing resource utilization without introducing unnecessary dependencies.

By leveraging these parallelization techniques, Orca achieves significant improvements in throughput and tail latency compared to traditional serving systems, making it more efficient in handling LLM outputs.

On the other hand, **Alpa**[3], a compiler for automating model-parallel training of large DL models. Alpa addresses the challenge of scaling out complex DL models on distributed compute devices by optimizing parallelization at both intra-operator and inter-operator levels. Key components and contributions of Alpa include:

Hierarchical View of Parallelism: Alpa distinguishes between intra-operator and inter-operator parallelisms, providing a hierarchical space for exploring various parallelization strategies efficiently.

Compilation Passes: Alpa employs compilation passes to derive optimized parallel execution plans at different levels of the parallelism hierarchy, considering model structure, compute requirements, and communication patterns.

Orchestration: Alpa orchestrates parallel execution within operators and between operators across distributed compute devices, ensuring efficient resource utilization and coordination.

Automated Optimization: Alpa automates the optimization process, making parallelization more accessible and less labor-intensive for model developers, thereby democratizing distributed model-parallel learning.

Through detailed experimental evaluations, Alpa demonstrates competitive performance with state-of-the-art distributed training systems across various models, showcasing its effectiveness in automating parallelization optimization for large-scale DL models.

These approaches contribute to advancing the field by addressing the unique challenges posed by complex models and large datasets.

What are the techniques that are explored in the recent literature to support parallelization in distributed ML/DL models?

In recent literature, several techniques have been explored to support parallelization in distributed ML/DL models, as evidenced by the following two papers that deal with distributed ML/DL models we have gone through:

Data Parallel Actors (DPA)[6]:

Actor-based architecture: DPA introduces a programming model based on actors, which encapsulate both data and functionality. This architecture simplifies the

development of scalable query serving systems by breaking down complex systems into manageable, single-node actors.
Parallel operators for communication: Instead of traditional message passing, DPA employs parallel operators for communication between actors. This structured approach simplifies communication and reduces development overhead.
Automatic distribution and management: DPA handles the distribution of actors across a cluster of machines automatically, along with managing fault tolerance and elasticity, freeing developers from manual management tasks.
Advantages: DPA reduces development complexity, simplifies communication, and automates distribution and management of actors.
Limitations: While DPA streamlines development, it may introduce overhead from abstraction and require developers to adapt to a new programming model. It may not be ideal for ultra-low latency systems or scenarios requiring frequent, fine-grained updates.

Efficient FPGA-based Accelerator for Post-Processing in Object Detection[4]:

Parallel microarchitecture: The accelerator leverages a parallel microarchitecture designed to enhance the post-processing stage of YOLO object detection algorithms. This architecture features parallel processing units for computing bounding box scores, decoding raw bounding boxes, and identifying object classes.
Parallel NMS microarchitecture: The true innovation lies in the parallelization of the Non-Maximum Suppression (NMS) unit, which significantly reduces latency and boosts post-processing speed by parallelizing the computation of Intersection over Union (IoU) and threshold comparison.
Efficiency: The FPGA-based accelerator achieves remarkable speedups compared to traditional CPU-based approaches, with minimal latency and processing times. The NMS unit, in particular, exhibits exceptionally low latency.
Advantages: The accelerator redefines the post-processing stage of object detection by dramatically improving speed and efficiency, paving the way for real-time applications.
These techniques demonstrate innovative approaches to parallelization in distributed ML/DL models, addressing challenges related to communication, computation, and efficiency. While DPA focuses on simplifying development and communication in distributed query serving systems, the FPGA-based accelerator targets performance optimization in object detection algorithms through parallel microarchitecture design. These advancements contribute to the broader goal of improving scalability, efficiency, and speed in distributed ML/DL systems.

References

Currently we have the following papers under review -

1. Li, Dacheng, et al. "**Lightseq: Sequence level parallelism for distributed training of long context transformers.**" arXiv preprint arXiv:2310.03294 (2023).
2. Kim, Jongyul, et al. "**Linefs: Efficient smartnic offload of a distributed file system with pipeline parallelism.**" Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles. 2021.
3. Zheng, Lianmin, et al. "**Alpa: Automating inter-and {Intra-Operator} parallelism for distributed deep learning.**" 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22). 2022.
4. Z. Guo, K. Liu, W. Liu and S. Li, "**Efficient FPGA-based Accelerator for Post-Processing in Object Detection**," 2023 International Conference on Field Programmable Technology (ICFPT), Yokohama, Japan, 2023, pp. 125-131, doi: 10.1109/ICFPT59805.2023.00019.
5. Schneider, Nadav, et al. "**Mpi-rical: Data-driven mpi distributed parallelism assistance with transformers.**" Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis. 2023.
6. Kraft, Peter, et al. "**Data-Parallel actors: A programming model for scalable query serving systems.**" 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22). 2022.
7. Yu, Gyeong-In, et al. "**Orca: A distributed serving system for Transformer-Based generative models.**" 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22). 2022.
8. Xupeng Miao, Yining Shi, Zhi Yang, Bin Cui, Zhihao Jia: **“SDPipe: A Semi-Decentralized Framework for Heterogeneity-aware Pipeline-parallel Training.”** Proc. VLDB Endow. 16(9): 2354-2363 (2023)
9. Weiyang Wang, Moein Khazraee, Zhizhen Zhong, Manya Ghobadi, Zhihao Jia, Dheevatsa Mudigere, Ying Zhang, Anthony Kewitsch: **“TopoOpt: Co-optimizing Network Topology and Parallelization Strategy for Distributed Training Jobs.”** NSDI 2023: 739-767