e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:05/Issue:10/October-2023 Impact Factor- 7.868 www.irjmets.com

SOFTWARE DATA STRATEGIES FOR NETWORK OPTIMIZATION SUPPORTING AI WORKLOADS

Sudheer Kandula*1, Sree Ranga Vasudha Moda*2

*1NVIDIA, Santa Clara, CA, USA.

*2Salesforce, San Francisco, CA, USA.

DOI : https://www.doi.org/10.56726/IRJMETS45318

ABSTRACT

In the contemporary landscape of Artificial Intelligence (AI), the convergence of advanced algorithms and massive datasets has ushered in a new era of capabilities and possibilities. At the heart of the modern Artificial Intelligence (AI) workloads lies the significance of data an indispensable resource that fuels the Artificial Intelligence (AI) revolution. Managing, processing, and extracting value at scale from large datasets in runtime demands sophisticated data infrastructure, storage solutions, and computational resources. One foundational component in the High-Performance Computing (HPC), AI infrastructure is Computer Networking, especially in layers such as LLMs (Large Language Models), big-data wrangling and computations. It serves as the backbone that enables efficient data movement, communication, collaboration, and operation of massive workloads with low latencies.

A well-designed software system should thrive for optimal utilization of network bandwidth that enhances the performance, speed, accuracy, and scalability of AI systems, allowing organizations to unlock the full potential of Artificial Intelligence (AI). This paper focuses on the data strategies needed to be embedded into our software systems, particularly in the scope of network bandwidth optimization and outputs a comprehensive comparison on a list of available technologies/techniques for each of the strategies listed.

Keywords: Network Bandwidth Optimization, Artificial Intelligence, Data Strategies.

I. INTRODUCTION

AI adoption in enterprises is exponentially growing globally, attributed to the wide variety of use cases around personalized recommendations, predictive insights, customer service and so on. Our research says that requirement for AI specialized infra, compute, high performing adaptive networks, is the need of the hour owing to the unconditional demand into AI markets.

II. NETWORK OPTIMIZATION SOFTWARE DATA STRATEGIES

Optimizing network bandwidth is crucial for efficient communication, especially in scenarios where resources are limited or expensive. Here are the techniques that can help achieve optimized network bandwidth for transferring data across distributed microservices/Data/AI systems.

1. Data Compression
2. Data serialization formats
3. Message Packing
4. Delta Encoding
5. Caching
6. Data Deduplication
7. Data Chunking
8. Header Optimization
9. Predictive Prefetching

Data Transfer Strategies

1. Data Compression: [1] [2] [3] Use data compression techniques to reduce the size of messages before transmitting them over the network. Common compression algorithms include gzip, snappy, zlib, and Brotli. This reduces the amount of data transferred, thus saving the network bandwidth. Choosing a right codec

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:05/Issue:10/October-2023 Impact Factor- 7.868 www.irjmets.com

depends on the requirement and trade off with compression vs speed. Our research shows the below comparison results across widely used codecs in the industry.

Table 1: Comparison of data compression techniques.

Measures	zlib	snappy	gZip	Brotli
Compression ratio	2:1 to 5:1	~6:1	~9:1	~11:1
Technique	LZ77 and Huffman's encoding	LZ77 and Huffman's encoding/ or arithmetic coding	LZ77 and Huffman's encoding	LZ77 algorithm, Huffman coding and 2nd order context modeling
CPU	Low	Low	High(2x)	Highest
Storage Space	Medium	High	Medium	Low
Speed/Throughput	Medium	Fast(~2-5x gZip) Compression 250MB/sec Decompression 500 MB/sec	Slow	Medium(Compression is slow and decompression is faster than gZip)
Network Bandwidth	Less	Highest	Medium	Less
Latency	High	Very less	High	Medium
Data loss	No	No	No	No
Checksum mechanism for integrity of data	No	CRC-32C checksum	CRC-32 checksum	Yes with Brotli framing
Splittable	No	Yes(With file formats like Parquet,ORC)	No	Yes
Applications	Transmission over the network (HTTP compression, SSH compression), compression in programming languages (e.g., Python's zlib module).	Real-time data processing, Streaming, Hadoop MapReduce, and other big data systems.	Web content compression (HTTP compression, serving compressed HTML, CSS, JavaScript), file archiving (e.g., .tar.gz files)	Web content compression (serving compressed assets like HTML, CSS, JavaScript), HTTP content encoding (supported by modern web browsers).

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:05/Issue:10/October-2023 Impact Factor- 7.868 www.irjmets.com

2. Data serialization formats: JSON, XML, Protocol Buffers [4] (protobuf), Thrift are the most popular all serialization formats used for data interchange in software systems. Each has its own strengths and weaknesses, making them suitable for different use cases. Here's a comparison of these formats based on various criteria: One should pick appropriate formats that could help in reducing the size of messages and consequently saving bandwidth.

Table 2: Comparison of data serialization techniques.

Criteria	XML	JSON	Protobuf	Thrift
Readability	Yes but Verbose	Yes, Easy	No, Binary & Compact	No, Binary & Compact
Serialization Efficiency	Less	Low	High	High
Schema Support	Yes	No	Yes	Yes
Performance	Slowest	Slow	Very fast	Very fast
Payload Size	Highest	High	Very less	Less
Network Bandwidth	Highest	High	Very less	Less
Ecosystem and adoption	Legacy applications with SOAP format	Commonly used for RESTful http APIs, configuration files, and web applications	Ideal for high-performance, cross-language data serialization, remote procedure call (RPC) frameworks(gRPC) , communication between cloud-native microservices or large-scale distributed systems.	Ideal for high-performance, cross-language data serialization, remote procedure call (RPC) frameworks, in the big data and storage domains

3. Message Packing: [5] Combine multiple smaller messages into a single larger message. Use batching of the resources and entities. This minimizes the overhead associated with individual message headers, improving efficiency by reducing the number of networks roundtrips.

4. Delta Encoding: [6] For big data that changes incrementally, transmit only the differences (delta) between consecutive versions of the data instead of sending the entire dataset. This is useful for scenarios like real-time collaborative editing. Popular data warehousing solutions such as Amazon Redshift, Snowflake, and Google Big Query support CDC as part of their data loading and transformation processes. They offer features for efficiently ingesting and processing change data into data warehouses.

5. Caching: [7] Implement client-side and server-side distributed caching mechanisms to avoid redundant data transfers. Cached data can be reused, reducing the need to fetch the same data repeatedly.

e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:05/Issue:10/October-2023 Impact Factor- 7.868 www.irjmets.com

Table 3: Comparison of Data Caching Techniques

Feature	Redis	Memcached	Hazelcast	Apache Ignite	Couchbase
Data Structures Supported	Versatile	Key-Value	Various	Various	Document
Performance	Excellent	High	Good	High	Good
Persistence Options(In-memory Data backup)	Yes	No	Yes	Yes	Yes
Publish-Subscribe	Yes	No	Yes	Yes	No
Network Bandwidth	Low to Moderate	Low to Moderate	Moderate to High	High	Moderate to High
Throughput (Operations per Second)	High	Very High	Good	Very High	Good
Latency	Very Low	Very Low	Low to Moderate	Low to Moderate	Low to Moderate
Scalability	Yes (Clustering)	Yes (Clustering/Multithreaded)	Yes (Built-in)	Yes (Built-in)	Yes (Clustering)
Use Cases	Caching, Real-time Analytics, Message Broker, Distributed Caching	Web Caching	Distributed Caching, Real-time Analytics	Real-time Analytics, Distributed Computing	Caching, Data Storage

6. Data Deduplication: [8] [9] Ensure we are not transferring redundant data, rather ensure we are deduplicating on all data operations. In scenarios where multiple clients may request the same data, implement deduplication techniques to serve the data once and share it among the clients, rather than sending the same data multiple times. Below are a few deduplication strategies which all could reduce network resources, compared.

Table 4: Comparison of Data Deduplication Strategies.

Deduplication Strategy	Key Characteristics	Applications	Advantages	Challenges
Content-Aware Deduplication	Analyzes the actual content of data to identify duplicates, even when data has been	Archiving, backup, cloud storage	Effective in handling modified or transformed data.	Computationally intensive

	transformed or slightly modified.			
Hashing	Uses hash functions to generate unique identifiers for data chunks, which are compared to identify duplicates.	Archiving, backup, cloud storage	Efficient and widely used for deduplication	Risk of hash collisions, especially with weak hash functions
Fixed-Size Blocks	Divides data into fixed-size blocks, comparing and storing duplicates at the block level.	Backup, network optimization	Simple and efficient for static data	May not be as effective for data with variable-size duplicates
Variable-Size Blocks	Divides data into variable-size blocks, optimizing deduplication for content of varying sizes.	Data synchronization, cloud storage	Efficient in handling variable-size data	Increased computational overhead
Data Fingerprints	Assigns unique fingerprints or checksums to verify data integrity and avoid transferring duplicate data.	Data transfer, backup	Efficient in eliminating duplicate data transfer	Computational overhead in generating fingerprints

7. Data Chunking: Break down large messages into smaller chunks and transmit them separately. This approach is particularly helpful when dealing with large files, streaming data, text/audio/video/image type of unstructured data. Below is the comparison on various chunking strategies that can be applied on unstructured data for numerous applications in AI, like Natural language processing, Generative AI.

Table 5: Comparison of Data Chunking strategies.

Strategy	Description	Applications
Fixed-Size Chunks	Divide unstructured data into equal-sized chunks to ensure consistent data transmission.	File transfers, data streaming, consistent bandwidth usage
Variable-Size Chunks (Adaptive)	Chunk unstructured data based on content, optimizing chunk sizes to minimize data transfer overhead.	Content-aware transmission, adaptive quality streaming
Document-Based Chunking	Divide unstructured data based on document boundaries to maintain document integrity during transmission.	Document sharing, web content delivery
N-gram Chunking	Divide text data into n-grams to enable partial retrieval and optimized transfer of relevant content.	Search engines, text analytics, efficient content transfer

Audio/Video Frame-Based Chunking	Divide audio and video data based on frames or frames of reference for efficient multimedia transmission.	Video streaming, multimedia content delivery
Keyframe Chunking	Chunks are created from one keyframe to the next, ensuring efficient navigation and reduced data transfer.	Video editing, fast-forward/rewind features
Segmentation for Adaptive Streaming [10]	Segment media data into varying-duration segments to optimize streaming quality and minimize data size.	Adaptive streaming (HLS, DASH)
Blob Chunking (Binary Data)	Divide binary data into equal or variable-sized chunks for efficient transmission of binary content.	Image storage, data transfer, multimedia content sharing
Content-Based Chunking	Analyze content and chunk data based on content changes to efficiently transfer relevant portions of data.	Multimedia content analysis, content-based retrieval

8. Header Optimization: Minimizing headers in the messages reduces overhead and ultimately reduces network bandwidth. If applicable, use binary headers instead of text-based headers. Below are few more techniques for optimizing headers.

- **HTTP/2 and HTTP/3:** Consider upgrading to HTTP/2 or HTTP/3, which are more efficient in terms of header compression and multiplexing, reducing header overhead.
- **Minimize Cookie Usage:** Cookies are commonly used for session management, but excessive use of cookies can increase header size. Minimize the number and size of cookies sent with each request.
- **Use Content Delivery Networks (CDNs)**: CDNs can optimize headers automatically. They handle tasks like compressing and caching resources and provide optimized headers to the client.
- **Use Content Compression:** Enable content compression by including the "Accept-Encoding" header in client requests and the "Content-Encoding" header in server responses.
- **Caching Headers:** Leverage caching headers like "Cache-Control" and "Expires" to instruct client browsers to cache resources locally. This reduces the need for repeated requests to the server.
- **Etag and Last-Modified Headers:** Use the "Etag" and "Last-Modified" headers to indicate whether a resource has changed. These headers enable conditional requests, reducing unnecessary data transfer.
- **Preconnect and Prefetch Headers:** Use "Link" headers to instruct the browser to preconnect to domains hosting resources and prefetch resources. This can improve resource loading times.
- **Connection Keep-Alive:** Enable HTTP keep-alive to allow multiple requests to be sent over a single connection, reducing the overhead of opening and closing connections for each request.
- **Progressive Rendering:** Implement progressive rendering by setting response headers to deliver critical resources early in the response, allowing the browser to display content as it loads.
- **Avoid Unnecessary Redirects:** Minimize the use of unnecessary redirects (HTTP 301 and 302 status codes), as they add overhead to request/response cycles.
- **Reduce DNS Lookups:** Limit the number of different domains for resources (e.g., scripts, styles, images) to reduce DNS lookups and improve loading times.

9. Predictive Prefetching: [11] Predict what data a client might need next and pre-fetch it, reducing the round trips, latency of subsequent requests. Few predictive prefetching techniques are listed below.

- **Adaptive Prefetching**: Continuously adapt and refine prefetching strategies based on real-time network conditions, user behavior, and performance feedback.
- **Lazy Loading**: Implement progressive loading of content, where initial content is delivered quickly, and non-essential content is loaded in the background. This approach can improve perceived performance.
- **Real-Time Bandwidth Monitoring**: Monitor real-time network bandwidth and latency to adjust prefetching strategies dynamically. If bandwidth is limited, reduce or defer prefetching.
- **Server Push**: In HTTP/2 and HTTP/3, use server push mechanisms to send resources to the client before the client explicitly requests them. This can help reduce the need for additional round-trip requests.
- **Resource Preloading**: Preload essential resources such as images, scripts, and stylesheets in web applications. Use browser hints like "prefetch," "preload," and "preconnect" to initiate early resource fetching based on user navigation patterns.
- **Content Delivery Networks (CDNs)**: Leverage CDNs that offer predictive prefetching capabilities. CDNs can automatically determine what resources should be cached and delivered to reduce the need for round-trip requests to origin servers.
- **Resource Caching**: Cache frequently accessed data or resources locally on the client or proxy servers to reduce the need for repeated network requests. This can be done using browser caches, local storage, or server-side caches.

III. CONCLUSION

Our research says that adapting to the above-mentioned data strategies could yield great benefits in optimizing network bandwidth specially in distributed data/AI ecosystems. The strategies compared could be very specific and can vary across a wide variety of use cases we have in transferring data across Software Systems/ Data stores. Each of those could be traded off depending on the requirements and chosen that could be appropriate for leveraging optimal network bandwidth in balance to other key metrics of operational compute metrics.

IV. FUTURE WORK

Network Optimization Strategies on Compute, Application, Hardware and Security fronts would be extensions to this research and coming up in the future.

V. REFERENCES

[1] Huffman, D. A., "A Method for the Construction of Minimum Redundancy Codes", Proceedings of the Institute of Radio Engineers, September 1952, Volume 40, Number 9, pp. 1098-1101.

[2] J. Ziv and A. Lempel, “A Universal Algorithm for Sequential Data Compression,” IEEE Transactions on Information Theory, May 1977, pp.337-343.

[3] R. N. Williams, “An Extremely Fast Ziv-Lempel Data Compression Algorithm,” Proceedings of the IEEE Data Compression Conference, IEEE Computer Society Press, April 1991, pp. 362-371.

[4] Currier, C. (2022). Protocol Buffers. In: Hummert, C., Pawlaszczyk, D. (eds) Mobile Forensics – The File Format Handbook. Springer, Cham. https://doi.org/10.1007/978-3-030-98467-0_9

[5] R. Friedman and R. van Renesse, "Packing messages as a tool for boosting the performance of total ordering protocols," Proceedings. The Sixth IEEE International Symposium on High Performance Distributed Computing (Cat. No.97TB100183), Portland, OR, USA, 1997, pp. 233-242, doi: 10.1109/HPDC.1997.626423.

[6] N. Samteladze and K. Christensen, "DELTA: Delta encoding for less traffic for apps," 37th Annual IEEE Conference on Local Computer Networks, Clearwater Beach, FL, USA, 2012, pp. 212-215, doi: 10.1109/LCN.2012.6423611.

[7] A. Balamash and M. Krunz, "An overview of web caching replacement algorithms," in IEEE Communications Surveys & Tutorials, vol. 6, no. 2, pp. 44-56, Second Quarter 2004, doi: 10.1109/COMST.2004.5342239.

[8] Qinlu He, Zhanhuai Li and Xiao Zhang, "Data deduplication techniques," 2010 International Conference on Future Information Technology and Management Engineering, Changzhou, 2010, pp. 430-433,

doi: 10.1109/FITME.2010.5656539.

[9] G. Lu, Y. Jin and D. H. C. Du, "Frequency Based Chunking for Data De-Duplication," 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Miami Beach, FL, USA, 2010, pp. 287-296, doi: 10.1109/MASCOTS.2010.37.

[10] M. Seufert, S. Egger, M. Slanina, T. Zinner, T. Hoßfeld and P. Tran-Gia, "A Survey on Quality of Experience of HTTP Adaptive Streaming," in IEEE Communications Surveys & Tutorials, vol. 17, no. 1, pp. 469-492, Firstquarter 2015, doi: 10.1109/COMST.2014.2360940.

[11] T. I. Ibrahim and Cheng-Zhong Xu, "Neural nets based predictive prefetching to tolerate WWW latency," Proceedings 20th IEEE International Conference on Distributed Computing Systems, Taipei, Taiwan, 2000, pp. 636-643, doi: 10.1109/ICDCS.2000.840980.