

Distributed Data Processing for Large-Scale Simulations on Cloud

Tianjian Lu, Stephan Hoyer, Qing Wang, Lily Hu, and Yi-Fan Chen*

Google Research

1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA

*Corresponding author: yifanchen@google.com

Abstract—The computational challenges encountered in the large-scale simulations are accompanied by those from data-intensive computing. In this work, we proposed a distributed data pipeline for large-scale simulations by using libraries and frameworks available on Cloud services. The building blocks of the proposed data pipeline such as Apache Beam and Zarr are commonly used in the data science and machine learning community. Our contribution is to apply the data-science approaches to handle large-scale simulation data for the hardware design community. The data pipeline is designed with careful considerations for the characteristics of the simulation data in order to achieve high parallel efficiency. The performance of the data pipeline is analyzed with two examples. In the first example, the proposed data pipeline is used to process electric potential obtained with a Poisson solver. In the second example, the data pipeline is used to process thermal and fluid data obtained with a computational fluid dynamic solver. Both solvers are in-house developed and finite-difference based, running in parallel on Tensor Processing Unit (TPU) clusters and serving the purpose of data generation. It is worth mentioning that in this work, the focus is on data processing instead of data generation. The proposed data pipeline is designed in a general manner and is suitable for other types of data generators such as full-wave electromagnetic and multiphysics solvers. The performance analysis demonstrates good storage and computational efficiency of the proposed data pipeline. As a reference, it takes 5 hours and 14 mins to convert simulation data of size 7.8 TB into Zarr format and the maximum total parallelism is chosen as 10,000.

Index Terms—Apache Beam, Data Pipeline, Distributed Computing, Simulation, Tensor Processing Unit, Zarr

I. INTRODUCTION

The computational simulation and data analytics are critical in almost all areas of science and engineering. In the electronics industry, advanced simulations enable the design and optimization with reduced prototyping cycles and costs; the predictive data analytics [1] allow extracting meaningful knowledge of a design from both simulation and measurement data. Large-scale simulations are often computationally challenging: in order to accurately resolve geometrical complexities and non-uniform material properties in an electronic design, a three-dimensional (3D) simulation is often preferred; the large problem size arising from the volumetric discretization imposes the computational challenges [2]; because broadband information is often required, the large linear system resulting from the volumetric discretization needs to be repeatedly solved for multiple frequencies or time steps, which exacerbates the computation burden; in addition, with computational electromagnetics (CEM) becoming mature, people

start to address multiphysics problems, which are even more computationally challenging [3].

The computational challenges encountered in the large-scale simulations are accompanied by those from data-intensive computing. The large-scale simulations running on high-performance computers are considered as a new type of data-generation instrument beside experiments and sensors [4]. Given the large amount of data being generated through the simulations, the data processing task becomes more and more challenging. In fact, the data-generation capabilities grow more rapidly than the computing capabilities [5]. Many efforts have been taken to improve the computational efficiency of data-intensive computing [4]–[9], to name a few: (1) minimizing the data movement across the memory hierarchy; (2) optimizing the communication strategy; (3) processing data in parallel; (4) developing high-speed and low-latency interconnects; and (5) co-designing the system components from hardware architecture to software.

Large-scale simulations benefit from efficient data processing: to make a prompt design decision from the simulations, the obtained data must be processed, visualized, and interpreted in an efficient manner; the knowledge derived timely from rich historic data in return reduces the amount of simulation task [10]–[14]. In this work, we proposed a distributed data pipeline for large-scale simulations by using libraries and frameworks available on Cloud services. The building blocks of the proposed data pipeline are commonly used in the data science and machine learning community. Our contribution is to apply the data-science approaches to handle large-scale simulation data for the hardware design community. The design of the data pipeline is based on the characteristics of simulation data. The simulation represents and emulates a physical system or process with computers, mathematical models, and numerical algorithms [15] and the obtained data describes physical quantities or fields. With domain decomposition methods, large-scale simulations run in parallel on multiple compute cores [2]. Correspondingly, the data is obtained in parallel and labeled by the identifier of a process, or process ID. In addition, the simulation data contains the temporal and/or spectral information of physical quantities. The implementation of the data pipeline is based on Apache Beam [16] and Zarr [17]. Beam is a unified, open-source programming model for building both batch- and streaming-data parallel-processing pipelines. By using Beam,

one can simply focus on the logical composition of the data processing task and bypass the low-level details of distributed computing. The orchestration of distributed processing is fully managed by the runner, for example, Dataflow on Google Cloud [18]. It is worth mentioning that Beam separates the programming layer from the runtime layer such that the driver program can be executed across various runners [19]. The processed simulation data is often represented as multidimensional arrays or tensors. In this work, the storage format of the output tensor in the data pipeline is Zarr [17], [20]. The physical representation of the Zarr output is a folder containing one metadata file and multiple chunk files. Zarr allows concurrent reading and writing, storage on a file system, and data compression before the storage.

The performance of the proposed data pipeline is analyzed with two examples. In the first example, the data pipeline is used to process electric potential distribution obtained with a Poisson solver. The Poisson solver is widely accepted to analyze electrostatic behaviors of interconnects [21] and perform circuit parameter extractions [22]. The second example handles data obtained from a computational fluid dynamic (CFD) solver. The CFD solver is commonly used for thermal and fluid flow analyses of integrated circuits at the chip, package, and board levels. Both solvers are in-house developed and finite-difference based, running in parallel on Tensor Processing Unit (TPU) [23] clusters and serving the purpose of data generation. It is worth mentioning that this work focuses on data processing instead of data generation. In addition, the proposed pipeline is designed in a general manner and can be used to process data from other types of data generators such as full-wave electromagnetic and multiphysics simulations. The performance analyses consists of both storage and computational efficiency analysis. The scaling analyses demonstrate good parallel efficiency of the proposed data pipeline.

II. DATA PIPELINE

In this section, we provide details on the proposed distributed data pipeline.

A. Simulation Data

The data in this work is acquired with a large-scale simulation running in parallel. Before building the pipeline, we need to understand the data and its characteristics.

First, the data describes physical quantities or fields. The simulation represents and emulates a physical system or process with computers [15]. To simulate the system or process computationally, one must have an appropriate mathematical model expressed by equations in terms of physical quantities or fields. For example, if the mathematical model describes the eddy current phenomena [2] in a voice-coil-actuator system such as a loudspeaker, a haptic sensor, or a camera module, the simulation data is magnetic flux density. If the simulation is used to identify the undesirable electromagnetic interference in a system consisting of high-speed routings and power delivery rails, the data is electric field and obtained by solving

the vector wave equation [2]. In modern electronic systems, multiple distinct physical processes governed by different physical laws are strongly coupled together at different scales. Therefore, the simulation data describes multiple fields. For example, when the thermal effect due to Joule heating becomes significant in an electronic system, the simulation data should include the temperature field as in an II-thermal co-simulation [24], [25]. If the integrated micro-channels becomes part of the cooling solution, the simulation data should also include velocity and pressure fields [26]. In addition, the simulation data should also include displacement and stress fields if the thermally-induced stress causes reliability-related concerns in an electronic system [27].

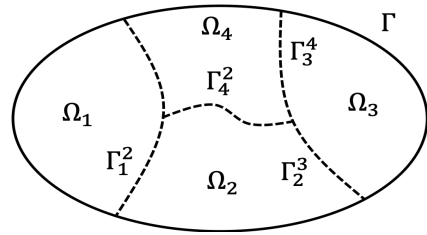


Fig. 1: A computational domain is decomposed into multiple subdomains [2].

Second, the simulation data is obtained in parallel on multiple processors. One way to enable the simulation in parallel is domain decomposition. The equations describing the mathematical model in a simulation are solved numerically. In general, a numerical algorithm that divides a computational domain into two or more subdomains [2] is considered as a domain decomposition method. For an unstructured mesh, which is often used by the finite element method, a decomposer such as METIS [28] divides the computational domain into a desired number of subdomains with excellent load balancing. For a structured mesh used by the finite difference method, the mesh partitioning and load balancing can be easily achieved and managed through the node indices. For parallel computing, subdomains are assigned to different processes. Therefore, the simulation data can be distinguished by the identifier of the process, or process ID. A process ID is defined by the virtual topology, specifying the communication pattern of a group of processes, instead of the physical topology, representing the connections among cores, chips, and nodes in the hardware. The virtual topology or the communication pattern can be graph- or grid-based. If the communication pattern is represented by a graph, the process ID is an integer representing a node on the graph. In many applications, the communication pattern is grid-based such as a ring or torus and the grid structure is defined by the number of dimensions and number of processes along each dimension. In a grid-based virtual topology, a process ID is defined by a set of Cartesian coordinates. It is worth mentioning that the parallel computing can also be enabled with spectral decomposition. In the electromagnetic analysis of an electronic system, the broadband information is often required. A spectral decompo-

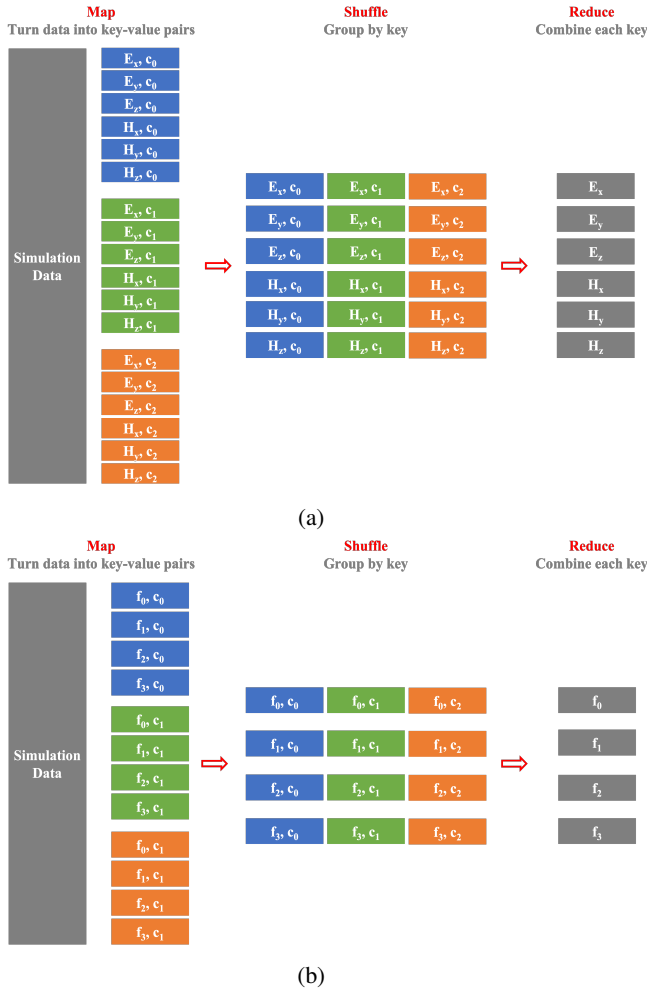


Fig. 2: Merge electric and magnetic fields obtained on different processes for visualization with an Map-Reduce operation, where (a) the field components and (b) the frequency sampling points as chosen as *keys*, respectively.

sition distributes the solutions of the vector wave equation in the frequency domain across multiple processors.

Third, the simulation data contains the temporal or spectral information of a physical system or a process, depending on whether the mathematical model is formulated in the time- or frequency-domain. In a multiphysics simulation, the data may contain both the temporal and spectral information, for example, electrical fields in the frequency domain and temperature fields in the time domain. This results from the multi-scale nature of the coupled physical events.

B. A Map-Reduce Operation over Simulation Data

As a technique for processing large-scale data, Map-Reduce is first published in 2004 [6]. Nowadays there are multiple prominent frameworks for distributed data processing, but the Map-Reduce operation is considered as the basis of these modern frameworks.

In this section, we use an example shown in Fig. 2 to illustrate the usage of a Map-Reduce operation through which

the simulation data are merged for visualization. We assume the simulation data are electric and magnetic fields, which are obtained with a 3D frequency-domain full-wave electromagnetic solver running in parallel on three processors. The process ID is denoted by (c_0, c_1, c_2) . A domain decomposition method is used by the simulation such that the obtained electric and magnetic fields on each process contains partial spatial information. Both the electric and magnetic fields, denoted by $\vec{E} = (E_x, E_y, E_z)$ and $\vec{H} = (H_x, H_y, H_z)$, respectively, are solved for at four frequency sampling points $f_i, i = 0, 1, 2, 3$. Therefore, a chunk of the simulation data can be labeled and distinguished by: (1) the field component, (2) the frequency sampling point, and (3) the process ID.

As shown in Fig. 2, there are three stages for a Map-Reduce operation: (1) at the map stage, the simulation data is turned into key-value pairs through a map function; in Fig. 2(a), the *key* is a field component; in Fig. 2(b), the *key* is a frequency sampling point; the *value* is simply a chunk of simulation data that contains partial spatial information and is obtained on one process; it is worth mentioning that the key is not necessarily unique; (2) at the shuffle stage, the chunks of simulation data is first redistributed and then grouped by the *keys*; and (3) at the reduce stage, the chunks of simulation data is merged by the *keys* with a reduce function.

Key-value pairs are important to the Map-Reduce operation. A larger number of key-value pairs in general allows finer-grained load balancing. For example, using field components as the *key* in Fig. 2(a) has higher parallelism than using the frequency sampling points in Fig. 2(b). An even higher parallelism can be achieved if both the field components and the frequency sampling points are used as the *keys*. However, higher parallelism incurs bigger cost, know as the overhead. All the operations shown in Fig. 2 are assigned to workers by a master program; the larger the number of key-value pairs, the more efforts it takes the master to keep track of the computation state and make scheduling decisions. The actual execution of a distributed data processing task is very complicated, especially when multiple Map-Reduce operations are chained together.

C. Beam Driver Program and Runner

By using Beam, one can simply focus on the logical composition of the data processing task and bypass the low-level details of distributed computing. The orchestration of distributed processing is fully managed by the runner.



Fig. 3: A pipeline example with Beam SDKs.

To create a pipeline for distributed data processing, one needs to write a driver program with Beam SDKs. The driver program contains the inputs, operations, outputs, and execution options for the pipeline runner. Beam currently

supports three language-specific SDKs: Java, Python, and Go. We used Python SDK in this work. Beam SDKs offer a number of abstractions such as `Pipeline`, `PCollection`, and `PTransform`. A distributed data set is represented by a `PCollection` object. A data processing operation is represented by a `PTransform` object. The `PTransform` object takes `PCollection` as input, operates on its elements, and possibly yields another `PCollection` object as the output. Because `PCollection` is immutable, the `PTransform` object does not modify its input `PCollection` object. A `Pipeline` object integrates the `PCollection` and `PTransform` objects into a directed acyclic graph. The driver program must have a `Pipeline` object. As shown in Fig. 3, the `Pipeline` object consists of multiple stages such as reading the simulation data, transforming the data, and writing to the output. The Beam SDKs provide a number of core transforms such as `ParDo`, `GroupByKey`, and `Combine`, etc.

Beam separates the programming layer from the runtime layer such that the driver program can be executed across various execution engines, or runners. Beam capability matrix [19] provides details on the capabilities of individual runners.

D. Parallel Output with Zarr

The post-processed simulation data is often represented as multidimensional arrays or tensors. In this work, the storage format of the output tensor is Zarr: the physical representation of the output is a folder containing one metadata file and multiple chunk files; and the chunk files are arranged by the fields (or components of a vector field) and coordinates from the simulation. The coordinates include the spatial, temporal, and spectral directions.

There are four major reasons for using Zarr. First, Zarr allows multiple processes to concurrently write to the output tensor. Many applications that build upon the simulation data are I/O bound: the data processing operations are in parallel and their computation complexity is often low. By having the tensor storage in parallel, it increases the throughput. In addition, the chunks of simulation data are often aligned between the computation and storage. Second, a Zarr tensor can also be read concurrently by multiple processes. In other words, the applications following the data processing task can access the Zarr tensor in parallel. Third, a Zarr tensor can be stored on a file system. The output tensor is often too large to fit into the main memory. For example, when six different fields (or field components, e.g. \vec{E} and \vec{H} with a 3D full-wave electromagnetic solver) of interest are computed with 512 processors for 1000 frequency points and each chunk of the data is 100 MB, the size of the output tensor is around 300 TB. Forth, each data chunk is compressed before storage in Zarr format. A number of libraries can be used for the compression.

III. PERFORMANCE ANALYSES

In this section, we analyze the performance of the proposed data pipeline with two examples. Because the data pipeline

TABLE I: Features of the simulation data obtained with the Poisson solver.

Field	Process ID	Time Step
Electric potential ϕ	(c_0, c_1, c_2)	t

has multiple stages and the required computing resources vary across stages, we use the maximum number of map worker threads among all worker processes as the control parameter when we conduct the scaling analysis. For simplicity, we name the control parameter as the maximum total parallelism.

A. Process Electric Potential Distribution

In this example, the simulation data represents the distribution of electric potential, which is obtained with a Poisson solver. The Poisson solver is based on the finite difference method and is implemented in TensorFlow. The Poisson solver runs in parallel on Tensor Processing Unit (TPU) [23] clusters. The features of the obtained simulation data are listed in Table I, which include (1) the field: electric potential ϕ ; (2) TPU process IDs (c_0, c_1, c_2) ; and (3) the time step t . The TPU logical processes are arranged in a 3D grid-based structure and a process ID represented by the coordinates along three dimensions. A key in the form of $(\text{field}, c_0, c_1, c_2, t)$ is assigned to a chunk of the simulation data, representing a field or a field component obtained on the process (c_0, c_1, c_2) at time step t . The computation domain is a slab and the mesh size is $32 \times 2044 \times 110376$ along x -, y -, and z -axis, respectively. The domain decomposition is one-way and is applied along the z -axis. There are a total number of 54 subdomains and each has the mesh size of $32 \times 2044 \times 2044$.


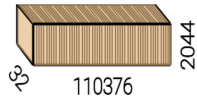
	Array	Chunk		1	
Bytes	26.02 TB	534.78 MB			
Shape	(901, 32, 2044, 110376)	(1, 32, 2044, 2044)		901	
Count	48655 Tasks	48654 Chunks			
Type	float32	numpy.ndarray			

Fig. 4: The data pipeline converts the electric potential distribution into Zarr tensors. The size of each data chunk is 534.8 MB before compression and 2.1 MB after being compressed into Zarr format.

1) *Storage efficiency*: As shown in Fig. 4, the data pipeline converts the electric potential into Zarr format. The physical representation of the output is a folder containing metadata file and chunks of simulation data. The original size of each data chunk is 534.8 MB, which is compressed into 2.1 MB with Zarr storage.

2) *Strong scaling*: In the strong scaling analysis, the amount of simulation data to be processed remains as 318 GB and the maximum total parallelism is proportionally increased. The strong scaling is shown in Fig. 5. The speed-up is defined by

$$\text{speed-up} = \frac{T_5}{T_N}, \quad (1)$$

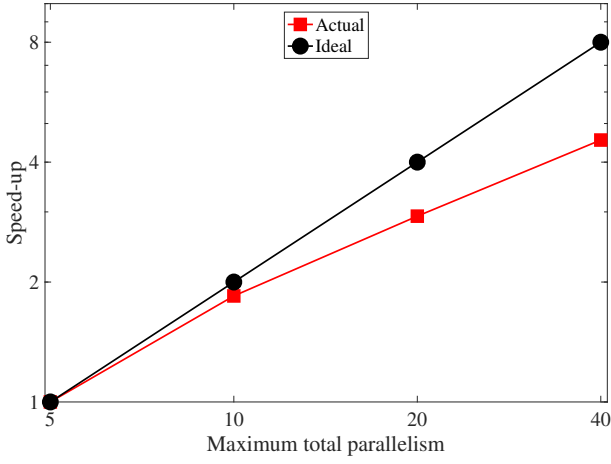


Fig. 5: Strong scaling of the proposed distributed data pipeline in processing electric potential. Maximum total parallelism is defined as the maximum number of map worker threads among all worker processes.

where N denotes the maximum total parallelism and T_5 represents the run time with the maximum total parallelism equal to 5. It can be seen from Fig. 5 that as the maximum total parallelism increases from 5 to 20, the strong scaling is approximately linear. When the maximum total parallelism exceeds 40, the gain of the speed-up saturates. It is because the work load is relatively small when the maximum total parallelism becomes greater than 40. As a reference, the runtime of the proposed data pipeline is 190.9 seconds when the maximum total parallelism is 40.

B. Process Thermal and Fluid Data

The simulation data in this example is obtained with a CFD solver. The features of the simulation data include (1) the fields: velocity field components u, v, w , pressure field p , a transported scalar Z , and temperature T ; (2) TPU process IDs (c_0, c_1, c_2); and (3) the time step t . The computation domain is a slab and the mesh size is $16 \times 2044 \times 65280$ along x -, y -, and z -axis, respectively. The domain decomposition is one-way and is applied along the z -axis. There are a total number of 64 subdomains and each has the mesh size of $16 \times 2044 \times 1020$. One subdomain is handled by one TPU core and there are 64 cores in total. For testing purpose, the CFD solver runs for 11 time steps.

1) *Storage efficiency*: The data pipeline converts five fields or field components obtained through the in-house CFD solver into Zarr format. Figure 6 shows the output Zarr tensors. Each Zarr tensor corresponds to one field. There are four coordinates associated with each field: x, y, z , and $time_step$. The physical representation of the output is a folder containing metadata file and chunks of simulation data. There are a total number of 704 data chunks associated with each Zarr tensor. The original size of each data chunk is 133 MB, which is compressed into 526 KB with Zarr storage. The original size of the simulation data in the test run is 470 GB, whereas in Zarr format, it is 1.8 GB.

In Fig. 6, the number of tasks is 705, one larger than the total number of chunks. This is due to the fact that one dummy data chunk is used to initialize the parallel Zarr storage.

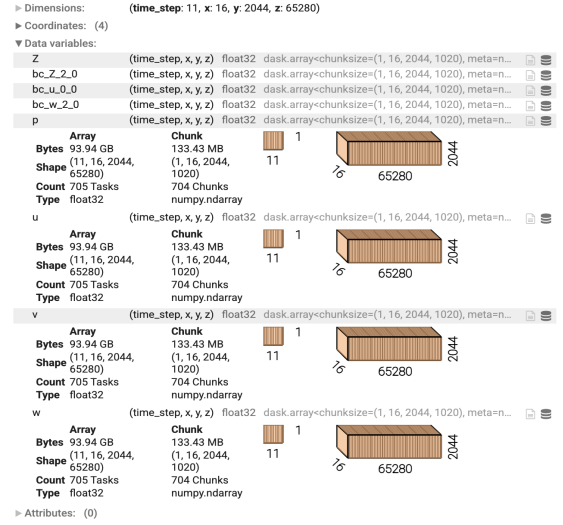


Fig. 6: The data pipeline converts the simulation data from a CFD solver on TPUs into Zarr tensors. The size of each data chunk is 133 MB before compression and 526 KB with compression in Zarr format.

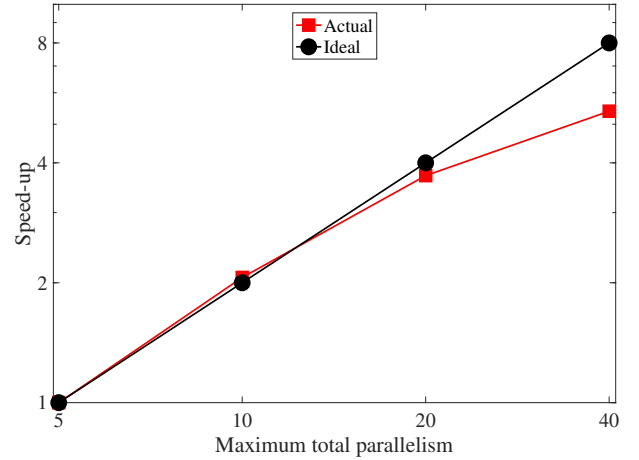


Fig. 7: Strong scaling of the proposed distributed data pipeline in processing thermal and fluid data. Maximum total parallelism is defined as the maximum number of map worker threads among all worker processes.

2) *Strong scaling*: In the strong scaling analysis, the amount of simulation data to be processed remain as 376 GB and the maximum total parallelism is proportionally increased. The strong scaling is shown in Fig. 7. It can be seen from Fig. 7 that as the maximum total parallelism increases from 5 to 20, the strong scaling is approximately linear. When the maximum total parallelism exceeds 40, the gain of the speed-up saturates. It is because the work load is considered as small for the maximum total parallelism greater than 40. As a reference, the

runtime of the proposed data pipeline is 362.1 seconds while processing the simulation data of 376 GB when the maximum total parallelism is chosen as 40.

3) *Weak scaling*: In the weak scaling analysis, the ratio between the amount of simulation data to be processed and the maximum total parallelism is kept the same. The weak scaling is shown in Fig. 8. As the maximum total parallelism increases from 5 all the way to 20, the runtime remains almost the same. As a reference, the runtime is 503.1 seconds when the size of simulation data is 188 GB and the maximum total parallelism equals 10. However, as the maximum total parallelism increases from 20 to 40, the runtime increases from 525.4 seconds to 671.5 seconds. The increase of runtime is due to the increase of work load at the stages other than the map stage. By using maximum total parallelism in the weak scaling analysis, the work load across all the threads remains the same only for the map operations.

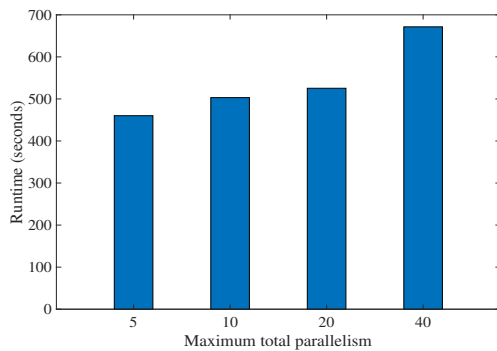


Fig. 8: Weak scaling of the proposed distributed data pipeline. Maximum total parallelism is defined as the maximum number of map worker threads among all worker processes.

4) *Large example*: We further apply the proposed pipeline to a larger example. The discretization to the 3D computational domain yields 8.03 billion nodes. The simulation data of 34 time steps are saved for data processing. As a reference, it takes 5 hours and 14 mins to convert the 7.8 TB simulation data into Zarr format. The maximum total parallelism is chosen as 10,000.

IV. CONCLUSION

We proposed a distributed data pipeline for large-scale simulations. We designed the data pipeline by carefully considering the characteristics of simulation data. The data pipeline is implemented with Beam and Zarr and can be executed on multiple environments offered by Cloud services. The performance analysis demonstrates good storage and computational efficiency of the proposed data pipeline.

REFERENCES

[1] T. Hey, S. Tansley, K. Tolle *et al.*, *The fourth paradigm: data-intensive scientific discovery*. Microsoft research Redmond, WA, 2009, vol. 1.
 [2] J.-M. Jin, *The finite element method in electromagnetics*. John Wiley & Sons, 2015.
 [3] J.-M. Jin and S. Yan, “Multiphysics modeling in electromagnetics: Technical challenges and potential solutions,” *IEEE Antennas and Propagation Magazine*, vol. 61, no. 2, pp. 14–26, 2019.

[4] J. Chen, A. Choudhary, S. Feldman, B. Hendrickson, C. Johnson, R. Mount, V. Sarkar, V. White, and D. Williams, “Synergistic challenges in data-intensive science and exascale computing: DOE ASCAC data subcommittee report,” 2013.
 [5] D. A. Reed and J. Dongarra, “Exascale computing and big data,” *Communications of the ACM*, vol. 58, no. 7, pp. 56–68, 2015.
 [6] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.
 [7] R. T. Kouzes, G. A. Anderson, S. T. Elbert, I. Gorton, and D. K. Gracio, “The changing paradigm of data-intensive computing,” *Computer*, vol. 42, no. 1, pp. 26–34, 2009.
 [8] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The hadoop distributed file system,” in *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. Ieee, 2010, pp. 1–10.
 [9] V. Borkar, M. Carey, R. Grover, N. Onose, and R. Vernica, “Hydracks: A flexible and extensible foundation for data-intensive computing,” in *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 2011, pp. 1151–1162.
 [10] Q. Huang and J. Fan, “Machine learning based source reconstruction for RF desense,” *IEEE Transactions on Electromagnetic Compatibility*, vol. 60, no. 6, pp. 1640–1647, 2018.
 [11] T. Lu, J. Sun, K. Wu, and Z. Yang, “High-speed channel modeling with machine learning methods for signal integrity analysis,” *IEEE Transactions on Electromagnetic Compatibility*, vol. 60, no. 6, pp. 1957–1964, 2018.
 [12] T. Nguyen, T. Lu, J. Sun, Q. Le, K. We, and J. Schut-Aine, “Transient simulation for high-speed channels with recurrent neural network,” in *2018 IEEE 27th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)*. IEEE, 2018, pp. 303–305.
 [13] J. Wen, X.-C. Wei, Y.-L. Zhang, and T.-H. Song, “Near-field prediction in complex environment based on phaseless scanned fields and machine learning,” *IEEE Transactions on Electromagnetic Compatibility*, 2020.
 [14] M. Swaminathan, H. M. Torun, H. Yu, J. A. Hejase, and W. D. Becker, “Demystifying machine learning for signal and power integrity problems in packaging,” *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 10, no. 8, pp. 1276–1295, 2020.
 [15] M. T. Heath, *Scientific Computing: An Introductory Survey, Revised Second Edition*. SIAM, 2018.
 [16] Apache Beam. [Online]. Available: <https://beam.apache.org/>
 [17] Zarr. [Online]. Available: <https://zarr.readthedocs.io/en/stable/>
 [18] Dataflow. [Online]. Available: <https://cloud.google.com/dataflow>
 [19] Beam capability matrix. [Online]. Available: <https://beam.apache.org/documentation/runners/capability-matrix/>
 [20] Zarr - scalable storage of tensor data for parallel and distributed computing. [Online]. Available: <https://zarr-developers.github.io/slides/scipy-2019.html>
 [21] T. Lu and J.-M. Jin, “Transient electrical-thermal analysis of 3-D power distribution network with FETI-enabled parallel computing,” *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 4, no. 10, pp. 1684–1695, 2014.
 [22] S.-C. Wong, G.-Y. Lee, and D.-J. Ma, “Modeling of interconnect capacitance, delay, and crosstalk in VLSI,” *IEEE Transactions on semiconductor manufacturing*, vol. 13, no. 1, pp. 108–111, 2000.
 [23] Cloud TPUs. [Online]. Available: <https://cloud.google.com/tpu/>
 [24] T. Lu and J.-M. Jin, “Electrical-thermal co-simulation for DC IR-drop analysis of large-scale power delivery,” *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 4, no. 2, pp. 323–331, 2013.
 [25] —, “Electrical-thermal co-simulation for analysis of high-power RF/microwave components,” *IEEE Transactions on Electromagnetic Compatibility*, vol. 59, no. 1, pp. 93–102, 2016.
 [26] T. Lu, F. Zhang, and J.-M. Jin, “Multiphysics simulation of 3-D ICs with integrated microchannel cooling,” *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 6, no. 11, pp. 1620–1629, 2016.
 [27] T. Lu and J.-M. Jin, “Coupled electrical–thermal–mechanical simulation for the reliability analysis of large-scale 3-D interconnects,” *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 7, no. 2, pp. 229–237, 2017.
 [28] G. Karypis and V. Kumar, “A fast and high quality multilevel scheme for partitioning irregular graphs,” *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.