Randomized SVD for Serverless Systems

Gabriel Raulet*†, Yen-Hsiang Chang*†
*Both authors contributed equally to this research.
†University of California, Berkeley
gabe.h.raulet@berkeley.edu, yenhsiangc@berkeley.edu

***Abstract*—Randomized singular value decomposition (SVD), a subroutine of large-scale principal component analysis, has been well developed in distributed computing. However, not everyone has access to high performance clusters. Fortunately, the relatively recent introduction of serverless computing offers the potential to close the accessibility gap. Therefore, we present an experimental study on randomized SVD for serverless systems. Prior works on serverless linear algebra suffer from high communication cost due to the lack of efficient collective communication primitives. To solve this issue, we integrate a newly proposed serverless message interface into a loosely coupled randomized SVD algorithm. Our evaluations demonstrate that high performance linear algebra kernels can be executed in the serverless setting with comparable performance and significantly better accessibility when compared to supercomputers. Furthermore, the long job queuing times associated with high performance clusters are completely bypassed by our approach. We believe that the increased accessibility our prototype demonstrates, along with its fast invocation time, show that the previously mentioned problems associated with high performance serverless computing can be overcome and provide benefit to users.**

I. Introduction

Large-scale principal component analysis (PCA) [1], [2] is a necessary component of many scientific and data analytic workloads, including bioinformatics [3], deep learning [4], and image compression [5]. Extensive research has been done on its subroutine, randomized singular value decomposition (SVD) [6], over many decades developing efficient parallel shared-memory and distributed algorithms [7]–[9]. It is increasingly the case that much of this research targets specific architectures and accelerators. While researchers develop faster and faster algorithms to run on specialized hardware and high performance computing (HPC) clusters, many scientists and data analysts not in the HPC community still have difficulty utilizing these advantages in their own work. The relatively recent introduction of cloud computing and its many variants offers the potential to close the accessibility gap. Serverless computing [10]–[12] is a paradigm in cloud computing that has attracted interest due to its low startup cost, relative ease of configuration, and flexible scalability. Cloud computing helps abstract away the need for organizations or individuals to handle the tedious management and provisioning of servers. Serverless computing takes this a step further. Through the concept of Function as a Service (FaaS), cloud providers abstract away the need to maintain a running server. Instead, computation is performed through stateless functions that scale elastically to the demands of applications. Crucially, it lets researchers perform highly scalable and parallel computations in a cost-effective manner without exposing them to the gritty and frustrating issues one deals with when interacting with a cluster.

Effectively making use of distributed randomized SVD kernels for serverless systems is not a trivial problem however. The most difficult problem to deal with is the stateless execution environment of FaaS, and the burden this imposes on efficient communication. Standard designs of distributed algorithms rely heavily on a stateful execution environment where it is assumed that compute nodes can communicate directly with each other in a point-to-point fashion, and also using collective primitives like gather and broadcast through the help of Message Passing Interface (MPI). In contrast, previous work [13] has pointed out that serverless linear algebra kernels suffer from high communication overheads due to the lack of efficient collective communication primitives. Nevertheless, distributed linear algebra kernels nowadays have already been optimized for minimizing communication cost [14]. Therefore, the central challenge of randomized SVD for serverless systems is how to reduce communication overheads on top of the fact that the distributed design is already optimized for minimizing communication cost.

In this paper, we address this central challenge by integrating a newly proposed serverless message interface [15] into a loosely coupled distributed randomized SVD algorithm [16], [17], where the former provides efficient collective communication primitives for serverless systems and the latter redesigns the algorithm to reduce its reliance on collective communication primitives.

We then present an experimental study of our integrated implementation. We analyze the error incurred by our randomized SVD approach. The runtime of both approaches is measured in a series of experiments. To verify that our approach is feasible, we analyze the total runtime of both approaches in terms of span and work. To demonstrate the improvement in total wait time the serverless implementation provides, we measured time from submission to completion for both approaches. Our evaluations show that high performance serverless linear algebra is a feasible alternative to traditional HPC solutions. Furthermore, our research demonstrates that it is technically feasible to obtain a competitive serverless linear algebra implementation of a widely used machine learning algorithm with simple modifications to a traditional HPC implementation targeting a supercomputer. Finally, our work addresses a common issue with HPC workloads: long job queues. Using FaaS mostly eliminates this hidden time cost,

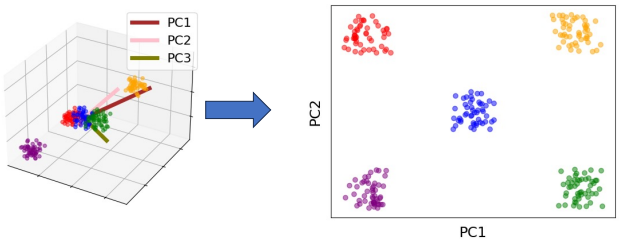

Fig. 1. Illustration of PCA for dimensionality reduction. The original data in three dimensional space is embedded into two dimensional space by keeping the first two PCs.

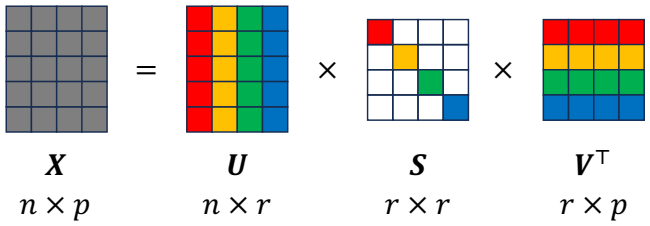

Fig. 2. Illustration of SVD. Matrix X is decomposed into U, S, and $V^\top$ where the columns of U are the left-singular vectors of X, the diagonal entries of S are singular values of X in decreasing order, and the columns of V (rows of $V^\top$) are the right-singular vectors of X.

making it easier for users to obtain results quickly.

II. Background

A. Principal Component Analysis

Given a columnwise standardized data matrix $X \in \mathbb{R}^{n\times p}$ with n samples in p dimensional space, PCA finds a lower dimensional space that maximizes the variance of the projected data among subspaces of a given dimension $k \ll p$ (Figure 1). This is achieved by transforming X into the so-called principal components (PCs) where the first k of them retain most of the variation present in X [18]. Mathematically, this can be solved by doing eigendecomposition on the $p \times p$ symmetric covariance matrix $C = X^\top X/(n-1)$, i.e., factorizing C as $VDV^\top$ where D is a $p\times p$ diagonal matrix with eigenvalues in decreasing order and columns of V contain the corresponding normalized eigenvectors. Then, the first k columns of V form the orthogonal basis of the targeted k-dimensional subspace. In other words, the j-th column of XV is the j-th PC and its corresponding explained variance is the j-th eigenvalue λ_j divided by the sum of all eigenvalues. However, computing eigendecomposition of $X^\top X/(n-1)$ is numerically unstable, and the solution is to use SVD on X instead.

B. Singular Value Decomposition

SVD [19] is a generalization of eigendecomposition for arbitrary matrices. Given $X \in \mathbb{R}^{n\times p}$ defined above with $r = \min\{n,p\}$, the SVD of X is $X = USV^\top$, where U and V are $n \times r$ and $p \times r$ matrices with orthogonal columns, respectively, and S is an $r \times r$ diagonal matrix with singular values σ_j in decreasing order (Figure 2). Then, we have $C = X^\top X/(n-1) = V\frac{S^2}{n-1}V^\top$. By comparing with the eigendecomposition $C = VDV^\top$, we observe that SVD and eigendecomposition share the same V and that $D = S^2/(n-1)$ and $XV = USV^\top V = US$. In other words, the PCs of X are given by US. Also, eigenvalues of C and singular values of X are related by $\lambda_j = \sigma_j^2/(n-1)$.

Numerically stable SVD algorithms [20]–[22] and its distributed version [23] have been developed in the past decades. Nevertheless, their cubic time complexity with respect to n or p makes them not suitable for large-scale PCA where n and p are typically several millions.

C. Randomized Singular Value Decomposition

One solution to overcome the cubic time complexity is to use randomized SVD. Since PCA only requires to compute the first $k \ll p$ PCs, we don't need to solve for the full SVD $X = USV^\top$. Instead, we only need to compute the truncated SVD $X \approx U_k S_k V_k^\top$, where U_k contains the first k columns of U, S_k is the leading principal $k\times k$ submatrix of S, and V_k contains the first k columns of V (Figure 3). However, exactly computing the truncated SVD is as hard as computing the full SVD, and here is the place where randomized SVD come into play.

Randomized SVD follows the intuition of random projections from the Johnson–Lindenstrauss lemma [24] with the fact that the failure probability decreases super-exponentially when oversampling the required dimension k. An algorithm is given in [6]. The first step is to generate an $n\times 2k$ random Gaussian matrix Ω where $2k$ corresponds to the oversampling parameter and find an orthogonal basis Q of the space spanned by the $2k$ columns of $XX^\top X\Omega$. Then, do an SVD on the projected data $Q^\top X$ to get $Q^\top X = \widetilde{U}_{2k}S_{2k}V_{2k}$ and an approximated rank-$2k$ truncated SVD $X \approx QQ^\top X = (Q\widetilde{U}_{2k})\widehat{S}_{2k}\widehat{V}_{2k} = \widehat{U}_{2k}\widehat{S}_{2k}\widehat{V}_{2k}$ where $\widehat{U}_{2k} = Q\widetilde{U}_{2k}$. Based on the oversampling property, we then get an approximated rank-k truncated SVD $X \approx \widehat{U}_k\widehat{S}_k\widehat{V}_k$ by extracting the leading columns and submatrix with low failure probability. See [6] for more details, bounds, and refinements.

Let's now have an analysis on the time complexity of randomized SVD. Recall that k is typically no more than 10 in PCA applications, implying $2k \ll p$ and $2k \ll n$ in large-scale PCA. The main bottleneck is computing $XX^\top X\Omega$ since SVD is only done on a $2k \times p$ matrix $Q^\top X$. Multiplying out $XX^\top X\Omega$ from the right hand side has quadratic complexity with respect to n and p if treating $2k$ as a constant. Overall, randomized SVD has quadratic time complexity hence being capable of handling large-scale PCA with n and p being several millions.

D. Serverless Computing

Serverless computing [10]–[12], a sub-category of cloud computing, is a computing paradigm that abstracts away the need for maintaining servers. Figure 4 shows the workflow

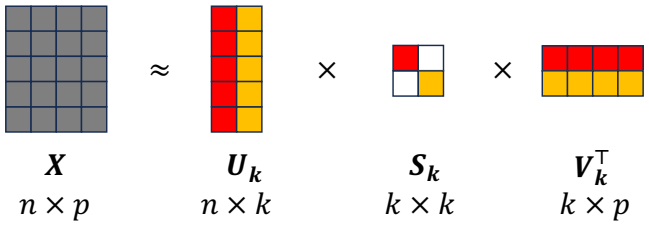

Fig. 3. Illustration of truncated SVD. Instead of keeping all singular values and singular vectors in SVD, the rank-k truncated SVD only keeps the first k largest singular values and their corresponding singular vectors.

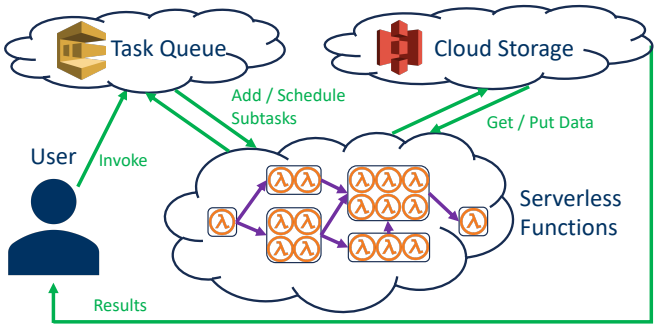

Fig. 4. Workflow of serverless computing using the AWS Ecosystem [25] as an example. When the user (or an event) triggers a task, the task is added to the task queue and scheduled as serverless functions. Due to their stateless nature, serverless functions can only read the inputs from the cloud storage and also write the outputs to the cloud storage. Subtasks generated by the serverless functions will be added back to the task queue and be scheduled as long as there is enough resource. Once the task queue is empty, indicating the whole workflow is done, the user can fetch the final outputs from the cloud storage.

of serverless computing using the Amazon Web Services (AWS) Ecosystem [25] as an example. Blue labels show the components involved in serverless computing, and green labels convey how different components interact. Note that serverless functions are typically isolated from each other, and communications are usually realized through the help of cloud storage, e.g., Amazon S3 [26], where one function puts data there and the other gets the data back. Purple arrows show the dependencies among serverless functions, which are maintained by the task queue, e.g., Amazon SQS [27]. The task queue has the ability to perform auto-scaling based on the requirement of each serverless function.

Through the concept of FaaS, computation in serverless computing is performed through stateless functions that scale elastically to the demands of applications, which is suitable for irregular and imbalanced workloads. Also, the fine-grained billing model allows user to only pay for the computing time and memory allocated for the serverless functions. The recent growth of serverless computing offers the potential to close the accessibility gap for people without knowledge of server provisioning.

E. Serverless Linear Algebra

Directly migrating linear algebra kernels from HPC to serverless computing is, however, not an easy task. Although existing general serverless programming frameworks [28], [29] simplify serverless development, their generality runs in opposite direction of performance-oriented linear algebra kernels. There are specialized serverless linear algebra frameworks [13], [30] with tens of thousands lines of code to deal with task scheduling and resource management, but they still suffer from high communication overheads due to the lack of efficient collective communication primitives. Nevertheless, linear algebra kernels nowadays have already been optimized for minimizing communication cost [14]. This issue leaves serverless linear algebra an interesting field worth investigating.

F. FaaS Message Interface

FaaS Message Interface (FMI) [15] is a recently proposed message passing library aims for providing MPI-like primitives for serverless systems. The motivation is to get rid of frequently interacting with secondary cloud storage and to have fast and cheap message passing instead. However, one issue of function isolation in serverless systems is that endpoints are hidden behind Network Address Translator (NAT) gateways [31], making direct communication between serverless functions not a simple task. FMI circumvents this obstacle by utilizing the hole punching technique [32] where a relay server is used to map and exchange IP addresses of serverless function instances. Afterward, serverless functions are able to communicate directly and build collective communication primitives on top of direct communication.

G. Loosely Coupled Singular Value Decomposition

The bottleneck of standard numerically stable SVD algorithms is to reduce the matrix X into a bidiagonal matrix [20]–[22]. This step is inherently coupled in a distributed setup when X is partitioned block-by-block since multiple orthogonal transformations need to be applied one after another. Loosely coupled SVD [16], [17], on the other hand, redesigns the algorithm, distribute X in a columnwise manner, and merge the results using a perfect binary tree. Figure 5 illustrates how loosely coupled SVD works when there are 2^2 processors.

Firstly, the input matrix X is partitioned in a columnwise manner and distributed across processors. Each processor at leaf nodes performs standard SVD on the columns being assigned. Then, each internal node in the binary tree merges results from its two children nodes to get the SVD of all the columns within its subtree. Therefore, the root node will have the full SVD of X at the end. When only computing the rank-k truncated SVD of X, each node in the binary tree does not need to hold the full SVD, but only needs to hold the rank-k truncated SVD of all the columns within its subtree. Nevertheless, loosely coupled SVD sacrifices its accuracy with a factor related to the amount of parallelism.

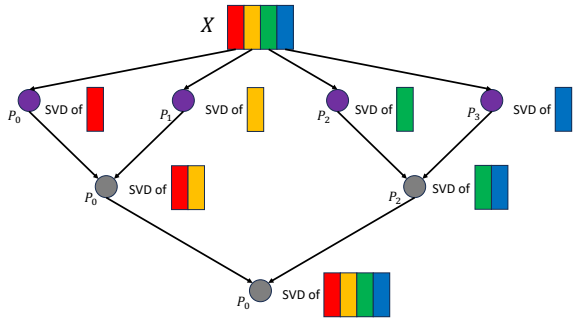

Fig. 5. Illustration of loosely coupled SVD. Suppose there are 2^2 processors organized as leafs in a perfect binary tree, and the input matrix X is first partitioned in a columnwise manner. Then, each processor performs SVD on the columns assigned. After that, processor 0 merges its result with that from processor 1, and processor 2 merges its result with that from processor 3. Finally, processor 0 merges its result with that from processor 2 and returns the SVD of X.

By distributing the input matrix X in a columnwise manner and organizing dependencies as a perfect binary tree, loosely coupled SVD reduces its reliance on collective communication primitives when compared with its numerically stable counterpart. Also, randomized SVD can be easily implemented following this scheme since the randomization part is embarrassingly parallelizable.

III. Related Work

A. Singular Value Decomposition

SVD is the Swiss Army knife of linear algebra. The first complete statement dates back to [19] and the first reliable algorithm is presented in [20]. Since then, SVD has been extensively studied [21], [33], [34] and incorporated into linear algebra libraries [22], [35]. Parallel and distributed SVD [23], [36] have also been considered for specialized hardware and HPC clusters, with a shift of focus from high accuracy to communication avoiding algorithms [14], [37]. Although already optimized for minimizing communication cost, distributed SVD still suffers from high communication overheads in serverless systems hence loosely coupled SVD [16], [17] is proposed to simplify dependencies in computation by sacrificing accuracy in order to reduce overheads. SVD also suffers from its inherent cubic time complexity when large-scale data sets come into play where sizes of matrices are several millions. One solution is to use randomized SVD [6], [7] to approximate SVD in a low dimensional space. The other solution is to use iterative methods [38], [39] to approximate SVD until desired accuracy is achieved. The recent growth of serverless computing makes large-scale serverless SVD a worth investigating problem since SVD needs to be redesigned to overcome constraints imposed by this new paradigm.

B. Serverless Computing

Serverless computing [10]–[12] is a recently introduced cloud computing paradigm that abstracts away the need for maintaining servers. It is initially intended for lightweight function execution such as real-time tools [40], [41] and data analytics [42], [43]. There exists general serverless programming frameworks [28], [29] that simplify serverless development and enables people to implement serverless linear algebra kernels. However, their generality introduces significant bloat in order to handle many different kinds of workloads. While this is good for application developers who are interested in serverless, it precludes programmers from fully optimizing their workloads for the cloud setting and therefore offers less impressive results for performance-oriented linear algebra kernels. Therefore, it is necessary to have specialized serverless linear algebra frameworks to overcome constraints imposed by serverless computing.

C. Serverless Linear Algebra

The need to express parallel numerical linear algebra in terms of directed acyclic graph (DAG) based computations is addressed in [13]. By representing subtasks and their dependencies explicitly in terms of a DAG, the lack of stateful functions in the serverless setting can be overcome. It is accomplished by introducing a domain-specific language which expresses linear algebra routines as operations on matrix tiles that can fit into the local memory of a single stateless function. The main insight is that by explicitly representing and referencing each subtask as a node in a graph whose edges represent data dependencies, parallel algorithms can be expressed in a manner agnostic to the particular machine layout and topology. However, its reliance on a general serverless framework [28] introduces significant overheads. Additionally, [13] points out that serverless linear algebra kernels still suffer from high communication overheads. Although this can be mitigated by introducing co-located cache [44] with the assumption that co-locating serverless functions are available, the main bottleneck comes from the lack of efficient collective communication primitives.

Task and resource scheduling is another issue in serverless linear algebra and has been extensively studied in [45]–[47]. Decentralized scheduling distributed across serverless functions proposed in [30] enhances data locality and resource elasticity. Data locality is also achieved by function co-locations in [44]. On the other hand, due to the predetermined time limit enforced by cloud providers on individual FaaS function invocations, serverless linear algebra frameworks need to decompose tasks into subtasks. This is mitigated in [13] by the fact that they only address dense matrix operations, leading to much more predictable subtask running times.

The reliance on secondary cloud storage due to the nature of stateless execution is also addressed by studying networking and communications in serverless systems. Solutions include intermediate functions or coordinators [48], [49] , specialized storage or query systems optimized for cloud and serverless computing [50], [51], and direct communication over TCP/IP [15], [52], [53].

Overall, numerous studies have been conducted to migrate distributed linear algebra kernels into serverless systems. Although thousands lines of code [13], [30] have been written to overcome issues mentioned above, the main bottleneck on

communication overheads is still a matter. Also, as we can observe that migrating well-studied distributed linear algebra kernels requires more efforts than the original implementations to achieve comparable performance, whether there is a need to redesign distributed algorithms to align with characteristics in serverless systems is a worth investigating problem.

D. Other Serverless Applications

Other than real-time tools, data analytics and linear algebra kernels mentioned above, there are other applications that can make use of serverless computing [54]. Scientific computations such as bioinformatics benefit from serverless computing due to their coarse-grained communication [55], [56] and on-demand resource requirements [57]. Media related applications such as video processing [58] and livestreaming [59] exploits serverless computing easily due to its inherent time-based intra-parallelism. Also, Internet of Things applications that are designed based on cloud computing can easily adopt the serverless computing paradigm [60], [61]. Moreover, there are experimental studies for the booming machine learning workflow on serverless systems [62], [63].

IV. Randomized SVD for Serverless Systems

A. Implementation Overview

Recall that the central challenge of serverless randomized SVD is how to reduce communication overheads on top of the fact that the distributed design is already optimized for minimizing communication cost. To overcome this obstacle, we propose to mitigate communication overheads from two perspectives: using a loosely coupled SVD algorithm and using FMI as an efficient serverless message passing interface.

Loosely coupled SVD is a solution to the lack of efficient collective communication primitives on serverless systems, as it only requires point-to-point communication when merging results in the binary tree. However, the reliance on secondary cloud storage for point-to-point communication still constitutes a large portion of communication overheads, and that's the reason why we add FMI on top of point-to-point communication to make serverless randomized SVD faster.

Figure 6 shows how FMI is added on top of loosely coupled SVD. When two processors want to merge their SVDs, both of them connect to a hole punching relay server so that the relay server can create an address mapping table. Then, both processors get the other's address from the relay server once the mapping is established. Direct communication comes afterward where one processor sends its SVD to the other so that two SVDs can be merged in a single processor. Note that the hole punching relay server is needed in order to circumvent the issue in NAT gateways. We have implemented loosely coupled SVD using MPI for HPC clusters and using FMI for serverless systems in order to conduct our experimental study.

B. Metrics of Success

On top of our distributed and serverless randomized SVD implementations, we are interested in the following questions:

- Does our serverless implementation have comparable performance as our distributed implementation? How many lines of code need to be changed to convert our distributed implementation to our serverless one? If two implementations have comparable performance and only a few lines of code need to be modified, it indicates that redesigning algorithms might be more beneficial than developing thousands lines of general serverless linear algebra frameworks.
- How long do we need to wait for serverless systems and HPC clusters to return results? One notorious issue in HPC is waiting in the queue for several hours or even days. A shorter response time in serverless systems has the potential to close the accessibility gap.
- How much CPU time is used in both implementations? Underutilization is a common problem in HPC clusters since a fixed amount of resource must be launched through the whole job session while serverless computing does not have this limitation hence being more cost-efficient.
- How much error is introduced when scaling up the number of processors? Recall that loosely coupled SVD sacrifices its accuracy with a factor related to the amount of parallelism. To make our implementation usable in the downstream PCA application, the error needs to be considerably small.

V. Evaluation

Generating Data Sets. There are two different categories of evaluations that require different setups given the limitations of the serverless setting. In the MPI setting, reading in an input matrix file from disk can be parallelized with MPI-IO functions. Each MPI task can start by reading the file from an evenly spaced chunk, ordered by processor rank. Using a prefix scan, the file can be quickly divided into regions corresponding to column-partitioned submatrices. In the serverless setting, the preferable method would be to use a single serverless function to pre-process the matrix file and find the coordinates for each chunk. These can then be written to cloud storage in a text file and upon invocation each serverless function can quickly look up which section of the file corresponds to the submatrix that it is responsible for.

In order to test the run-time of our prototype implementations, we focused on just timing the computation phase, since the pre-processing step just mentioned has comparably negligible run-time. Therefore, to test timing for both the MPI and AWS Lambda implementations, the input matrices were generated at runtime in parallel. Each processor/Lambda generated its submatrix by sampling uniform and independent random 64-bit floating point numbers from the interval $[0, 1]$.

We tested timing using square matrices of dimensions 2048 (2K) and 4096 (4K). We verified that our timing tests were not returning junk values by examining the leading reported singular values. For a uniformly random matrix $X \in \mathbb{R}^{2^k \times 2^k}$, the expected largest singular value is $\sigma_1 = 2^{k-1}$, and the second largest singular value is $\sigma_2 = \sqrt{2^{k-1}}/2$.

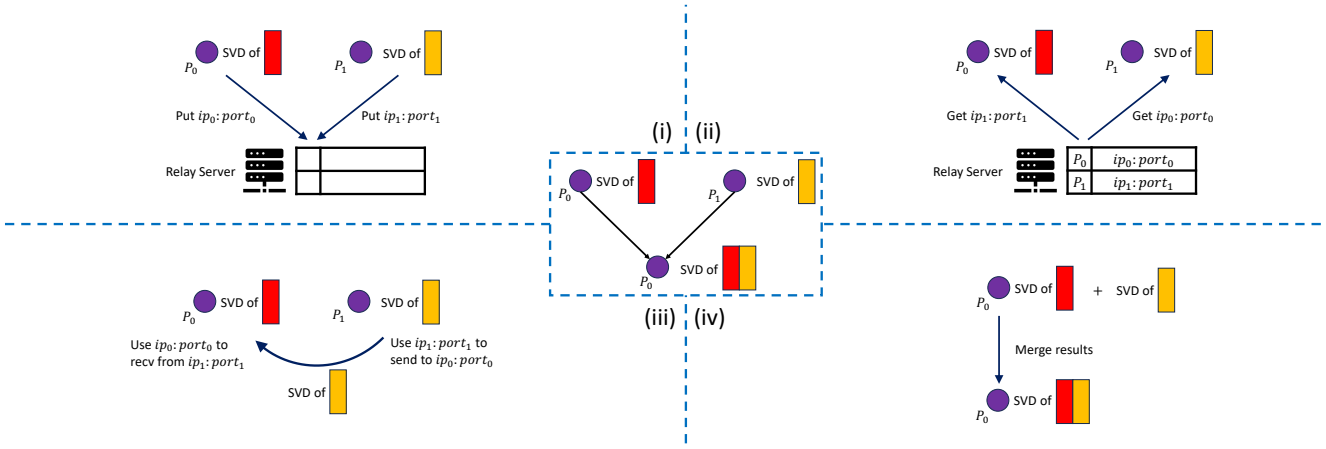

Fig. 6. Illustration of integrating loosely coupled SVD with FMI. (i) When processor 0 and processor 1 have their SVDs ready, they connect to a hole punching relay server to circumvent NAT issue. (ii) Once the address mapping table is established, both processors get the other processor's address. (iii) Processor 1 then sends its SVD directly to processor 0 via TCP. (iv) Processor 0 then follows the loosely coupled SVD algorithm to merge their SVDs.

For error analysis, it is preferable to generate matrices with known singular values beforehand. Furthermore, because our implementation approximates only a truncated SVD, and because most applications of PCA are only interested in a handful of the most important principal components, we generated matrices whose leading singular values are more significant than the rest. The largest singular values were all set to 100, and the following singular values $\sigma_2, \ldots, \sigma_r$ (where we assume r is the dimension of a square matrix) are a decreasing geometric sequence $\sigma_i = 2^{-i+1}\sigma_1$, $i = 2, \ldots, r$.

Evaluation Platforms. To test the MPI implementation of our program we used CPU nodes on the NERSC Perlmutter supercomputer, an HPE Cray EX system. The serverless implementation was run using AWS Lambda. We were granted a concurrency limit of 256 Lambda functions. We therefore tested both implementations ranging from 2 to 256 parallel tasks. Because the SVD algorithm requires at least one communication phase, we cannot run the algorithm with a single task. All timing tests were run with three trials to account for possible variance in the system load. The code for our implementations can be found on github at https://github.com/gabe-raulet/lambda_svd.git.

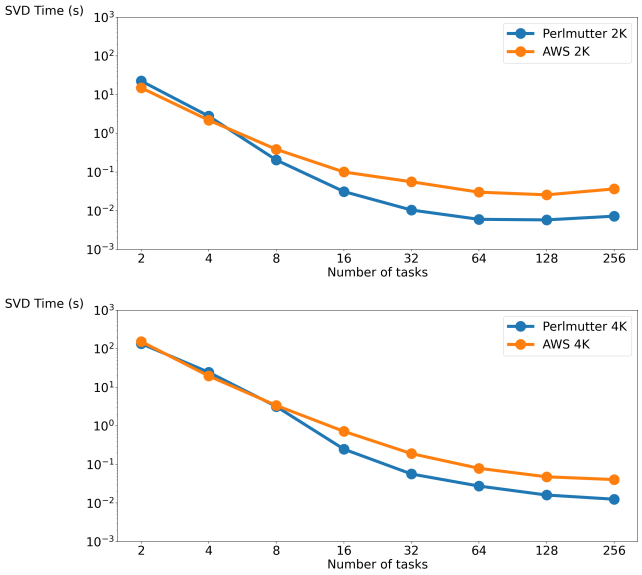

Fig. 7. SVD Time is measured from the moment execution of the distributed/serverless SVD algorithm starts until all tasks are complete. 2048×2048 matrices are measured in the top figure and 4096×4096 matrices are measured in the bottom figure.

A. Algorithm Performance

To compare the general performance of both implementations, we measured the total runtime span for the MPI implementation and the time to completion of the AWS Lambda implementation. Randomly generated square matrices with dimension 2048 and 4096 were tested. Our results demonstrate that performance on each system is comparable. For both systems, the SVD time starts to increase after too many tasks are added. Too many tasks can cause contention over resources. The comparable performance of both systems is noteworthy because the serverless implementation required changing only 26 lines of code from the MPI implementation, which has about 450 lines of code (excluding utility functions).

B. Waiting Time Comparison

In addition to the accessibility concern we attempt to address with this paper, another advantage of using a serverless cloud provider like AWS Lambda is that there is negligible wait times associated with function invocation. A common oversight in HPC research is that most work is focused exclusively on runtime performance, which neglects the long wait-queues generally associated with a highly demanded supercomputer. It is common on the Perlmutter system to wait days for a large job submission (e.g. hundreds or thousands of compute nodes) to be removed from the job queue. While we did not run our implementation on such a large number

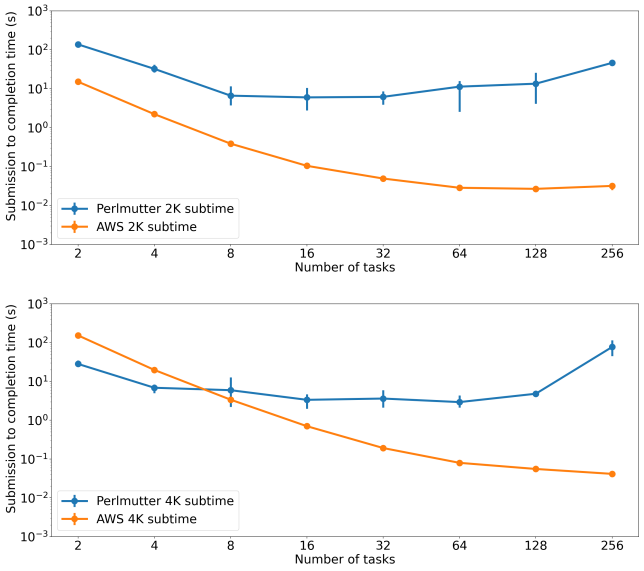

Fig. 8. Submission to completion time is measured from the time of function invocation (for AWS Lambda) and job submission (for Perlmutter). The bars behind each point span from the slowest to the fastest of three trials with the data point in the middle being the mean. 2048×2048 matrices are measured in the top figure and 4096×4096 matrices are measured in the bottom figure.

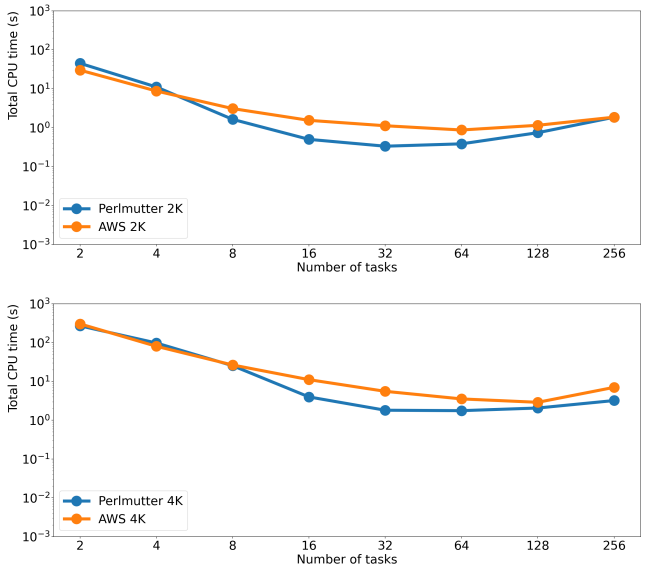

Fig. 9. CPU time measures the total number of processor-seconds starting from the moment execution begins. 2048×2048 matrices are measured in the top figure and 4096×4096 matrices are measured in the bottom figure.

of nodes, this problem is still noticeable for much smaller workloads. Figure 8 demonstrates the increased wait times associated with supercomputer resources do not affect the serverless setting. We measure submission to completion time for the Perlmutter implementation starting from the moment the job is submitted to the Slurm scheduler. In our experiments, the amount of time between submitting the requested invocations to their executing starting in AWS Lambda was too small to show up in our plots. We therefore used the same time measurement scheme as in Figure 7 for AWS Lambda. For both AWS and Perlmutter, we ran three trials for each experiment. This is represented in 8 using "error" bars spanning from the fastest trial to the slowest. The plotted points correspond to the average over the three trials. We note here that bars for the AWS experiments are so hard to see because the variance over all three trials was very low.

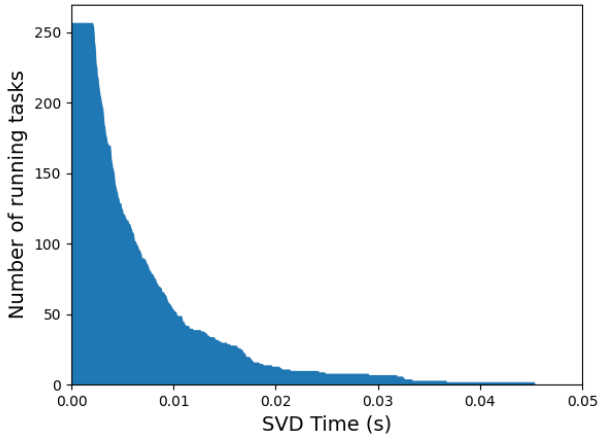

Fig. 10. Number of running tasks is the number of active Lambda functions over the course of execution. Ran using a total of 256 concurrent Lambda functions on a 2048×2048 matrix.

C. CPU Time Comparison

To measure possible under-utilization of resources on supercomputers, we measured total processor time spent for both approaches. One issue associated with MPI is that it is exceedingly difficult to throttle down the amount of resources being used during runtime. A selling point of the serverless approach is its ability to scale automatically to the runtime requirements of the application. We hypothesized at the outset of our project that a positive aspect of our approach would be the more efficient utilization of resources, because Lambda functions can be immediately released when they are no longer needed. MPI tasks must wait and idle, using up system resources, until the end of the executable runtime. We measured Total CPU time by taking the individual times associated with each MPI task and Lambda function and summing them. To help visualize the runtime characteristics of the AWS Lambda implementation, we plot in Figure 9 the number of of actively running tasks over time. The exponential decay is explained by the binary topology of the execution flow, where after each communication phase half of the currently active Lambdas return early.

D. Error Analysis

In order to measure the error incurred due to the use of an approximate algorithm, we compute the relative errors of a selection of the top 10 principal components of a PCA. Recall that a PCA can be obtained from the SVD by a

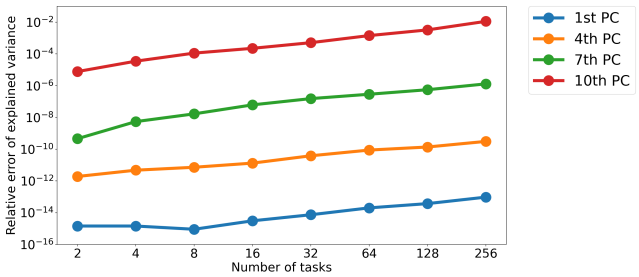

Fig. 11. Relative errors of explained variance (Relerr of expvar) for principal components (PC) in PCA v.s. number of processors used on a 4096×4096 matrix.

simple transformation. The error of our implementation can be seen to increase as the number of tasks goes up. This is explained by the nature of the algorithm we use. The seeding phase of algorithm runs truncated SVD on its original column-partitioned submatrix. When more tasks are employed, this means the seeding phase computes a less accurate initial estimate because there is less information.

VI. Discussions and Future Directions

A. Various Optimizations

The current implementation of our algorithm follows a naive scheduling strategy, where each Lambda function's responsibility is determined deterministically by the topology of a binary tree. A more efficient utilization of resources would be to introduce a runtime DAG task scheduler. A DAG scheduler represents parallel programs in terms of their dependencies. Whenever a task is completed, its successors are looked up in the DAG and notified that one of their dependencies is ready to communicate its results. This means that faster tasks don't waste CPU time as in the deterministic approach.

B. Serverless Sparse Linear Algebra

Although irregularity and load imbalance in sparse linear algebra make it align well with serverless computing at the first glance, there are still several obstacles that existing serverless linear algebra frameworks need to overcome.

One issue is the predetermined time limit enforced by cloud providers on individual FaaS function invocations. Existing serverless linear algebra frameworks only deal with dense matrix operations, leading to much more predictable subtask running times so that those frameworks can divide the tasks until they fit into the time limit. However, in the sparse linear algebra domain, even if the sizes of inputs are known beforehand, it is not easy to predict the running time as accessing data in an irregular pattern varies the running time a lot. Logging and checkpointing in serverless systems [64], [65] have been studied, but their high overheads make them incompatible with performance-oriented sparse linear algebra kernels.

Another issue is still the communication overheads. In order to fit each FaaS function invocation into the predetermined time limit, sparse linear algebra kernels need to be partitioned aggressively to address their unpredictable running times. However, over-partitioning introduces more small messages that need to be communicated between serverless functions, which aggravates the already congested communication traffics. Although there are studies [66], [67] trying to address this by regularizing irregular sparse communications, they are not panaceas since they focus on specific sparsity patterns and network topology.

From another point of view, real-world large sparse graphs can be represented as sparse matrices hence serverless graph processing is also worth investigating in order to understand serverless sparse linear algebra. Serverless graph processing has been studied in [68]. Although fine-grained elasticity is achieved, the paper indicates that their implementation is network-bound and not suitable for communication-intensive graph algorithms.

C. Serverless Machine Learning

Machine learning applications, including PCA discussed in this paper, rely heavily on linear algebra kernels. Although there exists serverless machine learning services such as [69], they still share the same difficulties as serverless linear algebra. For example, operators in deep learning models are highly coupled hence deploying them on serverless systems suffer from high communication overheads [63]. Consequently, designing loosely coupled deep learning models, an approach similar as the loosely coupled SVD, has been explored [62]. Nevertheless, there are still other difficulties such as the lack of serverless GPU systems need to be solved.

Large language models are also studied in serverless systems [70]. Downstream applications of large language models including generative artificial intelligence fit well into the serverless paradigm because requests are triggered on demand and inference needs to be done in real time. However, fitting a high-quality large language model into serverless systems can be hard since serverless computing is designed for lightweight tasks. Also, compared to a serverful setup where models can be pre-loaded, the stateless nature of serverless computing makes it suffer from high startup cost for loading models.

Overall, linear algebra kernels are not the only bottleneck in serverless machine learning. The highly coupled model designs, the reliance on GPUs, and the high startup cost for loading models are issues that need to be addressed in order to make serverless machine learning a maturer field.

D. Serverless GPU systems

Most of the well-developed serverless services at present only include CPUs but not GPUs. Although serverless GPU systems have been proposed [71], it has not yet been provided by mainstream cloud providers due to the imperfections stated below.

One issue is, again, communication overheads. Distributed GPU linear algebra kernels generally assumed that there is a fast network interconnect between all compute nodes such as NVLink [72] to transfer data between GPUs. However, such

an interconnect does not exist in serverless systems hence it is a worth exploring future direction.

Another issue is the startup time. Migrating data between CPUs and GPUs, kernel launching overheads, and synchronization overheads constitute the startup time, which is considerably longer than using CPUs alone. Only if there are enough jobs to do in the current serverless function after startup can this time be amortized. However, this is hindered by the predetermined time limit set by cloud providers, hence a new billing model must be considered.

Another direction is to use the AWS Spot Instances [73], where spare compute capacity is provided in a lower price. However, high-end GPUs are usually unavailable, making specialized GPU linear algebra kernels relying on special components such as Tensor Cores [74] unable to run.

As more and more linear algebra kernels are specialized for GPUs, the lack of mature serverless GPU systems is widening the accessibility gap. Overcoming the scarcity of GPUs is hard, but we believe that it is a necessary step in order to make everyone benefit from GPU based linear algebra kernels in their own workflow.

VII. Conclusion

In this paper, to alleviate high communication overheads found in existing serverless linear algebra frameworks, we integrate a FaaS message interface into a loosely coupled randomized SVD algorithm and derive a serverless randomized SVD implementation. We then conduct an experimental study on our randomized SVD implementation for serverless systems. Our evaluations demonstrate that high-performance linear algebra kernels can be executed in the serverless setting with comparable performance and significantly better accessibility when compared to supercomputers. Furthermore, the long job queuing times associated with high-performance clusters are completely bypassed by our approach.

Although we only focus on randomized SVD in this paper, we believe that our analysis can be extended to other linear algebra kernels suffering from high communication overheads. Specifically, to improve performance of serverless linear algebra kernels on top of the fact that distributed ones are already optimized for minimizing communication cost, either a more efficient serverless message interface is required or a redesign of algorithms aligning with constraints imposed by serverless computing is needed.

Acknowledgment

We thank Prof. John Kubiatowicz for his advice throughout this research. We also thank Berkeley Lab for providing access to the Perlmutter system.

References

[1] H. Hotelling, "Analysis of a complex of statistical variables into principal components," *J. Educ. Psychol.*, vol. 24, no. 6, pp. 417–441, 1933.

[2] K. Pearson, "On lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, vol. 2, no. 11, pp. 559–572, 1901.

[3] A. Amadei, A. B. Linssen, and H. J. Berendsen, "Essential dynamics of proteins," *Proteins*, vol. 17, no. 4, pp. 412–425, 1993.

[4] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, "Pcanet: A simple deep learning baseline for image classification?" *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 5017–5032, 2015.

[5] Q. Du and J. E. Fowler, "Hyperspectral image compression using jpeg2000 and principal component analysis," *IEEE Geoscience and Remote Sensing Letters*, vol. 4, no. 2, pp. 201–205, 2007.

[6] N. Halko, P.-G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM review*, vol. 53, no. 2, pp. 217–288, 2011.

[7] R. Murray, J. Demmel, M. W. Mahoney, N. B. Erichson, M. Melnichenko, O. A. Malik, L. Grigori, P. Luszczek, M. Dereziński, M. E. Lopes *et al.*, "Randomized numerical linear algebra: A perspective on the field with an eye to software," *arXiv preprint arXiv:2302.11474*, 2023.

[8] H. Chen, J. Zhao, Q. Luo, and Y. Hou, "Distributed randomized singular value decomposition using count sketch," in *2017 International Conference on Security, Pattern Analysis, and Cybernetics (SPAC)*, 2017, pp. 187–191.

[9] R. Maulik and G. Mengaldo, "Pyparsvd: A streaming, distributed and randomized singular-value-decomposition library," in *2021 7th International Workshop on Data Analysis and Reduction for Big Scientific Data (DRBSD-7)*, 2021, pp. 19–25.

[10] "Aws Lambda," https://aws.amazon.com/lambda/, accessed 2023-12.

[11] "Google Cloud Functions," https://cloud.google.com/functions, accessed 2023-12.

[12] "Microsoft Azure Functions," https://azure.microsoft.com/en-us/products/functions, accessed 2023-12.

[13] V. Shankar, K. Krauth, K. Vodrahalli, Q. Pu, B. Recht, I. Stoica, J. Ragan-Kelley, E. Jonas, and S. Venkataraman, "Serverless linear algebra," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, 2020, pp. 281–295.

[14] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, "Minimizing communication in numerical linear algebra," *SIAM Journal on Matrix Analysis and Applications*, vol. 32, no. 3, pp. 866–901, 2011.

[15] M. Copik, R. Böhringer, A. Calotoiu, and T. Hoefler, "Fmi: Fast and cheap message passing for serverless functions," in *Proceedings of the 37th International Conference on Supercomputing*, 2023, pp. 373–385.

[16] C. Boutsidis, D. P. Woodruff, and P. Zhong, "Optimal principal component analysis in distributed and streaming models," in *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, 2016, pp. 236–249.

[17] S. Fang, *Distributed computing of large-scale singular value decompositions*. Oxford University Research Archive, 2018.

[18] I. Jolliffe, *Principal Component Analysis*, ser. Springer Series in Statistics. Springer, 2002. [Online]. Available: https://books.google.com/books?id=_olByCrhjwIC

[19] C. Eckart and G. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, no. 3, pp. 211–218, 1936.

[20] G. Golub and W. Kahan, "Calculating the singular values and pseudo-inverse of a matrix," *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, vol. 2, no. 2, pp. 205–224, 1965.

[21] J. Demmel and W. Kahan, "Accurate singular values of bidiagonal matrices," *SIAM Journal on Scientific and Statistical Computing*, vol. 11, no. 5, pp. 873–912, 1990.

[22] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney *et al.*, *LAPACK users' guide*. SIAM, 1999.

[23] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet *et al.*, *ScaLAPACK users' guide*. SIAM, 1997.

[24] W. B. Johnson and J. Lindenstrauss, "Extensions of lipshitz mapping into hilbert space," *Conference on Modern Analysis and Probability*, vol. 26, pp. 129–138, 1984.

[25] "Amazon Web Services," https://aws.amazon.com/, accessed 2023-12.

[26] "Amazon Simple Storage Service," https://aws.amazon.com/s3/, accessed 2023-12.

[27] "Amazon Simple Queue Service," https://aws.amazon.com/sqs/, accessed 2023-12.

[28] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the cloud: Distributed computing for the 99%," in *Proceedings of the 2017 symposium on cloud computing*, 2017, pp. 445–451.

[29] W. Zhang, V. Fang, A. Panda, and S. Shenker, "Kappa: A programming framework for serverless computing," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, 2020, pp. 328–343.

[30] B. Carver, J. Zhang, A. Wang, A. Anwar, P. Wu, and Y. Cheng, "Wukong: A scalable and locality-enhanced framework for serverless parallel computing," in *Proceedings of the 11th ACM symposium on cloud computing*, 2020, pp. 1–15.

[31] P. Srisuresh and M. Holdrege, "RFC2663: IP network address translator (NAT) terminology and considerations," 1999.

[32] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-peer communication across network address translators," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ser. ATEC '05. USA: USENIX Association, 2005, p. 13.

[33] J. Demmel, M. Gu, S. Eisenstat, I. Slapničar, K. Veselić, and Z. Drmač, "Computing the singular value decomposition with high relative accuracy," *Linear Algebra and its Applications*, vol. 299, no. 1-3, pp. 21–80, 1999.

[34] Z. Drmač and K. Veselić, "New fast and accurate jacobi svd algorithm. i," *SIAM Journal on matrix analysis and applications*, vol. 29, no. 4, pp. 1322–1342, 2008.

[35] R. P. Dickinson, Jr, F. N. Fritsch, R. F. Hausman, Jr, and R. F. Sincovec, "Eispack user's guide," *U.S. Department of Energy Office of Scientific and Technical Information*, 1973. [Online]. Available: https://www.osti.gov/biblio/4426305

[36] E. Agullo, J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek, and S. Tomov, "Numerical linear algebra on emerging architectures: The plasma and magma projects," in *Journal of Physics: Conference Series*, vol. 180, no. 1. IOP Publishing, 2009, p. 012037.

[37] G. Ballard, J. Demmel, and N. Knight, "Avoiding communication in successive band reduction," *ACM Transactions on Parallel Computing (TOPC)*, vol. 1, no. 2, pp. 1–37, 2015.

[38] G. H. Golub, F. T. Luk, and M. L. Overton, "A block lanczos method for computing the singular values and corresponding singular vectors of a matrix," *ACM Transactions on Mathematical Software (TOMS)*, vol. 7, no. 2, pp. 149–169, 1981.

[39] G. H. Golub and Q. Ye, "An inverse free preconditioned krylov subspace method for symmetric generalized eigenvalue problems," *SIAM Journal on Scientific Computing*, vol. 24, no. 1, pp. 312–334, 2002.

[40] M. Yan, P. Castro, P. Cheng, and V. Ishakian, "Building a chatbot with serverless computing," in *Proceedings of the 1st International Workshop on Mashups of Things and APIs*, 2016, pp. 1–4.

[41] P. Saint-Andre, "Serverless messaging," 2018.

[42] S. Nastic, T. Rausch, O. Scekic, S. Dustdar, M. Gusev, B. Koteska, M. Kostoska, B. Jakimovski, S. Ristov, and R. Prodan, "A serverless real-time data analytics platform for edge computing," *IEEE Internet Computing*, vol. 21, no. 4, pp. 64–71, 2017.

[43] I. Müller, R. Marroquín, and G. Alonso, "Lambada: Interactive data analytics on cold data using serverless cloud infrastructure," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 115–130.

[44] K. Vodrahalli and E. Zhou, "Using software-defined caching to enable efficient communication in a serverless environment," 2018.

[45] M. Copik, M. Chrapek, A. Calotoiu, and T. Hoefler, "Software resource disaggregation for hpc with serverless computing," *ACM Student Research Competition (SRC), ACM/IEEE Supercomputing. Poster.*, 2022.

[46] B. Carver, J. Zhang, A. Wang, and Y. Cheng, "In search of a fast and efficient serverless dag engine," in *2019 IEEE/ACM Fourth International Parallel Data Systems Workshop (PDSW)*. IEEE, 2019, pp. 1–10.

[47] T. Mo and R. Li, "Iteratively solving sparse linear system based on parsec task scheduling," *The International Journal of High Performance Computing Applications*, vol. 34, no. 3, pp. 306–315, 2020.

[48] A. Bhardwaj, C. Kulkarni, and R. Stutsman, "Adaptive placement for in-memory storage functions," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 127–141.

[49] S. Fouladi, F. Romero, D. Iter, Q. Li, S. Chatterjee, C. Kozyrakis, M. Zaharia, and K. Winstein, "From laptop to lambda: Outsourcing everyday jobs to thousands of transient functional containers," in *2019 USENIX annual technical conference (USENIX ATC 19)*, 2019, pp. 475–488.

[50] M. Perron, R. Castro Fernandez, D. DeWitt, and S. Madden, "Starling: A scalable query engine on cloud functions," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 131–141.

[51] "Aws Elastic Cache," https://aws.amazon.com/elasticache/, accessed 2023-12.

[52] S. Thomas, L. Ao, G. M. Voelker, and G. Porter, "Particle: ephemeral endpoints for serverless networking," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, 2020, pp. 16–29.

[53] M. Wawrzoniak, I. Müller, R. Fraga Barcelos Paulus Bruno, and G. Alonso, "Boxer: Data analytics on network-enabled serverless platforms," in *11th Annual Conference on Innovative Data Systems Research (CIDR 2021)*, 2021.

[54] H. Shafiei, A. Khonsari, and P. Mousavi, "Serverless computing: a survey of opportunities, challenges, and applications," *ACM Computing Surveys*, vol. 54, no. 11s, pp. 1–32, 2022.

[55] L.-H. Hung, D. Kumanov, X. Niu, W. Lloyd, and K. Y. Yeung, "Rapid rna sequencing data analysis using serverless computing," *bioRxiv*, p. 576199, 2019.

[56] B. D. Lee, M. A. Timony, and P. Ruiz, "DNA visualization. org: a serverless web tool for DNA sequence visualization," *Nucleic acids research*, vol. 47, no. W1, pp. W20–W25, 2019.

[57] D. Kumanov, L.-H. Hung, W. Lloyd, and K. Y. Yeung, "Serverless computing provides on-demand high performance computing for biomedical research," *arXiv preprint arXiv:1807.11659*, 2018.

[58] L. Ao, L. Izhikevich, G. M. Voelker, and G. Porter, "Sprocket: A serverless video processing framework," in *Proceedings of the ACM Symposium on Cloud Computing*, 2018, pp. 263–274.

[59] K. Konstantoudakis, D. Breitgand, A. Doumanoglou, N. Zioulis, A. Weit, K. Christaki, P. Drakoulis, E. Christakis, D. Zarpalas, and P. Daras, "Serverless streaming for emerging media: towards 5G network-driven cost optimization: A real-time adaptive streaming faas service for small-session-oriented immersive media," *Multimedia Tools and Applications*, pp. 1–40, 2022.

[60] I. Wang, E. Liri, and K. Ramakrishnan, "Supporting iot applications with serverless edge clouds," in *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*. IEEE, 2020, pp. 1–4.

[61] P. Benedetti, M. Femminella, G. Reali, and K. Steenhaut, "Experimental analysis of the application of serverless computing to iot platforms," *Sensors*, vol. 21, no. 3, p. 928, 2021.

[62] V. Ishakian, V. Muthusamy, and A. Slominski, "Serving deep learning models in a serverless platform," in *2018 IEEE International conference on cloud engineering (IC2E)*. IEEE, 2018, pp. 257–262.

[63] J. Carreira, P. Fonseca, A. Tumanov, A. Zhang, and R. Katz, "A case for serverless machine learning," in *Workshop on Systems for ML and Open Source Software at NeurIPS*, vol. 2018, 2018, pp. 2–8.

[64] Z. Jia and E. Witchel, "Boki: Stateful serverless computing with shared logs," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 691–707.

[65] H. Zhang, A. Cardoza, P. B. Chen, S. Angel, and V. Liu, "Fault-tolerant and transactional stateful serverless workflows," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 1187–1204.

[66] O. Selvitopi and C. Aykanat, "Regularizing irregularly sparse point-to-point communications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–14.

[67] S. H. Mirsadeghi, J. L. Traff, P. Balaji, and A. Afsahi, "Exploiting common neighborhoods to optimize mpi neighborhood collectives," in *2017 IEEE 24th international conference on high performance computing (HiPC)*. IEEE, 2017, pp. 348–357.

[68] L. Toader, A. Uta, A. Musaafir, and A. Iosup, "Graphless: Toward serverless graph processing," in *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*. IEEE, 2019, pp. 66–73.

[69] "Amazon SageMaker," https://aws.amazon.com/sagemaker/, accessed 2023-12.

[70] "Amazon Bedrock," https://aws.amazon.com/bedrock/, accessed 2023-12.

[71] J. Kim, T. J. Jun, D. Kang, D. Kim, and D. Kim, "Gpu enabled serverless computing framework," in *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. IEEE, 2018, pp. 533–540.

[72] "NVIDIA NVLink and NVSwitch," https://www.nvidia.com/en-us/data-center/nvlink/, accessed 2023-12.

[73] "Amazon EC2 Spot Instances," https://aws.amazon.com/ec2/spot/, accessed 2023-12.

[74] "NVIDIA Tensor Cores,"
https://www.nvidia.com/en-us/data-center/tensor-cores/, accessed 2023-12.