OverSketch: Approximate Matrix Multiplication for the Cloud

Vipul Gupta*, Shusen Wang†, Thomas Courtade* and Kannan Ramchandran*
*Department of EECS, UC Berkeley
†Department of CS, Stevens Institute of Technology
Email: {vipul_gupta, courtade, kannanr}@eecs.berkeley.edu, shusen.wang@stevens.edu

***Abstract*—We propose OverSketch, an approximate algorithm for distributed matrix multiplication in serverless computing. OverSketch leverages ideas from matrix sketching and high-performance computing to enable cost-efficient multiplication that is resilient to faults and straggling nodes pervasive in low-cost serverless architectures. We establish statistical guarantees on the accuracy of OverSketch and empirically validate our results by solving a large-scale linear program using interior-point methods and demonstrate a 34% reduction in compute time on AWS Lambda.**

***Keywords*-serverless computing, straggler mitigation, sketched matrix multiplication**

I. Introduction

Matrix multiplication is a frequent computational bottleneck in fields like scientific computing, machine learning, graph processing, etc. In many applications, such matrices are very large, with dimensions easily scaling up to millions. Consequently, the last three decades have witnessed the development of many algorithms for parallel matrix multiplication for High Performance Computing (HPC). During the same period, technological trends like Moore's law made arithmetic operations faster and, as a result, the bottleneck for parallel computation shifted from computation to communication. Today, the cost of moving data between nodes exceeds the cost of arithmetic operations by orders of magnitude. This gap is increasing exponentially with time and has led to the popularity of communication-avoiding algorithms for parallel computation [1], [2].

In the last few years, there has been a paradigm shift from HPC towards distributed computing on the cloud due to extensive and inexpensive commercial offerings. In spite of developments in recent years, server-based cloud computing is inaccessible to a large number of users due to complex cluster management and a myriad of configuration tools. Serverless computing[1] has recently begun to fill this void by abstracting away the need for maintaining servers and thus removing the need for complicated cluster management while providing greater elasticity and easy scalability [3]–[5]. Some examples are Amazon Web Services (AWS) based Lambda and Google Cloud Functions. Large-scale matrix multiplication, being embarrassingly parallel and frequently encountered, is a natural fit for serverless computing.

This work was supported by NSF Grants CCF-1703678, CCF-1704967 and CCF-0939370 (Center for Science of Information)

[1]The term 'serverless' is a misnomer, servers are still used for computation but their maintenance and provisioning is hidden from the user.

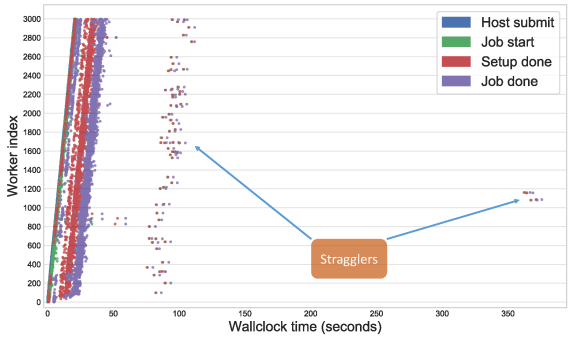


Figure 1: Job times for 3000 AWS Lambda nodes where the median job time is around 40 seconds, and around 5% of the nodes take 100 seconds, and two nodes take as much as 375 seconds to complete the same job.

Existing distributed algorithms for HPC cannot, in general, be extended to serverless computing due to the following crucial differences between the two architectures:

- Workers in the serverless setting, unlike cluster nodes, do not communicate amongst themselves. They read/write data directly from/to a single data storage entity (for example, cloud storage like AWS S3) and the user is only allowed to submit prespecified jobs and does not have any control over the management of workers [4], [5].
- Distributed computation in HPC/server-based systems is generally limited by the number of workers at disposal. However, in serverless systems, the number of inexpensive workers can easily be scaled into the thousands, but these low-commodity nodes are generally limited by the amount of memory and lifespan available.
- Unlike HPC, nodes in the cloud-based systems suffer degradation due to system noise [6], [7]. This causes variability in job times, resulting in a subset of slower nodes called *stragglers*. Time statistics for worker job times are plotted in Figure 1 for AWS Lambda. Notably, there are a few workers ($\sim 5\%$) that take much longer than the median job time, thus decreasing the overall computational efficiency of the system. Distributed algorithms robust to such unreliable nodes are desirable in cloud computing.

A. Main Contributions

This paper bridges the gap between communication-efficient algorithms for distributed computation and existing

methods for straggler-resiliency. To this end, we first analyze the monetary cost of distributed matrix multiplication for serverless computing for two different schemes of partitioning and distributing the data. Specifically, we show that row-column partitioning of input matrices requires asymptotically more communication than blocked partitioning for distributed matrix multiplication, similar to the optimal communication-avoiding algorithms in the HPC literature.

In applications like machine learning, where the data itself is noisy, solution accuracy is often traded for computational efficiency. Motivated by this, we propose OverSketch, a sketching scheme to perform blocked *approximate* matrix multiplication and prove statistical guarantees on the accuracy of the result. OverSketch has threefold advantages:

1) Reduced computational complexity by significantly decreasing the dimension of input matrices using sketching,
2) Resiliency against stragglers and faults in serverless computing by over-provisioning the sketch dimension,
3) Communication efficiency for distributed multiplication due to the blocked partition of input matrices.

Sketching for OverSketch requires linear time that is embarrassingly parallel. Through experiments on AWS Lambda, we show that small redundancy ($\approx 5\%$) is enough to tackle stragglers using OverSketch. Furthermore, we use OverSketch to calculate the Hessian distributedly while solving a large linear program using interior point methods and demonstrate a 34% reduction in total compute time on AWS Lambda.

B. Related Work

Recently, approaches based on coding theory have been developed which cleverly introduce redundancy into the computation to deal with stragglers [8]–[13]. Many of these proposed schemes have been dedicated to distributed matrix multiplication [9]–[12]. In [9], the authors develop a coding scheme for matrix multiplication that uses Maximum Distance Separable (MDS) codes to code $\mathbf{A}$ in a column-wise fashion and $\mathbf{B}$ in a row-wise fashion, so that the resultant is a product-code of $\mathbf{C}$, where $\mathbf{C} = \mathbf{A} \times \mathbf{B}$. An illustration is shown in Figure 2. Authors in [10] generalize the results in [9] to a d-dimensional product code with only one parity in each dimension. In [11], the authors develop polynomial codes for matrix multiplication, which is an improvement over [9] in terms of recovery threshold, that is, minimum number of workers required to recover the product $\mathbf{C}$.

The commonality in these and other similar results is that they divide the input matrices into row and column blocks, where each worker multiplies a row block (or some combination of row blocks) of $\mathbf{A}$ and a column block (or some combination of column blocks) of $\mathbf{B}$. These methods provide straggler resiliency but are not cost-efficient as they require asymptotically more communication than blocked partitioning of data, as discussed in detail in the next section. Another disadvantage of such coding-based methods is that there are separate encoding and decoding phases that require additional communication and potentially large computational burden at the master node, which may make the algorithm infeasible in some distributed computing environments.

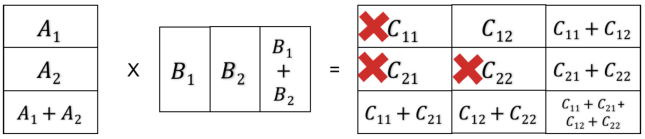
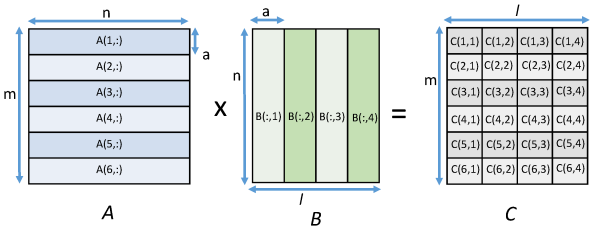
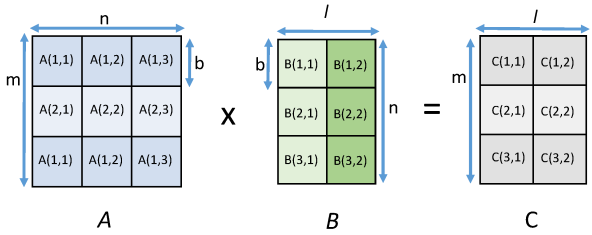


Figure 2: Matrix $\mathbf{A}$ is divided into 2 row chunks $\mathbf{A}_1$ and $\mathbf{A}_2$, while $\mathbf{B}$ is divided into two column chunks $\mathbf{B}_1$ and $\mathbf{B}_2$. During the encoding process, redundant chunks $\mathbf{A}_1 + \mathbf{A}_2$ and $\mathbf{B}_1 + \mathbf{B}_2$ are created. To compute $\mathbf{C}$, 9 workers store each possible combination of a chunk of $\mathbf{A}$ and $\mathbf{B}$ and multiply them. During the decoding phase, the master can recover the affected data (C_{11}, C_{12} and C_{22} in this case) using the redundant chunks.



(a) Distributed naive matrix multiplication, where each worker multiplies a row-block of $\mathbf{A}$ of size $a \times n$ and a column block of $\mathbf{B}$ of size $n \times a$ to get an $a \times a$ block of $\mathbf{C}$.



(b) Distributed blocked matrix multiplication, where each worker multiplies a sub-block of $\mathbf{A}$ of size $b \times b$ and a sub-block of $\mathbf{B}$ of size $b \times b$.

Figure 3: An illustration of two algorithms for distributed matrix multiplication.

II. Preliminaries

There are two common schemes for distributed multiplication of two matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times l}$, as illustrated in Figures 3a and 3b. We refer to these schemes as *naive* and *blocked* matrix multiplication, respectively. For detailed steps of naive and blocked matrix multiplication in the serverless setting, we refer the readers to algorithms 1 and 2 in [14], respectively. During naive matrix multiplication, each worker receives and multiplies an $a \times n$ row-block of $\mathbf{A}$ and $n \times a$ column-block of $\mathbf{B}$ to compute an $a \times a$ block of $\mathbf{C}$. Blocked matrix multiplication consists of two phases. During the computation phase, each worker gets two $b \times b$ blocks, one each from $\mathbf{A}$ and $\mathbf{B}$, which are then multiplied by the workers. In the reduction phase, to compute a $b \times b$ block of $\mathbf{C}$, one worker gathers results of all the n/b workers from the the cloud storage corresponding to one row-block of $\mathbf{A}$ and one column-block of $\mathbf{B}$ and adds them. For example, in Figure 3b, to get $\mathbf{C}(1,1)$, results from 3

workers who compute $\mathbf{A}(1,1) \times \mathbf{B}(1,1)$, $\mathbf{A}(1,2) \times \mathbf{B}(2,1)$ and $\mathbf{A}(1,3) \times \mathbf{B}(3,1)$ are added.

It is accepted in High Performance Computing (HPC) that blocked partitioning of input matrices takes less time than naive matrix multiplication [1], [2], [15]. For example, in [2], the authors propose 2.5D matrix multiplication, an optimal communication avoiding algorithm for matrix multiplication in HPC/server-based computing, that divides input matrices into blocks and stores redundant copies of them across processors to reduce bandwidth and latency costs. However, perhaps due to lack of a proper analysis for cloud-based distributed computing, existing algorithms for straggler mitigation in the cloud do naive matrix multiplication [9]–[11]. Next, we develop a new cost model for the serverless computing architecture and aim at bridging the gap between cost analysis and straggler mitigation for distributed computation in the serverless setting.

III. Cost Analysis: Naive and Blocked multiplication

There are communication and computation costs associated with any distributed algorithm. Communication costs themselves are of two types: latency and bandwidth. For example, sending n words requires packing them into contiguous memory and transmitting them as a message. The latency cost α is the fixed overhead time spent in packing and transmitting a message over the network. Thus, to send Q messages, the total latency cost is αQ. Similarly, to transmit K words, a bandwidth cost proportional to K, given by βK, is associated. Letting γ denote the time to perform one floating point operation (FLOP), the total computing cost is γF, where F is the total number of FLOPs at the node. Hence, the total time pertaining to one node that sends M messages, K words and performs F FLOPs is

$$T_{\text{worker}} = \alpha Q + \beta K + \gamma F,$$

where $\alpha \gg \beta \gg \gamma$. The (α, β, γ) model defined above has been well-studied and is used extensively in the HPC literature [1], [2]. It is ideally suited for serverless computing, where network topology does not affect the latency costs as each worker reads/writes directly from/to the cloud storage and no multicast gains are possible.

However, our analysis for costs incurred during distributed matrix multiplication differs from previous works in three principle ways. 1) Workers in serverless architecture cannot communicate amongst themselves, and hence, our algorithm for blocked multiplication is very different from optimal communication avoiding algorithm for HPC that involves message passing between workers [2]. 2) The number of workers in HPC analyses is generally fixed, whereas the number of workers in serverless setting is quite flexible, easily scaling into the thousands, and the limiting factor is memory/bandwidth available at each node. 3) Computation on the inexpensive cloud is more motivated by savings in expenditure than the time required to run the algorithm. We define our cost function below.

Table I: Costs comparison for naive and blocked matrix multiplication in the serverless setting, where $\delta < 2$.

Cost type	Naive multiply	Blocked Multiply	Ratio: naive/blocked
Latency	$O(mln^{2(1-\delta)})$	$O(mln^{1-3\delta/2})$	$O(n^{1-\delta/2})$
Bandwidth	$O(mln^{2-\delta})$	$O(mln^{1-\delta/2})$	$O(n^{1-\delta/2})$
Computation	$O(mln)$	$O(mln)$	1

If there are W workers, each doing an equal amount of work, the total amount of money spent in running the distributed algorithm on the cloud is proportional to

$$C_{\text{total}} = W \times T_{\text{worker}} = W(\alpha Q + \beta K + \gamma F). \quad (1)$$

Eq. (1) does not take into account the straggling costs as they increase the total cost by a constant factor (by re-running the jobs that are straggling) and hence does not affect our asymptotic analysis.

Inexpensive nodes in serverless computing are generally constrained by the amount of memory or communication bandwidth available. For example, AWS Lambda nodes have a maximum allocated memory of 3008 MB[2], a fraction of the memory available in today's smartphones. Let the memory available at each node be limited to M words. That is, the communication bandwidth available at each worker is limited to M words, and this is the main bottleneck of the distributed system. We would like to multiply two large matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times l}$ in parallel, and let $M = O(n^{\delta})$. For all practical cases in consideration, $\delta < 2$.

Proposition 3.1: For the cost model defined in Eq. (1), communication (i.e., latency and bandwidth) costs for blocked multiplication outperform naive multiplication by a factor of $O(n^{1-\delta/2})$, where the individual costs are listed in Table I.

We refer the readers to Appendix A of [14] for proof. The rightmost column in Table I lists the ratio of communication costs for naive and blocked matrix multiplication. We note that the latter significantly outperforms the former, with communication costs being asymptotically worse for naive multiplication. An intuition behind why this happens is that each worker in distributed blocked multiplication does more work than in distributed naive multiplication for the same amount of received data. For example, to multiply two square matrices of dimension n, where memory at each worker limited by $M = 2n$, $a = 1$ for naive multiplication and $b = \sqrt{n}$ for blocked multiplication. We note that the amount of work done by each worker in naive and blocked multiplication is $O(n)$ and $O(n^{3/2})$, respectively. Since the total amount of work is constant and equal to $O(n^3)$, blocked matrix multiplication ends up communicating less during the overall execution of the algorithm as it requires fewer workers. Note

[2]AWS Lambda limits are available at (may change over time) https://docs.aws.amazon.com/lambda/latest/dg/limits.html

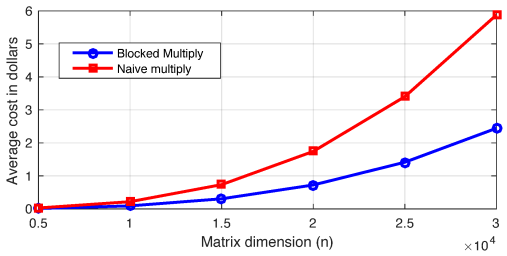


Figure 4: Comparison of AWS Lambda costs for multiplying two $n \times n$ matrices, where each worker is limited by 3008 MB of memory and price per running worker per 100 milliseconds is \$0.000004897.

that naive multiplication takes less time to complete as each worker does asymptotically less work, however, the number of workers required is asymptotically more, which is not an efficient utilization of resources and increases the expenditure significantly.

Figure 4 supports the above analysis where we plot the cost in dollars of multiplying two square matrices in AWS Lambda, where each node's memory is limited by 3008 MB and price per worker per 100 millisecond is \$0.000004897. However, as discussed earlier, existing schemes for straggler-resiliency in distributed matrix multiplication consider naive multiplication which is impractical from a user's point of view. In the next section, we propose OverSketch, a scheme to mitigate the detrimental effects of stragglers for blocked matrix multiplication.

IV. OverSketch: Straggler-resilient Blocked Matrix Multiplication using Sketching

Many of the recent advances in algorithms for numerical linear algebra have come from the technique of linear sketching, in which a given matrix is compressed by multiplying it with a random matrix of appropriate dimension. The resulting product can then act as a proxy for the original matrix in expensive computations such as matrix multiplication, least-squares regression, low-rank approximation, etc. [16], [17]. For example, computing the product of $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times l}$ takes $O(mnl)$ time. However, if we use $\mathbf{S} \in \mathbb{R}^{n\times d}$ to compute the sketches, say $\tilde{\mathbf{A}} = \mathbf{AS} \in \mathbb{R}^{m\times d}$ and $\tilde{\mathbf{B}} = \mathbf{S}^T\mathbf{B} \in \mathbb{R}^{d\times l}$, where $d \ll n$ is the sketch dimension, we can reduce the computation time to $O(mdl)$ by computing an approximate product $\mathbf{ASS}^T\mathbf{B}$. This is very useful in applications like machine learning, where the data itself is noisy, and computing the exact result is not needed.

Key idea behind OverSketch: Sketching accelerates computation by eliminating redundancy in the matrix structure through dimension reduction. However, the coding-based approaches described in Section I-B have shown that redundancy can be *good* for combating stragglers if judiciously introduced into the computation. With these competing points of view in mind, our algorithm OverSketch works by "oversketching" the matrices to be multiplied by reducing dimensionality not to the minimum required for sketching accuracy, but rather to a slightly higher amount which simultaneously ensures both the accuracy guarantees and speedups of sketching *and* the straggler resilience afforded by the redundancy which was not eliminated in the sketch. OverSketch further reduces asymptotic costs by adopting the idea of block partitioning from HPC, suitably adapted for a serverless architecture.

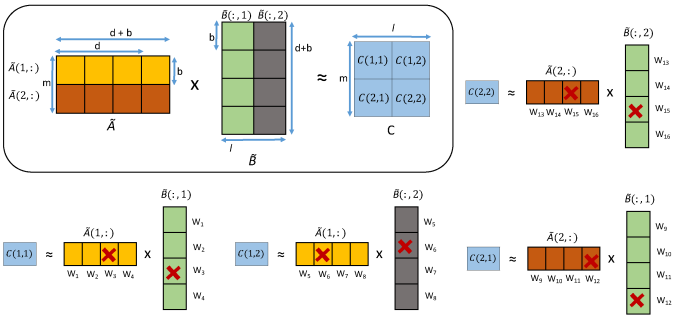


Figure 5: An illustration of multiplication of $m \times z$ matrix $\tilde{\mathbf{A}}$ and $z \times l$ matrix $\tilde{\mathbf{B}}$, where $z = d + b$ assures resiliency against one straggler per block of $\mathbf{C}$, and d is chosen by the user to guarantee a desired accuracy. Here, $m = l = 2b$, $d = 3b$, where b is the block-size for blocked matrix multiplication. This scheme ensures one worker can be ignored while calculating each block of $\mathbf{C}$.

Next, we propose a sketching scheme for OverSketch and describe the process of straggler mitigation in detail.

A. OverSketch: The Algorithm

During blocked matrix multiplication, the (i, j)-th block of $\mathbf{C}$ is computed by assimilating results from d/b workers who compute the product $\tilde{\mathbf{A}}(i, k) \times \tilde{\mathbf{B}}(k, j)$, for $k = 1, \cdots, d/b$. Thus, the computation $\mathbf{C}(i, j)$ can be viewed as the product of the row sub-block $\tilde{\mathbf{A}}(i, :) \in \mathbb{R}^{b\times d}$ of $\tilde{\mathbf{A}}$ and the column sub-block $\tilde{\mathbf{B}}(:, j) \in \mathbb{R}^{d\times b}$ of $\tilde{\mathbf{B}}$. An illustration is shown in Figure 5. Assuming d is large enough to guarantee the required accuracy in $\mathbf{C}$, we increase the sketch dimension from d to $z = d + eb$, where e is the worst case number of stragglers in $N = d/b$ workers. For the example in Figure 5, $e = 1$. To get a better insight on e, we observe in our simulations for cloud systems like AWS lambda and EC2 that the number of stragglers is $< 5\%$ for most runs. Thus, if $N = d/b = 40$, i.e. 40 workers compute one block of $\mathbf{C}$, then $e \approx 2$ is sufficient to get similar accuracy for matrix multiplication. Next, we describe how to compute the sketched matrices $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{B}}$.

Many sketching techniques have been proposed recently for approximate matrix computations. For example, to sketch a $m \times n$ matrix $\mathbf{A}$ with sketch dimension d, Gaussian projection takes $O(mnd)$ time, Subsampled Randomized Hadamard Transform (SRHT) takes $O(mn \log n)$ time, count sketch takes $O(nnz(A))$ time, where $nnz(\cdot)$ is the number of non-zero entries [17], [18]. Count sketch is one of the most popular sketching techniques as it requires linear time to compute the matrix sketch with similar approximation guarantees.

Algorithm 1: OverSketch: Distributed blocked matrix multiplication for the Cloud

Input : Matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{n \times l}$, sketch dimension z, straggler tolerance e

Result: $\mathbf{C} \approx \mathbf{A} \times \mathbf{B}$

1. **Sketching**: Obtain $\tilde{\mathbf{A}} = \mathbf{AS}$ and $\tilde{\mathbf{B}} = \mathbf{S}^T\mathbf{B}$ using $\mathbf{S} \in \mathbb{R}^{n \times z}$ from (3) in a distributed fashion
2. **Block partitioning**: Divide $\tilde{\mathbf{A}}$ into $m/b \times z/b$ matrix and $\mathbf{B}$ into $z/b \times l/b$ matrix of $b \times b$ blocks where b is the block-size
3. **Computation phase**: Multiply $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{B}}$ using blocked partitioning. This step invokes mlz/b^3 workers, where z/b workers are used per block of $\mathbf{C}$
4. **Termination**: Stop computation when any d/b workers return their results for each of the ml/b^2 blocks of $\mathbf{C}$, where $d = z - eb$
5. **Reduction phase**: Invoke ml/b^2 workers for reduction at each block of $\mathbf{C}$ on computed results

To compute the count sketch of $\mathbf{A} \in \mathbb{R}^{m \times n}$ of sketch dimension b, each column in $\mathbf{A}$ is multiplied by -1 with probability 0.5 and then mapped to an integer sampled uniformly from $\{1, 2, \cdots, b\}$. Then, to compute the sketch $\tilde{\mathbf{A}}_c = \mathbf{AS}_c$, columns with the same mapped value are summed. An example of count sketch matrix with $n = 9$ and $b = 3$ is

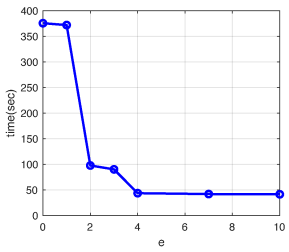
$$\mathbf{S}_c^T = \begin{bmatrix} 0 & 0 & 0 & 1 & -1 & 0 & -1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & -1 \end{bmatrix}. \quad (2)$$

The sparse structure of $\mathbf{S}_c$ ensures that the computation of sketch takes $O(nnz(A))$ time. However, a drawback of the desirable sparse structure of count sketch is that it cannot be directly employed for straggler mitigation in blocked matrix multiplication as it would imply complete loss of information from a subset of columns of $\mathbf{A}$. We refer the readers to [14] for a detailed example.

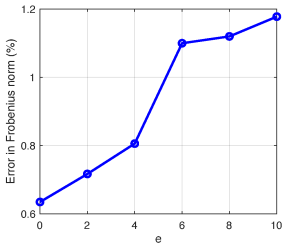
To facilitate straggler mitigation for blocked matrix multiplication, we propose a new sketch matrix $\mathbf{S}$, inspired by count sketch, and define it as

$$\mathbf{S} = \frac{1}{\sqrt{N}}(\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_{N+e}), \quad (3)$$

where $N = d/b$, e is the expected number of stragglers per block of $\mathbf{C}$ and $\mathbf{S}_i \in \mathbb{R}^{n \times b}$, for $i = 1, 2, \cdots, (N + e)$, is a count sketch matrix with dimension b. Thus, the total sketch-dimension for the sketch matrix in (3) is $z = (N + e)b = d + eb$. Computation of this sketch takes $O(nnz(\mathbf{A})(N + e))$ time and can be implemented in a distributed fashion trivially, where $(N + e)$ is the number of workers per block of $\mathbf{C}$ and is a constant less than 30 for most practical cases. We describe OverSketch in detail in Algorithm 1. We prove statistical guarantees on the accuracy of our sketching based matrix multiplication algorithm next.



(a) Time statistics for OverSketch on AWS Lambda for the straggler profile in Figure 1



(b) Frobenius norm error for sketched matrix product

Figure 6: Time and approximation error for OverSketch with 3000 workers when e, the number of workers ignored per block of $\mathbf{C}$, is varied from 0 to 10.

B. OverSketch: Approximation guarantees

Definition 4.1: We say that an approximate matrix multiplication of two matrices $\mathbf{A}$ and $\mathbf{B}$ using sketch $\mathbf{S}$, given by $\mathbf{ASS}^T\mathbf{B}$, is (ϵ, θ) accurate if, with probability at least $(1 - \theta)$, it satisfies

$$||\mathbf{AB} - \mathbf{ASS}^T\mathbf{B}||_F^2 \leq \epsilon ||\mathbf{A}||_F^2 ||\mathbf{B}||_F^2.$$

Now, for blocked matrix multiplication using OverSketch and as illustrated in Figure 5, the following holds

Theorem 4.1: Computing $(\mathbf{AS}) \times (\mathbf{S}^T\mathbf{B})$ using sketch $\mathbf{S} \in \mathbb{R}^{n \times z}$ in (3) and $d = \frac{2}{\epsilon\theta}$, while ignoring e stragglers among any $\frac{z}{b}$ workers, is (ϵ, θ) accurate.

We refer the readers to Appendix B in [14] for proof.

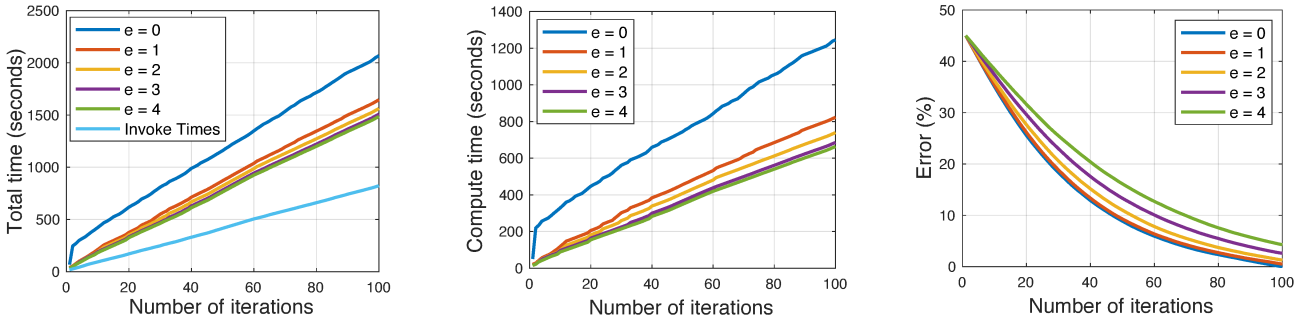
V. Experimental Results

A. Blocked Matrix Multiplication on AWS Lambda

We implement the straggler-resilient blocked matrix multiplication described above in the serverless computing platform *Pywren* [4], [5][3], on the AWS Lambda cloud system to compute an approximate $\mathbf{C} = \mathbf{AS} \times \mathbf{S}^T\mathbf{B}$ with $b = 2048, m = l = 10b, n = 60b$ and $\mathbf{S}$ as defined in (3) with sketch dimension $z = 30b$. Throughout this experiment, we take $\mathbf{A}$ and $\mathbf{B}$ to be constant matrices where the entries of $\mathbf{A}$ are given by $\mathbf{A}(x, y) = x + y$ for all $x \in [1, m]$ and $y \in [1, n]$ and $\mathbf{B} = \mathbf{A}^T$. Thus, to compute (i, j)-th $b \times b$ block of $\mathbf{C}$, 30 nodes compute the product of $\tilde{\mathbf{A}}(i, :)$ and $\tilde{\mathbf{B}}(:, j)$, where $\tilde{\mathbf{A}} = \mathbf{AS}$ and $\tilde{\mathbf{B}} = \mathbf{S}^T\mathbf{B}$. While collecting results, we ignore e workers for each block of $\mathbf{C}$, where e is varied from 0 to 10.

The time statistics are plotted in Figure 6a. The corresponding worker job times are shown in Figure 1, where the median job time is around 42 seconds, and some stragglers return their results around 100 seconds and some others take up to 375 seconds. We note that the compute time for matrix multiplication reduces by a factor of 9 if we ignore at most 4 workers per 30 workers that compute a block of

[3] A working implementation of OverSketch is available at https://github.com/vvipgupta/OverSketch



(a) Plot of total time (i.e., invocation time plus computation time) versus number of iterations.

(b) Plot of computation time versus number of iterations. When $e = 1$, algorithm takes 7 minutes less compared to when $e = 0$.

(c) Plot of percentage error versus iterations. Ignoring one worker per block of $\mathbf{C}$ has negligible affect on the convergence.

Figure 7: Time statistics and optimality gap on AWS Lambda while solving the LP in (4) using interior point methods, where e is the number of workers ignored per block of $\mathbf{C}$.

$\mathbf{C}$. In figure 6b, for same $\mathbf{A}$ and $\mathbf{B}$, we plot average error in matrix multiplication by generating ten instances of sketches and averaging the error in Frobenius norm, $\frac{||\mathbf{AB}-\mathbf{ASS}^T\mathbf{B}||_F}{||\mathbf{AB}||_F}$, across instances. We see that the average error is only 0.8% when 4 workers are ignored.

B. Solving Optimization Problems with Sketched Matrix multiplication

Matrix multiplication is the bottleneck of many optimization problems. Thus, sketching has been applied to solve several fairly common optimization problems using second-order methods, like linear programs, maximum likelihood estimation, generalized linear models like least squares and logistic regression, semi-definite programs, support vector machines, kernel ridge regression, etc., with essentially same convergence guarantees as exact matrix multiplication (see [19], [20], for example). As an instance, we solve the following linear program (LP) using interior point methods on AWS Lambda

$$\underset{\mathbf{Ax}\leq\mathbf{b}}{\text{minimize}}\ \mathbf{c}^T\mathbf{x}, \tag{4}$$

where $\mathbf{x} \in \mathbb{R}^{m\times 1}, \mathbf{c} \in \mathbb{R}^{m\times 1}, \mathbf{b} \in \mathbb{R}^{n\times 1}$ and $\mathbf{A} \in \mathbb{R}^{n\times m}$ is the constraint matrix with $n > m$. To solve (4) using the logarithmic barrier method, we solve the following sequence of problems using Newton's method

$$\min_{\mathbf{x}\in\mathbb{R}^m} f(\mathbf{x}) = \min_{\mathbf{x}\in\mathbb{R}^m} \left(\tau\mathbf{c}^T\mathbf{x} - \sum_{i=1}^{n}\log(b_i - \mathbf{a}_i\mathbf{x})\right), \tag{5}$$

where $\mathbf{a}_i$ is the i-th row of $\mathbf{A}$, τ is increased geometrically as $\tau = 2\tau$ after every 10 iterations and the total number of iterations is 100. The update in the t-th iteration is given by $\mathbf{x}_{t+1} = \mathbf{x}_t - \eta(\nabla^2 f(\mathbf{x}_t))^{-1}\nabla f(\mathbf{x}_t)$, where $\mathbf{x}_t$ is the estimate of the solution in the t-th iteration and η is the appropriate step-size. The gradient and Hessian for the objective in (5) are given by

$$\nabla f(\mathbf{x}) = \tau\mathbf{c} + \sum_{i=1}^{n}\frac{\mathbf{a}_i^T}{b_i - \mathbf{a}_i^T\mathbf{x}} \text{ and} \tag{6}$$

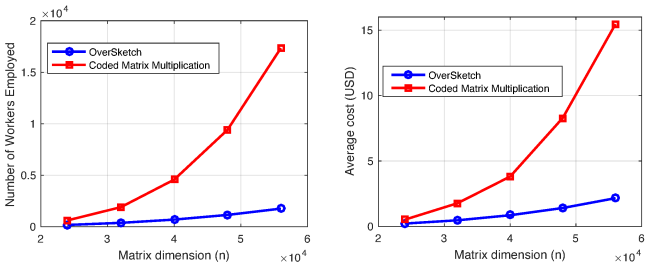
$$\nabla^2 f(\mathbf{x}) = \mathbf{A}^T\text{diag}\frac{1}{(bi - \mathbf{a}_i\mathbf{x})^2}\mathbf{A}, \tag{7}$$

respectively. The square root of the Hessian is given by $\nabla^2 f(\mathbf{x})^{1/2} = \text{diag}\frac{1}{|bi-\mathbf{a}_i\mathbf{x}|}\mathbf{A}$. The computation of Hessian requires $O(nm^2)$ time and is the bottleneck in each iteration. Thus, we use our distributed and sketching-based blocked matrix multiplication scheme to mitigate stragglers while evaluating the Hessian approximately, i.e. $\nabla^2 f(\mathbf{x}) \approx (\mathbf{S}\nabla^2 f(\mathbf{x})^{1/2})^T \times (\mathbf{S}\nabla^2 f(\mathbf{x})^{1/2})$, on AWS Lambda, where $\mathbf{S}$ is defined in (3).

We take the block size, b, to be 1000, the dimensions of $\mathbf{A}$ to be $n = 40b$ and $m = 5b$ and the sketch dimension to be $z = 20b$. We use a total of 500 workers in each iteration. Thus, to compute each $b \times b$ block of $\mathbf{C}$, 20 workers are assigned to compute matrix multiplication on two $b \times b$ blocks. We depict the time and error versus iterations in figure 7. We plot our results for different values of e, where e is the number of workers ignored per block of $\mathbf{C}$. In our simulations, each iteration includes around 9 seconds of invocation time to launch AWS Lambda workers and assign tasks. In figure 7a, we plot the total time that includes the invocation time and computation time versus iterations. In 7b, we exclude the invocation time and plot just the compute time in each iteration iteration and observe 34% savings in solving (4) when $e = 1$, whereas the effect on error with respect to the optimal solution is insignificant (as shown in figure 7c).

C. Comparison with Existing Straggler Mitigation Schemes

In this section, we compare OverSketch with an existing coding-theory based straggler mitigation scheme described in [9]. An illustration for [9] is shown in Figure 2. We multiply two square matrices $\mathbf{A}$ and $\mathbf{B}$ of dimension n on AWS



(a) Number of workers required for OverSketch and [9]

(b) Cost for distributed matrix multiplication for OverSketch and [9]

Figure 8: Comparison of OverSketch with coded theory based scheme in [9] on AWS Lambda. OverSketch requires asymptotically less workers which translates to significant savings in cost.

Lambda using the two schemes, where $\mathbf{A}(x, y) = x + y$ and $\mathbf{B}(x, y) = x \times y$ for all $x, y \in [1, n]$. We limit the bandwidth of each worker by 400 MB (i.e. around 48 million entries, where each entry takes 8 bytes) for a fair comparison. Thus, we have $3b^2 = 48 \times 10^6$, or $b = 4000$ for OverSketch and $2an + a^2 = 48 \times 10^6$ for [9], where a is the size of the row-block of $\mathbf{A}$ (and column-block of $\mathbf{B}$). We vary the matrix dimension n from $6b = 24000$ to $14b = 56000$. For OverSketch, we take the sketch dimension z to be $n/2 + b$, and take $e = 1$, i.e., ignore one straggler per block of $\mathbf{C}$. For straggler mitigation in [9], we add one parity row in $\mathbf{A}$ and one parity column in $\mathbf{B}$. In Figures 8a and 8b, we compare the workers required and average cost in dollars, respectively, for the two schemes. We note that OverSketch requires asymptotically fewer workers, and it translates to the cost of doing matrix multiplication. This is because the running time at each worker is heavily dependent on communication, which is the same for both the schemes. For $n = 20000$, the average error in Frobenius norm for OverSketch is less than 2%, and decreases as n is increased.

The scheme in [9] requires an additional decoding phase, and assume the existence of a powerful master that can store the entire product $\mathbf{C}$ in memory and decode for the missing blocks using the redundant chunks. This is also true for the other schemes in [10]–[12]. Moreover, these schemes would fail when the number of stragglers is more than the provisioned redundancy while OverSketch has a 'graceful degradation' as one can get away by ignoring more workers than provisioned at the cost of accuracy of the product.

References

[1] G. Ballard, E. Carson, J. Demmel, M. Hoemmen, N. Knight, and O. Schwartz, "Communication lower bounds and optimal algorithms for numerical linear algebra," *Acta Numerica*, vol. 23, pp. 1–155, 2014.

[2] E. Solomonik and J. Demmel, "Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms," in *Proceedings of the 17th International Conference on Parallel Processing*, 2011, pp. 90–109.

[3] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, and P. Suter, *Serverless Computing: Current Trends and Open Problems*. Springer Singapore, 2017.

[4] 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*. ACM, 2017, pp. 445–451.

[5] V. Shankar, K. Krauth, Q. Pu, E. Jonas, S. Venkataraman, I. Stoica, B. Recht, and J. Ragan-Kelley, "numpywren: serverless linear algebra," *ArXiv e-prints*, Oct. 2018.

[6] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.

[7] T. Hoefler, T. Schneider, and A. Lumsdaine, "Characterizing the influence of system noise on large-scale applications by simulation," in *Proc. of the ACM/IEEE Int. Conf. for High Perf. Comp., Networking, Storage and Analysis*, 2010, pp. 1–11.

[8] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, 2018.

[9] K. Lee, C. Suh, and K. Ramchandran, "High-dimensional coded matrix multiplication," in *IEEE Int. Sym. on Information Theory (ISIT), 2017*. IEEE, 2017, pp. 2418–2422.

[10] T. Baharav, K. Lee, O. Ocal, and K. Ramchandran, "Straggler-proofing massive-scale distributed matrix multiplication with d-dimensional product codes," in *IEEE Int. Sym. on Information Theory (ISIT), 2018*. IEEE, 2018.

[11] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Adv. in Neural Inf. Processing Systems*, 2017, pp. 4403–4413.

[12] S. Dutta, M. Fahim, F. Haddadpour, H. Jeong, V. Cadambe, and P. Grover, "On the optimal recovery threshold of coded matrix multiplication," *arXiv preprint arXiv:1801.10292*, 2018.

[13] J. Zhu, Y. Pu, V. Gupta, C. Tomlin, and K. Ramchandran, "A sequential approximation framework for coded distributed optimization," in *Annual Allerton Conf. on Communication, Control, and Computing, 2017*. IEEE, 2017, pp. 1240–1247.

[14] V. Gupta, S. Wang, T. Courtade, and K. Ramchandran, "OverSketch: Approximate Matrix Multiplication for the Cloud," *ArXiv e-prints*, Nov. 2018.

[15] R. A. van de Geijn and J. Watts, "Summa: scalable universal matrix multiplication algorithm," *Concurrency - Practice and Experience*, vol. 9, pp. 255–274, 1997.

[16] P. Drineas, R. Kannan, and M. W. Mahoney, "Fast monte carlo algorithms for matrices I: Approximating matrix multiplication," *SIAM Journal on Computing*, vol. 36, no. 1, pp. 132–157, 2006.

[17] D. P. Woodruff, "Sketching as a tool for numerical linear algebra," *Found. Trends Theor. Comput. Sci.*, vol. 10, pp. 1–157, 2014.

[18] S. Wang, "A practical guide to randomized matrix computations with MATLAB implementations," *arXiv:1505.07570*, 2015.

[19] M. Pilanci and M. J. Wainwright, "Newton sketch: A near linear-time optimization algorithm with linear-quadratic convergence," *SIAM Jour. on Opt.*, vol. 27, pp. 205–245, 2017.

[20] Y. Yang, M. Pilanci, and M. J. Wainwright, "Randomized sketches for kernels: Fast and optimal non-parametric regression," *stat*, vol. 1050, pp. 1–25, 2015.