Exploiting Locality in Codes for Serverless Straggler Mitigation

Dominic Carrano

EE 229B: Error Control Coding

May 2019

1 Introduction

Serverless platforms — such as Amazon Web Services (AWS) Lambda, Google Cloud functions, and Microsoft Azure functions — have introduced a new distributed cloud computing paradigm by managing the servers that computation is done on. This has several advantages for the user. It abstracts away any server management tasks from the user, since this is handled by the cloud provider — hence the name *serverless*. Recently, the use of serverless systems has garnered attention in both industry and academia, mainly due to its ease of use and capability to easily scale to large tasks [1, 2, 3]. According to the *Berkeley view on Serverless Computing* [3], the traditional client-server based computing model will see a noticeable decline in the near future, in favor of serverless systems. The authors advocate serverless computing as a paradigm that "provides an interface that greatly simplifies cloud programming, and represents an evolution that parallels the transition from assembly language to high-level programming languages".

However, the main problem with cloud-based systems is that workers suffer from system noise caused by a combination of hardware failure, limitied availability of shared resources, network latency, and other issues [4, 5]. As a result, there is significant variance in job completion times. This often appears in the form of a small fraction of the workers, called *stragglers*, that take much longer to complete than the median job time, significantly slowing the computation.

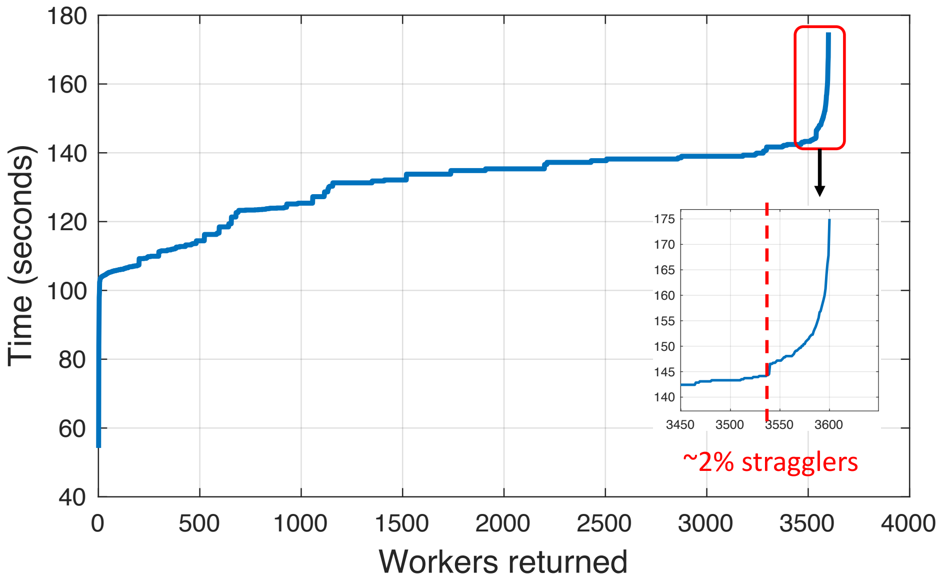


Figure 1: Average job times for matrix multiplication of two square matrices of dimension 120000 each using 3600 AWS Lambda workers. The median job time is around 135 seconds, and around 2% of the nodes take up to 175 seconds on average.

Empirical statistics for average worker job times for multiplying two square matrices of dimension 120000 using 3600 workers are shown in Fig. 1 for AWS Lambda. Notably, there are a few workers (approximately 2%) that take much longer than the median job time, severely degrading the overall efficiency of the system.

A technique known as *speculative execution* has traditionally been used to deal with stragglers (e.g., in Hadoop MapReduce [6] and Apache Spark [7]). Speculative execution works by speculating which workers are lagging, or may slow down in the future, and then assigning their jobs to new workers, leaving the original worker to continue running. Whichever of the two finishes first submits the result. This approach has several disadvantages. Constant monitoring of jobs is required, which might be costly if there are many workers in the system. Additionally, it is possible that a node will straggle just before completion. By the time the job is resubmitted, the additional time spent doing so will have added non-negligible overhead. The situation is even worse for smaller jobs, as spinning up an extra node requires additional invocation and setup time which can exceed the original job time.

Error control codes are a key element of digital transmission and storage technologies, vastly improving robustness over uncoded systems. Coding-theoretic ideas have recently been proposed to introduce redundancy into distributed computations for improved straggler resilience by reducing overall job time variability [8, 9, 10, 11, 12, 13, 14]. The idea behind this coded computation paradigm is that stragglers whose results are missing from the output can be treated as erasures. Thus, performing some small amount of redundant computation will provide straggler resiliency, just as sending redundant bits over a communication channel provides a degree of robustness. The main advantage of these techniques is that no feedback or coordination is necessary, which is the case with speculative execution. However, one disadvantage of coding-based methods is that separate encoding and decoding phases introduce additional communication and potentially large computational burden at a master node, which is generally incompatible with serverless computing platforms.

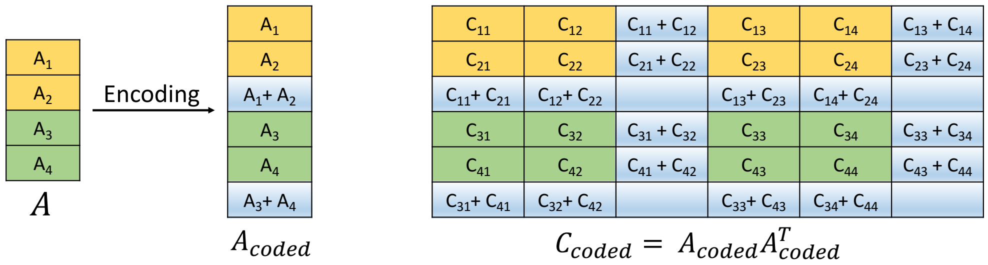
In this work, we advocate a coding-based approach to accelerate distributed computation in serverless computing. Specifically, we consider the case of matrix-matrix multiplication, and show that a coded approach outperforms speculative execution. We make the case for introducing locality into codes used in serverless straggler mitigation so that decoding and encoding is amenable to a parallel implementation and results in reduced communication overhead, making coded computation practical in the serverless setting.

2 Approach

A common computational bottleneck in machine learning algorithms is matrix-matrix multiplication. The problem is computing

$$\mathbf{C} = \mathbf{A}\mathbf{B}^T$$

where $\mathbf{A} \in \mathbb{R}^{m \times n}, \mathbf{B} \in \mathbb{R}^{\ell \times n}$. We consider blocked partitioning of $\mathbf{A}$ and $\mathbf{B}$ for the purpose of dividing up the large multiplication task among the workers. This is because it is known in High Performance Computing (HPC) that blocked partitioning yields communication-efficient algorithms for matrix multiplication [15, 16]. Moreover, for the case of serverless systems where the number of workers are flexible but the system is bottle-necked by memory at each worker, authors in [17] recently showed that blocked partitioning is cost-optimal.



C_{11}	C_{12}	$C_{11} + C_{12}$	C_{13}	C_{14}	$C_{13} + C_{14}$
C_{21}	C_{22}	$C_{21} + C_{22}$	C_{23}	C_{24}	$C_{23} + C_{24}$
$C_{11} + C_{21}$	$C_{12} + C_{22}$		$C_{13} + C_{23}$	$C_{14} + C_{24}$	
C_{31}	C_{32}	$C_{31} + C_{32}$	C_{33}	C_{34}	$C_{33} + C_{34}$
C_{41}	C_{42}	$C_{41} + C_{42}$	C_{43}	C_{44}	$C_{43} + C_{44}$
$C_{31} + C_{41}$	$C_{32} + C_{42}$		$C_{33} + C_{43}$	$C_{34} + C_{44}$	

Figure 2: A toy example of computing $\mathbf{C} = \mathbf{A}\mathbf{A}^T$ where $\mathbf{A} \in \mathbb{R}^{4\times 1}$ and $L = 2$. The result $\mathbf{C}$ is recovered by taking the systematic part of $\mathbf{C}_{\text{coded}}$. The local coding along the rows of $\mathbf{A}$ leads to a four sets of 3×3 blocks in $\mathbf{C}_{\text{coded}}$. This allows four workers to decode one 3×3 block each in parallel. For a general L, $\mathbf{C}_{\text{coded}}$ contains sets of $(L+1) \times (L+1)$ blocks that can each be decoded separately.

For our experiment, we coded the row blocks of $\mathbf{A}$ and $\mathbf{B}$ by inserting a parity block after every L blocks, where L is some parameter chosen to control the amount of redundancy the code introduces and is assumed to divide the number of rows in $\mathbf{A}$ and $\mathbf{B}$. The parity is computed as a sum over the previous L rows. In Fig. 2, we illustrate the encoding of $\mathbf{A}$ which is a 4×1 block data matrix with $L = 2$. Note the locality of the code: every L rows of A are separately encoded, leading to sets of $(L+1) \times (L+1)$ that can be separately decoded. This is the key idea behind the reduced decoding time. In essence, we have an $((L+1)^2, L^2)$ block erasure code and each worker is tasked with decoding one codeword.

This method gives involves a 3-stage process for coded matrix multiplication: encoding, computing, and decoding. In contrast, speculative execution involves two stages: computing, and then recomputing all tasks that straggled. The code's locality makes it particularly amenable to parallel encoding and decoding approaches. For encoding, each worker computes the sum of L blocks, producing a single parity block. For decoding, we use a peeling decoder to recover the systematic part of each $(L+1) \times (L+1)$ block, constructing the final result matrix $\mathbf{C}$ from these systematic results. For the multiplication stage, each worker computes one entry of $\mathbf{C}_{\text{coded}}$.

3 Results

For our experiments, we compared speculative execution to coded computation, both with 21% redundancy. In the case of coding, this is achieved by using

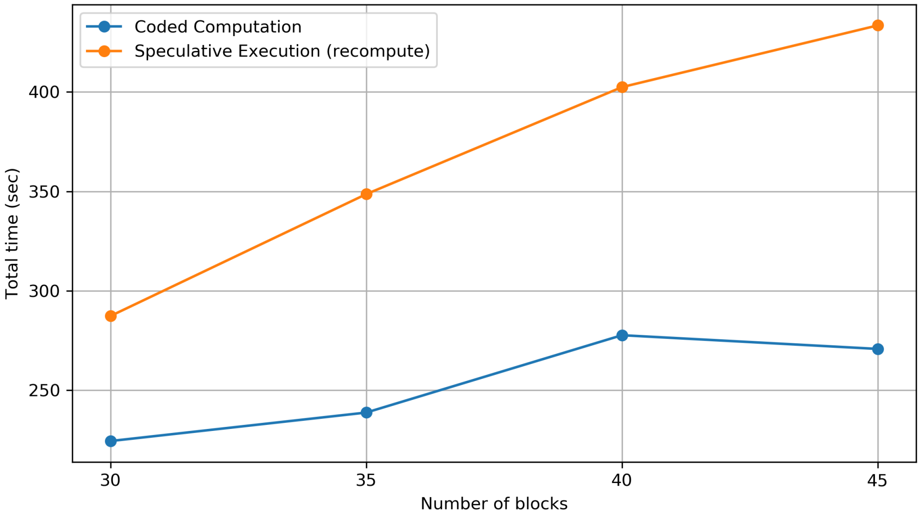


Figure 3: Comparison of coded computing versus speculative execution for matrix multiplication on $k \times k$ block matrices, with $k \in \{30, 35, 40, 45\}$ and 3000×3000 data blocks. This means the matrices' respective dimensions are $90000, 105000, 120000, 135000$. Note these times are not averaged, hence the drop in runtime from 40 to 45 for the coded case.

a code with $L = 10$, so that the 11×11 blocks considered by the decoding workers contain 100 data blocks for every 21 parity blocks. For speculative execution, we wait for 79% of workers to finish before recomputing any incomplete jobs. In both cases, we multiply two square matrices of various sizes with 3000×3000 blocks.

In Fig. 3, the runtimes are shown for both approaches as a function of the input sizes. These times are not averaged over multiple runs, hence the drop in time from 40 to 45 for the coded case. Coded multiplication outperforms speculative execution by approximately 25-35%.

For the coded approach, the computation and decoding phases account for approximately 90% of the time taken, with computation alone taking 60-80% of the time. In the case of speculative execution, computing and recomputing account for approximately 60% and 40% of the total time each. More formally, for speculative execution, the total time is

$$T_{\text{spec}} = \underbrace{T_{\text{comp, spec}}}_{\text{Compute}} + \underbrace{T_{\text{recomp}}}_{\text{Recompute}}$$

and for coded multiplication, the total time is

$$T_{\text{coded}} = \underbrace{T_{\text{enc}}}_{\text{Encoding}} + \underbrace{T_{\text{comp, coded}}}_{\text{Compute}} + \underbrace{T_{\text{dec}}}_{\text{Decoding}}$$

where $T_{\text{comp, coded}} > T_{\text{comp, spec}}$ on average since the coded approach involves more multiplications. Since $T_{\text{enc}} << T_{\text{dec}}$, we see coded multiplication outperforming speculative execution because of the low in decoding time of this scheme: due to the localized structure of $\mathbf{C}_{\text{coded}}$, enabling each worker to decode in parallel, we significantly reduce communication costs, and thus decoding costs.

4 Analysis

4.1 Resiliency to Stragglers

We have empirically verified that p, the probability of any worker straggling, is approximately 2%. We can analyze how robust the coded approach is by considering the probability it is unable to decode due to straggling workers. In doing so, it is expedient to introduce the notion of a *trapping set*.

Definition 1. Consider the $(L+1) \times (L+1)$ grid structure a decoding worker operates on. A trapping set is any subset of these $(L+1)^2$ blocks such that none of the blocks can be decoded if they all straggle.

In considering the case of a decoding worker encountering a trapping set S, it may come across more than $|S|$ stragglers, but we assume that exactly $|S|$ of them cannot be recovered. That is, any trapping set is equivalent to having the same workers straggle with possibly more workers straggling, so long as the others are all decodable. Some examples of trapping sets are shown Fig. 4. A key fact in our analysis of this scheme is that a given decoding worker is able to finish if and only if it does not encounter a trapping set. Thus, an upper bound on the probability of being unable to decode is equivalent to an upper bound on the probability of encountering a trapping set. While doing so may initially seem intractable, the analysis is simplified by two key observations, which we formalize as Lemmas.

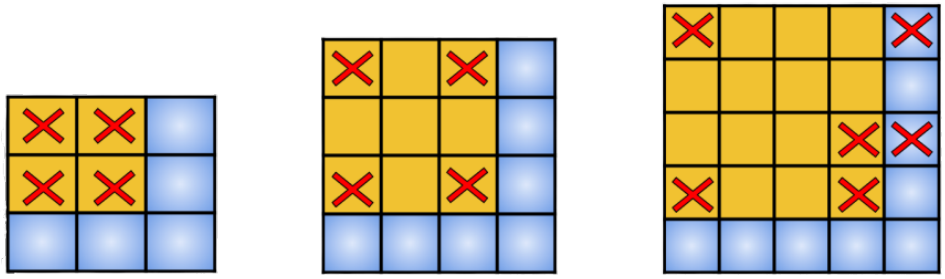


Figure 4: Some examples of trapping sets, as viewed from a single decoding worker's $(L+1) \times (L+1)$ grid. The yellow blocks correspond to the systematic part of the code, and blue blocks to the parity. Blocks marked with an "X" are stragglers. The 5x5 grid $(L = 4)$ on the right is a trapping set of size 6; the other two trapping sets $(L = 3$ in the middle and $L = 2$ on the left) have size 4.

Lemma 1. No trapping sets of size 3 or size 5 exist.

Proof: In the size 3 case, since $L \geq 1$, $L + 1 \geq 2$ so all workers operate on grids that are 2×2 or larger. Assuming 3 stragglers (it's possible there are more, but by definition, they are all decodable, and thus irrelevant), at least one of them is the only straggler in its row or column so the set is reduced to 2 stragglers, and is not a size 3 trapping set. In the size 5 case, consider the task of trying to construct such a trapping set. Pick any spot (i_1, j_1) on the $(L + 1) \times (L + 1)$ grid for the first straggler. Now, block it from being row decodable by adding the 2nd straggler at (i_1, j_2). Now, block the first straggler from being column decodable by adding the 3rd straggler at (i_3, j_1). This leaves the 2nd straggler column decodable and the 3rd row decodable. If we place the fourth at (i_3, j_2), simultaneously blocking stragglers 2 and 3, then the fifth block will be decodable regardless of where it is placed. Suppose that instead we place the fourth at (i_4, j_2) to block the second, and the fifth at (i_3, j_5) to block the third. If $i_4 \neq i_3$ and $j_2 \neq j_5$, then the fourth and fifth blocks are both decodable. Note we cannot have both $i_4 = i_3$ and $j_2 = j_5$, or these stragglers would correspond to the same block. Then, if $i_4 \neq i_3$, then the fourth block is decodable. If $j_2 \neq j_5$, then the fifth block is decodable. Thus no size 5 trapping sets can exist. ■

Lemma 2. Let S be a trapping set. Then $|S| \geq 4$.

Proof: Clearly, $|S| \geq 1$ as $|S| = 0$ would imply there are no stragglers. If $|S| = 1$, S cannot be a trapping set as the straggler can be recovered from either its row check or its column check, so $|S| \geq 2$. If $|S| = 2$, there are three cases to consider. In the first case, the stragglers lie on the same row and can be recovered using their respective column checks. In the second case, the stragglers lie on the same column and can be recovered using their respective row checks. In the third case, the stragglers lie on distinct rows and distinct columns, and can be separately decoded through either their respective row or column checks. Thus $|S| \geq 3$. By Lemma 1, $|S| \neq 3$, so $|S| \geq 4$. To show this bound is achievable, one can construct a size 4 trapping set by assuming blocks $(1,1),(1,2),(2,1),(2,2)$ all straggle, where these tuples correspond to row and column indices, respectively. That is, these four blocks are the upper leftmost block, its two neighbors, and the block diagonally down and to the right of it. This is indeed a trapping set: rows 1 and 2 as well as columns 1 and 2 each have two stragglers. Then no blocks are decodable, since each row or column check can handle exactly 1 erasure. ■

Corollary 1. The $((L+1)^2, L^2)$ block erasure code can recover any three stragglers.

These lemmas greatly simplify our analysis of trapping sets. Rather than consider all possible trapping set sizes from 1 to $(L+1)^2$, we only need consider size 4 and size 6 through $(L+1)^2$ trapping sets. This may seem like a trivial gain at first. Intuitively, however, since encountering a straggler is unlikely (the probability of an individual worker straggling is $\approx 2\%$), and only a small subset of possible straggler configurations result in trapping sets, we expect that the probability of encountering a trapping set of size t should drop off very quickly as t increases. The easiest trapping sets to analyze are the smallest ones, when $t = 4$, so we can first investigate these and draw inspiration from their analysis in generalizing our understanding of trapping sets.

Lemma 3. The number of trapping sets of size 4 with exactly 4 stragglers is $(L+1)^2 L^2/4!$.

Proof: To count the number of size 4 trapping sets, we can place the stragglers one at a time and count the number of choices we have. The first straggler can be anywhere, giving $(L+1)^2$ options. The second straggler, which we use to prevent the first from being row decodable, can go in any of the other L spots in that row. The third, which we use to prevent the first from being column decodable, can go in any of the other L spots in that column. The fourth is responsible for simultaneously rendering the second and third undecodable, with its position fully determined by the second and third stragglers. But any permutation of this trapping set is the same trapping set, so this process overcounts by a factor of 4!, and there are $(L+1)^2L^2/4!$ size 4 trapping sets. ■

The constructive proof given for Lemma 3 suggests a more general procedure for constructing size trapping sets of size $t > 4$: place the first straggler anywhere on the decoder's grid. Then, choose any of the L vacant spots in its row for the second straggler to block the first. Repeat by placing stragglers in the row (or column; the pattern will alternate) of the previous straggler. Eventually, there ceases to be L options for successive stragglers due to preexisting ones, and the generalized formula will serve as an upper bound. We formalize this result in the following theroem.

Theorem 1. There are at most $(L+1)^2L^{t-2}/t!$ trapping sets of size t with exactly t stragglers.

Note that this only counts trapping sets where the number of stragglers is equal to t, the size of the trapping set. For example, adding in a fifth straggler to any trapping set with exactly four stragglers is still, according to Definition 1, a size 4 trapping set. Nevertheless, this provides a useful proxy for an upper bound on the probability of finding a trapping set: sum the term in Theorem 1 over all t from 4 to $(L+1)^2$, excluding $t = 5$ based on the result of Lemma 1, multiplying each term of the sum by $p^t(1-p)^{(L+1)^2-t}$, since this is the probability of exactly t stragglers in any particular configuration. For the case of $L = 10, p = .02$, this gives a bound of 2×10^{-4} on the probability of encountering a trapping set.

4.2 Decoding Costs

In serverless systems, communication cost is much greater than computation cost. As a result, codes requiring a high degree of communication between decoding workers will perform poorly in serverless computation tasks. Informed by this, our code has been designed with an eye for locality, allowing us to consider each decoding worker in isolation. As a result, for our scheme, decoding costs are primarily a function of the number of blocks a worker has to read in recovering missing stragglers. By considering the relationship between the number of stragglers S, a random variable with known statistics, and the number of blocks read, we can obtain an upper bound on the number of blocks read by a single decoding worker.

Lemma 4. $S \sim \text{Bin}((L+1)^2, p)$, where S is the number of stragglers at a particular decoding worker.

Proof: Each decoding worker operates on a grid with $(L+1)^2$ entries, each of which straggles independently with probability p. ■

We assume each worker can only fit one block in memory at a time. To decode any given straggler, the worker must read all other blocks in the straggler's row or column to recover it. Since the worker has an $(L+1)\times(L+1)$ grid, this implies decoding each straggler requires reading L blocks - the other L blocks besides the straggler in its row or column. We formalize this observation into Lemma 5.

Lemma 5. Let R be the number of blocks read by a decoding worker, and S the number of stragglers it has. Then $R = LS$.

Lemma 5 provides a useful starting point for obtaining an upper bound on R. Since L is a fixed constant (a parameter of our code) and S a binomial random variable, which has a known MGF, we can obtain a Chernoff bound on R.

Theorem 2. Let R be the number of blocks read by a worker when decoding. Then by the Chernoff bound,

$$\Pr(R \geq r) \leq \exp\left(-\frac{r}{L}\left(\ln\left(\frac{r}{Lp(L+1)^2}\right)+1\right)+p(L+1)^2\right)$$

Proof: By Lemma 5, $R = LS$, so the MGF of R, denoted $M_R(t)$, is given by

$$\begin{aligned}M_R(t) &= \mathbb{E}(\exp(tR)) = \mathbb{E}(\exp(tLS))\\ &= M_S(\tau)|_{\tau=tL} = (1-p+p\exp(tL))^{(L+1)^2}\end{aligned}$$

where we have used the fact that $M_S(\tau) = (1-p+p\exp(\tau))^{(L+1)^2}$ in the final equality since $S \sim \text{Bin}((L+1)^2, p)$ by Lemma 4. Applying the Chernoff bound, we have that $\forall t > 0$,

$$\begin{aligned}\Pr(R \geq r) &\leq \exp(-tr)M_R(t)\\ &= \exp(-tr)(1-p+p\exp(tL))^{(L+1)^2}\\ &= \exp(-tr)(1-p(1-\exp(tL)))^{(L+1)^2}\\ &\leq \exp(-tr)\exp(-p(1-\exp(tL))(L+1)^2)\\ &= \exp(-(tr+p(1-\exp(tL))(L+1)^2))\end{aligned}$$

where the fourth line applies the inequality $1 - x \leq \exp(-x)\ \forall x \in \mathbb{R}$. [1] To tighten the bound, note that it is strictly convex in t, and thus minimized where its derivative is zero. Thus the minimizer t^* is the solution to the following equation in t:

$$-\left(r - L(L+1)^2pe^{tL}\right)\exp(-(tr + (L+1)^2p(1-e^{tL}))) = 0$$

which is zero if and only if $\left(r - L(L+1)^2pe^{tL}\right) = 0$ since the exponential function is strictly non-negative, thus

$$\begin{aligned} L(L+1)^2pe^{Lt^*} &= r \\ e^{Lt^*} &= \frac{r}{L(L+1)^2p} \\ t^* &= \frac{1}{L}\ln\left(\frac{r}{L(L+1)^2p}\right) \end{aligned}$$

and substituting this back into the original bound gives the original result. Note that since the Chernoff bound requires $t > 0$, this result is only meaningful in the regime $r > L(L+1)^2p$. For the setup considered here ($L = 10, p = .02$), this corresponds to $r > 24.2$. ■

Theorem 2 tells us that the probability a worker reads more than r blocks decays at a faster than exponential rate. Fig. 5 shows this bound for the case of $L = 10$ and $p = .02$ considered here.

5 Conclusion

We presented a scheme for coded matrix-matrix multiplication that outperforms the popular method of speculative execution in a serverless computing environment by exploiting the locality of the code to significantly reduce decoding costs. All three stages of the coded approach are amenable to a parallel implementation, utilizing the dynamic scaling capabilities of serverless platforms. We proved upper bounds on the probability of a worker being unable to decode and the probability of a worker reading more than any fixed number of blocks when decoding. We argue that in the serverless setting, where communication costs greatly outweigh computation costs, performing

[1] It may seem that we have lost a degree of tightness in applying this inequality. However, this step makes the minimization over t more tractable, and it can be numerically verified that this new bound is within 1% of the original when both are optimized over t.

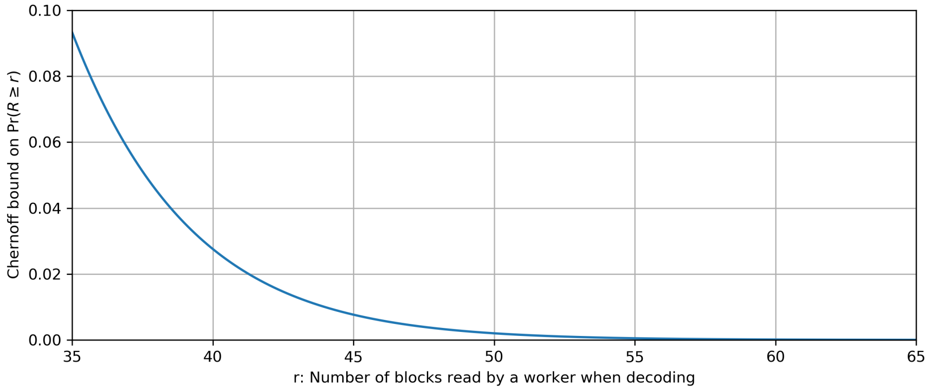


Figure 5: Chernoff bound on the probability of a decoding worker reading more than r blocks for the code and system considered here ($L = 10, p = .02$). With this code, each worker has 121 blocks, so in particular, the probability of a decoding worker reading more than half its blocks, $\Pr(R \geq 61)$, is at most 9×10^{-5}.

some redundant computation based on ideas from coding theory will outperform speculative execution, and that the design of such codes should leverage locality to attain low decoding costs.

6 Future Work

In the work here, we used a code with 21% redundancy and showed that it outperforms speculative execution. More fundamentally, the question of the minimum amount of redundancy needed to achieve a certain runtime or cost in US dollars remains open. Currently, there is no widely accepted cost model for evaluating and comparing serverless computing algorithms for various tasks such as matrix multiplication. We hope to develop such a model in the future informed by heuristics such as total runtime and US dollar costs as well as models currently in the HPC literature for server-based systems. This model would provide insight in addressing the question of what coding schemes are optimal in serverless computing tasks.

References

[1] 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*, pp. 445–451, ACM, 2017.

[2] 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.

[3] E. Jonas, J. Schleier-Smith, V. Sreekanti, C.-C. Tsai, A. Khandelwal, Q. Pu, V. Shankar, J. Carreira, K. Krauth, N. Yadwadkar, *et al.*, "Cloud programming simplified: A berkeley view on serverless computing," *arXiv preprint arXiv:1902.03383*, 2019.

[4] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, pp. 74–80, Feb. 2013.

[5] 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*, pp. 1–11, 2010.

[6] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, pp. 107–113, Jan. 2008.

[7] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, pp. 10–10, 2010.

[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] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, pp. 3368–3376, PMLR, 2017.

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

[11] 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.

[12] Q. Yu, M. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 4403–4413, Curran Associates, Inc., 2017.

[13] 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.

[14] 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*, pp. 1240–1247, IEEE, 2017.

[15] 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*, pp. 90–109, 2011.

[16] R. A. van de Geijn and J. Watts, "Summa: Scalable universal matrix multiplication algorithm," tech. rep., 1995.

[17] V. Gupta, S. Wang, T. Courtade, and K. Ramchandran, "Oversketch: Approximate matrix multiplication for the cloud," *IEEE International Conference on Big Data, Seattle, WA, USA*, 2018.