Dynamic Coding for Distributed Matrix Multiplication

Xian Su, Xiaodi Fan, Jun Li

School of Computing and Information Sciences, Florida International University

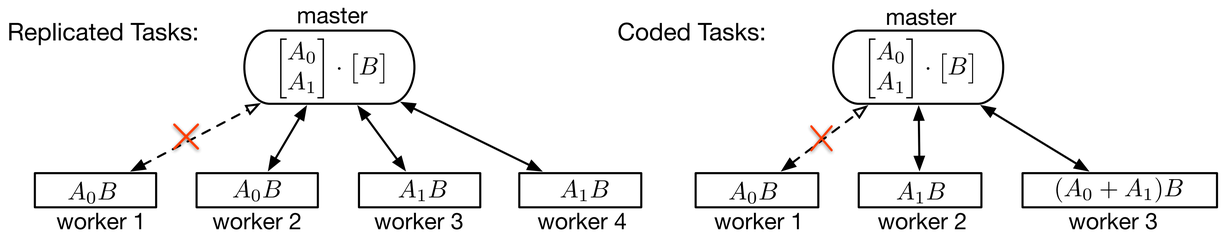
FIU Engineering & Computing — School of Computing & Information Sciences

Large-scale Matrix Multiplication

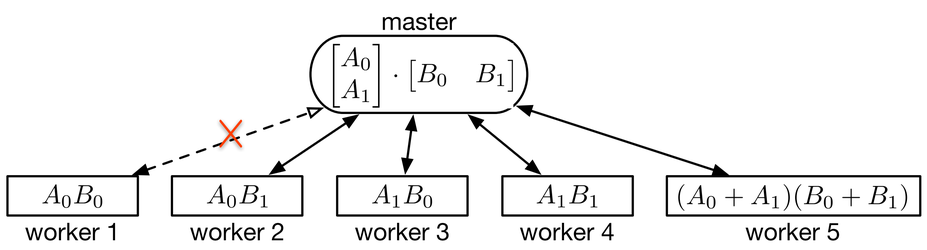
- Matrix multiplication is a fundamental operation in various machine learning algorithms. As the size of datasets grows, it becomes necessary to run matrix multiplication on multiple servers, where each server runs a task calculating a submatrix of the result.
- Servers in a distributed infrastructure are subject to various faulty behaviors.
- A task running on servers affected by a faulty server becomes a straggler. A straggler may significantly affect the performance of distributed matrix multiplication as the job depends on the results of all tasks.

Tolerating Stragglers with Coding

- Conventionally, stragglers can be tolerated by running replicated tasks on multiple servers.



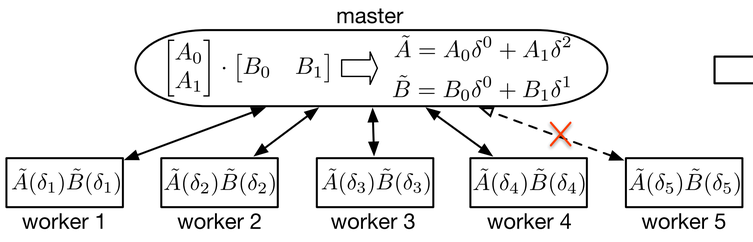
- To save the required resources, we can run additional coded tasks which compute the multiplication of coded matrices.
- Further, by dividing input matrices in more dimensions, we can reduce the size of tasks.



Major Coding Schemes for Matrix Multiplication

So far, there are three major coding schemes based on polynomials: polynomial codes, MadDot codes and entangled polynomial code. They have different partitions of input matrices.

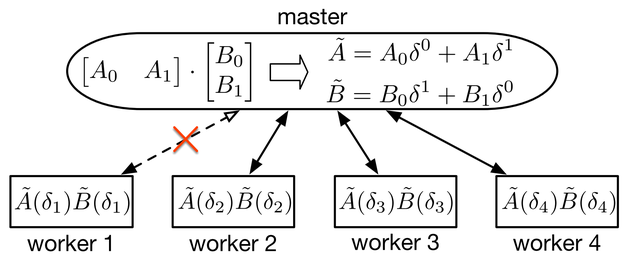
- Polynomial Code: Splitting input matrices A and B vertically and horizontally, respectively.



$$\Rightarrow \quad AB=\begin{bmatrix}A_0B_0 & A_0B_1\\ A_1B_0 & A_1B_1\end{bmatrix}$$

With $\tilde{A}(\delta)\tilde{B}(\delta)=A_0B_0\delta^0+A_0B_1\delta^1+A_1B_0\delta^2+A_1B_1\delta^3$, where δ is unique in each task, we can get AB based on the results of any 4 tasks.

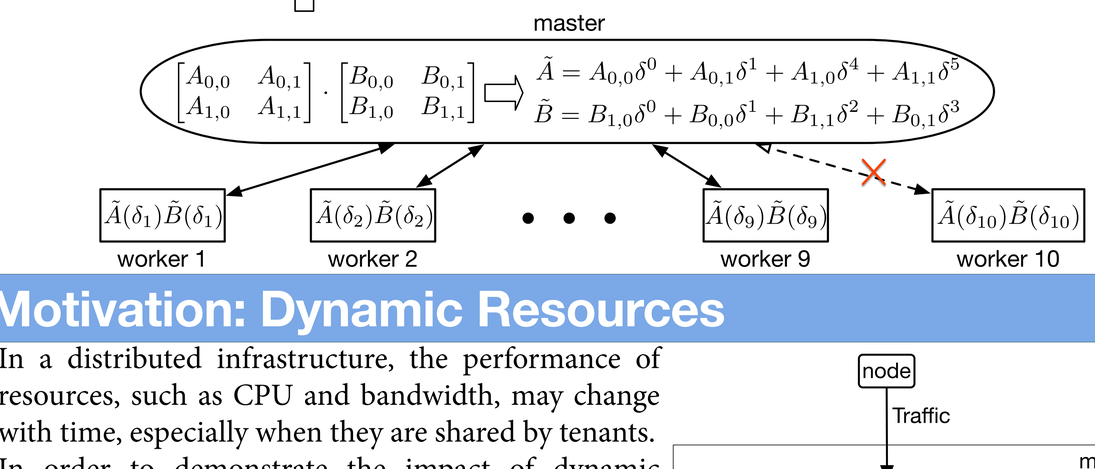
- MatDot Code: Splitting input matrices A and B horizontally and vertically, respectively.



$$\Rightarrow \quad AB=\begin{bmatrix}A_0B_0 & A_1B_1\end{bmatrix}$$

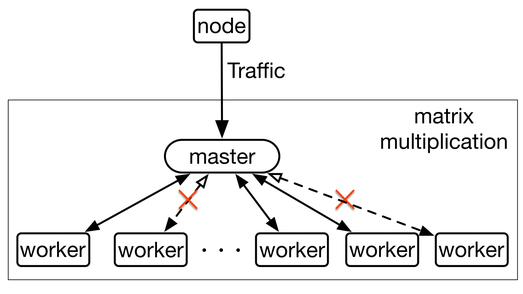
- Entangled Polynomial Code: Dividing input matrices in more dimensions to reduce the size of the task.

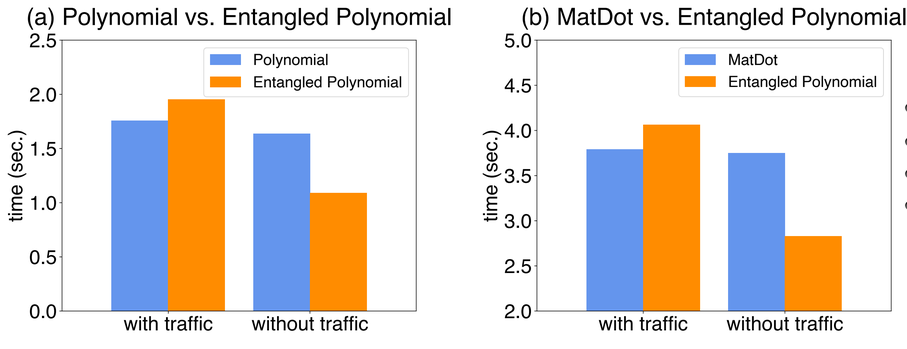
$$AB=\begin{bmatrix}A_{0,0}B_{0,0}+A_{0,1}B_{1,0} & A_{0,0}B_{0,1}+A_{0,1}B_{1,1}\\ A_{1,0}B_{0,0}+A_{1,1}B_{1,0} & A_{1,0}B_{0,1}+A_{1,1}B_{1,1}\end{bmatrix}$$



Motivation: Dynamic Resources

- In a distributed infrastructure, the performance of resources, such as CPU and bandwidth, may change with time, especially when they are shared by tenants.
- In order to demonstrate the impact of dynamic resources on the performance, we run an experiment where we inject traffic from an additional node.
- The experiment results below give a strong evidence of the necessity of changing coding schemes with respects to the dynamic resources.

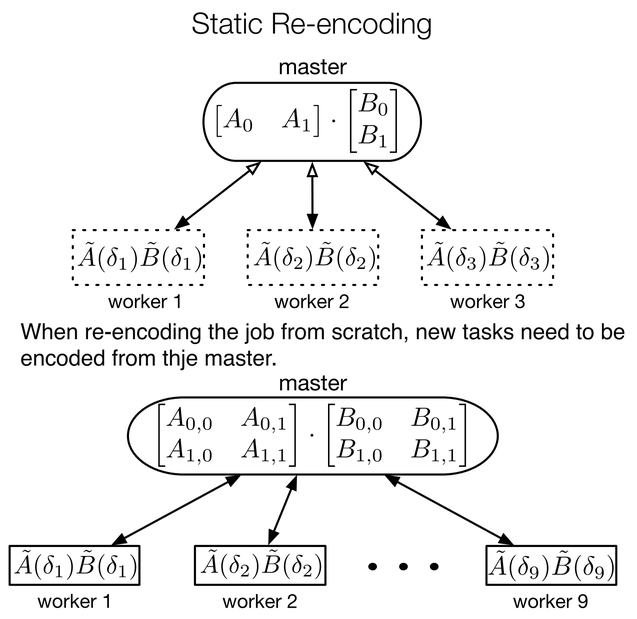


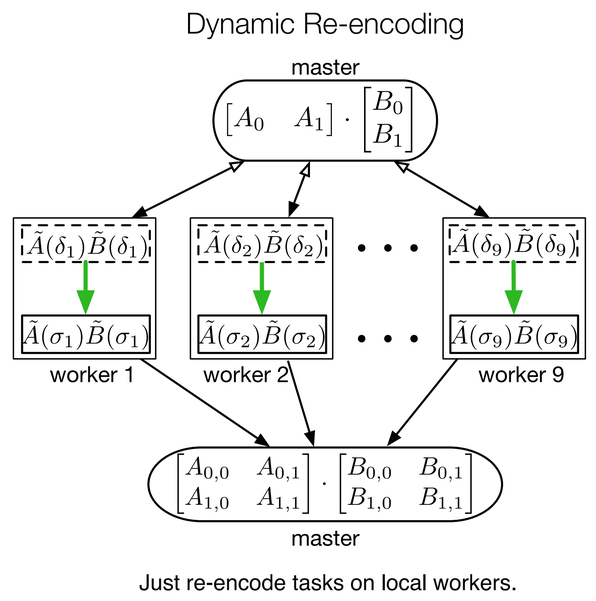


- Size: 4096×4096
- Platform: local cluster
- Data: average of 50 times
- Traffic: 3Gbps

Main Result

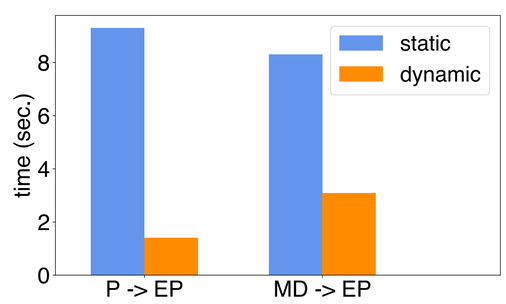
- Instead of re-encoding the job from scratch, we propose a dynamic framework that re-encodes each task locally on each worker without additional data.
- Given a job of A × B, an *(m,n,p)* entangled polynomial code splits A into m and p partitions vertically and horizontally, B into p and n partitions vertically and horizontally, and encode them into $\tilde{A}(\delta)$ and $\tilde{B}(\delta)$, respectively.
  - Polynomial codes ($p=1$) and MatDot codes ($m=n=1$) are special cases of entangled polynomial code.
- By simply re-encoding $\tilde{A}(\delta)$ and $\tilde{B}(\delta)$, our framework can switch coding schemes among them dynamically, as well as the values of their parameters.





Theorem: A task encoded with an (m, n, p) entangled polynomial code can be locally re-encoded into a task encoded with a $(\lambda_m m, \lambda_n n, \lambda_p p)$ entangled polynomial code, where λ_m, λ_n, and λ_p are positive integers.

- We compare the time of our framework of dynamic re-encoding with the time of re-encoding the job from scratch, *i.e.*, static re-encoding.



- Size: 4096×4096
- Platform: local cluster
- Data: average of 50 times
- Traffic: no traffic

- P -> EP: Polynomial Code ($m=2,n=2$) -> Entangled Polynomial Code ($m=2,n=4,p=2$)
- MD -> EP: MatDot Code ($p=2$) -> Entangled Polynomial Code ($m=2,n=1,p=4$)

- The time of re-encoding can be saved by 84.9% and 62.8%, respectively. The saving time mainly comes from the time of encoding and deploying tasks.

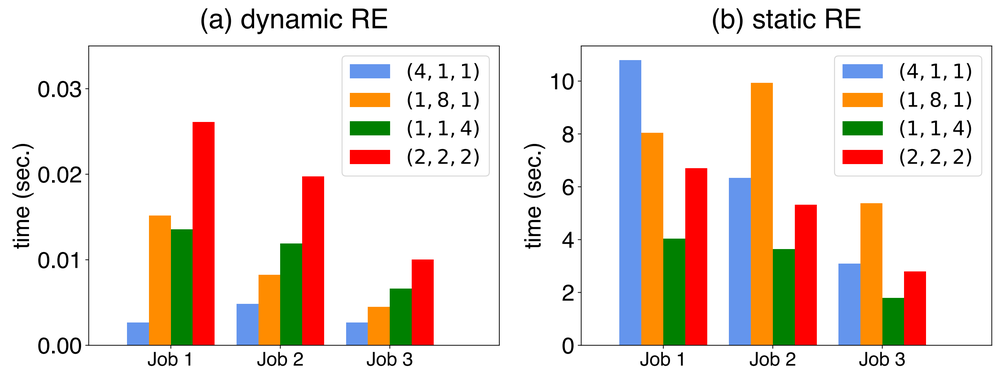
Evaluation

In Amazon EC2, we run the master on a virtual machine of type `t2.xlarge` and all workers on virtual machines of type `t2.small`. We set initial values of *(m, n, p)* as *(2, 2, 2)*, and encode input matrices of three jobs.

	Job 1	Job 2	Job 3
A	1024×2048	2048×2048	2048×1024
B	2048×4096	2048×2048	1024×2048

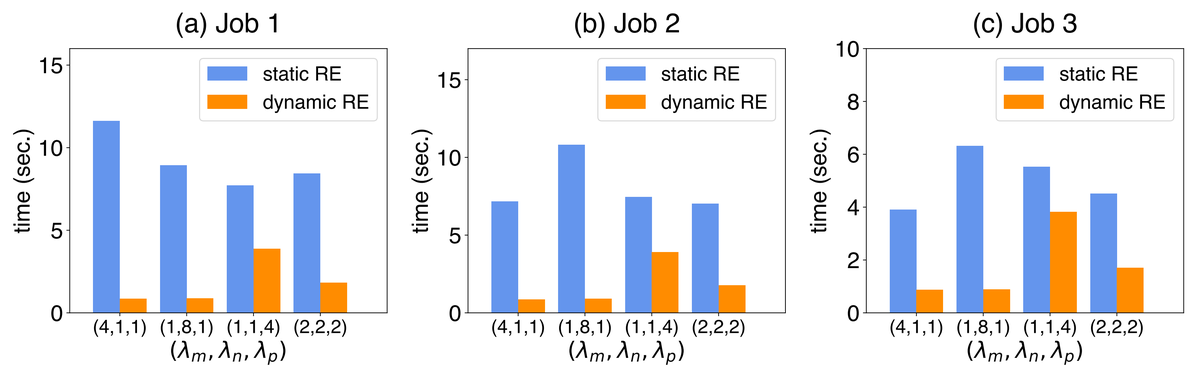
Sizes of input matrices in the three jobs

- Compared to the whole job completion time, the re-encoding (RE) overhead of dynamic re-encode is marginal.



Overhead of re-encoding with different re-encoding schemes

- Due to the saved re-encoding overhead, we can observe that the job completion time can also be saved by up to 92.7%.



Job completion time with re-encoding

This work was supported in part by the National Science Foundation under the grant CCF-1910447 and the "AWS Cloud Credits for Research" program.