Coded Redundancy for Reducing Task Average Latency in Distributed Stream Processing Systems

Yuxuan Lin[1,2], Bin Tang[1], Shengjing Wang[1], Lei Chen[1], Xutong Jiang[1] and Rajkumar Buyya[2]

[1] *Key Laboratory of Water Big Data Technology of Ministry of Water Resources, Hohai University, Nanjing, China*
[1]School of Computer Science and Software Engineering, Hohai University, Nanjing, China
[2]*Quantum Cloud Computing and Distributed Systems (qCLOUDS) Lab*
School of Computing and Information Systems, The University of Melbourne, Australia
Email: {linyx, cstb, shengjing, 23chl, xtjiang}@hhu.edu.cn, rbuyya@unimelb.edu.au

Abstract*—In cloud computing systems, coded computing has proven effective in mitigating stragglers and reducing latency in batch processing scenarios. However, in many applications such as prediction services, tasks arrive continuously as a stream, and it remains unclear whether coding-based redundancy can still provide latency benefits under such streaming workloads. This paper systematically investigates the effectiveness of coded redundancy in reducing average task latency in distributed stream processing systems, where service times are modeled by shifted exponential distributions to capture straggling behavior. Through theoretical analysis and simulations, we compare five representative scheduling strategies, including uncoded, replication-based, and traditional coding-based methods. The results suggest that, under the adopted queueing assumptions, fixed-block batch coding may provide limited latency benefits in streaming scenarios because its coding gain can be offset by batching delay. Based on this observation, we propose a k-periodic inserted coding strategy (KIC), which periodically encodes unfinished tasks and inserts coded tasks to provide more flexible redundancy for streaming workloads. Experimental results show that as the system load increases (e.g., higher task arrival rate λ), the latency of baseline methods increases rapidly, while KIC remains relatively stable, leading to a widening performance gap. Compared with the best baseline, the average latency reduction of KIC increases from 16.7% at $\lambda = 21$ to 98.8% at $\lambda = 35$, indicating that with appropriate design, coding-based redundancy holds promise for latency reduction in distributed stream processing.

***Index Terms*—coded computing, streaming tasks, distributed systems, cloud computing**

I. Introduction

In modern distributed computing systems, such as cloud computing, a critical challenge is the straggler problem, where some computing nodes often experience unpredictable slowdowns or even failures due to hardware malfunctions, resource contention, network fluctuations, and other system-level factors [1], [2]. Because tasks are typically divided into subtasks and distributed across multiple workers, the overall job completion time is often determined by the slowest node. As a result, even a single straggler can significantly delay task completion and degrade system performance. Therefore, mitigating stragglers is essential to improve the efficiency and reliability of distributed systems.

Redundancy has proven to be an effective way to reduce the impact of stragglers by enabling task completion without waiting for all workers. Among various redundancy strategies, coding-based redundancy has received particular attention due to its high fault tolerance with lower overhead. By encoding tasks and distributing them to multiple workers, the system can reconstruct the final result from a subset of completed outputs, thus mitigating delays caused by slow or failed nodes. Early studies demonstrated the benefits of applying erasure codes to distributed matrix multiplication tasks [3]. Since then, coding techniques have been extended to a wide range of distributed computing workloads, including gradient descent [4]–[6], multivariate polynomial evaluation [7], [8], and deep neural network (DNN) inference tasks [9]–[12].

Most existing coded computing strategies are developed under the assumption of static or batch-based workloads, where the full set of tasks is known beforehand and can be jointly encoded. However, many real-world distributed systems operate under streaming workloads, where tasks arrive continuously and unpredictably rather than in a batch. This streaming characteristic is common in real-time prediction services such as recommendation systems [13], video stream analysis [14], anomaly detection [15], and interactive applications. Compared to batch processing, streaming tasks require immediate responses under high concurrency, making the system more sensitive to performance fluctuations.

In such streaming settings, stragglers not only slow down individual tasks but can also block the processing of subsequent ones, leading to cascading delays and degraded overall system responsiveness. These characteristics pose significant challenges for applying traditional coding strategies, which typically rely on static task structures and global coordination. As a result, it remains unclear whether coding-based redundancy can still provide latency benefits in streaming systems. This uncertainty motivates our study to systematically investigate the effectiveness of coded redundancy in reducing average task latency under stream processing systems.

This work is partially supported by the National Key R&D Program of China under Grant 2023YFC3006505, Jiangsu Provincial Special Project for the Transformation of Scientific and Technological Achievements under Grant BA2023030, Key Projects of Jiangsu Provincial Basic Research Program under Grant BK20253011, and the China Scholarship Council (funding for PhD research studies in the University of Melbourne).

Bin Tang and Rajkumar Buyya are the corresponding authors.

In this paper, we investigate this open problem assuming that task execution times follow a shifted exponential distribution, which is commonly used to model stragglers. We first analyze five representative scheduling strategies via queuing theory and observe that, under streaming workloads, traditional fixed-block batch coding tends to degenerate into replication at its optimum, indicating that directly applying existing batch-oriented coding designs may provide limited latency benefits in such scenarios. Motivated by this insight, we propose a k-periodic inserted coding (KIC) method as an exploratory inter-task coding design, which demonstrates promising performance under high-load conditions. The main contributions of this paper are:

- We conduct systematic study on the effectiveness of coding-based redundancy in reducing average task latency under streaming task arrivals. By constructing queuing-theoretic models, we analyze and compare the latency performance of five representative scheduling strategies, including uncoded, replication-based, and coding-based methods. Our analysis reveals that traditional fixed-block batch coding may not outperform replication-based grouping strategies in terms of average latency reduction for streaming tasks.
- Based on the insights from our analytical comparison, we propose a k-periodic inserted coding strategy (KIC) as an exploratory attempt to adapt coding techniques to streaming scenarios. KIC periodically inserts a coded task constructed from currently unfinished tasks, helping to accelerate the slowest one and reducing average latency with minimal redundancy. Rather than being a definitive solution, KIC serves to illustrate the potential of more adaptive and dynamic coding designs.
- We conduct discrete-event simulations to evaluate classical scheduling methods and KIC under different system conditions. Experimental results demonstrate that under high or overloaded system conditions, the proposed KIC strategy consistently outperforms five classical scheduling methods. These findings indicate that, with proper design, coding-based redundancy has strong potential to improve the performance of streaming prediction services.

The rest of the paper is organized as follows. Sec. II discusses related work. Sec. III introduces the system model and system assumptions. Sec. IV provides latency modeling and analysis for five conventional task scheduling methods, while Sec. V validates this analysis and compares their performance. Section VI proposes our newly developed coding strategy, KIC, and Section VII evaluates its performance compared to the other five methods. Finally, Sec. VIII concludes the paper with future directions.

II. Related Work

To ensure low-latency performance in distributed computing, many studies have explored strategies to mitigate the impact of stragglers. Among them, redundancy-based approaches have gained wide adoption due to their simplicity and effectiveness. A typical example is replication [16]–[18], where multiple copies of a task are executed in parallel to reduce the risk of delays caused by stragglers. Another widely studied method is coding-based redundancy, such as Maximum Distance Separable (MDS) codes [3], which splits a task into k parts and encodes them into n pieces, allowing recovery from any k completed results. With higher fault tolerance and lower overhead, coding-based redundancy has been widely applied in distributed computing systems [19]–[23]. However, many existing coding-based redundancy schemes are designed for batch tasks that arrive all at once, without taking into account the continuous arrival of streaming tasks [24]–[27].

For the scheduling problem in streaming tasks, some studies have attempted to introduce coding techniques to alleviate the stragglers and reduce task latency. Cohen et al. [28] proposed a joint scheduling and coding optimization framework for streaming tasks to improve distributed computing efficiency. Building on this, Esfahanizadeh et al. [29] further enhanced the framework by selecting optimal worker subsets and load distribution for each arriving task. Cheng et al. [30] propose an erasure coding-based framework to enable low-redundancy fault tolerance in stream machine learning systems. Although these methods have advanced coding-based scheduling, they still focus on partitioning and encoding individual tasks within the stream. However, many practical tasks are inherently indivisible and cannot be further partitioned, such as inference tasks. In addition, these works do not explicitly model the stochastic arrival process of tasks, limiting their ability to characterize system performance under dynamic workloads.

Different from the above works, Ji et al. [31] explicitly modeled the dynamic arrival process of streaming tasks as a Poisson process and analyzed the end-to-end delay performance of coded computing under this realistic setting. It derived tight upper and lower delay bounds for both purging and non-purging coding schemes, demonstrating significant coding gains even when task queuing is considered. However, this work focuses on encoding by partitioning and coding individual tasks, without considering coordination among multiple concurrent tasks, and only compares coded methods with uncoded ones, without evaluating other scheduling strategies.

In summary, existing studies on coded computing have primarily focused on static or batch settings, where coding is applied within individual tasks to mitigate stragglers. In contrast, this paper explores inter-task coding strategies tailored for streaming task arrivals, aiming to exploit redundancy across multiple tasks rather than within a single one. Based on this perspective, we systematically investigate the effectiveness of coding-based redundancy under dynamic and continuous workloads in stream processing systems. The key differences are summarized in Table I.

III. System Model

We begin by describing the system model, followed by an illustrative example as shown in Fig. 1. The distributed computing system consists of a master node and n homogeneous worker nodes denoted by $\mathcal{N} = \{1, 2, \ldots, n\}$, and is responsible for performing computation tasks $g_{\mathbf{A}}(\cdot)$ where $\mathbf{A}$

TABLE I
COMPARISON OF EXISTING WORKS AND OUR WORK.

Considerations	Related Work					This Work
	[9], [10]	[16]–[18]	[19]–[27]	[28]–[30]	[31]	
Stream processing environment	×	✓	×	✓	✓	✓
Coded computing	✓	×	✓	✓	✓	✓
Inter-task coding	✓	×	×	×	✓	✓
Analytical characterization of average task latency	×	✓	×	×	✓	✓
Comparison with other redundancy schemes (e.g., replication, other coding schemes)	×	✓	✓	✓	×	✓

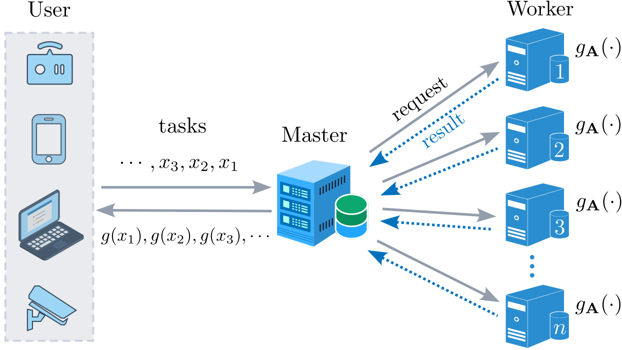


Fig. 1. An illustration of the distributed stream processing system.

referse to some local data blocks or machine learning models. Specifically, the master node receives the task requests, each represented by x, and needs to compute the $g_{\mathbf{A}}(x)$, where $\mathbf{A}$ is prestored on each worker node to enable distributed computing. The task requests are continuously arriving at the system. Following prior studies like [28], [29], [31], we assume that the task arrival process is a Poisson process with the average arrival rate λ. Denote T_a as the inter-arrival time between two consecutive tasks, then T_a follows an exponential distribution with parameter λ, with the cumulative distribution function (CDF) as

$$P(T_a \leq t) = 1 - e^{-\lambda t}, \text{ if } t \geq 0. \tag{1}$$

For analytical tractability, we assume homogeneous workers, a common simplification in theoretical studies of coded computing and redundancy-based latency analysis [3]. Following common straggler modeling practice [3], [4], [32], the computation time on each worker is modeled as a shifted exponential random variable with rate parameter μ and shift parameter α, denoted by T_s. Specifically, the CDF of T_s is given by

$$P(T_s \leq t) = \begin{cases} 1 - e^{-\mu(t-\alpha)}, & \text{if } t \geq \alpha, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

In this work, we aim to minimize average task latency through scheduling design. Since the service-time parameters may vary across system scenarios, we characterize how different scheduling strategies perform under different values of the arrival rate, service rate, and shift parameter. In particular, we focus on whether coded redundancy can effectively reduce average task latency in streaming systems.

IV. LATENCY ANALYSIS FOR CONVENTIONAL METHODS

In this section, we analyze the average latency of five typical scheduling methods under streaming task arrivals.

A. First-Come First-Served Method (FCFS)

In this method, when a task arrives at the system, it will first enter a first-in-first-out (FIFO) queue maintained by the master node. Once there exists some worker node idle, the first task of the queue will be assigned to the worker node.

Theorem 1. *The execution process of the FCFS system can be modeled as an M/G/n queue with arrival rate λ and n homogeneous worker nodes acting as servers. Under this M/G/n model, when the system is in a stable state, the average latency of tasks scheduled by the FCFS strategy is:*

$$\mathbb{T}^{FCFS} = \frac{1}{\mu} + \alpha + \frac{1}{(\frac{n\mu}{1+\mu\alpha} - \lambda)} \times \frac{(1/\alpha\mu + 1)^2 + 1}{2\left(1 + (1 - \frac{\lambda(1+\mu\alpha)}{n\mu})\left(\frac{n!}{(\frac{\lambda(1+\mu\alpha)}{\mu})^n}\right)\sum_{k=0}^{n-1}\frac{(\frac{\lambda(1+\mu\alpha)}{\mu})^k}{k!}\right)}$$

Proof. Since the service time T_s^{FCFS} follows a shifted exponential distribution, its expected value and variance are given by:

$$\mathbb{E}[T_s^{FCFS}] = \frac{1}{\mu} + \alpha, \qquad \mathbf{Var}(T_s^{FCFS}) = \frac{1}{\mu^2}.$$

The server utilization ρ^{FCFS} represents the proportion of time each server is occupied, and is defined as the ratio between the task arrival rate and the total service rate. Specifically, we have:

$$\rho^{FCFS} = \frac{\lambda}{n(1/\mathbb{E}[T_s^{FCFS}])} = \frac{\lambda(1+\mu\alpha)}{n\mu}.$$

Lemma 2. *In a M/G/n queuing system, if $\rho < 1$, then the queue is stable, and the average latency, which is the average time a task spends in the system, is*

$$\mathbb{T}^{M/G/n} = \mathbb{E}[T_q^{M/G/n}] + \mathbb{E}[T_s^{M/G/n}],$$

where $\mathbb{E}[T_s^{M/G/n}]$ is the average service time, $\mathbb{E}[T_q^{M/G/n}]$ is the average waiting time, and according to [33]:

$$\mathbb{E}[T_q^{M/G/n}] = \frac{c^2+1}{2}\mathbb{E}[T_q^{M/M/n}],$$

where $\mathbb{E}[T_q^{M/M/n}]$ is the average waiting time of M/M/n model, and c is the the coefficient of variation (that is the standard deviation of times divided by the mean time) for service times.

Lemma 3 ([34]). *In a M/M/n queuing system, the average waiting time is*

$$\mathbb{E}[T_q^{M/M/n}] = \frac{C(n,\rho)}{n\mu' - \lambda'},$$

where ρ denotes the system utilization, with λ' and μ' representing average task arrival and service rates of the system, respectively. The $C(n,\rho)$ is the probability that an arriving customer is forced to join the queue (all servers are occupied), and it is given by

$$C(n,\rho) = \frac{1}{1 + (1-\rho)\left(\frac{n!}{(n\rho)^n}\right)\sum_{k=0}^{n-1}\frac{(n\rho)^k}{k!}}.$$

According to Lemma 2 and Lemm 3, when $\rho < 1$, the average queuing time can be computed by:

$$c^{FCFS} = \frac{\sqrt{\mathbf{Var}\,(T_s^{FCFS})}}{\mathbb{E}[T_s^{FCFS}]} = \frac{1}{\alpha\mu + 1},$$

$$C(n,\rho^{FCFS}) = \frac{1}{1 + (1 - \frac{\lambda(1+\mu\alpha)}{n\mu})\left(\frac{n!}{(\frac{\lambda(1+\mu\alpha)}{\mu})^n}\right)\sum_{k=0}^{n-1}\frac{(\frac{\lambda(1+\mu\alpha)}{\mu})^k}{k!}},$$

$$\mathbb{E}[T_q^{FCFS}] = \frac{(c^{FCFS})^2 + 1}{2} \cdot \frac{C(n,\rho^{FCFS})}{n/\mathbb{E}[T_s^{FCFS}] - \lambda}.$$

Finally, summing the service time and the waiting time yields the result in Theorem 1:

$$\mathbb{T}^{FCFS} = \mathbb{E}[T_q^{FCFS}] + \mathbb{E}[T_s^{FCFS}].$$

The proof is completed. □

B. Full Replication Method (FR)

In this method, when a task arrives at the system, it will be replicated into n copies and distributed to n worker nodes for execution. Once any one of the nodes completes the task, all other nodes immediately cancel their execution. If the previous tasks have not yet been completed, the subsequently arriving tasks will enter a FIFO queue maintained by the master node and wait for execution.

Theorem 4. *The execution process of a group can be modeled as a M/G/1 model, and when the system is in a stable state, the average latency of tasks scheduled by the FR strategy is:*

$$\mathbb{T}^{FR} = \frac{\frac{\lambda(1+\alpha\mu n)}{\mu n} + \frac{\lambda}{(1+\mu n\alpha)\mu n}}{2(\frac{\mu n}{1+\mu n\alpha} - \lambda)} + \frac{1}{\mu n} + \alpha.$$

Proof. The task arrival follows the Poisson distribution with parameter λ as defined in formula (1), and the average service time corresponds to the completion time of the fastest node among those assigned to execute the task. Specifically, the service time of node i, denoted by T_i^{FR}, follows a shifted exponential distribution as defined in formula (2). Therefore, the computation time can be characterized as the first order statistic of n i.i.d. shifted exponential distributed random variables, denoted by $T_{(1)}^{FR}$, whose distribution is given by:

$$F_{T_{(1)}^{FR}}(t) = 1 - P(T_i^{FR} > t)^n = 1 - e^{-\mu n(t-\alpha)}, \text{ if } t \geq \alpha.$$

Thus, $T^{FR}(1)$ follows a shifted exponential distribution with rate parameter μn and shift parameter α. The expectation and variance of $T^{FR}(1)$ are given by:

$$\mathbb{E}[T_{(1)}^{FR}] = \frac{1}{\mu n} + \alpha, \qquad \mathbf{Var}\left(T_{(1)}^{FR}\right) = (\frac{1}{\mu n})^2.$$

The server utilization of this system is

$$\rho^{FR} = \frac{\lambda}{(1/\mathbb{E}[T_{(1)}^{FR}])} = \frac{\lambda(1+\alpha\mu n)}{\mu n}.$$

The mean service rate in this method is given by $\mu' = \frac{\mu n}{1+\mu n\alpha}$. By substituting the above expressions into the Pollaczek–Khinchine formula [35], [36], we obtain the result presented in Theorem 4, which completes the proof. □

C. Group-based Replication Method (GR)

The system divides the worker nodes into l groups, under the assumption that n is divisible by l, with each group containing $\frac{n}{l}$ nodes. When a task arrives, it first enters a FIFO queue managed by the master node and waits for computing. If there is at least one idle group of nodes, the task at the front of the queue will be replicated into $\frac{n}{l}$ copies and sent to that group for execution. As soon as any node in the group completes the computation, the result will be returned to the master node, and all remaining replicas within the same group will be immediately canceled.

Theorem 5. *The execution process of the system can be modeled as a M/G/l model. When the system is in a stable state, the average latency of tasks scheduled by the GR strategy is:*

$$\mathbb{T}^{GR} = \frac{1}{2(\frac{l\mu n}{l+\alpha\mu n} - \lambda)} + \frac{l}{\mu n} + \alpha \times \frac{(\frac{l}{\alpha\mu n + l})^2 + 1}{\left(1 + (1 - \frac{\lambda(l+\alpha\mu n)}{l\mu n})\left(\frac{l!}{(\frac{\lambda(l+\alpha\mu n)}{\mu n})^l}\right)\sum_{k=0}^{l-1}\frac{(\frac{\lambda(l+\alpha\mu n)}{\mu n})^k}{k!}\right)}.$$

Proof. By dividing the n nodes into l groups, each containing $\frac{n}{l}$ servers, and treating each group as a single server, the system can be modeled as an M/G/l queue. The task arrival follows a Poisson distribution with parameter λ as shown in formula (1), and the average service time of each task refers to the service time of the fastest completing node in a group. Let the service time of node i be denoted as T_i^{GR}. Similar to the FR method, the computation time of each group corresponds to the first order statistic of $\frac{n}{l}$ i.i.d. shifted exponential random variables. We denote this time as $T_{(1)}^{GR}$, which follows the distribution:

$$F_{T_{(1)}^{GR}}(t) = 1 - P(T_i^{GR} > t)^{n/l} = 1 - e^{-\frac{\mu n}{l}(t-\alpha)}, \text{ if } t \geq \alpha.$$

The expectation and the variance of $T_{(1)}^{GR}$ are

$$\mathbb{E}[T_{(1)}^{GR}] = \frac{l}{\mu n} + \alpha, \qquad \textbf{Var}\left(T_{(1)}^{GR}\right) = (\frac{l}{\mu n})^2.$$

Accordingly, the server utilization of the system is

$$\rho^{GR} = \lambda \cdot \mathbb{E}[T_{(1)}^{GR}]/l = \frac{\lambda(l + \alpha\mu n)}{l\mu n}.$$

Based on Lemma 2 and Lemma 3, when the system remains stable (i.e., $\rho^{GR} < 1$), the average waiting time in the M/G/l queue can be estimated using the coefficient of variation $c^{GR} = \frac{l}{\alpha\mu n + l}$ and the Erlang C formula. Letting $C(l, \rho^{GR})$ denote the corresponding correction term, the average waiting time is given by:

$$\mathbb{E}[T_q^{GR}] = \frac{\left((c^{GR})^2 + 1\right) C(l, \rho^{GR})}{2(\frac{l}{\mathbb{E}[T_{(1)}^{GR}]} - \lambda)}.$$

Adding the expected service time $\mathbb{E}[T_{(1)}^{GR}]$ yields the average task latency $\mathbb{T}^{GR}$ in Theorem 5. The proof is completed. □

D. Batch MDS Coding Method (MDS)

In this method, the system operates in a batch processing manner. Initially, it waits for the arrival of k ($k \leq n$) tasks, groups them into a batch, and encodes them into n subtasks using an (n, k) MDS code. These subtasks are then processed concurrently by n worker nodes. In each batch, as long as any k of the n subtasks are completed, the batch can be decoded and considered finished. The system processes tasks in a blocking manner: all tasks within a batch start their computation simultaneously, and the next batch will only begin processing once the current batch has been fully completed. Prior studies [3], [19]–[22], [31], [37]–[39] have commonly assumed that the encoding and decoding overhead is negligible compared to the computation time. Following this line of work, we do not explicitly consider the encoding and decoding latency in this paper.

Theorem 6. *The execution process of a batch can be modeled as a G/G/1 model. When the system is in a stable state, the average latency of tasks scheduled by the MDS strategy is:*

$$\mathbb{T}^{MDS} \approx \frac{k-1}{2\lambda} + \frac{\lambda\left((H_\Delta + \mu\alpha)^2 + kH_{\Delta,2}\right)}{2k\mu\left(k\mu - \lambda(H_\Delta + \mu\alpha)\right)} + \frac{H_\Delta + \mu\alpha}{\mu}.$$

where $H_\Delta \triangleq H_n - H_{n-k}$, and $H_{\Delta,2} \triangleq H_{n,2} - H_{n-k,2}$. Here, $H_j = \sum_{i=1}^{j} \frac{1}{i}$ denotes the j-th harmonic number, and the $H_{j,2} = \sum_{i=1}^{j} \frac{1}{i^2}$ denotes the corresponding generalized harmonic number of order 2.

Proof. Tasks are processed in batches, and a batch is only considered complete when k out of n subtasks finish. Thus, the entire system can be abstracted as a single server that processes one batch at a time. Tasks arrive according to a Poisson process with rate λ. Since each batch contains k tasks, the batch inter-arrival time T_a^{MDS} follows an Erlang distribution with shape k and rate λ, corresponding to the sum of k i.i.d. exponential random variables. As a result, we can get

$$P(T_a^{MDS} \leq t) = 1 - \sum_{i=0}^{k-1} \frac{(\lambda t)^i e^{-\lambda t}}{i!}, \quad t \geq 0. \tag{3}$$

Therefore, the expectation and variance of the batch inter-arrival time are given as follows.

$$\mathbb{E}[T_a^{MDS}] = \frac{k}{\lambda}, \qquad \textbf{Var}\left(T_a^{MDS}\right) = \frac{k}{\lambda^2}. \tag{4}$$

The service time T_i^{MDS} of each node i follows the shifted exponential distribution as shown in formula (2), so the service time for a batch can be viewed as the k-th order statistic of n shifted exponential distributed random variables, denoted by $T_{(k)}^{MDS}$, which follows the distribution [40] :

$$F_{T_{(k)}^{MDS}}(t) = \sum_{i=k}^{n} \binom{n}{i} [1 - e^{-\mu(t-\alpha)}]^i [e^{-\mu(t-\alpha)}]^{n-i}, t \geq \alpha. \tag{5}$$

The expectation and variance can be calculated as:

$$\mathbb{E}[T_{(k)}^{MDS}] = \frac{(H_n - H_{n-k})}{\mu} + \alpha. \tag{6}$$

$$\textbf{Var}\left(T_{(k)}^{MDS}\right) = \frac{(H_{n,2} - H_{n-k,2})}{\mu^2}. \tag{7}$$

The server utilization of this system is

$$\rho^{MDS} = \frac{1/\mathbb{E}[T_a^{MDS}]}{1/\mathbb{E}[T_{(k)}^{MDS}]} = \frac{\lambda(H_n - H_{n-k} + \mu\alpha)}{k\mu}.$$

Lemma 7 ([41]). *In a G/G/1 queuing system, if $\rho < 1$, then the queue is stable, and the average latency is*

$$\mathbb{T}^{G/G/1} = \mathbb{E}[T_q^{G/G/1}] + \mathbb{E}[T_s^{G/G/1}],$$

where $\mathbb{E}[T_s^{G/G/1}]$ is the average service time, $\mathbb{E}[T_q^{G/G/1}]$ is the average waiting time, and according to Kingman's formula [42], an approximation for the average waiting time is:

$$\mathbb{E}[T_q^{G/G/1}] \approx \left(\frac{\rho}{1-\rho}\right)\left(\frac{c_a^2 + c_s^2}{2}\right)\mathbb{E}[T_s^{G/G/1}],$$

where ρ is the system utilization, c_a is the coefficient of variation for arrivals, and c_s is the coefficient of variation for service times.

According to Lemma 7, the coefficients of variation for the batch inter-arrival and service times are

$$c_a^{MDS} = \frac{1}{\sqrt{k}}, \quad c_s^{MDS} = \frac{\sqrt{H_{n,2} - H_{n-k,2}}}{H_n - H_{n-k} + \mu\alpha}.$$

Using these, the average waiting time approximates to

$$\mathbb{E}[T_q^{MDS}] \approx \frac{\lambda\left((H_\Delta + \mu\alpha)^2 + kH_{\Delta,2}\right)}{2k\mu\left(k\mu - \lambda(H_\Delta + \mu\alpha)\right)}.$$

Based on the above, the average latency $\mathbb{T}_{batch}^{MDS}$ for a batch is

$$\mathbb{T}_{batch}^{MDS} = \mathbb{E}[T_q^{MDS}] + \mathbb{E}[T_{(k)}^{MDS}].$$

Compared to the latency of the entire batch, the average latency for each task includes an additional component: the average waiting time for the batch to be formed. Suppose that $\mathcal{K} = \{1, \ldots, k\}$, and the average waiting time for task j (where $j \in \mathcal{K}$) is denoted as T^j_{wait} , measured from its arrival until the arrival of task k. It is evident that $T^j_{\text{wait}} = \frac{k-j}{\lambda}$. Therefore, the average latency of this method is

$$\mathbb{T}^{MDS} = \frac{1}{k}\sum_{j=1}^{k} T^j_{\text{wait}} + \mathbb{T}^{MDS}_{batch}.$$

Substituting the above results, we can derive Theorem 6. The proof is completed. □

E. Group-based MDS Coding Method (GMDS)

This approach, like the grouping replication method, divides the n nodes into l groups, each containing $\frac{n}{l}$ nodes. Within each group, tasks are processed using the batch MDS coding method described before. Specifically, the system waits for k tasks (where $k \leq \frac{n}{l}$) to form a batch, which is encoded via an $(\frac{n}{l}, k)$ MDS code into $\frac{n}{l}$ coded tasks. When a group becomes idle, the encoded tasks are assigned for parallel execution. A batch completes once any k out of the $\frac{n}{l}$ nodes in the group finish their tasks, and the remaining tasks in that group are immediately canceled. The system follows a blocking execution policy, similar to the batch MDS coding method, where a new batch does not begin execution until all nodes in the previous group become idle. Similar to batch MDS-based methods, we assume that the encoding and decoding overhead is negligible in our analysis.

Theorem 8. *The execution process of this method can be modeled as a G/G/l model. When the system is in a stable state, the average latency of tasks scheduled by the GMDS strategy is:*

$$\mathbb{T}^{GMDS} \approx \frac{\Phi^2 + kH_{\Delta',2}}{\left(1 + (1 - \frac{\lambda\Phi}{kl\mu})\left(\frac{l!}{(\frac{\lambda\Phi}{k\mu})^l}\right)\sum_{i=0}^{l-1}\frac{(\frac{\lambda\Phi}{k\mu})^i}{i!}\right)} \times \frac{1}{2\Phi\,(kl\mu - \lambda\Phi)} + \frac{\Phi}{\mu} + \frac{k-1}{2\lambda}.$$

where $\Phi \triangleq H_{\frac{n}{l}} - H_{\frac{n}{l}-k} + \mu\alpha$*, and* $H_{\Delta',2} \triangleq H_{\frac{n}{l},2} - H_{\frac{n}{l}-k,2}$.

Proof. Similar to the GR strategy, the system is divided into l groups, each acting as an independent server. Within each group, task execution follows the batch MDS coding method, where batches of k tasks arrive according to an Erlang distribution with shape k and rate λ. We denote the inter-arrival time as T^{GMDS}_a, which follows the Erlang distribution in (3), with its expectation and variance given in (4). The service time corresponds to the k-th order statistic of $\frac{n}{l}$ i.i.d. shifted exponential random variables, denoted as $T^{GMDS}_{(k)}$. According to formula (6) and (7), its expected value and variance are given by:

$$\mathbb{E}[T^{GMDS}_{(k)}] = \frac{(H_{\frac{n}{l}} - H_{\frac{n}{l}-k})}{\mu} + \alpha,$$

$$\textbf{Var}\left(T^{GMDS}_{(k)}\right) = \frac{(H_{\frac{n}{l},2} - H_{\frac{n}{l}-k,2})}{\mu^2}.$$

The server utilization of this system is

$$\rho^{GMDS} = \frac{1/\mathbb{E}[T^{GMDS}_a]}{l/\mathbb{E}[T^{GMDS}_{(k)}]} = \frac{\lambda(H_{\frac{n}{l}} - H_{\frac{n}{l}-k} + \mu\alpha)}{kl\mu}.$$

Lemma 9. *In a G/G/l queuing system, if $\rho < 1$, then the queue is stable, and the average latency is*

$$\mathbb{T}^{G/G/l} = \mathbb{E}[T^{G/G/l}_q] + \mathbb{E}[T^{G/G/l}_s],$$

where $\mathbb{E}[T^{G/G/l}_s]$ is the average service time and $\mathbb{E}[T^{G/G/l}_q]$ is the average waiting time. Using the approximation in formula (2.14) of [43], the average waiting time can be approximated as follows.

$$\mathbb{E}[T^{G/G/l}_q] \approx \left(\frac{c_a^2 + c_s^2}{2}\right)\mathbb{E}[T^{M/M/l}_q],$$

where c_a is the coefficient of variation for arrivals, and c_s is the coefficient of variation for service times.

Based on Lemma 3 and Lemma 9, the average waiting time $\mathbb{E}[T^{GMDS}_q]$ of this system can be calculated by:

$$c^{GMDS}_a = \frac{1}{\sqrt{k}}, \quad c^{GMDS}_s = \frac{\sqrt{H_{\frac{n}{l},2} - H_{\frac{n}{l}-k,2}}}{H_{\frac{n}{l}} - H_{\frac{n}{l}-k} + \mu\alpha},$$

$$\mathbb{E}[T^{GMDS}_q] \approx \left(\frac{(c^{GMDS}_a)^2 + (c^{GMDS}_s)^2}{2}\right)\mathbb{E}[T^{M/M/l}_q]$$
$$= \frac{\Phi^2 + kH_{\Delta',2}}{2\Phi\,(kl\mu - \lambda\Phi)\left(1 + (1 - \frac{\lambda\Phi}{kl\mu})\left(\frac{l!}{(\frac{\lambda\Phi}{k\mu})^l}\right)\sum_{i=0}^{l-1}\frac{(\frac{\lambda\Phi}{k\mu})^i}{i!}\right)}.$$

Accordingly, the average latency $\mathbb{T}^{GMDS}_{batch}$ for a batch is

$$\mathbb{T}^{GMDS}_{batch} = \mathbb{E}[T^{GMDS}_q] + \mathbb{E}[T^{GMDS}_{(k)}].$$

Like the batch MDS coding method, this method also includes the average batch formation delay. Therefore, its average latency is:

$$\mathbb{T}^{GMDS} = \frac{k-1}{2\lambda} + \mathbb{T}^{GMDS}_{batch}.$$

Similarly, substituting the above results yields Theorem 8, which completes the proof.

□

V. Comparison and Characterization of Conventional Methods

In this section, we first perform simulation experiments and numerical calculations for the five conventional scheduling methods to assess the accuracy of their theoretical latency models. We then analyze the transformation relationships among these strategies and focus on evaluating the performance of coding-based methods in reducing the average latency of streaming tasks.

A. Simulation Verification and Numerical Accuracy

We compare simulation results with theoretical calculations to evaluate the accuracy of the latency models for each method under different system settings. The five methods are modeled using different queuing systems to characterize their average latency. Specifically, we adjust system parameters to align the utilization levels across all methods, and then evaluate the relative error between the theoretical and simulated average latency at different utilization points ranging from 0 to 1. Overall, the results show good agreement, with most deviations remaining below 6%, confirming the effectiveness of our theoretical analysis.

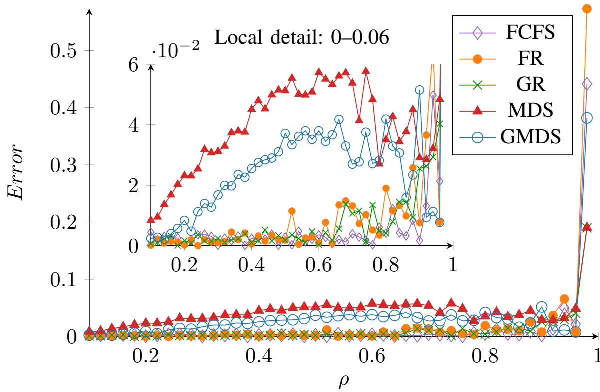


Fig. 2. The corresponding system utilization rate under each method

In terms of numerical accuracy, the M/G-based models for FCFS, FR, and GR demonstrate high precision, with relative errors typically below 2%. This accuracy benefits from the close match between their modeling assumptions and simulation settings. In contrast, the G/G/1 and G/G/l models used for MDS and GMDS exhibit higher errors—though still within 6%—due to the difficulty of analytically capturing non-Poisson arrivals and variable service times induced by coding redundancy.

We also observe distinct trends in model accuracy as system utilization increases. For the M/G-based models, the relative error stays low and stable at low utilization levels, but starts to fluctuate with rising utilization and rises sharply as the system approaches saturation. This is mainly because queueing delays become more sensitive to small variations in parameters under heavy load. In contrast, the G/G-based models exhibit a non-monotonic error pattern. As utilization increases, the error initially grows, then decreases slightly at higher utilization levels, likely because redundancy helps smooth out traffic burstiness. However, the error rises sharply again when utilization approaches saturation, where queue instability amplifies the mismatch between theoretical assumptions and actual behavior. Despite these fluctuations, the overall error of the G/G-based models remains low across most operating conditions, demonstrating the effectiveness of the analytical approximations. These findings demonstrate the reliability of the proposed analytical models.

B. Transition Relationships Among Scheduling Methods

We observe that the five scheduling strategies discussed earlier are not entirely independent, as they can be transformed into each other under specific parameter settings. This section outlines the intrinsic relationships among FCFS, FR, GR, MDS, and GMDS by analyzing their behavior in special cases. These connections help clarify the roles of the key parameters (e.g., l and k) in controlling the structure and degree of redundancy, and how different parameter choices lead to distinct scheduling behaviors. The relationships are illustrated in Table. II.

TABLE II
THE CONVERSION RELATIONSHIPS AMONG THE FIVE METHODS

Variable / Method	**GR**	**MDS**	**GMDS**
$l=1$	=FR	\	=MDS
$l=n$	=FCFS	\	=FCFS
$k=1$	\	=FR	=GR
$k=n$	\	\	\

For the Group-based Replication (GR) strategy, when the number of groups $l = 1$, all nodes belong to a single group. In this case, each task is replicated to all nodes, making GR equivalent to FR. Conversely, when $l = n$, each node forms its own group without redundancy across groups, reducing GR to the FCFS strategy without any redundancy.

In the case of the batch MDS coding (MDS) strategy, when the parameter $k = 1$, the system no longer needs to wait for multiple tasks to form a batch. Instead, each individual task is encoded into n coded fragments, which is functionally equivalent to generating n replicas of the task. As a result, the batch MDS strategy is equivalent to the FR strategy.

The Group-based MDS coding (GMDS) strategy, as a flexible framework with two tunable parameters l and k, can transform into the other four strategies under specific configurations. When $l = 1$, all nodes belong to one group, and the strategy becomes equivalent to the MDS strategy, with k as the only adjustable parameter. When $l = 1$ and $k = 1$, it coincides with the FR strategy, consistent with the special case of MDS. When $k = 1$, the coding redundancy is functionally equivalent to replication, and with l as the only tuning parameter, the strategy corresponds to GR. Finally, when $l = n$, each node forms an individual group and cannot provide redundancy. In this case, k must also equal 1, and the strategy matches the FCFS strategy.

These relationships show that different redundancy strategies can be represented as special cases of a common parameterized framework. This allows us to analyze system performance directly in terms of the key parameters l and k, and attribute performance differences to how redundancy is structured. As a result, we focus on the optimal parameter settings of GMDS and GR, since other strategies can be obtained as special cases under specific parameter configurations.

C. Performance Evaluation of Coding-based Strategies

To assess whether traditional fixed-size batch coding can effectively reduce the average latency of streaming tasks, we compare the performance of several methods introduced earlier based on the latency expressions and the relationships derived in previous sections.

Since GR reduces to FR ($l = 1$) and FCFS ($l = n$), its latency $\mathbb{T}^{GR}(l)$ interpolates between these two extremes, and the optimal group number l^* yields $\mathbb{T}^{GR^*} \leq \min\{\mathbb{T}^{FCFS}, \mathbb{T}^{FR}\}$. Because GR is itself the special case $k = 1$ of GMDS, we have

$$\mathbb{T}^{GMDS^*} = \min_{k,l} \mathbb{T}^{GMDS}(k,l) \ \leq \ \min_{l} \mathbb{T}^{GMDS}(1,l) = \mathbb{T}^{GR^*}.$$

However, this inequality alone does not establish the advantage of batch coding, since the optimal parameters (k^*, l^*) may still reduce GMDS to GR (and hence to FR or FCFS).

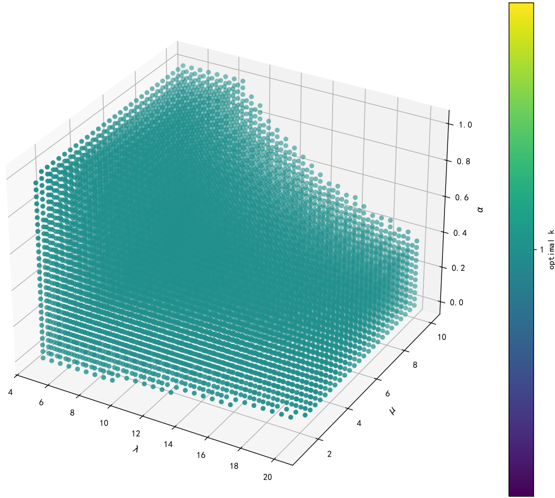


Fig. 3. Optimal k of GMDS under varying λ, μ, and α.

Due to the analytical complexity of the GMDS latency expression, deriving the optimal parameters (k^*, l^*) in closed form is intractable. Therefore, we conduct an exhaustive numerical evaluation by enumerating multiple feasible parameter pairs, i.e., $l \in \{1, \ldots, n\}$ and $k \in \{1, \ldots, \lfloor n/l \rfloor\}$, to identify the optimal configuration under different system settings.

The results are illustrated in Fig. 3. Under the considered stable operating conditions ($\rho < 1$), and for the tested ranges of λ, μ, and α, the optimal encoding block size is mostly observed to be $k^* = 1$. This observation suggests that, in these parameter regimes, the optimal GMDS strategy often reduces to GR.

A possible explanation is that increasing k may improve coding efficiency in some cases, but it also introduces additional waiting and synchronization delays for forming coding blocks in streaming scenarios. When these delays dominate the coding gain, directly replicating individual tasks can achieve comparable or even lower average latency. Nevertheless, this does not exclude the possibility that batch coding may be beneficial in certain parameter regions where the coding gain outweighs the additional batching delay.

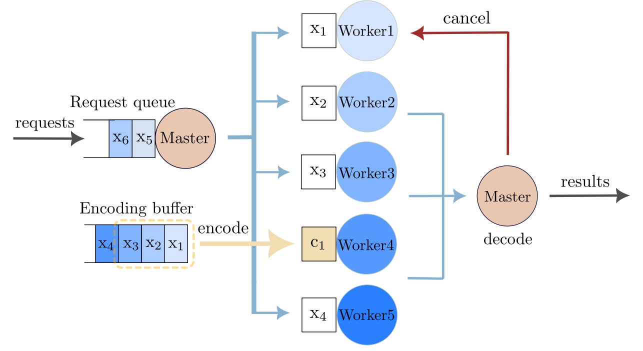


Fig. 4. An example about the insert coding method with $n = 5, k = 3$.

Therefore, the experimental results indicate that conventional fixed-size batch coding does not consistently outperform group-based replication in terms of average latency for streaming task systems. Instead, its advantage appears to be parameter-dependent.

VI. K-PERIODIC INSERTED CODING METHOD

Since existing coding strategies are limited in streaming scenarios due to their reliance on fixed block sizes and lack of flexibility, we further introduce a k-periodic inserted coding (KIC) method as a supplementary attempt. In this section, we first describe the KIC scheme and then present its corresponding latency expression.

A. Method Description

As illustrated in Fig. 4, when a task arrives at the system, it first queues in a first-in-first-out (FIFO) request queue, at the same time, the system maintains a coding buffer that records all tasks that have not yet completed. Once a task is completed, it is removed from the buffer. Whenever k tasks have been sent to nodes for execution, all unfinished tasks in the coding buffer are encoded into a coded task, which is then inserted to be executed before the tasks currently in the request queue. If a task is successfully decoded first through the coded block, the computation on the node directly executing that task will be canceled. Similarly, if all tasks in the coded block are completed, the computation on the node processing that coded block will be canceled. The detailed execution procedure of the method is shown in Algorithm 1.

B. Latency Expression

To better illustrate how the proposed method helps reduce the average task latency, we provide a formulation of the latency under the KIC strategy. For a given task, its latency includes two possible cases: the task can be completed either through normal execution by a node or earlier through decoding a coded block. Denote the completion time of task i under direct execution without coding as T_i^0, while $\mathcal{C}_i$ represents the set of coded blocks that contain task i. The decoding completion time of task i from coded block m is represented

Algorithm 1: K-periodic Inserted Coding Method (KIC)

Input: insertion interval k, number of nodes n
Initialization: FIFO request queue $\mathcal{Q} \leftarrow \emptyset$, coding buffer $\mathcal{B} \leftarrow \emptyset$, counter $c \leftarrow 0$

```
1  Function Encode(B):
2  |   Encode all tasks in B.
3  |   return coded task
4  while task i arrives do
5  |   Q ← Q ∪ {i}
6  end
7  while exits an idle node do
8  |   if B ≠ ∅, c > 0 and c mod k = 0 then
9  |   |   coded task j^code ← Encode(B)
10 |   |   Assign j^code to the idle node.
11 |   else if Q ≠ ∅ then
12 |   |   task j ← Dequeue(Q)
13 |   |   Assign j to the idle node.
14 |   |   c ← c + 1, B ← B ∪ {j}
15 |   end
16 end
17 while task j is completed do
18 |   B ← B \ {j}
19 |   if j is decoded from a coded task then
20 |   |   Cancel execution of the original task of j.
21 |   end
22 |   if all tasks in the coded block are completed then
23 |   |   Cancel execution of the coded task.
24 |   end
25 end
```

by T_i^m. Accordingly, the actual completion time of task i is given by

$$T_i = \min\left(T_i^0, \min_{m \in \mathcal{C}_i}\{T_i^m\}\right).$$

To analyze this in detail, we next focus on modeling the completion time of tasks recovered from coded blocks. Notably, the completion time is determined by the decodability of the coded block rather than its computation finish time. Let $\mathcal{D}_m$ be the set of tasks encoded in coded block m, and let S_m denote the service completion time of coded block m, including queuing and processing time. Then, the decoding completion time of task i from coded block m is given by:

$$T_i^m = \max\left\{S_m, \max_{j \in \mathcal{D}_m \setminus \{i\}} T_j\right\}.$$

Therefore, the average latency of this method is:

$$\mathbb{T}^{KIC} = \mathbb{E}\left[\min\left(T_i^0, \min_{m \in \mathcal{C}_i} \max\left\{S_m, \max_{j \in \mathcal{D}_m \setminus \{i\}} T_j\right\}\right)\right].$$

The insertion frequency k controls the formation frequency and size of $\mathcal{C}_i$ and $\mathcal{D}_m$, thereby influencing both the probability and the number of times each task is covered by coded blocks. A smaller k leads to more frequent coding with larger $\mathcal{C}_i$ and smaller $\mathcal{D}_m$, whereas a larger k results in sparser coding, yielding smaller $\mathcal{C}_i$ and larger $\mathcal{D}_m$.

While we provide a formal expression for the average task latency under the KIC strategy, deriving an analytical solution remains intractable due to the recursive and interdependent nature of the completion time terms. Specifically, the decoding time of a coded block T_i^m is jointly determined by the completion time of the coded block itself and all other tasks it encodes. These task completion times, in turn, may depend on the decoding of other coded blocks, resulting in a complex dependency graph with no closed-form solution. Moreover, the asynchronous and stochastic arrival of streaming tasks further complicates the modeling of queuing dynamics and coded block formation.

Given this complexity, we adopt a data-driven approach to determine the optimal coding interval parameter k. Under the assumption that the task arrival process, service time distribution, and the number of system nodes n are known or can be estimated, we first simulate the system behavior across a range of k values to empirically identify the one that minimizes average latency. This offline simulation process can be viewed as a digital twin of the real system, providing performance insights without interfering with actual operations. Once the optimal k is obtained in the simulation environment, it can be deployed in the real-time scheduling of streaming tasks to achieve near-optimal latency performance in practice.

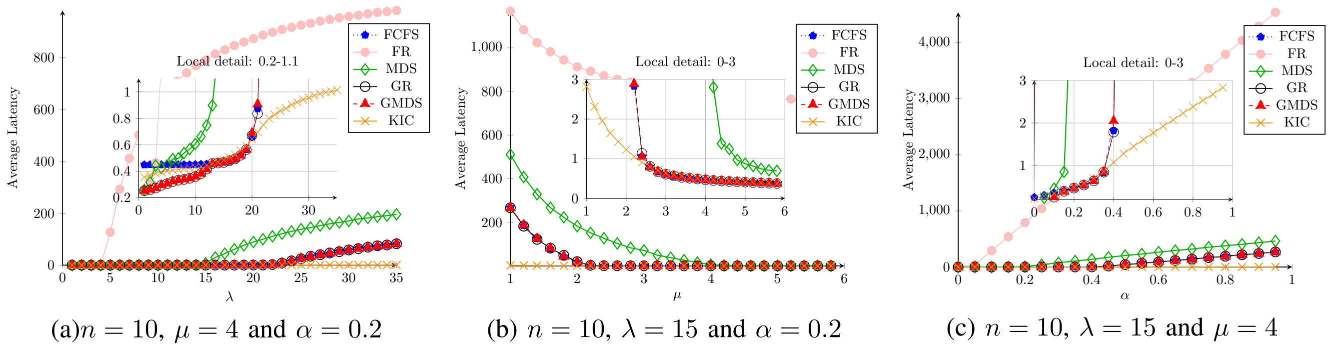
VII. Performance Evaluation

To evaluate the performance of the proposed *K-periodic Inserted Coding (KIC)* method in streaming prediction service systems, we conduct discrete-event simulations that model the execution processes of six task scheduling strategies: FCFS, FR, GR, MDS, GMDS, and the proposed KIC. These strategies are simulated under identical system conditions, and we compare their average task latency by varying system parameters.

A. Simulation Settings

We simulate a cloud computing environment by modeling both streaming tasks and compute nodes. Tasks arrive according to a Poisson process with rate λ. The system consists of $n = 10$ worker nodes operating in parallel. The service time of each node follows a shifted exponential distribution with rate μ and shift factor α, capturing both the inherent processing delay and stochastic variability across nodes. Each arriving task is assigned to a node for execution, and system performance is evaluated based on task completion times.

For each strategy, we simulate the processing of 5000 tasks. For every task, we record its arrival and departure times, and compute its latency as the difference between the two. The average task latency is obtained by averaging over all tasks. To ensure statistical reliability, each simulation is repeated 10 times under the same parameter configuration, and the reported results are averaged across these runs.



(a)$n = 10$, $\mu = 4$ and $\alpha = 0.2$ (b) $n = 10$, $\lambda = 15$ and $\alpha = 0.2$ (c) $n = 10$, $\lambda = 15$ and $\mu = 4$

Fig. 5. Performance of the schemes under different λ, μ and α.

For the parameterized strategies GR, MDS, and GMDS, we determine optimal internal parameters such as k and l through numerical search. Specifically, we enumerate the possible values of these parameters, simulate the corresponding average latency for each configuration, and select the setting that achieves the lowest latency. Simulations for these methods are then conducted using these optimized parameters.

For the proposed KIC method, we optimize the insertion interval k by enumerating all values from 1 to n. For each candidate k, we run simulations with 100 task arrivals per run and repeat the process 10 times to reduce randomness. The value of k that achieves the lowest average latency is selected as the optimal configuration.

B. Experimental Results and Analysis

We consider three key system parameters: task arrival rate λ, service rate μ, and shift factor α. Their default values are set to $\lambda = 15, \mu = 4$, $\alpha = 0.2$, with the number of nodes in the system set to $n = 10$. Based on this baseline configuration, we conduct four sets of experiments, each varying one parameter individually to examine its impact on the scheduling performance. The experimental results are shown in Fig. 5.

The results indicate that the proposed KIC method achieves superior performance under high-load conditions, such as low μ, large λ, or large α. As these factors increase system load and push utilization toward saturation, traditional redundancy-based strategies such as replication-based and coded-based scheduling suffer from rapidly growing latency (e.g., exceeding 60 when $\lambda = 35$). This is because excessive redundancy consumes additional computing and queueing resources, further amplifying delays in overloaded systems. As a result, under such circumstances, the non-redundant method emerges as the most efficient among the five traditional strategies.

Unlike traditional approaches, KIC introduces redundancy in a selective and adaptive manner, specifically targeting tasks executed on slower computing nodes. Each coded chunk is intended to accelerate only the slowest task it contains, resulting in a highly focused rather than broadly distributed performance gain. This targeted compensation mechanism is less effective under low-load conditions, where system resources are abundant and traditional redundancy strategies can aggressively insert additional redundancy to tolerate multiple slow nodes simultaneously. However, when the system is overloaded, traditional redundancy methods introduce substantial overhead by uniformly applying redundancy, which often leads to degraded performance and can even be outperformed by the non-redundant FCFS method. In contrast, KIC inserts redundancy only when necessary and targets tasks on slow nodes, thereby effectively accelerating stragglers. This selective mechanism enables KIC to maintain low latency even under high load (e.g., below 1 when $\lambda = 35$, corresponding to about 98.8% reduction compared to the best baseline) and achieve better overall performance when the system is operating near or beyond saturation.

VIII. Conclusions and Future Work

This paper investigated the effectiveness of coding-based redundancy in distributed stream processing systems. Through queuing-theoretic analysis, we modeled and evaluated the latency performance of five representative scheduling strategies, including uncoded, replication-based, and traditional coding-based methods. The results suggest that, under the adopted assumptions, traditional fixed-block coding may provide limited latency benefits in streaming scenarios, since its coding gain can be offset by batching and synchronization delays. Based on this observation, we proposed k-periodic inserted coding (KIC), an exploratory coding strategy that periodically inserts coded tasks for unfinished requests to provide more flexible redundancy. Simulation results show that KIC can effectively reduce average latency under the tested high-load settings, suggesting the potential of coding-based redundancy mechanisms specifically designed for stream task arrivals.

As part of our future work, we will extend the analytical framework to heterogeneous workers and other service distributions, and conduct a sensitivity analysis of KIC with respect to different values of k under these settings. We also plan to replace the offline parameter search with an online tuner that dynamically adjusts k under non-stationary workloads. Finally, we will consider deploying KIC on real-world cloud platforms such as AWS EC2 to evaluate its performance and quantify the practical impact of encoding, decoding, and coordination overheads in realistic distributed environments.

REFERENCES

[1] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[2] M. F. Aktaş and E. Soljanin, "Straggler mitigation at scale," *IEEE/ACM Transactions on Networking*, vol. 27, no. 6, pp. 2266–2279, 2019.

[3] 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, 2017.

[4] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Codedreduce: A fast and robust framework for gradient aggregation in distributed learning," *IEEE/ACM Transactions on Networking*, vol. 30, no. 1, pp. 148–161, 2021.

[5] W. Tang, J. Li, L. Chen, and X. Chen, "Design and optimization of hierarchical gradient coding for distributed learning at edge devices," *IEEE Transactions on Communications*, vol. 72, no. 12, pp. 7727–7741, 2024.

[6] C. Li and M. Skoglund, "Distributed learning based on 1-bit gradient coding in the presence of stragglers," *IEEE Transactions on Communications*, vol. 72, no. 8, pp. 4903–4916, 2024.

[7] M. Fahim and V. R. Cadambe, "Numerically stable polynomially coded computing," *IEEE Transactions on Information Theory*, vol. 67, no. 5, pp. 2758–2785, 2021.

[8] W. Kim, S. Kruglik, and H. M. Kiah, "Verifiable coded computation of multiple functions," *IEEE Transactions on Information Forensics and Security*, 2024.

[9] J. Kosaian, K. Rashmi, and S. Venkataraman, "Parity models: erasure-coded resilience for prediction serving systems," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 30–46.

[10] ——, "Learning-based coded computation," *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 227–236, 2020.

[11] H. V. K. G. Narra, Z. Lin, G. Ananthanarayanan, S. Avestimehr, and M. Annavaram, "Collage inference: Using coded redundancy for lowering latency variation in distributed image classification systems," in *Proceedings of the 40th IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2020, pp. 453–463.

[12] J. Zhang, X. He, and H. Dai, "Speeding up distributed learning via sparse and flexible coded computing," *IEEE Transactions on Information Theory*, 2024.

[13] Y. Gong, Z. Jiang, Y. Feng, B. Hu, K. Zhao, Q. Liu, and W. Ou, "Edgerec: recommender system on edge in mobile taobao," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 2477–2484.

[14] X. Chen, W. Zhu, J. Chen, T. Zhang, C. Yi, and J. Cai, "Edge computing enabled real-time video analysis via adaptive spatial-temporal semantic filtering," in *2023 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*. IEEE, 2023, pp. 262–267.

[15] W. Marfo, E. A. Rico, D. K. Tosh, and S. V. Moore, "Network anomaly detection in distributed edge computing infrastructure," in *2025 IEEE 22nd Consumer Communications & Networking Conference (CCNC)*. IEEE, 2025, pp. 1–6.

[16] K. Gardner, M. Harchol-Balter, A. Scheller-Wolf, M. Velednitsky, and S. Zbarsky, "Redundancy-d: The power of d choices for redundancy," *Operations Research*, vol. 65, no. 4, pp. 1078–1094, 2017.

[17] A. Behrouzi-Far and E. Soljanin, "Efficient replication for fast and predictable performance in distributed computing," *IEEE/ACM Transactions on Networking*, vol. 29, no. 4, pp. 1467–1476, 2021.

[18] ——, "Balanced nonadaptive redundancy scheduling," *IEEE Journal on Selected Areas in Information Theory*, vol. 3, no. 2, pp. 422–430, 2022.

[19] K. T. Kim, C. Joe-Wong, and M. Chiang, "Coded edge computing," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, 2020, pp. 237–246.

[20] C.-S. Yang, R. Pedarsani, and A. S. Avestimehr, "Edge computing in the dark: Leveraging contextual-combinatorial bandit and coded computing," *IEEE/ACM Transactions on Networking*, vol. 29, no. 3, pp. 1022–1031, 2021.

[21] K. Son and W. Choi, "Coded matrix computation in wireless network," *IEEE Transactions on Wireless Communications*, vol. 23, no. 6, pp. 6394–6410, 2023.

[22] M. Dai, Z. Zhang, Z. Zheng, Z. Zhang, X. Lin, and H. Wang, "2d-sazd: A novel 2d coded distributed computing framework for matrix-matrix multiplication," *IEEE Transactions on Services Computing*, vol. 17, no. 3, pp. 705–717, 2024.

[23] S. Song and S. Zhao, "Research on task encoding scheme in edge computing scenarios," in *2025 International Conference on Sensor-Cloud and Edge Computing System (SCECS)*, 2025, pp. 14–21.

[24] B. Wang, J. Xie, K. Lu, Y. Wan, and S. Fu, "On batch-processing based coded computing for heterogeneous distributed computing systems," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 3, pp. 2438–2454, 2021.

[25] P. Soto, X. Fan, A. Saldivia, and J. Li, "Rook coding for batch matrix multiplication," *IEEE Transactions on Communications*, vol. 70, no. 6, pp. 3641–3654, 2022.

[26] B. Wang, J. Xie, K. Lu, Y. Wan, and S. Fu, "Learning and batch-processing based coded computation with mobility awareness for networked airborne computing," *IEEE Transactions on Vehicular Technology*, vol. 72, no. 5, pp. 6503–6517, 2022.

[27] A. Khalesi and P. Elia, "Tessellated distributed computing," *IEEE Transactions on Information Theory*, 2025.

[28] A. Cohen, G. Thiran, H. Esfahanizadeh, and M. Médard, "Stream distributed coded computing," *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 3, pp. 1025–1040, 2021.

[29] H. Esfahanizadeh, A. Cohen, and M. Médard, "Stream iterative distributed coded computing for learning applications in heterogeneous systems," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 230–239.

[30] Z. Cheng, L. Tang, Q. Huang, and P. P. Lee, "Enabling low-redundancy proactive fault tolerance for stream machine learning via erasure coding: Design and evaluation," *Available at SSRN 4192493*, 2022.

[31] Z. Ji, L. Chen, H. Fu, X. Zhao, J. Xu, L. Meng, and J. Liu, "Multiple-task coded computing for distributed computation framework: Modeling and delay analysis," *IEEE Transactions on Communications*, vol. 72, no. 8, pp. 5032–5046, 2024.

[32] Y. Wang and Y. Wu, "Coded distributed computing with pre-set data placement and output functions assignment," *IEEE Transactions on Information Theory*, 2025.

[33] N. Gans, G. Koole, and A. Mandelbaum, "Telephone call centers: Tutorial, review, and research prospects," *Manufacturing & Service Operations Management*, vol. 5, no. 2, pp. 79–141, 2003.

[34] M. Barbeau and E. Kranakis, *Principles of ad-hoc networking*. John Wiley & Sons, 2007.

[35] F. Pollaczek, "Über eine aufgabe der wahrscheinlichkeitstheorie. i: Mitteilung aus dem telegraphentechnischen reichsamt," *Mathematische Zeitschrift*, vol. 32, no. 1, pp. 64–100, 1930.

[36] A. Y. Khintchine, "Mathematical theory of stationary queues," *Matem. Sbornik*, vol. 39, pp. 73–84, 1932.

[37] Z. Guo, X. Ji, W. You, H. Guo, Y. Zhang, Y. Zhao, M. Xu, and Y. Bai, "Workload distribution with rateless encoding: A low-latency computation offloading method within edge networks," *Computer Networks*, p. 111381, 2025.

[38] Y. Wang, S. Gu, Z. Zhang, Q. Zhang, and W. Xiang, "Block allocation of systematic coded distributed computing in heterogeneous straggling networks," in *GLOBECOM 2023-2023 IEEE Global Communications Conference*, 2023, pp. 1066–1071.

[39] T. H. Nguyen, H. L. N. Thi, H. Le Hoang, J. Tan, N. C. Luong, S. Xiao, D. Niyato, and D. I. Kim, "Coded distributed computing for vehicular edge computing with dual-function radar communication," *IEEE Transactions on Vehicular Technology*, vol. 73, no. 10, pp. 15 318–15 331, 2024.

[40] H. A. David and H. N. Nagaraja, *Order statistics*. John Wiley & Sons, 2004.

[41] U. N. Bhat, "The general queue g/g/1 and approximations," in *An Introduction to Queueing Theory: modeling and analysis in applications*. Springer, 2008, pp. 169–183.

[42] J. F. Kingman, "The single server queue in heavy traffic," in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 57, no. 4. Cambridge University Press, 1961, pp. 902–904.

[43] W. Whitt, "Approximations for the gi/g/m queue," *Production and Operations Management*, vol. 2, no. 2, pp. 114–161, 1993.