Tackling Heterogeneous Traffic in Multi-access Systems via Erasure Coded Servers

Tuhinangshu Choudhury
Carnegie Mellon University
Pittsburgh, PA

Weina Wang
Carnegie Mellon University
Pittsburgh, PA

Gauri Joshi
Carnegie Mellon University
Pittsburgh, PA

ABSTRACT

Most data generated by modern applications is stored in the cloud, and there is an exponential growth in the volume of jobs to access these data and perform computations using them. The volume of data access or computing jobs can be heterogeneous across different job types and can unpredictably change over time. Cloud service providers cope with this demand heterogeneity and unpredictability by over-provisioning the number of servers hosting each job type. In this paper, we propose the addition of erasure-coded servers that can flexibly serve multiple job types without additional storage cost. We analyze the service capacity region and the response time of such erasure-coded systems and compare them with standard uncoded replication-based systems currently used in the cloud. We show that coding expands the service capacity region, thus enabling the system to handle variability in demand for different data types. Moreover, we characterize the response time of the coded system in various arrival rate regimes. This analysis reveals that adding even a small number of coded servers can significantly reduce the mean response time, with a drastic reduction in regimes where the demand is skewed across different job types.

CCS CONCEPTS

• **Mathematics of computing** → **Queueing theory**; *Coding theory*; • **Networks** → **Network performance analysis**.

KEYWORDS

Erasure coding, coded computation, latency, service capacity region

ACM Reference Format:
Tuhinangshu Choudhury, Weina Wang, and Gauri Joshi. 2022. Tackling Heterogeneous Traffic in Multi-access Systems via Erasure Coded Servers. In *The Twenty-third International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (MobiHoc '22), October 17–20, 2022, Seoul, Republic of Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3492866.3549713

1 INTRODUCTION

Modern-day cloud computing systems are used for various big-data applications such as performing machine learning (ML) inference tasks [9], hosting large files for web services [24], computing the PageRank of web graphs [21], and other data-intensive computations. Since these jobs require access to the specific data stored on the cloud server(s), each server is dedicated to one job type depending on the availability of data and high-performance hardware required for it. Therefore, the massive volume of users performing such cloud computing jobs have to contend for the server(s) storing data relevant to their job. For example, an ML inference system may store one trained model on each server, users that need to perform inference using that model are directed to that server. Or a cloud storage system hosting files for OTT platforms such as Netflix might use each server to store one movie, and users requesting that movie are assigned to that server. Therefore, the massive volume of users performing different cloud computing jobs have to contend for the server(s) storing data relevant to their job.

The traffic for different job types can vary due to diurnal variations or unpredictable demand fluctuations. These traffic variations often exhibit a negative correlation, i.e., if one job type is experiencing high traffic, another job type experiences low traffic. For e.g., in an ML inference system hosting different specialized models to process pictures captured by users, the inference traffic for each model can vary periodically depending on the time zone of the users accessing them. Such computing and inference jobs are latency-sensitive. Thus, a larger number of servers needs to be allocated to job types experiencing high traffic to satisfy the latency requirements of those users. The ideal solution for such situations is to enable dynamic allocation of servers where the number of servers provided to a job type depends on its current traffic. However, the servers often host large files (for e.g., the size of the Google Web crawler is over 10 million GB) and have specialized computing hardware. Dynamically reconfiguring a server for a different job type would require the movement of large amounts of data and may not even be possible due to hardware constraints. Hence, there is a critical need to design multi-access computing systems resilient to traffic variations that avoid dynamic reconfiguration of servers.

One standard solution to handle traffic variations is to overprovision the number of servers dedicated to various job types to meet their peak demand. Overprovisioning can be achieved by adding replicas of servers dedicated to a job type [8, 26] in proportion to the maximum historical demand for that job type over a large time horizon. While over-provisioning can meet latency requirements under traffic variations, it comes at the cost of severe underutilization of expensive computing resources and a massive energy footprint. Another solution is to supplement job-type-specific servers with flexible general servers that can serve more than one job type. However, adding such servers can be expensive because they need access to data, memory, and computing capabilities relevant to multiple job types. One such model was studied by Tsitsiklis and Xu in [27]

MobiHoc '22, October 17–20, 2022, Seoul, Republic of Korea

ACM ISBN 978-1-4503-9165-8/22/10.
https://doi.org/10.1145/3492866.3549713

which showed that the addition of a small number of flexible servers could give a dramatic reduction in the queueing delay.

While the solutions mentioned above provide some robustness to variations in traffic, none of them considers the correlation between traffic. For negatively correlated arrival rates, the overall traffic in the system varies slowly over time, even though individual jobs may experience significant changes. A novel approach that is well-suited for such skewed traffic patterns was proposed in [1, 2], where the authors supplement replicas of the servers of each type by a set of *erasure coded servers*. When a job is sent to an erasure-coded server, the output is a linear combination of the outputs of two or more of the regular servers. For example, in a matrix computation task where job type 1 (or type 2) seeks to compute the product $\boldsymbol{Ax}$ (or $\boldsymbol{Bx}$) of an incoming vector $\boldsymbol{x}$ with the matrix $\boldsymbol{A}$ (or $\boldsymbol{B}$) stored at a server, a coded server can store $\boldsymbol{A}+\boldsymbol{B}$ such that its output is $(\boldsymbol{A}+\boldsymbol{B})\boldsymbol{x}$. Thus, a type 1 job can be served and its output $\boldsymbol{Ax}$ can be obtained using an uncoded server storing $\boldsymbol{B}$ and a coded server storing $\boldsymbol{A}+\boldsymbol{B}$. Using this property, erasure-coded servers can serve multiple job types when combined with regular servers. Furthermore, unlike the flexible servers proposed in [27], the coded servers do not require extra resources such as memory. For various encoding schemes, [1, 2] showed an improvement in the service capacity region of the system, the set of arrival rates for which the system is stable. The addition of coded servers significantly expands the service capacity region, especially in regions where the traffic for different job types is negatively correlated.

1.1 Main Contributions

While [1, 2] proposed the idea of using coded servers to handle traffic variations and showed an expansion of the service capacity region, these works did not analyze the impact of coded servers on performance metrics such as mean response time or tail latency. Latency is important as often the request for a service has a deadline, in which case just ensuring stability of the system might not satisfy the user requirements. Since a coded server has to be used with one or more other servers, coding can increase the system load and result in a higher response time in some traffic regimes. However, this effect on the mean response time is not yet well-understood.

In this paper, we build upon the model provided in [1, 2] and generalize it to consider ≥ 2 job types and an arbitrary allocation of servers to each job type and the coded servers. We compare the coded system with an uncoded system with the same total number of servers and corroborate the insight that the coded system significantly improves the system's stability by increasing the volume of the service capacity region. In addition, we characterize the mean response times of the coded and uncoded systems in several traffic regimes. We show that for a large number of servers, our coded system has a comparable or significantly smaller mean response time in most traffic regimes. The reduction in mean response time is prominent when the traffic of various job types is heavy and skewed, i.e., some job types are experiencing significantly higher traffic than other job types. The only regime where the uncoded system is better is when all job types simultaneously have high traffic, which is unlikely to occur in practical applications. Thus, we show our coded system can better handle heterogeneous traffic for different job types, and it improves the stability as well as the latency of multi-access systems.

1.2 Related Work

Improving the latency and service capacity of multi-access cloud systems has been extensively studied in the literature. In the context of storage systems, it has been studied in the caching literature [4, 19, 22], where the number and location of replicas are dynamically adjusted based on the traffic. However, in data-intensive computing and content access jobs, such dynamic reconfiguration of servers is not feasible. Another approach proposed in [27] is to add flexible servers that can serve multiple job types, but these servers will require additional memory and computing capabilities. Our proposed coding strategy does not require dynamic server reconfiguration or expensive flexible servers.

The use of erasure coding in distributed storage and computing systems is not new, in most prior works, the purpose of coding is straggler mitigation in jobs with parallel computing tasks, or latency reduction by launching redundant replicas of a job. For jobs that are divided into many parallel tasks such as MapReduce workloads, the tail latency of waiting for the slowest task(s) can be reduced by replication of tasks [3, 10, 14, 16, 28] such that the completion of only a subset of tasks is sufficient to complete the job. A generalized form of replication is erasure coding using (n, k) maximum-distance-separable (MDS) codes, which offers a more efficient method to mitigate stragglers. In the context of content download from distributed storage, erasure coded systems divide a file into k chunks that are coded into n chunks such that downloading any k out of n is sufficient to recover the file. The latency of such distributed systems with redundant requests is analyzed in [6, 12, 14, 15, 18, 23]. All the above works consider homogeneous jobs of only one type. In contrast, this paper employs erasure coding to achieve a different goal of handling skews in the traffic of heterogeneous jobs. To the best of our knowledge, this novel application of coding has only been previously studied in [1, 2], which we build on by considering a more general system and characterizing the mean response time in addition to the service capacity region.

2 PROBLEM FORMULATION

We consider a multi-access cloud system consisting of n servers that are used to provide service to k types of jobs. Jobs of each type i arrive into the system according to a Poisson process. We assume that the traffic variation happens at a slower time-scale such that the system reaches a steady state before the traffic pattern changes. Therefore, it suffices to consider the setting where type i jobs have a constant arrival rate λ_i. The service times of n servers are unit-rate exponential random variables and are i.i.d. across servers and the jobs assigned to the server.

Uncoded System. In most current implementations, the n servers are divided into k disjoint subsets $\boldsymbol{S}_i, i = 1, 2, \ldots, k$, each consisting of $|\boldsymbol{S}_i| = n_i = \alpha_i n$ servers for fractions $0 < \alpha_i < 1$ such that $\sum_{i=1}^{k} \alpha_i = 1$. The fraction α_i of servers dedicated to type i jobs is a hyperparameter that can be set by the system designer. For example, α_i can be proportional to the long-term average of past values of the arrival rate λ_i. Since future arrival rates are not known

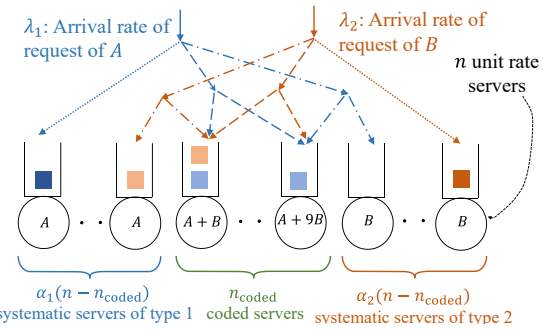


Figure 1: An example of a multi-access computing system hosting two matrices A and B, and consisting of n unit-rate servers of which n_{coded} are coded servers. The value written inside each server shows matrix stored in the server. Tasks corresponding to A is represented by blue squares, while the one for B is shown using orange squares. Different line types represents different ways in which a job can be served.

beforehand or they may vary with time, the number of servers dedicated to each job type may not be sufficient to satisfy its arrival rate. Dynamically re-configuring a server to serve a different job type is expensive and slow in cloud systems, because it involves the movement of large amounts of data. Thus, we consider that the server types are fixed beforehand and cannot be modified on the fly, that is, type i systematic servers cannot be quickly reconfigured to serve jobs of type j for $j \neq i$.

Coded System. In this paper, we consider a more generalized setup where out of n servers, we reserve a set $\mathcal{S}_{\text{coded}}$ of $n_{\text{coded}} \triangleq |\mathcal{S}_{\text{coded}}|$ servers to host *erasure coded* versions of the job types. The remaining $n - n_{\text{coded}}$ systematic servers are split across the k job-types such that $\alpha_i(n - n_{\text{coded}})$ servers are the systematic set $\mathcal{S}_i$ of job type i, with $0 < \alpha_i < 1$ for all $i = 1, \ldots, k$ and $\sum_{i=1}^{k} \alpha_i = 1$. We illustrate this server assignment for the $k = 2$ case in Fig. 1.

Erasure codes [11] were originally developed in communication systems to add redundancy to transmitted messages in order to provide resilience against noise in the communication channel. In this paper, we employ them for a new purpose – to handle skews in the arrival rate of various job types. When the arrival rate λ_i of type i jobs exceeds the cumulative service rate of the type i systematic servers, the excess arrivals can be served using a combination of the coded servers and systematic servers of type(s) $j \neq i$. To illustrate the benefit of coded servers, we first give two concrete examples of erasure coded systems for $k = 2$ job types. Then we describe the proposed coded system for general k.

Example 1. (Coded storage system)

Consider an online storage platform with two video files, V_1 and V_2 of equal size, and 3 servers that can host one file each. Consider that the servers store V_1, V_2 and $(V_1 \oplus V_2)$ respectively, where $(V_1 \oplus V_2)$ is the bit-wise XOR of V_1 and V_2. Note that the data-size stored on the coded server $(V_1 \oplus V_2)$ is the same as the file size V_1 and V_2. The jobs of type 1 and 2 are download requests for files V_1 and V_2. A type 1 job can be served in two ways: 1) it can be sent to the systematic server storing V_1, or 2) it can be sent to the systematic server storing V_2 and the coded server storing $(V_1 \oplus V_2)$, and after downloading these two files, V_1 can be recovered by taking their bit-wise XOR, $(V_1 \oplus V_2) \oplus V_2 = V_1$. Thus, if the server storing V_1 cannot meet the demand for file V_1, the excess requests can be served using V_2 and $(V_1 \oplus V_2)$.

To design more general coded storage systems with more than two files and more than one coded server, the files can be represented in a higher alphabet size than bits which will allow more flexible linear combinations. Such erasure coding of files is currently used in commercial cloud storage systems such as RAID [5] for the purpose of resilience against disk failures.

Besides coded storage systems where the jobs represent download requests, erasure coding can be applied to computing systems, as illustrated in Example 2 below.

Example 2. (Coded computing system) Consider an online computing system, where a computing job seeks to find the product of an input vector $\boldsymbol{x}$ with the matrix $\boldsymbol{A}$ (or $\boldsymbol{B}$). Matrix computations are backbone of ML inference systems and such jobs are common where the vector $\boldsymbol{x}$ is an inference query and the matrix-vector product $\boldsymbol{Ax}$ (or $\boldsymbol{Bx}$) is the predicted output of a linear model. We assume that the matrices to be of same shapes. For different shapes, one can pad zeros to the matrices to unify their shapes. Suppose we have a system consisting of 5 servers, where each server stores one of two large matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, or their linear combination. Consider that the 5 servers store $\boldsymbol{A}, \boldsymbol{B}, (\boldsymbol{A}+\boldsymbol{B}), (\boldsymbol{A}-\boldsymbol{B})$, and $(\boldsymbol{A}+2\boldsymbol{B})$, respectively. Given an input vector $\boldsymbol{x}$, the server multiplies the vector with the stored matrix and outputs the matrix-vector product. A type 1 job that seeks to find the product $\boldsymbol{Ax}$ can be served in the following ways: 1) sending the job to the server storing $\boldsymbol{A}$, 2) sending the job to any two of the three coded servers $(\boldsymbol{A}+\boldsymbol{B})$, $(\boldsymbol{A}-\boldsymbol{B})$ and $(\boldsymbol{A}+2\boldsymbol{B})$ and obtain $\boldsymbol{Ax}$ by taking a linear combination of the resulting matrix-vector products, or 3) sending the job to the systematic server storing $\boldsymbol{B}$ and any one of the three coded servers, and solving for $\boldsymbol{Ax}$ from the resulting matrix-vector products.

Since erasure codes are inherently linear, the coded computing framework described above can be directly applied only to linear computations such as matrix-vector multiplication. However, some recent research in coding theory is designing ways to apply erasure codes to non-linear computations [17, 20] such as kernel methods and neural networks. The queueing and scheduling insights presented in this paper can be extended to the non-linear coded computing frameworks proposed in these works.

Maximum-Distance Separable Codes and Recovery Sets. The examples shown above considered just 2 types of jobs. More generally, when there are k types of jobs, we propose a coded multi-access system that employs a class of erasure codes called *maximum-distance-separable (MDS) codes* [11]. MDS codes are often used in distributed storage systems to provide resilience against disk failures. If there are k files that need to be stored on l disks, an (l, k) MDS code constructs l independent linear combinations of the k files such that a file i can be recovered from any set of k coded files. A commonly used MDS code is the Reed-Solomon code, which constructs the linear combinations by evaluating a $k-1$-th degree polynomial at l points. A special case of an (l, k) MDS code is the *systematic* MDS code, where k of the l combinations are uncoded copies of the k files, and the remaining $l - k$ are other independent

linear combinations. In our coded multi-access cloud system, we consider such a systematic MDS code. We have $|\boldsymbol{S}_i| = \alpha_i(n - n_{\text{coded}})$ servers storing uncoded copies of each job type i and n_{coded} independent linear combinations of the k job types.

In our MDS coded system, we can serve a type i job using one of the following options: 1) one of the systematic servers from set $\boldsymbol{S}_i$, 2) any k coded combinations from the set $\boldsymbol{S}_{\text{coded}}$ of coded servers, or 3) k_1 coded servers and $k - k_1$ distinct systematic servers from the sets $\boldsymbol{S}_j$, where $j \in \{1, 2, \cdots, k\}\backslash\{i\}$ and for some integer k_1 such that $1 \leq k_1 < k$. We denote the union of all these possible subsets of servers that can serve a type i job by $\boldsymbol{R}_i$, and refer to it as the *set of recovery sets*. The size $|\boldsymbol{R}_i|$ is the number of possible recovery sets for job type i, and $\boldsymbol{R}_{ij}$ denotes the set of servers corresponding to the j-th recovery set, for $j = 1, \ldots, |\boldsymbol{R}_i|$. For instance, in Example 1, the set of recovery sets for the file V_1 is $\boldsymbol{R}_1 = \{\{1\}, \{2, 3\}\}$, consisting of the two possible combinations that can be used to recover V_1.

Queueing model. Next, we describe the queueing model for our system, which is also illustrated in Fig. 1. We consider a first-come-first-served (FCFS) queue at each server. When a type i job arrives, it needs to be assigned to a recovery set in $\boldsymbol{R}_i$ immediately. Specifically, if the chosen set consists of a single systematic server of type i, then the job needs to receive service from that systematic server, and thus we say that *the job consists of one task*. If the chosen set contains $(k - k_1)$ systematic servers and k_1 coded servers for some $k_1 \geq 0$, then the job needs to receive service from all the k servers, and thus we say that *the job consists of k tasks*. This service model induces the following queueing dynamics: when a job arrives, we send a task to the queue at each server in the chosen set in $\boldsymbol{R}_i$. The job is completed when all of its tasks are completed. We call the policy that determines which recovery set to assign to each arriving job as the *routing policy*.

Performance Metrics. Since the coded system provides the flexibility of having multiple ways of serving a job, it improves load balancing. However, this flexibility comes at the cost of redundancy because to serve a job using a coded combination, we need responses from k servers. To compare the coded and uncoded systems in terms of flexibility vs redundancy trade-off, we use two performance metrics: (1) the service capacity region, the set of arrival rate vectors for which the system is stable, and (2) the mean response time of jobs. We define these metrics formally below.

Definition 1 (Service Capacity Region)**.** *The service capacity region of a multi-access cloud system, denoted by Λ, is the region such that for any arrival rate vector $\boldsymbol{\lambda} = (\lambda_1, \cdots, \lambda_k)$ in the interior of the region, there exists a routing policy under which*

$$\lim_{c\to\infty} \lim_{t\to\infty} \mathbb{P}\left(\textit{Number of jobs in the system at time } t > c\right) = 0. \quad (1)$$

The service capacity region consists of all supportable throughput. Therefore, it is a notion independent of policies, and it measures the fundamental limit of the system.

Definition 2 (Mean Response time)**.** *The* response time T *of a job is the total time that a job spends in the system from its arrival until all its tasks are completed. If the system is stable, then the* mean response time *of the system, denoted by $\mathbb{E}[T]$, is defined as follows when the limit on the right-hand-side exists with probability* 1*:*

$$\mathbb{E}[T] = \lim_{m\to\infty} \frac{1}{m} \sum_{j=1}^{m} T_j. \quad (2)$$

where T_j denotes the response time of the j-th job that departs from the system. For an unstable system, we define the mean response time $\mathbb{E}[T]$ to be ∞.

The mean response time is a performance metric specific to the routing policy. In this paper, we consider probabilistic job routing policies. In particular, for any $\boldsymbol{R}_i$, we fix a probability vector $\boldsymbol{p}_i$ of length $|\boldsymbol{R}_i|$, and any incoming job of type i is assigned to $\boldsymbol{R}_{ij}$ with probability p_{ij}, the j-th element of $\boldsymbol{p}_i$. Then, the total arrival rate of type-i job to the set $\boldsymbol{R}_{ij}$ is given by $\lambda_{ij} = \lambda_i p_{ij}$. When comparing the mean response times of the coded and uncoded systems, we compare the best achievable performances by considering the optimal probabilistic routing policy that minimizes the mean response time. For example, for the uncoded system, the optimal routing policy corresponds to sending a job of type i to one of the systematic servers of type i chosen uniformly at random. Obtaining the optimal routing policy for the coded system can be difficult. We provide some routing policies that perform well, both theoretically and practically in Section 4 and Section 5.

For the analysis of the mean response time, we assume that the number of coded servers is $n_{\text{coded}} = o(n)$. The case of $n_{\text{coded}} = \Theta(n)$ is both feasible and interesting. However, the comparison of the mean response times of the uncoded and coded systems becomes intractable in this regime. That is why we focus on $n_{\text{coded}} = o(n)$ for the response time characterization presented in Section 4. However, we provide some insights for $n_{\text{coded}} = \Theta(n)$ regime in our simulations in Section 5.

We also assume that $\lambda_i = \Theta(n)$ for all i. We believe that this is a mild assumption because the traffic experienced by a job is usually proportional to the servers allocated to the job type.

Organization of the paper. In Section 3, we analyze the service capacity region of the coded and uncoded system. Section 4 compare the mean response times of the coded and uncoded system. In Section 5.1, we present simulation results that corroborate our theoretical result given in Theorem 2 for a fixed arrival rate vector. In Section 5.2, we provide simulations with variable arrival rate vectors and show how negative correlation in traffic benefits the coded system. Finally, in Section 6 we provide a conclusion and directions for future work.

3 SERVICE CAPACITY REGION

The service capacity region is the set of job arrival rates for which the system is stable, that is, the cumulative arrival rate to any server does not exceed its service rate, as defined in Definition 1. Lemma 1 and Theorem 1 below provide the service capacity regions for the uncoded and coded multi-access cloud systems. We show that the service capacity region of the coded system is significantly larger than that of the uncoded system.

Lemma 1 (Uncoded Service Capacity Region)**.** *Consider an uncoded multi-access cloud system consisting of n servers and k job types, with $|\boldsymbol{S}_i| = \alpha_i n$ servers allocated to job type i. Then, the service capacity*

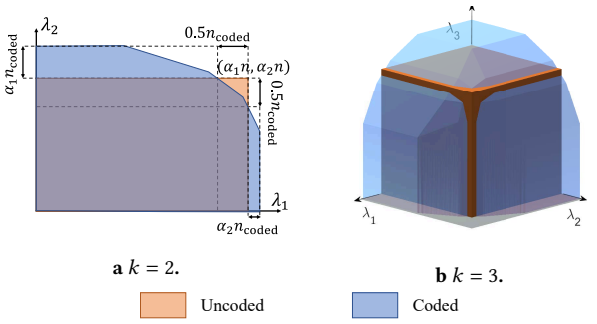


Figure 2: Service Capacity region of our system for $k = 2, 3$. The blue region represents the service capacity region for the coded system, and the orange one shows the service capacity region for the uncoded system. The service capacity region of the coded system expands in regimes where the traffic is skewed. However, it loses some area when all job types have similar and high arrival rates.

region is the set

$$\Lambda_{uncoded} = \{(\lambda_1, \ldots, \lambda_k) : 0 \le \lambda_i \le \alpha_i n, \forall i \in \{1, \ldots, k\}\}. \quad (3)$$

The service capacity region of the uncoded system is the hypercuboid of size k and length of the i-th side being $\alpha_i n$. This comes from the fact that the total service capacity of systematic servers of type i is $\alpha_i n$. The orange region in Fig. 2 illustrates the service capacity region of the uncoded system for $k = 2, 3$. Next, we characterize the service capacity region of the coded system in Theorem 1. We first characterize the service capacity region of the coded system using (4). However, for any vector $\boldsymbol{\lambda}$, it is difficult to verify whether $\boldsymbol{\lambda}$ satisfies (4) or not. Hence, we provide a simpler characterization via (5) that is a necessary and sufficient condition for stability.

Theorem 1 (Coded Service Capacity Region). *Consider a coded multi-access cloud system consisting of n servers and k job types. Let $\boldsymbol{R}_{ij}$ be the j-th element of $\boldsymbol{R}_i$, the set of recovery sets of type i. Then, the service capacity region is the set*

$$\Lambda_{coded} = \Big\{(\lambda_1, \ldots, \lambda_k) : \exists \lambda_{ij} \ge 0, \ s.t.\ \lambda_i = \sum_{j=1}^{|\boldsymbol{R}_i|} \lambda_{ij}, \forall i \in \{1, \ldots, k\},$$
$$\sum_{i=1}^{k} \sum_{j:\ell \in \boldsymbol{R}_{ij}} \lambda_{ij} \le 1, \forall \ell \in \{1, \ldots, n\}\Big\} \quad (4)$$

Let r_i denote the residual capacity *for type i jobs, given as $r_i \triangleq \alpha_i(n - n_{coded}) - \lambda_i$. Also define $r_i^+ \triangleq \max\{r_i, 0\}$, and $r_i^- \triangleq -\min\{r_i, 0\}$. Without loss of generality, assume that $r_1 \le r_2 \le \cdots \le r_k$. Then, an arrival rate vector $(\lambda_1, \ldots, \lambda_k) \in \Lambda_{coded}$ if and only if*

$$\min_{k_0 \in \{1, \ldots, k\}} \left\{ \frac{n_{coded} + \sum_{i=1}^{k_0} r_i^+}{k_0} \right\} \ge \sum_{i=1}^{k} r_i^-. \quad (5)$$

The proof sketch of Theorem 1 is given in Section 3.1 below, and the complete proof is given in [7].

For the special case of $k = 2$ and 3, the blue regions in Fig. 2 illustrate the service capacity region of the coded system. The service capacity region for $k = 2$ was previously derived in [1, 2]. However, [1, 2] did not consider the general case when the number of job types $k > 2$. From Fig. 2a, observe that there is an expansion in the service capacity region in the top-left and bottom-right areas, the regions where the traffic is heavy and skewed. The routing policy reduces the traffic in the higher loaded systematic servers by redirecting some of their load to the coded servers and less loaded systematic servers. The induced load balancing then allows the traffic of a job type to extend beyond the capacity of its systematic servers, leading to an expansion in the service capacity region.

However, the coded system loses portion of the service capacity region in the top right corner, where the traffic is heavy but not skewed. In this region, the traffic of all job types is usually greater than the total capacity of their systematic servers, hence the systematic servers are not enough on their own to serve all job types. In addition, the coded servers add a significant amount of redundancy which further increases the traffic, effectively making it unstable.

For $k = 2$, the total area of the regions gained is $\Theta(nn_{\text{coded}})$ since the maximum possible arrival rate of a job of type i can increase by $\Theta(n_{\text{coded}})$. Likewise, the area of lost region is $\Theta((n_{\text{coded}})^2)$ which makes the total area of service capacity region in the coded system to be $\left(\alpha_1\alpha_2 n^2 + \Theta(nn_{\text{coded}}) - \Theta((n_{\text{coded}})^2)\right)$. If the number of coded servers is small, i.e., $n_{\text{coded}} = o(n)$, then there is an effective gain in the service capacity region for the coded system.

3.1 Proof Sketch of Theorem 1

The proof of Theorem 1 contains two parts, the proofs of (4) and (5), and both follow a similar pattern. We first prove that if any arrival rate vector satisfies the property (the requirement in (4) or (5)), there is a way to stabilize the system. We then prove that if the property is not satisfied, no policy can stabilize the system. Below, we present the proof sketches of (4) and (5).

Proof of equation (4). We first prove that if any arrival rate vector $\boldsymbol{\lambda}$ satisfies (4), then there exists a way to stabilize the system. By the definition of the set Λ_{coded} given in Theorem 1, for any arrival rate vector $\boldsymbol{\lambda} \in \Lambda_{\text{coded}}$, there exists a set of λ_{ij}'s that satisfies (4). One routing policy for this arrival rate vector is to serve a job of type-i using servers in the recovery set $\boldsymbol{R}_{ij}$ with probability (λ_{ij}/λ_i). Under this policy, the total arrival rate to ℓ-th server is $\sum_{i=1}^{k} \sum_{j:\ell \in \boldsymbol{R}_{ij}} \lambda_{ij}$, where the inner sum represents the total arrival rate of tasks corresponding to the job of type i. Equation (4) ensures that for all $\ell \in \{1, \ldots, n\}$, the total arrival rate to ℓ-th server is less than 1, the service rate of the ℓ-th server, which implies that the system is stable.

The other direction of the result, i.e., any $\boldsymbol{\lambda}$ outside Λ_{coded} makes the system unstable, can be proven using standard techniques involving the *Strict separation theorem* and the *Strong law of large numbers* (see Chapter 4.2 of [25] for an example).

Proof sketch of equation (5). We first prove that for any arrival rate vector satisfying (5), there exists a policy that stabilizes the system. We use a water-filling argument where we initially serve a job only using its systematic servers. We start using the coded servers when the systematic servers reach their maximum capacity. Then, the right-hand side of (5) is the total excess service requirement after systematic servers are fully utilized. The left-hand side is the total service capacity available at the coded servers plus the

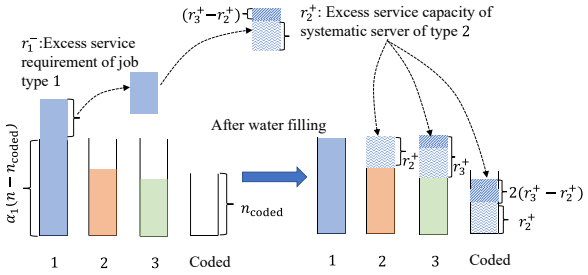


Figure 3: An illustration of the water filling method for $k = 3$ used in the proof of (5). The water-filling strategy involves first filling the systematic servers to the brink. The remaining service requirement is then served using the coded servers and the under-utilized systematic servers.

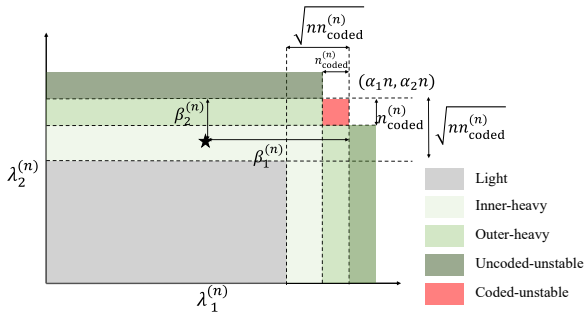


Figure 4: A pictorial representation of the traffic regimes for the case $k = 2$. The star represents a possible value of arrival rate vector and $\beta_i^{(n)}$ is represented by it's distance from the edge. The response time comparison of the coded and uncoded system for these regimes is provided in Theorem 2.

excess service capacity at the underutilized systematic servers. The system is stable if the excess service capacity exceeds or equals the excess service requirement. A pictorial representation of the water-filling argument is given in Fig. 3.

We then show that every arrival rate vector $\boldsymbol{\lambda} \in \Lambda_{\text{coded}}$ satisfies (5). The key idea is that for any $\boldsymbol{\lambda} \in \Lambda_{\text{coded}}$, there exist λ_{ij}'s satisfying (4) such that for any i either a job of type i only use its own systematic servers or type i systematic servers is used to serve only jobs of type i. This decomposition of $\boldsymbol{\lambda}$ ensures that the service requirement from the coded servers exceeds $(\sum_{i=1}^{k} r_i^-)$. Moreover, the service capacity available at the coded servers plus the under-utilized systematic servers is less than $(n_{\text{coded}} + \sum_{i=1}^{k_0} r_i^+)/k_0$ for any $k_0 \in \{1, \ldots, k\}$. The proof concludes by using the fact that the system is stable which implies equation (5) is true.

4 RESPONSE TIME CHARACTERIZATION

In this section, we compare the minimum mean response times of the coded and uncoded systems, denoted by $\mathbb{E}\left[T_{\text{coded}}^{(n)}\right]$ and $\mathbb{E}\left[T_{\text{uncoded}}^{(n)}\right]$ respectively. For the remainder of the paper, we use the notation superscript (n) to denote that the quantity depends on n. To characterize and compare the response times, we consider five traffic regimes listed and illustrated in Fig. 4. We formally define these regimes in Section 4.1, and then we compare the response times in each of these regimes in our main theorem in Section 4.2.

Note that Fig. 4 illustrates the *orders* of the edges of different areas as n becomes large. Hence, the slanted edges in Fig. 2a translates to vertical and horizontal lines in Fig. 4.

4.1 Traffic Regimes

Recall that jobs of type i arrive at the system according to a Poisson process with rate $\lambda_i^{(n)}$. Let $\beta_i^{(n)}$, referred to as the *slack capacity* of systematic servers of type i, be defined as

$$\beta_i^{(n)} = \alpha_i n - \lambda_i^{(n)}. \tag{6}$$

We define five traffic regimes based on the orders of the slack capacities of all job types. For ease of exposition, we first describe the regimes for the case where the system has two job types, i.e., $k = 2$, and then we generalize them to system with any value of k.

Traffic regimes for $k = 2$. Without loss of generality, we assume that $\beta_1^{(n)} \le \beta_2^{(n)}$.

Light regime: $\beta_i^{(n)} \ge 0$ for all $i \in \{1, 2\}$, and

$$\beta_i^{(n)} = \omega\left(\sqrt{nn_{\text{coded}}^{(n)}}\right), \quad \text{for all } i \in \{1, 2\}.$$

In this regime, the slack capacities are large for both job types.

Inner-heavy regime: $\beta_i^{(n)} \ge 0$ for all $i \in \{1, 2\}$, and

$$\beta_1^{(n)} = o\left(\sqrt{nn_{\text{coded}}^{(n)}}\right), \beta_1^{(n)} = \Omega\left(n_{\text{coded}}^{(n)}\right), \beta_2^{(n)} = \omega\left(\beta_1^{(n)}\right).$$

In this regime, type 1 jobs experience heavier traffic than type 2 jobs do, and thus type 1 jobs can benefit from a coded system.

Outer-heavy regime: $\beta_i^{(n)} \ge 0$ for all $i \in \{1, 2\}$, and

$$\beta_1^{(n)} = o\left(n_{\text{coded}}^{(n)}\right), \beta_2^{(n)} = \omega\left(n_{\text{coded}}^{(n)}\right).$$

Compared to the inner-heavy regime, the outer-heavy regime has an even heavier traffic for type 1 jobs, so the traffic is further skewed.

Uncoded-unstable: In this regime, the uncoded system is unstable while the coded system is stable.

Coded-unstable: In this regime, the coded system is unstable while the uncoded system is stable.

Traffic regimes for a general k. Without loss of generality, we assume that $\beta_1^{(n)} \le \ldots \beta_k^{(n)}$. To generalize the definitions of the five traffic regimes, we divide the k job types into two groups based on their slack capacities, (1) *beneficiaries*, and (2) *helpers*.

Intuitively speaking, the beneficiaries are the job types that experience heavier traffic, and the helpers are those who experience lighter traffic. Therefore, the beneficiaries can benefit from sending jobs to coded servers and systematic servers of the helpers. In contrast, the helpers only need their own systematic servers. Formally, consider the index i^* defined as

$$i^* = \max\left\{j : \left(j \sum_{i=1}^{j} \alpha_i\right) < 1\right\}. \tag{7}$$

Then we call job type in $\{1, 2, \ldots, i^*\}$ the beneficiaries, and job types in $\{i^*+1, \ldots, k\}$ the helpers. Note that in the special case where $k = 2$, the index $i^* = 1$, and thus job type 1 is the beneficiary and job type 2 is the helper. In another special case where $\alpha_i = 1/k$ for all i, the index i^* is around $\sqrt{k}$.

The choice of i^* in (7) has an intuitive explanation based on the stability of coded servers as follows. For any j, if there are j beneficiaries, then their total traffic is given by $\sum_{i=1}^{j}\left(\alpha_i n - \beta_i^{(n)}\right)$. We demonstrate in [7] that a good routing policy is as follows. Each helper assigns its jobs only to its own systematic servers. Each beneficiary assigns its jobs to its own systematic servers with probability $(1 - \Theta(n_{\text{coded}}^{(n)}/n))$, and to a recovery set consisting of $(k-j)$ systematic servers from helpers and j coded servers with the rest of the probability. Under this routing policy, the total traffic to the coded servers is $j \cdot \Theta\left(n_{\text{coded}}^{(n)}/n\right)\left(\sum_{i=1}^{j}\alpha_i n - \beta_i^{(n)}\right)$ which is $\Theta\left(n_{\text{coded}}^{(n)}\left(j\sum_{i=1}^{j}\alpha_i\right)\right)$. The choice of the index i^* in (7) then intuitively ensures the stability of coded servers.

Now, to define the traffic regimes, it is sufficient to look at the orders of $\beta_1^{(n)}$ and $\beta_{i^*+1}^{(n)}$. We define the traffic regimes as follows.

Light regime: $\beta_i^{(n)} \geq 0$ for all $i \in \{1, 2, \ldots, k\}$, and

$$\beta_i^{(n)} = \omega\left(\sqrt{n n_{\text{coded}}^{(n)}}\right), \text{ for all } i \in \{1, 2, \ldots k\}.$$

Inner-heavy regime: $\beta_i^{(n)} \geq 0$, for all $i \in \{1, 2, \ldots, k\}$, and

$$\beta_1^{(n)} = o\left(\sqrt{n n_{\text{coded}}^{(n)}}\right), \beta_1^{(n)} = \Omega\left(n_{\text{coded}}^{(n)}\right), \beta_{i^*+1}^{(n)} = \omega\left(\beta_1^{(n)}\right).$$

Outer-heavy regime: $\beta_i^{(n)} \geq 0$, for all $i \in \{1, 2, \ldots, k\}$, and

$$\beta_1^{(n)} = o\left(n_{\text{coded}}^{(n)}\right), \beta_{i^*+1}^{(n)} = \omega\left(n_{\text{coded}}^{(n)}\right).$$

The *coded-unstable* and *uncoded-unstable* regimes are defined in the same way as those given in Section 4.1. Note that the assumption of $n_{\text{coded}}^{(n)} = o(n)$ is necessary for defining the light, inner-heavy and outer-heavy regimes, but it is not required to define the uncoded-unstable and coded-unstable regimes. A sufficient condition for an arrival rate vector to lie inside the coded-unstable regime is given as $\beta_{i^*+1}^{(n)} = \omega(n_{\text{coded}}^{(n)})$ and $\beta_{i^*+1}^{(n)} \geq 0$. Likewise, a sufficient condition for an arrival rate vector to lie inside the uncoded-unstable regime is given as $|\beta_{i^*}^{(n)}| = o\left(n_{\text{coded}}^{(n)}\right), \beta_{i^*}^{(n)} \leq 0, \beta_{i^*+1}^{(n)} \geq 0$, and $\beta_{i^*+1}^{(n)} = \omega\left(n_{\text{coded}}^{(n)}\right)$. The proof of the sufficient conditions are given in [7].

4.2 Main Result

We state our main result in Theorem 2, which compares the response time in a coded system with that in an uncoded system in the five traffic regimes.

Theorem 2 (Response Time Comparison)**.** *Consider a multi-access cloud system consisting of n servers and k job types. Consider the minimum mean response times in a coded system and an uncoded system over all probabilistic job assigning policies, denoted by* $\mathbb{E}\left[T_{coded}^{(n)}\right]$ *and* $\mathbb{E}\left[T_{uncoded}^{(n)}\right]$*, respectively. Then we have the following comparison in the five traffic regimes:*

$$\text{Light regime: } \left|\mathbb{E}\left[T_{coded}^{(n)}\right] - \mathbb{E}\left[T_{uncoded}^{(n)}\right]\right| = o(1); \quad (8)$$

$$\text{Inner-heavy regime: } \mathbb{E}\left[T_{coded}^{(n)}\right] \leq \mathbb{E}\left[T_{uncoded}^{(n)}\right] - \omega(1); \quad (9)$$

$$\text{Outer-heavy regime: } \mathbb{E}\left[T_{coded}^{(n)}\right] = o\left(\mathbb{E}\left[T_{uncoded}^{(n)}\right]\right); \quad (10)$$

$$\text{Uncoded-unstable regime: } \mathbb{E}\left[T_{coded}^{(n)}\right] < \infty, \mathbb{E}\left[T_{uncoded}^{(n)}\right] = \infty; \quad (11)$$

$$\text{Coded-unstable regime: } \mathbb{E}\left[T_{coded}^{(n)}\right] = \infty, \mathbb{E}\left[T_{uncoded}^{(n)}\right] < \infty. \quad (12)$$

4.2.1 Proof sketch of Theorem 2. In this subsection, we provide a proof sketch of Theorem 2. We first analyze the mean response time of the uncoded system, and then analyze same for the coded one.

Uncoded system. The arrival rate of jobs of type i is $\lambda_i^{(n)}$, which is served using $\alpha_i n$ servers. One can prove that the optimal probabilistic routing policy is to serve an incoming job of type i using one of the systematic server of type i chosen uniformly at random. Under this optimal policy, all systematic servers of type i behave like independent $M/M/1$ queues with arrival rate $\lambda_i^{(n)}/(\alpha_i n)$ and unit service rate. The mean response time of a type i job is then $1/\left(1 - \frac{\lambda_i^{(n)}}{\alpha_i n}\right) = \frac{\alpha_i n}{\beta_i^{(n)}}$. Therefore, the mean response time of the uncoded system is given by

$$\mathbb{E}\left[T_{\text{uncoded}}^{(n)}\right] = \sum_{i=1}^{k} \frac{\lambda_i^{(n)}}{\sum_{\ell=1}^{k}\lambda_\ell^{(n)}} \frac{\alpha_i n}{\beta_i^{(n)}}. \quad (13)$$

Coded system. For the coded system, the routing probability is the key component deciding the mean response time. To better understand job routing policies in the coded system, we first give the Property 1 we observe for any stabilizing policy.

Property 1. *Consider any routing policy and let* $q_{i0}^{(n)}$ *be the total probability of sending a job of type i to a systematic server of type i. Then, the policy can stabilize the system only if*

$$1 - q_{i0}^{(n)} = O\left(\frac{n_{coded}^{(n)}}{n}\right), \textit{for all } i. \quad (14)$$

Property 1 states that the fraction of traffic that any job type can divert away from its own systematic servers is limited to a $O\left(n_{\text{coded}}^{(n)}/n\right)$ fraction. This provides a lower bound on the traffic that has to be served by systematic servers, which further leads to a lower bound on the mean response time. We use this lower bound in the analysis of the light regime. Moreover, in the heavy regimes where traffic is more skewed, we need to divert the traffic of heavily loaded job types as much as we can. In this case, Property 1 also provides a guideline for choosing a good job routing policy. The analysis of the individual regimes then proceeds as follows.

(1) *Light regime*: To prove (8), it suffices to show a lower bound,

$$\mathbb{E}\left[T_{\text{coded}}^{(n)}\right] \geq \mathbb{E}\left[T_{\text{uncoded}}^{(n)}\right] + o(1), \quad (15)$$

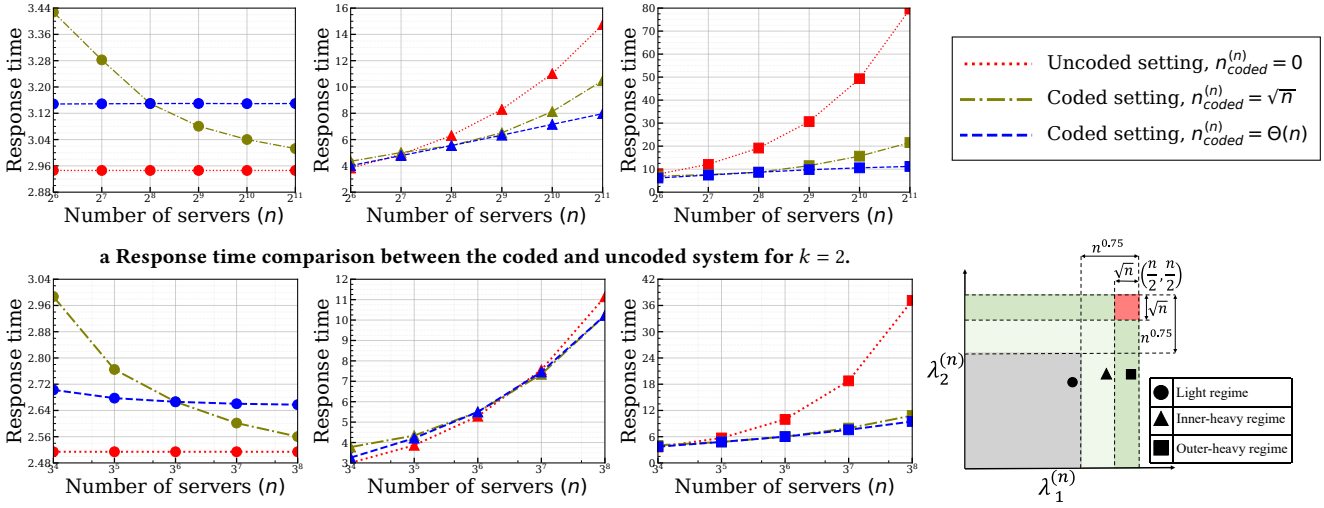


Figure 5: Comparison of the response times of the coded and uncoded systems. The different marker types (circle, triangle, square) represent arrival rates from different traffic regimes (light, inner-heavy and outer-heavy), as shown in the bottom-right illustration. The empirical standard error of the plotted points is $O(10^{-3})$. The coded servers significantly improve the system's performance in the inner-heavy and outer-heavy regimes. The coded and uncoded systems have similar response times in the light regime.

and an upper bound given by

$$\mathbb{E}\left[T^{(n)}_{\text{coded}}\right] \leq \mathbb{E}\left[T^{(n)}_{\text{uncoded}}\right] + o(1). \tag{16}$$

As mentioned earlier, the lower bound is proved using Property 1. To show the upper bound, note that it suffices to focus on a particular routing policy and show that its mean response satisfies (16). In the light regime, the slack capacity for every job type is large enough. Therefore, we consider the routing policy that assigns every job to its own systematic server. Computing the corresponding mean response time verifies (16).

(2) *Heavy regimes*: The analysis of the inner-heavy and outer-heavy regimes follow the same structure. In these regimes, the beneficiaries experience heavy traffic and have small slack capacities. To reduce the mean response times, we divert the traffic of beneficiaries from their systematic servers to the recovery sets that utilize coded servers as much as possible.

In fact, we consider the following routing policy. For some appropriate index $k^* \leq i^*$, we choose the routing probability $q^{(n)}_{i0} = 1$ for all $i > k^*$. For any $i \leq k^*$, we use the routing probability $q^{(n)}_{i0} = 1 - vn^{(n)}_{\text{coded}}/n$ and $q^{(n)}_{ik^*} = vn^{(n)}_{\text{coded}}/n$, for some constant v.

For any $i > k^*$, if the routing option is chosen corresponding to the probability $q^{(n)}_{ik^*}$, then a recovery set consisting of k^* coded servers and $(k - k^*)$ systematic servers of type $k^* + 1, \ldots, k$ respectively is chosen uniformly at random from all recovery sets satisfying the property. For any i, if the routing option is chosen corresponding to the probability $q^{(n)}_{i0}$, then a systematic server of type i is chosen uniformly at random. Upper-bounding the mean response time for this policy gives the upper bounds in (9) and (10)

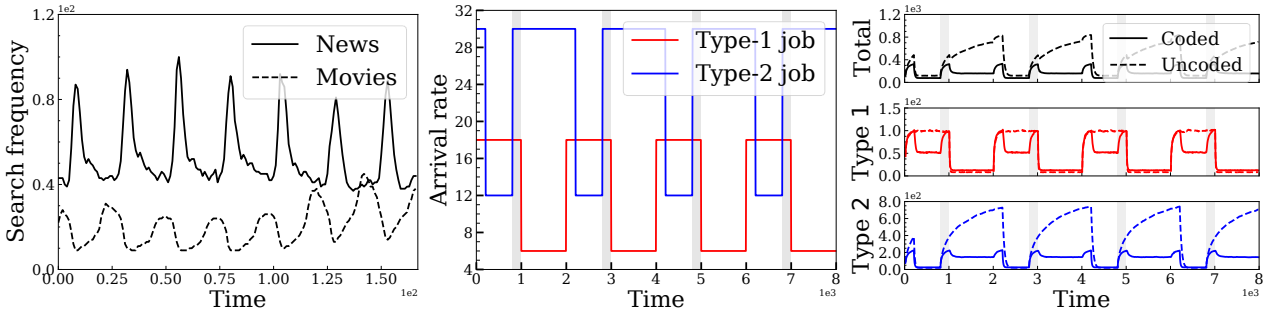
5 SIMULATION RESULTS

In this section, we present our simulation results to demonstrate the performance comparison between the uncoded and coded system under various traffic settings. In Section 5.1, we focus on arrival rates that are time-invariant. Our main goal is to demonstrate the performance comparison given in Theorem 2, but we have also investigated the choice of the $n^{(n)}_{\text{coded}}$ not covered in Theorem 2. In Section 5.2, we consider arrival rates that are time-varying, with a traffic pattern commonly observed in practical systems.

Before we get into the simulation settings, we first describe the routing policy we use in the simulations for the coded system.

Pseudo-optimal routing policy. Each queue behaves like a $M/M/1$ queue; hence the response time of each task is an exponentially distributed random variable. However, finding the optimal routing policy is non-trivial since the response time of a job is the maximum of the response time of its tasks, and the queues at each server are not independent. Moreover, the routing policy discussed in Section 4.2.1 does not perform well empirically for smaller values of n even though it works well asymptotically.

The difficulty in obtaining the optimal routing policy is the dependence among queues. We derive a policy that we call the *pseudo-optimal* routing policy by treating the queues as if they were independent. This approximation is based on the commonly observed phenomenon that queues are asymptotically independent in large systems [29]. With the independence assumption, one can calculate a job's response time as the expectation of maximum of independent exponential random variables is known. The pseudo-optimal routing policy is then the policy that minimizes the approximated mean response time. In our simulations, we find the pseudo-optimal routing policy numerically using Scipy Optimization libraries.



a Search frequency on Google **b Arrival rates used in simulation** **c Number of jobs in the system**

Figure 6: A comparison of the coded and uncoded systems for time-varying traffic in terms of total job in the system. The leftmost picture shows the search frequency of the words *news* and *movies* on Google, demonstrating a real-world example of negatively correlated traffic. The central figure shows the arrival rate vector as a function of time for a simplified system with two job types that has a negative correlation in traffic. The rightmost figure shows the number of jobs in the system. The presence of coded servers helps reduce the load from the heavier loaded systematic servers, making the coded system enjoy a lesser total job in the system on average.

5.1 Time-invariant Arrival Rates

In this subsection, we experimentally demonstrate the performance comparison between a coded system and an uncoded system. We provide simulations for systems with 2 and 3 job types. For these simulations, apart from using $n^{(n)}_{\text{coded}} = o(n)$, we also consider $n^{(n)}_{\text{coded}} = \Theta(n)$ and show the effect of large number of coded servers. We simulate until 10^8 jobs leave the system and average it over 50 runs to calculate the mean response time. Finally, based on our simulation results, we provide intuitions on how our main result would change for the case of $n^{(n)}_{\text{coded}} = \Theta(n)$.

Simulation for systems with* 2 *job types. We consider $n = 2^m$ servers, where we vary $m \in \{6, 7, \cdots, 11\}$. For the uncoded system, we calculate the response time theoretically. For the coded system, we consider two cases of $n^{(n)}_{\text{coded}} = o(n)$ and $n^{(n)}_{\text{coded}} = \Theta(n)$.

Fig. 5a shows numerical comparison between mean response of the coded and uncoded system for $k = 2$. The results for the case of $n^{(n)}_{\text{coded}} = o(n)$ resembles the theoretical results provided in Theorem 2. In the light regime, the coded systems with $o(n)$ coded servers perform similar to the uncoded system with a diminishing performance gap as n increases. In the inner-heavy and outer-heavy regimes, the coded system with $o(n)$ servers outperforms the uncoded system and the gap increases as n increases. Moreover, the performance gap between the coded and uncoded systems increases with the skewness in arrival rate, i.e., the coded system performs significantly better in the outer-heavy regime.

However, as illustrated in Fig. 5a, the results are slightly different when the number of coded servers increases as $n^{(n)}_{\text{coded}} = \Theta(n)$. In the light regime, the coded system with $\Theta(n)$ coded server performs worse than the uncoded system. The redundancy added by $\Theta(n)$ coded servers worsens the system. However, the performance gap does not change much as n increases as the traffic is light enough. The cost of redundancy is not substantial, and the coded system is worse only by an $\Theta(1)$ term. Compared to the light regime, the coded system with $\Theta(n)$ servers performs considerably better than both uncoded and coded systems with $o(n)$ servers in the inner-heavy and outer-heavy regimes. Because of the higher skew, the coded system with $\Theta(n)$ coded servers allows more uniform load balancing, thus greatly improving the performance.

Simulation for systems with* 3 *job types. We also perform experiments for system with three types of jobs. We consider $n = 3^m$ servers, where we vary $m \in \{4, 5, \cdots, 8\}$.

Fig. 5b shows the response time comparison of the coded and uncoded system for this simulation setup. For the light and outer-heavy regime, the trends in Fig. 5b is similar to the simulation results for the two-job type system and hence follows a similar reasoning. However, for the inner-heavy regime, the trends are slightly different. The coded system outperforms the uncoded system asymptotically; however, the difference is not as significant as seen in the simulation with two job types. One plausible reason is that the system has two beneficiaries and only one helper based on our arrival rate choice and allocation of servers. Hence, a slight skew in the arrival rate vector is insufficient to reduce the beneficiaries' load. However, there is enough skew in the outer-heavy region such that the coded system outperforms the uncoded system regardless of the number of coded servers.

Based on the simulations for systems with two or three job types, we conjecture that for $n^{(n)}_{\text{coded}} = \Theta(n)$, the inner-heavy and outer-heavy regimes would merge into a single heavy regime where the coded system would outperform the uncoded system. The light regime would also change, and instead of $o(1)$, the mean response time's of the coded and uncoded system can differ by $\Theta(1)$.

5.2 Time-Varying Arrival Rates

In practical systems such as Google search, the traffic of a job type often varies periodically. For example, as shown in Fig. 6a, search for news in Google [13] peaks during early hours, while the search for movies peaks during the night. To simulate such traffic patterns, we consider a system with two job types where the arrival rate of each job type is a square wave, as illustrated in Fig. 6b. The arrival

rates of the two job types have a similar period but are negatively correlated, i.e., if one job experiences higher traffic, the other job type should experience lesser traffic. We consider a total of $n = 60$ servers. In the coded system, 7 of these servers are coded servers. The remaining servers are distributed in the same proportion to the two job types for the coded and uncoded systems.

Fig. 6c shows a comparison between the mean number of jobs in the coded and uncoded systems. The top plot shows the total number of jobs in the system as a function of time, while the second and the third plot shows the number of jobs of type 1 and 2, respectively. When a job type is experiencing low traffic, the number of jobs of that type in the uncoded system is slightly less than that in the coded system. However, when a job type is experiencing heavy traffic, the number of jobs in the uncoded system is significantly higher than the coded system. This is because coding provides a load balancing effect where lightly loaded servers can be used to serve heavier traffic for job types.

6 CONCLUSION

This paper proposes the use of erasure-coded servers to handle traffic variations in heterogeneous jobs. We show that adding a few erasure coded servers significantly expands the capacity region of the coded system thereby improving the stability. We also compare the latency of the coded and uncoded systems and show that the coded system is better or at least comparable in most traffic regimes. The erasure-coded servers also improve the system's flexibility as the system can quickly adapt to changes in traffic, especially when the traffic is negatively correlated. Thus, at a slight cost of redundancy, our coded solution provides a general framework to improve the system's stability and latency.

There are substantial directions for future work. While our proposed coded system to handle k job types and its analysis holds for any k, in practice, a large k would be impractical because the decoding cost scales as $O(k^3)$. To handle large k, we could divide the servers into r subsystems with k/r job types in each subsystem. A large r will save the decoding cost, but it loses some flexibility offered by coding. In future work, we can determine the optimal choice of r and strategies to group the k job types into the r subsystems. Another future direction is to consider redundant replicas of a job that are sent to multiple recovery sets. A job is served when any one of the requests is served. Finally, we also plan to analyze our system for queue-length-based routing policies instead of probabilistic policies considered in this paper. Then the technical challenges in comparing the latency involve proving a complicated state-space collapse and lower bounding the mean response time.

ACKNOWLEDGMENTS

This work was supported in part by the NSF CCF #2007834 and #2045694, NSF CNS #2007733 and #2112471, NSF ECCS #2145713, and a Carnegie Bosch Institute Research Award. We thank Mor Harchol-Balter and Isaac Grosof for helpful discussions.

REFERENCES

[1] Mehmet Aktaş, Sarah E. Anderson, Ann Johnston, Gauri Joshi, Swanand Kadhe, Gretchen L. Matthews, Carolyn Mayer, and Emina Soljanin. 2017. On the service capacity region of accessing erasure coded content. In *Proc. Ann. Allerton Conf. Communication, Control and Computing*. Monticello, IL, USA, 17–24.

[2] Mehmet Aktaş, Gauri Joshi, Swanand Kadhe, Fatemeh Kazemi, and Emina Soljanin. 2021. Service Rate Region: A New Aspect of Coded Distributed System Design. *IEEE Trans. Inf. Theory* 67, 12 (Oct. 2021), 7940–7963.

[3] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. 2013. Effective Straggler Mitigation: Attack of the Clones. In *USENIX Symp. Networked Systems Design and Implem. (NSDI)*. USENIX Association, Lombard, IL, 185–198.

[4] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. 1999. Web caching and Zipf-like distributions: evidence and implications. In *Proc. IEEE Int. Conf. Computer Communications (INFOCOM)*, Vol. 1. New York, NY, USA, 126–134.

[5] Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz, and David A. Patterson. 1994. RAID: high-performance, reliable secondary storage. *ACM Comput. Surv.* 26 (1994), 145–185.

[6] Shengbo Chen, Yin Sun, Ulas¸ C. Kozat, Longbo Huang, Prasun Sinha, Guanfeng Liang, Xin Liu, and Ness B. Shroff. 2014. When queueing meets coding: Optimal-latency data retrieving scheme in storage clouds. In *Proc. IEEE Int. Conf. Computer Communications (INFOCOM)*. 1042–1050.

[7] Tuhinangshu Choudhury, Weina Wang, and Gauri Joshi. 2022. Tackling Heterogeneous Traffic in Multi-access Systems via Erasure Coded Servers. https://arxiv.org/abs/2207.03983

[8] Walfredo Cirne, Francisco Brasileiro, Daniel Paranhos, Luís Fabrício W. Góes, and William Voorsluys. 2007. On the efficacy, efficiency and emergent behavior of task replication in large distributed systems. *Parallel Comput.* 33, 3 (2007), 213–234.

[9] William Dally. 2015. High-performance Hardware for Machine Learning. *NIPS Tutorial* (July 2015).

[10] Jeffrey Dean and Luiz André Barroso. 2013. The Tail at Scale. *ACM Commun.* 56, 2 (Feb. 2013), 74–80.

[11] Elwyn Berkelamp. 1968. *Algebraic coding theory.* McGraw-Hill, New York, USA.

[12] Kristen Gardner, Mor Harchol-Balter, Alan Scheller-Wolf, Mark Velednitsky, and Samuel Zbarsky. 2017. Redundancy-d: The Power of d Choices for Redundancy. *Oper. Res.* 65, 4 (Aug. 2017), 1078–1094.

[13] Google. 2022. *Google Trends.* https://trends.google.com/trends

[14] Gauri Joshi and Dhruva Kaushal. 2021. Synergy via Redundancy: Adaptive Replication Policies and Fundamental Limits. *IEEE/ACM Trans. Netw.* 29, 02 (March 2021), 737–749.

[15] Gauri Joshi, Emina Soljanin, and Gregory Wornell. 2017. Efficient Redundancy Techniques for Latency Reduction in Cloud Systems. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 2, 2 (April 2017), 30.

[16] Yasaman Keshtkarjahromi, Yuxuan Xing, and Hulya Seferoglu. 2018. Dynamic Heterogeneity-Aware Coded Cooperative Computation at the Edge. In *Proc. IEEE Int. Conf. Network Protocols (ICNP)*. 23–33.

[17] Jack Kosaian, K. V. Rashmi, and Shivaram Venkataraman. 2019. Parity Models: Erasure-Coded Resilience for Prediction Serving Systems. In *Proc. ACM Symp. Operating Systems Principles (SOSP)* (Huntsville, Ontario, Canada). 30–46.

[18] Bin Li, Aditya Ramamoorthy, and R. Srikant. 2018. Mean-Field Analysis of Coding Versus Replication in Large Data Storage Systems. *ACM SIGMETRICS Perform. Evaluation Rev.* 3, 1, Article 3 (Feb. 2018), 28 pages.

[19] Mohammad Ali Maddah-Ali and Urs Niesen. 2016. Coding for caching: fundamental limits and practical challenges. *IEEE Communications Magazine* 54, 8 (Aug. 2016), 23–29.

[20] Ankur Mallick, Sophie Smith, and Gauri Joshi. 2021. Rateless Codes for Distributed Non-linear Computations. In *IEEE Int. Symp. Topics in Coding (ISTC)*. 1–5.

[21] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank Citation Ranking: Bringing order to the Web.* Technical Report. Stanford InfoLab.

[22] Michael Rabinovich and Oliver Spatscheck. 2002. *Web caching and replication.* Vol. 67. Addison-Wesley Boston, USA.

[23] Nihar B. Shah, Kangwook Lee, and Kannan Ramchandran. 2016. When Do Redundant Requests Reduce Latency? *IEEE Trans. Commun.* 64, 2 (2016), 715–722.

[24] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The Hadoop Distributed File System. In *IEEE Symp. Mass Storage Systems and Technologies (MSST)*. IEEE Computer Society, USA, 1–10.

[25] R. Srikant and Lei Ying. 2014. *Communication Networks: An Optimization, Control and Stochastic Networks Perspective.* Cambridge University Press, USA.

[26] Yin Sun, C. Emre Koksal, and Ness B. Shroff. 2017. On Delay-Optimal Scheduling in Queueing Systems with Replications. *arXiv:1603.07322 [cs.PF]* (March 2017).

[27] John N. Tsitsiklis and Kuang Xu. 2017. Flexible Queueing Architectures. *Operations Research* 65, 5 (July 2017), 1398–1413.

[28] Da Wang, Gauri Joshi, and Gregory W. Wornell. 2019. Efficient Straggler Replication in Large-Scale Parallel Computing. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 4, 2, Article 7 (April 2019), 23 pages.

[29] Weina Wang, Mor Harchol-Balter, Haotian Jiang, Alan Scheller-Wolf, and R. Srikant. 2019. Delay asymptotics and bounds for multitask parallel jobs. *Queueing Syst.* 91, 3-4 (April 2019), 207–239.