Brief Announcement: Distributed Download from an External Data Source in Asynchronous Faulty Settings

John Augustine
Indian Institute of Technology Madras, Chennai, India

Soumyottam Chatterjee
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Valerie King
University of Victoria, Canada

Manish Kumar
Indian Institute of Technology Madras, Chennai, India

Shachar Meir
Weizmann Institute of Science, Rehovot, Israel

David Peleg
Weizmann Institute of Science, Rehovot, Israel

Abstract

The distributed *Data Retrieval (DR)* model consists of k peers connected by a complete peer-to-peer communication network, and a trusted *external data source* that stores an array $\mathbf{X}$ of n bits ($n \gg k$). Up to βk of the peers might fail in any execution (for $\beta \in [0, 1)$). Peers can obtain the information either by inexpensive messages passed among themselves or through expensive queries to the source array $\mathbf{X}$. In the DR model, we focus on designing protocols that minimize the number of queries performed by any nonfaulty peer (a measure referred to as *query complexity*) while maximizing the resilience parameter β.

The Download problem requires each nonfaulty peer to correctly learn the entire array $\mathbf{X}$. Earlier work on this problem focused on *synchronous* communication networks and established several deterministic and randomized upper and lower bounds. Our work is the first to extend the study of distributed data retrieval to *asynchronous* communication networks. We address the Download problem under both the Byzantine and crash failure models. We present query-optimal deterministic solutions in an asynchronous model that can tolerate any fixed fraction $\beta < 1$ of crash faults. In the *Byzantine* failure model, it is known that deterministic protocols incur a query complexity of $\Omega(n)$ per peer, even under synchrony. We extend this lower bound to randomized protocols in the asynchronous model for $\beta \geq 1/2$, and further show that for $\beta < 1/2$, a randomized protocol exists with near-optimal query complexity. To the best of our knowledge, this is the first work to address the Download problem in asynchronous communication networks.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Byzantine Fault Tolerance, Blockchain Oracle, Data Retrieval Model, Distributed Download

Digital Object Identifier 10.4230/LIPIcs.DISC.2025.48

Related Version *Previous Version*: `https://arxiv.org/abs/2412.19649` [3]

Funding *John Augustine*: Supported by the Centre for Cybersecurity, Trust and Reliability (CyStar), IIT Madras.
Soumyottam Chatterjee: Work done while at IIT Madras supported by the Centre for Cybersecurity, Trust and Reliability (CyStar), IIT Madras.
Manish Kumar: Supported by the Centre for Cybersecurity, Trust and Reliability (CyStar), IIT Madras.

David Peleg: Venky Harinarayanan and Anand Rajaraman Visiting Chair Professor. The funds from this professorship enabled exchange visits between IIT Madras, India, and the Weizmann Institute of Science, Israel.

1 Introduction

1.1 Background and motivation

The Data Retrieval Model (DR) was first introduced in [2] to abstract the fundamental process of a group learning from a reliable external data source, where the data source is too large or it is too expensive to be learned individually (i.e., requires members of the group to collaborate), and some members of the group might crash during execution or act in other ways to deliberately sabotage the learning process. One key example of systems where this process takes place is *blockchain oracles* [6, 7]. We address this Oracle data delivery process in detail later and present a method for improving its performance using the DR model and the protocols designed in this work.

The DR model contains two entities: (i) a peer-to-peer network and (ii) an external data source in the form of an n bit array $\mathbf{X}$. There are k peers, up to β fraction of which may be faulty (and at least $\gamma = 1 - \beta$ fraction of which are nonfaulty). Each peer has access to the content of the array through queries. The general class of *retrieval problems* consists of problems $\mathsf{Retrieve}(f)$ requiring every peer to output $f(\mathbf{X})$ for some computable function f of the input array $\mathbf{X}$. In this work, we focus on the most fundamental retrieval problem, $\mathsf{Retrieve}(f_{id})$ where $f_{id}(\mathbf{X}) = \mathbf{X}$, referred to hereafter as the Download problem[1], where every peer needs to learn the entire input $\mathbf{X}$.

In [2] it is shown that *deterministic* Download in *synchronous* systems with Byzantine faults requires $\Omega(\beta n)$ queries, for every $\beta < 1$. This implies that in the presence of Byzantine faults, one cannot attain the ideal query complexity of $\frac{n}{\gamma k}$ without using randomization.

In this work, we consider Download protocols in the *asynchronous* setting for both the crash and Byzantine fault models. In the asynchronous Byzantine fault setting, we prove that, unlike the synchronous setting, where randomization can overcome the deterministic lower bound, $\Omega(n)$ queries per peer are required when $\beta \geq 1/2$, even for randomized protocols. We complement this lower bound with a protocol for the $\beta < 1/2$ regime that achieves a query complexity of $\tilde{O}\left(\frac{n}{(\gamma-\beta)k}\right)$, which, for a constant $\gamma - \beta$, is within log factors of the generic lower bound of $\Omega(n/\gamma k)$.

Turning our attention to the more benign setting of *crash* faults (i.e., where all peers are honest but some β fraction may stop functioning), the picture is brighter. For this model, it turns out that even in the asynchronous setting, one can get efficient *deterministic* Download protocols that achieve the optimal query complexity of $O\left(\frac{n}{\gamma k}\right)$, for any fraction $\beta < 1$ of crashes.

1.2 The Model

In the Data Retrieval (DR) model, the system consists of two components. The first is a collection of k peers, each equipped with a unique ID from the range $[1, k]$, connected by a complete communication network (or clique). The network provides *peer-to-peer message passing*, namely, every peer can send at time t a (possibly different) message of size at most ϕ bits to each other peer.

[1] It is fundamental since every retrieval problem $\mathsf{Retrieve}(f)$ can be solved by first performing download and then locally computing $f(\mathbf{X})$.

The second component of the DR model is an *external data source*. The *source* stores an n-bit input array $\mathbf{X} = \{b_1, \ldots, b_n\}$. It provides the peers with read-only access, allowing each peer to retrieve the data through queries of the form $\texttt{Query}(i)$, for $1 \leq i \leq n$. The answer returned by the source would then be b_i, the i^{th} element in the array. This type of communication is referred to as *source-to-peer* communication.

We consider asynchronous communication, where any communication (both among peer-to-peer and source-to-peer) can be delayed by any finite amount of time. For randomized protocols, we use the following notion of *cycles*.

Cycles. In the asynchronous model, there is no global notion of *rounds*, as each peer operates at a different pace. Nevertheless, to describe our protocols and analyze their performance, it is convenient to divide the *local* execution of each peer μ into (varying time) *cycles*. Each such local cycle consists of the following stages.

- Sending (0 or more) queries and getting answers.
- Sending (0 or more) messages.
- Waiting to receive messages.

We assume that local computation takes 0 time and can be performed at any point in a cycle. Moreover, when waiting for messages, after every message is received, the peer can adaptively decide whether to keep waiting for an additional message or continue to the next cycle. Note that the local cycle r of peer μ might coincide with a different local cycle r' of another peer μ'.

In the absence of global time units, it is convenient to break the time axis into "virtual blocks" by defining t^r, for integer $r \geq 1$, as the first time any peer started its local cycle r.

Every message is of size at most ϕ bits, where ϕ is a system parameter. Note that throughout the paper, we either set ϕ to a specific value, or leave it as a parameter, in which case increasing the message size parameter ϕ would result in faster protocols.

The adversary. Our analysis uses the notion of an *adversary*, representing the adverse conditions in which the system operates, including the asynchronous communication and the possibility of failures.

The adversary has two types of operation. First, it can fail up to βk peers, under the restriction that it can only fail a peer between its cycles (or before the first cycle), meaning that a peer can make random decisions in its current cycle without the adversary being able to react until the end of the cycle. Second, it can set the time $t^r_{\mu,\mu'}$ it takes a message sent by peer μ in its local cycle r to reach peer μ', under the restriction that it must set the time $t^r_{\mu,\mu'}$ for every pair of peers μ, μ', before time t^r. In other words, the adversary must set the latency of each message sent during a cycle r before any peer starts cycle r. The adversary can also decide when every peer starts its execution (i.e., we do not assume a simultaneous start). Note that in the case of deterministic protocols, the notion of cycles is irrelevant, and we consider a standard adversary that can fail a peer at any point of the execution and can delay messages for any finite amount of time.

The adversary $\mathcal{A}$ selects the input data and determines the failure pattern of the peers. In the *crash* failure model, the adversary's power is limited to crashing some of the peers in every execution of the protocol. Once a peer crashes, it stops its local execution of the protocol arbitrarily and permanently. This could happen *in the middle of operation*, e.g., after the peer has already sent some, but perhaps not all, of the messages it was instructed by the protocol to send out at a given point in time. In contrast, in the *Byzantine* failure model, a failed peer can deviate from the protocol in arbitrary ways. We assume that the adversary

can fail at most βk peers, for some given[2] $\beta \in [0, 1)$. We let $\gamma = 1 - \beta$, so there is (at least) a γ fraction of nonfaulty peers in every execution. Denote the set of faulty (respectively, nonfaulty) peers in the execution by $\mathcal{F}$. (resp., $\mathcal{H}$). We assume that the adversary knows the protocol and hence can simulate it (up to random coins).

We concentrate on the following complexity measures.

***Query* Complexity ($\mathcal{Q}$):** the maximum number of bits queried by a nonfaulty peer during the execution.

***Time* Complexity ($\mathcal{T}$):** the time it takes for the protocol to terminate.

***Message* Complexity ($\mathcal{M}$):** the total number of messages sent by nonfaulty peers during the execution.

We assume that queries to the source are the more expensive component in the system, so we focus mainly on optimizing the query complexity $\mathcal{Q}$. Measuring the *maximum* cost per peer (rather than the *total* cost) gives priority to a balanced load of queries over the nonfaulty peers.

Let us now formally define the Download problem. Consider a DR network with k peers, where at most βk can be faulty, and a source that stores a bit array $\mathbf{X} = [b_1, \ldots, b_n]$. Each peer is required to learn $\mathbf{X}$. Formally, each nonfaulty peer μ outputs a bit array res^μ, and it is required that, upon termination, $res^\mu[i] = b_i$ for every $i \in \{1, \cdots, n\}$ and $\mu \in \mathcal{H}$.

In the absence of failures, this problem can be solved by sharing the task of querying all n bits evenly among the k peers, yielding $\mathcal{Q} = \Theta(n/k)$. The message complexity is $\mathcal{M} = \tilde{O}(nk)$, assuming small messages of size $\tilde{O}(1)$, and the time complexity is $\mathcal{T} = \tilde{O}(n/k)$ since $\Omega(n/k)$ bits need to be sent along each communication link when the workload is shared.

1.3 Contributions

We present the Download problem in asynchronous communication networks, under both crash and Byzantine failures settings. In the crash-fault setting, our deterministic results are optimal w.r.t. to the resilience (for any $\beta < 1$) and query complexity. Notice that this optimality also holds for randomized algorithms. In the Byzantine failure setting, we provide deterministic and randomized lower bounds as well as upper bounds. The main results are the following (more details can be found in [3]).

(1) Deterministic Download **in Crash-Fault:** We present a deterministic protocol for solving Download problem in the asynchronous setting with at most $f < k$ crash faults ($\gamma = 1 - f/k$) with $\mathcal{Q} = \Theta(\frac{n}{\gamma k})$, $\mathcal{T} = O\left(\frac{n}{\phi} + \log_{k/f}(\phi)\right)$ and $\mathcal{M} = O(nk^2)$ where ϕ is the message size. Our result achieves the optimal query complexity for any fraction of crash fault, $\beta < 1$.

(2) Deterministic Lower Bound in Byzantine Fault: We show that for $\beta \geq 1/2$, every deterministic asynchronous Download protocol that is resilient to Byzantine faults requires $\mathcal{Q} = n$.

(3) Deterministic Download **in Byzantine Fault:** We show that for $\beta < 1/2$, there exists a deterministic asynchronous protocol that solves Download with $Q = O(\beta n)$, $\mathcal{T} = O\left(\frac{\beta n}{\phi}\right)$ and $\mathcal{M} = O(f \cdot n)$.

(4) Randomized Lower Bound in Byzantine Fault: We show that for any randomized asynchronous Download protocol where $\beta \geq 1/2$, there does not exist any execution in which every peer queries less than or equal $n/2$ bits.

[2] We do not assume β to be a fixed constant (unless mentioned otherwise).

(5) 2-cycle Randomized Download **in Byzantine Fault:** We present a 2-cycle asynchronous randomized protocol for Download with $\mathcal{Q} = O\left(\sqrt{\frac{n}{\gamma-\beta}} + \frac{n \log n}{(\gamma-\beta)k}\right)$ and $\mathcal{M} = O(k^2)$ where $\beta \leq 1/2$, and the message size is $\phi = O(\frac{n}{\gamma k})$.

(6) Randomized Download **in Byzantine Fault:** We present a $O\left(\log\left(\frac{\gamma k}{\ln n}\right)\right)$-cycle protocol that computes Download whp in the point-to-point model having expected query complexity $\mathcal{Q} = O\left(\frac{n \log n}{(\gamma-\beta)k}\right)$ and $\mathcal{M} = O\left(\log\left(\frac{\gamma k}{\ln n}\right) k^2\right)$ where $\beta \leq 1/2$, and the message size is $\phi = O(n)$.

To the best of our knowledge, this work is the first to study retrieval problems in the Data Retrieval (DR) model under asynchronous communication. The DR model has previously been explored in synchronous networks, most notably in [2] and the companion paper [4]. In particular, [2] introduced the Download problem, motivated by practical applications such as Distributed Oracle Networks (DONs), which form a crucial component of blockchain systems and employ protocols like OCR and DORA[6, 7].

2 Application: Efficient Blockchain Oracles

Blockchain systems [8] have seen a rise in popularity due to their ability to provide both transparency and strong cryptographic guarantees of agreement on the order of transactions, without the need for trusted third-party entities. More general computational abilities have also been well sought out for blockchains. Smart contracts [10] fulfill that need by providing users of the blockchain a way to run programs *on* the blockchain that ensures reliable and deterministic execution while providing transparency and immutability of both the code of the program and its state(s). Note that since the execution is required to be deterministic, i.e., every node must produce the same result, smart contracts are restricted to accessing *on-chain* data (that has been agreed upon), as *off-chain* data may introduce non-determinism to the execution.

Blockchain oracles [1, 5, 9] are components of blockchain systems that provide multiple services that support and extend the functionality of smart contracts (and other on-chain entities). The most important and fundamental service a blockchain oracle provides is bridging between the on-chain network and off-chain resources [6, 7], providing smart contracts access to external data without introducing non-determinism into their execution. We focus on this service and artificially consider it to be the sole responsibility of a blockchain oracle. In the remainder of this section, we explain in detail a possible application of the DR model and the Download problem for improved query efficiency within the context of *blockchain oracles*.

Blockchain oracles general structure. Blockchain oracles consist of an on-chain component and an off-chain component. The off-chain component encompasses the different data sources that store the required external information (e.g., stock prices, weather predictions) and the network of nodes in charge of retrieving that information and transmitting it to the on-chain component. The on-chain component can be thought to be (but is not necessarily) a smart contract that is responsible for verifying the validity of the report, making the information public on the blockchain it is hosted on, and using it for its execution.

Formally, the off-chain component consists of two parts: an asynchronous[3] *oracle network*, with peers (nodes) v_i, $i \in [1, k]$, capable of exchanging direct messages among themselves, and *data sources* DS_j, $j \in [1, m]$, each storing an array $\mathbf{X}_j$ of n variables in which the on-chain

[3] The network is sometimes assumed to be *partially synchronous* in blockchain oracles.

component is interested. Each peer can read the i-th cell from the j-th data source DS_j by invoking $\texttt{Query}(i, j)$. A fraction of up to $\beta_t \leq 1/3$ of the peers may be Byzantine, and a fraction of up to $\beta_d \leq 1/2$ of the m data sources may be Byzantine. Denote the on-chain component by SC.

The goal of blockchain oracles, as mentioned above, is to pull information from external sources and push a final value on-chain. There are a few difficulties that may arise when trying to develop such a system. First, it might be the case that different data sources report slightly different values (e.g., prices of a specific stock), even if all of them act honestly. Moreover, corrupted data sources might provide false and even inconsistent values (providing some nodes with value α and another with α'). The system needs to pick a final value in a way that (1) represents the range of honest values pulled from honest data sources and (2) malicious players (both data sources and peers) cannot force the system to pick a final value that does not represent the range of honest values.

The Oracle Data Delivery (ODD) problem. Denote by $\mathcal{H}_{ds}$ the set of honest data sources. Let $h_{\min}(i) = \min_{j \in \mathcal{H}}\{\mathbf{X}_j[i]\}$ and $h_{\max}(i) = \max_{j \in \mathcal{H}}\{\mathbf{X}_j[i]\}$. The *honest range* of i is the range $\sigma(i) = [h_{\min}(i), h_{\max}(i)]$. The *ODD problem* requires the on-chain SC to publish an array of values res to the target blockchain such that $res[i] \in \sigma(i)$, for every $i \in [n]$.

A blockchain oracle protocol can generally be split into three distinct steps: (1) collecting data, (2) reaching an agreement on the collected data, and (3) deriving and publishing a final value. Note that this abstraction is the minimum required abstraction to capture the operation of blockchain oracle protocols such as OCR [6] and DORA [7][4].

We now show how our Download protocols can be used to significantly reduce the cost of the *Oracle Data Collection (ODC)*, i.e, step (1) of blockchain oracles.

Improving ODC by blockchain oracles via Download**.** Current protocols perform the data collection step by the following ODC process:
For every node:

- Pick $2m\beta_d + 1$ data sources into a set ADS.
- Perform $o_{i,j} \leftarrow \texttt{Query}(i, j)$, for every $i \in [1, n]$ and $j \in ADS$.
- Calculate the median $o_i \leftarrow median(\{o_{i,j} \mid j \in ADS\})$ and proceed to step (2).

The results of [6, 7], cast in our abstract formulation, yield the following result.

▶ **Theorem 1.** **[6, 7]** *The ODC process guarantees that $o_i \in \sigma(i)$ for every $i \in [1, n]$ and has total query cost $O(mnk)$ and worst case individual query cost $\mathcal{Q} = O(mn)$.*

Instead, we propose utilizing the guarantees of Download protocols, namely, that for an honest data source DS_j, the output of each peer is exactly $\mathbf{X}_j$, to construct the following modification of the ODC steps.

- For every node, pick $2m\beta_d + 1$ data sources into a set ADS.
- For every data source $j \in ADS$, run a Download protocol (denote the result for cell i from data source j by $o_{i,j}$).
- Calculate the median $o_i \leftarrow median(\{o_{i,j} \mid j \in ADS\})$ and proceed to step (2).

[4] These protocols have many additional technical aspects, different structures, and different ways of handling steps (2) and (3). As our focus is on improving step (1), we may w.l.o.g. assume the abstract structure.

It is easy to verify that this modified construction yields the following.

▶ **Theorem 2.** *The* Download*-based ODC process guarantees* $o_i \in \sigma(i)$ *for every* $i \in [1, n]$ *and takes* $\tilde{O}(mn)$ *total queries and* $\mathcal{Q} = \tilde{O}(mn/k)$ *w.h.p.*

Note that the Download protocol presented in this paper assumes a binary input array, but this can be extended to numbers via a relatively simple extension. However, it is important to note that our solution relies on the following restrictive assumption. For two honest peers v, v', if both v and v' issue the query $\texttt{Query}(i, j)$, then they get the same result, for every $i \in [1, n]$ and honest data source j (i.e., the data does not change if queried at different times). Getting rid of this assumption and solving the problem efficiently for dynamic data is left as an open problem for future study.

References

1 John Adler, Ryan Berryhill, Andreas Veneris, Zissis Poulos, Neil Veira, and Anastasia Kastania. Astraea: A decentralized blockchain oracle. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1145–1152, 2018. doi:10.1109/Cybermatics_2018.2018.00207.

2 John Augustine, Jeffin Biju, Shachar Meir, David Peleg, Srikkanth Ramachandran, and Aishwarya Thiruvengadam. Byzantine Resilient Distributed Computing on External Data. In Dan Alistarh, editor, *38th International Symposium on Distributed Computing (DISC 2024)*, volume 319 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 3:1–3:23, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DISC.2024.3.

3 John Augustine, Soumyottam Chatterjee, Valerie King, Manish Kumar, Shachar Meir, and David Peleg. Distributed download from an external data source in faulty majority settings. *CoRR*, abs/2412.19649, 2024. doi:10.48550/arXiv.2412.19649.

4 John Augustine, Soumyottam Chatterjee, Valerie King, Manish Kumar, Shachar Meir, and David Peleg. Distributed Download from an External Data Source in Byzantine Majority Settings. In *DISC*, 2025. These proceedings.

5 Lorenz Breidenbach, Christian Cachin, Benedict Chan, Alex Coventry, Steve Ellis, Ari Juels, Farinaz Koushanfar, Andrew Miller, Brendan Magauran, Daniel Moroz, Sergey Nazarov, Alexandru Topliceanu, Florian Tram‘er, and Fan Zhang. Chainlink 2.0: Next steps in the evolution of decentralized oracle networks. Technical report, Chainlink Labs, 2021.

6 Lorenz Breidenbach, Christian Cachin, Alex Coventry, Ari Juels, and Andrew Miller. Chainlink off-chain reporting protocol. Technical report, Chainlink Labs, 2021.

7 Prasanth Chakka, Saurabh Joshi, Aniket Kate, Joshua Tobkin, and David Yang. DORA: distributed oracle agreement with simple majority. *CoRR*, abs/2305.03903, 2023. doi:10.48550/arXiv.2305.03903.

8 Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *whitepaper*, May 2009.

9 Supra Research. Supra’s blockchain infrastructure stack. Whitepaper, Supra Labs, November 2024. URL: https://supra.com/documents/SupraTech-Whitepaper.pdf.

10 Nick Szabo. Formalizing and securing relationships on public networks. *First Monday*, 2, 1997. doi:10.5210/FM.V2I9.548.