Demo: HyperPubSub: a Decentralized, Permissioned, Publish/Subscribe Service using Blockchains

Nejc Zupan[1], Kaiwen Zhang[1,2,3], Hans-Arno Jacobsen[2]

[1]Technical University of Munich
[2]Middleware Systems Research Group
[3]University of Toronto

Abstract

Since the introduction of Bitcoin in 2008, blockchain systems have evolved immensely in terms of performance and usability. There is a massive focus on building enterprise blockchain solutions, with providers such as IBM and Microsoft already providing Blockchain-as-a Service (BaaS). To facilitate the adoption of blockchain technologies across various business verticals, we argue that middleware plays an integral role in accelerating the development of automated business processes (i.e., smart contracts). We argue that decentralized messaging is a key requirement of many distributed applications and should be provided as a reusable blockchain middleware. Our system, called HyperPubSub, provides decentralized publish/subscribe messaging for a multi-federated, permissioned, environment. HyperPubSub provides secure and privacy-preserving messaging, which is audited using blockchains for validation and monetization purposes. We demonstrate our implementation using Kafka and Hyperledger.

CCS Concepts • **Software and its engineering → Publish-subscribe / event-based architectures**; • **Computer systems organization → Distributed architectures**; *Real-time system architecture*;

Keywords Hyperledger, Kafka, Publish/Subscribe, Blockchain

ACM Reference Format:
Nejc Zupan[1], Kaiwen Zhang[1,2,3], Hans-Arno Jacobsen[2]. 2017. Demo: HyperPubSub: a Decentralized, Permissioned, Publish/Subscribe Service using Blockchains. In *Proceedings of Middleware Posters and Demos '17: Proceedings of the Posters and Demos Session of the 18th International Middleware Conference (Middleware Posters and Demos '17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3155016.3155018

1 Introduction

Publish/Subscribe is well studied and widely adopted in distributed systems, and has traditionally focused on performance and reliability. Typically, clients are assumed to trust the pub/sub agents (called brokers) to be functioning correctly. In practice, this assumption cannot hold true in multi-federated environments, with limited trust across domains.

Since blockchains are ideal for regulating interactions in a low trust environment, we argue that there is a need to support publish/subscribe messaging in such blockchain platforms. To this end, we propose HyperPubSub, a decentralized pub/sub system which provides secure and privacy-preserving messaging. Pub/sub clients can audit the messaging service by querying the blockchain, validate past operations, and even monetize pub/sub operations. We demonstrate our concept using an implementation built with Kafka and Hyperledger.

Middleware Posters and Demos '17, December 11–15, 2017, Las Vegas, NV, USA

ACM ISBN 978-1-4503-5201-7/17/12…$15.00
https://doi.org/10.1145/3155016.3155018

2 Background

2.1 Blockchain systems

Blockchain was introduced in 2008 [4] and quickly became very popular as a new approach for digital currency. It introduced a peer-to-peer electronic cash system where all transactions are public but encrypted, therefore providing anonymity to the users and removed the need for central authority with Proof-of-Work-based validation.

There is a noticeable increase in blockchain systems which do not focus on cryptocurrency (e.g., Ethereum, Hyperledger, Corda) [1], and provide a business platform for streamlining processes with the issuance of smart contracts. In particular, some of these systems are notable for being private ledgers, which limit access to permissioned entities and rely on weaker consensus algorithms in exchange for enhanced performance.

A good example of such implementation is Hyperledger: a private ledger platform with a modular architecture, designed for confidentiality, resiliency, flexibility, and scalability [1]. The extensible architecture is a unique feature, which provides an open blockchain platform enabling developers to easily deploy custom chaincode (i.e., smart contracts). This is realized by the Hyperledger Composer component which supports a purpose-built modeling language for defining a business network, multiple ways of integrating blockchains with other services, and a platform for developing, testing and deploying custom business logic. The main system maintaining the blockchain using the PBFT [2] algorithm is called Hyperledger Fabric [1].

2.2 Publish/subscribe systems

Publish/Subscribe is a messaging paradigm which allows data producers (publishers) to send data (publication) matching the interests of data consumers (subscribers) [5]. The filtering and forwarding of publications is provided by pub/sub agents (brokers). Pub/Sub systems provide scalability, high throughput, and anonymity between subscribers and publishers. However, publishers and subscribers are assumed to trust the brokers to reliably deliver publications without compromising their privacy.

Kafka is a distributed messaging system that was developed for collecting and delivering high volumes of log data with low latency [3]. It offers a combination of traditional log aggregation and messaging while providing a scalable, distributed, and high

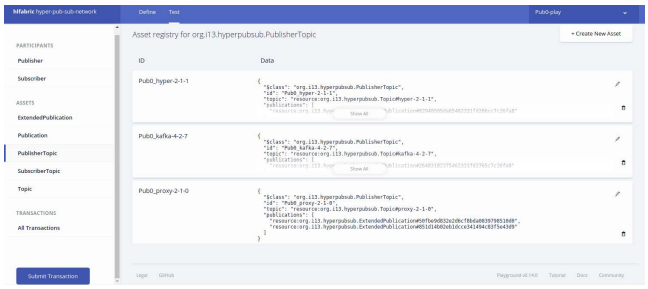

Figure 1. Client Web Interface

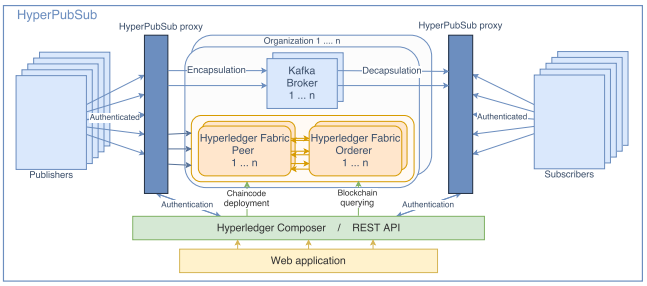

Figure 2. HyperPubSub Reference Architecture

throughput system. Popularly used in industry, Kafka provides features beyond pub/sub with a set of connector APIs for connecting various systems through *sinks* and *sources*, multiple messaging models [6], ability to efficiently process streams of data in real time and persistently store them in a distributed replicated cluster, while maintaining high throughput.

3 System Requirements and Features

HyperPubSub is designed with the following objectives in mind:

Pub/sub verification: For each pub/sub operation (e.g., publications and subscriptions), HyperPubSub keeps a validated record on the blockchain. Due to the inherent immutability of blockchain records, in addition to their correctness enforced via smart contracts, HyperPubSub provides verifiability of past pub/sub operations, which increases the trust in the service when deployed in a multi-federated environment. As illustrated in Figure 1, clients (e.g., publishers and subscribers) can check for complete publication delivery to intended recipients, and check received publications for possible tampering.

Low integration overhead: The performance of pub/sub operations, namely publishing and subscribing, should not be degraded due to the blockchain integration. We wish to minimize the impact of blockchains on the latency and throughput of online operations, while still providing a way to query and verify fast operations.

Access control: Fully configurable rules allow fine-grained control over the blockchain data that publishers and subscribers are allowed to access. In particular, they should only be allowed to query information which is relevant to their pub/sub activities. For instance, a subscriber should be able to query the blockchain for past publications which are matching its subscriptions.

Decentralized monetization: HyperPubSub provides a secure way for subscribers and publishers to respectively send and receive payments for data sent over the pub/sub service, without violating the decoupling properties of the system. By leveraging the blockchain information, clients can accurately track the amount of money they are owed/owing. Furthermore, the blockchain records are anonymized to prevent the identity of publishers/subscribers to be exposed to one another.

4 Demo Architecture

As seen in Figure 2, our proposed architecture consists of three major components: Hyperledger, Kafka, and Proxies.

Publishers and subscribers communicate to Kafka via the proxies, which intercept the operations and transmit them to Hyperledger. Pub/Sub operations are recorded in the blockchain in an out-of-band manner, which does not interfere with the online filtering and forwarding operations of Kafka, while still providing a validation mechanism for past operations.

Proxies provide an interface layer which allows the internal pub/sub component and the blockchain to be swapped with alternatives. To allow for a wide variety of clients, communication with the proxy is modeled in gRPC, which is supported by over 10 commonly used programming languages. An operation received by a proxy is encapsulated and published to Kafka.

To retrieve publication, the subscriber connects to the proxy, which opens a stream for a specified topic with Kafka. Upon data receipt in the proxy, the publication is processed and forwarded to the client over an already established stream.

In order to validate the operations sent from Kafka, we have modeled the pub/sub semantics using smart contracts in Hyperledger Composer. This design allows clients to audit the system and check for complete delivery of publications and validate consumed data. Clients use certificated issued by Fabric's CA to authenticate to the system, whereas the access control system limits their access to allow their querying only relevant data thus providing permissioned environment.

Concerning the choice of systems to integrate for our reference implementation, we chose Kafka and Hyperledger Fabric. Reasons are three-fold: (i) We opt for Hyperledger over Ethereum because of its consensus mechanism with lower inclusion time, resulting in higher throughput and thus it's better suited to match Kafka's performance; (ii) Hyperledger Fabric provides certificate authority service that all system components can use to ensure trust across complete system; and (iii) Good natural mapping between Kafka and Hyperledger components allow for efficient deployment.

5 Acknowledgements

Research is supported by Alexander von Humboldt Foundation. We thank Richard Hull (IBM) for helpful discussions on Hyperledger.

References

[1] 2016. *Hyperledger Whitepaper.* Technical Report. Hyperledger. http://www.the-blockchain.com/docs/Hyperledger%20Whitepaper.pdf
[2] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186.
[3] Jay Kreps, Neha Narkhede, and Jun Rao. 2011. Kafka: a Distributed Messaging System for Log Processing. (2011).
[4] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).
[5] Lukasz Opyrchal and Atul Prakash. 2003. *Publish Subscribe Middleware.* Springer US, Boston, MA, 249–285. https://doi.org/10.1007/978-1-4615-0389-7_8
[6] Hein Ph. 2014. Apache Kafka: Next Generation Distributed Messaging System. 03, 47 (2014), 9478–9483.