Clock Synchronisation in Provenance Systems for Serverless Computing Platforms

Project - Part 1 report

submitted in partial fulfilment of the requirements for the degree of

Master of Technology

in

Computer Science and Engineering

by

Ishan Sharma

(18CS30020)

Under the supervision of

Prof. Sandip Chakraborty

Department of Computer Science and Engineering

Indian Institute of Technology Kharagpur

Autumn Semester, 2023-23

November, 2022

DECLARATION

I certify that

(a) The work contained in this report has been done by me under the guidance of my supervisor.

(b) The work has not been submitted to any other Institute for any degree or diploma.

(c) I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

(d) Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Date: November, 2022 (Ishan Sharma)
Place: Kharagpur (18CS30020)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

KHARAGPUR - 721302, INDIA

CERTIFICATE

This is to certify that the project report entitled "**Clock Synchronisation in Provenance Systems for Serverless Computing Platforms**" submitted by **Ishan Sharma** (Roll No. 18CS30020) to the Indian Institute of Technology Kharagpur towards partial fulfilment of requirements for the award of the degree of Master of Technology in Computer Science and Engineering is a record of bonafide work carried out by him under my supervision and guidance during Autumn Semester, 2023-23.

Date: November, 2022
Place: Kharagpur

Prof. Sandip Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur - 721302, India

Acknowledgements

I would like to thank my Thesis supervisor *Prof. Sandip Chakraborty* and *Dr. Subhrendu Chattopadhyay* for directing, guiding, and supporting me throughout the project. I would also like to extend my gratitude towards *Rajat*, *Vamshidhar*, and *Utkalika*, who worked with me on this project and provided invaluable support whenever needed.

I am also thankful to *Prof. Arobinda Gupta*, Head, Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, and *Prof. Dipanwita Roy Chowdhury*, former Head, Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, for providing necessary facilities during the project work.

I am thankful to our Faculty Advisor *Prof. Palash Dey*, Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, and all the faculty members of the department for their valuable suggestions, which helped me improve on this work.

Abstract

In recent research, provenance tracking has been extensively employed to identify system flaws and the underlying causes of faults, crashes, and errors over a running system. There is a need to develop graph-based models for provenance tracking over modern DevOps-based serverless platforms. Application logs from containers and system logs need to be merged to form a causally dependent graph of events based on their timestamp of occurrence. To compare events based on timestamps and to offset the effect of clock drift, clocks on hosts spread across the globe must be synchronised.

In this work, we (i) find managed serverless services such as AWS Time Sync are not sufficient to generate a total ordering of logs (ii) conduct a literature survey to find the most feasible clock synchronisation technique, so it can be used on off-the-shelf servers in public and private clouds (iii) propose a design which can be readily deployed on serverless frameworks such as Apache OpenWhisk to achieve causality without modifying the core applications.

Contents

Declaration i

Certificate ii

Acknowledgements iii

Abstract iv

Contents v

List of Figures vii

Abbreviations viii

1 Introduction 1
- 1.1 Distributed Provenance Tracking . . . 1
  - 1.1.1 Provenance Graphs . . . 2
- 1.2 Problem Statement . . . 3
  - 1.2.1 Clock drift . . . 4

2 Literature Survey 5
- 2.1 Provenance Graph . . . 5
  - 2.1.1 OmegaLog . . . 5
  - 2.1.2 ALASTOR . . . 5
- 2.2 Clock Synchronisation . . . 6
  - 2.2.1 Network Time Protocol. . . . 6
  - 2.2.2 Logical Time . . . 6
  - 2.2.3 Hybrid Logical Clock . . . 6
  - 2.2.4 Hardware Solutions . . . 7

3 Approach and Methodology 9
- 3.1 Understanding time synchronisation in Managed Serverless Platforms . . . 9

3.2 Apache OpenWhisk: Open-Source Serverless Platform 10
3.3 A lean Unmanaged Serverless Platform 12
3.3.1 Docker networking modes 12
3.4 Analysis of packets using WireShark 13
3.5 Proposed Design 14
3.6 Implementation 15

4 Conclusion **16**
4.1 Summary 16
4.2 Future Works 17

Bibliography **18**

List of Figures

2.1 Hybrid Logical Clock timestamp 7

3.1 Apache OpenWhisk architecture 11
3.2 Simplified version of serverless platform 12
3.3 Packet capture of traffic from a container 13
3.4 Proposed Design for writing application logs 14
3.5 Tracing docker traffic using eBPF 15

Abbreviations

USP	**U**nmanaged **S**erverless **P**latform
CFG	**C**ontrol **F**low **G**raph
KNI	**K**ernel **N**IC **I**nterface
LMS	**L**og **M**essage **S**tring
NIC	**N**etwork **I**nterface **C**ard
TCP	**T**ransmission **C**ontrol **P**rotocol
UDP	**U**ser **D**atagram **P**rotocol
UPG	**U**niversal **P**rovenance **G**raph

Chapter 1

Introduction

1.1 Distributed Provenance Tracking

Modern software applications are built with microservice architecture patterns and serverless operational models [4] where platforms provide a developer with tools to write, deploy, and run code without configuring and managing the shared environment. This is achieved by the abstraction of the underlying infrastructure using sandboxed containers in which the stateless functions written by developers are packaged and deployed.

However, there is little support for error reporting, execution tracing, and provenance tracking in the serverless-specific industry solutions that are currently available [2]. The existing tools primarily employ graph-based models for provenance tracking over monolithic applications running directly over the operating system kernel, which is not the norm anymore.

1.1.1 Provenance Graphs

The metadata of a process that documents the specifics of its origin and the history of modifications or transformations that took place over time during its lifecycle is known as provenance data in the field of system security. The provenance graph of a process is a causal graph that stores the dependencies between system subjects (such as processes) and system objects. It is the graph where nodes are system artefacts we wish to trace (e.g., files, network sockets) and edges are events causing the state change (e.g., system calls). The edges represent the causality relationship between the log entries. [18]

This graph can be used to find the root cause of a failure or illegal events triggered by malicious entities. There have been several works that attempted to generate the provenance graphs by combining system and application logs together [11]. We define a Universal Provenance Graph (UPG) as a graph that combines the interactions among all the micro-services and can provide a meaningful platform for distributed provenance tracking.

It is important to note that a UPG is generated by (i) finding an ordering of events from application and system log entries' timestamps generated on servers that could be running in different geographical locations and (ii) judiciously using applications' control flow graph (CFG) by performing static analysis on application binaries and source code [10].

1.2 Problem Statement

Functions written by developers and packaged into containers run on servers managed by cloud service providers. These servers could be located in different geographical locations, and a container could be spun up from an image on demand in any of these regions (or data centres) to meet the client's needs (e.g.: to provide low latency response to client's users in a region).

Let's say n different services of an application $\mathcal{A}$ are packaged into docker containers $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$, which are deployed on m servers $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$. The containers $\mathcal{C}$ generate service application logs $\mathcal{L} = \{L_1, L_2, \ldots, L_n\}$. Each log entry j from container i is a tuple of form $[S_{ID}, T, LE_{ID}, LMS]$, where S_{ID} is service ID, T is the timestamp when this log entry with sequence number LE_{ID} was generated with Log Message String LMS. In order to create Universal Provenance Graph from logs $\mathcal{L}$ generated by $\mathcal{C}$, these logs need to be merged to form Universal Log File which contains all log entries from each container and the log entries are ordered by causality.

Generally, the timestamp when an event was executed and Control Flow Graph are together used to infer causality. In this thesis, we will focus on the ordering of events generated by services running in a geo-distributed fashion. Let's say each of these servers uses some Linux distribution. If we were to simply timestamp log entries with UTC version of `CLOCK_REALTIME` which represents the machine's best-guess as to the current wall-clock time-of-day, the generated ordering of events $\mathcal{O}_{gen}$ across conatiners $\mathcal{C}$ could be different from the actual ordering of events $\mathcal{O}_{act}$ because of clock drift.

1.2.1 Clock drift

Computers keep time with the help of Real-Time Clocks (RTCs), which are electronic devices (most often in the form of an integrated circuit) that measure the passage of time. Most RTCs use an electrical oscillator regulated by the frequency of a quartz crystal. The oscillation frequency depends on the clock's quality, age, sometimes the stability of the power source (voltage), ambient temperature, and other subtle environmental variables.

Clock drift refers to several related phenomena where a clock does not run at exactly the same rate as a *reference clock*. That is, after some time the clock "drifts apart" or gradually desynchronizes from the other clock. All clocks are subject to drift, causing eventual divergence unless resynchronized [14].

Clock skew refers to the instantaneous difference between the readings of any two clocks.

The aim of this thesis is therefore to develop a mechanism to ensure $\mathcal{O}_{gen} = \mathcal{O}_{act}$ when the clock skew among servers $\mathcal{S}$ is non zero.

Chapter 2

Literature Survey

2.1 Provenance Graph

2.1.1 OmegaLog

OmegaLog is an end-to-end provenance tracker that merges application event logs with the system log to generate a universal provenance graph(UPG). It says that by compiling all forensically significant system events into a single comprehensive record, attack investigation capabilities can be greatly enhanced. In order to determine the temporal correlations between LMSs, OmegaLog performs static analysis on application binaries and control flow analysis, resulting in a list of all possible LMS control flow paths [11].

2.1.2 ALASTOR

The serverless platform's central repository collects provenance from many functions. Local provenance collectors are used in each host, and a global builder service is used to create a comprehensive provenance graph. [6]

2.2 Clock Synchronisation

2.2.1 Network Time Protocol.

NTP enables synchronisation within the servers by making use of a hierarchical system of time resources. First, the highest level (stratum) has extremely precise clocks such as atomic or GPS clocks. These timekeeping resources are known as stratum 0 servers, and they are connected to the stratum 1, 2, 3 NTP servers below. Then, these servers offer the precise date and time so that communicating hosts can synchronise with one another using round-trip-time estimation and removing random error. [16]

2.2.2 Logical Time

Logical clocks are a way of timestamping and ordering events in a distributed system. [15]

Lamport's Clock. Lamport's clock is divorced from physical time (e.g., NTP clocks). The causality relationship captured, called happened-before (**hb**), is defined based on passing of information, rather than passing of time.

Vector Clock. In 1988, the vector clock (VC) [7, 19] was proposed to maintain a vectorized version of LC. VC maintains a vector at each node which tracks the knowledge this node has about the logical clocks of other nodes.

2.2.3 Hybrid Logical Clock

In an effort to bridge the gap between theory and practice of distributed systems on the topic of time hybrid logical clock, HLC combines the best of logical clocks and physical clocks. HLC captures the causality relationship like logical clocks, and

enables easy identification of consistent snapshots in distributed systems. Dually, HLC can be used in lieu of physical/NTP clocks since it maintains its logical clock to be always close to the NTP clock. Moreover HLC fits in to 64 bits NTP timestamp format. [12]

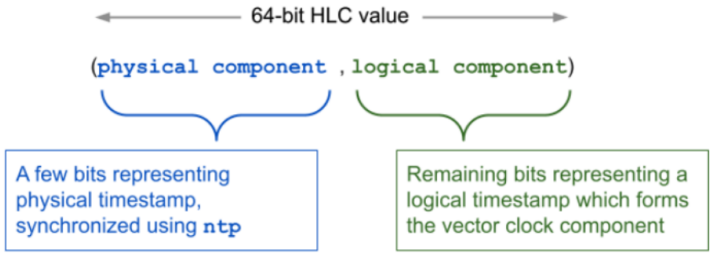

FIGURE 2.1: Hybrid Logical Clock timestamp

2.2.4 Hardware Solutions

TrueTime. by Google Spanner can guarantee an upper bound on the clock skew between the nodes in the cluster to under 7ms. TrueTime is implemented by a set of time master machines per datacenter and a timeslave daemon per machine. The majority of masters have GPS receivers with dedicated antennas; these masters separate physically to reduce the effects of antenna failures, radio interference, and spoofing. [5]

HUYGENS. Probes identify and reject impure probe data due to queuing delay, random jitter, etc. Processes the purified data with SVMs to accurately estimate one-way propagation times and achieve clock synchronization to within 100 nanoseconds. Exploits the idea that a group of pair-wise synchronized clocks must be transitively synchronized—to detect and correct synchronization errors even further. [9]

Sundial. Sundial is able to get epsilon less than 100 nanoseconds. Does timestamping and synchronization at L2 data link level in a point-to-point manner using the hardware NIC. Send a synchronization signal every 500 microseconds; Performing synchronization this frequently would not work over software. Sundial uses a

spanning tree as the multi-hop synchronization structure and synchronizes the nodes with respect to a single root [13]

The hardware solutions are not a commodity service, thereby making them infeasible to be easily deployed and run on most public or private clouds. Hence we use HLC timestamps because they are the best available software timestamping solution available at the time of writing this thesis.

Chapter 3

Approach and Methodology

3.1 Understanding time synchronisation in Managed Serverless Platforms

The first step is to determine whether the time synchronisation services such as Amazon Time Sync Service [3] offered by cloud providers like AWS are adequate to order logs. This service is delivered over NTP, which uses a fleet of redundant satellite-connected and atomic clocks in each region to deliver a highly accurate reference clock.

Experiment. We deploy functions in AWS regions `us-east-1` (US) and `ap-northeast-1` (Japan); in Google Cloud regions `us-east-1` (US) and `europe-west-1` (Germany) and send a request to each one of them simultaneously using multithreading from IIT Kharagpur. The above experiment is repeated for a hundred tries.

Results. The variance of timestamps across provider-region pairs is found to be around 220 ms. Table 3.1 shows timestamps from one such experiment.

Provider	**Region**	**Timestamp**	
		seconds	*milliseconds*
Google Cloud	us-east-1	08	270
Google Cloud	europe-west-1	08	198
AWS Lambda	us-east-1	08	680
AWS Lambda	ap-northeast-1	09	228

TABLE 3.1: Timestamps for one request-response cycle

This experiment shows physical clocks provided by cloud providers which are synchronised exclusively using NTP are not sufficient to ensure causal ordering because services would communicate over the internet resulting in uncertainty interval (δ) of the order of Wide Area Network delay ($\sim 200ms$), during which the timestamps t_1 and t_2 can't be compared if $|t_1 - t_2| < \delta$.

3.2 Apache OpenWhisk: Open-Source Serverless Platform

We have established a need to develop a mechanism to generate causal ordering from log timestamps in geo-distributed applications. This issue is even more prevalent in self-hosted **unmanaged serverless platforms (USPs)** such as Apache OpenWhisk, IronFunctions, Fn from Oracle, OpenFaaS, Kubeless, Knative, Project Riff, etc. which could be deployed in a hybrid setup (multicloud + in-house datacenters). No single accurate reference clock exists in this setup, and hence NTP synchronisation can't be trusted as NTP estimates clock skew based on round-trip-time from the reference clock.

We investigate and understand the system architecture of one such open-source distributed platform for serverless computing, Apache OpenWhisk [1].

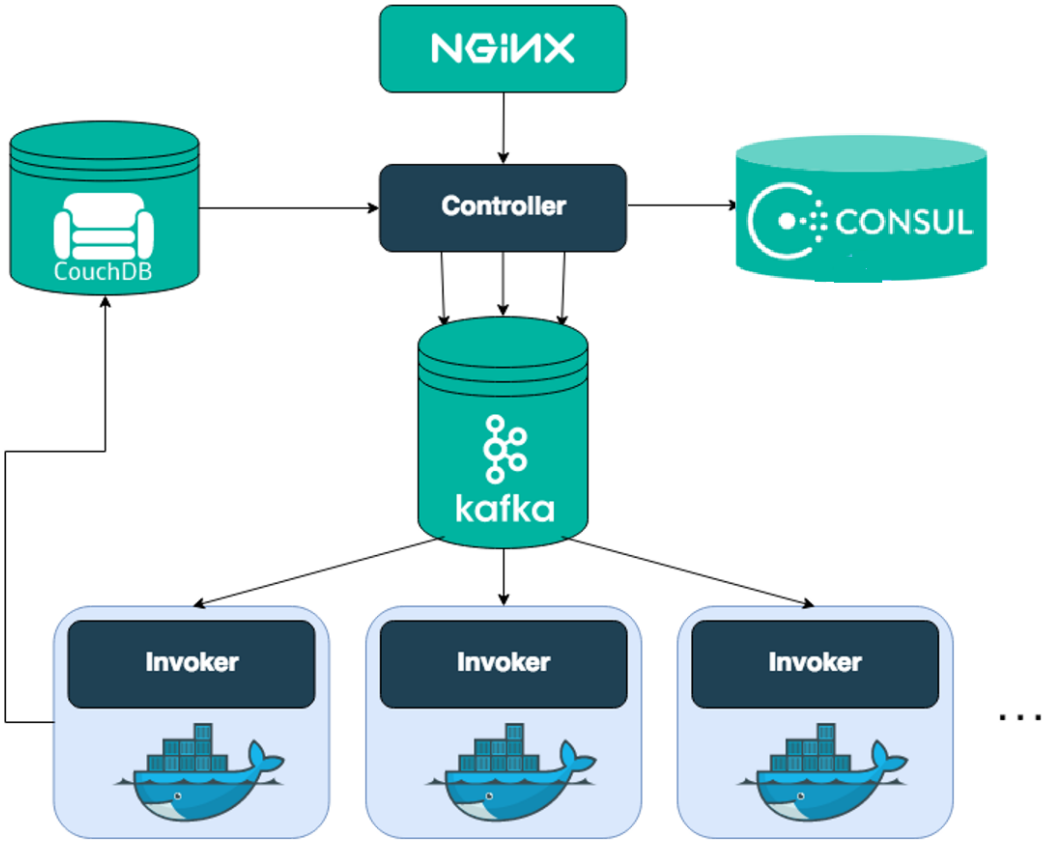

FIGURE 3.1: Apache OpenWhisk architecture

OpenWhisk is event-driven. When you invoke an action in response to an event, an HTTP request is sent to the gateway (nginx), which forwards the HTTP request to the Controller, which disambiguates what the user is trying to do based on the HTTP method. CouchDB is used for state management, which facilitates authentication and authorization checks on the request. If the user has privileges to invoke the specified action, the code to be executed along with user-specified parameters are loaded in the Controller. Next, an executor is assigned using Consul, a service discovery and the request invocation is published to Kafka to be consumed by the specified executor. The executor runs multiple user-defined functions using Docker containers.

3.3 A lean Unmanaged Serverless Platform

To ensure that the proposed time synchronisation mechanism is compatible with multiple USPs, we implement a lean version of USP comprising a gateway, a controller and two executors running docker containers within them shown in Figure 3.2.

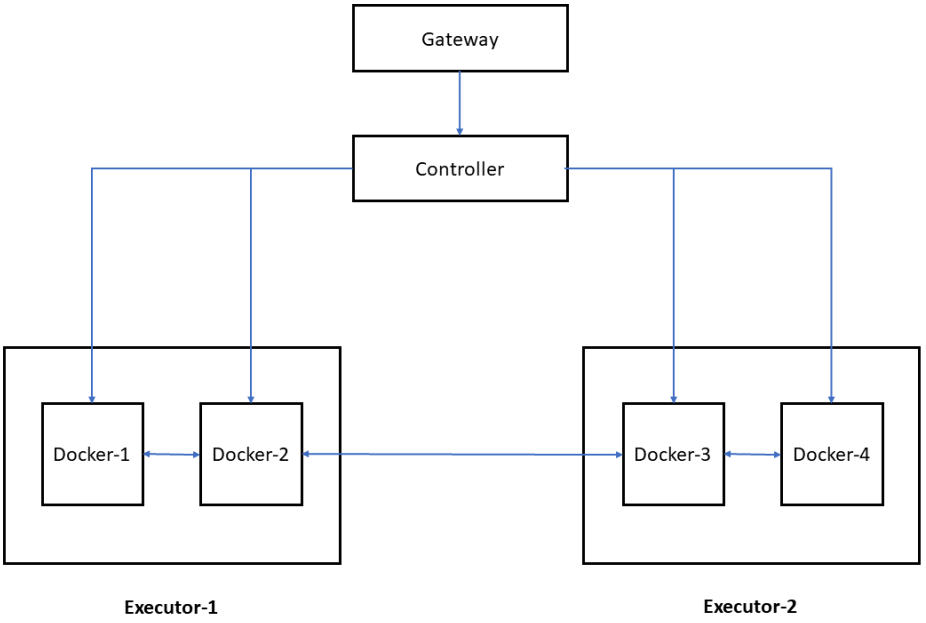

FIGURE 3.2: Simplified version of serverless platform

We attempt to understand Docker networking across hosts and intercept packets in the user space.

3.3.1 Docker networking modes

Docker's networking subsystem is pluggable, using drivers. Several drivers exist by default and provide core networking functionality. [7]

Bridge: A bridge network is a Link Layer device which forwards traffic between network segments. Containers connected to the same bridge network can communicate with each other while being isolated from containers which are not connected

to that bridge network. Bridge networks are usually used when applications run in standalone containers that need communication.

Host: Container's network stack is not isolated from the Docker host (the container shares the host's networking namespace), and the container does not get its own IP-address allocated.

Overlay: The overlay network driver creates a distributed network among multiple Docker daemon hosts. This network sits on top of (overlays) the host-specific networks, allowing containers connected to it (including swarm service containers) to communicate.

3.4 Analysis of packets using WireShark

We use Wireshark, a free and open-source packet analyzer [17] to trace docker communication over the overlay network used by docker swarm shown in Figure 3.2. The packet-trace for network traffic on container Docker-2 on Executor-1 is shown in Figure 3.3.

Source	Destination	Protocol	Length	Info
127.0.0.1	127.0.0.1	HTTP	769	GET / HTTP/1.1
172.19.0.1	172.19.0.3	HTTP	769	GET / HTTP/1.1
10.0.2.28	10.0.2.25	HTTP	265	GET / HTTP/1.1
10.0.2.25	10.0.2.28	HTTP	1336	HTTP/1.1 200 OK (text/html)
172.19.0.3	172.19.0.1	HTTP	1303	HTTP/1.1 200 OK (text/html)
127.0.0.1	127.0.0.1	HTTP	1303	HTTP/1.1 200 OK (text/html)

FIGURE 3.3: Packet capture of traffic from a container

Docker uses different virtual interfaces for internal communication on a single host (172.X) and for communication with containers on different hosts (10.X). Upon further investigation, it was found that subnet 172.X is managed by bridge `docker_gwbridge` while subnet 10.X is connected using bridge `br0`.

3.5 Proposed Design

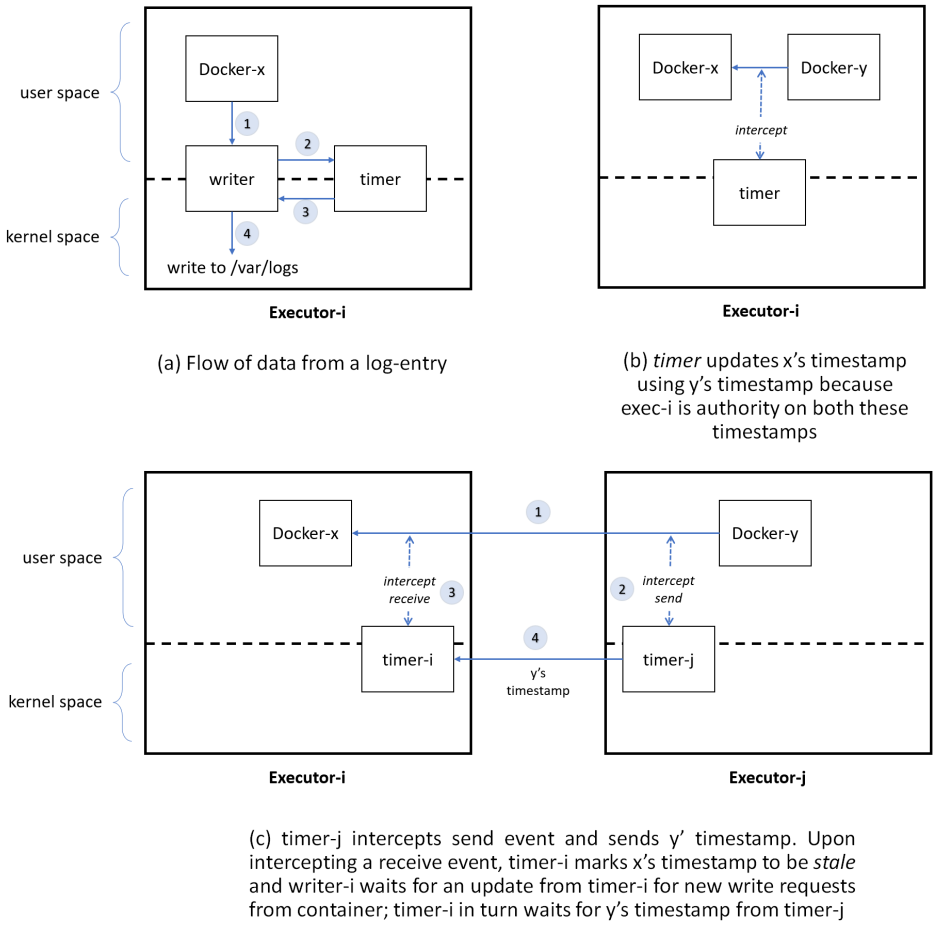

FIGURE 3.4: Proposed Design for writing application logs

Finally, we propose a *writer* daemon and a *timer* daemon, both running on each executor. The **writer** intercepts write calls to log file from containers, pushes the intermediate log-entry $[S_{ID}, LE_{ID}, LMS]$ into a buffer, queries timer for the most up-to-date timestamp T of that container and upon receiving the timestamp publishes log-entry to the log file.

The **timer** maintains Hybrid Logical Clock timestamps for all containers across all executors, but it holds authority over timestamps for containers residing in that executor. For example: in Figure 3.4 (c), timer-i holds authority for container-x

while timer-j holds authority for container-y. This is because the most up-to-date timestamp of a container has to pass through the timer service of that executor.

It is important to note that in Figure 3.4 (c), the delay of step-4 is bounded in real systems by means of timeouts and retries. Real systems are not purely asynchronous in nature.

3.6 Implementation

In this work, we focus on developing the timer service. We use eBPF, which stands for extended Berkeley Packet Filter [8], to capture network events directly from the kernel without copying them to user space, filtering packets that correspond to communication between the containers and then funnelling them out through a network tap.

We attach eBPF to kprobes of network events (like tcp_v4_connect) and trigger user space functions to update container timestamps.

A sample output of packet trace of traffic from containers of Figure 3.2 using eBPF is shown in Figure 3.5.

```
T  PID    COMM             IP SADDR           DADDR           SPORT  DPORT
C  2686   Socket Thread    4  127.0.0.1       127.0.0.1       45630  5001
A  40498  python           4  172.19.0.3      172.19.0.1      5001   56870
A  40235  docker-proxy     4  127.0.0.1       127.0.0.1       5001   45630
C  40235  docker-proxy     4  172.19.0.1      172.19.0.3      56870  5001
C  40498  python           4  10.0.2.27       10.0.2.28       45938  5002
A  40898  python           4  10.0.2.28       10.0.2.27       5002   45938
C  40898  python           4  10.0.2.28       10.0.2.25       50350  5002
X  40898  python           4  10.0.2.28       10.0.2.25       50350  5002
X  40898  python           4  10.0.2.28       10.0.2.27       5002   45938
X  40498  python           4  10.0.2.27       10.0.2.28       45938  5002
X  40498  python           4  172.19.0.3      172.19.0.1      5001   56870
X  2686   Socket Thread    4  127.0.0.1       127.0.0.1       45630  5001
X  40235  docker-proxy     4  127.0.0.1       127.0.0.1       5001   45630
X  40235  docker-proxy     4  172.19.0.1      172.19.0.3      56870  5001
```

FIGURE 3.5: Tracing docker traffic using eBPF

Chapter 4

Conclusion

4.1 Summary

This work establishes the need for time synchronisation and causality inference in both managed and unmanaged serverless applications for distributed provenance tracking. We then implement a lean unmanaged serverless platform to run experiments on. The architecture of this platform is based on Apache OpenWhisk. Communication between docker containers is analysed using Wireshark to understand the working of docker *overlay* network. Finally, we propose a pipeline for application log-entries before they are written to log files. The proposed design guarantees generated ordering of events across containers would be same as the actual ordering of events. We take first steps in implementation of the proposed *timer* service by accomplishing packet tracing using eBPF.

4.2 Future Works

We aim to (i) complete implementation of the design as soon as possible (ii) deploy and test the implementation by generating ordering of events on hosts with skewed clocks (iii) list limitations of this work (eg: what happens in uncertainty interval of clock sync) and work towards mitigating the risks, if any (iv) make a generic clock sync framework which can be easily integrated with OpenWhisk, OpenFaas, etc.

Bibliography

[1] Apache. Apache openwhisk `https://openwhisk.apache.org/`.

[2] AWS. Aws xray `https://aws.amazon.com/xray/`.

[3] AWS. Keeping time with amazon time sync service `https://aws.amazon.com/blogs/aws/keeping-time-with-amazon-time-sync-service/`.

[4] Kerry Shih-Ping Chang and Stephen J Fink. Visualizing serverless cloud application logs for program understanding. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 261–265. IEEE, 2017.

[5] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)*, 31(3):1–22, 2013.

[6] Pubali Datta, Isaac Polinsky, Muhammad Adil Inam, Adam Bates, and William Enck. {ALASTOR}: Reconstructing the provenance of serverless intrusions. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2443–2460, 2022.

[7] Docker. Docker networking `https://docs.docker.com/network/`.

[8] eBPF. ebpf `https://ebpf.io/`.

[9] Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. Exploiting a natural network effect for scalable, fine-grained clock synchronization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 81–94, 2018.

[10] Wajih Ul Hassan, Adam Bates, and Daniel Marino. Tactical provenance analysis for endpoint detection and response systems. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1172–1189. IEEE, 2020.

[11] Wajih Ul Hassan, Mohammad Ali Noureddine, Pubali Datta, and Adam Bates. Omegalog: High-fidelity attack investigation via transparent multi-layer log analysis. In *Network and distributed system security symposium*, 2020.

[12] Sandeep S Kulkarni, Murat Demirbas, Deepak Madappa, Bharadwaj Avva, and Marcelo Leone. Logical physical clocks. In *International Conference on Principles of Distributed Systems*, pages 17–32. Springer, 2014.

[13] Yuliang Li, Gautam Kumar, Hema Hariharan, Hassan Wassel, Peter Hochschild, Dave Platt, Simon Sabato, Minlan Yu, Nandita Dukkipati, Prashant Chandra, et al. Sundial: Fault-tolerant clock synchronization for datacenters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1171–1186, 2020.

[14] wikipedia. Wikipedia `https://en.wikipedia.org/wiki/Clock_drift`.

[15] wikipedia. Wikipedia `https://en.wikipedia.org/wiki/Logical_clock`.

[16] wikipedia. Wikipedia `https://en.wikipedia.org/wiki/Network_Time_Protocol`.

[17] Wireshark. Wireshark `https://www.wireshark.org/`.

[18] Shams Zawoad, Ragib Hasan, and Kamrul Islam. Secprov: Trustworthy and efficient provenance management in the cloud. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 1241–1249. IEEE, 2018.