Flexible I/O Monitoring for Distributed Systems

Tânia Esteves
tania.c.araujo@inesctec.pt
INESC TEC & University of Minho
Supervisors: ***João Paulo*** *and* ***Rui Oliveira*** *(INESC TEC & University of Minho)*

Developing, configuring, and managing distributed systems are difficult, costly, and challenging tasks.
Tracing and analysis frameworks provide insights into how the systems' state evolves over time, which is key for performance, correctness, and security analysis.
We propose a novel content-aware monitoring solution to analyze the data flow of distributed systems in a non-intrusive way.

Transparency

The usual method to capture events issued by a system is to instrument its code. However, the applications' source code may be difficult or even impossible to access.

We use kernel-level tracing tools to capture applications' storage and network events in a non-intrusive way.

Performance Overhead

Capturing I/O events requires extra processing in the critical path of operations that may lead to significant performance and resource usage overhead.

We provide two tracers that offer different performance impact and resource usage.

Storage Overhead

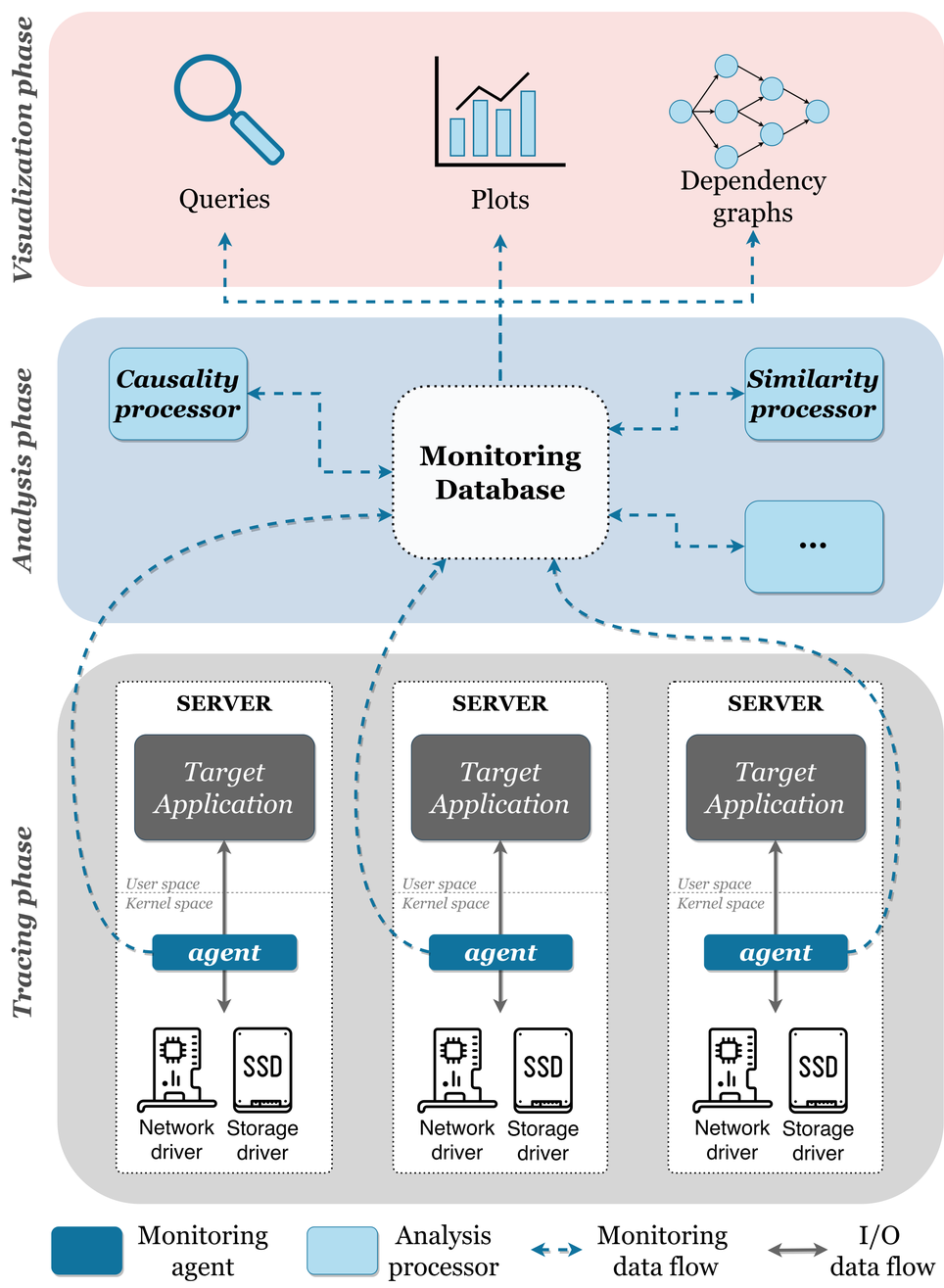
As the number of captured events increases so does the amount of information that must be stored and analysed. Thus, the resulting traces can easily occupy a large amount of storage space.

We use hashing techniques to summarize events' content.

Causality

Events' causality can provide a coherent view of what is happening among different distributed nodes. However, as there is no global clock and precisely synchronizing the physical clocks on all relevant nodes is often impossible, causality must be inferred.

We adapted a state-of-the-art solution that infers events' causality by organizing them according to their logical clocks and happens-before relationship constraints.



Automation and Visualization

Manually analyzing traces is costly and hard due to the number of events contained in them.

We provide two processors to find I/O events' causal relationships and similarities between their content.

Accuracy

The more information collected, the more accurate the analysis will be. However, capturing and processing a significant amount of data can increase the application's performance overhead and require extra processing time.

We provide options to filter information by event type, PID or TID, or file paths.

Real-time analysis

Although offline analysis offers valuable information about the target system's execution, a real-time analysis would offer the possibility to act upon irregular behavior.

We explore approaches to slice captured information into smaller parts and analyze these as soon as possible.

Querying results

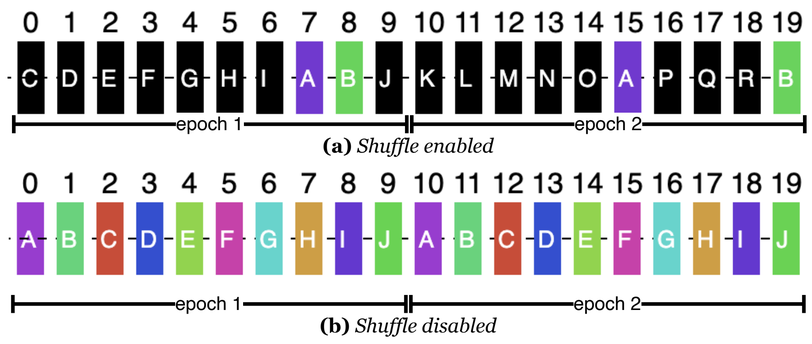
Analyzing the massive amount of collected information is challenging. Thus, it is essential to provide a mechanism to filter and query this information.

We added a database component to store and query the traced information, with pre-defined queries that show the dependencies between events.

Preliminary results

Disk access pattern for TensorFlow's dataset shuffle

	Vanilla	*CatBpf*	*CatStrace*
Elapsed time (min)	169.86	173.83	610.56
Images per second	2 527.75	2 495.92	703.07
Events handled	---	11 836 041	*
Events incomplete	---	0	---
Events truncated	---	11 788 963	*
Events Lost	---	0	*



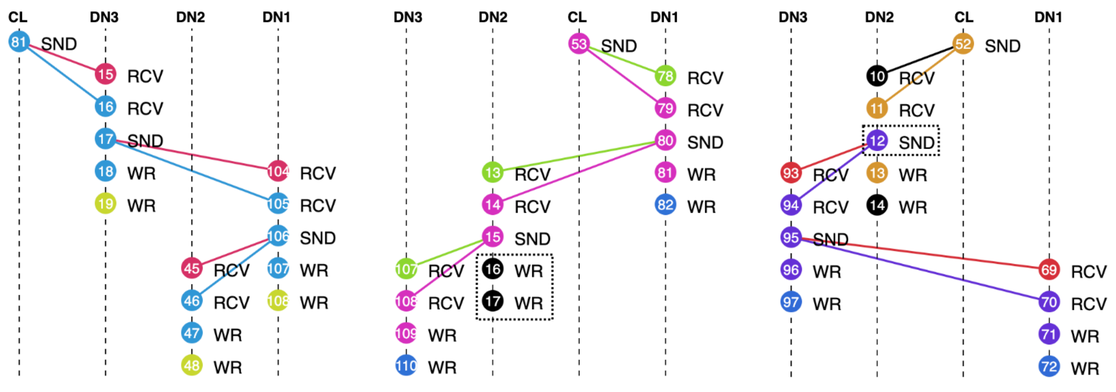
(a) *Shuffle enabled*

(b) *Shuffle disabled*

- **CaT tracers offer different trade-offs in terms of performance overhead, storage space usage, and accuracy.**
- **CatBpf** imposes minimal performance overhead (~1.26%) and generates a compacter tracing file (5.1 GiB) but truncates most of the events' content.
- **CatStrace** captures all events and their entire content, but it imposes significant performance overhead (~72.19%) and generates a larger tracing file (7.6 TiB).

- By using a **color-based scheme** and representing the events by the order they happened, CaT can pinpoint the **access pattern** performed by TensorFlow when training the LeNet model (i.e., reading the dataset from disk).
- When the shuffling mechanism is enabled, TensorFlow reads <u>different content</u> in a <u>different order</u> during the two model training epochs.
- When the shuffling mechanism is disable, TensorFlow reads the <u>same content</u> by the <u>same order</u> in the two epochs.

HDFS file replication



(a) *Normal execution* **(b)** *Storage corruption* **(c)** *Network corruption*

- CaT can pinpoint the <u>client's data flow</u> (blue color) going from the client's machine through all the DataNodes, and show the <u>replication pattern</u> done by each DataNode, **confirming the correct behavior** of the system.
- In scenarios where a corrupted DataNode alters the original content, CaT can show the difference between the content stored by each DataNode and the original content, **suggesting an erroneous behavior**.

dsrhaslab/cat <u>*https://doi.org/10.1145/3464298.3493396*</u>

Acknowledgments: *This work was financed by the FCT - Portuguese Foundation for Science and Technology, through Ph.D. grant DFA/BD/5881/2020, and realized within the scope of the project POCI-01-0247-FEDER-045924, funded by the ERDF - European Regional Development Fund, through the Operational Programme for Competitiveness and Internationalization and by National Funds through FCT, I.P. within the scope of the UT Austin Portugal Program.*