Flowstate: A Language for Secure Replicated Computation

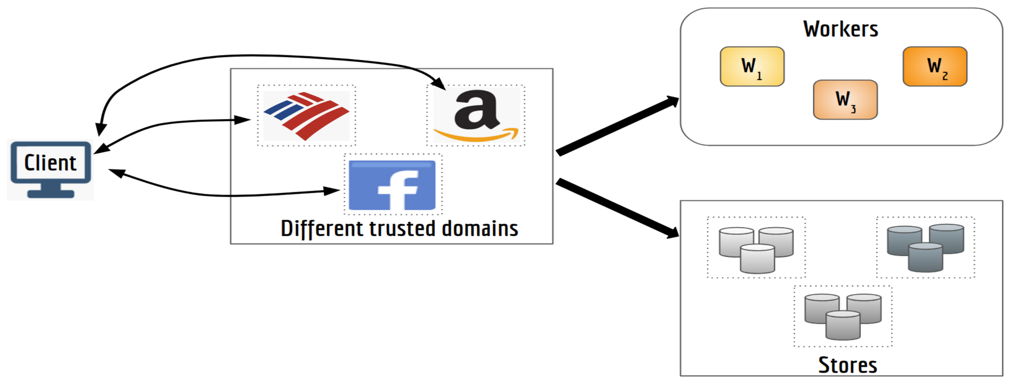
Priyanka Mondal   Owen Arden

pmondal@ucsc.edu   owen@soe.ucsc.edu

University of California, Santa Cruz

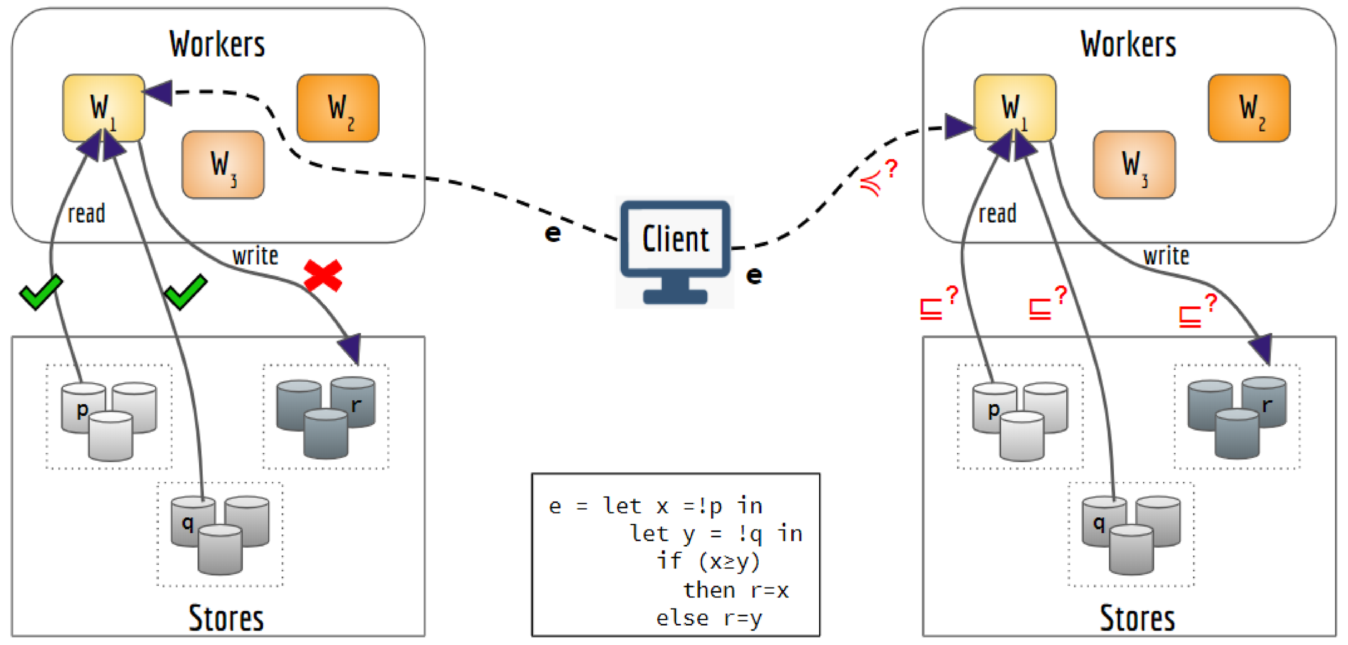
Challenges in a distributed environment

- Interactions in a distributed system span over multiple trusted domains.
- Interactions within a trusted domain are secure.
- There is mutual distrust among independent domains.



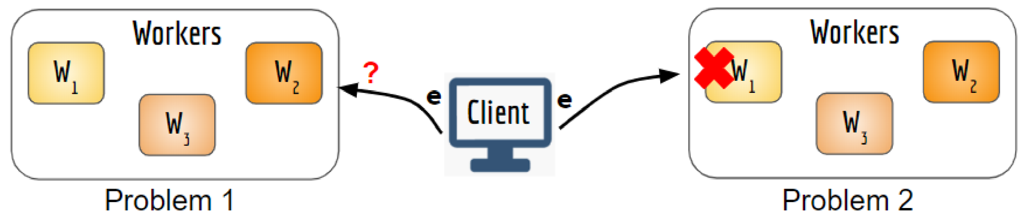
Information flow control in distributed systems

- Information flow and trust ordering can be used to prevent data leaks.



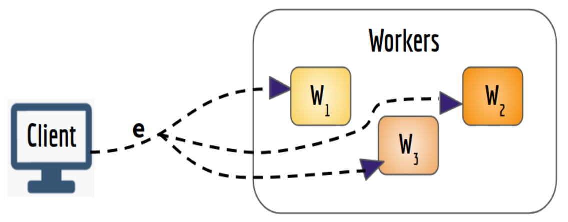
Lack of integrity and availability

- **Problem 1:** What if none of the workers is trusted enough to execute e ?
- **Problem 2:** What if the intended worker is unavailable ?



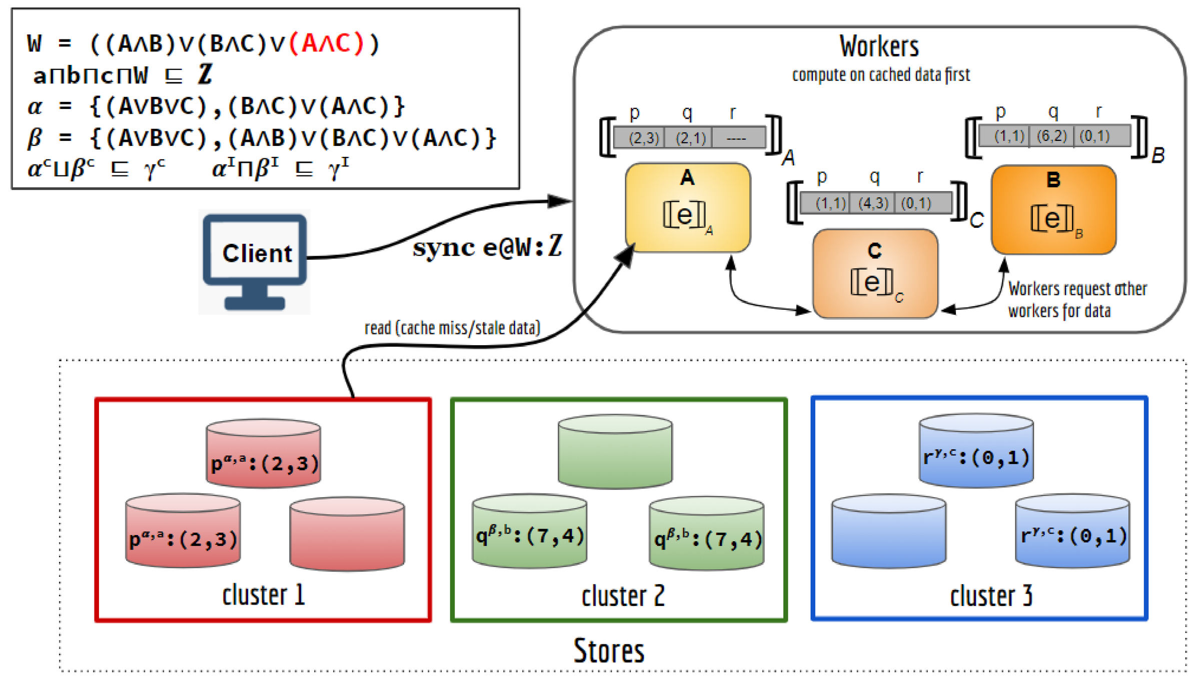
Replication of computation

- Replicate the computation at multiple workers.
- It increases trust, distribute the authority and increases availability.



Computation in Flowstate language

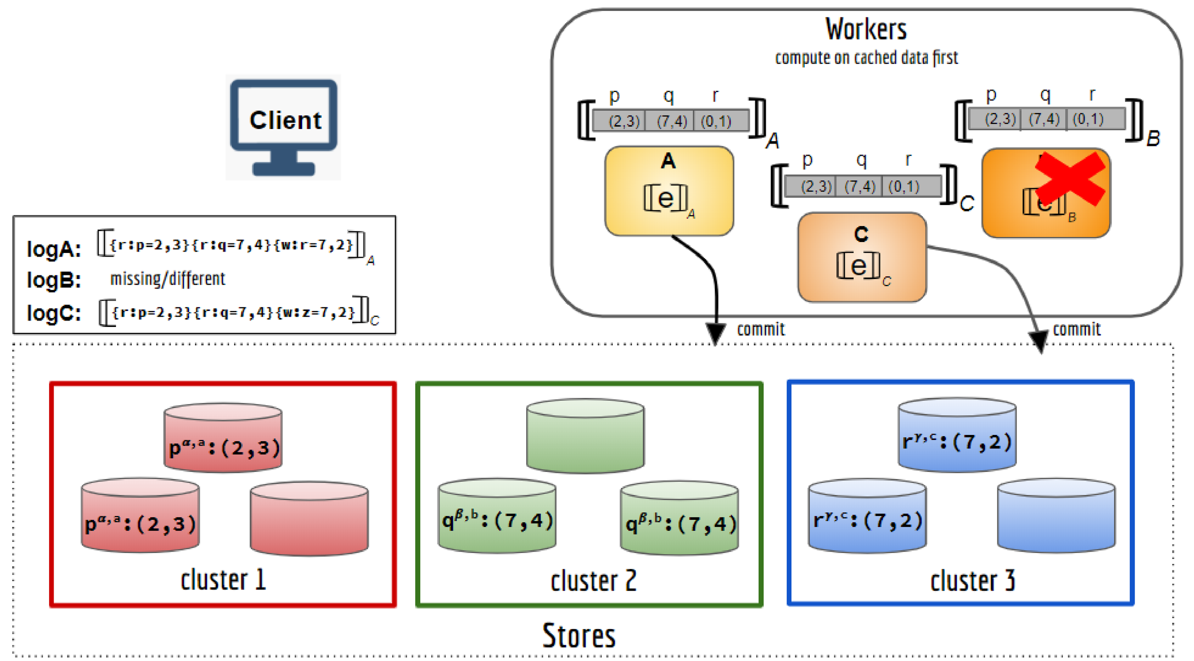
- **Problem:** Can not replicate the expression e directly, as the workers are having different integrity labels.
  Solution: Expression e is attenuated to the integrity level of the workers with a translation function $[\![\,]\!]$. If integrity of e is I then integrity of $[\![e]\!]_w$ at worker w is $(I \vee w)$. Translation function is defined over expressions, types, memory locations, and security labels.
- Replication Scheme: W is called 'replication scheme' which represents the integrity (and availability) required for the computation to be successful.



- If an insufficient number of workers have latest data then their caches are updated and the execution is restarted from the beginning.

Combined trust from the workers ensures correctness

- Multiple workers need to come to a consensus on their logs to commit the changes. In this example $(A \wedge B) \vee (B \wedge C) \vee (A \wedge C)$ specifies that at least two of the workers must have consistent (and available) log entries.
- Worker B is unavailable but still the commit is successful as combined trust from A and C is enough to prove the correctness of the result.



Trade-off between integrity and availability

- $W = (A \wedge B \wedge C)$ will not tolerate failure of B but it gives more computational integrity, while $W = (A \vee B \vee C)$ will tolerate failure of two workers among A,B and C but provides lesser computational Integrity.
- Programmers need to choose W wisely based on the requirements of the application they want to run.

Flowstate operational semantics

- Two kinds of operational semantics:
  1. Global semantics: $\langle e, \Sigma, L, c\rangle \rightarrow \langle e', \Sigma', L', c\rangle$
  2. Local semantics: $\langle e, \Sigma, L[w], w\rangle \rightarrow \langle e', \Sigma', L[w]', w\rangle$
- Sync evaluation rules
  1. $\langle sync\ e@W : Z[\,], \Sigma, [\ ], c\rangle \rightarrow \langle sync\ e@W : Z[\,[\![e]\!]_{w_1}@w_1 \ldots [\![e]\!]_{w_n}@w_n], \Sigma, [\ ], c\rangle$
     where $\{w_1, \ldots, w_n\} \in flatten(W)$
  2. If $\langle sync\ e@W : Z[\ldots e_i@w_i \ldots], \Sigma, L, c\rangle \rightarrow \langle sync\ e@W : Z[\ldots e_i'@w_i \ldots], \Sigma', L', c\rangle$
     then $\langle e_i, \Sigma, L[w_i], w_i\rangle \rightarrow \langle e_i', \Sigma', L[w_i]', w_i\rangle$
  3. $\langle sync\ e@W : Z[\ldots [\![v]\!]_{w_i}@w_i \ldots], \Sigma, L, c\rangle \rightarrow \langle v, \Sigma', [\ ], c\rangle$ - consensus on result.
  4. $\langle sync\ e@W : Z[\ldots e_i@w_i \ldots], \Sigma, L, c\rangle \rightarrow \langle sync\ e@W : Z[\,[\![e]\!]_{w_1}@w_1 \ldots [\![e]\!]_{w_n}@w_n], \Sigma', [\ ], c\rangle$
     restart computation.

Flowstate type system

- Our typing judgment looks like $\Gamma; pc; w; Z \vdash e : \tau$
- Sync typing-rule :

$$\frac{\forall w_i \in flatten(W).\Gamma; pc_i; w_i; Z \vdash [\![e]\!]_{w_i} : [\![\tau]\!]_{w_i} \quad \vdash pc \sqsubseteq pc_i \quad \vdash W^a \sqsubseteq Z \quad c \succcurlyeq pc}{\Gamma; pc; c; Z \vdash sync\ e@W : Z[\ ] : \tau}$$

Future goals

- Instantiating Flowstate with consensus based systems and protocols like Blockchain, State Channels, BFT etc.
- Beyond reads and writes: what can we do with higher-level abstractions for distributed operations ?

References

[1] Andrew Myres. Lantian Zheng.
A language-based approach to secure quorum replication, plas'14.

[2] Jed Liu et. al.
Fabric: A platform for secure distributed computation and storage,sosp,2009.

[3] Zdancewic et. al.
Secure program partitioning.