Modern applications place increasingly stringent requirements on the storage systems that underpin them. Social networks serve a **global population**, e-commerce sites must sustain an ever **growing load**, and medical applications increasingly offload **sensitive information to the cloud**. These systems must all remain **available** in the presence of natural disasters and human attacks. They consequently require (*i*) geo-replication, (*ii*) high-throughput through aggressive concurrency control mechanisms or weaker consistency guarantees, and (*iii*) strong privacy guarantees.

My research [4, 7–9, 11, 14, 21] focuses on building systems that meet these requirements. I draw from database theory, distributed computing and security to build transactional storage systems with clean abstractions, strong formal guarantees, and good performance.

Accordingly, my work centers around three topics:

(*i*) *Identifying primitives to better reason about consistency in geo-replicated systems (§1)*. The high latency between replicas in geo-replicated settings has encouraged developers to favour performance and availability over consistency, at the cost of introducing confusing application behaviours. In my dissertation, I argue that existing storage systems, by attempting to preserve the abstraction of a sequential execution model, increase the complexity of programming these weakly consistent systems. **My research instead suggests thinking about geo-replicated systems as a sequence of client-observable states, organised in distinct *branches* that capture the independent executions that naturally arise in weakly consistent systems. I show that this approach can improve semantics, performance, and programmability [8, 9].**

(*ii*) *Improving the performance of data-processing systems through federation (§2)* Data processing systems are increasingly specialised: they specifically target batch processing, graph processing, or machine learning. Database research favours a similar trend: aggressive concurrency control mechanisms are tailored to optimise performance for specific use-cases. Increased specialisation, however, often comes at the cost of generality. The very assumptions that allow systems to perform well in some cases, hurt their performance in others. Modern applications exhibit complex data processing pipelines and thus rarely fit neatly into one box. A promising approach is instead to *federate* approaches, applying each given system or concurrency control to partitions of the workload for which it shines. **My research demonstrates that, by carefully identifying the appropriate level of abstraction to reason about federations, it is possible to cleanly combine diverse sets of data processing systems [11], and concurrency controls for database systems [21].**

(*iii*) *Providing privacy at reasonable cost, by tailoring implementations to their deployment contexts (§3)* There is a growing desire to share data with untrusted third parties. Medical providers, for instance, offload data to the cloud for fault-tolerance, but do not necessarily trust the cloud to keep this data private. However, guaranteeing privacy is often costly, as general techniques to protect information usually incur overheads proportional to the number of objects in the system [5, 12, 16]. These systems, however, do not exist in a vacuum, but are deployed in a specific context (e.g. medical services or media streaming) that acts as a double-edged sword. On the one hand, it imposes additional constraints that general privacy-preserving algorithms fail to capture (like content protection in media delivery services). On the other, it can allow for more flexibility. Media files, for instance, need only be streamed as fast as users are viewing the content. **My research leverages this observation and seeks to amortise the cost of privacy. It *contextualises* the implementation of privacy-preserving mechanisms for the specific environments in which they are deployed. My work suggests that specialising private information retrieval (PIR [5, 16]) and oblivious RAM (ORAM [12]) for media streaming [14] and transactional key-value stores [7] can significantly reduce the cost of privacy.**

1 Primitives for geo-replication

My initial research focused on causally consistent systems, as they strike a sweet point between the high latency of stronger consistency guarantees, and the programming complexity of eventual consistency. Unfortunately, causal consistency has seen only limited industrial adoption. My colleagues and I argue that this reluctance stems from causal consistency's poor handling of *write operations*, both in terms of efficiency [18] and of semantics [9]. I will focus on the latter here.

Merging of write-write conflicts (TARDiS [9]) Geographically distinct replicas can issue conflicting write operations, which may cause their states to diverge. Current storage systems attempt to shield applications from the existence of such conflicts by aggressively and greedily merging them through per-object, syntactic resolution policies [17, 22]. Unfortunately, conflicting writes corrupt data in ways that the storage system cannot understand, as they create potentially incompatible states. In effect, conflicting operations create distinct, conflicting *branches of execution* that can only be resolved with detailed knowledge of application semantics. Attempting to greedily hide these conflicting executions thus makes matters worse, not better. TARDiS, the replicated transactional key-value store that we designed, consequently renounces the abstraction of sequential storage and directly exposes these conflicting branches to applications, and to the states at which the branches are created (fork points) and merged (merge points). TARDiS both simplifies semantically meaningful merging, and improves its efficiency: branches, fork and merge points directly capture the context necessary to identify objects that must be merged and pinpoint when and how the conflict developed. We find for example, that implementing CRDTs [20] - a library of weakly-consistent datatypes - using TARDiS cuts code size by half and improves performance by up to eight times.

State-based isolation ([8]) Commercial systems further hide the complexity of distribution by interacting with applications through a front-end that provides the illusion of a scalable, failure-free black-box and hides the system's internal implementation. This front-end restricts applications' view of the storage system to the states that they observe through read operations. Yet, correctness (isolation and consistency) guarantees are still defined in terms of the low-level mechanisms (i.e. histories capturing the relative ordering of read-write operations [2]) that the front-end is precisely designed to obscure. To address this mismatch, we argue that isolation and consistency guarantees should be expressed in the way they are perceived: as a contract between the storage systems and its clients, which specifies the set of observable system states. We propose a new model that directly defines isolation guarantees in terms of these high-level states. Each isolation level is expressed as constraints on the states that the application observed. Serializability, for instance, requires that the state that a transaction *observes* is the same state from which it *commits*. This approach makes it easier for programmers to understand the behaviours that weaker consistency and isolation levels allow. Using our model, we propose the first clean hierarchy of snapshot-based guarantees, and provide that several isolation levels are equivalent. Our model is general: we provide equivalent definitions of most isolation and consistency definitions, even as we make no reference to traditional notions such as dependency graphs or histories.

2 High-performance data processing through federation

Federating highly specialised data processing systems or concurrency control mechanisms into a single, unified framework achieves much of the performance benefit of specialisation without sacrificing generality. To establish a clean federation, one must address two challenges: (*i*) how to correctly combine the systems' APIs and guarantees (*ii*) how to efficiently compose their implementation.

Musketeer [11] Musketeer, a framework for large-scale distributed processing, decouples how data processing workflows are defined from how they are executed. To do so, it identifies a common intermediate abstraction that captures the semantics of most data processing computations: a direct acyclic graph (DAG) of data-flow operators, based on the relational algebra. Musketeer then takes jobs written in existing workflow specification languages (Hive, Lindi, GraphLinq, etc.), maps them to this intermediate abstraction, and compiles them dynamically to the best data-processing system(s). Our prototype speeds up realistic workflows by up to 9x by targeting different execution engines, without requiring any manual effort. The overheads of federation remain moderate: Musketeer's automatically generated back-end code comes within 5%–30% of the performance of hand-optimized implementations.

Tebaldi [21] Tebaldi, a transactional key-value store, harnesses the performance opportunities offered by federating optimised concurrency controls. Tebaldi partitions conflicts between transactions *hierarchically* and matches them to specialised concurrency controls (CCs). Tebaldi's secret sauce lies in reasoning about concurrency control mechanisms in terms of the *ordering decisions* that they make. Ordering decisions provide a common language to (*i*) correctly compose concurrency controls' *guarantees* and (*ii*) efficiently compose

their *implementation*. To achieve this, Tebaldi's framework for CC coordination observes that, despite their diversity, the steps taken by most CC protocols to order a transaction can be decomposed into four phases. In each phase, Tebaldi allows concurrency controls to constrain, delegate, or follow the ordering decisions of the other concurrency controls. These well-defined and narrow communication channels allow for the implementation of each concurrency control to remain independent of other CCs in the system and for new CCs to be added when necessary. Tebaldi achieves over 3.7× the throughput of other federated systems.

3 Embedding privacy into context

Contextualising privacy-preserving algorithms enables systems that can address the constraints of the targeted environment, while also leveraging any performance opportunities that it might allow.

Popcorn [14] Popcorn implements a Netflix-like delivery system that provably hides which movies users access. Popcorn does this with moderate cost, while respecting the current legal framework for media dissemination. Popcorn takes as its starting point Private Information Retrieval (PIR) and makes three observations. First, Popcorn combines the two types of PIR: large media objects are retrieved using the lighter weight ITPIR [5] for performance, while the smaller decryption keys are stored at a centralised Netflix-like server and retrieved using the heavier-weight CPIR [16]. Second, Popcorn amortises the high linear cost of PIR by batching requests from large number of concurrent clients accessing the service. Popcorn further leverages the specificities of media streaming to create large batches without introducing playback delays. Third, Popcorn side-steps the fixed size object requirement of PIR with minimal overhead by observing that the size of media objects can be modulated by changing their bitrate. We find that, while Popcorn has high overheads (1080x CPU and 14x I/O bandwidth), it can scale to Netflix-sized libraries and successfully stream movies at a reasonable dollar-cost.

Obladi [7] Like Popcorn, Obladi correctly and efficiently hides access patterns for transactional key-value stores specifically. Obladi is the first system to support oblivious, serializable transactions on top of untrusted storage. The system takes as its starting point oblivious RAM (ORAM), which hides access patterns for read and write operations. As in the case for media streaming, adding support for serializable transactions on top of ORAM brings forth several challenges. Existing ORAMs cannot guarantee durability, do not support transactions, and afford only limited concurrency. But like Popcorn, specifically targeting serializable transactions also offers a performance opportunity: transactions actually afford more flexibility than the simple read/write operations supported by previous ORAMS. Serializability requires that the effects of transactions be reflected consistently in the database only after they commit. Obladi leverages this flexibility to delay committing transactions until the end of fixed-size epochs, enforcing consistency and durability only at epoch boundaries. Delaying operations in this way allows Obladi to securely parallelise Ring ORAM [19] and amortise the cost of expensive ORAM operations across many transactional requests. This same principle also allows Obladi to recover efficiently and securely from failures. Our results are promising: Obladi achieves within 5x-12x of the throughput of MySQL on standard OLTP benchmarks like TPC-C. Latency is higher (70×), but remains reasonable (in the hundreds of milliseconds).

4 Future Work

I plan to target two primary lines of work in my future research. First, I will to continue exploring the benefits that branches can bring to geo-replicated systems. Second, I hope to investigate how mutually distrustful entities can consistently, reliably, and privately share data.

Branching Consistency Model Our state-based isolation and consistency model (§1) expresses guarantees as constraints on states, but assumes a linear ordering between states. In constrast, modern isolation and consistency levels like parallel snapshot isolation explicitly allow for *forks*. These forks are akin to the branching executions implemented in TARDiS. I would like to investigate whether our state-based model can be generalised to more naturally express branching executions as a function of their fork and merge points. This greater expressivity, combined with the ability to express constraints between branches, can bring benefits beyond traditional isolation guarantees. Polyinstantiated databases for instance, intentionally expose

multiple views of the same data to users, for security [15] or performance [10]. These systems, however, cannot formally reason about relationships between views. More generally, modern applications often require consistency criteria that are premised on the ability to *coordinate* concurrently executing operations. For example, a ticketing application may allow friends to commit their transactions if and only if they both have a seat on the plane [13]. This is in contrast to the traditional definition of isolation, whose primary goal is to isolate transactions from the effect of concurrently executing operations. The ability to fork, constrain, and merge concurrent branches of executions appears can cleanly express these new correctness criteria.

Branches for Strong Consistency Strongly consistent geo-replicated systems [6, 23] expose to applications clean serializable abstractions, at the cost of expensive coordination between replicas. To minimise this cost, these systems intelligently reduce the cost of committing transactions. To determine whether transactions should commit however, they fall back to traditional concurrency control mechanisms (2PL, OCC). These mechanisms implicitly rely on the network to order transactions. At each shard, the first transaction to arrive will acquire the lock. Yet, in WAN deployments, transactions often arrive in different orders at different shards, which results in transactions repeatedly aborting or deadlocking. I am interested in designing concurrency control mechanisms that avoid these issues, targeted specifically to geo-replicated systems. A promising approach is to rely on speculation. Replicas receiving transactions in different orders create distinct branches of execution that may be incompatible. Instead, why not allow replicas to (intelligently) explore multiple serializations in parallel? Every replica would then agree on the set of branches, from which they can select the final result. Implementing such a system naturally raises several questions: (*i*) can the cost of speculation be tamed? (*ii*) can branches be efficiently represented? (*iii*) with what API?

Scalably hiding access patterns Obladi has two primary drawbacks that I plan to address in future work. First, it assumes the existence of a centralised proxy that regulates the interaction between clients and the ORAM. Removing the centralised proxy is challenging: coordinating updates to ORAM metadata without leaking access patterns is expensive as the cost grows linearly with the number and size of objects in the system. Techniques developed for deterministic databases [1] can address this issue. These systems agree *a priori* on a specific execution schedule, giving each shard the ability to execute transactions without coordination. This raises two questions: (*i*) how can shards agree on this initial schedule and (*ii*) how can they execute that schedule obliviously. Obladi's second limitation is its poor scalability: while it amortises expensive ORAM operations across concurrent requests, Obladi's overhead still grows proportionally with the size of the database. To mitigate this dependence, I propose to target log-structured databases. Systems like Corfu [3] are inherently oblivious as they require playing back the full set of updates for every client. Their cost thus does not depend on the size of the database, but on the overall system throughput. Can one obliviously solve the client playback problem that prevents these systems from scaling? How can one obliviously checkpoint the log?

Computing without trust Increased data sharing also raises questions of trust. How can one remove reliance on a centralised authority? Conversely, how can one establish trust when no such authority exist, like in a globally distributed supply chain? Current techniques (BFT or blockchains) build a totally ordered log shared amongst mutually distrustful participants. However, many applications do not require this total ordering: Alice's surgery, for instance, need not be ordered with respect to Bob's X-rays. Imposing an ordering on non-conflicting operations is unnecessary and costly. Existing work mitigates this scalability bottleneck through sharding, but the latent total order requirement introduces unnecessary coordination. I propose to take the opposite approach: instead of adding database-like transactions on top of a BFT log, why not add BFT to an efficiently shardable database? The latter would allow us to directly provide the illusion of a centralised shared log, leveraging traditional serializable transactions. Naturally, this approach also comes with challenges: replicated databases' good performance in geo-replicated settings is driven by the inherent optimism of their protocol. However, this optimism cannot be straightforwardly maintained in the presence of byzantine actors. How can we ensure that byzantine replicas do not return incorrect values, while still allowing clients to read from their local datacenters? How can we prevent byzantine clients from aborting non-byzantine transactions. And finally, can we scale the system to many participants: specifically, can we design a system where replicas can contribute to establishing trust about the system, without storing the data itself?

References

[1] ABADI, D. J., AND FALEIRO, J. M. An overview of deterministic database systems. *Commun. ACM 61*, 9 (Aug. 2018), 78–88.

[2] ADYA, A., AND ADYA, A. Weak consistency: A generalized theory and optimistic implementations for distributed transactions. Tech. rep., 1999.

[3] BALAKRISHNAN, M., MALKHI, D., PRABHAKARAN, V., WOBBLER, T., WEI, M., AND DAVIS, J. D. CORFU: A shared log design for flash clusters. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)* (San Jose, CA, 2012), USENIX, pp. 1–14.

[4] BERNSTEIN, P. A., BURCKHARDT, S., BYKOV, S., CROOKS, N., FALEIRO, J. M., KLIOT, G., KUMBHARE, A., RAHMAN, M. R., SHAH, V., SZEKERES, A., AND THELIN, J. Geo-distribution of actor-based services. *Proc. ACM Program. Lang. 1*, OOPSLA (Oct. 2017), 107:1–107:26.

[5] CHOR, B., GOLDREICH, O., KUSHILEVITZ, E., AND SUDAN, M. Private information retrieval. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science* (Washington, DC, USA, 1995), FOCS '95, IEEE Computer Society, pp. 41–.

[6] CORBETT, J. C., DEAN, J., EPSTEIN, M., FIKES, A., FROST, C., FURMAN, J. J., GHEMAWAT, S., GUBAREV, A., HEISER, C., HOCHSCHILD, P., HSIEH, W., KANTHAK, S., KOGAN, E., LI, H., LLOYD, A., MELNIK, S., MWAURA, D., NAGLE, D., QUINLAN, S., RAO, R., ROLIG, L., SAITO, Y., SZYMANIAK, M., TAYLOR, C., WANG, R., AND WOODFORD, D. Spanner: Google's Globally Distributed Database. *ACM Transactions on Computer Systems (TOCS) 31*, 3 (2013), 8:1–8:22.

[7] CROOKS, N., BURKE, M., CECCHETTI, E., HAREL, S., AGARWAL, R., AND ALVISI, L. Obladi: Oblivious serializable transactions in the cloud. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)* (Carlsbad, CA, 2018), USENIX Association, pp. 727–743.

[8] CROOKS, N., PU, Y., ALVISI, L., AND CLEMENT, A. Seeing is Believing: A Client-Centric Specification of Database Isolation. In *ACM Symposium on Principles of Distributed Computing (PODC)* (2017).

[9] CROOKS, N., PU, Y., ESTRADA, N., GUPTA, T., ALVISI, L., AND CLEMENT, A. TARDiS: A Branch-and-Merge Approach To Weak Consistency. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)* (2016).

[10] GJENGSET, J., SCHWARZKOPF, M., BEHRENS, J., ARAÚJO, L. T., EK, M., KOHLER, E., KAASHOEK, M. F., AND MORRIS, R. Noria: dynamic, partially-stateful data-flow for high-performance web applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)* (Carlsbad, CA, 2018), USENIX Association, pp. 213–231.

[11] GOG, I., SCHWARZKOPF, M., CROOKS, N., GROSVENOR, M. P., CLEMENT, A., AND HAND, S. Musketeer: All for one, one for all in data processing systems. In *Proceedings of the Tenth European Conference on Computer Systems* (New York, NY, USA, 2015), EuroSys '15, ACM, pp. 2:1–2:16.

[12] GOLDREICH, O., AND OSTROVSKY, R. Software protection and simulation on oblivious rams. *J. ACM 43*, 3 (May 1996), 431–473.

[13] GUPTA, N., KOT, L., ROY, S., BENDER, G., GEHRKE, J., AND KOCH, C. Entangled queries: Enabling declarative data-driven coordination. *ACM Trans. Database Syst. 37*, 3 (Sept. 2012), 21:1–21:34.

[14] GUPTA, T., CROOKS, N., MULHERN, W., SETTY, S., ALVISI, L., AND WALFISH, M. Scalable and Private Media Consumption with Popcorn. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2016).

[15] JAJODIA, S., SANDHU, R., AND VA., G. M. U. F. *Solutions to the Polyinstantiation Problem*. Defense Technical Information Center, 1994.

[16] KUSHILEVITZ, E., AND OSTROVSKY, R. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science* (Washington, DC, USA, 1997), FOCS '97, IEEE Computer Society, pp. 364–.

[17] LLOYD, W., FREEDMAN, M. J., KAMINSKY, M., AND ANDERSEN, D. G. Don't settle for eventual: Scalable causal consistency for wide-area storage with cops. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2011), SOSP '11, ACM, pp. 401–416.

[18] MEHDI, S. A., LITTLEY, C., CROOKS, N., ALVISI, L., BRONSON, N., AND LLOYD, W. I Can't Believe It's Not Causal! Scalable Causal Consistency with No Slowdown Cascades. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)* (2017).

[19] REN, L., FLETCHER, C. W., AND KWON, A. Constants Count: Practical Improvements to Oblivious RAM. In *USENIX Security Symposium (USENIX)* (2015).

[20] SHAPIRO, M., PREGUIÇA, N., BAQUERO, C., AND ZAWIRSKI, M. Conflict-free replicated data types. In *Proceedings of the 13th International Conference on Stabilization, Safety, and Security of Distributed Systems* (Berlin, Heidelberg, 2011), SSS'11, Springer-Verlag, pp. 386–400.

[21] SU, C., CROOKS, N., DING, C., ALVISI, L., AND XIE, C. Bringing Modular Concurrency Control to the Next Level. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)* (2017).

[22] TERRY, D. B., THEIMER, M. M., PETERSEN, K., DEMERS, A. J., SPREITZER, M. J., AND HAUSER, C. H. Managing update conflicts in bayou, a weakly connected replicated storage system. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 1995), SOSP '95, ACM, pp. 172–182.

[23] ZHANG, I., SHARMA, N. K., SZEKERES, A., KRISHNAMURTHY, A., AND PORTS, D. R. K. Building Consistent Transactions with Inconsistent Replication. In *ACM Symposium on Operating System Principles (SOSP)* (2015).