Research Statement

Malte Schwarzkopf

I am a computer systems researcher and build large, practical systems to explore fundamentally new designs. In the past few years, my work has focused on systems for *datacenter computing*, an area critically important to modern applications such as web search and social networks. My research addresses fundamental mismatches between these new applications' needs and conventional system designs. To do so, I rethink system designs from their first principles, and aim to build systems that change industry practice.

Today, we frequently waste computer processor cycles and human developer time because system designs and application needs are misaligned. This mismatch forces application developers to either accept inefficiency or develop complex workarounds. Consider, for example, how web applications interact with database backends today. The application developer can either issue straightforward SQL queries and accept the inefficiency of repeated query evaluation, or she can deploy an in-memory caching layer (*e.g.*, memcached) and accept the complexity of querying, invalidating, and managing it. By contrast, redesigning the backend around web applications' needs—fast common-case reads and easily changeable queries—yields an efficient solution without extra complexity, as I discuss in detail below.

In devising a new system design, I rely on my understanding of application needs and existing data center systems to guide a redesign of the system's core abstractions from first principles. This often requires inventing a new key abstraction or adapting an existing abstraction for a novel setting. I pick simple and general abstractions that solve not merely a specific problem, but which satisfy a broad category of application needs. The abstractions must also admit an efficient implementation and have the potential to change practice.

Refining the new system design, testing it, and fully understanding its advantages and limitations requires a full-scale implementation. I am happy to take on the work to implement a complex system, and I evaluate my implementations using real applications and workloads—an important step, as accurate evaluation of datacenter systems is challenging [1].

In this statement, I primarily discuss three data center systems projects I have led: *Noria*, a high-performance backend for web applications based on a new data-flow computing model; *Omega*, a cluster scheduler architecture for datacenters that shares a cluster between several independent schedulers; and *Firmament*, a new cluster manager that demonstrates a general and high-performance approach to scheduling complex application mixes on large clusters.

Since I focus on building practical systems, my work has had impact outside of academia. For example, Omega influenced the design of several industrial cluster schedulers, and Firmament is now a community-maintained scheduler in the Kubernetes cluster manager. Engaging with practitioners often provides me with ideas for new research directions, and I outline some current directions at the end of this statement.

High-performance web application backend. A web application with millions of users easily overloads a database like MySQL. The overload is a result of the database architecture, which encourages applications to issue complex SQL read queries and evaluates them at query time. This emphasizes query flexibility, but read-heavy web application workloads are a poor match for this design: web applications re-evaluate read queries much more often than they issue updates or change the queries. An in-memory cache (*e.g.*, memcached or Redis) can improve performance, but induces serious complexity, as the application must maintain the cache.

Noria [2] addresses this mismatch without the extra complexity of separate caching. Instead of evaluating the SQL expressions during reads, Noria evaluates them when processing *writes*, and stores the precomputed results in materialized views that serve reads efficiently from in-memory state. To update these views, Noria's write-side processing feeds updates to the underlying records through a streaming data-flow computation. Data-flow lends itself to incremental computation and is easy to scale and parallelize, as the data-flow graph captures the computation's dependencies.

The materialized views and intermediate results required for incremental update processing form the data-flow's *state*. The key idea in Noria is *partially-stateful data-flow*, a new data-flow model that I designed. Partially-stateful data-flow avoids explosion in the size of this state with many queries and supports downtime-free query changes. The partially-stateful model introduces a notion of *absent*—but recomputable—state.

When a read encounters absent state, an "upquery" through the data-flow rebuilds the absent state from ancestral state (or, ultimately, the underlying "base" input records). State for new queries starts out absent and upqueries compute state entries on demand. Partially-stateful data-flow therefore maintains classic databases' query flexibility, but combines that flexibility with the advantages of streaming data-flow, such as incremental update processing and parallel scalability. It also permits reducing the data-flow's memory footprint by evicting unnecessary state, which eliminates update processing costs for rarely-read results.

An efficient implementation of partially-stateful data-flow requires careful attention to correctness when upqueries and streaming update propagation interact under concurrency. My work contributes the correctness invariants of partially-stateful data-flow, and Noria's multi-threaded and multi-machine implementation that respects these invariants. Evaluation with a production workload from the Lobsters news aggregator website shows that Noria performs well and simplifies the application. With "natural" queries—*i.e.*, queries free of hand-tuning—Noria scales to 5× higher load than MySQL does with the Lobsters developers' complex, hand-optimized queries. Noria serves up to 14M requests/sec on a single server, compared to 200k requests/sec for the widely-used memcached/MySQL combination, and scales well on multiple servers.

My OSDI 2018 paper on Noria uses partial state to resolve a mismatch between web applications' needs and existing backend designs, but I believe that data-flow with partial state is a key missing piece that enables a more fundamental rethink of backend designs. In particular, partial state gives rise to new solutions to traditionally difficult problems like dynamic sharding (new shards can start with absent state), and efficient replication for fault tolerance (recomputing absent replica state on recovery *vs.* maintaining replica state at runtime), and I am excited to explore these directions.

Shared-state cluster schedulers. Backend services like Noria often share datacenter machines with other applications. A *cluster scheduler* manages this sharing by assigning resources on cluster machines to different applications' tasks. But any single scheduler's implicit assumptions mismatch some application needs: their differences—*e.g.*, between long-running services and finite-duration batch jobs—add complexity if handled, or reduce assignment quality if not. Poor assignments reduce application performance or leave costly hardware idle (*e.g.*, saturating CPU but not memory).

With Omega [3], I introduced an efficient approach for *multiple* cluster schedulers to share a single cluster. This simplifies scheduler implementation and resolves a mismatch between different application needs and conventional scheduler architectures. Omega's key insight is to expose the full cluster state to all schedulers, and to let all schedulers make independent decisions. This *shared-state* approach allows Omega schedulers to consider all available information in their decisions—for example, a service scheduler can see running lower-priority background work and preempt it. By contrast, prior approaches either made all decisions in a single scheduler, or offered only select idle resources to the different schedulers in isolation.

Sharing full cluster state between schedulers requires them to coordinate in order to avoid impossible assignments and overloaded machines. Omega schedulers each have a local replica of cluster state and receive regular updates to it from a centralized master state (*e.g.*, if a machine failed, or if another scheduler assigned work). When a scheduler places work, it sends *deltas* to the master state that specify its own desired changes. Deltas are optimistically-concurrent transactions that only commit to the master state if no conflicting change to the same machines has occurred. In case of a conflict, the issuing scheduler must retry either the full transaction or part of it. This abstraction leaves individual scheduler implementations unconstrained, allowing them to use any internal design (*e.g.*, work queues, batched bin-packing, or constraint solvers), as long as they produce shared state deltas. Evaluation with Google production workloads shows that conflicts are rare in practice, and that Omega's shared state scales to long scheduler decision times (*e.g.*, in complex service schedulers) and to over a dozen schedulers.

My EuroSys 2013 paper on Omega has impacted several industrial cluster schedulers. Omega caused Google to add multi-scheduler support to its Borg cluster manager, focused the Kubernetes cluster manager's design on a shared store, and made HashiCorp choose the same architecture for the Nomad scheduler [4].

Scalable cluster schedulers. Some emerging applications, such as training reinforcement learning models via simulations, generate huge numbers of short-running tasks that schedulers like those atop Omega must handle. The high decision throughput required by these applications mismatches the classic, queue-based scheduler design, which computes the assignment of

each task individually, creating a scalability bottleneck.

With Firmament [5], I address this mismatch and show that even a centralized scheduler with access to all cluster state can scale to high throughput. The key idea is to use highly-efficient constraint solvers, whose algorithms amortize the costs of placement decisions over batches of tasks, thus making individual decisions cheap. Constraint solvers also allow for higher-quality placement decisions, as they globally consider all assignments, rather than making heuristic decisions myopically for one task at a time. Firmament can express a wide variety of scheduling policies as min-cost, max-flow (MCMF) constraint problems, but must address the latency that constraint solvers can induce while producing a solution. Firmament reduces this latency by combining multiple MCMF algorithms, and through an incremental MCMF formulation that reuses prior work. Using these techniques, Firmament matches the low latency of recent distributed parallel schedulers. As distributed schedulers only see statistical samples of cluster state, Firmament's full state yields better decisions.

I implemented Firmament as a full-scale prototype cluster manager. After the OSDI 2016 paper appeared, an open-source community made Firmament available as a Kubernetes scheduler, and now maintains it [6].

Other datacenter systems. During my Ph.D., I also worked on several other collaborative research projects.

CIEL [7] improved the efficiency of parallel processing over large data for algorithms with data-dependent control flow, such as iteration to convergence. CIEL was the first parallel processing system for "big data" to natively express data-dependent control flow. The key idea is a *dynamic* data-flow computing model that modifies the executing data-flow computation to spawn additional parallel tasks if necessary (*e.g.*, for a new loop iteration). Other researchers have since applied ideas from CIEL in systems for scientific computing and scale-out reinforcement learning.

Datacenter applications can *interfere* with each other through the shared datacenter network—for example, a MapReduce job's network packets can delay latency-critical responses of a web application backend. QJUMP [8] introduces an immediately deployable approach to controlling such network interference. QJUMP's key idea is to rely on packet prioritization and rate limiting mechanisms readily available in commodity network switches and the Linux kernel. Prior approaches, by contrast, required hardware or kernel changes. With QJUMP, application performance improves by 2–5× due to reduced network interference.

Finally, I co-designed the Musketeer workflow manager [9] to remove the need for developers to choose a parallel data processing system when implementing "big data" pipelines. Musketeer's key idea is to *decouple* workflow expression from execution. Users write a declarative workflow (using *e.g.*, SQL, language-integrated queries, or a vertex-centric bulk-synchronous parallel program) and Musketeer translates the workflow to executable code that runs on parallel execution engines such as Hadoop MapReduce, Spark, Naiad, PowerGraph, and others. This decoupling saves developers the effort to manually port workflows when upgrading to a new data processing system. Musketeer also suggests efficient backend execution engines for the workflow to further reduce complexity. Musketeer's key idea of decoupling expression and execution of data processing pipelines was timely: the Apache Beam project now provides similar abstractions for production use.

Musketeer's ideas also triggered a current project of mine. Conclave [10] is a query compiler that transforms a relational query into an efficient secure multi-party computation (MPC). MPC allows mutually distrusting parties (*e.g.*, different companies) to run joint computations without revealing their private data. Conclave combines cheap, local cleartext processing with costly, secure MPC steps that are needed when parties exchange data. Crucially, Conclave automatically determines the split into local and MPC parts—simplifying an otherwise onerous developer task, and making execution more efficient. Compared to baselines that run the entire query inside a secure MPC, Conclave scales to three to six orders of magnitude larger input data while still returning results in minutes.

Future directions. I am enthusiastic to continue research in computer systems, both building on my experience with datacenter systems and expanding into new areas. I am particularly excited about the following directions: securing the user data stored and processed in datacenters against leakage through inevitable application bugs; devising new abstractions for interaction between systems; and building infrastructure for new application domains such as machine learning, while also investigating how these domains can improve systems.

Securing user data. Current datacenter systems lack structural protections to secure users' private data against accidental leakage. Web applications today store data in backends shared by all users without sepa-

rate authentication. Bugs in the application logic hence easily expose users' private data, such as a private post intended only for friends. The last—and only—line of defense is for developers to carefully write queries that correctly apply the necessary security filters. A much better approach is to specify, a single, *global security policy* that the backend system enforces on all frontend queries. Such a policy might specify what data are visible, directly or in modified form, to each user. In effect, this puts each user—and her queries—into her own "parallel universe", within which applications can add arbitary queries without worrying about security.

It might be promising to realize this "multiverse" model using a data-flow system similar to Noria. Data-flow naturally lends itself to delineating universes: each user's universe forms a subgraph, created perhaps in response to user login, of a global data-flow. If the data-flow enforces security policies on records that pass the subgraph's boundary nodes, no data in violation of the security policies can reach that universe. Therefore, even if the user adds arbitrary queries to her universe, she cannot read any data that violate the security policies. This approach can apply fundamentally more sophisticated security policies than classic row-level and column-level access control, but the challenge is to make it efficient. Supporting thousands or millions of user universes in a single data-flow creates much larger data-flow graphs and state sizes than any current system (including Noria) can handle. Building such a system will require new mechanisms for compression, efficient execution, state sharing, and live modification of enormous data-flows. I have started exploring these ideas in a new prototype system, "MultiverseDB", derived from Noria, with encouraging initial results.

Beyond the systems challenges, other questions remain. How can we specify declarative security policies and check them for self-consistency? How can we prove their manifestation as data-flow nodes correct? Answers may draw on research in programming languages, information flow control, or formal verification, ideally in collaboration with experts in these fields.

Cross-system data management. I am also interested in whether we can extend these security policies *across* multiple datacenter systems, such as a database and a photo store. This requires mechanisms to enforce global security policies on interaction between different systems, and to contain security breaches if they occur. For example, RPCs may exist in a particular "universe", and security policies may apply to their handling and responses. This may require abstractions similar to those operating systems expose, albeit for datacenter systems [11]. Moreover, recent legislation like the EU's General Data Protection Regulation (GDPR) gives users more control over their personal information held in data centers—requiring *e.g.*, extraction and removal from all systems on request. Compliance with such legislation is onerous with current systems, but new cross-cutting abstractions may simplify it.

Fault-tolerant machine learning. New systems innovations will also be required to efficiently support emerging application domains such as distributed machine learning. Machine learning model training and serving, for example, often relies on stateful simulators and services. Scaling these components and making them fault-tolerant requires new techniques. In particular, I am interested in whether scalable data-flow paradigms are compatible with classic primary/backup failover. This would avoid having to roll back to a checkpoint, and improve the recovery time of—perhaps safety-critical—ML systems. For efficiency, the backup component would ideally avoid replicating the full processing of the primary prior to a failure.

Machine learning for cluster schedulers. Machine learning techniques may also themselves be helpful to systems. For example, designing cluster scheduling policies for systems like Omega and Firmament remains a challenging, manual task. With collaborators at MIT, I am currently exploring whether modern reinforcement learning techniques can *learn* a good scheduling policy directly from the workload. Our prototype already learns scheduling algorithms for high-level objectives like low average batch job duration and outperforms state-of-the-art, human-optimized batch schedulers by 1.2–2× [12]. However, further work is needed to investigate whether reinforcement learning can learn policies for complex application mixes in datacenters where no single high-level objective exists.

Summary. Computer systems research is often driven by exogenous factors, like changing hardware, changing applications, or a changing legal environment (*e.g.*, the GDPR). I am excited to continue to design and build systems that address mismatches induced by such changes, rethinking key abstractions from their fundamentals up. Making these new abstractions efficient, easy-to-use, and secure requires insights from across many areas of computer science. I believe that building full-scale systems is a good way to drive this collaboration, to understand the advantages and limitations of possible solutions, and to change real-world practice.

References

[1] Malte Schwarzkopf, Derek G. Murray, and Steven Hand. "The Seven Deadly Sins of Cloud Computing Research". In: *Proceedings of the 4th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*. Boston, Massachusetts, USA, June 2012, pages 1–6.

[2] Jon Gjengset, Malte Schwarzkopf, Jonathan Behrens, Lara Timbó Araújo, Martin Ek, Eddie Kohler, M. Frans Kaashoek, and Robert Morris. "Noria: dynamic, partially-stateful data-flow for high-performance web applications". In: *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Carlsbad, California, USA, Oct. 2018, pages 213–231.

[3] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. "Omega: flexible, scalable schedulers for large compute clusters". In: *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys)*. Prague, Czech Republic, Apr. 2013, pages 351–364. **Best student paper award.**

[4] HashiCorp, Inc. *Nomad Project – Scheduling*. URL: `https://www.nomadproject.io/docs/internals/scheduling.html` (visited on 10/23/2018).

[5] Ionel Gog, Malte Schwarzkopf, Adam Gleave, Robert N. M. Watson, and Steven Hand. "Firmament: fast, centralized cluster scheduling at scale". In: *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Savannah, Georgia, USA, Nov. 2016, pages 99–115.

[6] Kubernetes Contributors. *Poseidon: a Firmament-based Kubernetes scheduler*. URL: `https://github.com/kubernetes-sigs/poseidon` (visited on 10/23/2018).

[7] Derek G. Murray, Malte Schwarzkopf, Christopher Smowton, Steven Smith, Anil Madhavapeddy, and Steven Hand. "CIEL: a universal execution engine for distributed data-flow computing". In: *Proceedings of the 8th USENIX Symposium on Networked System Design and Implementation (NSDI)*. Boston, Massachusetts, USA, Mar. 2011, pages 113–126.

[8] Matthew P. Grosvenor, Malte Schwarzkopf, Ionel Gog, Robert N. M. Watson, Andrew W. Moore, Steven Hand, and Jon Crowcroft. "Queues don't matter when you can JUMP them!" In: *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. Oakland, California, USA, May 2015. **Best paper award.**

[9] Ionel Gog, Malte Schwarzkopf, Natacha Crooks, Matthew P. Grosvenor, Allen Clement, and Steven Hand. "Musketeer: all for one, one for all in data processing systems". In: *Proceedings of the 10th ACM European Conference on Computer Systems (EuroSys)*. Bordeaux, France, Apr. 2015, 2:1–2:16.

[10] Nikolaj Volgushev, Malte Schwarzkopf, Andrei Lapets, Mayank Varia, and Azer Bestavros. "Conclave: Secure Multi-Party Computation on Big Data". In: *Proceedings of the 14th ACM European Conference on Computer Systems (EuroSys)*. Dresden, Germany, Mar. 2019. To appear.

[11] Malte Schwarzkopf, Matthew P. Grosvenor, and Steven Hand. "New Wine in Old Skins: The Case for Distributed Operating Systems in the Data Center". In: *Proceedings of the 4th Asia-Pacific Systems Workshop (APSYS)*. Singapore, July 2013, 9:1–9:7.

[12] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. "Learning Scheduling Algorithms for Data Processing Clusters". In: *Proceedings of the 2019 ACM SIGCOMM Conference (SIGCOMM)*. To appear August 2019.