

Research Statement

Robert Soulé

With my research, I want to simplify the development of complex systems through the use of programming language technologies. Most of my work has focused on distributed stream processing [2, 3, 6, 7, 8], but I have also explored distributed storage systems [1], and security in peer-to-peer content distribution networks [4]. My methodology is to start by developing a formal model, and then to use the model to guide the design and implementation of a working system. Starting with a formalism enables a clear understanding of the system's behavior, and confidence in the correctness of its implementation. Building a working implementation helps to uncover details that the formalism elides, and provides an artifact that can be evaluated in practice.

I am extremely collaborative, and have worked with researchers from several institutions including MIT, IBM [2, 3, 6, 7, 8], and Intel [5]. Working with UT Austin on PADS [1], in particular, helped shape my approach to research. PADS is a policy architecture that allows developers to build storage systems by specifying routing and blocking policies. Routing policies determine how data flows through the system (e.g. send updates to a particular node, etc.) and blocking policies specify predicates for when nodes should block updates or read/write requests (e.g. to meet consistency goals). The routing policies are expressed as a series of logic rules written in a language called Overlog. Previous implementations of Overlog lacked a clearly defined semantics, and rules executed non-deterministically. Systems built using the prior implementations suffered from subtle bugs that were difficult to fix. Therefore, the first step for implementing my own version of the language was to specify the desired semantics. As a result, the eventual implementation avoided the non-deterministic behavior. For me, this experience illuminated the need for systems to include formal specification as part of the design process.

My thesis work focuses on language support for distributed stream processing. Streaming has become popular for applications that need to process high-volumes of data with low latency. It offers a simple but powerful programming model in which data passes through a sequence of computational steps, or operators. Programmers write their applications in a streaming language without regard to how operators map to the underlying distributed system. The compiler can then replicate and distribute operators to exploit parallelism on multicores and clusters. Many of these streaming languages are tailored to the needs of a particular application domain, such as audio/video processing, high-frequency trading, and security monitoring.

Unfortunately, while language diversity is clearly beneficial to stream processing, languages and runtimes are usually tightly coupled. As a result, streaming applications are not portable across runtimes, and languages cannot share common optimizations. The traditional approach to decoupling languages from their runtimes is to use a virtual execution environment. Classical execution environments such as the JVM or CLR support multiple languages and perform common optimizations such as function inlining, loop invariant code motion, and register allocation. In contrast, the kinds of optimizations that compilers must perform in the streaming domain are quite different. Prominent examples of streaming optimizations include operator placement, fusion, and fission (or replication). Traditional execution environments were not designed

for these types of optimizations, which involve reasoning about an entire distributed application across multiple machines. As a result, they are not appropriate for stream processing. My thesis presents an execution environment designed specifically for streaming.

As a first step, I developed the Brooklet calculus [7]. Brooklet provides a semantics that is general enough to express a wide range of streaming languages. It makes explicit the information necessary to understand how languages support communication between operators; keep state consistent; and enforce determinism. These three central concepts—state, communication, and non-deterministic execution—capture the reality of distributed implementations and enable reasoning about the correctness of streaming optimizations. The paper provides a formal specification for translating three popular streaming languages into Brooklet, and proofs for the safety of three vital streaming optimizations. This specification establishes a formal foundation for implementing an execution environment for stream processing.

The River execution environment [6] is a practical realization of the Brooklet calculus. To bridge the gap between theory and practice, it is designed to address the real-world details that the calculus elides. Specifically, River maximizes the concurrent execution of operators while guaranteeing Brooklet’s sequential semantics, and uses back-pressure to avoid buffer overflows in the presence of bounded queues. Because River is intended as the target of a translation from source languages, I also explored how to simplify language development through the composition of language modules. Moreover, River provides a common target for language-agnostic streaming optimizations. My experience has been that River significantly reduces the effort it takes to port streaming languages to an existing streaming runtime. More importantly, though, because River is based on a calculus, it supports formal proofs showing that front-ends realize the semantics of their source languages, and that optimizations are safe. The paper is currently under submission.

To be useful, an execution environment must support a broad range of optimizations that are common to streaming languages. There is an abundance of prior work on streaming optimizations. The problem is that many different research communities have independently arrived at stream processing as a programming model for high-performance and parallel computation, including digital signal processing, databases, operating systems, and complex event processing. As a result, each of these communities has developed some of the same optimizations, but often with conflicting terminology and unstated assumptions. To both consolidate terminology and make assumptions explicit, we created a survey of optimizations for stream processing [3]. This survey is also currently under submission.

While River and Brooklet simplify streaming language translation and implementation, the ideas behind the design are more widely applicable. With colleagues from IBM Research China, I observed that under certain load conditions, applications written as traditional relational database queries would be better expressed as streaming queries. I was able to leverage my experience with translating streaming languages to the problem of translating SQL queries to streaming database queries [8]. The translation largely follows the approach outlined in the Brooklet work. Using the translated queries, we demonstrated substantial performance benefits for three real-world applications.

Recently, I spent the summer as a visiting student at MIT, and in my current work, I am investigating adding dynamic scheduling and optimizations to the StreamIt stream processing language. StreamIt is a synchronous data-flow language, which means that each time an operator fires, it consumes a fixed number of data items and produces a fixed number of data items. Because of this restriction, the compiler can create a fully static schedule for the streaming application which takes advantage of data locality, and minimizes communication costs between operators. However, applications for video and image compression, such as MPEG and JPEG, require dynamic data transfer rates for decoding, and other compression schemes, such as H264, require dynamic data routing. Because these applications can not be expressed completely statically, the compiler cannot calculate an optimal schedule. However, I have observed that in practice,

many applications including MPEG, JPEG, and H264 are largely synchronous, with only a small number of dynamic regions. Therefore, to balance the tradeoffs between performance and expressivity, I am exploring a hybrid scheduling approach. This hybrid scheme combines the benefits of static scheduling with the flexibility of dynamic scheduling.

In the short term, there are several projects that I would like to pursue that build on my thesis work, such as adding dynamic optimizations to River and Brooklet; exploring a high-performance distributed CQL; and designing a new stream processing languages that leverages my experience with streaming optimizations. However, more broadly, I plan to continue to explore how language support can simplify the development of complex systems. My methodology has two distinguishing qualities. First, I emphasize the importance of formal specification as part of the systems design process, and second, I enjoy, and actively seek out collaborations from across institutions. This approach will continue to characterize my future research efforts.

References

- [1] N. Belaramani, J. Zheng, A. Nayate, R. Soulé, M. Dahlin, and R. Grimm. PADS: A policy architecture for distributed storage systems. In *Proc. 6th ACM/USENIX Symposium on Networked Systems Design and Implementation*, pp. 59–73, Apr. 2009.
- [2] M. Hirzel, H. Andrade, B. Gedik, V. Kumar, G. Losa, M. Mendell, H. Nasgaard, R. Soulé, and K.-L. Wu. Streams processing language specification. Research Report RC24897, IBM, Nov. 2009.
- [3] M. Hirzel, R. Soulé, S. Schneider, B. Gedik, and R. Grimm. A catalog of stream processing optimizations. Research Report RC25215, IBM, Sept. 2011.
- [4] N. Michalakis, R. Soulé, and R. Grimm. Ensuring content integrity for untrusted peer-to-peer content distribution networks. In *Proc. 4th ACM/USENIX Symposium on Networked Systems Design and Implementation*, pp. 145–158, Apr. 2007.
- [5] R. Soulé, R. Grimm, and P. Maniatis. Auto-Parallelization for Declarative Network Monitoring. Poster at the 21st ACM Symposium on Operating Systems Principles, Oct. 2007.
- [6] R. Soulé, M. Hirzel, B. Gedik, and R. Grimm. From a calculus to an execution environment for stream processing. Tech. Report NYU-CS-TR2011-945, New York University, Nov. 2012.
- [7] R. Soulé, M. Hirzel, R. Grimm, B. Gedik, H. Andrade, V. Kumar, and K.-L. Wu. A universal calculus for stream processing languages. In *Proc. 19th European Symposium on Programming*, vol. 6012 of *Lecture Notes in Computer Science*, pp. 507–528, Mar. 2010.
- [8] Q. Zou, H. Wang, R. Soulé, M. Hirzel, H. Andrade, B. Gedik, , and K.-L. Wu. From a stream of relational queries to distributed stream processing. In *Proc. 36th International Conference on Very Large Data Bases*, pp. 1394–1405, Sept. 2010.