In the past decade, innovation in distributed programming has revolutionized many different industries and fields of research. Most visibly, frameworks like MapReduce [3] and Dryad [4] have made it possible to organize the world's data in web search engines. These frameworks have enabled so many advances because they separate the concerns of infrastructure development from algorithm development, and enable experts in information retrieval and machine learning to focus attention on improvements in their own areas. My principal research goal is to create new systems, languages and abstractions that extend the class of applications that can be deployed in this setting. By doing this, I aim to help experts in further domains to exploit the vast available computational resources, and thereby revolutionize other industries and fields of research.

So far, my research has focused on the *expressiveness* of distributed frameworks. My main contributions have included two systems [10, 5] and a programming language [7] that facilitate the execution of *iterative* algorithms in a distributed setting. In the following section, I reflect on how this work has contributed towards my research goal, before discussing my overall approach to carrying out research.

Looking ahead, it has become apparent that the "best-effort" use of distributed resources—which was sufficient when MapReduce was designed to consume large batches of data from disk—will become a bottleneck when these frameworks attempt to tackle more challenging problems. For example, existing general-purpose systems fare poorly when attempting to provide consistently low latency, and I propose that low-level systems research has the potential to solve many such issues. By engaging with researchers and developers in a variety of areas, I will establish the limitations of current approaches, and lead the research that ultimately resolves them.

## Current and previous research

The topic of my Ph.D dissertation was "A distributed execution engine supporting data-dependent control flow" [6]. The aim of this work was to demonstrate a system that could execute iterative programs reliably on a distributed cluster, by designing a new primitive for supporting run-time control flow decisions while executing a dataflow graph. As part of this work, I developed the **CIEL** distributed execution engine [10], which is the first such engine to support the end-to-end execution of arbitrary iterative or recursive computations. The key idea in this work is the *dynamic task graph* model, which generalizes Dryad-style acyclic dataflow graphs with support for run-time rewriting. In a dynamic task graph, a task that is expected to produce some result $r$ can either produce $r$ directly or spawn a subgraph of tasks that collectively produce $r$. This makes it possible to express iterative algorithms by implementing a convergence test in a task that either yields the converged result or spawns a further iteration and test; Dryad and MapReduce programs are simply special cases that never spawn additional tasks. Furthermore, each step in the execution of a dynamic task graph is logically monotonic, which makes it straightforward to implement fault-tolerant execution; CIEL provides fault tolerance for all machines involved in the computation, including the master, workers and client.

The dynamic task graph model allows an entire iterative computation to be offloaded to an execution engine like CIEL for reliable execution. However, it forces a tail-recursive style of programming that, while expressive, is less natural than the synchronous submit-job-and-await-result pattern used in a less powerful system. To ameliorate this complexity, I created the **Skywriting** programming language [7], which provides Javascript-like syntax for imperatively building dynamic task graphs. Skywriting extends Javascript with two constructs: a `spawn` function that creates a *future* for the result of a closure, and a dereference operator that blocks until the value of a future has been computed. Together, these constructs enable the construction of arbitrary acyclic dataflows, and the dereference operator brings the result of a dataflow into the control plane so that it can be used to influence subsequent control flow. The Skywriting runtime translates a script into a dynamic task graph by storing the current continuation whenever a future is dereferenced, and spawning a new task that depends on both that continuation and the dereferenced task result. Since Skywriting was first developed, the technique has been applied to several general-purpose languages—including Scala, Python, OCaml, Haskell, and C—and each of these has been used to write programs that run on CIEL.

During my postdoc at Microsoft Research, I have primarily worked on the **Naiad** project, in which my colleagues and I are developing a new framework for incremental and iterative data-parallel computation. With Naiad, we have taken a different approach from CIEL: instead of supporting the general composition of arbitrary code, Naiad represents a computation as a graph of related *collections*, which are implemented as distributed, versioned key-value stores. Naiad employs a new execution model called *differential dataflow* [5], which allows collections to vary for several reasons—such as external changes and internal feedback—and

stores each change as a compact "diff", in a manner similar to a version control system. The result is a system that can react very efficiently to small changes in the input, even when the computation is an iterative (or nested iterative) algorithm. My main role in this project has been to design and implement the distributed version of Naiad, which requires a substantially different architecture to previous systems like CIEL, Dryad or MapReduce. Whereas those systems rely on a central master to schedule every task, the task completion times in Naiad are on the order of microseconds, and a central master would introduce unacceptable latency. Instead, each core schedules tasks independently of the others, and the workers send data and control messages directly to each other. The resulting system can complete a minimal iteration in less than one millisecond, and process real-time streaming updates from Twitter into a complex graph algorithm with sub-second latency. At present, I am investigating techniques for efficient failure recovery and elastic cluster resizing, to transform Naiad into an always-on distributed service.

In addition to these systems, I have worked on other projects that are related to data-center computing. As an intern at Microsoft Research in 2009, I developed the **Steno** compiler for LINQ [8], which transforms high-level declarative queries into an efficient inline code. When integrated with the distributed DryadLINQ framework [12], Steno can achieve large speedups for computationally-intensive queries. I have also worked on projects related to operating system virtualization, looking at security [9] and memory deduplication [11].

## Approach to research

To motivate a new research project, I first gain experience with a state-of-the-art existing system, and investigate its drawbacks when applied to real-world problems. For example, my experience with using DryadLINQ to train the Kinect body parts labeling algorithm [2] exposed limitations that directly inspired components of Steno and CIEL. Many of the optimizations in Steno were based on manual workarounds in the training algorithm, and the start-to-end fault tolerance in CIEL was motivated by the need to perform user-level checkpointing between the jobs that formed each round of decision tree expansion.

Once I have identified an interesting research problem, I prefer to develop a new, self-contained system that solves the problem. While this approach can be more laborious than simply adding features to an existing system, the blank canvas affords the freedom to take more radical design decisions, leading to more innovative research and a more coherent result. Consider the difference between CIEL and HaLoop [1], which are both designed to support iterative data-parallel computations: because HaLoop extends the Hadoop MapReduce framework, it only supports the iterative execution of a chain of MapReduce jobs, whereas CIEL can execute any acyclic dataflow in each iteration. This flexibility allows CIEL to support Dryad-style as well as MapReduce jobs, and highlights a further principle in my research: when possible, it is preferable to create a new system that generalizes and subsumes the functionality of existing systems, rather than specializing to a particular problem.

After the initial prototyping phase, I recruit users for the system, and their feedback helps in the process of make the system more robust and mature. For example, CIEL has been the basis of several student projects and a practical taught course in data center computing. Whenever possible, I prefer to release the system as open-source software, because this admits a much larger audience. The process of preparing Naiad and CIEL for public release had a dramatic effect on the quality of the software, and also improved them as platforms for subsequent research.

## Future research agenda

Over the next five years, I plan to continue my research into systems that simplify the task of developing distributed parallel computations. However, there is a limit to the improvement that can be achieved by enhancing expressiveness. As distributed frameworks and their computational models have matured, we have reached a point where the best-effort use of distributed resources is becoming the bottleneck, and is limiting the performance of these computations. Therefore, I plan to focus my research on the lower-level systems problems that are arising in large-scale distributed computation frameworks.

**Fine-grained multitenancy** Systems like Naiad demonstrate that streaming data, from sources like online social networks, can be processed by complex algorithms in real-time. A large organization will have many such streaming data sources and many potential consumers, but the present technology for sharing a compute cluster between these consumers is inadequate. Existing cluster schedulers—such as Mesos and Quincy—spatially and temporally partition a cluster between users, thus preventing two different users from sharing the (possibly processed) contents of a stream. A new approach to multitenancy is needed, whereby different computations can co-exist while sharing access to the same continually updated state. This will require lightweight isolation and fine-grained scheduling at the intra-process level. In addition, a higher-level scheduling algorithm will be required to make decisions about the placement of processed data,

whether subexpressions should be shared or replicated, and how these decisions should be revised while the computations execute.

**Language integration** Strongly-typed managed languages such as Java and C♯ simplify the development of data-parallel computations, but their use of garbage collection introduces unpredictable overheads to a distributed computation. However, since many frameworks use a functional or declarative programming model, the object lifecycle can be much simpler than in an arbitrary imperative program. Just as Steno is able to specialize declarative code into efficient loop-based code, a similar approach could generate automatic memory management routines for the records in a data-parallel computation. With judicious modifications to a language runtime, it would be possible to execute user-defined functions in a garbage-collected sandbox, where in the common case all auxilliary objects are allocated in a scratch region that is discarded on return, and transparent read-only access is provided to the unmanaged heap.

**Network protocols** Existing frameworks use TCP to transfer data between stages of a computation, which works relatively well when large chunks of data must be transferred. However, if an efficient computational model such as incremental or differential dataflow is used, the amount of data transferred between each host-pair might be very small, or it may vary over time. In this regime, latency spikes can result from occasional packet loss and retransmission, which becomes more likely when the traffic pattern resembles incast. The configuration of the networking stack has had a huge effect on the performance of Naiad, and a future multi-tenant system would likely be even more sensitive to the communication patterns of each tenant program. This creates an opportunity for research into new network protocols that exploit application-level knowledge to achieve low latency and high throughput as necessary. Further research could explore how best to exploit modern datacenter network architectures: when multiple paths exist between end hosts and connections are long-lived, additional work must be done to ensure that the aggregate bandwidth is used efficiently.

**Longer term: specialized OS architecture** Existing commodity OS architectures introduce several layers of abstraction betweeen the users' code and the hardware, whereas the received wisdom in supercomputing is that the OS should be invoked as infrequently as possible. With the advent of cloud computing and pervasive virtualization, the boundary between OS and application as a unit of deployment has blurred. This creates the opportunity to reconsider whether that boundary should remain, or whether it would be possible to compile specialized kernels that include a large proportion of the application code. Two related approaches would be to investigate new OS-level abstractions that make it possible to use hardware resources while retaining safety guarantees, and to consider new software or hardware techniques for virtualizing high-performance devices.

The above examples give a taste for how fundamental systems research is necessary to provide the next boost in the performance and predictability of large-scale data processing systems. In addition to personally leading these projects, I foresee many opportunities to collaborate in related areas of computer science, ranging from the use of specialized accelerator hardware to designing IDEs that encourage programmers to reuse existing program fragments. I look forward to working with researchers in all of these areas to advance the state of the art even further.

# References


[1] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst. HaLoop: Efficient iterative data processing on large clusters. In *Proceedings of VLDB*, 2010.

[2] Mihai Budiu, Jamie Shotton, Derek G. Murray, and Mark Finocchio. Parallelizing the training of the Kinect body parts labeling algorithm. In *Proceedings of BigLearn*, 2011.

[3] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. In *Proceedings of OSDI*, 2004.

[4] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. In *Proceedings of EuroSys*, 2007.

[5] Frank McSherry, Derek G. Murray, Rebecca Isaacs, and Michael Isard. Differential dataflow. In *Proceedings of CIDR*, 2013.

[6] Derek G. Murray. *A distributed execution engine supporting data-dependent control flow*. PhD thesis, University of Cambridge, 2011.

[7] Derek G. Murray and Steven Hand. Scripting the cloud with Skywriting. In *Proceedings of HotCloud*, 2010.

[8] Derek G. Murray, Michael Isard, and Yuan Yu. Steno: automatic optimization of declarative queries. In *Proceedings of PLDI*, 2011.

[9] Derek G. Murray, Grzegorz Miłos, and Steven Hand. Improving Xen security through disaggregation. In *Proceedings of VEE*, 2008.

[10] 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 NSDI*, 2011.

[11] Grzegorz Miłos, Derek G. Murray, Steven Hand, and Michael A. Fetterman. Satori: enlightened page sharing. In *Proceedings of USENIX ATC*, 2009.

[12] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In *Proceedings of OSDI*, 2008.