Each issue of IC has multiple aspects: several “theme” articles, one or two “track” articles (part of a series spanning a calendar year), a few nontHEME articles, and myriad columns and departments. The theme for this issue is data stream management, a topic near and dear to my heart (I’ll get back to nontHEME content later).

**Data Stream Management**

I’ll begin by thanking Frank Olken and Le Gruenwald, as well as Oliver Spatscheck of IC’s editorial board, for their efforts in selecting the articles for this issue. I’ve read a draft of their introduction, which gives a clear and concise tutorial on what data stream management means and what issues surrounding it must be addressed. I recommend that readers unfamiliar with this research area read their introduction on p. 9 before continuing with my column.

Data stream management is a great topic—at least, anyone who’s been working on the topic himself pretty much full time for five years ought to think it’s a great topic. I’ve been participating in the IBM Research “System S” project since its inception in 2003. I’m not here to plug System S, but I’ll provide the obligatory URL: http://domino.research.ibm.com/comm/research_projects.nsf/pages/system_s.index.html. However, I’ve made some observations working on the project that are worth presenting here. In some respects, they augment Frank and Le’s theme introduction by posing additional research questions, and in some respects, they simply speculate on what’s to come.

My first observation is that data stream management is itself one piece of a larger pie. Much work in this area, as Frank and Le indicate, arose from the database community: this community essentially extended the database paradigm (such as relation joins) to a streaming context, for instance, via windows of data. In fact, in “The 8 Requirements of Real-Time Stream Processing,”1 Mike Stonebraker and his colleagues argued for the use of a SQL derivative rather than general-purpose code written in C++ or Java, saying that general-purpose applications had “long development cycles and high-maintenance costs.”

Our project has had an evolutionary path that both supports and refutes this claim. We started with a general-purpose stream-processing system with applications developed directly in C++ or Java, and we found that experienced users could develop applications efficiently. However, a team of IBM researchers offered a new “declarative” programming interface, the Stream Processing Application Declarative Engine (SPADE). The System S Web site describes this interface:

The Stream Processing Application Declarative Engine (SPADE) provides a language and runtime framework to support streaming applications. Users can create applications without needing to understand the lower-level stream-specific operations. SPADE provides some built-in operators, the ability to bring streams from outside System S and export results outside the system, and a facility to extend the underlying system with user-defined operators.

Thus, SPADE has a simple high-level interface that allows for fast development and the ability to intermix user-defined operators from C++ and Java. The extensibility of user-provided code has proven itself a powerful feature and enables a variety of general-purpose applications; pushing functionality into the programming language and the infrastructure expands the range of system capabilities.

My second observation is that the applications mentioned in the guest editors’ introduction and the articles in this issue are merely the tip of the iceberg. Handling streaming data
from sensor networks is clearly a relevant use case. So are financial applications; although Frank and Le describe stock ticker tape as the original streaming data source, they didn’t mention stock transactions further. In fact (at least until the recent turmoil in the financial markets), issues relating to stock markets, energy trading, and other financial environments have been some of the key application areas driving this technology. Scientific applications are another fertile area for data stream management — IBM is currently applying stream processing to projects on radio astronomy, environmental monitoring, health monitoring, and other areas.

Going forward, interoperability will be important. This can arise in the context of standards, something the recently formed Event Processing Technical Society (www.ep-ts.com) is pursuing. Ultimately, we can imagine stream computing as an extension of cloud computing (running streaming applications in infrastructure hosted by another organization) or grid computing (independent stream-processing complexes interoperating by sharing resources).

The Glut of Content
Streaming data is one way to overwhelm a computer system if it’s not prepared for the onslaught. IC is currently facing an onslaught of its own, in a way: a glut of high-quality nontheme content, also known as “regular submissions.” This doesn’t affect the magazine’s readers to a great extent, but it has important implications for potential authors.

As regular submissions are accepted, they’re queued for publication on a space-available basis. A few years ago, our backlog of accepted manuscripts was fairly small, but in the past year or so, it’s grown significantly: currently, manuscripts are published roughly a year after final revisions are accepted. This is long by IC’s historical standards, although it’s short compared to many journals.

What to do? One option we plan to explore is to publish content online before it appears in print. This option is tempered by the need to perform professional editing before publication (unlike journals); that is, we don’t have the resources to edit the papers immediately rather than as they’re prepared for printed publication.

Another obvious option is to vary selectivity — just like a stream-processing system might filter more or less data depending on the current load. Realistically, we must factor in a large backlog when deciding whether to add a specific manuscript to the queue.

What does this mean to you? As a reader, an increase in publishable material should mean a better magazine. However, it might mean that IC at some point has no theme per se and instead publishes several articles from this backlog, or that we’ll hold off on creating new departments. As an author, my advice isn’t that you...
should not submit regular manuscripts for consideration, but that you ask yourself some questions:

- How timely is the work? If it takes a few months to be accepted and then another year to appear, will it be “stale”?
- How good is the work? I see a wide variety of submissions. I can characterize several as being student projects, or similar; this is the sort of content that might be great when presented by an undergraduate to a professor but that isn’t technically deep enough for IC.
- How big is your bribe? OK, just kidding about that one.

Please don’t get me wrong. We still want your best and most interesting work — as long as you don’t mind if it takes a while to get out there.

I’ve had fun at the helm of IC, and I look forward to continuing for another two years. As I conclude my first term as EIC, I’d like to give my warmest thanks to the staff (Rebecca Deuel, lead editor; Hazel Kosky, publications coordinator; and Steve Woods, magazine editorial manager), the associate editors in chief (Siobhán Clarke, handling columns and departments; Doug Lea, finishing up with managing theme issues; and Misha Rabinovich, who has recently assumed Doug’s duties), and the editorial board and other regular contributors who generously donate their time in support of the magazine. I thank Tim Dinger and Mark Feblowitz for helpful comments to improve this column. The opinions expressed in this column are my personal opinions. I speak neither for my employer nor for IEEE Internet Computing in this regard, and any errors or omissions are my own.

References