Lazy and Speculative Execution in Computer Systems

Butler Lampson
Microsoft Research
Butler.Lampson@microsoft.com

Abstract

The distinction between lazy and eager (or strict) evaluation has been studied in programming languages since Algol 60's call by name, as a way to avoid unnecessary work and to deal gracefully with infinite structures such as streams. It is deeply integrated in some languages, notably Haskell, and can be simulated in many languages by wrapping a lazy expression in a lambda. Less well studied is the role of laziness, and its opposite, speculation, in computer systems, both hardware and software. A wide range of techniques can be understood as applications of these two ideas. Laziness is the idea behind:

Redo logging for maintaining persistent state and replicated state machines: the log represents the current state, but it is evaluated only after a failure or to bring a replica online.

Copy-on-write schemes for maintaining multiple versions of a large, slowly changing state, usually in a database or file system.

Write buffers and writeback caches in memory and file systems, which are lazy about updating the main store.

Relaxed memory models and eventual consistency replication schemes (which require weakening the spec).

Clipping regions and expose events in graphics and window systems.

Carry-save adders, which defer propagating carries until a clean result is needed.

"Infinity" and "Not a number" results of floating point operations.

Futures (in programming) and out of order execution (in CPUs), which launch a computation but are lazy about consuming the result. Dataflow is a generalization.

"Formatting operators" in text editors, which apply properties such as "italic" to large regions of text by attaching a sequence of functions that compute the properties; the functions are not evaluated until the text needs to be displayed.

Stream processing in database queries, Unix pipes, etc., which conceptually applies operators to unbounded sequences of data, but rearranges the computation when possible to apply a sequence of operators to each data item in turn. Speculation is the idea behind:

Optimistic concurrency control in databases, and more recently in transactional memory.

Prefetching in memory and file systems.

Branch prediction, and speculative execution in general in modern CPUs.

Data speculation, which works especially well when the data is cached but might be updated by a concurrent process. This is a form of optimistic concurrency control.

Exponential backoff schemes for scheduling a resource, most notably in LANs such as WiFi or classical Ethernet.

All forms of caching, which speculate that it's worth filling up some memory with data in the hope that it will be used again. In both cases it is usual to insist that the laziness or speculation is strictly a matter of scheduling that doesn't affect the result of a computation but only improves the performance. Sometimes, however, the spec is weakened, for example in eventual consistency. I will discuss many of these examples in detail and examine what they have in common, how they differ, and what factors govern the effectiveness of laziness and speculation in computer systems.

Categories and Subject Descriptors C.4 [*Performance of Systems*]: Design studies; I.1.3 [*Languages and Systems*]: Evaluation strategies

General Terms Design, Performance.

Keywords lazy evaluation

Bio

Butler Lampson is a Technical Fellow at Microsoft Corporation and an Adjunct Professor of Computer Science and Electrical Engineering at MIT. He was on the faculty at Berkeley and then at the Computer Science Laboratory at Xerox PARC and at Digital's Systems Research Center. He has worked on computer architecture, local area networks, raster printers, page description languages, operating systems, remote procedure call, programming languages and their semantics, programming in the large, fault-tolerant computing, transaction processing, computer security, WHSIWYG editors, and tablet computers. He was one of the designers of the SDS 940 time-sharing system, the Alto personal distributed computing system, the Xerox 9700 laser printer, two-phase commit protocols, the Autonet LAN, the SDSI/SPKI system for network security, the Microsoft Tablet PC software, the Microsoft Palladium high-assurance stack, and several programming languages.

He received an AB from Harvard University, a PhD in EECS from the University of California at Berkeley, and honorary ScD's from the Eidgenssische Technische Hochschule, Zurich and the University of Bologna. He holds a number of patents on networks, security, raster printing, and transaction processing. He is a member of the National Academy of Sciences and the National Academy of Engineering and a Fellow of the Association for Computing Machinery and the American Academy of Arts and Sciences. He received the ACM Software Systems Award in 1984 for his work on the Alto, the IEEE Computer Pioneer award in 1996, the National Computer Systems Security Award in 1998, the IEEE von Neumann Medal in 2001, the Turing Award in 1992, and the National Academy of Engineering's Draper Prize in 2004.

ICFP '08 September 22-24, 2008, Victoria, BC, Canada
ACM 978-1-59593-919-7/08/09.