Many ideas are ahead of their time. You can find thousands of examples of technologies that were conceived long before they could be put into practice. For instance, the theory behind GPS dates back to the 1940s, but the ability to have sufficiently accurate clocks in space wasn’t available until the late 1960s, and a practical system wasn’t developed until the late 1970s, even as a prototype (http://en.wikipedia.org/wiki/Gps). (As an aside, the importance of fine-grained accurate time for global positioning is a modern-day analogy to the problem of using accurate shipboard clocks to determine longitude for ship navigation hundreds of years ago. This is the subject of an excellent book by Dava Sobel that Steven Levy recently selected as one of IEEE Spectrum’s “10 Great Tech Books” [see www.spectrum.ieee.org/print/6354]. It would make my top-ten list as well.)

With respect to software, being ahead of your time usually results from one of two problems: you either need better hardware for the software to be feasible, or you can’t adopt a new technology because the need for it hasn’t yet been demonstrated. In this column, I’ll discuss some examples of both.

**Moore’s Law to the Rescue**

Arthur C. Clarke is credited with the famous saying, “Any sufficiently advanced technology is indistinguishable from magic.” Even as a computer scientist, I’m impressed by some of the technologies our colleagues have enabled in the past few years. The notion that I can call my cable television provider, my bank, the Amtrak railroad company, or many other providers and converse with an “automated assistant” in natural language would have been unfathomable a few years ago. Our ability to both produce and recognize speech is due both to advances in the algorithms used to perform this “magic” and to the enormous improvements in processing speed, memory sizes, and other environmental changes.

In a similar vein, I recently reflected on an experience in the late 1970s, when I was in high school and played in a chess tournament in Westfield, New Jersey. One round was against someone using some sort of portable computer with a modem: this was Ken Thompson, of Unix fame, who had turned his attention to a chess-playing program called Belle. (Ken graciously took me and another student to Bell Labs to see the computer at the other end of the phone line; it was only when I entered college and used my first Unix system with its predefined “ken” login that I realized the connection.) Twenty years later, IBM (now my employer) built Deep Blue, the first computer to beat a human world champion, and I can buy a PDA-sized chess-playing computer for US$50 that can play better than (I would guess) 95 percent of the world’s human players.

A great example of a technology that’s truly come into its own is virtualization. Virtual machines (VMs) date back at least to the 1970s and were used extensively in the IBM VM/370 operating system (http://en.wikipedia.org/wiki/VM_(Operating_system)), which provided an execution environment that was a “virtualized” copy of the real underlying hardware. In the mid-1990s, I tried running a virtual PC environment on a Macintosh, but the hardware was too limited to let me run the PC emulator alongside my usual set of Mac applications. Once again, Moore’s law to the rescue: in recent years, virtual environments such as VMware and Xen are quite usable and are being deployed in a wide variety of environments. The isolation and security virtualization offers are enabling technologies such as cloud computing (see the Trend Wars interview on p. 7 for more on this technology).

**Process Mobility**

My PhD thesis in the late 1980s was on the subject of process migration, which is the act of moving an active process from one computer...
to another.² I built process migration in the Sprite Network Operating System,³ a Unix-like environment that a team of grad students developed under the leadership of John Ousterhout. We used migration to support applications such as parallel compilation. Its goal in Sprite was to provide complete transparency, such that neither an application nor a user would really be aware that a process had moved from one workstation to another, except for the migration’s impact on performance: applications would be moved to idle workstations to exploit otherwise idle processor capacity, and they would migrate off those workstations when the local user returned and expected good interactive performance.

In the same timeframe, numerous other systems allowed users to take advantage of idle resources. They fell into two categories: true process migration, like that in Sprite, and checkpoint-restart, which would freeze an application, save its state to a shared file system, and start a new process with approximately the same state. Examples of full migration include Locus,⁴ Mach,⁵ and Mosix;⁶ examples of checkpoint-restart include Condor⁷ and Platform Computing’s Load Sharing Facility.⁸

Several years ago, Dejan Milojčić (inventor of process migration in Mach, editor of Computing Now and IEEE Distributed Systems Online, and gracious guest author of this column last issue) led an effort to publish a survey of process migration research. ACM Computing Surveys published our article in 2000,⁹ but it’s probably time for someone to substantially update it. Back then, we placed significant emphasis on answering the critical question, “Why hasn’t process migration been widely adopted?” Possible answers included a lack of compelling applications, a lack of infrastructure, and more sociological issues. Technological advances have finally rendered the greatest issue, a lack of infrastructure, moot. They’ve also suggested more compelling needs for migration.

Virtualization has provided, if you will, the missing link. The past few years have seen several efforts to support migration of VMs, which carry their running applications with them. The ability to checkpoint VMs means that, at a minimum, you...
can move applications similar to the earlier checkpoint-restart systems. However, because these VMs carry more state than just application memory, they can encapsulate entire applications, complete with their file systems, limited inter-process communication, and other environmental components. Such encapsulation enables technologies such as cloud computing, in which an exterior organization provides execution resources that are provisioned only as needed.

We’ve also seen efforts on efficient VM migration, \(^9,10\) migrating local file system state along with a VM, \(^11\) and using migration to optimize application performance. \(^12\) In addition, we’re starting to see VM migration in the commercial arena, such as the Amazon Computing Cloud (www.enomalism.com/features/amazon-ec2-migration/). It’s finally happening!

Time to rewrite that survey...

Acknowledgment
I thank the aforementioned Dejan Milojicic for suggesting that the adoption of process migration is one example of a larger class of technologies that took time to mature into fruition, and 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