

Position Summary: Supporting Hot-Swappable Components for System Software

Kevin Hui[†] Jonathan Appavoo[†] Robert Wisniewski[‡]
Marc Auslander[‡] David Edelsohn[‡] Ben Gamsa[§]
Orran Krieger[‡] Bryan Rosenberg[‡] Michael Stumm[§]

A hot-swappable component is one that can be replaced with a new or different implementation while the system is running and actively using the component. For example, a component of a TCP/IP protocol stack, when hot-swappable, can be replaced—perhaps to handle new denial-of-service attacks or improve performance—without disturbing existing network connections. The capability to swap components offers a number of potential advantages such as: online upgrades for high availability systems, improved performance due to dynamic adaptability and simplified software structures by allowing distinct policy and implementation options to be implemented in separate components (rather than as a single monolithic component) and dynamically swapped as needed.

In order to hot-swap a component, it is necessary to (i) instantiate a replacement component, (ii) establish a quiescent state in which the component is temporarily idle, (iii) transfer state from the old component to the new component, (iv) swap the new component for the old, and (v) deallocate the old component. In doing so, three fundamental problems need to be addressed:

- The first, and most challenging problem, is to establish a quiescent state when it is safe to transfer state and swap components. The swap can only be done when the component state is not currently being accessed by any thread in the system. Perhaps the most straightforward way to achieve a quiescent state would be to require all clients of the component to acquire a reader-writer lock in read mode before any call to the component. Acquiring this external lock in write mode would thus establish that the component is safe for swapping. However, this would add overhead in the common case, and cause locality problems in the case of multiprocessors.
- The second problem is transferring state from the old component to the new one, both safely and efficiently.

[†]University of Toronto, Dept of Computer Science

[‡]IBM T. J. Watson Research Center

[§]University of Toronto, Dept of Electrical and Computer Engineering

Although the state could be converted to some canonical, serialized form, one would like to preserve as much context as possible during the switch, and handle the transfer efficiently in the face of components with potentially megabytes of state accessed across dozens of processors.

- The final problem is swapping all of the references held by client components so that the references now refer to the new one. In a system built around a single, fully typed language, like Java, this could be done using the same infrastructure as used by garbage collection systems. However, this would be prohibitively expensive for a single component switch, and would be overly restrictive in terms of systems language choice.

We have designed and implemented a mechanism for supporting hot-swappable components that avoids the problems alluded to above. More specifically, our design has the following characteristics:

- zero performance overhead for components that will not be swapped
- zero impact on performance when a component is not being swapped
- complete transparency to client components
- minimal code impact on components that wish to be swappable
- zero impact on other components and the system as a whole during the swapping operation
- good performance and scalability; that is, the swapping operation itself should incur low overhead and scale well on multiprocessor systems.

Our mechanism has been implemented in the context of the K42 operating system (www.research.ibm.com/K42), in which components in the operating system and in applications that run on K42 have been made hot-swappable. Our design and implementation, preliminary performance numbers with respect to swapping overhead, and some of the performance benefits such a facility can provide are presented in www.research.ibm.com/K42/full-hotos-01.ps.