Chapter 13
Adaptive Replacement Algorithm Templates and EELRU

Yannis Smaragdakis
University of Massachusetts, Amherst, USA

Scott Kaplan
Amherst College, USA

ABSTRACT

Replacement algorithms are a major component of operating system design. Every replacement algorithm, however, is pathologically bad for some scenarios, and often these scenarios correspond to common program patterns. This has prompted the design of adaptive replacement algorithms: algorithms that emulate two (or more) basic algorithms and pick the decision of the best one based on recent past behavior. The authors are interested in a special case of adaptive replacement algorithms, which are instances of adaptive replacement templates (ARTs). An ART is a template that can be applied to any two algorithms and yield a combination with some guarantees on the properties of the combination, relative to the properties of the component algorithm. For instance, they show ARTs that for any two algorithms A and B produce a combined algorithm AB that is guaranteed to emulate within a factor of 2 the better of A and B on the current input. They call this guarantee a robustness property. This performance guarantee of ARTs makes them effective but a naïve implementation may not be practically efficient—e.g., because it requires significant space to emulate both component algorithms at the same time. In practice, instantiations of an ART can be specialized to be highly efficient. The authors demonstrate this through a case study. They present the EELRU adaptive replacement algorithm, which pre-dates ARTs but is truly a highly optimized multiple ART instantiation. EELRU is well-known in the research literature and outperforms the well-known LRU algorithm when there is benefit to be gained, while emulating LRU otherwise.

DOI: 10.4018/978-1-60566-850-5.ch013
INTRODUCTION

An Operating System is an arbiter of hardware resources: it decides how to allocate low-level resources to user-level entities. One of the most important resources in a modern system is main memory. The OS typically uses main memory as a cache for the data of all OS processes. The question that arises is how to decide which data stay in main memory vs. which data get placed on disk, for maximum execution efficiency. This decision is equivalent to picking a replacement algorithm: an algorithm to compute which data get evicted (i.e., “replaced” with other data), when main memory is full and room needs to be made. Most traditional replacement algorithms are approximations of an LRU (Least Recently Used) policy, where the data not used the longest get evicted.

It is a well-known theoretical result, however, that all replacement algorithms can be lured into performing highly sub-optimally. Even worse, most replacement algorithms have failure scenarios that correspond to common program behavior. For instance, a recency-based policy, like LRU, fails badly for loops slightly larger than main memory and can cause Thrashing (Wiseman, 2009), (Jiang, 2009). A frequency-based policy, like LFU, can be fooled when different data have different usage patterns. This has given rise to the idea of adaptively combining replacement algorithms and emulating the better algorithm for the workload at hand.

Many past adaptive replacement algorithms have been ad hoc: they are designed as combinations of two (or more) specific algorithms. (E.g., see ARC (Megiddo & Modha, 2003), LRFU (Lee et al., 2001), LIRS (Jiang & Zhang, 2002).) We next describe a more general approach, which is based on adaptive replacement templates (ARTs). An ART is a template that can be applied to any two algorithms and yield a combination with some guarantees, relative to the properties of the component algorithms. In this way, an ART can be used to fix the failure mode of any particular replacement algorithm, by combining it with a different algorithm and adaptively switching between the two.

Specifically, given two replacement algorithms A and B, we show that we can produce an adaptive algorithm AB that will never incur more than a small multiplicative constant (e.g., 2 times) as many faults as either A or B. (The term “faults” refers to the number of replacements that need to take place overall and is the standard quality measure for replacement algorithms: A good algorithm prevents future replacements, and thus suffers fewer faults, by keeping in memory the data that are likely to be needed in the future.) Application of an ART can be generalized to more than 2 algorithms by adapting between the result of a previous ART instantiation, AB, and a third algorithm C, etc. The bound of 2x in the number of faults is usefully low because it is a worst-case guarantee. Therefore, even if algorithm A incurs many times (e.g., tens or hundreds) as many faults as B for some inputs, while B incurs many times as many faults as A for others, the combined algorithm AB will never be much worse than either A or B, drawing the best of both worlds. In practice, we find that good adaptive algorithms emulate the better of their two component algorithms very closely (within a few percent of total faults).

ARTs give the designer of a replacement algorithm significant ammunition. Even when the behavior of a workload is known (e.g., the workload has many small loops and one large one) it is hard to combine all the information and design a near-optimal replacement algorithm. For example, it is hard to design a good replacement policy that a) behaves comparably to LRU in replacing data that were not recently accessed; b) eagerly replaces data that are accessed only once, similarly to LFU; and c) behaves optimally for large linear loops. In contrast, ARTs make it easy to combine simple policies (LRU, LFU, loop detection) that do well for a single behavioral pattern each. The result has guaranteed effectiveness relative to the original algorithms.