possible to reschedule operations according to an application-

Figure 1: LLADD architecture. The shaded region covers extensions which we call *operations*. Arrows point in the direction

Here is the increment operation; decrement is analogous:

6 Linear Hash Table

LLADD provides a clean abstraction of transactional pages, allowing for many different types of customization. In general, when a monolithic system is replaced with a layered approach there is always some concern that levels of indirection and abstraction will degrade performance. So, before moving on to describe some optimizations that LLADD allows, we evaluate the performance of a simple linear hash table that has been implemented as an extension to LLADD. We also take the opportunity to describe an optimized variant of the hash table and describe how LLADD's flexible page and log formats enable interesting optimizations. We also argue that LLADD makes it easy to produce concurrent data structure implementations.

We decided to implement a *linear* hash table [16]. Linear hash tables are able to increase the number of buckets incrementally at runtime. Imagine that we want to double the size of a hash table of size 2^n and that we use some hash function $h_n(x) = h(x)\,mod\,2^n$. Choose $h_{n+1}(x) = h(x)\,mod\,2^{n+1}$ as the hash function for the new table. Conceptually, we are simply prepending a random bit to the old value of the hash function, so all lower-order bits remain the same.

outs to implement ArrayList. The page-oriented list addresses
and allocates data with respect to pages in order to preserve lo-

ject cache, multiple update modifications will incur relatively inexpensive log additions, and are only coalesced into a single modification to the page file when the object is flushed.

LLADD provides several options to handle UNDO records

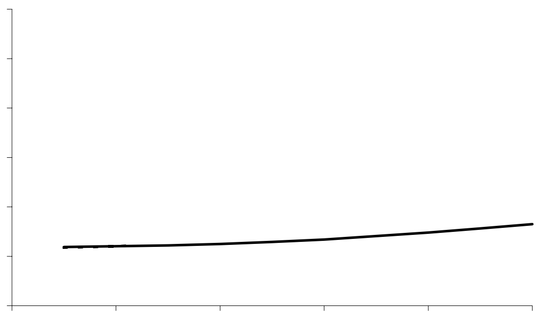

Figure 5: LLADD optimizations for object serialization. The first graph shows the effect of the two LLADD optimizations as a function of the portion of the object that is being modified. The second graph focuses on the benefits of the update/flush optimization in cases of system memory pressure.

combined with direct access to the page file and buffer pool, drastically reduces disk and memory usage for write intensive

We believe that development tools could be used to improve the quality and performance of our implementation and extensions written by other developers. Well-known static analysis techniques could be used to verify that operations hold locks (and initiate nested top actions) where appropriate, and to ensure compliance with LLADD's API. Our benchmarking section shows that our simple default hash table implementation is 3 to 4 times slower than our optimized implementation, but that automated techniques can increase its performance. Further application of static checking and high-level automated code optimization techniques may allow us to narrow or close this gap, and enhance the performance and reliability of application-specific extensions.

We would like to extend our work into distributed system development and believe that LLADD's implementation anticipates many of the thatthedefthedistributn-