Integrating Replacement Policies in StorM: An Extensible Approach

Chong Leng Goh, Beng Chin Ooi, Stephane Bressan, Kian-Lee Tan
Department of Computer Science
National University of Singapore
3 Science Drive 2, Singapore 117543
email: {gohcl,ooibc,steph,tankl}@comp.nus.edu.sg

1 Java-based *Stor*age *M*anager

StorM is a storage manager, consisting of a set of classes and objects for the development and tuning of non-standard data intensive applications in Java. It is entirely written in Java (JDK 1.2), and hence it does not require a special compiler or a special abstract machine or a set of (possibly OS specific) native methods. Exploiting the serialization properties of Java objects, StorM offers support for persistence for almost any Java object, and support for an independent notion of persistent homogeneous collection of Java objects. Since StorM aims at providing suitable support for a wide range of applications, it has been designed with an emphasis on extensibility.

2 StorM's Extensible Buffer Manager

The buffer manager of StorM is extensible. Supporting extensibility at the buffer management level is essential for the following reasons. First, extensions made at one level of a DBMS require extension support at other levels as well. For example, having a new data type extension typically requires making extensions to both the access method and query optimizer. With buffer replacement extensibility, it becomes possible to custom-tailor a smart buffer replacement policy to manage buffer replacements intelligently by exploiting the reference behavior of new access methods or transactions. Second, the emergence of more complex applications also provides more opportunities for exploiting domain specific semantics to improve system performance at the buffer manager level. Finally, supporting buffer level extensibility facilitates the evaluation and fine-tuning of new buffer replacement policies for domain specific operators, algorithms or applications. Buffer extensibility in StorM is achieved using a hierarchical buffer pool model and a priority scheme. A buffer replacement policy is modeled by a collection of types (buffer group types, local selection policy types, and abstract selection policy types) and mapping among the types (among buffer group types, and between buffer group types and selection policy types). Using a priority scheme allows us to set priorities for the various group types, so that the page to be replaced will come from group types with the lowest priority. The separation of a replacement policy into the abstract selection and local selection policies not only supports flexibility, reusability, but also allows us to design new replacement policies by different combinations of existing or new local and abstract selection policies. In summary, StorM provides a generic interface for specifying buffer replacement policies in which introduction of a new policy into StorM requires registering only five methods with the buffer manager.

In order to concretely demonstrate the effectiveness of the incorporation of a new replacement policy into the buffer manager, a visualization interface was purposely built to show the B$^+$-tree traversal for the concurrent queries as well as the actions (page accessed, page flushed, page fetched, etc.) taking place in the buffer pool. In this demonstration, we will show how a customized replacement policy based on [1] that exploits the access patterns of indexes to increase hit ratio, can be easily coded and incorporated into our buffer pool model on the fly. We also coded various replacement policies and used various access traces to demonstrate the extensibility provided by the system and the benefit of supporting them.

References

[1] C.Y. Chan, B.C. Ooi, and H.J. Lu. Extensible buffer management of indexes. In *Proceedings of the 18th Very Large Data Bases Conference*, British Columbia, 1992.