Automatic Prefetching by Traversal Profiling in Object Persistence Architectures*

Ali Ibrahim and William R. Cook

Department of Computer Sciences, University of Texas at Austin
{aibrahim, wcook}@cs.utexas.edu

Abstract. Object persistence architectures support transparent access to persistent objects. For efficiency, many of these architectures support queries that can *prefetch* associated objects as part of the query result. While specifying prefetch manually in a query can significantly improve performance, correct prefetch specifications are difficult to determine and maintain, especially in modular programs. Incorrect prefetching is difficult to detect, because prefetch is only an optimization hint. This paper presents AutoFetch, a technique for automatically generating prefetch specifications using traversal profiling in object persistence architectures. AutoFetch generates prefetch specifications based on previous executions of similar queries. In contrast to previous work, AutoFetch can fetch arbitrary traversal patterns and can execute the optimal number of queries. AutoFetch has been implemented as an extension of Hibernate. We demonstrate that AutoFetch improves performance of traversals in the OO7 benchmark and can automatically predict prefetches that are equivalent to hand-coded queries, while supporting more modular program designs.

1 Introduction

Object persistence architectures allow programs to create, access, and modify *persistent objects*, whose lifetime extends beyond the execution of a single program. Examples of object persistence architectures include object-relational mapping tools [10, 6, 28, 24], object-oriented databases [8, 21], and orthogonally persistent programming languages [25, 2, 19, 22].

For example, the Java program in Figure 1 uses Hibernate [6] to print the names of employees, their managers, and the projects they work on. This code is typical of industrial object-persistence models: a string representing a query is passed to the database for execution, and a set of objects is returned. This query returns a collection of employee objects whose first name is "John". The fetch keyword indicates that related objects should be loaded along with the main result objects. In this query, both the manager and multiple projects are prefetched for each employee.

* This work was supported by the National Science Foundation under Grant No. 0448128.

D. Thomas (Ed.): ECOOP 2006, LNCS 4067, pp. 50–73, 2006.

```
1  String query = "from Employee e
2    left join fetch e.manager left join fetch e.projects
3    where e.firstName = 'John' order by e.lastName";
4  Query q = sess.createQuery(query);
5  for (Employee emp : q.list()) {
6    print(emp.getName() + ": " + emp.getManager().getName());
7    for (Project proj : emp.getProjects()) {
8      printProject(prog);
9    }
10 }
```

Fig. 1. Java code using `fetch` in a Hibernate query

While specifying prefetch manually in a query can significantly improve performance, correct prefetch specifications are difficult to write and maintain manually. The prefetch definitions (line 2) in the query must correspond exactly to the code that uses the results of the query (lines 6 through 8).

It can be difficult to determine exactly what related objects should be prefetched. Doing so requires knowing all the operations that will be performed on the results of a query. Modularity can interfere with this analysis. For example, the code in Figure 1 calls a printProject method which can cause additional navigations from the project object. It may not be possible to statically determine which related objects are needed. This can happen if class factories are used to create operation objects with unknown behavior, or if classes are loaded dynamically.

As a program evolves, the code that uses the results of a query may be changed to include additional navigations, or remove navigations. As a result, the query must be modified to prefetch the objects required by the modified program. This significantly complicates evolution and maintenance of the system. If a common query is reused in multiple contexts, it may need to be copied in order to specify different prefetch behaviors in each case.

Since the prefetch annotations only affect performance, it is difficult to test or validate that they are correct – the program will compute the same results either way, although performance may differ significantly.

In this paper we present and evaluate AutoFetch, which uses traversal profiling to automate prefetch in object persistence architectures. AutoFetch records which associations are traversed when operating on the results of a query. This information is aggregated to create a statistical profile of application behavior. The statistics are used to automatically prefetch objects in future queries.

In contrast, previous work focused on profiling application behavior in the context of a single query. While this allowed systems such as PrefetchGuide [13] to prefetch objects on the initial execution of query, AutoFetch has several advantages. AutoFetch can prefetch arbitrary traversal patterns in addition to recursive and iterative patterns. AutoFetch can also execute fewer queries once patterns across queries are detected. AutoFetch's disadvantage of not optimizing initial query executions can be eliminated by combining AutoFetch with previous work.