THERE'S NOTHING LIKE SHARED NOTHING

Peter Thanisch* and Michael G. Norman**

*Department of Computer Science
University of Edinburgh
Kings Bldgs., Mayfield Road, Edinburgh, EH9 3JZ, UK
Email: pt@dcs.ed.ac.uk

**Makespan Limited
Bonnington Mill, 72 Newhaven Road, Edinburgh, EH6 5QG, UK
Email: mgn@makespan.demon.co.uk

ABSTRACT

In order to achieve higher performance, the developers of object-oriented database management systems (ODMSs) have concentrated on exploiting client/server computing. Recently, however, the ODMS research community has been investigating parallel computing technology. The nature of their research prototypes and simulation models indicate that the favoured parallel ODMS architecture is 'shared-nothing', an architecture much discussed in the context of parallel relational databases.

In an influential paper that appeared three years ago, it was asserted that the shared-nothing architecture is the 'consensus' parallel hardware architecture for relational database systems. In this paper, we argue that if there ever was a consensus about shared-nothing as a hardware architecture, there certainly is not a consensus now. Even as a software architecture for database systems, shared-nothing is far from being the consensus.

We conclude that ODMS researchers interested in parallel computing may well want to 'shop around' before committing themselves to a particular architectural approach.
1 INTRODUCTION

It is most amusing to look back a few years at what the megapundits were saying about the prospects for object servers providing persistence for object-oriented programming applications. Their predictions were that object-oriented database management systems (ODMSs) would out-perform relational databases by a factor of a hundred or even a thousand [10]. Some of the megapundits were confidently predicting the demise of the relational DBMS. There are several reasons why their predictions have been confounded, but one of the main factors is to do with parallel computing.

Commercial relational DBMS vendors have been quick to exploit parallel computing technology. By contrast, researchers in the field of object-orientation, with or without persistence, have not been able to make comparable progress. This is partly because of the intrinsic difficulty of parallelising arbitrarily complex object-related operations, but the lack of progress is also self-inflicted. In the object-oriented literature there is a confusion between concurrency and parallelism. There is a widespread misconception that opportunities for concurrency must somehow translate into strategies for parallelisation. It is sad how often one comes across a hopelessly flawed argument that goes like this:

Object A and object B send each other messages and can invoke methods concurrently. Therefore, we have an opportunity for exploiting parallelism by mapping these objects to separate processors.

Anyone with experience in the use of parallel computers will draw precisely the opposite conclusion: the more message-passing activity there is between a pair of objects, the more compelling is the case for mapping them to the same processor because the overhead of communication will swamp the benefits of parallel method invocation.

In order to achieve higher performance, the developers of ODMSs have concentrated on exploiting client/server computing. The major commercial ODMS products seem better positioned to exploit client/server computing, in comparison to relational database products, because the division of labour between the server and the client is very different in ODMSs and relational systems. The ODMS client is able to do a lot more of the hard work than the relational database client.