Session 4: Overview

Antony L. Hosking

Department of Computer Sciences, Purdue University,
West Lafayette, IN 47907-1398, USA
hosking@cs.purdue.edu

The papers in this session both address the behavior of persistent object stores in response to the applications they are intended to support.

The first paper, by Richer and Shapiro, was presented by Nicolas Richer [NR]. It evaluates the behavior of five memory allocation and clustering strategies for the PerDiS persistent distributed object store, when it is used as the underlying storage platform for sites on the World Wide Web. The workloads comprise recorded traces of actual accesses to two different web sites. These are used to drive a simulation of the PerDiS allocation strategies, in which the web sites are modelled as an object graph.

Results of the study include demographics on object size (80% are less than 50K bytes), the fraction of objects participating in cycles (many), and the size of those cycles in bytes and number of objects (small). The simulations show that a large fraction of references cross the boundaries of „bunches“, which are regions of clustered objects in the PerDiS store. Inter-bunch references are more expensive than intra-bunch references. Moreover, the bunch is the unit of garbage collection in PerDiS, so cycles that cross bunch boundaries cannot easily be collected. The high number of inter-bunch references is unexpected and negatively impacts performance for all of the PerDiS allocation and clustering strategies. Still, strategies that perform allocation depth-first by graph topology yield marginally better locality of reference overall.

Several questions followed the presentation, paraphrased as follows. Brian Lewis [BL] asked why simulation was used instead of using PerDiS directly. Richer responded that they wanted to work in parallel with the evolution of the PerDiS platform in order to obtain a result. He noted that the simulation included essentially the same allocation strategies as PerDiS.

Malcolm Atkinson [MPA] noted that it is certainly feasible to use the web as an accessible application workload, but wondered if the web can give any real information relating to what goes on in distributed applications. His concern is that the web has no reachability graph that makes much sense, nor update models that are managed in any organized way. He wanted to know how one might transfer knowledge from a Web-based study to other application domains. Richer conceded that this is a difficult point to counter. At the beginning they only wanted to study true persistent applications, but these are not easily available, so they wanted to find something comparable between the Web and some persistent applications. Unfortunately, this comparison is not very encouraging, but a better match might be confirmed by other measurement. He didn’t think that the results would transfer

G.N.C. Kirby, A. Dearle, and D.I.K. Sjøberg (Eds.): POS-9, LNCS 2135, pp. 157-160, 2001.

directly, though they can yield some information about widely distributed data; in fact, he could think of nothing more widely distributed than the Web.

Atkinson then made another comment, stating that the talk began by asking how many cycles are not garbage collected by PerDiS since they span the units of garbage collection (i.e., the bunch), but that the paper doesn't seem to address that question directly since it doesn't ask what bunches are going to coincide at any given time while garbage collecting. Again, Richer conceded the difficulty of answering this, since it would require experiments that drive the real behavior of PerDiS with real users from different sites and to measure what is never reclaimed. Currently, the PerDiS platform is not used in this way so there is no real way to measure it yet. They tried in this study to measure the maximum number of cycles that might not be reclaimed; there may be fewer, but they wanted an upper bound. Without real experience of the real platform they cannot say how much for real is not reclaimed.

Alex Wolf [AW] wondered why the study used a simulation instead of analysis, noting that the workload and data are stable enough for analysis. Richer responded that simulation seemed easier, since it directly encodes the basic policies used in the actual PerDiS platform. He said that they used parts of the real platform in the simulator, so it was simpler to simulate by using the allocation policies directly from PerDiS.

Wolf then made a second observation in that the study took a very traditional programming language approach, worrying about clustering for the particular reason of garbage collection only. He noted that with application to persistent object systems there are so many other reasons to form clusters, such as access patterns, units of transfer between clients and servers, locking granularity, and so on. Looking at things from the one perspective does not give such an interesting result with respect to persistent object systems. In reply, Richer conceded as much, but also stated that they were dealing with PerDiS and wanted to address that alone, in which garbage collection is an important part of the system.

The second paper, by Garratt, Jackson, Burden and Wallis, was presented by Andrea Garratt [AG]. Their study compares two Java-based persistent storage mechanisms and their suitability for implementing the indexing component of an experimental World Wide Web search engine. The platforms compared are PJama and an unnamed commercially available object-oriented database system. The authors were interested in both ease of use and performance, on which scores both systems were judged fairly easy to use, while PJama demonstrates significantly superior performance, though at the cost of more disk space. The following discussion ensued.

Lewis asked what is needed to make a persistent object system successful for a commercial web index. Garratt's answer was that the sheer performance needs to be scalable, and that they did not appear to be getting good scaling from either of the stores they were using, especially regarding response times for very large postings lists. Garratt posited that for sheer performance they would need to implement their own disk management software to get results, but that they were pleased with PJama for what they were doing.

Tony Printezis [TP] commented that with everything in postings lists, sheer performance would be difficult to achieve. Garratt's answer was that compression, and storing the lists contiguously on disk, should improve performance.

Fausto Rabitti [FR] then offered an observation that the work seemed naive from the point of information retrieval, so they cannot compare the results with information retrieval systems for web or for text. There are many things in the literature that could

be used here to improve performance. His major point was that the long postings lists must be scanned sequentially, whereas other systems would avoid this by applying techniques to eliminate the post work and get a superset of results which can then be post-processed. He also noted that commercial systems based on basic index strategies go directly to disk, and that indexing in an object-oriented database system is not the same as in PJama, where the B+-tree index must be programmed on top of the system instead of being built-in. Rabitti found it strange that they did not use the internal indexes of the object-oriented database system making the comparison with PJama seem unfair.

Steve Blackburn [SB] followed up by saying that the question is really whether orthogonal persistence can provide a panacea for data management, and that one should not necessarily assume that an orthogonally persistent system will be suitable for all data management tasks. Blackburn felt that one should be asking what the limits are to orthogonal persistence, what it can be applied to, and whether the applications used in these papers are good applications for orthogonal persistence. Garratt answered that her group is pleased with the results for PJama, and that for an experimental search engine such as theirs, PJama offered a good platform in which to try out ideas. They might look for something else for better performance. Bernd Matthiske [BM] asked what scale of application might need to be supported. Garratt indicated that in future they intended to use much larger corpuses (on the order of terabytes).

Tony Hosking [AH] asked if the time measure in the results was wall-clock or CPU time, and if it included the time to access the disk. Garratt responded that the measure was system and user time. Hosking responded that this would not then include disk access time.

Eliot Moss [EM] had a question regarding text indexing, and referred back to work by Eric Brown from the mid-1990s where implementing special access methods and representations for inverted lists inside the Mneme object store seemed to be better than using whatever data structures one gets with a high-level language. Moss asked Garratt to comment on that, in terms of both compactness and speed. Garratt responded that they simply wanted to test out ideas and so did not go down that path. Compression might yield performance improvements.

The whole discussion then took a more general turn. Atkinson commented that he was experiencing a strong sense of *deja vu* from the 1970s, in which he attended lots of meetings about relational database systems and heard many people saying that they made life so much easier, while others claimed they could never go fast enough so one should not pursue them. Atkinson felt that if one takes any technology and tries to map it to disks (which are very complicated beasts), while also supporting a wide range of applications, it takes a very long time and effort in optimization to make it go fast enough. He noted that there are many applications that one can throw at a mature relational database system that will make it crawl. Atkinson felt that it is important to look at the real costs of building applications both in terms of development hours and the number of skilled programmers needed. It would be nice if one could actually measure that. From the Wolverhampton team he was hearing that PJama helped in development time, but he wasn't hearing anything from the PerDiS group on that score.

In response, Richer said it was difficult to quantify. PerDiS was designed from the outset to be easy to use by application programmers, for whom distribution is not familiar, so they wanted to support something that increased productivity and the

possibility of new applications without sophisticated knowledge of distributed systems. Richer felt that PerDiS was a success on this count, but that for now performance was a problem. He felt that if the abstractions provided by PerDiS were useful then performance could be enhanced in many ways, but it might take a lot of work.

Ron Morrison [RM] followed with a comment on Atkinson's comment, saying that we might be trying to be far too ambitious. The research space is enormous, but there are a number of valid ways to approach the research. One approach is to build a new computational artefact that has not been built before. In this community persistence and hyper-programming are examples of this. There are many applications that could not have been built without the infrastructure that these technologies provide. Within the Snyder classification these constitute proofs of existence and yield valid science. Alternatively, one can seek performance that is better than prior implementations by showing a new techniques has an edge under certain conditions—proofs of performance. Trying to do that for the whole research space is impossible since it is just too big. Another approach is to demonstrate that a particular configuration of ideas or an approach achieves predefined objectives—this is proof of concept. As a community we must guard against knocking results because they are „not faster" or „not better". There is always an argument that will take away from those results, but that criticism itself must be judged on its pertinence to the issues addressed by the work it criticises.

The last word was had by Blackburn, commenting on orthogonal persistence and the temptation to take it and find direct applications for it. Orthogonal persistence provides an abstraction to help the programming task. He outlined his experience working with some data mining people who were interested in using orthogonal persistence in their domain. Blackburn found that the key people involved were mathematicians devising models using Mathematica, which were then instantiated into code to run data mining systems. He looked at producing Java bytecodes directly from Mathematica and semantically extending those with persistence abstractions. This would have given the data mining people something they could have used: the mathematicians could use the tool they were familiar with, while persistence abstractions sufficient to their needs were incorporated transparently, even though it was not really full-blown orthogonal persistence. Blackburn wondered if applying orthogonal persistence directly is the best way, or whether we should instead take our experiences with orthogonal persistence and extend from them into other domains.