Chained Declustering using Multiple Conventional Filesystems

Koichi Konishi  Kentaro Kikuchi  Hideki Kawai  Kunihiko Kojima†
Ken'ichi Ohmachi†  Susumu Akamine  Toshikazu Fukushima
NEC Corporation  NEC Informatec Systems
4-1-1 Miyazaki Miyamae  3-2-1 Sakado Takatsu
KAWASAKI Japan 216-8555  KAWASAKI Japan 216-0012
{k-konishi@cq,k-kikuchi@d,hideki-kawai@ab}.jp.nec.com
{s-akamine@kt,t-fukushima@cj}.jp.nec.com

Abstract

We have designed a Parallel Web search engine, implemented on a PC cluster, that employs chained declustering of index files stored in conventional filesystems. While the declustering scheme initially resulted in poor disk cache locality and would have been outperformed by mirroring, we were able to overcome this problem by lowering the frequency of work reassignment. This improved cache locality and helped achieve better throughput than mirroring.

1. Introduction

One of the most successful applications of cluster computing is large scale Web searching. The current largest commercial Web search service [5] is built on top of PC clusters, employing both parallel processing and mirroring to attain high performance and reliability.

Mirroring of a parallel processing cluster is expensive, however, requiring at least twice as many servers as the original configuration. Moreover, once any one node is down, it will perform only as fast as the original configuration because then the mirror will be left alone to perform both its own tasks and those of its partner.

Chained declustering [3], in which two copies of every data block are stored on adjacent servers, and every pair of adjacent servers has data blocks in common, has been successfully used to avoid this problem in non-search-engine applications. It has been used, for example, to provide raw disk I/O access to a database management system[3], to a filesystem[4], and to other applications [2].

We have found, however, that when the scheme is used to provide access to files on a conventional buffer-

Figure 1. Replication schemes

caching filesystem for an application like a Web search engine, cache locality tends to be poor and performance is worse than that of mirroring. In this paper, we discuss the problem and propose a preliminary solution.

2. Platform

Our platform, NAMIS-V2[1], is a parallel Web search engine implemented on top of a Linux PC cluster. It exploits intra-query parallelism. An entire set of documents is range-partitioned into subsets according to their document IDs. An index is then built for each subset and stored in the ext2 filesystem running on each engine server (replication, in which copies of indices are stored on other servers, may also be performed). A query is received by a scheduler process on the entrance server. When there has been no replication, the query is distributed to all engines, each of which independently uses its index to conduct a local query. All the local results are then gathered by the scheduler to produce a complete result for the query.

3. Replication Schemes

We evaluated four declustering schemes: simple parallel(SP), mirroring(MR), query tossing(Qt), and range assigning(RA). In SP, there is no replication at all; in the others, all data is replicated once (Figure 1). In MR, each engine has only one index, a copy or an
4. Performance Evaluation

We ran a performance evaluation using 1.2 million Web pages and a series of 50,000 (71,000 for RA) queries taken from the access log of NEC's Web search service[6]. The evaluation ran on a cluster of 9 PCs (an entrance and 8 engines). Each engine contained 200 MHz Pentium Pro, a 128 MB memory, and a 4 GB SCSI disk for indices. All PCs were connected by Myrinet as well as by 100BaseT Ethernet. All measurements started with a cold disk cache.

Table 1 shows the throughput of the replication schemes, normalized with respect to the throughput of SP with indices of the same type. All numbers are for the last 10,000 queries. The third and fourth columns show the throughput of clusters with, respectively, all engines up and one engine down. MR outperformed QT both in the normal and failed modes. We have yet to determine throughput for MR with merged indices, but assuming it to be about 100% in the failed mode, which seems very likely, RA would perform 15% better.

In RA, in order to simulate an outage of engine 1, the scheduler was statically programmed to use range assignment A (Table 2) for the first 30,000 queries and B for the rest. Figure 2 shows that the throughput of RA dropped to 43% of that of SP immediately after the outage, then caught up to SP after processing about 5000 queries in 132 seconds. It was 15% better than SP for the last 10,000 queries.

QT performs poorly because the same data read from an index for a frequently-issued query occupies the disk caches of two engines at the same time. This happens because a frequently-issued query is tossed to one engine at one time then to another engine the next time. QT inherently relies on frequent assignment changes for load balancing. Double caching also occurs in RA when assignments change, but then rapidly dissolves away since in RA the next assignment change does not necessarily occur soon.

5. Conclusion

Chained declustering of data stored in conventional filesystems shows poorer performance than mirroring when work reassignment is performed frequently, but less frequent work reassignment results in better performance than mirroring in the failed mode.

References