File Caching in Data Intensive Scientific Applications on Data-Grids

Ekow Otoo[1], Doron Rotem[1], Alexandru Romosan[1], and Sridhar Seshadri[2]

[1] Lawrence Berkeley National Laboratory,
University of California, Berkeley, California 94720
[2] Leonard N. Stern School of Business, New York University,
44 W. 4th St., 7-60, New York, 10012-1126

Abstract. We present some theoretical and experimental results of an important caching problem which arises frequently in data intensive scientific applications that are run in data-grids. Such applications often need to process several files simultaneously, i.e., the application runs only if all its needed files are present in some disk cache accessible to the compute resource of the application. The set of files requested by an application, all of which must be in cache for the application to run, is called a *file-bundle.* This requirement introduces the need for cache replacement algorithms that are based on file-bundles rather then individual files. We show that traditional caching algorithms such as *Least Recently Used (LRU)* and *GreedyDual-Size (GDS)* are not optimal in this case since they are not sensitive to file-bundles and may hold in the cache non-relevant combinations of files. We propose and analyze a new cache replacement algorithm specifically adapted to deal with file-bundles. Results of experimental studies of the new algorithm, using a disk cache simulation model under a wide range of conditions such as file request distributions, relative cache size, file size distribution, and incoming job queue size, show significant improvement over traditional caching algorithms such as GDS.

1 Introduction

1.1 Overview

Data intensive scientific applications concern application software that have very large data and storage resource requirements. Such applications are becoming increasingly prevalent in domains of scientific and engineering research. Examples include long running simulations of time-dependent phenomena that periodically generate snapshots of their state as in Astrophysics and climate modeling, simulation of combustion phenomena, and very large data sets generated from experiments such as BaBar [1] or the Large Hadron Collider (LHC) in the domain of high energy particle physics. The large datasets, from simulations and actual experiments, are preprocessed and maintained in units of files on geographically dispersed mass storage systems of a data-grid. Subsequent data analyses and visualization applications retrieve subsets of these files into locally accessible disk storage systems of high performance computing resources. This gives rise to

J.-M. Pierson (Ed.): VLDB DMG 2005, LNCS 3836, pp. 85–99, 2005.

large demands for disk and tape storage resources, and high network bandwidth. The disk storage effectively cache the requested files according to the demands of the application.

Caching has long been recognized as one of the most important techniques for reducing bandwidth consumption [2,3]. The general use of the term caching implies a specialized buffer storage that is used to speed up access when the data is transferred between different levels of a storage hierarchy with different characteristics: speed of access, size and cost per bit (see Fig. 1).

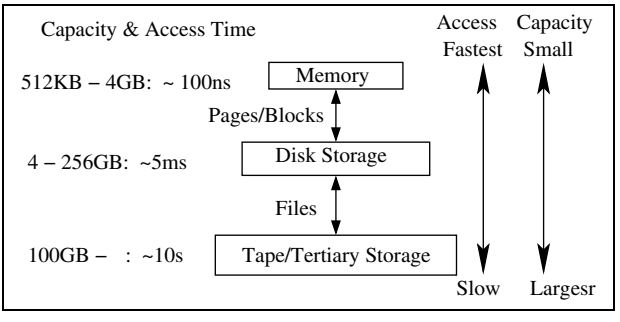

Fig. 1. The Different Levels of Caching in Data Intensive Applications

Successful caching relies on two properties of the access patterns of most application to be effective: *temporal locality* - if a file is accessed once, it is likely to be accessed again soon; *spatial locality* - if a file is accessed then files in close proximity (e.g., on the same storage tape) are also likely to be accessed. In the context of data-grids [4], a middle-ware service such as a storage resource manager [5] or distributed caching service [6], provides staging disks for files being requested. The disk cache manager often has no knowledge of the request stream of files being requested into the cache. Consequently, many cache systems are based on recognition of patterns of either recently used or frequently used files and use this to determine which files should be kept in cache and which should be evicted.

1.2 Problem Description and Previous Work

Consider a sequence of jobs that make requests for files at a computational resource where each job is comprised of one or more file requests. The requests are serviced in some order: *first come first serve (FCFS)*, *shortest job first (SJF)*, etc. A cache C of some fixed size $s(C)$, is available for storing a subset of all the requested files. A job is serviced only if all the files it needs are already in the cache C, otherwise it waits in a queue until all its requested files are transferred from a Mass Storage System (located either locally or at a remote site), into C. These data transfers cause time delays for the job execution, as well as consumption of valuable resources such as network and data storage bandwidth.