DynamO: Dynamic Objects with Persistent Storage

Jiong Yang, Silvia Nittel, Wei Wang, and Richard Muntz
Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90095
{jyang, silvia, weiwang, muntz}@cs.ucla.edu

Abstract

In light of advances in processor and networking technology, especially the emergence of network attached disks, the traditional client-server architecture becomes suboptimal for many computation/data intensive applications, e.g., data mining, scientific computing, image processing, etc. In this paper, we introduce a revised architecture for this kind of application: the dynamic object server environment (DynamO). The main innovation of this architecture is that the functionality of a persistent storage server is divided into modules which are dynamically migrated to the client on demand. Also, data is transfered directly to the client's cache from network-attached disks, thus avoiding multiple copies from a disk to the server buffer to the network and the client. In this way, a client only places a small load on the server, and also avoids the I/O bottleneck on the server. Furthermore, DynamO employs a distributed cache management allowing several clients to share the in-memory data by using the concept of "who uses it, serves it to others". We show via simulation models how this architecture increases the system's adaptability, scalability and cost performance.

1 Introduction

Client-server architectures have been popular over the past decades, and have also gained wide acceptance in the database community. In a client-server architecture, the system code is divided into two portions: one portion, the so-called *server*, provides basic services such as data I/O, buffer management and concurrency control and runs on a dedicated machine such as a workstation or SMP, and another portion, the so-called *client*, which provides the API and executes on the same application machine. Normally, a server interacts with many clients. In such an architecture, the scalability and performance of the overall system significantly depends on the compute power, aggregate bandwidth, etc. of the server machine, the data I/O rate of clients, and the scalability of the server itself.

Advances in processor and local area network technology make it possible for different software architectures to emerge. In recent years, two major trends in hardware development have impacted the efficiency of the client-server architecture: the emergence of network attached storage, and the increase of CPU power of server and client machines. At the beginning of the nineties, the typical bandwidth of a "fast" network was in the range of 10 Mbit/sec while a typical system bus' bandwidth was in the range of tens of MByte/sec second making the network the bottleneck for data intensive computing and disk I/O. Servers used directly attached disks and often processed data first and sent the output data to clients, thus, reducing the load on the network. Today, point-to-point connected fibre channel is capable of transferring data at 100 MByte/sec and the industry projection is that its bandwidth will reach 400MByte/sec soon [FCA] while during the past decade the sustained bandwidth of the typical system bus has increased only several tens of MByte/sec. Thus, the network is no longer the bottleneck in a LAN environment and it is feasible to attach storage devices directly to the network instead of to a server machine.

The rate of increase in the bandwidth of a single disk is about 40% a year, and the price per MB drops about 60% per year [Gro96]. By the year 2000, a megabyte of disk will cost about four cents and each individual disk will sustain a bandwidth on the order of 40 to 50 MB/sec. This trend suggests that systems will have much higher aggregate disk I/O bandwidth in the near future (400 MB/sec point-to-point) if attached to a fibre channel based network. As a result, large aggregate disk bandwidth is not a problem until we consider how to get the data to the processor considering that the sustained bandwidth of a single S-bus or PCI bus is in the range of 20 to 60 MB/sec [Arp97] today, especially if we assume a single server machine that performs the disk I/O for many client machines.

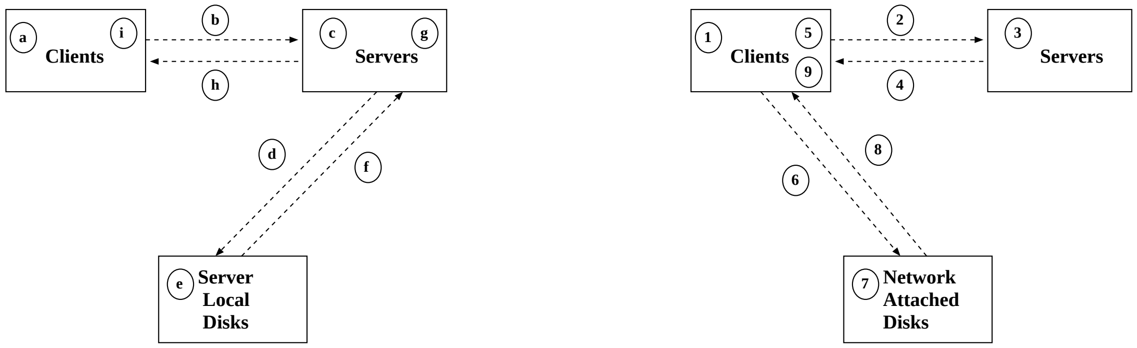
Another relevant hardware trend concerns the CPU power of client and server machines. Ten years ago, server machines were equipped with much more powerful CPUs than the client machines. As a result, servers were designed to perform most of the work within a client-server based system. These facts have changed dramatically during the past ten years. Today, client machines are equipped with powerful CPUs similar to server machines; furthermore, they are normally less utilized than server machines. The average number of CPUs in a server machine is up to thirty (in an SMP machine) compared to two or four CPUs in current client workstations; however, for applications that do not exhibit high parallelism, it does not necessarily increase response time if more work is transferred to the clients. Usually, there are many more client machines than server machines and the aggregate compute power of the clients is often more than that of the server machines. (Under these circumstances, moving same of the workloads to clients can relieve the server and reduce queueing delays.)

Based on the technology trends discussed above, we introduce a more scalable approach to persistent object systems: *Dynamic Objects with Persistent Storage* (DynamO). DynamO provides an application interface and functionality similar to traditional persistent object systems (POS) such as Exodus [Car86], Mneme [Mos88] or KIOSK [Nit96], and allows the storing, clustering and retrieving of storage objects which each consists of an object identifier and an unstructured byte container. Dynamo's architecture, however, is different from the traditional client-server architecture of such systems. The system has a layered architecture consisting of an I/O layer, a buffer management layer, and an object management layer, also providing transaction management. In DynamO, the object management layer resides on the client machine, and interacts with a server part on the server machine. However, the server part is much smaller in DynamO, and acts like a coordinator. At data access time, the necessary server code for buffer management, and catalog information is *dynamically downloaded* to the client machine, and then runs on the client machine. (The dynamic download is not really necessary; alternatively it could reside permanently on the client machine.) The object management layer communicates with the coordinator on the server machine about the location of relevant data. However, instead of loading data through the server machine, the object management on the client machine interacts with DynamO's I/O layer that resides on the disk controllers of the network-attached disk. This I/O layer performs physical and logical device management, and provides the abstraction of data pages to the object management layer. Requested data is directly retrieved from the network-attached disk and cached locally on the client machine, thus, eliminating the bottleneck caused by the server's bus bandwidth limitation (see Figure 1).

DynamO eliminates the traditional buffer architecture of POS in which the system buffer resides on the server machine. In DynamO, each client acts as a cache for local data that is shared with other clients, thus, providing a distributed cache. Since the collective memory on all client machines is usually much larger than that on server machines, the cache hit rate can be improved, and disk I/O avoided. For example, it is reported that the cache hit rate doubles in the NOW environment which employs this type of distributed client cache scheme [And96].

The redesigned architecture of a persistent object system accounts for higher performance and significantly improved scalability. In a data intensive computing application, such as data intensive persistent programs, the server machine's bus can easily become a bottleneck in the traditional architecture. However, in DynamO, since the data does not go through the server machine's bus, this bottleneck is eliminated. On the other hand, for those computation- and data-intensive applications, such as database applications with a large number of clients a large percentage of work is done on server machine(s), so that the server machine' CPU is highly utilized in the client server environment, thus, impacting the performance of the overall system. In DynamO, most of this same work is done on client machines, thus, the server machine's CPU will not saturate as quickly and the proposed architecture is more scalable. Also for data-intensive real time applications, such as multimedia, the number of clients a server can accommodate is limited by the server machine's compute power and the aggregate bandwidth it can support in the client server environment. However, in DynamO, it is only limited by the aggregate bandwidth of the network and the aggregate compute power on client machines, which is much larger. Our simulation results show that DynamO has much better scalability and performance than the traditional client server architecture.

We do not claim that DynamO works better than the traditional client server architecture for all applications. However, if an application requires a large number of CPU cycles and/or access to a large quantity of



(a) The Client-Server Architecture

(a) An application is invoked in the client machine. (b) The client sends a request to the server. (c) The server processes the request and locate the necessary data. (d) The server sends the I/O request to the its local disks. (e) The local disks retrieves the data. (f) The local disks send the data to the server via PCI bus. (g) The server processes the data. (h) The server sends the results to the client. (i) The client continues its work.

(b) The DynamO Architecture

(1) An application is invoked in the client machines. (2) The client sends a request to the server. (3) The server processes the request. (4) The server sends the necessary code and the handler of the data to the client. (5) The client processes the server's message. (6) The client sends the data I/O request to the Network Attached Storage (NAS). (7) The NAS retrieved data. (8) NAS sends the data to the client. (9) The client executes the code on the data.

Figure 1: Application Processing Paradigms

data, then moving method execution to clients and enabling direct access to storage devices can effectively remove the server bottleneck.

The remainder of this paper is organized as follows. We introduce related work in Section 2, and discuss the requirements for and problems of a distributed persistent object system in Section 3. In Section 4, we present and discuss the DynamO approach, and compare the performance of DynamO with the traditional approach to persistent objects systems in Section 5. Section 6 contains our conclusions and future work.

2 Related Work

Work related to DynamO can mostly be found in the area of file systems in distributed environments using network-attached disks, and the research done on network-attached disks, distributed cache management, and delegation of processing to clients.

2.1 Serverless Network File System

As clients are added to a LAN, the file server can become saturated. To address this problem, the serverless network file system (xFS) was developed on the Network of Workstations (NOW) at the University of California at Berkeley [And96]. All workstations are connected by a fast local area network and disk devices are attached to all workstations. In NOW, part or all client workstations can act cooperatively as a file manager or storage server, or both. Here a file manager maps a file into a set of pages while a storage server maps pages into disk blocks. As a result, this file system architecture provides high scalability. Moreover, it employs a cooperative cache technique. When one client tries to access data which is not cached in its memory, it asks the distributed file manager for that data. In turn the file manager checks whether or not the data is cached at some other client. If so, the cached data is sent to the client. Otherwise, the requested data is fetched from disks.

NOW successfully uses all workstations' memory and bus bandwidth. As a result, the cache hit ratio is higher and better scalability of the file server is achieved [And96]. Moreover, client workstations not only act in concert as the file manager and storage server, but also to execute application code. Therefore, the workloads on clients can be highly variable and the utilization of resources, e.g., memory, CPUs, disks, and system buses, can be quite different among peer workstations. Which pages of a file are served by which file manager is determined statically. This means that a file manager manages a fixed set of pages. In an environment where the resources can be frequently changed, (e.g., people bring in their own laptops and plug into the network in the morning, and bring the laptops home at night) how to utilize these resources as file managers becomes a challenge which is not addressed in the xFS system. Thus, how to balance the utilization of resources among all collaborating workstations remains an open question. To balance the workload evenly, DynamO allows dynamic change of ownership (file manger) of files (data), thus it provides the machines to support adaptability to dynamic workloads and resources environment.

In addition, xFS was developed for UNIX file systems, and uses a log based file writing technique. However, this technique does not work well in the environment of databases or object servers because database and object server usually exploit explicit requirements for data allocation, e.g., sequential access which is not well served by a log structured file system. Furthermore, xFS uses traditional server attached disks. The storage server executes on client workstations and consumes many precious workstation CPU cycles. To conserve these workstation CPU cycles, DynamO puts much of the storage server functionality (i.e., I/O manager) on the network attached disk controllers.

2.2 NASD

Usually when a file server retrieves data from storage, it first copies the data to its own memory, then sends the data to the clients. In order to eliminate copying the data to the server's buffers first, some file systems today use the "Third Party Transfer" mechanism such as e.g. the Network Attached Secure Disks (NASD) system. The NASD is a currently research project at CMU [Gib97]. In NASD, a client contacts the file manager on the server machine when it tries to open a file. The file manager will verify whether the client has permission to access that file. If so, the file manager then notifies the disk controller and gives a file handle to the client. On subsequent read accesses, the client does not need to contact the file manager; the client can directly contact the disk controller.

NASD still employs a centralized file manager that enforces consistency control. In addition, the NASD project focuses more on security issues. DynamO is more focused on the issues of distributed cache management, it uses a distributed object manager and distributed cache manager which can provide better scalability.

2.3 Condor and Thin Clients

The server bottleneck has been a long fought battle, and several solutions have been proposed. We refer to two influential approaches: Condor and the 'Thin clients' architecture. Condor [Tan95] treats all workstations in its environment connected via a fast local area network as a pool of resources: memory, disk, and processors. If a workstation becomes idle and a job is waiting, the job is assigned to the idle workstation for execution. Once the user of the workstation invokes its own computations, the Condor controlling daemons will halt execution of any "visiting" job and move its execution to another machine. However, Condor is not designed for persistent object server because any program can be executed on any machine. DynamO addresses the persistent object server issues by dynamically loading the server functionality to all client machines.

The thin client architecture is another innovative approach to dynamically move the execution of application code. "Thin" refers to both the machine and the application. A Thin client (machine) has no hard disk and minimal memory, thus, this approach is only useful for simple computations using small amounts of data. In the 'thin clients' architecture, clients dynamically download the code and data that they need for execution of a user job. Data intensive execution, however, has to be moved to the server machine. Also, there is no cache kept on the client machine, and it is less feasible for database applications.

2.4 Related work to Cache Coherency Control

With the upcoming of distributed and shared disk environments, some related work to DynamO's cache coherency system has been done. Similar to DynamO, the work of [Dias89] combines the CPU power of several low end computer systems, and introduces integrated concurrency-coherency control to reduce the overhead of coupling a large number of systems. Furthermore, this system uses an 'intermediate' buffer that can be accessed by all systems so that data I/O is minimized for all participants. It is not clear whether in this approach the intermediate buffer is made available by one machine or via shared main memory of all the participating systems; however, like in DynamO each of the participating systems manages a disjoint region of the intermediate buffer. In contrast to DynamO, the intermediate buffer partitions are statically assigned here lacking the flexibility DynamO offers by assigning work based on usage pattern and actual workload of the client system. Furthermore, DynamO employs a more flexible scheme of allocating and managing chucks of memory in order to keep the management effort per page minimal since we assume that a large number of pages is managed in such a system.

3 Scalability and Performance of a Persistent Object System

Designing and implementing a persistent object system, for certain applications such as databases systems with a large number of clients, scientific data- and computation persistent applications and multimedia applications the scalability of the client-server architecture and the server bottleneck resulting from a server machine's system bus bandwidth limitation are important design issues to consider. In this section, we describe the basic architecture of a persistent storage system for data- and computation intensive applications, and discuss its bottlenecks and problems related to scalability.

3.1 Overview

Storage object servers have been developed as storage back-ends for persistent programming languages, and non-standard database management systems (DBMS) such as object-oriented DBMS. Today, this storage system technology is also used for high-intensity applications such as large database systems with many users or as storage systems for data- and computation scientific programming systems as well as multimedia applications.

A storage system offers applications storage objects consisting of a storage object identifier and an unstructured, variable-sized byte container. The middle layer between the I/O manager and the application interface consists of a specialized buffer management, employing flexible buffering strategies that support the specialized access behavior of the above mentioned applications, and improve the buffer hit rate.

While the classical client-server architecture for a storage object server has performed well for OODBMS with a limited number of users and data, its architecture has limited scalability and performance for high-intensity applications. Typically, scientific and multimedia applications require a much larger data throughput then the more traditional applications for a storage object server. Also, traditional DBMS applications encounter scalability and performance problems if a large number of DBMS clients has to be served. For the remainder of the paper, we assume that both client and server machines are workstations. Assumptions about their performance characteristics will be described later.

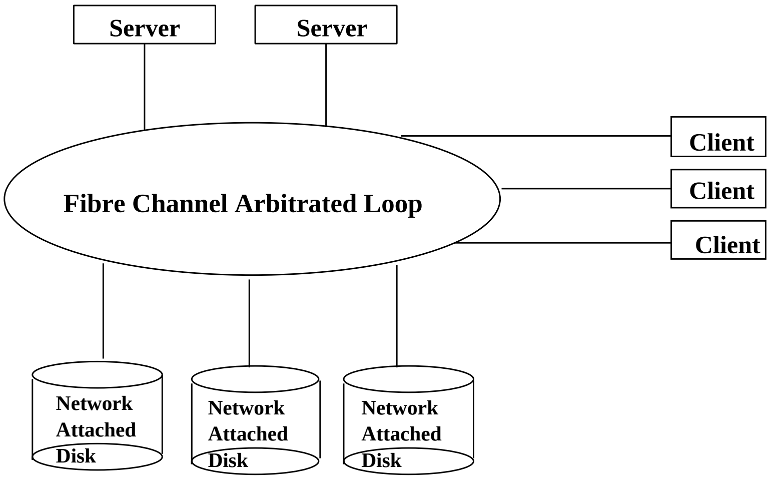


Figure 2: Fibre Channel Arbitrated Loop

3.2 Problems

In this paper, we will compare the performance of the DynamO architecture with the traditional client-server architecture. The main purpose is to (a) evaluate the potential benefits of the DynamO approach and (b) understand what first order factors influence the performance tradeoffs.

We use bottleneck analysis to discuss some of the major aspects of performance comparison between two system models: traditional client-server and DynamO. The arguments in this section are more qualitative and are meant to serve as a road map for the more detailed simulation results which follow in later sections. We are interested in how the bottleneck shifts in response to changes in the arrival rates, the service rates, and so on. Figure 3 illustrates some aspects of the queuing network model for the general system environment.

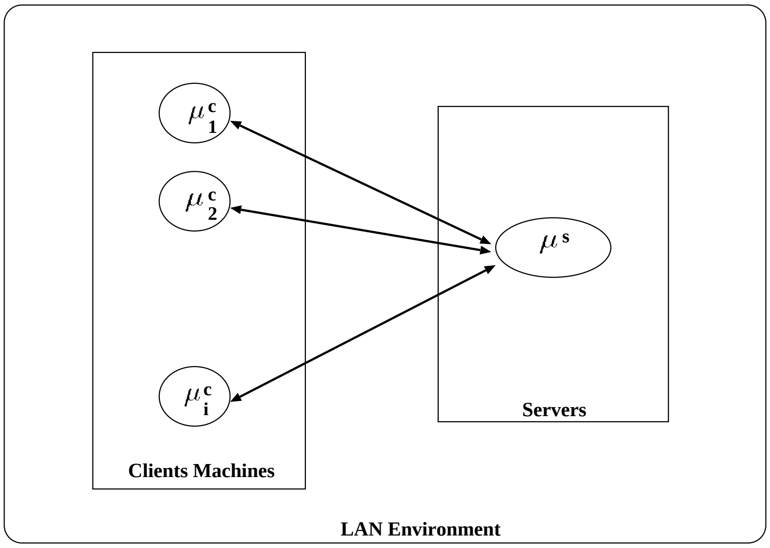


Figure 3: High Level System Description

In a LAN system, many different applications may be running at the same time. Without loss of generality, we assume that only one kind of application exists in the LAN for simplicity. We assume a closed queuing system, in which there are multiple client machines and only one server machine. The average service rate of client machine i is denoted by μ_i^c while the average service rate of the server machine is denoted by μ^s. μ_i^c include both the application processing time and client processing time because both of them are executed on client machines.

For an application, the average server service rate μ could be lager than the average client service rate μ^c. (in other words, the average server service time is shorter than the average client service time.) but the utilization level of the server machine can be much higher in the traditional client server model due to the large number of client machines. In the DynamO architecture, the μ^c gets smaller while the μ^s gets larger because a large fraction of the load is migrated to the client machines. Thus, the server machine utilization level is reduced significantly while the client machine utilization level increases only by a small fraction. Therefore, the server bottleneck can be alleviated.

We assume that storage devices are not a bottleneck for data intensive applications because we can use RAID systems to stripe the data over multiple disk devices. Thus, the server system bus is most likely to be the bottleneck in the client-server architecture. On the other hand, DynamO enables client machines to directly access the storage devices, and thus the aggregate system bandwidth is not limited by the server system buses.

4 The DynamO Architecture

A persistent object server can be congested in data/computation intensive applications. To eliminate this problem and achieve higher performance, we propose the DynamO architecture. The main idea of DynamO is to dynamically move object server functionality to client (machines) for execution, and allow the client machines to directly access from network-attached storage. In the following, we present the architecture and principles of DynamO, and focus on data I/O, cache management, and interfaces to applications. The architecture of DynamO is depicted in Figure 4; we will describe the system from the bottom up. From an application point of view, the functionality of DynamO is very similar to a persistent storage system such as Exodus [Car86], and Mneme [Mos88]; therefore, we will focus on architecture issues in this section.

4.1 I/O Service

The lowest layer of DynamO is the I/O layer providing data I/O from and to storage devices. The I/O layer maps files and data pages to storage locations on a storage device in a similar fashion to an I/O layer in a conventional persistent object system. To load data, DynamO employs a technique similar to that employed by NASD [Gib97]. When an application invokes an operation on an object, the object manager running on the client machine requests some storage objects from the server portion (coordinator) on the server machine by passing the object identifiers which contains the data blocks. However, instead of fetching data for the client, the server plays the role of a coordinator, and performs mapping of the object identifiers to pages. This mapping is performed centrally on the server machine, so that consistency problems for catalog information are minimized. The coordinator returns to the clients the relevant page and file identifiers as well as the disk identifier(s) of the disk(s) that contains the data pages. The NASD servers implement consistency control; in contrast, the DynamO coordinator does not implement consistency control. In DynamO, the clients cooperate in providing cache consistency. Further, the cache consistency protocol can be tailored to the objects.

After obtaining the page, file and disk identifiers, the object manager on the client machine directly interacts with the portion of DynamO that runs on the disk controller(s) of the network-attached disks. This I/O model is based on the following facts: since the disk controller's CPU is mostly only lightly utilized, we can use it for delegating the execution of the two lowest levels of the persistent storage system, i.e. disk block allocation and free space management as well as object page to disk block mapping (page manager). The page manager is divided into two components: a strategy component, and a storage allocation component. The allocation component uses the input from the strategy component to allocate disk blocks, while the strategy component decides how blocks are allocated for sets of pages. We assume that we have dedicated storage devices for the persistent object server available so that the allocation strategy is common for the entire disk (e.g., continuous block allocation, clustering, etc.). However, if this is not the case, the storage device is shared with other applications such as a page allocator for a Unix file system, we assume that a storage partition is allocated for the persistent object server, and that the allocation for this partition is managed via the DynamO page allocation strategy component. The I/O manager on the disk controller retrieves the relevant pages from the disk, and sends them to the requesting object manager. The pages are stored in the local cache on the client. The object manager, finally, performs the object to page mapping, and makes the requested storage objects available for processing. The I/O layers are illustrated in Figure 4.

4.2 Cache Management

Data retrieved from disk can reside in the client's cache. In order to avoid repeated retrievals from disk, we employ a distributed cache management scheme, and allow clients to retrieve data directly from other clients' cache. The cache management consists of two layers: the distributed cache management layer and local cache management layer as shown in Figure 4.

Hierarchical Model

Cache management is a complicated issue, mainly due to consistency requirements. To address this issue, persistent objects in DynamO is organized in a hierarchical manner, as shown in Figure 5. At the bottom level, the granule is a page, i.e., each entity represents a page. At the level above, each entity represents a set of pages with a varying number of pages per set. (In current version, the system user who creates these data pages decides which page set it goes into.) The granule at the next level up is a cluster, set of clusters, set of sets of clusters, and so on. On the top level, there are from a few tens to a few hundreds root entities, and each root entity representing a large set of data pages.

The coordinator on the server maintains a set of ownership tables for clients and a non-owner table (as illustrated in Figure 6). The non-owner table lists the entities that are not currently owned by any clients. We say "X owns entity Y" when the object manager X can grant read and write access to other clients for any data pages in Y, and the cache manager associated with X knows whether a data page in Y is cached and if so, where. (Note, that we do not assume all pages of Y are loaded in the cache of X) The reason that we use the hierarchical model to manage cache coherence is that it can minimize the bookkeeping overhead per object manager. (Looking ahead, the hierarchical model also provides for more flexible ownership transfer.) When object manager A requests some data from the DynamO server, the coordinator checks whether there

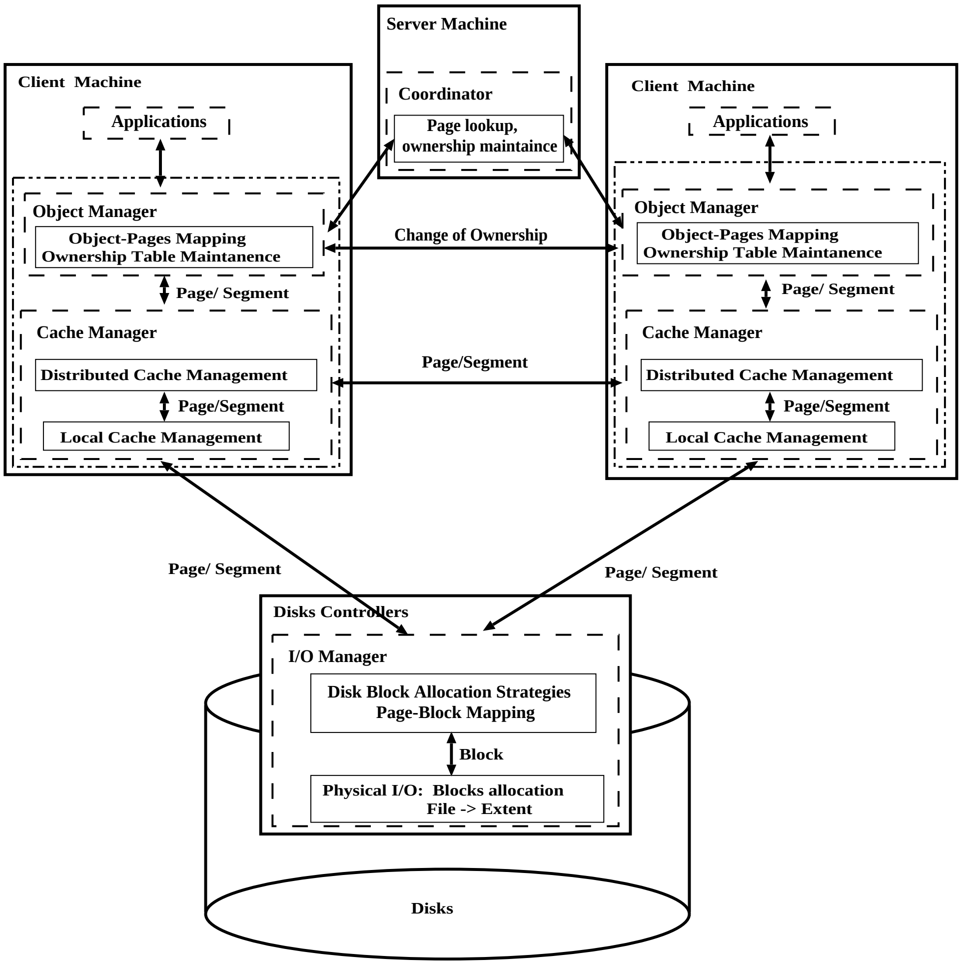


Figure 4: DynamO layers

is a client who **owns** the object being requested. If there is no such client, the server makes client A the **owner** of the entity, sends client A the handle for the entity, and make the proper entity in the owner table. On the other hand, if the data is owned by the object manager B, then the coordinator refers A to B.

Assuming, that object manager A is the owner of a cluster, it performs the I/O as described above, and downloads the cache manager code from the server if it does not have this code already. (This code includes the consistency protocol, different server may use different protocols. Moreover, a server may use different consistency protocols on different data sets.) The object manager caches the data in its local cache which is managed via buffer management strategies specified in the cache manager code from the server. As the owner of an entity, object manager A has the right to grant read/write access to any data in the entity to other clients.

If object manager A retrieves the information from the coordinator that the data is already owned by another object manager B on another client machine, object manager A interacts with this object manager B to request access to the data, and negotiates access rights and data granularity.

Ownership Transfer

When A accesses data currently owned by B, object managers A and B will also decide whether the ownership of the data should be transferred. If it is determined that an ownership transfer is desirable (e.g., object manager B thinks it will not need to access the data in near future,) then the question is what subset of the set owned by B should be transferred. To illustrate this process more clearly, let's assume that the entity in question is v' and v' is a descendant of v. Object manager B owns v and object manager A requests

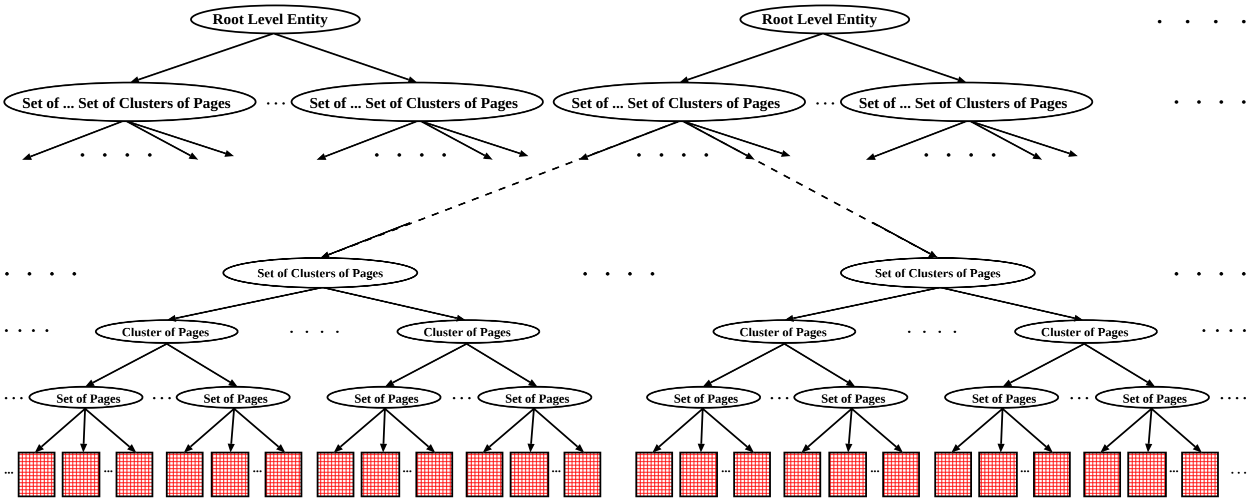


Figure 5: Hierarchical Object Organization

Object Manager	Entities
A	entity 1, entity 2, entity 4
B	entity 3, entity 5
•••	•••

Owners Table in Coordinator

Non-owner entities
entity 6, entity 8 •••

Non-owner Table in Coordinator

	write access	read access
entity 1	none	A, C
entity 2	A	A
entity 4	B	A, B

Ownership Table in Object Manager A

Figure 6: Owners Tables in Coordinator and Object Manager

data in v'. B decomposes v into a set of children entities, and one of the children contains v', B then checks whether it needs the child entity of v. If yes, then the children entity is decomposed and this process is repeated until there is an entity that is a descendant of v and contains v', which it is estimated, will not be used by B in the near future. Then this entity is the unit for ownership transfer. If such an entity does not exist, then no ownership transfer will occur. This is only one possible scheme with no claims to being optimal. For example, the algorithm described above does not account for the total size of entities owned by B or the frequency of access for each entity, etc..

If object manager B wants to transfer the ownership of v_1 to A, it will notify the coordinator. In turn, the coordinator starts an ownership transfer process involving A and B. This process is similar to a two-phase commit because it is very important that A, B, and the coordinator all agree that the transfer of ownership occurs or does not. The result is that B releases entity v_1 to A. Object manager B decomposes v to a set of disjoint entities v_1, v_2, $\ldots$, v_k where $\bigcup_{i=1}^{k} v_i = v$. Then B removes v_1 from this set. Now, instead of owning v, Client B owns v_2, v_3, $\ldots$, v_k. Also, B sends all information associated with v_1 (e.g., which client have cached data in v_1, which client has read/write locks on data in v_1, etc.) to A. There is a difference between being the owner of some data and locking the data. If a client want to lock some data, it have to ask the owner of that data to grant the lock. The owner of the data is not necessary locking the data currently, it serves the lock to others, e.g., scheduling the lock requests.

Object manager A, in turn updates its ownership table by adding v_1. Then, it prunes its ownership table to see whether it also owns v_2, v_3, ..., v_k. If it owns these objects, A would remove v_1, v_2, ..., v_k from its ownership table and put back v and continue the pruning process until no more sub-objects can be removed. The goal of the pruning process is to keep a minimal list of objects in the ownership table. Then, A downloads the code for cache coherence from the server if A does not have this code.

On the server side, after receiving the message of ownership change, the coordinator updates its own coordinator table which contains both client A and B's ownership tables using essentially the same procedure as A and B. When A requests the cache coherence code from the server, the server will simply return the code to client A.

Distributed Cache Retrieval

After obtaining the proper access permission, object manager A can ask its distributed cache management layer for the data. A distributed cache management layer (DCML) maintains a list of cached data of entities owned by it. The DCML on Client A will contact the DCML of client machine, say B, which owns the data. (A can contact the coordinator and the coordinate will tell it that B owns the data.) The DCML on client B knows which, if any, client caches the data. If the data is not cached, then B will send back a handle to the data file, page, and disk identifier(s), and a disk I/O has to be performed by client A. The local cache management layer (LCML) on A asks the proper network attached disks for the data.

On the other hand, if the DCML on Client B discovers that another client has cached the data (say Client C), then it sends a request to the DCML on Client C with the file id and page ids. The DCML on Client C will ask its LCML for the data, and the cached data is returned to Client A.

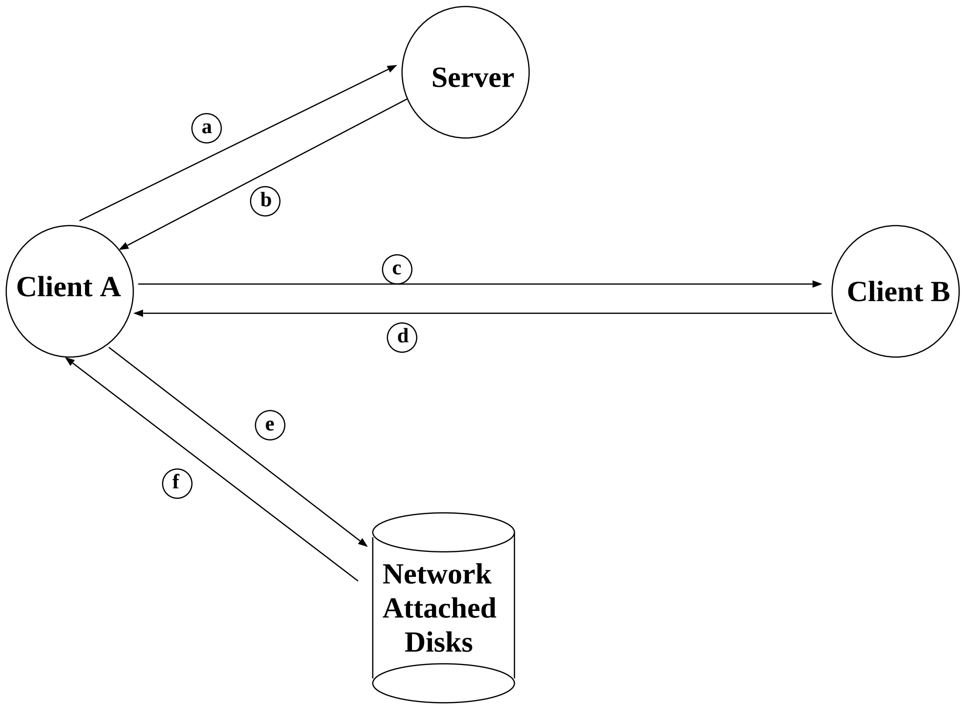
After client A finishes processing the data, it can return the ownership of the entity to the server by sending a message of **deletion of ownership** to the coordinator and removing v_1 from its ownership table. When the server receives a message of deletion of ownership. It removes the object from its owner and puts it to the no-owner table and prunes the no-owner table. This process is illustrated in Figure 7.

From the process described above, it clear that when a client requests data from the coordinator, the coordinator will return the largest entity that contains the data and not owned by any other clients. In other words, if a client requests an object at level 5 (a 'low level'), and there is a level 2 object in the non-owner table, which is an ancestor of the object requested, then the server will give the level 2 object to the client rather than decompose the level 2 object and give the client a higher level object.

Recovery from a client machine crash

We now discuss the scenario that occurs when a client crashes. If the crashed client is an owner of entities, then other surviving clients may request the data owned by the crashed client machine, and of course, these requests can not be served. We apply a crash recovery protocol to eliminate this kind of problem. When an object manager requests data from another object manager, it has a timeout mechanism. If the object manager does not respond within the timeout period, the requesting object manager will report to the coordinator that the other object manager could have crashed. Then, the coordinator will contact the specified object manager. If the object manager still does not respond, the coordinator will revoke all ownership of the specified object manager, and assign its entities to the non-owner table. Also during normal operation, each object manager writes logs and periodically puts it on the network attached storage so that during a crash, the coordinator can fetch the log (the location of the log on the disks is known to the coordinator) and roll back the changes to the last consistent state. This procedure is similar to that used in distributed server failure recovery. When a client machine recovers from a crash, the object manager on that client will contact the coordinator to retrieve the data that it needs.

In this scheme, the work on the server machine is minimal. In fact, only the coordinator remains on the server, and the coordinator only tracks who is the owner of each entity. Therefore, the congested server problem is eliminated. In addition, each object is owned by at most one owner. Therefore, cache coherency can be maintained by the owner. This avoids complicated distributed algorithms for cache updating. Moreover,



a: Client A asked the server for object data
b: If no other client owns the data, the server makes the A the owner of the data and returns the handler and goes to e. Otherwise, the server returns the id of the client that owns the data.
c: Client A asks B for the object data.
d: If some client cached data, Client B will ask the client that caches the object give A a copy of the cached object. Otherwise, if Client B wants to transfer the ownership of the object requested, B gives A a handle to the object and makes A the ownner.
e: Client A goes to the correct disks and asks for the relevent data blocks.
f: Data is returned.

Figure 7: The process of Data Fetching

the hierarchical object model makes the partition of cache management easy and fair. If one client accesses a large amount of data, it could serve a large amount of data to other clients. Furthermore, if an object manager requests some data from the coordinator, it may become the owner of a very large root entity. However, if other clients request data in this root entity, this client machine can transfer the ownership of some entities (which are descendant of the root entity) to other clients. As a result, the service partition is based on dynamic usage rather than on some static a priori partition, such as used in xFS. Especially in the environment that machines can join in and leave a network frequently, (e.g., people brings their notebooks to the office and plug them into the network in the morning and bring them home at night), a static partition may not work well. In addition, since there may be more than one server in a network, DynamO lets a client download the coherency control code from the server. A client can download different coherency control protocols from different servers for different objects and all of them can work concurrently and correctly as long as there is only one owner and one coherency control protocol for a given object.

4.3 Application Interface

DynamO's application interface is similar to Exodus. DynamO offers variable-sized storage objects consisting of a storage identifier and an unstructured byte container. Storage objects can be clustered, and are changed in the scope of transactions (however, not described in this paper). For the application, the fact that storage server functionality is dynamically moved between machines is hidden. Since portability is an important issue and heterogeneous platforms do exist, the movement of code is based on the principles developed and explored in FALCON [She97].

5 Benchmark Analysis and Comparison

In this section, we have chosen three typical applications as testbeds for a performance comparison of DynamO with the traditional client-server model. These three applications to the persistent storage system are data/computation intensive persistent programming applications (IPP), database applications, and multimedia applications. The estimated characteristics of these three applications are illustrated in Table 1.

	IPP	Database	Multimedia
Number of Clients	Low ($1 \sim 10$)	High ($20 \sim 40$)	Low ($1 \sim 10$)
I/O	High ($1K \sim 10K$)	Medium ($10 \sim 100$)	Constant ($50/s \sim 200/s$)
CPU Cycles (in million)	High (~ 1000)	Medium ($20 \sim 100$)	Constant ($\sim 20/s$)
Cache Hit Rate	Low ($\sim 10\%$)	High ($\sim 40\%$)	Very low ($< 3\%$)
Cache Coherence Cost	Low	High	None

Table 1: Characteristics of Three Applications

In order to fairly compare these two models, we assume that there is one server in the client-server model and one coordinator/server in the DynamO model. Moreover, we assume that no data will be cached on the client local disks. In addition, in the traditional client server architecture, we assume that the server always has to fetch the data from its disks through its PCI bus into its main memory if the data is not cached, then go through the PCI bus again onto the network on the way to the client. On the other hand, when the client requests data, if the data is not cached, the client will send a request to the network attached disks, in turn the disk controller fetches data into its buffer then sends it to the network, and the data flows through the PCI bus on the client machine and to the client's main memory. In this comparison, we assume all machines are the same (server machine and client machines). Each machine is equipped with a 100 MIPS CPU and one PCI-bus with sustained bandwidth of 80 MB/sec. In addition, we do not consider that disks are the bottleneck.

5.1 Data/Computation Intensive Persistent Programming

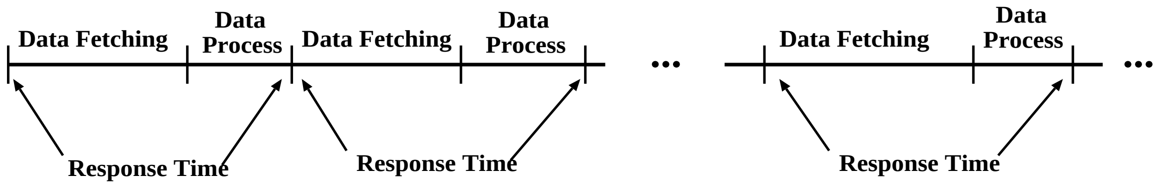


Figure 8: Measurement of Average Response Time

The path length of a typical IPP application consists of a sequence of data fetching and data processing phases as shown in Figure 8. A *data fetching phase* is the time interval during which the client fetches the relevant data to its main memory while the *data processing phase* is the time interval during which the client processes the fetched data. Then the procedure repeats. We are interested in the average response time, i.e., the average time it takes a client to fetch data and process it.

In IPP applications, the number of I/Os and the CPU consumption are very high. Although the majority of the CPU load is on the client, the server processors still spends a significant time per client job (approximately 2 seconds for each client job on average). Although the I/O request rate is very high, on average the PCI bus takes less than half second.

Figure 9(a) shows the effect of an increase in the number of clients. Since the number of sessions increases, the overall CPU increases, and once the server bus becomes congested, the average response time in the client-server model increases significantly. The average I/O per client is 3,000, and the average number of CPU instructions is 1000 million instructions and 20% of these are executed on the server and the remainder is executed on the client in a client server model. From this figure, it is clear that the server's CPU reach saturation around 4 clients. However, in DynamO, the server/coordinator only executes 1 million instructions since it only performs the role of coordinator while the client executes 1050 million instructions.

Figure 9(b) and (c) show the performance as a function of CPU consumption and I/O requests, respectively. Since we set the number of client to be 6 in this two case, the server's CPU is always saturated. Thus,

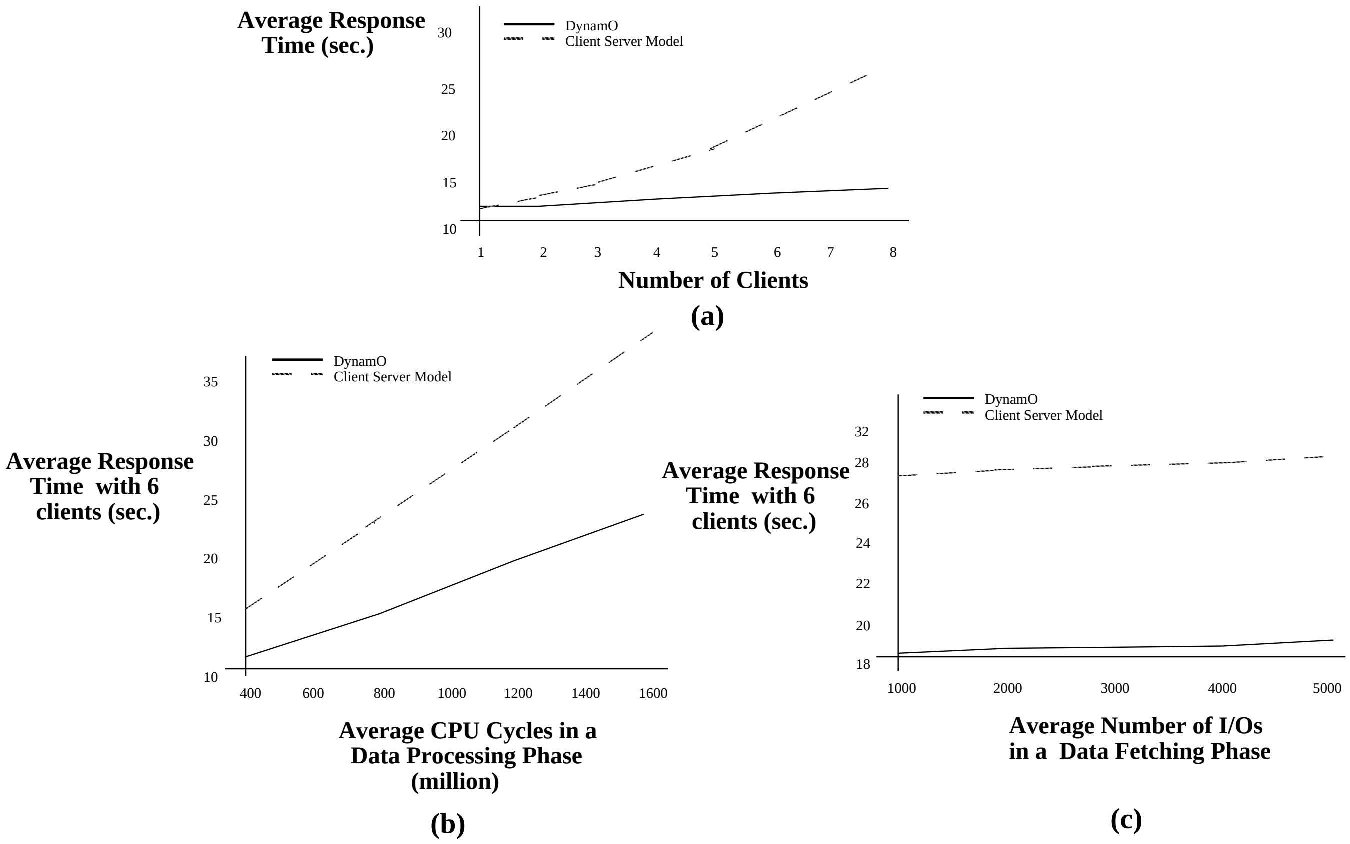


Figure 9: Performance of Data/Computation Intensive Persistent Programming

when the CPU workload is increased, the average response time of the client server model increases at a much faster pace than that of DynamO. because the bottleneck is in CPU rather than I/O. On the other hand, as the number of I/Os increases (as in Figure 9(c)), the performance of the client server model degrades at a similar pace as that of DynamO because the server machine PCI-bus is not the bottleneck.

5.2 Database Applications

In this environment, there are many clients; on the order of dozens. The cache hit rate can be very high in DynamO, since there are more machines that can serve as cache managers (i.e., each client machine can cache data for other clients). We assume that in the client server model, the cache hit ratio is 40% while the cache hit rate is about 50% for DynamO. Moreover, since there is significant work done on the server, e.g., cache management, transaction management, I/O service, etc., the server processor can become saturated. In turn, performance is impacted when the number of clients increases (scalability) or the workload increases. This is illustrated in Figure 10.

Figure 10(a) shows the relative scalability of the two models. We choose the average number of CPU cycles in a session to be 40 million. Among these cycles, 35% is executed on the server's processor while the rest is executed on the clients in the client server model. In DynamO, 1 million instructions per session are executed on the server while the client executes 42 million instructions per session.

In this type of environment, the server CPU is the bottleneck. As a result, Figure 10(b) shows that when the average number of instructions per session increases, the performance of the client server model is impacted severely. On the other hand, Figure 10(c) shows that as the number of I/O per session increases,

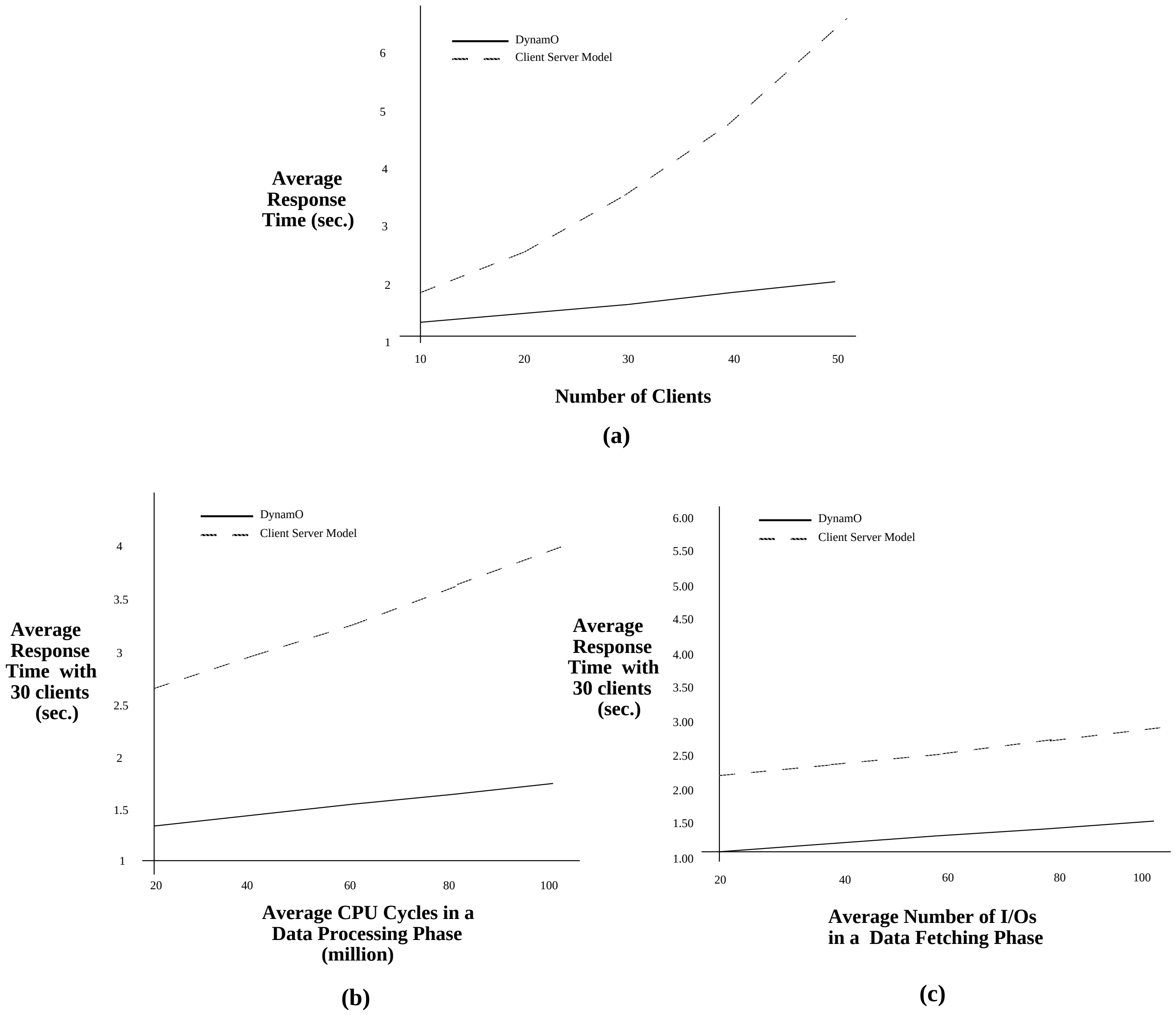


Figure 10: Databasee Applications

the average response time of an application in the client server model increases at a similar pace as the increase in DynamO.

5.3 Multimedia Applications

Multimedia applications are a special class of application. They require a constant stream of data. For example, MPEG-1 video requires 1.5 MBits/sec while MPEG-2 video requires 4-6 MBits/sec. The data decoding is done on client machines and there is no need for transaction management and coherency control because there is no write-back. Therefore, Multimedia is an I/O intensive benchmark.

Multimedia applications are real time applications, which require that a request has to be served within a specified time period. Therefore, instead of showing the average response time of an application, we show the number of multimedia streams that can be served by client server and model and DynamO in Table 11. Since multimedia applications are data intensive and the network attached storage has much higher aggregate bandwidth than that of client server model, the DynamO can accommodate much more streams than the client server model., e.g., the DynamO can serve around 250 MPEG-2 streams comparing to about 50 streams in the client server model.

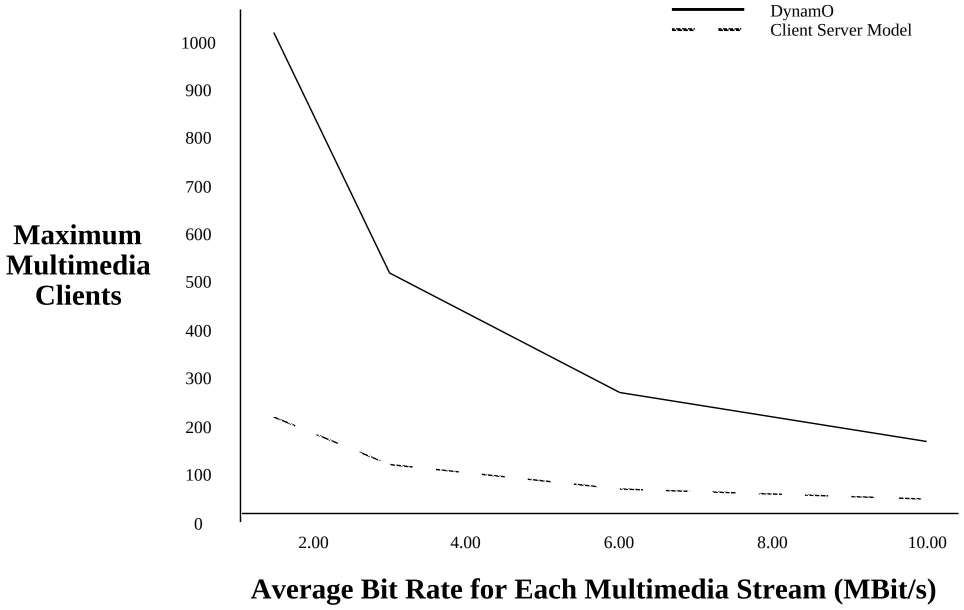


Figure 11: Multimedia Applications

6 Conclusion and Future Work

With new developments in computer hardware such as improved processor speed and network bandwith, as well as network-attached storage devices, there is also a need to reconsider software system architectures. In this paper, we introduced DynamO (Dynamic Objects with Persistent Objects), an alternate model to the client-server architecture for computation/data intensive applications that offers significantly improved scalability and performance. Instead of managing a file system buffer on the server machine, DynamO downloads most server functionality to clients, and also transfer data directly from network-attached disks to client machines, thus, eliminating the server bottleneck.

We studied the performance of the DynamO architecture. With temporal variation of application path length, DynamO has better adaptability because it dynamically changes the "client/server" compute power ratio automatically according to the workload. Moreover, an added client machine can not only share the "server" workload, but the "client" workload as well. Therefore, better scalability can be achieved. Although DynamO has extra cache coherence overhead, the percentage overhead is low. We believe that DynamO provides a more cost effective, scalable, and skew insensitive solution than the traditional client-server architecture.

Implementation for DynamO is currently underway at UCLA's Data Mining Lab. DynamO is implemented in C++ on Sun UltraSparcs using Solaris. We have installed fibre channel devices, including 4 Segate Fibre Channel disks drives and Febre channel adaptors for Sun workstations that we use as the hardware basis for DynamO.

References

[And96] T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, and others. Serverless network file systems. *ACM Transactions on Computer Systems*, Feb. 1996, vol.14, (no.1):41-79.

[Arp97] A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, D. E. Culler, J. M. Hellerstein, and D. A. Patterson. High-Performance Sorting on Network of Workstations. *Proceedings ACM SIGMOD International Conference on Management of Data*, pp. 243-254, May 1997.

[Blo96] R. Bloor. The Coming of the Thin Client. *Database and Network Journal*. vol.26, (no.4):2-4, August 1996.

[Car86] Carey, M.J., DeWitt, D.J., Richardson, J.E. and Shekita, E.J., Object and File Management in the EXODUS Extensible Database System. Twelfth International Conference on Very Large Databases, 1986.

[Dias89] Dias, D.M., Balakrishna, R.I. Robinson, J.T., Yu, P.S., Integrated Concurrency-Coherency Controls for Multisystem Data Sharing. IEEE Transactions of Software Engineering, Vol. 15, No. 4, April 1989.

[Fab97] F. Fabbrocino, E. C. Shek, and R. R. Muntz. The Design and Implementation of the Conquest Query Execution Environment. *UCLA CSD Technical Report #970029*, July 1997.

[FCA] Fibre Channel Association. *http://www.amdahl.com/ext/CARP/FCA/FCA.html.*

[Gib97] G. A. Gibson, D.F. Nagle, K. Amiri, F. W. Chang. File Server Scaling With Network-attached Secure Disks. *Performance Evaluation Review*, June 1997, vol.25, (no.1):272-284.

[Gro96] E. Grochowski and R. F. Hoyt. Future Trends in Hard Disk Drives. *IEEE Transactions on Magnetics*, May 1996, vol.32, (no.3, pt.2):1850-1854.

[Hei88] P. Heidelberger and M. S. Lakshmi. A Performance Comparison of Multimicro and Mainframe Database Architectures. *IEEE Transaction on Software Engineering*, vol. 14, No. 4, April 1988.

[Hei95] J. Heidemann and G. Popek. Performance of Cache Coherence in Stackable Filing. *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pp. 127-142, December 1995.

[Mos88] Moss, J. Eliot B. and Sinofsky, S., Managing Persistent Data with Mneme: Designing a Reliable Shared Object Interface, *Advances In Object-Oriented Database Systems: Second Int. Workshop on OODBS, Bad Münster, Germany*, 1988.

[Nit96] S. Nittel and K.R. Dittrich, A Storage Server for the Efficient Support of Complex Objects. *Proceedings POS-7 International Workshop on Persistent Object Systems*, Cape May, June 1996.

[Pat88] D. A. Patterson, G. A. Gibson, R. H. Katz, The Case for Redundant Arrays of Inexpensive Disks (RAID) *Proceedings ACM SIGMOD International Conference on Management of Data*, pp. 109-116, May 1988.

[She97] E. Shek, R. R. Muntz, and L. Fillion. The Design of the FALCON Framework for Application Level Communication Optimization. *Technical Report No. 960039*, Computer Science Department, UCLA, November 1996.

[Sun97] JavaBeans. *http://java.sun.com/beans.* Sun Microsystems.

[Tan95] T. Tannenbaum and M. Litzkow. The Condor Distributed Processing System. *Dr. Dobb's Journal*, February 1995.