DynamO: Dynamic Objects with Network Attached Persistent Stores

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

January 15, 1998

Abstract

In light of advances in processor and networking technology, the traditional client-server architecture becomes inefficient in many computation/data intensive applications, e.g., image processing. In this paper, we introduce a new architecture for this kind application: the dynamic object server environment. The main innovation of the new architecture is the dynamic migration of server functionality to the client. This enables client machines to execute a job (application request) directly without placing a load on the server other than an initial small load at method invocation. As a result, our proposed architecture can eliminate potential bottlenecks at the general servers. Also client machines do not need to waste cycles waiting for a response from a busy server machine. We show via simulation models how this architecture can increase the system's adaptability, scalability and cost performance.

1 Introduction

The client-server architecture has been popular over the past several decades. Over the years, it has gained wide acceptance in the computer systems community and has become the dominant architecture for almost all kinds of services. Although in a distributed object management paradigm a client is expected to know the interface of the target object, the implementation of the object is only known by an object server. In fact, the implementation can be different for different object instances even though the same interface is supported. The platform on which the object implementation resides is the server for that object. As a result, a client which invokes a method on an object is (transparent to the client) invoking an execution on the object server. Therefore, an application is divided into two parts according to its characteristics and functionality as server code and client code. Usually client code includes the user interface while the server code includes consistency control, transaction management, persistent storage and so on. Of the two kinds of code, server code is more system-dependent while the client code is more application/user dependent. Client code is often executed at one kind of machine called a client (a desktop is the common choice for a client platform), while server code is executed at another kind of machine called a server.

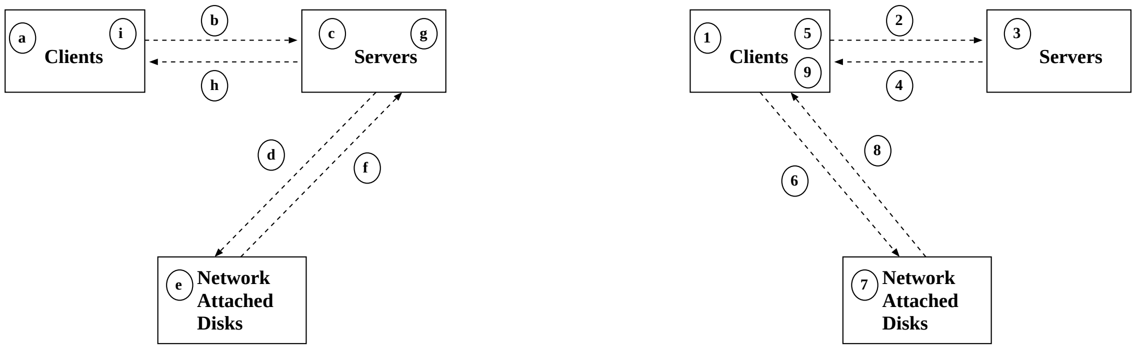
However, advances in processor and local area network technology make it possible for other software architectures to emerge. In recent years, there have been two major trends in hardware development that impact the efficiency of the client-server architecture: First is the emergence of network attached storage, e.g., fibre channel disk drives. At the beginning of the decade, the typical bandwidth of a "fast" network was in the range of several Mbits/sec and the bandwidth of a typical system bus was in the range of tens of megabytes per second. Therefore, the network was the bottleneck for data intensive computing. Under these conditions system designers chose servers with directly attached storage to process the data first and then send the output data to clients. This is an advantage when the output data volume is often much smaller than the data processed since the load on the network is reduced. Today point-to-point connected fibre channel is capable of transferring data at 100 MB/sec and the industry projection is that the bandwidth of point-to-point connected fibre channel will reach 400MB/sec soon [FCA]. Moreover, depending on the actual implementation, the total aggregate bandwidth of the fabric can be much larger than individual point-to-point connections. Currently, the aggregate bandwidth of a dual arbitrated loop is twice that of the point-to-point connection. However, during the past decade the sustained bandwidth of the typical system bus has increased only several tens of MBytes per second. Thus the network is no longer the bottleneck in a LAN environment. Also, the length of a fibre channel link is up to 10 kilometers. Therefore, fibre channel can be used in local area networks or campus area networks, but not the wide area networks.

The price of network attached storage is comparable to the price of conventional SCSI storage devices. For example, the current market price of a 9 GB fibre channel disk drive is about $1300. This is about the same price as a 9 GB disk drive with a SCSI controller. The price of a fibre channel adapter for a PC or workstation is currently about $2000. The price of a fiber channel adapter will get much cheapter in the near future to be only a percentage of the price of a PC. The actual fibre cable is in the price range of fifty dollars. Therefore, upgrading a LAN environment to the network attached persistent storage is economically feasible.

The second relevant hardware trend concerns the CPU power of client and server machines. Ten years ago, servers were equipped with much more powerful CPUs than the clients. As a result, system designers preferred servers to do most of the work despite the fact that clients have much lower utilization than servers. Things have changed dramatically during the past ten years. Now clients are equipped with powerful CPUs similar to servers. The only difference is that the maximum number of CPUs in a server is about thirty compared to two or four CPUs in current client workstations. For applications that do not exhibit high parallelism, it does not necessarily increase response time if most work is transferred to the clients. Moreover, the cost per MIPS is much higher for a server than for a client machine.

In current object server systems, clients are usually desk-tops, e.g., PCs or workstations and servers are usually a cluster of workstations or an SMP. Most commonly, clients and servers are connected by a fast local area network, e.g., 100 Mb/sec Myrinet, gigabit Ethernet, or fibre channel. Moreover, the storage devices, i.e., disks are attached to the servers.

Based on the technology trends discussed above, we introduce a new approach to object servers: Dynamic Objects (DynamO). In this architecture, storage devices are directly connected to the local area networks (i.e., fibre channel arbitrated loop) so that all clients can directly access the storage devices. A major objective is to maintain the distributed object programming paradigm which we consider mandatory. Note that the object server code implements the invoked interface and the object's persistent storage. In the proposed architecture, when an application on a client machine tries to access some object, (i.e., invoke one of the object methods) there are two possible scenarios as illustrated in Figure 1. (1) The distributed object management middleware first finds the object



(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 NAS. (e) The NAS retrieves the data. (f) The NAS sends the data to the server. (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

and loads the object method code to the client machine if it does not exist on the client already. Then the method is executed on the client machine. Moreover, the clients will also handle cache coherence and consistency control as necessary. (2) The application sends the method invocation request to the server and the method is executed on the server. (The second scenario is the traditional client server method.)

A method invocation can be done in either way. Which way is "best" can depend on the characteristics of the method, the workload at the server, etc. In the extreme, 100% of the load is moved to the client. At the other extreme, the method is executed completely on the server. The latter results the traditional client and server architecture. The DynamO architecture lies somewhere between the two extremes as we shall describe in detail later. We focus on the performance aspects of this model in this paper and explore the potential benefits of the proposed architecture.

Compute Intensive Applications

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 servers. However, per application instance, the server code usually consumes more CPU cycles than the client code in computation intensive applications. This imbalance can result in the servers being saturated while the client systems are lightly utilized. In this situation, the response time of an application can suffer due to contention on server(s). This will not occur in the DynamO architecture, since much of the work is done on the clients, and saturation of the servers is avoided.

Data Intensive Applications

Figure 2 shows an example physical system configuration in which fiber channel is used to connect components. Currently, fibre channel disk drive controllers are able to queue the I/O requests from different machines. For example, the newest Seagate fibre channel disk drive has this capability. As a result, many client machines can simultaneously access the same disk controller and the controller will schedule the I/O requests in some order, e.g.,

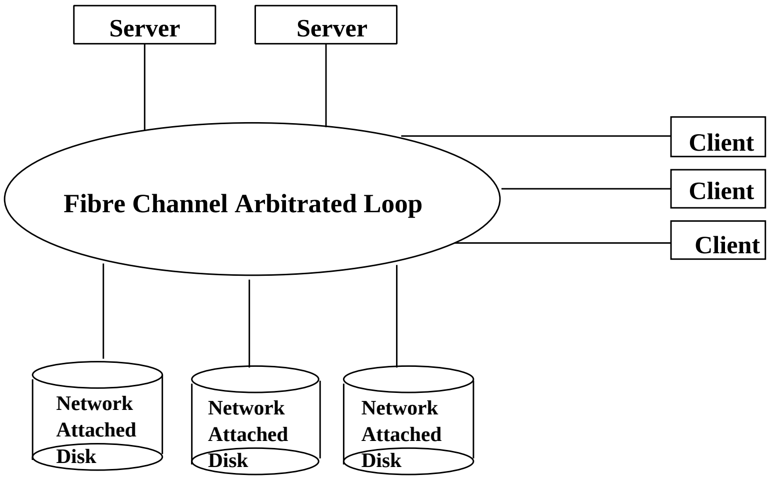


Figure 2: Fibre Channel Arbitrated Loop

minimum distance, FCFS, or some other order. This enables clients to make directly I/O requests to the disk controllers.

	Agg. Server-client Commun Bandwidth	Agg. Server System Bus Bandwidth	Agg. Client System Bus Bandwidth	Agg. Disk Bandwidth
Current	200 MB/s	$\sim$ 180 MB/s (5×36)	$\sim$ 3 GB/s (100×36)	$\sim$ 500 MB/s (50×10)
Near Future	800 MB/s	$\sim$ 300 MB/s	$\sim$ 5 GB/s	$\sim$ 2 GB/s

Table 1: Aggregate Bandwidths for Example System

Typically, there could be several hundred clients and a few servers in a local area network. Therefore, the aggregate bandwidth of client system buses far exceeds that of the server system buses. As an example, assume the servers have a total 5 system buses while the client platforms have a total of 100 system buses. The client-server communication is via a dual arbitrated fibre channel loop and there are 50 disk drives in the system. For data intensive applications, the applications' aggregate I/O bandwidth in traditional client-server architecture is limited by the minimum of the four factors in Table 1. However, the applications' aggregate I/O bandwidth in DynamO eliminates the constraint imposed by the server system buses which is typically the bottleneck.

The rate of increase in the bandwidth of a single disk is about 40% a year, but the price per MB drops about 60% per year [Gro96]. Currently, a MByte of disk storage costs about ten cents. By the year 2000, a megabyte of disk will cost about four cents and each individual disks 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). With RAID technology [Pat88], we can manage parallel disk arrays and the workload can be more or less evenly distributed among a large number of disks. As a result, large aggregate disk bandwidth is not a problem until we consider how to get the data to the processor. According to the Fibre Channel Association [FCA], the fibre channel arbitrated dual loop can deliver 200 MB/sec bandwidth currently. Moreover, the bandwidth of a fibre channel dual arbitrated loop will reach 800MB/sec in the near future. However, the sustained bandwidth of a single S-bus or PCI bus is in the range of 20 to 60 MB/sec [Arp97]. We propose in this paper an architecture that moves the execution of server functionality to clients and allows clients direct access to disk storage controllers, thus eliminating the bottleneck caused by the server system bus bandwidth limitation.

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 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. In Section 3, we will discuss the parameters of the hardware in the simulation models. The application path length is described in Section 4. We then define a client server model and a model of the DynamO architecture in Section 5 and Section 6, respectively. Bottleneck Analysis is explored in Section 7. In Section 8, we compare the performance of these two models. Finally, Section 9 presents conclusions.

2 Related Work

2.1 Condor

The machines in a local area network can have widely varying workload at any time: some jobs may be waiting to be served on one machine, while some other machine is idle. To achieve better utilization, at the University of Wisconsin-Madison a powerful, distributed batch-processing system called Condor was developed [Tan95]. Condor treats all workstations connected via a fast local area network as a pool of resources: memory, disks, and processors. In recent years, Condor has been used across machines more widely dispersed. If a workstation becomes idle and some job is waiting, then the job is sent to the idle workstation. Once the owner of the workstation invokes some computation on the workstation, the Condor controlling daemons will halt execution of any "visiting" job and move its execution to another machine. A checkpoint file can be written to disk periodically to facilitate management of the computations. When a workstation becomes idle or the load goes below a certain threshold, that machine can be assigned a job and continue its execution. This allows Condor to utilize otherwise idle CPU cycles in all connected workstations. However, this approach has some limitations:

1. Condor requires a certain degree of homogeneity of workstations, but in a local area network, different kinds of workstations can coexist. A process state from a SPARC workstation can not be easily migrated to a HP workstation.

2. Condor does not address the I/O bandwidth issue. In a Condor system, the I/O can still be a bottleneck.

3. Condor still employs the traditional client-server architecture. If many applications have to access a certain set of server machines, then the servers can still be the bottleneck.

Despite the above issues, Condor is a very successful product. In practice, it utilizes all connected workstations in a local area network and effectively shortens the response time of a job.

2.2 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 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 file storage is attached to all workstations. Therefore, part or all client workstations can act cooperatively as a file manager or storage server, or both. As a result, this file system architecture has very good scalability. Moreover, it employs a cooperative cache technique. When one client tries to access some 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.

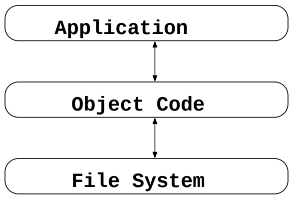


Figure 3: General System Hierarchy

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]. But as the name indicates, it is only a file system even though it largely eliminates the file server bottleneck. However, there is much more in the path length of an application than file system code as shown in Figure 3. In a distributed object management system, a significant part of the path length of an application is object server code. Many clients may access a single object server with the result that the server becomes a CPU bottleneck. A serverless file system along does not address the object server part of the problem. Therefore, users still need object servers to access the data even with the serverless file system. The path length of the server code varies of course, depending on the application and the server can still be the bottleneck as the number of clients grow.

Moreover, client workstations not only act in concert as the file manager and storage server, but also 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. Some resource on some client workstation can be saturated and data accesses through those saturated machines can be impacted. How to balance the utilization of resources among all collaborating workstations remains an open question.

2.3 NASD

Usually when a file server retrieves data from storage, it first copies the data to its own memory, then forwards it to the clients. In order to eliminate the extra copy of data, file system researchers invented the "Third Party Transfer" mechanism. Network Attached Secure Disks (NASD) is the perfect example of this kind "Third Party Transfer" technique.

NASD is currently under research and development at CMU [Gib97]. A client contacts the file manager 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. This largely eliminates the bottleneck of file servers. As mentioned in the previous section, an object server is usually more than an interface to files. Therefore, there still can be bottleneck in object servers where most computation is done. NASD also focuses on security which is not within the scope of this paper.

2.4 Thin Clients

Although Java is a relatively new development, it has already gained a wide ranging support. Java is a platform-independent language, which can be run anywhere the Java virtual machine is installed. Based on Java, the "thin client" architecture has been recently proposed [Blo96].

Thin clients are stateless desktop devices linked via a network to one or more servers. These thin clients are nothing but Java applet executors. When an action is invoked on one of these thin clients, the client downloads the Java applet along with necessary data and executes the applet. Therefore, the clients have much less computing power (smaller memory, less hard disk, and so on). In addition, the clients are "dynamically maintained". Individual machines do not have to be configured, serviced, etc, which results in less cost for each client.

The thin client architecture is proposed for the corporate computing environment. In this environment, most applications are "thin". A "thin" application does not require many CPU cycles nor too much data. Thus, its response time is very short on average. As a result, the utilization of "fat" client platforms are low and unnecessarily expensive. Bloor suggests that the use of less powerful and cheaper "thin" clients to run these "thin" applications and let servers do most of the work to achieve high cost performance.

However, in this paper, we are interested in data intensive and computation intensive applications. In this environment, the applications are usually "fat", i.e., an application consumes significant CPU cycles and can access a significant amount of data. As a result, clients need significant local resources to run these applications. If we shift most work to the servers, we potentially create a bottleneck on the servers. Moreover, if our application is run on a thin client, then the response time can be significantly prolonged. Last but not least, it is known that SMP machines have much higher dollar per MIPS ratio. The DynamO moves server functionality from servers to clients and thereby hope to yield better cost performance in the computation/data intensive environment.

2.5 FALCON

The Framework for Application Level Communication OptimizatioN (FALCON) was developed in the Data Mining Lab at UCLA [She97]. Many data intensive applications are running in networks and platforms of greatly varying characteristics. FALCON is a flexible object-oriented framework for application level communication optimization which allows complementary network communication optimization techniques to be combined in the form of matching stack layers at the endpoints of each communication connection. Each stack layer is composed of a pair of matching modules, executed in the sender and receiver endpoint respectively. To exploit application knowledge that is only available to one of the communicating peers, FALCON provides a protocol to exchange stack layer module to the other. Currently, those stack layer modules are implemented in C and Java.

Based upon FALCON's implementation, we can implement the object server as a stack of modules. During the method invocation, the module(s) that correspond to this method can be dynamically moved to the client machine and executed there. FALCON serves as a first proof of concept implementation example of the DynamO approach.

2.6 JavaBeans

JavaBeans, currently under development at Sun Microsystems, is another potential tool that can be used within the DynamO framework. It defines a software component model for Java, so that a third party can create and ship Java components which can then be composed into an application by end users. One of the main goals of the JavaBeans architecture is to provide a platform neutral component architecture. Thus an object implemented on a SPARC platform using JavaBeans should be easily migrated to a Window95 platform.

JavaBeans provides us a module migration tool for DynamO; the methods in DynamO server objects can be implemented using JavaBeans. During a method invocation, we can dynamically decide where the method module should be executed, i.e., on the server machine or the client machine.

3 Model Setup

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. In this section, we present the main features of the models used in the performance comparison study. Figure 4 illustrates the two architectures. The major difference between the two architectures is that the method execution has to be done on the server(s) in the client-server architecture while the method execution can be dynamically migrated to the client(s) in the DynamO architecture.

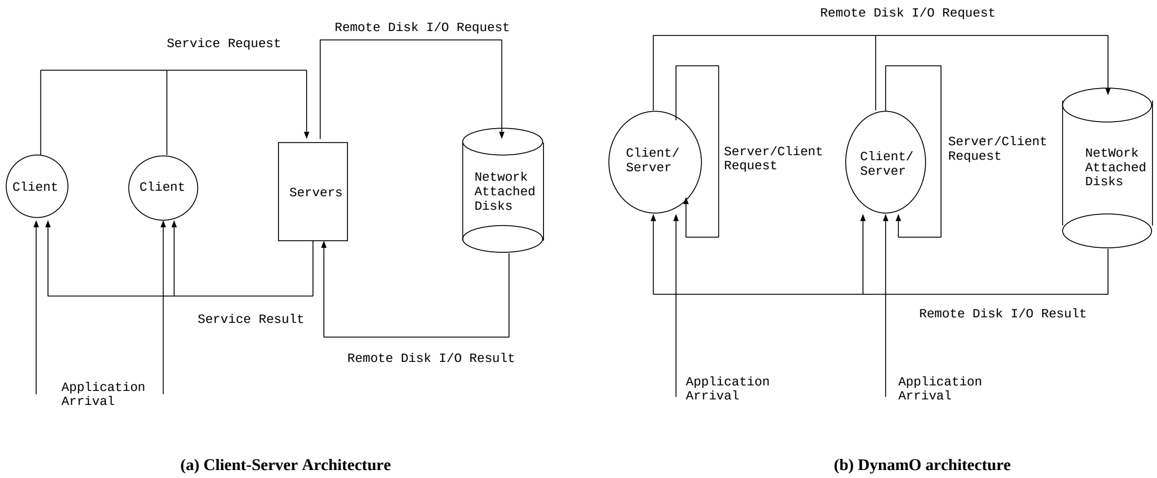


Figure 4: Two Architectures

3.1 Machine Model

Here we assume all processors in a machine, desktop or SMP, have the same compute power. Also, a cluster of workstations can also take on the role of server(s) [And96] in the client-server model. We model each machine, workstation or SMP with the following parameters:

k: number of processors in the machine.

M: the MIPS rating of each processor in the machine.

D: number of disks attached to a machine.

θ_{bus}: bandwidth of a system bus.

For the sake of model comparison, we assume two kinds of machines: SMP and workstations. An SMP, we assume, is configured with 20 CPUs while a workstation is a dual processor system. Each processor's computing power is 200 MIPS. There are 30 disks connected to an SMP, and 5 disks connected to a workstation. We assume one system bus for a workstation and two system buses for an SMP machine. Also we assume 36 MB/sec bandwidth for a system bus [Arp97].

3.2 Disk Model

If SMPs are used as servers in the client server model, then we assume that all data is stored on the disks attached to servers. Otherwise, we assume that the data is stored on the network attached disks so that any workstation can access the data. The following are the parameters for disks:

l_d: Disk I/O latency. This includes the average seek, rotation time, and settle time.

θ_d: Disk bandwidth.

In the model comparison, we assume that all disks have the same latency (8 ms) and bandwidth (10 MB/sec).

3.3 Network

The network is a fast local area network. In the past several years, fibre channel arbitrated loop has become a popular fast network choice and we assume our network is a fibre channel dual arbitrated loop which currently renders 200 MB/sec bandwidth overall.

4 Application Path Length

Developments in Condor [Tan95] and like efforts indicate that workloads on the workstations can be effectively balanced. Thus we assume that the application request (or task) arrival process to each client machine (from the total population of client machines) is a Poisson process. The execution of an application is modeled as a sequence of segments as illustrated in Figure 5.

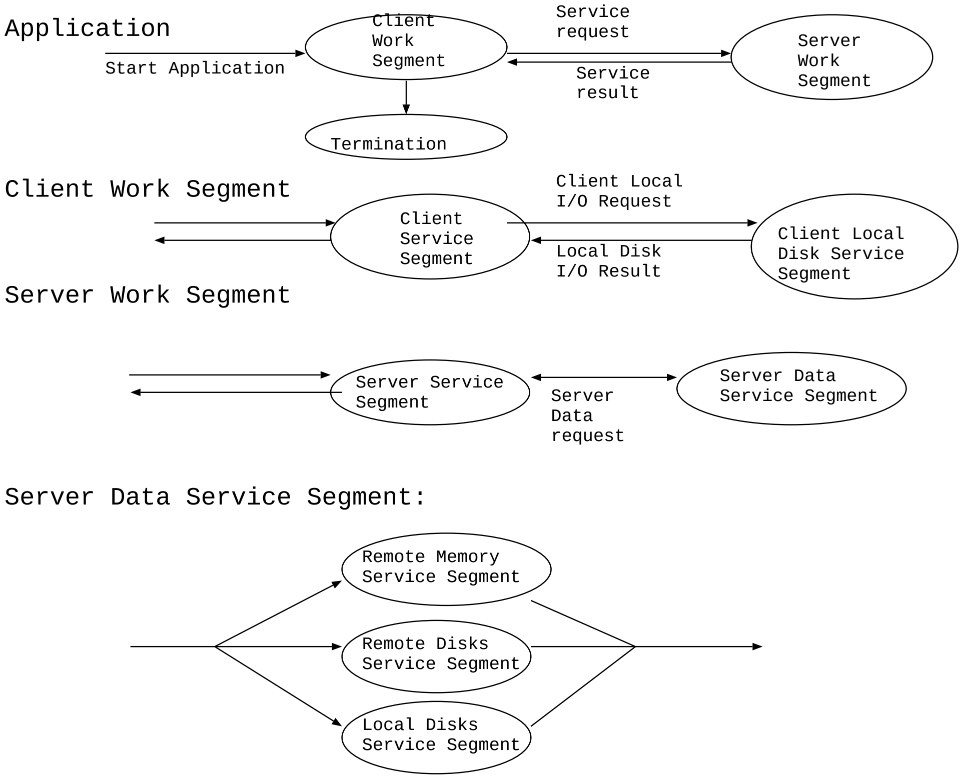


Figure 5: Path Length of an Application

An application consists of two kinds of segments: Client Work Segment (CWS) and Server Work Segment (SWS). The CWS is the part of the application that traditionally runs on a client machine, while SWS is the part of the application that traditionally runs on a server machine. (An SWS can be regarded as one service invocation from the client.) A CWS ends with either a request for an SWS or the termination of the application. Whether or not an application terminates after a CWS is assumed independent of past activity. Which server a client chooses depends on the system and application. An SWS always ends with a request for a CWS. We assume the path lengths of both CWS and SWS are exponentially distributed.

A CWS consists of two kinds of segments: Client Service Segment (CSS) and Client Local Disk Service Segment (CLDSS). A CSS is a continuous processor execution segment of an application program on a client machine. It ends with either a local disk request, a request to a server, or termination of the application. The request to a server results in an SWS, while a local Disk I/Os results in a CLDSS.

An SWS consists of two kinds of segments: Server Service Segment (SSS), and Server Data Service Segment (SDSS). SSS is a CPU execution segment of a server program on a server machine. An SSS ends with either an SDSS or by returning the result to the client. When the client receives the result for a service call, it starts a new CWS. The following are the parameters for application path length:

λ_c^i: arrival rate of applications on a client machine i.

p_{cd}: probability that a CSS ends with a CLDSS.

δ_{ld}: average number of data blocks on a client local disk for a client I/O.

p_{cs}: probability that a CWS ends with an SWS.

δ_{cs}: average size of message from client to server.

N_{css}: the average number of instructions in a CWS.

l_n: network latency.

θ_n: network Bandwidth.

N_{sss}: average number of instructions in an SSS.

p_{sd}: probability that an SSS ends with an SDSS.

δ_{sd}: average number of data blocks on a disk for a server disk I/O.

5 Client-Server Architecture

In a traditional client-sever model, clients and servers are two disjoint sets of machines. When a client wants to invoke a service call, the request has to be routed to a server that is capable of this service. Generally, not all server machines are able to provide the service. For example, the object servers could be file servers. In order to obtain data from a file, the client has to contact the file server which manages that file. This means the server has to process (fetch) the data first, then send it to the client machines.

Let ps_i be the probability that a server request (from a client) is sent to server i. The nature of $\{ps_i\}$ defines the skew of server access. The uneven distribution of workload on servers may result from uneven distribution of object accesses. Figure 4(a) illustrates the traditional client-server architecture.

In this environment, we assume that each client machine is a workstation. According to the computing power and number of server machines, we identify three cases for parallel servers: single SMP, multiple SMPs, and a cluster of workstations.

5.1 Case 1: Single SMP

In this case, the server consists of only one SMP machine. The machine has multiple processors. However, due to limits in scalability, the number of processors in an SMP is limited. We assume 20 processors in the server SMP unless otherwise stated.

Caching in main memory is very beneficial in an SMP environment because all the processors in the SMP can share the cache and there is no extra cost for cache coherence. However, there is some cost for cache coherence in the next two cases.

5.2 Case 2: Multiple SMPs

As mentioned before, SMPs generally scale to only a few tens of processors. Therefore, in order to serve a large number of clients, we may need several SMP machines to act as servers. If each SMP machine manages a disjoint set of objects, then performance of this configuration may be vulnerable to a skewed distribution in object server accesses. If all SMPs manage the same set of objects, then the path length of SSS may increase due to the overhead in cache coherence. We define the following parameter to account for this.

ω_1: the percentage increase of SSS from case 1 to case 2.

In [Hei95], Heidemann and Popek identified 3-5% overhead for cache coherence in a stackable file system environment. We believe the overhead to be similar in an object server environment and therefore assume ω_1 to be 5%.

5.3 Case 3: Multiple Workstations

A cluster of workstations could be used as parallel servers instead of SMPs. As mentioned before, we assume each workstation has 2 processors while an SMP have 20 processors. While the total compute power of a workstation is less than an SMP, the advantage is that workstations are much cheaper than SMPs per processor. The cache coherence overhead in this case is higher than that in the single SMP case, but the same as that in the multiple SMPs case. The only difference is that in this case, a smaller number of processors are colocated per server.

6 The DynamO Architecture

Traditionally, storage has been attached to one machine, and the data has to be first transferred to the main memory of that machine, and then to the client. This makes the machine to which the data is attached the logical choice as the data server. However, the traditional server notion becomes unnecessary in network attached storage because the data can be directly transferred to the client machines.

We assume that all machines are workstations. When a client issues a server request, instead of sending the request to a server, it imports the code which embodies the requested server functionality (if not already there) to the client machine, and then executes the function on the client machine. In many cases, the clients can cache all the server functionality in its local disks. Since now the client and the server functionality colocate on the same machine, the communication between servers and clients are inter-processor communication (IPC). An IPC only requires one message and, since the server and client processors are each located on a single machine, the IPC is a relatively inexpensive operation. On the downside, since DynamO needs to manage cache coherence, the number of instructions in an SSS may increase. Figure 4(b) illustrates the architecture of DynamO.

We define the following parameters for the DynamO architecture:

ω_2: the percentage increase of SSS from the single SMP case in the client server architecture to the DynamO.

T_{ipc}: the IPC time.

Assume that in the single SMP client server architecture, an SSS has X instructions. In DynamO, it has $(1+\omega_2)X$ instructions. We assume ω_2 to be the same as ω_1 (5%), and T_{ipc} to be 100μs.

7 Bottleneck Analysis

In this section, we use bottleneck analysis to discuss some of the major aspects of performance comparison between the different system models. The arguments 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 6 illustrates some aspects of the queuing network model for the general system environment.

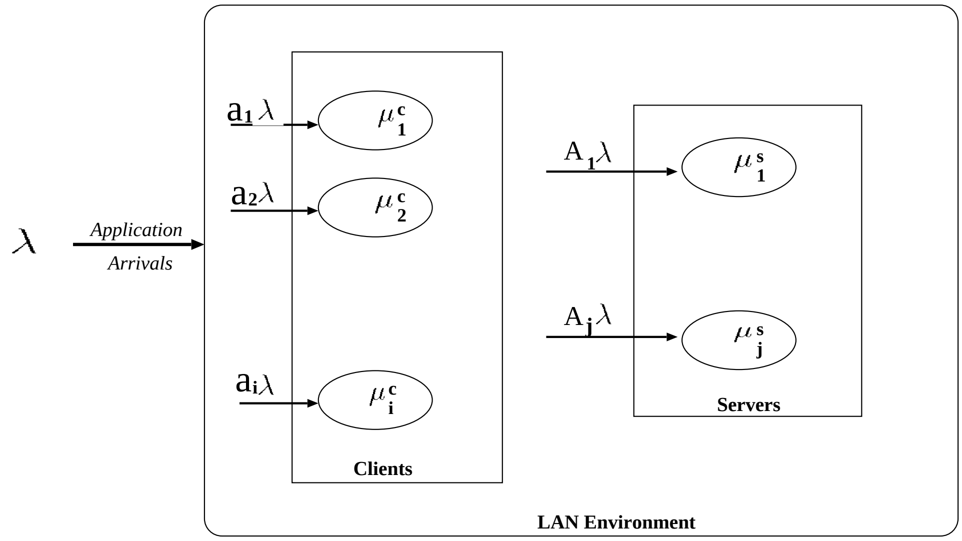


Figure 6: High Level System Description

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. Let λ denote the application overall arrival rate for all the client machines and let a_i denote the probability that the application is invoked at client machine i. Also let the probability that an application sends a request to sever j be denoted by A_j. The average service rate of client machine i is denoted by μ_i^c while the average service rate of server machine j is denoted by μ_j^s.

In the traditional client-server architecture, there are more client machines than server machines. For an application, the average server service rate μ^s is much smaller than the average client service rate μ^c. (In other words, the average server service time is much longer than the average client service time.) Moreover, the distribution of the A_j can be skewed because certain requests only can be served by certain servers. As a result, a server is much more likely to become a bottleneck.

The number of client machines is considerably more than the number of the server machines. However, 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. Therefore, the server bottleneck can be alleviated.

As in any system, there are potential bottlenecks in the DynamO architecture. If the distribution of a_i is skewed, some client machines can become bottlenecks. For example, when certain users or applications are assigned to certain client machines, some client machines could have high utilization levels while others have low utilization levels.

Since we can intelligently assign the applications to client machines, the a_i can be evenly distributed in DynamO. Therefore, the DynamO architecture eliminates the first order bottleneck in the server-client architecture (servers). It will deliver lower response time and higher throughput for applications that have evenly distributed client access pattern.

Price is a major consideration in evaluating a system. As mention easily, the dollar per MIPS ratio is much higher of the server machines (servers are usually SMPs) than the client machines (typically desktops). Thus, in order to achieve similar performance, we expect that the cost of the client-server architecture is considerably more than that in DynamO.

We assume that storage devices are no more a bottleneck for data intensive applications for the reasons explained earlier. Thus, the server system bus is most likely to be the bottleneck in the client-server architecture as shown in Table 1. 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. However, if a_i is highly skewed, then the client system bus can become a bottleneck. As mentioned earlier, the a_i is more likely to be evenly distributed in DynamO while A_j can be highly skewed in the client-server architecture. Therefore, in general, the I/O bottleneck in the client server architecture occurs at a lower aggregate I/O bandwidth while the I/O bottleneck in the DynamO architecture occurs in higher aggregate I/O bandwidth.

8 Model Comparison

In the client server model, there are N_c clients and N_s servers. A request from a client machine goes to one of the server machines. In DynamO, we have N logical clients and N logical servers where $N = N_c + N_s$. A client always sends its request to one server and a server only serves one client. This means that the client and its corresponding server are tightly coupled in the DynamO architecture. In order to fairly compare those two models, we assume the overall application arrival rates on clients are the same for both models.

We compare the DynamO architecture with the three cases of client-sever architecture as described previously. For the following reasons, we did not compare the DynamO architecture with all three alternatives of the client server architecture for all performance metrics.

1. Applicability: The candidate architecture has to be applicable to the comparison metric. For example, for Case 1 (single SMP) there can be no skewed access distribution to servers.

2. Fairness: The candidate should not have an obvious disadvantage over others. For example, if we use Case 1 in cost performance comparison, the total cost of the client-server architecture could be much higher than DynamO due to the higher dollar per MIPs ratio. Therefore, we choose Case 3 for this particular performance metric which is more competitive in this aspect than the other client-server architectures.

3. Best in the client-sever model: If more than one of the client-server architectures was compared with respect to a metric (e.g. data intensive applications), we choose the case with best result for this metric as the representative of the client-server architecture.

Based on these criterion, in the following section we compare DynamO with one variant of client server for each metric. The following is the list of comparisons made with an indication of the specific client server architecture chosen as competitor to DynamO:

- Adaptability: Case 3 (workstation clusters as servers).

- Scalability: Case 1 (a single SMP server).

- Cost Performance: Case 3 (workstation clusters as servers).
- Increase of Workload: Case 1 (a single SMP server).
- Effect of a Skewed Access Distribution: Case 2 (multiple SMP servers).
- Data intensive Applications: Case 2 (multiple SMP servers).

We also compare the DynamO architecture with the client-server architecture for a real application in the end of this section.

8.1 Adaptability

In the parallel server model, the compute power of clients and servers are statically fixed. It can be difficult and costly to change the configuration of clients and servers. Therefore, a configuration may remain unchanged for months. However, the workload can shift quite frequently; changes in load are common over minutes or hours. As a result, the ratio of client load and server load ($W_{s/c}$) can also change relatively fast. For a fixed client-server configuration and a given workload, the response time depends heavily on $W_{s/c}$. The optimal mean response time occurs when the ratio of server compute power and client compute power ($C_{s/c}$) is equal to the $W_{s/c}$. On the other hand, the DynamO system will have the same mean response time for all $W_{s/c}$ because with DynamO the compute power applied to an application varies with the load. If there were no extra overhead in DynamO, then the response time of the DynamO is equal to the optimal response time of client-server model when the two models have same overall computing power.

We compare the DynamO architecture against Case 3 of the client-server model (workstation clusters as servers). In the simulation results which follow, 50 workstations are assumed in each of the two models. For the parallel client-server model, there could be several different possible partitions of clients and servers. We choose four different configurations for the parallel client-server model:

- 10 workstations as clients and 40 workstations as servers.
- 20 workstations as clients and 30 workstations as servers.
- 30 workstations as clients and 20 workstations as servers.
- 40 workstations as clients and 10 workstations as servers.

We recognize that some of the above configurations are far from ideal and would not be candidates in practice. We consider them in order to demonstrate the relative adaptability of the two models. All the servers in the parallel client-server model have the same functionality. Therefore, a client can access any server for any service. (This assumption is often not true in practice. It favors the traditional client server model.) We assume that an application has a uniform probability of accessing any client or server. Since the servers in the parallel client-server model also need to take care of cache coherence and consistency control, we assume that an application has the same path length in both models.

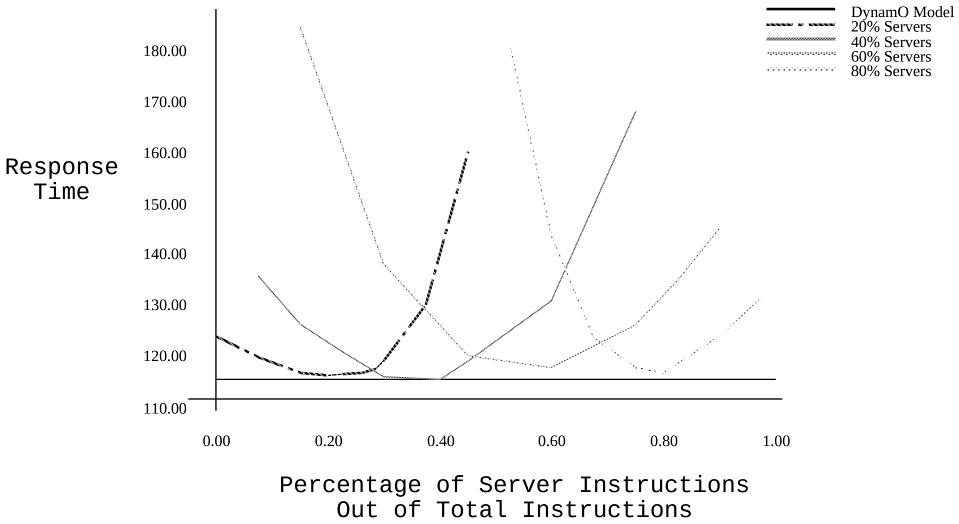


Figure 7: Adaptability

Figure 7 shows the simulation results. In this figure, the overall workload in both models is fixed. The mean response time in DynamO is a horizontal line in this figure. For a particular client server configuration, the corresponding performance is a concave function of the percentage of load on the servers. In each case, the minimum point yields the same mean response time as the DynamO architecture. This is due to the fact that once $C_{s/c} \neq W_{s/c}$, in the client-server architecture, either servers or clients have higher utilization. The more loaded component begins to become a bottleneck and the overall response time is impacted. Different parallel server configurations have different performance curves, but all of their minimum points intersect with the Dynamic Object Server performance at $C_{s/c} = W_{s/c}$.

Each point in the performance curve only represents the mean response time for a particular $W_{s/c}$. In a real system, applications will not have constant $W_{s/c}$ over time. The value of $W_{s/c}$ will vary with time over some range. Therefore, the average response time of an application is more accurately computed as weighted average of the performance curves. For example, suppose there exists an application which spends 60% of its time at $W_{s/c} = 0.4$, 20% time at $W_{s/c} = 0.3$, and 20% at $W_{s/c} = 0.5$. We assume these applications are running in a system with client-server architecture ($C_{s/c} = 0.4$). Here the average $W_{s/c} = 0.4$ which is the same as $C_{s/c}$. However, the average response time will be $0.2 \times 118 + 0.6 \times 114 + 0.2 \times 126 = 117.2$ which is larger than 114. This is due to the fact that once $W_{s/c} \neq C_{s/c}$, DynamO will outperform the parallel client-server model. Of course, the actual average response time depends on the distribution which indicates time spent on each $W_{s/c}$.

In this simulation, we assumed the application path lengths are same for the two models. However, due to cache coherency overhead, the path length in DynamO could be longer than that of the client-server architecture. If longer path lengths were associated with DynamO, the response time would be higher. There would be a small region in which DynamO does worse. But since the $W_{s/c}$ varies on average, the DynamO architecture will most likely perform better overall, i.e. it adapts to changes in workload. (as illustrated in the previous example).

8.2 Scalability

Scalability is a very important issue in evaluating a system. When we add new machines into the system, we expect to improve the average response time. The question is to what extent the improvement will be proportional to the investment. In this section, we investigate the scalability of the DynamO architecture and parallel server models.

In the client server architecture, the server configuration is assumed to be fixed - at least for a long time relative to the fluctuations in load. However, the number of client machines change more frequently. With more users or higher workloads, there is a tendency to add more clients.

The newly added client machines can not absorb any of the increased servers' workload in the client-server architecture. The result can be that the servers become the bottleneck and the mean response time increases. The client-server model clearly does not scale well with increasing client population.

However, scalability in the DynamO architecture is another story. When the workload increases, more workstations are added, the new machines can not only share client workload, but also the server's workload as well. This provides better scalability.

In the DynamO simulation model, we assume that the server in the client-server model is an SMP machine (case 1). To illustrate the simulation results we use the following specific configuration parameters. The "Total Number of CPUs" is the overall number of CPUs in the system. For example, when the "Total Number of CPUs" is 70, there are 25 clients in the client server model (each client machine has two processors and the SMP has twenty processors) while there are 35 clients in DynamO.

Since we use an SMP as server in the client-server model. There could be some extra overhead for cache coherence for the DynamO architecture. Therefore, we increase the path length of the distributed server model by 5% over that in the client-server model [Hei95].

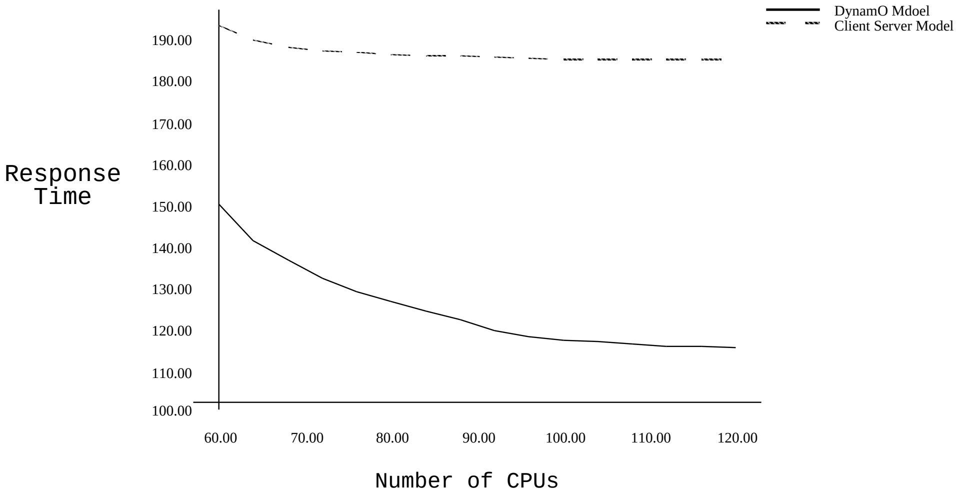


Figure 8: Scalability

Figure 8 illustrates the simulation results. With a fixed overall workload and $W_{s/c}$, the average response time of the Dynamic Object Server Environment decreases more rapidly with increasing total compute power. In contrast, the mean response time in the client server model decreases at a slower pace while the aggregate compute power of the clients gets larger and larger. Moreover, when the total number of CPUs reaches 60, the response time of the client-server architecture does not decrease any further. This is due to the fact that the utilization of client machines is clearly very low at this point (about 20%) and the additional clients do not help much. On the other hand, the response time in the DynamO architecture decreases measurable until there are about 120 CPUs because all of the workload is executed on the clients. Therefore, the client utilization in DynamO is much higher and as a result, the scalability of DynamO is clearly better.

The differences in scalability and adaptability derive from the fact that the client server model has to fix the server configuration statically. In turn, this means an a priori restriction of the compute power allocated to servers. On the other hand, the server compute power is not statically fixed in the DynamO architecture. The compute power that can be applied to an application automatically adjusts to achieve better scalability and adaptability. From another point of view, as clients move from one application to another, they bring their local compute power with them.

8.3 Cost Performance

As shown in the previous two subsections, we can see that the client server model usually results in larger mean response time than the distributed object server model. The next logical question is "how many more machines are needed for client-server model and how they should be configured to achieve the same response time as the DynamO architecture."

It was illustrated in the previous subsection that it is necessary to increase the compute power of both clients and servers to lower response time because the workload on the client and server changes frequently. (However, there is some particular cases, we can add more clients or servers, but not both to achieve better performance.) In this simulation, we assume Case 3 (workstation clusters as servers) and the server and client compute power ratio stays constant in the parallel client-server model.

Since the client-server model employs multiple parallel servers, both models have the same path length for an application. We assume that there are 50 workstations in the DynamO system.

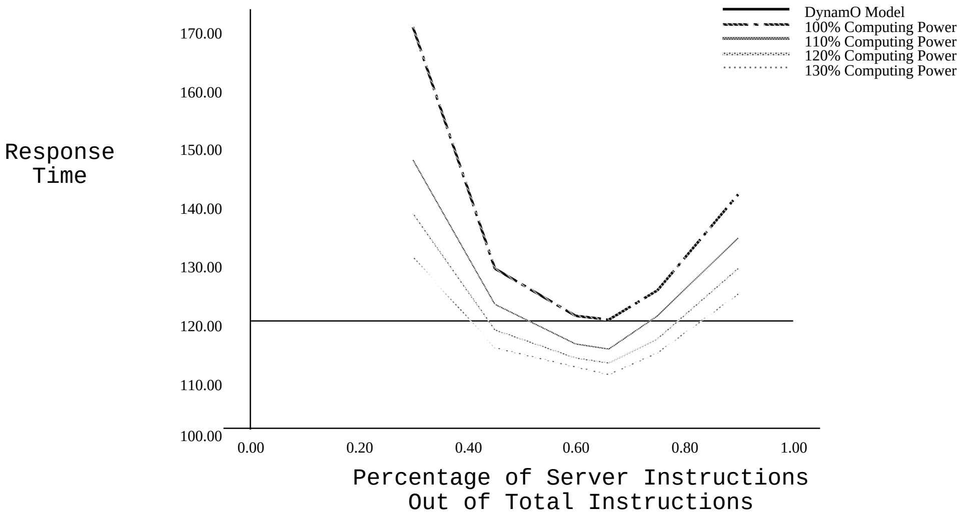


Figure 9: Cost Performance

Figure 9 shows that the response time indeed improves with an increase in server and client compute power. It is clear from Figure 9 that as $W_{s/c}$ approaches $C_{s/c}$, the client-server model with more compute power outperforms

DynamO. However, when $W_{s/c}$ differs from $C_{s/c}$, DynamO still outperforms the client-server model. For example, even with 30% increase of overall computing power, DynamO still performs better than the client server architecture when $W_{s/c} < 2/3$ and $W_{s/c} > 5/1$. The more $W_{s/c}$ differs from $C_{s/c}$, the more either the clients or the servers become a bottleneck. Utilization of the more highly loaded component still remains high when significantly more compute power is available. As a result, the average response time will tend to be high.

As we mentioned before, $W_{s/c}$ typically varies with time; the exact nature of the temporal variation depends on the application, user behavior, and so on. Its value could change within minutes. However, it is impossible to change the system configuration to track these changes in workload. Therefore, in many cases, most of the time $C_{s/c}$ does not match $W_{s/c}$. In turn, the client-server configuration performs at a sub-optimal setting. Hence, one may have to pay a large compute power penalty in the client-server model in order to guarantee the same response time when there is a large temporal variance of $W_{s/c}$.

8.4 Increase of Workload

In most situations, users of the system increase with time and a higher workload is introduced. We want to examine how sensitive performance is to this type of workload increase for the architectures being considered.

First, in the DynamO architecture, the compute power of "clients" and "servers" is dynamically and automatically adjusted. Therefore, DynamO always works at the optimal operating point. In contrast, the client-server model suffers since as load increases, the system operates at a sub-optimal point until a reconfiguration is possible. Hence, its performance degrades faster with the increase of workload.

In this simulation, we assume one SMP machine as server and 30 workstations as clients for the client-server model. We also assume 40 workstations in DynamO. The workload is measured by the number of applications invoked every unit of time on all client machines.

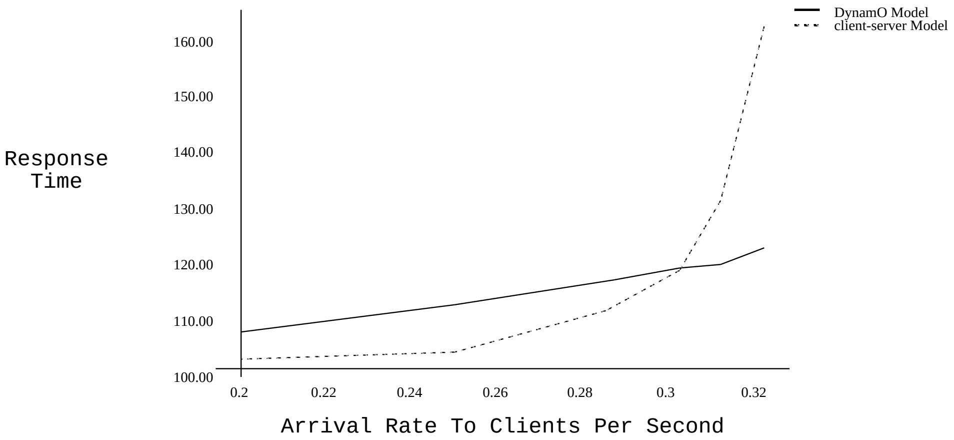


Figure 10: Increase of Workload

Figure 10 shows the results of this simulation. When the workload is low, the client-server architecture outperforms DynamO due to two facts: 1) a 20-processor SMP is a much more powerful machine than 10 dual processor workstations. Even if the SMP has higher utilization, it still has better response time. 2) Since one SMP works as a server, it does not need to expend compute power on cache coherence or data consistency control. As a result, the average path length in the DynamO model is longer than on the client-server model.

However, with a high enough workload, the server begins to saturate while the clients work at a relatively low utilization level. Therefore, DynamO outperforms the client-server model by a large margin when the workload is high.

8.5 Effect of a Skewed Access Distribution

In the parallel server model, there are three kinds of skew in access distribution: skew in client access distribution, skew in server access distribution and skew in disk access distribution. Skew in client access distribution occurs when different client machines may have different application arrival rates. This can result in uneven distribution of workload on client machines. When a client issues a service request, a server has to be chosen. As mentioned before, for many reasons, a significant percentage of service requests may go to a relatively small set of servers. This results in skew in the server access distribution. This scenario could also occur in disk access.

These three kinds of skew can also be applied to the DynamO architecture with one difference: the skew in client access distribution implies the same skew in server access distribution in DynamO. This is due to the tightly coupled association of server and client function. This is a two-edged sword. When the application arrival rate is the same for all client machines, DynamO is intuitively the winner because it will result in more equal utilization on each server. However, in the client-server model, the server access distribution is independent of the client access distribution. Therefore, the server access distribution can still be skewed.

On the other hand, when the client access distribution is highly skewed, the server access distribution in DynamO is also highly skewed. The skew in the client access pattern could result from many causes, e.g., (1) in a system, certain users are assigned to certain client machines. (2) Certain applications can only be running on certain client machines. However, in the parallel server model, the server access distribution can be smooth. As a result, the client-server model can have better performance in this situation.

In most environments, it is easier to achieve even distribution on client accesses than server accesses since all clients can execute any client code in most cases. Therefore, the applications can go to any client machine. However, this is not true for the servers. Due to the file managers and other considerations, servers have different capabilities. One server can not do other server's work. As a result, the DynamO should have better load balance than the parallel server model in this situation.

We define the skew in the server access distribution with two parameters P and Q. Here, P percent of all service requests go to Q percent of servers. For example, 70% of all service requests are served by 30% of all servers in a system. When $P = 0.7$ and $Q = 0.3$. In this simulation, we fix Q to be 0.5 which means that when $P = 0.5$, all servers have the same workload. When $P = 0$, half of the servers are idle and the rest of the servers have to handle all server requests.

Figure 11 shows the simulation results for skewed server access distribution. Here we assume that all clients have the same workload. The evenly distributed client access distribution in turn causes the evenly distributed access distribution in DynamO. As a result, the response time of DynamO is independent of server access distribution, as shown in Figure 11.

However, the response time of the client-server model is highly dependent on the server access distribution. When

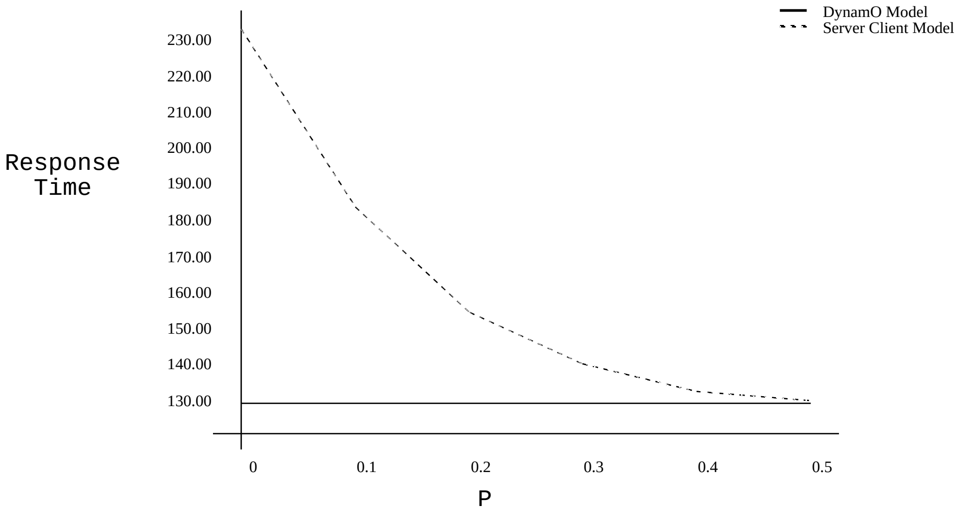


Figure 11: Skew in Server Access Distribution

the distribution is highly skewed (P=0), the average response time of the client-server model is very high because only half of the servers are working and the rest are idle. As the server access becomes evenly distributed, more and more servers are working on the server requests and the average response time in the client-server model decreases. When the server access is evenly distributed (P=0.5), all servers and clients have about the same workload. (Here we assume $W_{s/c} = C_{s/c}$.) Therefore, the response time of the client-server model is about the same as DynamO.

8.6 Data Intensive Applications

The previous simulation results illustrate the performance of the architecture for computation intensive applications. In this sub-section, we investigate performance of data intensive applications, e.g., video on demand, medical image databases, etc. For this kind of application, processor computation power is not as critical as the aggregate I/O bandwidth.

We assume two SMP machines as servers in the client server model and 40 client workstations. We also assume that the data access is evenly distributed. In other words, an application has the same probability of access to any one of the two servers for data. The server and clients are connected with dual fibre channel arbitrated loop (100 MB/sec bandwidth for each loop).

For the DynamO architecture, we assume 40 client workstations. There are 100 disks for storage, and dual fibre channel arbitrated loop is also employed to connect the clients to the storage.

In this simulation, we assume that a service request is equally likely to go to either of the two servers. If the server access is skewed, then some server will have to deliver more data than others. In turn, the aggregate application I/O bandwidth could be reduced further. In the DynamO architecture, each application issues an I/O request every two seconds on average. In the client server model, each application executes 1.5 seconds on average on the client CPU, then it sends a request to a server. In turn, the server executes 0.5 second on behalf of the application, then it sends I/O requests to the disks if the data has not been cached. Otherwise, it sends back the cached data. We assume 20% cache hit ratio in the server.

In both models, the application is blocked until the I/O is done. The mean I/O size in both cases is set to 10

MBytes in this example. Each I/O consists of many disk blocks. We assume that one disk block is 200KB and it takes 20ms to serve this block and there is another 8ms seek/latency time. Disk blocks are striped over many disks. Figure 12 shows the simulation results.

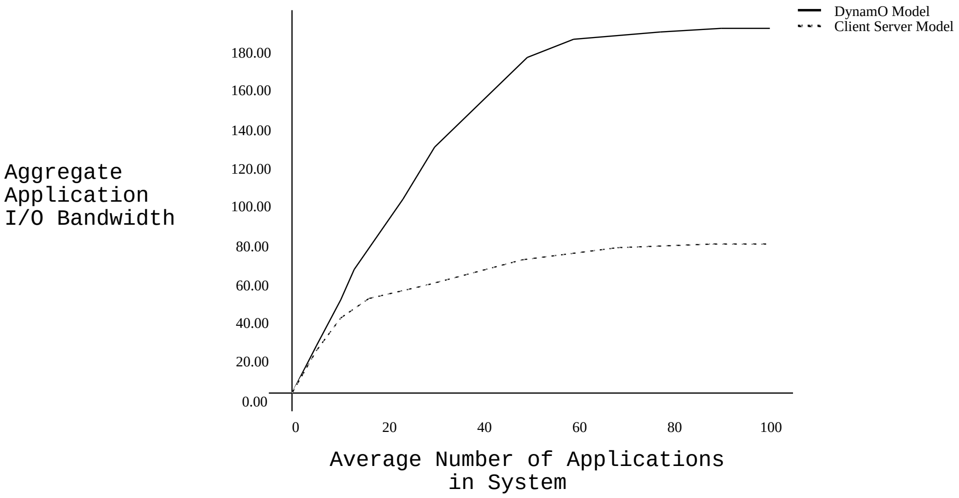


Figure 12: Bandwidth of Data Intensive Application

From this simulation, it is easy to see that when the number of applications is low, the client-server model and DynamO have similar bandwidth this is because all components have low utilization and there are no bottlenecks. When the average number of applications in the system reaches 10, the servers' system buses have over 50% utilization. As a result, the application bandwidth in the client server architecture has a much slower increase than DynamO - in which no bottleneck occurs yet. The maximum bandwidth in client server model is about 77 MB/sec while the maximum bandwidth in DynamO is 188 MB/sec. The reason is that most data has to go through the server system bus twice for each client request, flowing from the disk to the server, then flowing from the server to the client. This cuts the bandwidth of the server system bus by half. The results also are simply a confirmation that the bandwidth bottleneck of the client server architecture is the server system buses while the bandwidth bottleneck in the DynamO architecture is the fibre channel loop and thus the DynamO architecture can deliver a higher aggregate bandwidth. With the increase of disparity between system bus bandwidth and fibre channel bandwidth, the disparity between the aggregate application bandwidth in client server model and DynamO will also increase.

8.7 Performance Model of a Real Application

In the previous sections, we studied the general nature of the performance tradeoffs of DynamO versus traditional client server architectures using simulation. Now we focus on the impact of DynamO on a real application. The real application we choose to study is a GeoScientific query engine, Conquest [Fab97]. Among many operations in Conquest, we are interested in the performance of the query that finds the minimum and maximum value of a particular attribute among a large amount of data. In this application, the client machine sends the request and the server returns the minimum and maximum value back to the client.

We first implemented the object in the client-server architecture and measured the CPU time spent in each functions. We then decided how much of the code can be moved to the client for execution. The result we found is that the query spends average of 22 million CPU cycles overall and fetches 50 KB data from the disk. Among the 22 million CPU cycles, 19 million cycles are spent in calculating the minimum and maximum values. Therefore, we can move the execution of this part of the code to the client machine in the DynamO architecture. The remaining 3 million CPU cycles are spent in opening the file containing the data set, locating the data in the file, etc, and due to the security and file accessing issues, we assume this part of the code still remains on the sever machines in the DynamO architecture. In addition, we assume that the path length increases 5% in the DynamO architecture. Thus, for this application, the client executes approximately 20 million CPU cycles per invocation while the server executes about 3.15 million CPU cycles per invocation.

We assume that there are 5 identical machines in the test, of which one is dedicated to be a server, and the rest are client machines. The disks are attached to the fibre channel network, and has bandwidth 10 MB/s and the average latency 8 ms. Figure 13 is the result of the simulation.

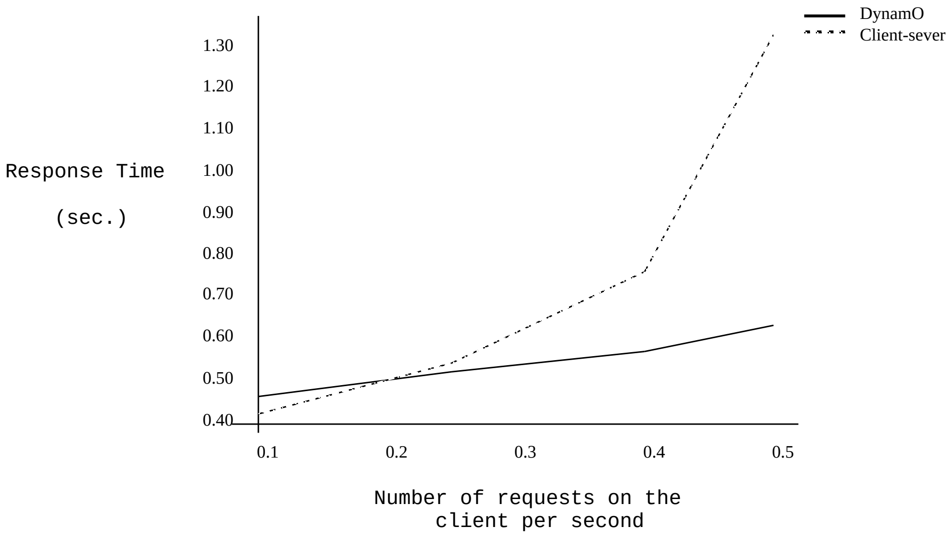


Figure 13: Performance on Real Applications

In Figure 13, it shows that when the workload is light, the client-server architecture outperforms the DynamO architecture slightly due to the increase of path length. When the workload (number of requests) increases, the utilization of the server in the client-server architecture increases fast and it becomes the bottleneck. Thus, the DynamO architecture outperforms the client-server architecture significantly. Also in this simulation, it is more computation intensive rather than data intensive because the processors have higher utilization than the networks. This simulation demonstrates that the DynamO architecture outperforms the client-server architecture by a large margin for some real applications.

9 Conclusion

With new developments in computer hardware, there is also a need to reconsider software system architecture. In this paper, we introduced a new model to replace client-server architecture in the area of computation/data intensive applications. This new model downloads most server functionality to clients and eliminates the server bottleneck.

We also studied the performance of the DynamO architecture. With the temporal variation of application path length, the 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 the DynamO has extra cache coherence overhead, the percentage overhead is not high. We believe that DynamO provides a more cost effective scalable and skew insensitive solution than the traditional client server architecture.

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.

[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.

[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.