ActiveStorage: A Storage Server with Mobile Code Engine

Michael Golm, Christian Wawersich, Meik Felser, Jürgen Kleinöder
Dept. of Computer Science
University of Erlangen-Nürnberg
{golm, wawersich, felser, kleinoeder}@informatik.uni-erlangen.de

Abstract

Many applications must examine a large amount of file data and have only a very small result set. Moving such a computation to the network storage server saves network bandwidth and improves response time. Executing a `find`[1] *on an NFS mounted directory that contains the linux-2.4.6 kernel sources needs 2.86 sec. The same command executed locally needs 0.05 sec*[2]*. The ActiveStorage architecture allows a client to transfer a code component - a StoreLet - to the NFS storage server. This StoreLet can communicate with the file system using fast local invocations instead of NFS RPCs. An exact resource control renders "Denial of Service" attacks impossible and guarantees a certain quality of service level for the storage server.*

1 The JX Operating System Architecture

The storage server runs the JX operating system. JX is a single address space OS which is almost completely written in Java. Only a small core (about 100kBytes) is written in C and assembler. The system is structured into domains, which are the unit of protection and resource management. A special domain - DomainZero - allows access to the core and provides basic services, such as a name service. Interaction between domains is performed via portals. Because all domains execute in the same physical address space, a portal call is very fast (about 600 cycles).

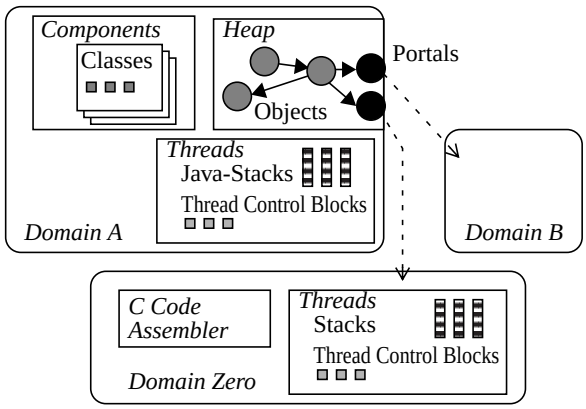

Figure 1: JX - Structure of a domain

The resource consumption (CPU, memory, network bandwidth, etc.) of one domain can be precisely controlled. When a domain uses a service of another domain, it can donate its resources to the service domain.

2 ActiveStorage

Figure 3 gives an overview of the ActiveStorage architecture. The NFS domain allows access to the file system via the NFS protocol. A separate network protocol is used by the RemoteExec domain to allow StoreLets to be transferred to the host and installed in their own domain.

2.1 Filesystem

As every JX-OS component the file system is written in Java and is completely untrusted. The file system uses a block layout that is identical to the Linux ext2 file system, so existing Linux partitions can be used. The file system runs in its own domain and exports its service as a portal. The main entry point into the file system is registered at the DomainZero name service. Another domain can look up this portal using the name service and invoke a method to open a file or directory. This method returns a portal to a file. The file system domain returns an ordinary object that implements a portal interface and the portal is automatically created by the portal invocation system. This way the file system has not the burden of manually creating and exporting portals for each file.

2.2 NFS

The JX NFS server implements the NFSv2 protocol [10].

Figure 2 depicts the achievable request rate measured with a single client (relevant part of the benchmark code is given in Appendix A). The sharp drops at 1.5 sec, 3.3 sec, and 5 sec are caused by a garbage collector run. The garbage

1. find . -name itimer.c
2. 500MHz PIII, 3C905B 100MBit/s, 128 MB, Maxtor 91303D6 Caldera Linux 2.2.14, mount options "proto=udp,nfsvers=2,noac"

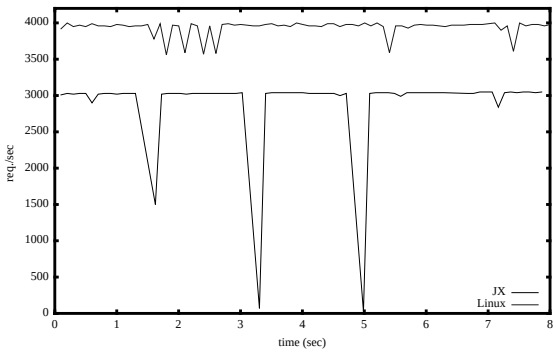

Figure 2: JX NFS performance compared to Linux

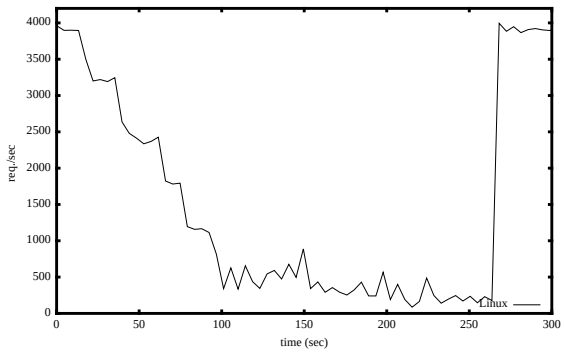

Figure 4: Linux NFS performance under load

collector is a copying collector that stops all threads of a domain during collection. We are currently improving the garbage collector by making it more efficient and thus shorten the time intervals where no requests can be processed and by using other collection algorithms that do not stop the whole domain. Figure 4 shows Linux NFS rate when processes are run on the NFS server host that compete for the CPU resource (see Appendix B). After 15 seconds the load generator creates 10 competitor processes, waits 20 seconds, creates another 10 competitors (time 35), and so on. This procedure is repeated 10 times. At the end, 100 competitor processes are running for 50 seconds, then they are killed (time 260). This figure clearly shows that it is necessary to guarantee a certain amount of CPU time to the NFS daemon, for example by using a proportional share scheduler [8].

2.3 RemoteExec and StoreLets

Once transferred to the storage server, a StoreLet is verified, compiled to machine code, and installed as a domain. The StoreLet can obtain a portal to the file system using the DomainZero naming service. Once it obtained the portal it can communicate with the file system. A StoreLet can only consume a limited amount of resources. Operations performed by the file system are also accounted to the StoreLet domain.

The resource consumption of StoreLets must be controlled and bounded to guarantee a certain QoS level for the storage server. It is especially important to control the resource consumption inside the file system for activity that is caused by a StoreLet.

Mobile programs consume not only resources for their own computations but extensively use services of the host system. Most StoreLets, for example, spend much of their time in the file system. On the other side, most of these calls only need a few microseconds to complete. Because these calls are very frequent, their aggregate CPU consumption is not negligible. Therefore it is necessary to control CPU consumption at an resolution that is much smaller than a timeslice.

3 Related Work

Many new architectures try to move computation closer to the data they process.

Active Disk research [1] [11] suggests that modern semiconductor technology allows to embed a general purpose processor in the disk drive. This processor is then used to process the data by disklets before it is transferred to the host computer. Network Attached Secure Disks [9] offload work from the storage server by allowing the client to

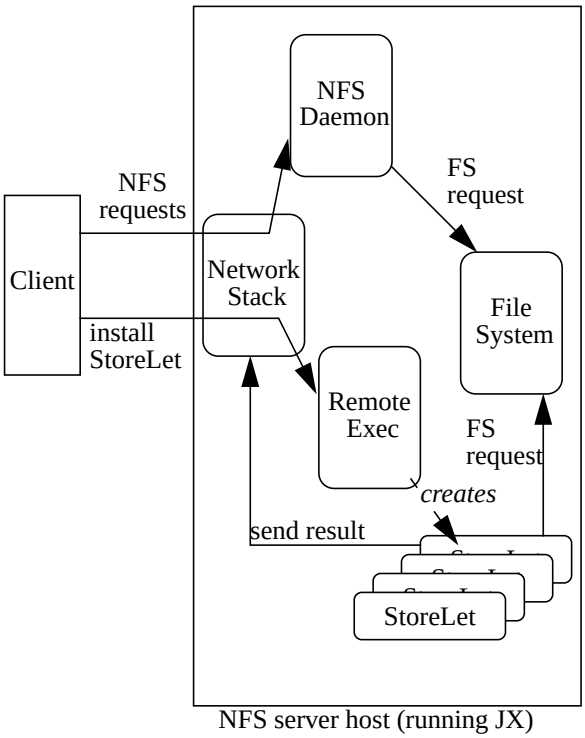

Figure 3: The ActiveStorage architecture

directly access the network attached disk drive. This assumes a fast link between the client and the disk and is contrary to our assumption, that the network connection is much slower than the local disk access. xFS achieves good scalability by using the resources of clients instead of doing all the work in a central file server. At first glance this seems to be the opposite approach to ActiveStorage. But our architecture could also be used to move computation from the server to the client.

The same architectural trend can be observed in network research. Active networks [3] allow the network packets to execute code in the routers. Xensoservers [2] are application servers with a high bandwidth network connection.

All mobile code systems must control the resource consumption of the downloaded code. Traditional OSes have problems controlling the resource consumption of processes, because they employ best-effort resource management strategies. They are especially limited, when the resource principle spawns multiple processes. Resource containers [7] are an extension for Unix-like systems to control resources when the resource principal extends into the kernel. The Rialto scheduler [6] allows to give CPU QoS guarantees in Windows 2000. Eclipse [5] is a system based on Plan9, with QoS-aware scheduler. Nemesis [4] is vertically structured QoS OS. JRes is a resource accounting extension for Java. It operates by rewriting the bytecodes of a class. With this implementation technique it can only account for resources that are visible at the bytecode level. Memory used for stacks or operating system resources can not be accounted.

4 References

[1] Acharya, A., Uysal, M. and Saltz, J: *Active Disks*, Technical Report TRCS98-06, March 1998.

[2] D. Reed, I. Pratt, S. Early, P. Menage, and N. Stratford: *Xenoservers: Accountable execution of untrusted programs*, HotOS-VII

[3] K. Calvert (ed.), *Architectural Framework for Active Networks*, Version 1.0, Active Networks Working Group, July 1999

[4] I. M. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, and E. Hyden: The design and implementation of an operating system to support distributed multimedia applications, IEEE Journal on Selected Areas in Communications 14(7), pp. 1280-1297, 1996

[5] John Bruno, Eran Gabber, Banu Ozden and Abraham Silberschatz: The Eclipse Operating System: Providing Quality of Service via Reservation Domains, In *Proc. of Usenix* 1998

[6] M. B. Jones, J. Regehr, S. Saroiu: Two Case Studies in Predictable Application Scheduling Using Rialto/NT, RTAS 2001

[7] G. Banga, P. Druschel: Resource containers: A new facility for resource management in server systems, OSDI 1999

[8] C. A. Waldspurger, W. E. Weihl: Lottery scheduling: Flexible proportional share resource management. In *Proc. of OSDI* 1994

[9] G. A. Gibson, R. Van Meter: Network Attached Storage Architecture. *CACM* 43(11), Nov. 2000

[10] Sun Microsystems, Inc: NFS: Network File System Protocol Specification, RFC 1094, Mar. 1989

[11] Riedel, E., Faloutsos, C., Gibson, G.A. and Nagle, D.F: Active Disks for Large-Scale Data Processing. IEEE Computer, June 2001

A Source code of the rate benchmark

```
start = gettime();
startbench = gettime();
for(i=0;i<nrequests;i++) {
  if (lseek(fd, 0, SEEK_SET) == -1) {
    perror("lseek file");
    exit(1);
  }
  if (read(fd, data, SIZE) == -1) {
    perror("read file");
    exit(1);
  }
  end = gettime();
  diff = end-start;
  if (diff > OUTPUT_RATE) {
    printf("%f %d\n", end-startbench, (int)((i-n)/diff));
    fflush(stdout);
    n=i;
    start = end;
  }
}
```

B Source code of the competitor

```
for(;;) {
  sleep(1);
  for(j=0;j<1000000;j++);
}
```