SMAS: THE SMART AUTONOMOUS STORAGE APPROACH TO NETWORK ARTACHED DISKS

V.V. Dimakopoulos, A. Kinalis, E. Pitoura and I. Tsoulos

2-2000

Preprint no. 2-2000/2000

Department of Computer Science
University of Ioannina
451 10 Ioannina, Greece

SmAS: The Smart Autonomous Storage Approach to Network Attached Disks*

V. V. Dimakopoulos, A. Kinalis, E. Pitoura and I. Tsoulos
Computer Science Department,
University of Ioannina,
GR 45110 Ioannina, Greece
{dimako,pitoura}@cs.uoi.gr

Abstract

In this paper, we present SmAS systems which are network-attached disks with processing capabilities, particularly suited for distributed computation. In a SmAS system, through specific API calls, clients can invoke disk-resident code to be executed remotely on the disk. Such code is in the form of pre-compiled filters with predefined memory requirements and a stream-based programming interface. The SmAS operating system at the disk provides support for scheduling filters and for memory management. We report on an initial implementation of SmAS and present performance results that validate our approach.

1 Introduction

There is an increasing demand for storage capacity and storage throughput. This demand is driven largely by new data types such as video data and satellite images as well as the growing use of the Internet and the www that generate and transmit rapidly evolving datasets, take for example the huge amount of data produced by e-commerce transactions [8, 6]. Furthermore, there is growing interest in data mining applications that efficiently analyze large datasets for decision support. Thus, there is a need for storage architectures that scale the processing power with the growing size of the datasets.

Recent disks embed powerful ASIC designs in order to deliver their high bandwidth; such chips are capable of doing considerable processing (for instance, see how complex the SCSI protocol is [2]). In addition, current disks have caches in the order of MB

[1]Work supported in part by the Hellenic General Secretariat of Research and Technology through grant PENED-99/495.

Also available as Univ. of Ioannina, Department of Computer Science, Technical Report No 2000-01.

(for example, Seagate's Cheetah has up to 16 MB of cache). This suggests that data can be processed locally at the disk. This is the idea behind active [1, 10] or intelligent [7] disks. Such disks have a general purpose processor and memory in the order of MB and are capable of doing on-the-disk processing. They have been proposed as a cheap replacement to expensive disks; they communicate with the host processor through the local bus (typically SCSI or Fiber Channel). It is envisioned that they will be able to relieve the host processor by acquiring and executing part of the application very close to the data. Such architectures are capable of handling large datasets, since the number of processors scale with the number of disks. In addition, they can effectively reduce the amount of data transfered from the disk to the host processor.

At the same time, distributed file systems are used increasingly nowadays. A good reason is the huge databases maintained by various companies [14]. Thus, a distributed storage architecture is needed in order to provide efficient and scalable access to data. Such an architecture is provided by NASD [5] coupled with their object-oriented file system for network attached storage.

In this paper, we present the SMAS system that employs network attached devices with active disk functionality. We consider it more important (and practical) for the active disks to be stand-alone (autonomous) rather than plugged in a local bus. This is feasible as the processing power at the disk is already capable enough of executing a simple TCP/IP stack. SMAS disks can also be configured to operate when attached to the local bus.

Our focus is on building a running system that would allow clients to execute application code at such network attached disks. We consider the actual constraints placed on the application programs to be run on the disk and the necessary operating system support for executing them. In particular, we provide a specific interface for clients to deploy application programs at the disk through filters that have predefined memory requirements and a stream-based interface. We also provide operating system support at the disk for scheduling such filters and for performing memory management. We have tested our initial implementation of SMAS and compare its performance with NFS. The results are encouraging since the measured performance is close to the expected ideal speed-up.

The remaining of this paper is structured as follows. In Section 2, we briefly review related work. In Section 3, we present our design, while in Section 4 we report on our initial implementation and results. Section 5 concludes the paper.

2 Related work

The idea of data processing at the disk is not a new one. Early proposals include the IBM 360 I/O processors [9] and the specialized database servers of the 80's [4]. What makes active disk architectures attractive today is that current technology is such that sufficient processing power and memory can be economically embedded in disks.

A stream-based programming model for disklets (i.e., disk-resident code) is presented in [12, 1]. Their active disks are attached to the local bus of the host processor. The disklet programming model is similar to ours in terms of providing a stream-based interface, restricting local memory and disallowing disk access. They justify their design decisions by providing a detailed simulation of active disks. In contrast, our focus is on building an actual system which introduces practical restrictions. In [12, 13], an evaluation of the disklet model is provided against two alternative architectures: shared memory multiprocessors (SMPs) and workstation clusters. For most of the applications tested, active disks and clusters significantly outperformed the SMP architecture.

The IDISKs (Intelligent disks) architecture proposed in [7] is based on replacing the nodes in a shared-nothing cluster server with intelligent disks that is disks capable of local processing. The main difference in the IDISKs architecture is that the disks are directly connected with each other via switches thus exhibiting much higher bandwidth disk-to-disk communication.

The architecture closest to SmAS is the active disks of [10]. The authors of [10] concentrate on developing a number of applications to validate the active disks approach. Their analytical and experimental results promise linear speed-ups in disk arrays of hundreds of active disks for certain data-intensive applications. Instead, the alternative of directly attaching a number of traditional SCSI disks to the local bus of a single server machine caused the server CPU or the interconnect bandwidth to saturate even when a small number of disks (less that ten) was attached.

3 The SmAS approach

SmAS devices are autonomous (i.e. network-ready) disks but they also have significant processing capabilities like Active/Intelligent disks. Active disks have been mainly envisioned as attachments to the local bus of a server [12]. This would however require expensive server architectures especially if one also considers the resources required for supporting multiple interconnected active disks.

It is essential, from a practical point of view, to investigate devices that do not require alterations to a given infrastructure. A smart disk should use part of its processing capabilities to support a simple TCP/IP stack and attach itself easily on a local network. Apart from the obvious cost reductions, this approach has the additional advantage of true distributed processing, without the bottleneck of a front-end processor (in a multiple active disk arrangement [12] disks can only communicate with clients through the server processor's connection to the network). However, a SmAS device could easily include interface support for SCSI or FiberChannel local buses, so as to be easily attached on a local bus if desired.

Next, we present in detail the specifics of SMAS design. This includes the programming interface for invoking remotely the disk-resident code and the necessary operating system support for the disk.

3.1 Programming interface

An application uses the SMAS device through a client-side API. In order for the disk to be employed seamlessly in an existing network, all standard operations are supported (`open()`, `read()`, `write()`, `lseek()` etc.); a SMAS disk can easily replace a conventional NFS server.

The API includes additional calls that allow for local processing at the disk. This way a disk-side part of the application can be defined; it is executed on the smart disk and its sole purpose is to minimize the communication between the disk and the client by eliminating the unnecessary data movements. For example, instead of transferring the whole file at the client side and performing an SQL-Select there, the disk-side code may perform an SQL-Select locally and thus communicate only the selected data to the application.

In contrast with the disklet programming model [1], we have decided to use a library of pre-compiled *filters* stored at the disk. While downloadable disklets offer a high degree of flexibility, they also place an additional security burden on the disk. Also, it is not clear how the whole disklet mechanism described in [1] would actually be implemented in practice.

We have implemented a collection of parametrized filters including a general SQL Select filter, filters for aggregate operations (min, max, count, etc.) plus an implementation of the `grep` command for unstructured data. Filters are registered and placed in a filter library at the server. This is done statically, through a strict `register_filter()` interface which requires for each filter, apart from its main code, an initialization and a parameter checking routine.

A client selects the required filter with a `smas_usefilter()` call; this call identifies the filter to be used plus the required parameters (if any). After that, the client has access to the relevant remote data, one entry at a time, by continuously calling `smas_nextrecord()`. The simple application discussed in Section 4 (Fig. 1(a)) illustrates the SMAS API conventions.

Filters are executed at the disk and are responsible for selecting and processing the data that is going to be sent to the client. A filter receives data from an `open()`ed file and possibly outputs processed data for delivery to the application at the client's side. Filters are, like disklets, simple stream-based processes with no file access rights. The on-disk operating system is responsible for supplying a filter's input with file data.

A filter is not allowed to acquire or release memory dynamically. All its memory requirements are known apriori (they are declared at the time the filter is placed in the disk's filter library). Consequently, whenever an application decides to use a particular filter (through a `smas_usefilter()` API call) the disk's operating system knows exactly the necessary memory size and allocates it statically to the filter.

In order to allow for higher-level processing at the disk, filters should know the *structure* of file entries. Files are assumed to consist of a sequence of records, each described as a collection of fields. Before a filter can be utilized, the application should inform the SMAS system about the number and type of fields of each file record (through a `smas_recordstruct()` API call).

3.2 The operating system

An operating system is necessary for a smart disk. The SMAS operating system (SMAS OS) should be as thin as possible in order to minimize execution overheads and memory costs. Consequently, its functionality should be kept down to a minimum. The only offered services must be:

- Networking,
- Process (filter) management, and,
- Memory management.

Full networking functionality is necessary for attaching the disk to an existing network. Such functionality should be consider rather common place and inexpensive (consider for example the network-ready CD-ROM server by Axis [3]). Particular attention should be paid at optimizing the network interface (for example minimization of the number of messages exchanged between the server and the client and compression of messages).

A limited form of process management will be required in order to schedule the filter execution. Such a need arises when multiple clients are connected to the disk. Process management should be simple and efficient. However, as discussed in [1], simple strategies like run-to-completion could ultimately limit the disk's performance. In our case, the fact that filters are pre-compiled and embedded in the disk's library allows for accurate estimations of their running times and, consequently, for more informed scheduling strategies, like shortest-job-first [11].

Finally, memory management is probably the most important service since disk's memory is usually its most limited resource. However, memory management is highly simplified due to the filter-based processing: the memory requirements of the filters are known apriori and are satisfied as soon as a filter starts execution. Since the filter cannot allocate dynamically any more memory, it is a simple (and quick) matter to free the allocated memory as soon as the filter completes its execution.

Finally, it should be mentioned that minimum additions are required at the client's operating system so as to have kernel support for SMAS functionality. In order to avoid such interventions the client-side API might as well be implemented as a user-space library, with only slight performance degradation.

4 Implementation and evaluation

We have emulated a SMAS device using an old Pentium-based PC, running at 166 Mhz, with 32Mbyte of memory and with a minimized version of Linux as its operating system. The hardware components are analogous to that found inside a present-day disk only much more economical. The SMAS system software (SMASOS) is running as a Linux daemon and awaits at a particular port for a client connection. Communication is handled by the standard `socket` library. At startup, SMASOS acquires a 16Mbyte memory chunk and uses it for implementing its own memory management.

In order to verify the validity of our approach as well as reveal any inefficiencies in the implementation, we have experimented with a number of applications. Although simple, a particularly illuminating one is shown in Fig. 1(a). The client utilizes the SQL-Select filter in order to obtain and process relevant records from a file stored at the smart disk. The filtering is done locally at the disk, which returns only the appropriate records to the application, minimizing thus network communication.

We then executed the same application by NFS-mounting the file (Fig. 1(b)) and compared the total running times. If T_{SMASS} and T_{NFS} are the corresponding running times of the two approaches, the observed speedup is defined as T_{NFS}/T_{SMASS} and is plotted in Fig. 2, for various record sizes.

Let s be the *selectivity factor*, that is the probability that a data record is selected by the SQL-Select filter. If a file contains N records then the SMAS code will only deliver sN of them to the application. We generated files that would result in prescribed values for the selectivity factor s (between 10% and 100%). Fig. 2 shows clearly that the SMAS version is able to deliver superior performance, especially for smaller selectivity values. This fact should actually be expected because of the reduced network communication.

The results in Fig. 2 were obtained for file sizes in the area of 100MBytes. In order to determine the socket messaging overheads we experimented with various record sizes. It can be seen from the figure that the performance did not exhibit a wide variance; however, the smaller the record size the smaller the observed speedup was. This is due to the headers inserted by the socket library to a message; these extra bytes account for a smaller percentage as the message size grows, improving thus performance.

Assuming that the communication / computation (processing) time ratio per record is greater than one, computation (processing) time, then the total running time should be dominated by the total communication time. Since the SMAS version only communicates

```
/* Open remote file */
fd = smas_open("zeus.cs.uoi.gr:/pub/testfile", O_RDONLY);
/* Inform the disk about the record's structure (100 bytes) */
smas_recordstruct(fd, INT, 4, CHAR, 96);
/* Specify the filter to use: check if 1st field > 10 */
smas_usefilter(fd, SQLSELECT_FILTER, 1, GREATER, 10);

/* Now get and process all selected records */
while ( smas_nextrecord(fd, buffer) != 0 )
   process(buffer);
```

(a) SMAS (client) application

```
/* Open the remote file - NFS mounted */
fd = open("/mnt/zeus/pub/testfile", O_RDONLY);

/* Get, filter and process records */
while ( read(fd, buffer, 100) != 0 )
   /* Filter at our (client) side */
   if ( checkcondition(buffer) )
      process(buffer);
```

(b) Traditional NFS approach

Figure 1: Sample application

a portion s of the total data, while the NFS-mounted file approach communicates all of it to the client, it is expected that (ideally) a speedup of the order of $1/s$ should be observed.

Fig 2 however shows that despite the improvements over NFS, the performance of the SMAS application, is actually well below the ideal. This is largely due to the following reasons:

- SMASOS in its present form places a significant overhead on the filter execution, which makes filtering time per record non-negligible. This is worsened by the fact that SMASOS currently runs as a user-space process in Linux.
- NFS is highly optimized, utilizing prefetching and caching techniques which in effect pipeline computation and communication to a high degree.

We are currently optimizing SMASOS and investigate data prefetching techniques. We are confident that performance will move much closer to the ideal. It is also in our future plans to provide kernel support for SMAS in Linux, which will improve performance

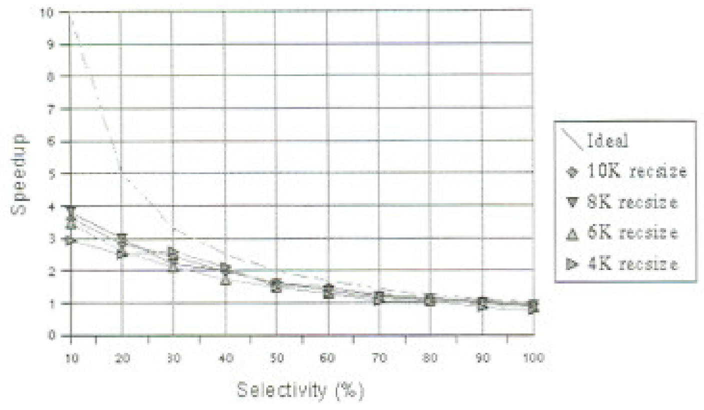

Figure 2: Speedup with respect to traditional NFS handling

significantly.

5 Conclusions and future work

In this paper, we introduced the SMAS network attached disk architecture with programming functionality on the disk. As compared to a classical file server, an autonomous network-attached device offers significant advantages. First of all, the cost is much smaller, so that one could purchase a number of smart disks for the price of a mid-sized server. Second, SMAS devices have dedicated processing on the disk that provides for efficient distributed processing. A server on the other hand is a general purpose machine that has to deal with many other things apart from file processing. The results of an initial implementation of SMAS are encouraging and justify our design decisions.

Our future plans include: (a) optimizations for pipelining disk access, processing at the disk and communication, (b) an interface at the client-side for clients to program and register their own filters and (c) advanced filter scheduling.

References

[1] A. Acharya, M. Uysal, and J. Saltz. Active disks: programming model, algorithms and evaluation. In *ASPLOS '98, 8th Conference on Architectural Support for Programming Languages and Operationg Systems*, pages 212–217, San Jose, California, October 1998.

[2] ANSI. Information systems - small computer system interface-2 (scsi-2). Technical report, ANSI X3.131-1994, 1994.

[3] Axis Communications. Cd-rom servers, white paper. Technical report, 1996.

[4] D. J. DeWitt and P. Hawthorn. A performance evaluation of database machine architectures. In *VLDB '81*, September 1981.

[5] G. Gibson, D. Nagle, K. Amiri, F. Chang, E. Feinberg, H. Gobioff, C. Lee, B. Ozceri, E. Riedel, D. Rochberg, and J. Zelenka. File server scaling with network-attached secure disks. In *Sigmetrics '97, Int'l ACM Conference on Measurement and Modeling of Computer Systems*, Seattle, Washington, June 1997.

[6] J. Gray. What happens when processors are infinitely fast and storage is free? In *5th Workshop on I/O in Parallel and Distributed Systems*, November 1997.

[7] K. Keeton, D. A. Patterson, and J. M. Hellerstein. A case for intelligent disks (idisks). *SIGMOD Record*, 27(3):42–52, July 1998.

[8] George Lawton. Storage technology takes central state. *IEEE Computer*, 32(11), November 1999.

[9] D. Patterson and J. Hennessey. *Computer Architecture: A Quantitative Approach*. Morgan Kaufman, 1996.

[10] E. Riedel, G. Gibson, and C. Faloutsos. Active storage for large-scale data mining and multimedia. In *VLDB '98, 24th Int'l Conference on Very Large Data Bases*, pages 62–73, New York, USA, August 1998.

[11] A. S. Tanenbaum and A. S. Woodhull. *Operating Systems: Design and Implementation. 2nd ed.* Prentice Hall, 1997.

[12] M. Uysal, A. Acharya, and J. Saltz. An evaluation of architectural alternatives for rapidly growing datasets: active disks, clusters, smps. Technical report, Dept. of Computer Science, University of California, Santa Barbara, Technical Report TRCS98-27, October 1998.

[13] M. Uysal, A. Acharya, and J. Saltz. Evaluation of active disks for decision support databases. In *HPCA*, 2000.

[14] R. Winter and K. Auerbach. The big time: the 1998 vldb survey. *Database Programming and design*, 11(8), August 1998.