40Gbps De-Layered Silicon Protocol Engine for TCP Record

H.Shrikumar
Ipsil Inc., Cambridge MA, shri@ipsil.com

Abstract

We present a de-layered protocol engine for termination of 40Gbps TCP connections using a reconfigurable FPGA silicon platform. This protocol engine is designed for a planned attempt at the Internet Speed Record. In laboratory demonstrations at 40Gbps, this core beat the previous record of 7.2Gbps by a factor of five. We present an aggressive cross-layer optimization methodology and corresponding design-flow and tools used to implement this record-breaking TCP Protocol Engine.

The 40Gbps TCP Offload Engine has been implemented on a Xilinx FPGA platform, based on a VirtexII-pro 2VP7 device. Each FPGA device terminates a 10Gbps OC-768 channel, and the aggregate capacity of the four FPGA devices is 40Gbps. The four 10Gbps channels are intended to be connected to four trunked 10GbE ethernet ports on a router. The 40Gbps TCP implementation has been demonstrated in the lab in system level as well as gate-level simulations, and live implementations have been tested with each 10Gbps channel FPGA board connected back-to-back in transmission tests at full wire-speed. We believe this to be the fastest TCP protocol engine to have been implemented so far.

1 Introduction

1.1 De-Layered Silicon Protocol Stacks

The demand for scalable storage accessed through high-speed networking is increasing; "big-science" applications such as LHC and LSST [22] are expected to produce 5-10TB per day, while commercial databases are beginning to reach beyond 20TB, and are expected to grow a 100-fold, as shown in Figure 1 [36] At the same time, optical telecommunication technologies such as DWDM have enabled the transmission of 200 lambdas on a single optical fibre, each such channel being modulated at 10 to 40Gbps (OC-768), for an aggregate capacity of over 6.5Tbps per fibre. Transcontinental science grids such as the 400Gbps National Lambda Rail (NLR) are already being deployed. Thanks to cluster computing, the data-production and consumption throughputs of number-crunching computing clusters have also managed to keep up with these telecom data-rates, growing at a rate faster than Moore's Law.

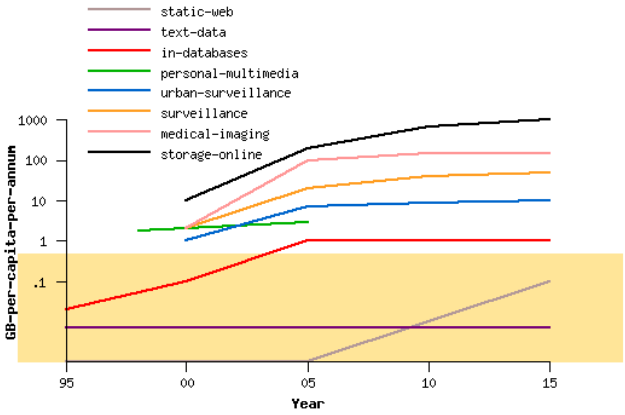

Figure 1: Storage Demands (Source [36])

It has been widely recognized that transport protocols such as TCP have the potential to overwhelm the processing power of a current generation CPU at 10Gbps and higher speeds. This has led to the development of TCP offload engines. The fastest commercial implementation of TCP in such an offload engine today clocks at 7.2Gbps for a single connection [17].

These trends serve as our motivation for "de-layering", which is a methodology to implement multiple layers of protocol stacks in a single flattened aggressively co-optimized implementation. In the work reported in this paper, we base our implementation on a scalable silicon implementation of TCP, called FlowStack. Using this silicon protocol engine, we construct a storage array, called the Grid Storage Server. It consists of a distributed disk block- and file-server architecture well suited for Grid Computing applications that can be implemented at a significantly lower cost that current CPU-based file-server architectures (27 USD-per-GB for 2TB, and approaching raw-disk costs for 48TB, Figure 2).

1.2 Organization of this paper

In the following sections, we first introduce the different components of the 40Gbps storage server, and their system archhitecture. Each component of the storage array has an instance of the FlowStack TCP engine implemented in an FPGA. The architecture of this protocol engine is described next, including a brief description of the design methodology called "de-layering" which allows for aggressive cross-layer oiptimization. Finally, we conclude with a statement of the current

3-9810801-0-6/DATE06 © 2006 EDAA

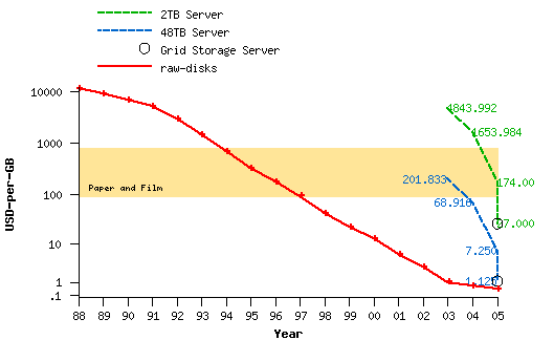

Figure 2: Storage and Server Cost Trends (Source [21])

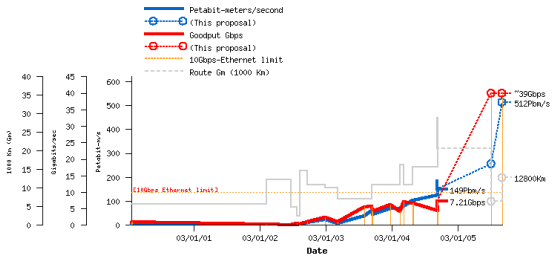

Figure 3: TCP Speed Record and FlowStack

state of the developments, the tests and demonstrations that have been completed by the implementation for operation at wire-speed.

2 All-Silicon Grid Storage Server

The Grid Storage Server (Figure 4) is implemented using two different types of protocol engines, both of which are based on the FlowStack machine described in this paper. (1) The 40Gbps WAN connection is terminated, via an OC-768 framer, by a set of 4 Xilinx FPGA boards, which together implement the 40Gbps TCP termination engine. (2) On the storage side, an array of 96 hard-disk drives are connected to the server each using an instance of a 1Gbps silicon protocol engine, also implemented using FlowStack.

As shown in Figure 4, these direct IP-attached disks together form a distributed file-server. This aggregation of disks is orchestrated by a small set of meta-data servers, which run traditional software implementation of control algorithms for metadata management, access-control, locking, logging, recovery and backup, but since they are not on the fast-path they are realized using inexpensive and low-performance machines.

2.1 TCP-Termination Engine and File Server

The TCP termination engine uses the FlowStack machine to implement a fully standards-complaint TCP/IP stack, and is capable of saturating a 40Gbps OC-768 pipe with a single or multiple TCP stream(s). It is implemented completely in silicon, using a set of four FPGA boards each with two 10Gbps interfaces. These four boards are shown in the right half of Figure 5.

The TCP-termination engine supports multiple simultaneous TCP streams. It maintains one TCP connection per client on the network side, delivering file-services over RDMA. In addition it separately maintains one TCP connection per disk-drive, connecting via a block-server protocol. These two different classes of connections are interconnected to each other via the file-server application-layer component which is also resident in the FlowStack silicon (Figure 8).

The TCP/IP stack implemented in silicon in the Grid Server is fully compliant with the applicable TCP/IP standards. The TCP protocol is nominally not parallelizable , the shared TCB state memory, or connection control block, is a point of contention. In our implementation, we use a rotating token constantly cycling between the four boards to allow the updates to the TCB to be delayed by a few packets, thereby acheiving both short-term local reordering of packets as well as distributed state consensus between the four boards.

2.2 Disk Block Server: `nbd`

Figure 7 shows our all-silicon implementation of the disk block-server component. Each disk server consists primarily of a traditional inexpensive commodity disk directly connected to the IP router by a silicon protocol engine. This protocol engine implements in FlowStack hardware the equivalent functionality of a TCP/IP stack plus `nbd`, a disk block-server upper layer protocol. Thus, while the meta-data is managed by the CPU-based meta-data servers, the data-blocks themselves flow directly from the IP attached disks into the wide-area Grid fabric; scalably bypassing von Neumann and IO-bus bottlenecks.

3 Design of Silicon Protocol Engine

In this section, we introduce the architectural and design aspects of the silicon protocol engine. First we describe some of the performance limitations of a CPU based approach, and the adaptiveness shortcomings of a custom ASIC based designs. We follow this up with a description of the FlowStack engines, its architecture and design methodology.

3.1 Traditional NPU v/s ASIC dichotomy

We would like to compare and contrast FlowStack with the two competing existing approaches for the implementation of

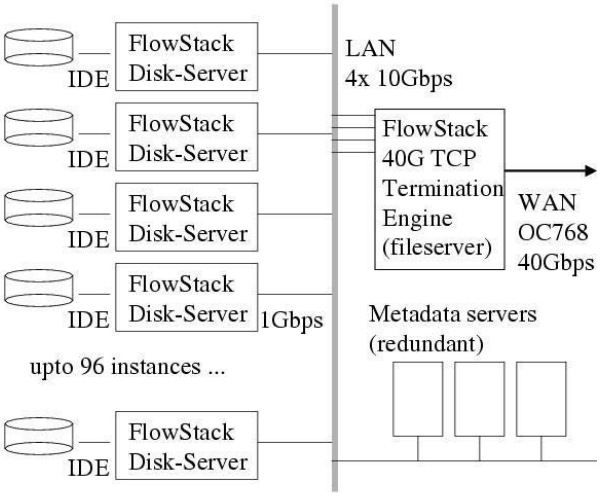

Figure 4: "Serverless" Grid Storage Server

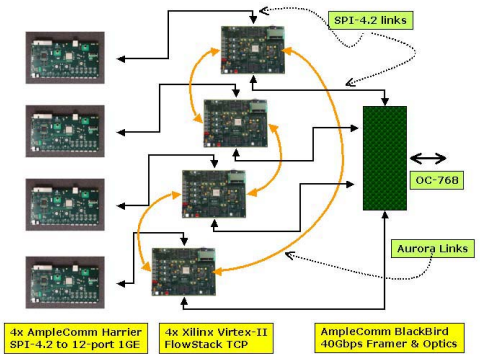

Figure 5: 4x12port linecard and 40Gbps Grid-Storage Server

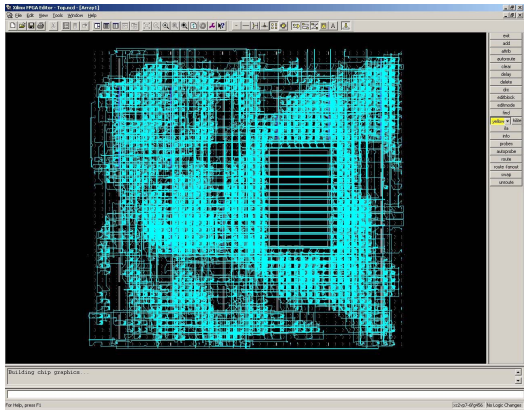

Figure 6: FlowStack engine in a Xilinx 2VP7 FPGA

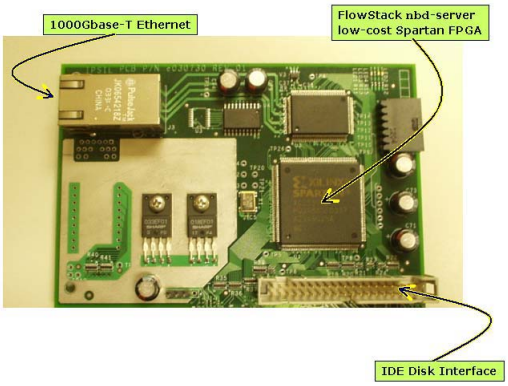

Figure 7: FlowStack disk-block server with IDE

protocol offload engines[1], both of which have their serious limitations. One approach implements the protocol engines directly in silicon, while the other uses a set of special purpose RISC CPU cores on a single chip.

The first type of implementation uses custom state-machines implemented in VLSI ASICs to implement various elements of a protocol stack. This approach is exemplified by the iReady TCP Offload processor, EthernetMAX[2] or the Univ of Oulu `WebChip`[3]. Manually translating complex protocol specifications into Verilog gate structures for implementation in silicon is expensive and inflexible; investments in the range of $50-70mn have been reported[4].

The second approach to implementing protocol offload is in the form of a pipeline of RISC processors on a system-on-a-chip (SoC). This class of implementation is also known as NPU (Network Processing Units); a number of NPUs have been benchmarked in[1]. The current crop of NPU devices scale upto 4 to 10 Gigabits per second and cost around $200-500, or in the range of $1000-3000 for a complete network interface card containing the chip. While the RISC CPU approach is certainly more flexible and programnmable, it is not inexpensive and does not scale very well. The technological future of this architecture is forever tied to the limitations of Moore's law, which for our purposes is simply not fast enough.

3.2 The FlowStack Protocol Engine

The FlowStack architecture avoids this conundrum by following a design approach which strikes a new balance between ease of programming and the speed of silicon implementation.

The principal components of the FlowStack protocol engine are depicted in Figure 3. The engine consists of a rather thin `scaffolding` or `ECA harness` of hand-coded random logic which supports the operation of the ECA Table, a

rather large AND-OR plane of logic which is implemented using semi-automated means. This latter logic plane is called the `ECA-table`, or Event-Condition-Action table.

The `FlowStack Scaffolding` contains a collection of primitive functional units, including counters, registers and other protocol-specific functional units which are used by the ECA table. For instance, there are several different types of counters, some of them are upcounting while other downcount, and some saturate at the top-count while others are designed to roll-over. Other functional elements include barrel-shifters, ones-complement adders and LFSR implementations of CRC32 and CRC16. These counters and functional units dont have any *a-priori* assignment; the ECA table can use any of these structures for any purpose from time to time. However, each such facility is ideally suited to perform a class of tasks that are commonly expected in protocol processing.

The `ECA-table` orchestrates the operation of all the counters and other functional units in the scaffolding. As the data arrives into the device, it flows past the ECA table structure at full wire-speed. The ECA table contains all the boolean logic terms to parse the packets and to save relavant information into various machine registers. These registers are saved in a `Context Memory` at the end of a packet, and restored upon the arrival of the next packet in the same flow.

In a sense, the FlowStack machine can been visualized as a giant ALU with one single instruction[1] and the context memory is akin to a register file. Conceptually the machine takes an entire packet as its input word for each beat of its operation. It indexes into and reads the current status of the context memory pertaining to the connection and after completing the computation of all the protocol layers for this packet, it writes the new status words back to the context memory.

3.3 Layers v/s Slices

The ECA-table is implemented by composting a collection of slices that are individually implemented and tested. However, unlike horizontal layers in traditional protocol implementations, each slice represents a vertical section through the protocol stack, which follows the life of a packet from the network interface all the way up the protocol stack to the application layer and back down the stack to the egress port on the network. This vertical orientation of the slices allows the programmer a first opportunity for cross-layer optimizations.

Each slice is programmed in a language that is syntactically a simple subset of `C` and Verilog, and is as easy to code as traditional software. The statements are of the form –

```
if (network_event) // event
at (context == boolean) // condition
{ context = new_values } // action.
```

While the final compiled ECA-table is monolithic hardware,

[1] Is it a most complex CISC or a most reduced RISC?

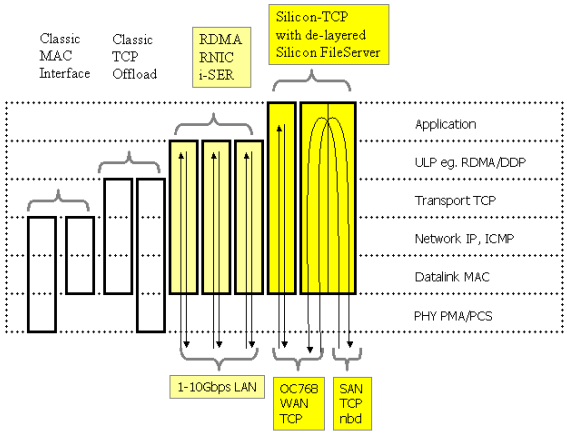

Figure 8: Slices - De-layered Programming Model

the steps that lead to its implementation are modular, with striking resemblence to software development.

Each such slice is then combined, or `composited`, along with all the other slices into the final ECA-table. The scripts that accomplish this compositing perform some consistency and coding style checks and can detect and report conflicts between slices, as follows:

1. Protocol Functional Verification stage: A conflicting case is a packet that is claimed by two or more ECA slices. This is disambiguated by specifying a priority order among the slices, and is analyzed for correctness using reachability and bisimulation[16] against a traditional software implementation.

2. Compositing Time: The ECA slice compositing tool can identify and flag all the overlap cases, which can be verified against the list of overlaps that have been explicitly analyzed during the protocol functional verification stage.

3. Synthesis time: During logic synthesis, the silicon compiler or synthesis is instructed via the `fullcase` pragma to identify and flag any cases that overlap in their enabling conditions.

4. At Runtime: The remaining logical conflicts between cases in the ECA table slices have now been separated by explicit prioritization of the choices. Each such conflict case gets reduced into priority encoder by the FPGA synthesis tool.

All transitions irrespective of the layer find themselves resident in the ECA table, and anything that is a state variable goes into the context memory (Figure 9). The synthesis tool is able to automatically identify homomorphic subsets of the gate structures in the ECA table, even if they be from conceptually unrelated layers or slices, and is able to further combine them together to reduce the logic and to improve both clock

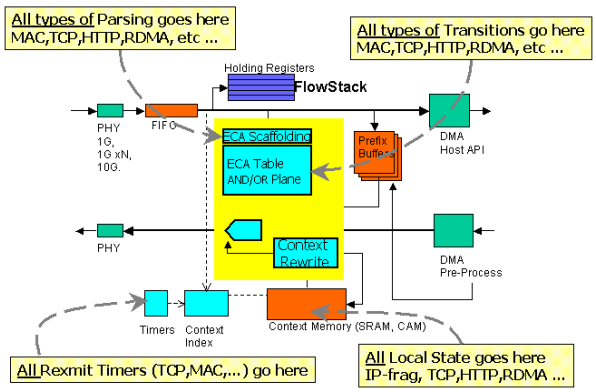

Figure 9: De-layering and Compositing

speed as well as area and power consumption. This is the second opportunity for optimization which is not available in either the NPU or custom ASIC approach described earlier.

3.4 FlowStack Advantages

As a result of benefiting from the cross-layer optimization opportunities, both those arising during manual implementation the vertical protocol slices as well as those arising during the automated synthesis of the composite ECA table, the FlowStack engine produces a silicon core that is simultaneously very compact, high performance and easy to program.

For instance, a complete implementation of TCP/IP along with support the the fast path components of RDMA (Remote Direct Memory Access) and nbd (network block daemon) protocols is only about 25,000 gates. This occupies only about 30% of an inexpensive FPGA device, such as the Xilinx Spartan. Alternatively, it can be commercially implemented in custom silicon for even further savings.

Arising from a combination of the fact that the FlowStack design already has very few gates complexity to begin with, and the fact that most real-life protocols, especially when carefully optimized, have a fairly regular structure, the resulting FlowStack core is both very small and capable of very high speed operation. In a Xilinx Virtex-II FPGA, the FlowStack engine can be clocked in excess of 125Mhz clock-speeds, at which speed it is capable of handling several 100Gbps of network traffic.

3.5 FlowStack Engine 40-100Gbps Capability

3.5.1 FlowStack Comparison with traditional NPUs

The FlowStack engine combines the programmability, flexibility and adaptability of software/firmware programmed Network Processing Units (NPUs) with the speed power and silicon-area advantages of hard-logic implementation of communication protocols.

In a CPU-based NPU, each word of the incoming packets requires a variable number of instructions to be processed by a pipeline stage CPU. This processing time variance requires that the adjacent pipeline CPU stages be connected using elastic buffers. Since these elastic buffers have to be configured with memories to hold data comparable to either entire or significant portions of network packets, they are usually implemented using off-chip memory.

The total bandwidth required of this FIFO/buffer memory is lower-bounded by twice the wire-rate speed, at the least; typically for some headroom to absorb larger peaks in packet rates, these memories are designed with a throughput of at least 4x network throughput, for each direction. At line-speeds of 40-100Gbps, this means that these internal system buffer/FIFO memories would need a read-write port throughput of approximately 320Gbps to 800Gbps; these rates are difficult to attain in affordable designs using current technology or in the near future.

In contrast, the process of converting complete protocol stacks into Verilog or VHDL for implementation in the ASIC is both very complex and risky. The environment in which Verilog/VHDL designs are developed is a much lower level of abstraction than the level at which upper level protocol engine state machines are defined; this makes the process of manual translation of protocol engines to ASIC devices both very complex, time-consuming and a risky engineering project.

The FlowStack engine, on the other hand, combines the best of both worlds. The protocol engine is itself coded in a process that is rather similar to a software development environment, making it easy to modify and verify. Once the verification is complete, the protocol engine is reduced or compiled into a low-level gate implementation by an automated reduction/compositing process. The result of this process is a single flattened `AND-OR` tabled called the `ECA Table`, which can process one word of data from the network port in each cycle of operation, with zero variance. The design therefore completely avoids expensive FIFOs.

Word widths in the FlowStack engine can range between 8 to 512 bits. The clock speeds at which the FlowStack engine can operate depends on process technology; speeds in the range of 100MHz are easily attained in FPGA delivery-platforms, while speeds in ASIC realizations can certainly be higher by an order of magnitude. A 256 or 512 bit word-width can accomodate almost all of the header/control information in a packet. Thus a FlowStack engine is capable of processing 100 Mega-packets per second. With typical payload sizes per packet (1 to 10KBytes), this translates into a processing capability in the range of 100-200 Gbps; the limit therefore is not in the inherent capability of the FlowStack engine itself but ultimately in the limits placed on clock speed by the silicon pcess-technology used and in limits placed on I/O by the pin-bandwidth.

4 Implementation and Records

The TCP protocol engines have been implemented in Xilinx 2VP7 FPGA platforms, connected with Rocket-IO ports. The device successfully clocks at 125MHz, which is the speed needed to process data at 10Gbps at wire-speed. The disk `nbd` termination TCP engines have been implemented in Xilinx Spartan devices; they support 1Gbps each using a clock of 62.5MHz. The remaining system has been succesfully simulated using the ns2 TCP network simulator for 40Gbps operation and TCP fairness.

As further work, we plan to test this TCP protocol engine across a suitable optical fiber cable, as an attempt to better the current Internet Speed Record.

References

[1] Memik et al., "Evaluating Network Processors using Netbench, ACM Trans. on Embedded Computing System (2002)"

[2] Nikos Kontorinis, Dustin McIntire, Zero Copy TCP/IP, `http:/ / www.ee.ucla.edu/ ingrid/ Courses/ ee201aS03/ lectures/ Zero-CopyTCP.ppt`

[3] Riihijarvi, P.Mahonen, M.J.Saaranen, J.Roivainen, J.-P.Soininen, "Providing network connectivity for small appliances: a functionally minimized embedded Web server", IEEE Communications Magazine, 39(10), Oct 2001, pp74-79.

[4] "iReady to Go", Byte and Switch, April 14, 2004 `http:/ / www.byteandswitch.com/ document.asp?doc.id=51001`

[5] eVLBI fringes to Arecibo `http:/ / www.evlbi.org/ evlbi/ te024/ te024.html`

[6] H.-I. Hsiao and D. J. DeWitt. "Chained Declustering: A New Availability Strategy for Multiprocessor Database Machines.", Proceedings of the 6th Intl Conference on Data Engineering, 1990.

[7] Tom Barclay, Wyman Chong, Jim Gray "A Quick Look at SATA Disk Performance", Microsoft Research, 455 Market St., Suite 1690, San Francisco, CA 94105

[8] Pei Cao, Swee B. Lim, Shivakumar Venkataraman, and John Wilkes, "The TickerTAIP parallel RAID architecture", Proceedings of the 20th Annual International Symposium of Computer Architecture, May 1993, 52-63.

[9] M. Stonebraker and G. A. Schloss. "Distributed RAID - A New Multiple Copy Algorithm", Sixth Int'l. Conf on Data Engineering, pages 430–437, 1990.

[10] J. Ousterhout. "Why aren't operating systems getting faster as fast as hardware?" In Proc. of the Summer USENIX Conference, pages 247–256, June 1990.

[11] J. Gray, B. Horst, and M. Walker. "Parity striping of disc arrays: Low-cost reliable storage with acceptable throughput". In Proceedings of the Int. Conf. on Very Large Data Bases, pages 148–161, Washington DC., Aug. 1990.

[12] Lee, E.K., "Highly-Available, Scalable Network Storage", 1995 Spring COMPCON, Mar. 1995.

[13] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. "The HP AutoRAID Hierarchical Storage System", In Proc. of the 15th Symp. on Operating Systems Principles, Dec 1995.

[14] B. R. Montague, "The Swift/RAID distributed transaction driver," Tech. Rep. UCSC– CRL–93–03, Computer and Information Sciences Board, UCSC., 1993.

[15] K. Hwang, H. Jin, and R. Ho, "RAID-x: A New Distributed Disk Array for I/O-Centric Cluster Computing", Proceedings of 9th IEEE International Symposium on High Performance Distributed Computing (HPDC-9), August 1-4, 2000, Pittsburgh, pp.279-286.

[16] R. Milner, J. Parrow, D. Walker : "A Calculus of Mobile Processes - Part I" – LFCS Report 89-85. University of Edinburgh.

[17] Fifth Annual HPC Bandwidth Challenge, `http:/ / www.sc-conference.org/ sc2004/ bandwidth.html`

[18] C.Salter, T.Ghosh, Arecibo observatory eVLBI experimental setup, personal communication, Dec 2004.

[19] Christian Tanasescu, SGIInc., "From Top500 to Top20Auto Survey of HPC Installations in the Automotive Industry", SC-2003 Conference, Phoenix, November 18,2003. `http:/ / www.top500.org/ lists/ 2003/ 11/ Top20Auto_Top500V2.pdf`

[20] Alan Dix, Janet Finlay, Gregory Abowd and Russell Beale, "Human Computer Interaction", Prentice Hall Europe.

[21] E. Grochowski, R.D. Halem, "Technological impact of magnetic hard disk drives on storage systems", IBM Systems Journal, July, 2003.

[22] Richard Mount et al, "The Office of Science Data-Management Challenge", Report from the DOE Office of Science Data-Management Workshops, March - May 2004

[24] Winter Consulting's 2003 survey of Largest DBs,
tt http:/ / mxtest.wintercorp.com/ vldb/ 2003_TopTen_Survey/ TopTenWinners.asp

[25] Jeffrey C. Mogul, TCP offload is a dumb idea whose time has come, Proceedings of HotOS IX: The 9th Workshop on Hot Topics in Operating Systems, May 18-21, 2003, Lihue, Hawaii, USA.

[26] D. D. Clark, D. L. Tennenhouse, "Architectural Considerations for a New Generation of Protocols", Proc. ACM SIGCOMM'90.

[27] H. Shrikumar, "De-Layered Grid Storage Server", to appear in ACM SIGBED Review, Fall 2005.

[28] Pat Selinger, "Top Five Data Challenges for the Next Decade", Keynote Address at the ICDE 2005, April 2005.

[29] M. Stonebraker, et al, "C-Store: A Column-oriented DBMS", Proceedings of the 31st VLDB Conference, Norway, 2005.

[30] M. Stonebraker, "One Size Fits All: An Idea Whose Time Has Come and Gone", `http:/ / www.cs.brown.edu/ ugur/ fits_all.pdf`

[31] Daniel J. Abadi, "The Design of the Borealis Stream Processing Engine", Proceedings of the 2005 CIDR Conference, `http:/ / nms.lcs.mit.edu/ papers/ borealis-cidr05.pdf`

[32] C. A. Thekkath, T. Mann, and E. K. Lee. Frangipani: A Scalable Distributed File System. In Proceedings of the 16th ACM Symposium on Operating Systems Principles, Oct. 1997.

[33] L. Lamport, "Time Clocks and the Ordering of Events in a Distributed System," Comm. ACM, vol. 21, no. 2, 1979.

[34] M. Herlihy and J. Moss, "Transactional Memory: Architectural Support for Lock-Free Data Structures ", Proceedings of the 20th International Symposium in Computer Architecture, 1993

[35] Miltos D. Grammatikakis and Stefan Liesche, "Priority Queues and Sorting Methods for Parallel Simulation", Software Engineering, vol. 26, no.5, 2000.

[36] Pat Selinger, "Top Five Data Challenges for the Next Decade", Keynote Address at the ICDE 2005, April 2005.